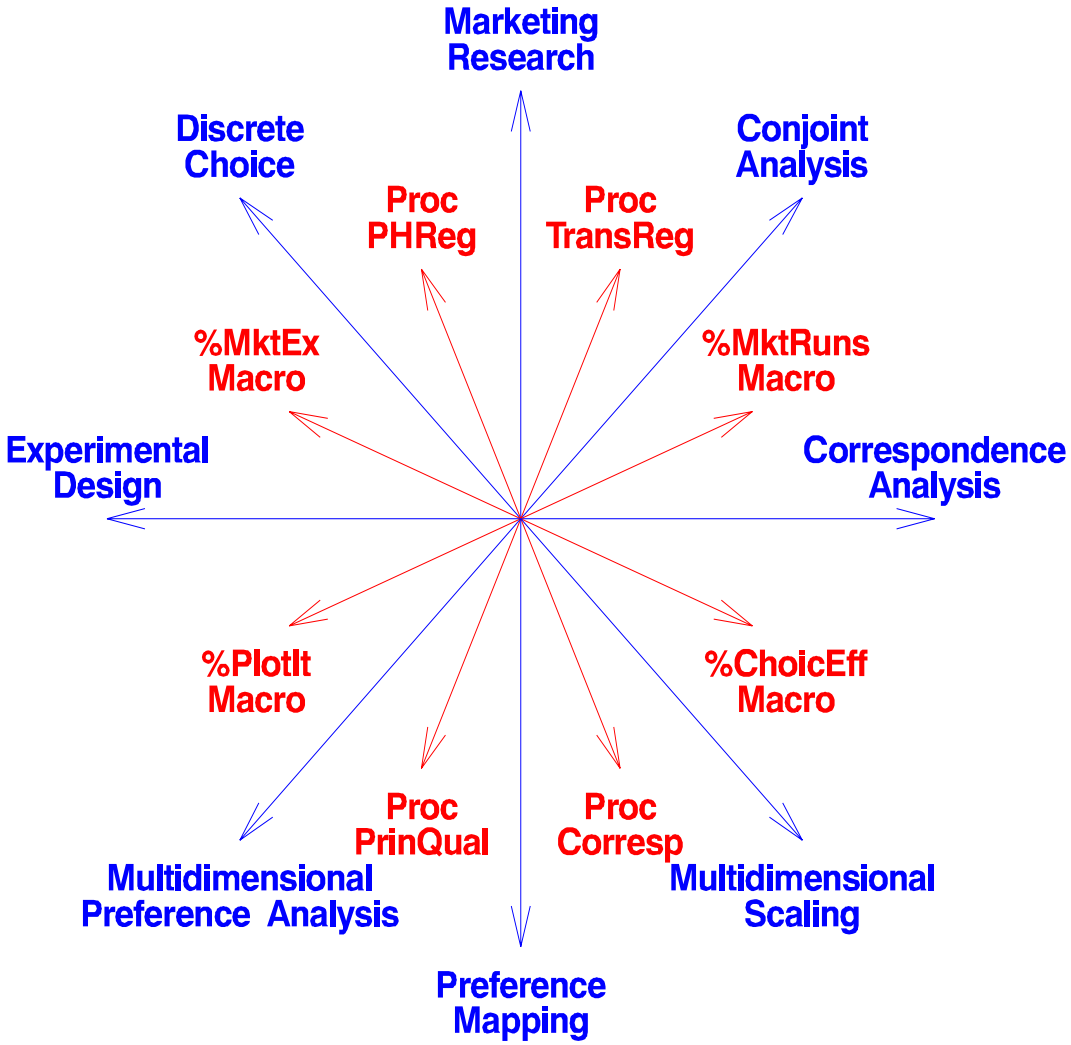
Marketing Research Methods in SAS

Experimental Design, Choice, Conjoint, and Graphical Techniques



Warren F. Kuhfeld

January 1, 2005
SAS 9.1 Edition
TS-722

Marketing Research Methods in SAS

Marketing Research Methods in SAS discusses experimental design, discrete choice, conjoint analysis, and graphical and perceptual mapping techniques. The book has grown and evolved over many years and many revisions. For example, the section on choice models grew from a two-page handout written by Dave DeLong over 12 years ago. This is the January 1, 2005 edition for SAS 9.1. All of the macros and code used in this book should work in SAS 8.2, 9.0, and 9.1. However, you must get the latest macros from the web. All other macros are obsolete. I hope that this book and tool set will help you do better research, do it quickly, and do it more easily. I would like to hear what you think. Many of my examples and enhancements to the software are based on feedback from people like you. If you would like to be added to a mailing list to receive periodic e-mail updates on SAS marketing research tools (probably no more than once every few months), e-mail Warren.Kuhfeld@sas.com. This list will not be sold or used for any other purpose. Copies of this book and all of the macros are available on the web http://support.sas.com/techsup/tnote/tnote_stat.html#market (reports beginning with "TS-722").

The following people contributed to writing portions of this book: Mark Garratt, Joel Huber, Ying So, Randy Tobias, Wayne Watson, and Klaus Zwerina. My parts could not have been written without the help of many people. I would like to thank Joel Huber, Ying So, Randy Tobias, and John Wurst. My involvement in the area of experimental design and choice modeling can be traced to several conversations with Mark Garratt in the early 1990's and then to the influence of Don Anderson, Joel Huber, Jordan Louviere, and Randy Tobias. Don Anderson, Warwick de Launey, Neil Sloane, Chung-yi Suen, J.C. Wang, and Yingshan Zhang kindly helped me with some of the orthogonal arrays in the `%MktEx` macro. Brad Jones helped me with coordinate exchange. Much of our current success with creating highly restricted designs is due to the difficult and very interesting design problems brought to me by Johnny Kwan. Randy Tobias has been a great colleague and a huge help to me over the years in all areas of experimental design, and many components of the `%MktEx` macro and other design macros are based on his ideas.

Finishing a 857-page book causes one to pause and reflect. As always, I am proud of this edition of the book and tools, however it is clear that I have stood on the shoulders of giants. There are a few other people that I would like to acknowledge. Without these people, I would have never been in the position to write a book such as this. From my undergraduate days at Kent State, I would like to thank Roy Lilly, Larry Melamed, my mentor Ben Newberry, and Steve Simnick. Sadly, Dr. Lilly recently passed away after a courageous battle with cancer. He was an outstanding teacher and scholar, and he helped me in more ways than he could ever know. From graduate school at UNC, I would like to thank Ron Helms, Keith Muller, and especially my advisor and mentor Forrest Young. From SAS, I would like to thank Warren Sarle, Bob Rodriguez, and all of my colleagues in SAS/STAT Research and Development. It is great to work with such a smart, talented, productive, and helpful group of people. Finally, I would like to thank my mother, my late father, and my stepfather Ed, for being so good to my Mom and for being such a wonderful grandfather to my children. Sadly, Ed passed away after a lengthy illness just as I was finishing this edition. We all miss him very much. I dedicate this book to my wife Peg and my children Megan and Rusty.

Warren F. Kuhfeld, Ph.D.
Manager, Multivariate Models R&D
SAS Institute Inc.
January 1, 2005

About this Edition

This edition is mostly a minor revision of the 2004 edition. Some new orthogonal arrays were added and there are a few other changes to the %MktEx macro. The %MktEx macro has grown and evolved a lot over the years. At first, %MktEx was going to be just a minor new design tool. Then it evolved to be our main design work horse, incorporating PROC FACTEX, PROC OPTEX, the %MktDes macro, and a great deal of new code. At that point, I envisioned it as a design solution not a design tool. I thought people would just specify a factor list, the number of runs, and maybe some interactions, and that would be it. %MktEx would do the rest with little user involvement. For many problems, I was correct. However, what I did not envision at that point was how %MktEx would open up whole new design frontiers, particularly in the area of large and highly-restricted designs. For those problems, %MktEx became a tool and not a solution. It takes a sophisticated user, many options, and a potentially complicated restrictions macro to make highly-restricted designs. Most of the new options in %MktEx are designed to make that process easier. Still, many highly-restricted design problems are hard, and I will continue to do what I can in future releases to make them easier.

The new %MktEx option, order=matrix=SAS-data-set, allows you to specify exactly what columns are worked on in what order and in what groups. This can be very useful for certain highly-restricted designs and designs with interactions. The init= data set is much more flexible now. You can initialize any arbitrary part of the design and let %MktEx search for the rest. One way you can use this is when you want to force perfect balance and orthogonality in certain specific factors. See page 390. The %MktEx balance= option has changed with this release. It now has a stage, based on the new mintry= option, where it just seeks to improve efficiency before imposing balance restrictions. This approach seems to be superior to the old approach. Also, you can now differentially weight the contributors to design badness when there are ad hoc restrictions and also balance= or partial= restrictions. Another new option, repeat= gives you more control on the algorithm with restricted designs. When you specify examine=i v the formatting for the information and variance matrices has been improved.

The %MktKey macro, which aids the creation of the Key data set, for use with the %MktRoll macro, has been changed. You can still use it the way you always did, but now it can rearrange the list into a matrix, and for simple generic designs, it can directly create the Key data set without cutting and pasting and running a subsequent data step.

In this edition, the **Experimental Design, Efficiency, Coding, and Choice Designs** chapter on pages 47-97 has been revised and some new material was added. If you are interested in choice modeling, read this chapter first. Among other things, this chapter now has a complete choice modeling example, from start to finish. Since it does not have all of the facets and nuances of the examples in the discrete choice chapter, it should be better than those examples in helping you get started. A new section starting on page 70 lists the steps in designing a choice experiment and analyzing choice data and points you to all of the examples of each step in the discrete choice chapter.

In this edition, almost all of the examples have been modified to use the %ChoicEff macro to evaluate the choice design under the assumption that $\boldsymbol{\beta} = \mathbf{0}$. This is in addition to using the %MktEval macro to evaluate the linear design. I really like the idea of using %ChoicEff specifying the most complicated model that you intend to use to ensure that all the right parameters are estimable before you collect any data.

The **Food Product Example with Asymmetry and Availability Cross Effects** example has had an error in it from the start. Previous versions confused cross effects and availability cross effects. That has been fixed with this edition. I apologize for the error.

A big part of this book is about experimental design. Efficient experimental-design software, like some other search software, is notorious for not finding the exact same results if anything changes (operating system, computer, SAS release, code version, compiler, math library, phase of the moon, and so on), and `%MktEx` is no exception. It will find the same design if you specify a random number seed and run the same macro over and over again on the same machine, but if you change anything, it might find a different design. The algorithm is seeking to optimize an efficiency function. All it takes is one little difference, such as two numbers being almost identical but different in the last bit, and the algorithm can start down a different path. We expect as things change and the code is enhanced that the designs will be similar. Sometimes two designs may even have the exact same efficiency, but they will not be identical. `%MktEx` and other efficient design software take every step that increases efficiency. One could conceive of an alternative algorithm that repeatedly evaluates every possible step and then takes only the largest one with fuzzing to ensure proper tie handling. Such an algorithm would be less likely to give different designs, but it would be *much* slower. Hence, we take the standard approach of using a fast algorithm that makes great designs, but not always the same designs.

For many editions and with every revision, I regenerated every design, every sample data set, every bit of output, and then made changes all over the text to refer to the new output. Many times I had to do this more than once when a particularly attractive enhancement that changed the results occured to me late in the writing cycle. It was difficult, tedious, annoying, error prone, and time consuming, and it really did not contribute much to the book since you would very likely be running under a different configuration than me and not get exactly the same answers as me, no matter what either you or I did. Starting with the January 2004 edition, I said enough is enough! For many versions now, in the accompanying sample code, I have hard-coded in the actual example design after the code so you could run the sample and reproduce my results. I am continuing to do that, however I have not redone every example. Expect to get similar but different results, and use the sample code if you want to get the exact same design that was in the book. I would rather spend my time giving you new capabilities than rewriting old examples that have not changed in any important way.

In this and every other edition, all of the data sets in the discrete choice and conjoint examples are artificial. As a software developer, I do not have access to real data. Even if I did, it would be hard to use since most of those chapters are about design. Of course the data need to come from people making judgments based on the design. If I had real data in an example, I would no longer be able to change and enhance the design strategy for that example. Many of the examples have changed many times over the years as better design software and strategies became available. In this edition, like all previous editions, the emphasis is on showing the best-known design strategies not on illustrating the analysis of real data.

The orthogonal array catalog is now complete to the best of my knowledge up through 143 runs with pretty good coverage from 144 to 513 runs. If you know of any orthogonal arrays that are not in it, please e-mail Warren.Kuhfeld@sas.com. Also, if you know how to construct any of these difference schemes, I would appreciate hearing from you: D(60, 36, 3), D(102, 51, 3), D(60, 21, 4), D(112, 64, 4), D(30, 15, 5), D(35, 17, 5), D(40, 25, 5), D(55, 17, 5), D(60, 25, 5), D(65, 25, 5), D(85, 35, 5), D(48, 10, 6), D(60, 11, 6), D(84, 16, 6), D(35, 11, 7), D(42, 18, 7), D(63, 28, 7), D(70, 18, 7), D(40, 8, 10), D(30, 7, 15), D(21, 6, 21). The notation D(r, c, s) refers to an $r \times c$ matrix of order s. For the first time with this release, the list of missing difference schemes does not contain any generalized Hadamard matrices.

I hope you like these enhancements. Feedback is welcome. Your feedback can help make these tools better.

Contents Overview

Marketing Research: Uncovering Competitive Advantages **21–34**
This chapter is based on a SUGI (SAS Users Group International) paper and provides a basic introduction to perceptual mapping, biplots, multidimensional preference analysis (MDPREF), preference mapping (PREFMAP or external unfolding), correspondence analysis, multidimensional scaling, and conjoint analysis.

Introducing the Market Research Analysis Application **35–46**
This SUGI paper discusses a point-and-click interface for conjoint analysis, correspondence analysis, and multidimensional scaling.

Experimental Design, Efficiency, Coding, and Choice Designs **47–97**
This chapter discusses experimental design including full-factorial designs, fractional-factorial designs, orthogonal arrays, nonorthogonal designs, choice designs, conjoint designs, design efficiency, orthogonality, balance, and coding. If you are interested in choice modeling, read this chapter first.

Efficient Experimental Design with Marketing Research Applications **99–120**
This chapter is based on a *Journal of Marketing Research* paper and discusses *D*-efficient experimental designs for conjoint and discrete-choice studies, orthogonal arrays, nonorthogonal designs, relative efficiency, and nonorthogonal design algorithms.

A General Method for Constructing Efficient Choice Designs **121–139**
This chapter discusses efficient designs for choice experiments.

Discrete Choice **141–465**
This chapter discusses the multinomial logit model and discrete choice experiments. This is the longest chapter in the book, and it contains numerous examples covering a wide range of choice experiments and choice designs. Study the chapter **Experimental Design, Efficiency, Coding, and Choice Designs** before tackling this chapter.

Multinomial Logit Models **467–481**
This SUGI paper discusses the multinomial logit model. A travel example is discussed.

Conjoint Analysis **483–596**
This chapter discusses conjoint analysis. Examples range from simple to complicated. Topics include design, data collection, analysis, and simulation. PROC TRANSREG documentation that describes just those options that are most likely to be used in a conjoint analysis is included.

Experimental Design and Choice Modeling Macros **597–784**
This chapter provides examples and documentation for all of the autocall macros used in this book.

Linear Models and Conjoint Analysis with Nonlinear Spline Transformations .. **785–802**
This chapter is based on an AMA ART (American Marketing Association Advanced Research Techniques) Forum paper and discusses splines, which are nonlinear functions that can be useful in regression and conjoint analysis.

Graphical Scatter Plots of Labeled Points **803–816**
This chapter is based on a paper that appeared in the SAS journal *Observations* that discusses a macro for graphical scatterplots of labeled points.

Graphical Methods for Marketing Research **817–828**
This chapter is based on a National Computer Graphics Association Conference presentation and discusses the mathematics of biplots, correspondence analysis, PREFMAP, and MDPREF.

Contents

Marketing Research: Uncovering Competitive Advantages **21**

Abstract . . . 21
Introduction . . . 21
Perceptual Mapping . . . 22
Conjoint Analysis . . . 31
Software . . . 32
Conclusions . . . 34

Introducing the Market Research Analysis Application **35**

Abstract . . . 35
Conjoint Analysis . . . 35
Discrete Choice Analysis . . . 38
Correspondence Analysis . . . 40
Multidimensional Preference Analysis . . . 43
Multidimensional Scaling . . . 44
Summary . . . 46
Acknowledgements . . . 46

Experimental Design, Efficiency, Coding, and Choice Designs **47**

Abstract . . . 47
Introduction . . . 47
The Basic Conjoint Experiment . . . 48
The Basic Choice Experiment . . . 48
Experimental Design Terminology . . . 49
Eigenvalues, Means, and Footballs . . . 52

Experimental Design Efficiency 53
Experimental Design: Rafts, Rulers, Alligators, and Stones 54
The Number of Factor Levels 59
Conjoint, Linear, and Choice Designs 60
Blocking the Choice Design 62
Efficiency of a Choice Design 62
Coding, Efficiency, Balance, and Orthogonality 64
Orthogonally Coding Price and Other Quantitative Attributes 68
Canonical Correlations 70
The Process of Designing a Choice Experiment 70
A Simple Choice Experiment from A to Z 73
Optimal Generic Choice Designs 89
Conclusions 96
Choice Design Glossary 96

Efficient Experimental Design with Marketing Research Applications 99

Abstract 99
Introduction 99
Design of Experiments 101
Design Comparisons 105
Design Considerations 107
Examples 111
Conclusions 116

A General Method for Constructing Efficient Choice Designs 121

Abstract 121
Introduction 121
Criteria For Choice Design Efficiency 122
A General Method For Efficient Choice Designs 124
Choice Design Applications 125
Conclusions 133
Appendix 136

Discrete Choice **141**

Abstract . . . 141

Introduction . . . 141

Experimental Design . . . 143

Customizing the Multinomial Logit Output . . . 143

Candy Example . . . 144

The Multinomial Logit Model . . . 144

The Input Data . . . 146

Choice and Survival Models . . . 149

Fitting the Multinomial Logit Model . . . 149

Multinomial Logit Model Results . . . 150

Fitting the Multinomial Logit Model, All Levels . . . 152

Probability of Choice . . . 154

Fabric Softener Example . . . 156

Set Up . . . 156

Designing the Choice Experiment . . . 158

Examining the Design . . . 160

The Randomized Design and Postprocessing . . . 163

From a Linear Design to a Choice Design . . . 164

Testing the Design Before Data Collection . . . 166

Generating the Questionnaire . . . 169

Entering the Data . . . 171

Processing the Data . . . 171

Binary Coding . . . 173

Fitting the Multinomial Logit Model . . . 174

Multinomial Logit Model Results . . . 175

Probability of Choice . . . 177

Custom Questionnaires . . . 178

Processing the Data for Custom Questionnaires . . . 182

Vacation Example . . . 184

Set Up . . . 185

Designing the Choice Experiment . . . 188

The %MktEx Macro Algorithm . . . 191
Examining the Design . . . 194
From a Linear Design to a Choice Design . . . 200
Testing the Design Before Data Collection . . . 203
Generating the Questionnaire . . . 206
Entering and Processing the Data . . . 208
Binary Coding . . . 209
Quantitative Price Effect . . . 213
Quadratic Price Effect . . . 216
Effects Coding . . . 218
Alternative-Specific Effects . . . 222
Vacation Example with Alternative-Specific Attributes . . . 229
Choosing the Number of Choice Sets . . . 230
Designing the Choice Experiment . . . 232
Ensuring that Certain Key Interactions are Estimable . . . 233
Examining the Design . . . 242
Blocking an Existing Design . . . 244
Testing the Design Before Data Collection . . . 248
Generating the Questionnaire . . . 251
Generating Artificial Data . . . 253
Reading, Processing, and Analyzing the Data . . . 254
Aggregating the Data . . . 259
Brand Choice Example with Aggregate Data . . . 261
Processing the Data . . . 261
Simple Price Effects . . . 263
Alternative-Specific Price Effects . . . 265
Mother Logit Model . . . 268
Aggregating the Data . . . 276
Choice and Breslow Likelihood Comparison . . . 282
Food Product Example with Asymmetry and Availability Cross Effects . . . 283
The Multinomial Logit Model . . . 283
Set Up . . . 284

Designing the Choice Experiment . . . 286
When You Have a Long Time to Search for an Efficient Design . . . 291
Examining the Design . . . 292
Designing the Choice Experiment, More Choice Sets . . . 294
Examining the Subdesigns . . . 305
Examining the Aliasing Structure . . . 306
Blocking the Design . . . 308
The Final Design . . . 311
Testing the Design Before Data Collection . . . 316
Generating Artificial Data . . . 330
Processing the Data . . . 331
Cross Effects . . . 333
Multinomial Logit Model Results . . . 334
Modeling Subject Attributes . . . 338
Allocation of Prescription Drugs . . . 345
Designing the Allocation Experiment . . . 345
Processing the Data . . . 351
Coding and Analysis . . . 357
Multinomial Logit Model Results . . . 358
Analyzing Proportions . . . 360
Chair Design with Generic Attributes . . . 363
Generic Attributes, Alternative Swapping, Large Candidate Set . . . 364
Generic Attributes, Alternative Swapping, Small Candidate Set . . . 370
Generic Attributes, a Constant Alternative, and Alternative Swapping . . . 374
Generic Attributes, a Constant Alternative, and Choice Set Swapping . . . 378
Design Algorithm Comparisons . . . 381
Initial Designs . . . 383
Improving an Existing Design . . . 383
When Some Choice Sets are Fixed in Advance . . . 385
Partial Profiles and Restrictions . . . 397
Pair-wise Partial-Profile Choice Design . . . 397
Linear Partial-Profile Design . . . 401

Choice from Triples; Partial Profiles Constructed Using Restrictions 403
Six Alternatives; Partial Profiles Constructed Using Restrictions 409
Five-Level Factors; Partial Profiles Constructed Using Restrictions 423
Partial Profiles and Incomplete Blocks Designs 435

Multinomial Logit Models **467**

Abstract 467
Introduction 467
Modeling Discrete Choice Data 469
Fitting Discrete Choice Models 470
Cross-Alternative Effects 475
Final Comments 480

Conjoint Analysis **483**

Abstract 483
Introduction 483
Conjoint Measurement 483
Conjoint Analysis 484
Choice-Based Conjoint 485
Experimental Design 485
The Output Delivery System 485
Chocolate Candy Example 489
Metric Conjoint Analysis 489
Nonmetric Conjoint Analysis 492
Frozen Diet Entrées Example (Basic) 496
Choosing the Number of Stimuli 496
Generating the Design 498
Evaluating and Preparing the Design 499
Printing the Stimuli and Data Collection 501
Data Processing 503
Nonmetric Conjoint Analysis 505
Frozen Diet Entrées Example (Advanced) 509

Creating a Design with the %MktEx Macro . . . 509
Designing Holdouts . . . 511
Print the Stimuli . . . 516
Data Collection, Entry, and Preprocessing . . . 516
Metric Conjoint Analysis . . . 521
Analyzing Holdouts . . . 535
Simulations . . . 537
Summarizing Results Across Subjects . . . 541
Spaghetti Sauce . . . 549
Create an Efficient Experimental Design with the %MktEx Macro . . . 549
Generating the Questionnaire . . . 557
Data Processing . . . 561
Metric Conjoint Analysis . . . 562
Simulating Market Share . . . 566
Simulating Market Share, Maximum Utility Model . . . 569
Simulating Market Share, Bradley-Terry-Luce and Logit Models . . . 575
Change in Market Share . . . 576
PROC TRANSREG Specifications . . . 585
PROC TRANSREG Statement . . . 585
Algorithm Options . . . 586
Output Options . . . 587
Transformations and Expansions . . . 588
Transformation Options . . . 590
BY Statement . . . 591
ID Statement . . . 591
WEIGHT Statement . . . 592
Monotone, Spline, and Monotone Spline Comparisons . . . 592
Samples of PROC TRANSREG Usage . . . 594
Metric Conjoint Analysis with Rating-Scale Data . . . 594
Nonmetric Conjoint Analysis . . . 594
Monotone Splines . . . 595
Constraints on the Utilities . . . 595

A Discontinuous Price Function 596

Experimental Design and Choice Modeling Macros 597

Abstract 597

Introduction 597

Changes and Enhancements 598

Installation 598

%ChoicEff Macro 600

%ChoicEff Macro Options 626

%ChoicEff Macro Notes 631

%MktAllo Macro 632

%MktAllo Macro Options 633

%MktAllo Macro Notes 634

%MktBal Macro 635

%MktBal Macro Options 636

%MktBal Macro Notes 637

%MktBlock Macro 638

%MktBlock Macro Options 644

%MktBlock Macro Notes 647

%MktDes Macro 648

%MktDes Macro Options 649

%MktDes Macro Notes 654

%MktDups Macro 655

%MktDups Macro Options 660

%MktDups Macro Notes 662

%MktEval Macro 663

%MktEval Macro Options 665

%MktEval Macro Notes 666

%MktEx Macro 667

%MktEx Macro Notes 677

%MktEx Macro Iteration History 678

%MktEx Macro Options 681

Advanced Restrictions . . . 700
%MktKey Macro . . . 710
%MktKey Macro Options . . . 711
%MktLab Macro . . . 712
%MktLab Macro Options . . . 719
%MktLab Macro Notes . . . 722
%MktMerge Macro . . . 723
%MktMerge Macro Options . . . 723
%MktMerge Macro Notes . . . 724
%MktOrth Macro . . . 725
%MktOrth Macro Options . . . 728
%MktOrth Macro Notes . . . 730
%MktPPro Macro . . . 731
%MktPPro Macro Options . . . 734
%MktPPro Macro Notes . . . 734
%MktRoll Macro . . . 735
%MktRoll Macro Options . . . 738
%MktRoll Macro Notes . . . 739
%MktRuns Macro . . . 740
%MktRuns Macro Options . . . 745
%MktRuns Macro Notes . . . 747
%PhChoice Macro . . . 748
%PhChoice Macro Options . . . 752
%PlotIt Macro . . . 753
%PlotIt Macro Options . . . 761
Macro Errors . . . 784

Linear Models and Conjoint Analysis with Nonlinear Spline Transformations 785
Abstract . . . 785
Why Use Nonlinear Transformations? . . . 785
Background and History . . . 786
The General Linear Univariate Model . . . 786

Polynomial Splines . . . 787
Splines with Knots . . . 788
Derivatives of a Polynomial Spline . . . 790
Discontinuous Spline Functions . . . 791
Monotone Splines and B-Splines . . . 793
Transformation Regression . . . 794
Degrees of Freedom . . . 795
Dependent Variable Transformations . . . 796
Scales of Measurement . . . 796
Conjoint Analysis . . . 797
Curve Fitting Applications . . . 797
Spline Functions of Price . . . 799
Benefits of Splines . . . 802
Conclusions . . . 802

Graphical Scatter Plots of Labeled Points 803

Abstract . . . 803
Introduction . . . 803
An Overview of the %PlotIt Macro . . . 804
Examples . . . 805
Availability . . . 816
Conclusions . . . 816

Graphical Methods for Marketing Research 817

Abstract . . . 817
Introduction . . . 817
Methods . . . 818
Notes . . . 828
Conclusions . . . 828

Concluding Remarks **829**

References **831**

Index **839**

Marketing Research: Uncovering Competitive Advantages

Warren F. Kuhfeld

Abstract

SAS provides a variety of methods for analyzing marketing data including conjoint analysis, correspondence analysis, preference mapping, multidimensional preference analysis, and multidimensional scaling. These methods allow you to analyze purchasing decision trade-offs, display product positioning, and examine differences in customer preferences. They can help you gain insight into your products, your customers, and your competition. This chapter discusses these methods and their implementation in SAS.*

Introduction

Marketing research is an area of applied data analysis whose purpose is to support marketing decision making. Marketing researchers ask many questions, including:

- Who are my customers?
- Who else should be my customers?
- Who are my competitors' customers?
- Where is my product positioned relative to my competitors' products?
- Why is my product positioned there?
- How can I reposition my existing products?
- What new products should I create?
- What audience should I target for my new products?

*This is a minor modification of a paper that was presented to SUGI 17 by Warren F. Kuhfeld and to the 1992 Midwest SAS Users Group meeting by Russell D. Wolfinger. Copies of this chapter (TS-722A) and all of the macros are available on the web http://support.sas.com/techsup/tnote/tnote_stat.html#market.

Marketing researchers try to answer these questions using both standard data analysis methods, such as descriptive statistics and crosstabulations, and more specialized marketing research methods. This chapter discusses two families of specialized marketing research methods, perceptual mapping and conjoint analysis. Perceptual mapping methods produce plots that display product positioning, product preferences, and differences between customers in their product preferences. Conjoint analysis is used to investigate how consumers trade off product attributes when making a purchasing decision.

Perceptual Mapping

Perceptual mapping methods, including correspondence analysis (CA), multiple correspondence analysis (MCA), preference mapping (PREFMAP), multidimensional preference analysis (MDPREF), and multidimensional scaling (MDS), are data analysis methods that generate graphical displays from data. These methods are used to investigate relationships among products as well as individual differences in preferences for those products.†

CA and MCA can be used to display demographic and survey data. CA simultaneously displays in a scatter plot the row and column labels from a two-way contingency table (crosstabulation) constructed from two categorical variables. MCA simultaneously displays in a scatterplot the category labels from more than two categorical variables.

MDPREF displays products positioned by overall preference patterns. MDPREF also displays differences in how customers prefer products. MDPREF displays in a scatter plot both the row labels (products) and column labels (consumers) from a data matrix of continuous variables.

MDS is used to investigate product positioning. MDS displays a set of object labels (products) whose perceived similarity or dissimilarity has been measured.

PREFMAP is used to interpret preference patterns and help determine why products are positioned where they are. PREFMAP displays rating scale data in the same plot as an MDS or MDPREF plot. PREFMAP shows both products and product attributes in one plot.

MDPREF, PREFMAP, CA, and MCA are all similar in spirit to the biplot, so first the biplot is discussed to provide a foundation for discussing these methods.

The Biplot. A *biplot* (Gabriel, 1981) simultaneously displays the row and column labels of a data matrix in a low-dimensional (typically two-dimensional) plot. The "bi" in "biplot" refers to the *joint* display of rows and columns, not to the dimensionality of the plot. Typically, the row coordinates are plotted as points, and the column coordinates are plotted as vectors.

Consider the artificial preference data matrix in Figure 1. Consumers were asked to rate their preference for products on a 0 to 9 scale where 0 means little preference and 9 means high preference. Consumer 1's preference for Product 1 is 4. Consumer 1's most preferred product is Product 4, which has a preference of 6.

†Also see pages 803 and 817.

	Consumer 1	Consumer 2	Consumer 3
Product 1	4	1	6
Product 2	4	2	4
Product 3	1	0	2
Product 4	6	2	8

Figure 1. Preference Data Matrix

$$\underset{Y}{\begin{bmatrix} 4 & 1 & 6 \\ 4 & 2 & 4 \\ 1 & 0 & 2 \\ 6 & 2 & 8 \end{bmatrix}} = \underset{A}{\begin{bmatrix} 1 & 2 \\ 2 & 0 \\ 0 & 1 \\ 2 & 2 \end{bmatrix}} \times \underset{B'}{\begin{bmatrix} 2 & 1 & 2 \\ 1 & 0 & 2 \end{bmatrix}}$$

Figure 2. Preference Data Decomposition

The biplot is based on the idea of a matrix decomposition. The $(n \times m)$ data matrix $\mathbf{Y}$ is decomposed into the product of an $(n \times q)$ matrix $\mathbf{A}$ and a $(q \times m)$ matrix $\mathbf{B}'$. Figure 2 shows a decomposition of the data in Figure 1.[‡] The rows of $\mathbf{A}$ are coordinates in a two-dimensional plot for the row points in $\mathbf{Y}$, and the columns of $\mathbf{B}'$ are coordinates in the same two-dimensional plot for the column points in $\mathbf{Y}$. In this artificial example, the entries in $\mathbf{Y}$ are exactly reproduced by *scalar products* of coordinates. For example, the $(1,1)$ entry in $\mathbf{Y}$ is $y_{11} = a_{11} \times b_{11} + a_{12} \times b_{12} = 4 = 1 \times 2 + 2 \times 1$.

The rank of $\mathbf{Y}$ is $q \leq MIN(n, m)$. The rank of a matrix is the minimum number of dimensions that are required to represent the data without loss of information. The rank of $\mathbf{Y}$ is the full number of columns in $\mathbf{A}$ and $\mathbf{B}$. In the example, $q = 2$. When the rows of $\mathbf{A}$ and $\mathbf{B}$ are plotted in a two-dimensional scatter plot, the scalar product of the coordinates of $\mathbf{a}'_i$ and $\mathbf{b}'_j$ *exactly* equals the data value y_{ij}. This kind of scatter plot is a biplot. When $q > 2$ and the first two dimensions are plotted, then $\mathbf{AB}'$ is *approximately* equal to $\mathbf{Y}$, and the display is an *approximate biplot*.[§] The best values for $\mathbf{A}$ and $\mathbf{B}$, in terms of minimum squared error in approximating $\mathbf{Y}$, are found using a singular value decomposition (SVD).[¶] An approximate biplot is constructed by plotting the first two columns of $\mathbf{A}$ and $\mathbf{B}$.

When $q > 2$, the full geometry of the data cannot be represented in two dimensions. The first two columns of $\mathbf{A}$ and $\mathbf{B}$ provide the best approximation of the high dimensional data in two dimensions. Consider a cloud of data in the shape of an American football. The data are three dimensional. The best one dimensional representation of the data—*the first principal component*—is the line that runs from one end of the football, through the center of gravity or *centroid* and to the other end. It is the longest line that can run through the football. The second principal component also runs through the centroid and is perpendicular or *orthogonal* to the first line. It is the longest line that can be drawn through the centroid that is perpendicular to the first. If the football is a little thicker at the laces, the second principal component runs from the laces through the centroid and to the other side of the football. All of the points in the football shaped cloud can be projected into the plane of the first two principal components. The resulting scatter plot will show the approximate shape of the data. The two longest dimensions are shown, but the information in the other dimensions are lost. This is the principle behind approximate biplots. See Gabriel (1981) for more information on the biplot.

[‡]Figure 2 does not contain the decomposition that would be used for an actual biplot. Small integers were chosen to simplify the arithmetic.

[§]In practice, the term biplot is sometimes used without qualification to refer to an approximate biplot.

[¶]SVD is sometimes referred to in the psychometric literature as an Eckart-Young (1936) decomposition. SVD is closely tied to the statistical method of principal component analysis.

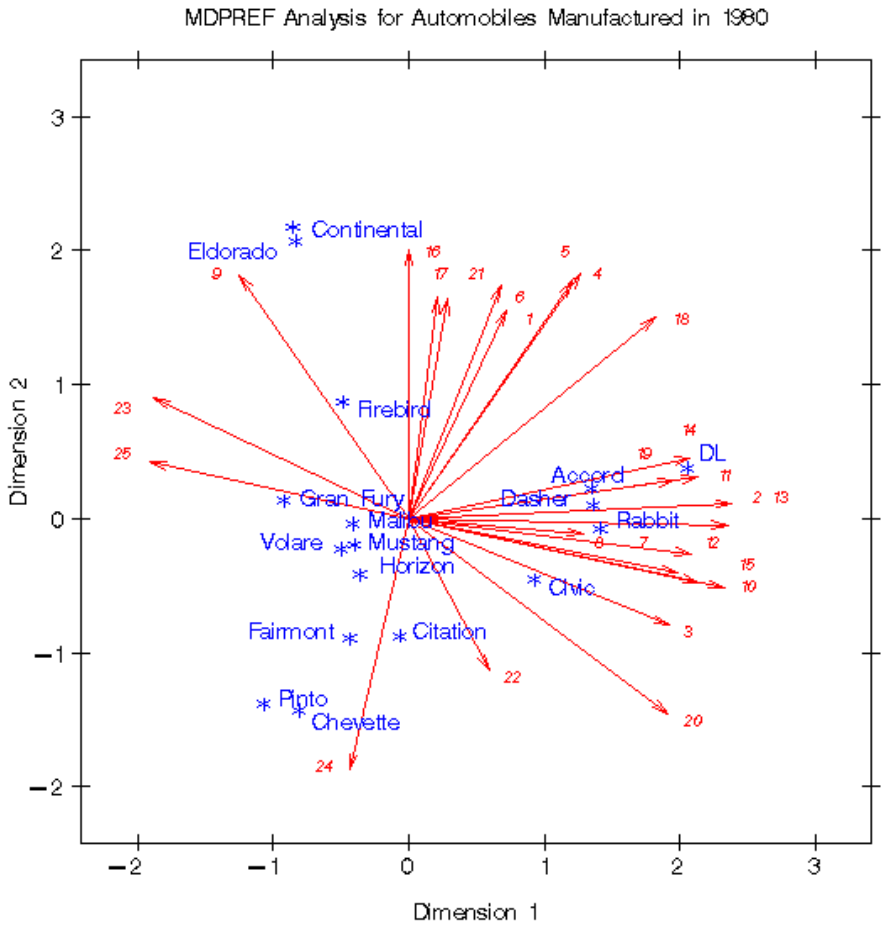


Figure 3. Multidimensional Preference Analysis

Multidimensional Preference Analysis. Multidimensional Preference Analysis (Carroll, 1972) or MDPREF is a biplot analysis for preference data. Data are collected by asking respondents to rate their preference for a set of objects—products in marketing research.

Questions that can be addressed with MDPREF analyses include: Who are my customers? Who else should be my customers? Who are my competitors' customers? Where is my product positioned relative to my competitors' products? What new products should I create? What audience should I target for my new products?

For example, consumers were asked to rate their preference for a group of automobiles on a 0 to 9 scale, where 0 means no preference and 9 means high preference. $\mathbf{Y}$ is an $(n \times m)$ matrix that contains ratings of the n products by the m consumers. Figure 3 displays an example in which 25 consumers rated their preference for 17 new (at the time) 1980 automobiles. Each consumer is a vector in the space, and each car is a point identified by an asterisk (*). Each consumer's vector points in *approximately* the direction of the cars that the consumer most preferred.

The dimensions of this plot are the first two principal components. The plot differs from a proper biplot of $\mathbf{Y}$ due to scaling factors. At one end of the plot of the first principal component are the most preferred automobiles; the least preferred automobiles are at the other end. The American cars on the

average were least preferred, and the European and Japanese cars were most preferred. The second principal component is the longest dimension that is orthogonal to the first principal component. In the example, the larger cars tend to be at the top and the smaller cars tend to be at the bottom.

The automobile that projects farthest along a consumer vector is that consumer's most preferred automobile. To project a point onto a vector, draw an imaginary line through a point crossing the vector at a right angle. The point where the line crosses the vector is the *projection*. The length of this projection differs from the predicted preference, the scalar product, by a factor of the length of the consumer vector, which is constant within each consumer. Since the goal is to look at projections of points onto the vectors, the absolute length of a consumer's vector is unimportant. The relative lengths of the vectors indicate fit, with longer vectors indicating better fit. The coordinates for the endpoints of the vectors were multiplied by 2.5 to extend the vectors and create a better graphical display. The direction of the preference scale is important. The vectors point in the direction of increasing values of the data values. If the data had been ranks, with 1 the most preferred and n the least preferred, then the vectors would point in the direction of the least preferred automobiles.

Consumers 9 and 16, in the top left portion of the plot, most prefer the large American cars. Other consumers, with vectors pointing up and nearly vertical, also show this pattern of preference. There is a large cluster of consumers, from 14 through 20, who prefer the Japanese and European cars. A few consumers, most notably consumer 24, prefer the small and inexpensive American cars. There are no consumer vectors pointing through the bottom left portion of the plot between consumers 24 and 25, which suggests that the smaller American cars are generally not preferred by any of these consumers.

Some cars have a similar pattern of preference, most notably Continental and Eldorado. This indicates that marketers of Continental or Eldorado may want to try to distinguish their car from the competition. Dasher, Accord, and Rabbit were rated similarly, as were Malibu, Mustang, Volare, and Horizon. Several vectors point into the open area between Continental/Eldorado and the European and Japanese cars. The vectors point away from the small American cars, so these consumers do not prefer the small American cars. What car would these consumers like? Perhaps they would like a Mercedes or BMW.

Preference Mapping. Preference mapping[‖] (Carroll, 1972) or PREFMAP plots resemble biplots, but are based on a different model. The goal in PREFMAP is to project external information into a configuration of points, such as the set of coordinates for the cars in the MDPREF example in Figure 3. The external information can aid interpretation.

Questions that can be addressed with PREFMAP analyses include: Where is my product positioned relative to my competitors' products? Why is my product positioned there? How can I reposition my existing products? What new products should I create?

[‖] Preference mapping is sometimes referred to as external unfolding.

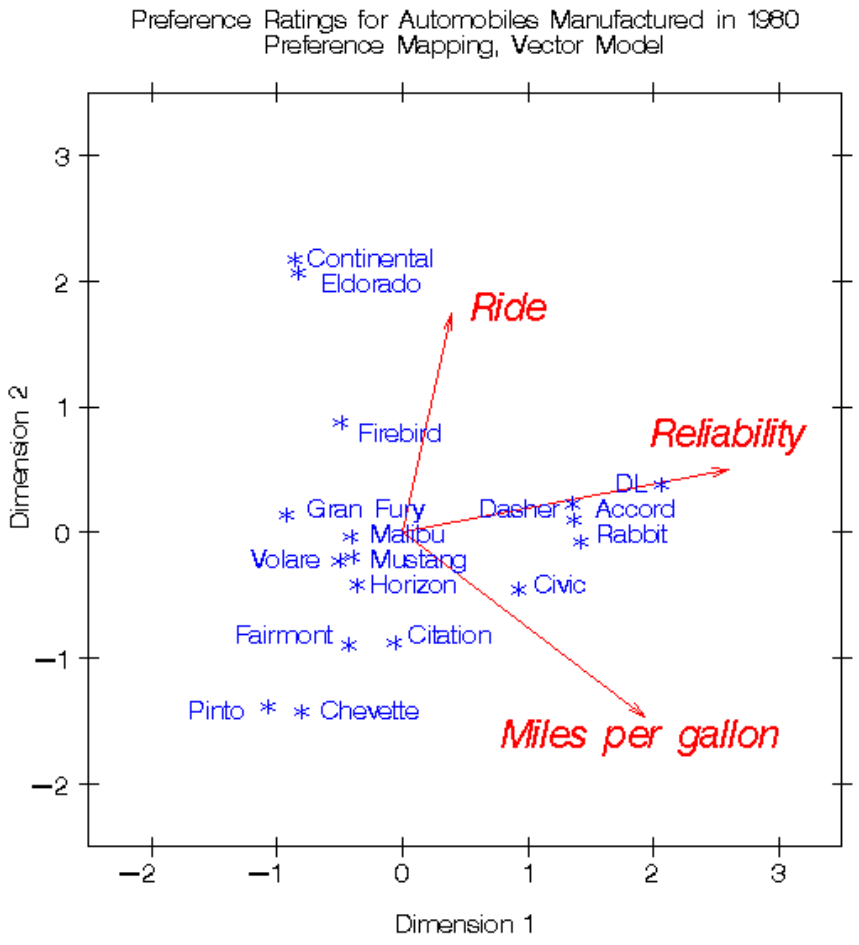


Figure 4. Preference Mapping, Vector Model

The PREFMAP Vector Model. Figure 4 contains an example in which three attribute variables (ride, reliability, and miles per gallon) are displayed in the plot of the first two principal components of the car preference data. Each of the automobiles was rated on a 1 to 5 scale, where 1 is poor and 5 is good. The end points for the attribute vectors are obtained by projecting the attribute variables into the car space. Orthogonal projections of the car points on an attribute vector give an approximate ordering of the cars on the attribute rating. The ride vector points almost straight up, indicating that the larger cars, such as the Eldorado and Continental, have the best ride. Figure 3 shows that most consumers preferred the DL, Japanese cars, and larger American cars. Figure 4 shows that the DL and Japanese cars were rated the most reliable and have the best fuel economy. The small American cars were not rated highly on any of the three dimensions.

Figure 4 is based on the simplest version of PREFMAP—the *vector model*. The vector model operates under the assumption that some is good and more is *always* better. This model is appropriate for miles per gallon and reliability—the more miles you can travel without refueling or breaking down, the better.

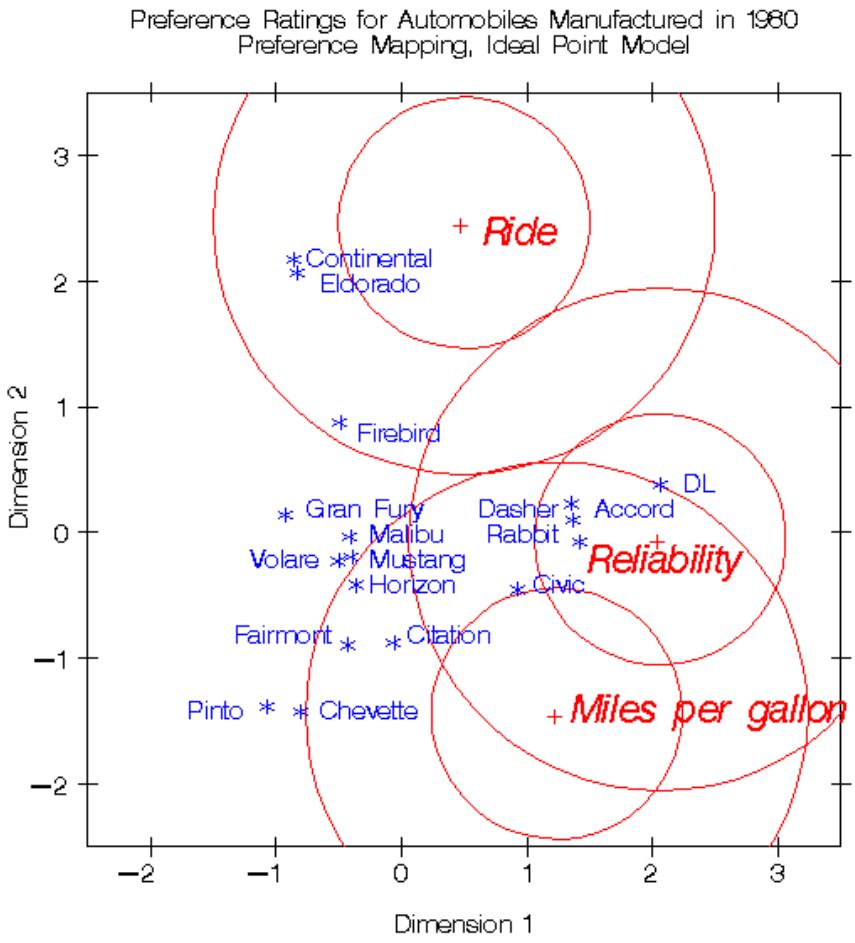


Figure 5. Preference Mapping, Ideal Point Model

The PREFMAP Ideal Point Model. The *ideal point* model differs from the vector model, in that the ideal point model does not assume that more is better, *ad infinitum*. Consider the sugar content of cake. There is an ideal amount of sugar that cake should contain—not enough sugar is not good, and too much sugar is also not good. In the cars example, the ideal number of miles per gallon and the ideal reliability are unachievable. It makes sense to consider a vector model, because the ideal point is infinitely far away. This argument is less compelling for ride; the point for a car with smooth, quiet ride may not be infinitely far away. Figure 5 shows the results of fitting an ideal point model for the three attributes. In the vector model, results are interpreted by orthogonally projecting the car points on the attribute vectors. In the ideal point model, Euclidean distances between car points and ideal points are compared. Eldorado and Continental have the best predicted ride, because they are closest to the ride ideal point. The concentric circles drawn around the ideal points help to show distances between the cars and the ideal points. The numbers of circles and their radii are arbitrary. The overall interpretations of Figures 4 and 5 are the same. All three ideal points are at the edge of the car points, which suggests the simpler vector model is sufficient for these data. The ideal point model is fit with a multiple regression model and some pre- and post-processing. The regression model uses the MDS or MDPREF coordinates as independent variables along with an additional independent variable that is the sum of squares of the coordinates. The model is a constrained *response-surface model*.

The results in Figure 5 were modified from the raw results to eliminate *anti-ideal points*. The ideal point model is a distance model. The rating data are interpreted as distances between attribute ideal points and the products. In this example, each of the automobiles was rated on these three dimensions, on a 1 to 5 scale, where 1 is poor and 5 is good. The data are the reverse of what they should be—a ride rating of 1 should mean this car is similar to a car with a good ride, and a rating of 5 should mean this car is different from a car with a good ride. So the raw coordinates must be multiplied by −1 to get ideal points. Even if the scoring had been reversed, anti-ideal points can occur. If the coefficient for the sum-of-squares variable is negative, the point is an anti-ideal point. In this example, there is the possibility of *anti-anti-ideal points*. When the coefficient for the sum-of-squares variable is negative, the two multiplications by −1 cancel, and the coordinates are ideal points. When the coefficient for the sum-of-squares variable is positive, the coordinates are multiplied by −1 to get an ideal point.

Correspondence Analysis. Correspondence analysis (CA) is used to find a low-dimensional graphical representation of the association between rows and columns of a contingency table (crosstabulation). It graphically shows relationships between the rows and columns of a table; it graphically shows the relationships that the ordinary chi-square statistic tests. Each row and column is represented by a point in a Euclidean space determined from cell frequencies. CA is a popular data analysis method in France and Japan. In France, CA analysis was developed under the strong influence of Jean-Paul Benzécri; in Japan, under Chikio Hayashi. CA is described in Lebart, Morineau, and Warwick (1984); Greenacre (1984); Nishisato (1980); Tenenhaus and Young (1985); Gifi (1990); Greenacre and Hastie (1987); and many other sources. Hoffman and Franke (1986) provide a good introductory treatment using examples from marketing research.

Questions that can be addressed with CA and MCA include: Who are my customers? Who else should be my customers? Who are my competitors' customers? Where is my product positioned relative to my competitors' products? Why is my product positioned there? How can I reposition my existing products? What new products should I create? What audience should I target for my new products?

MCA Example. Figure 6 contains a plot of the results of a multiple correspondence analysis (MCA) of a survey of car owners. The questions included origin of the car (American, Japanese, European), size of car (small, medium, large), type of car (family, sporty, work vehicle), home ownership (owns, rents), marital/family status (single, married, single and living with children, and married living with children), and sex (male, female). The variables are all categorical.

The top-right quadrant of the plot suggests that the categories single, single with kids, one income, and renting a home are associated. Proceeding clockwise, the categories sporty, small, and Japanese are associated. In the bottom-left quadrant you can see the association between being married, owning your own home, and having two incomes. Having children is associated with owning a large American family car. Such information can be used to identify target audiences for advertisements. This interpretation is based on points being located in approximately the same direction from the origin and in approximately the same region of the space. Distances between points are not interpretable in MCA.

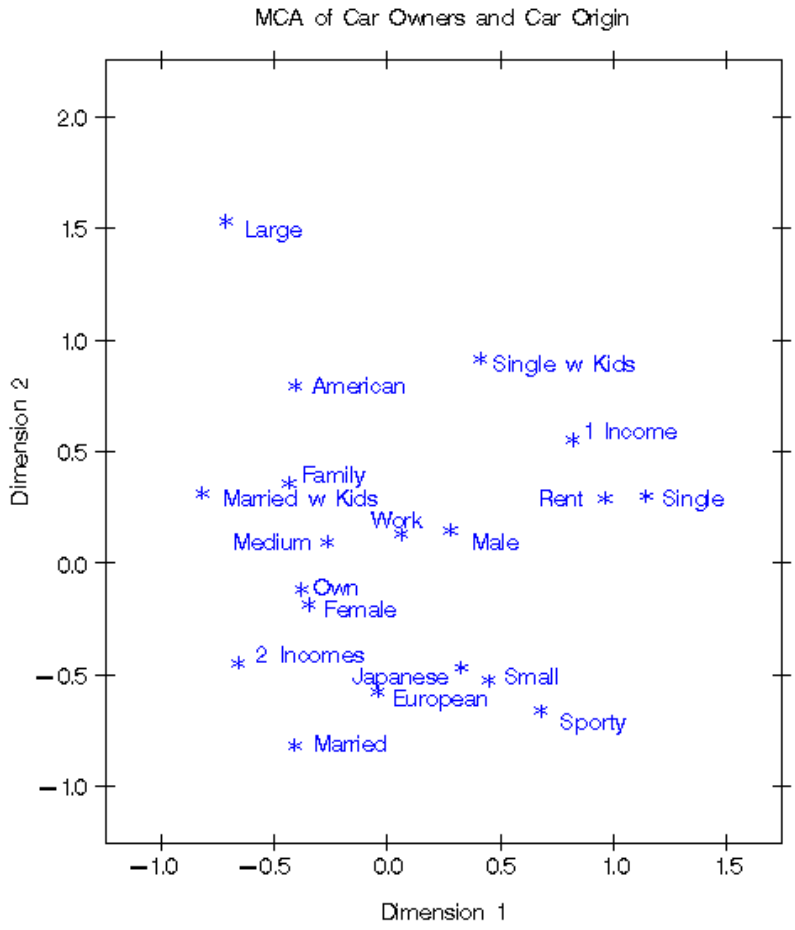


Figure 6. Multiple Correspondence Analysis

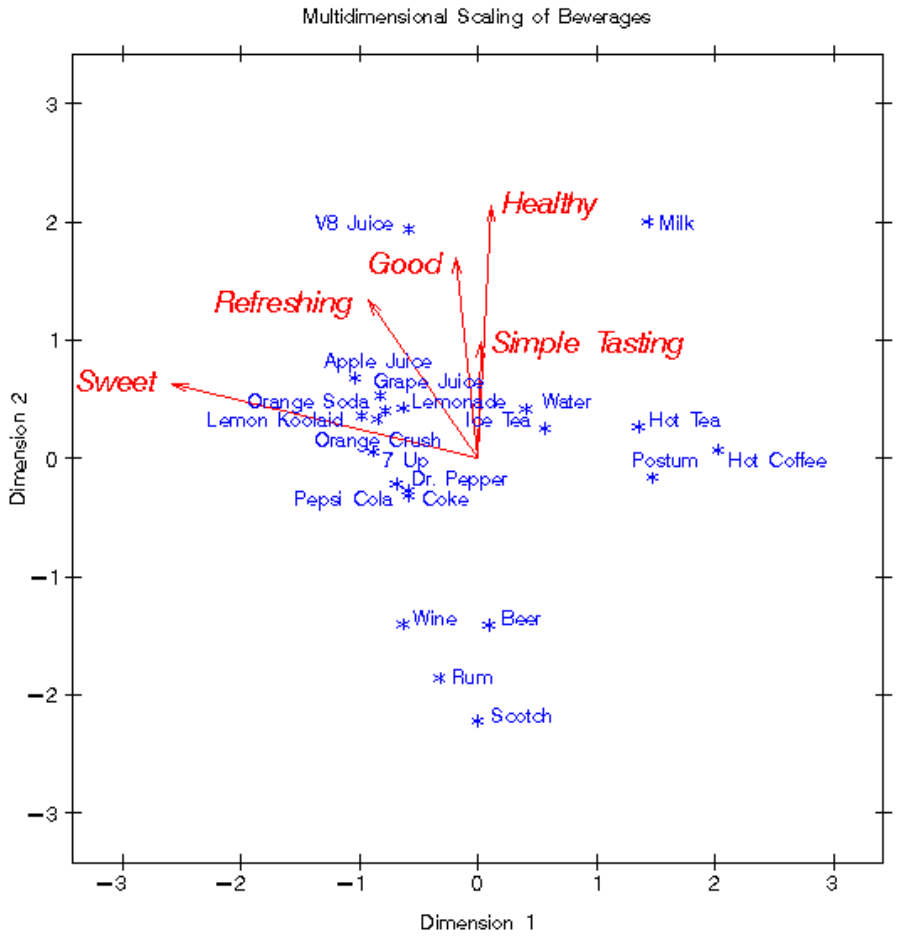


Figure 7. MDS and PREFMAP

Multidimensional Scaling. Multidimensional scaling (MDS) is a class of methods for estimating the coordinates of a set of objects in a space of specified dimensionality from data measuring the distances between pairs of objects (Kruskal and Wish, 1978; Schiffman, Reynolds, and Young, 1981; Young, 1987). The data for MDS consist of one or more square symmetric or asymmetric matrices of similarities or dissimilarities between objects or stimuli. Such data are also called *proximity data*. In marketing research, the objects are often products. MDS is used to investigate product positioning.

For example, consumers were asked to rate the differences between pairs of beverages. In addition, the beverages were rated on adjectives such as Good, Sweet, Healthy, Refreshing, and Simple Tasting. Figure 7 contains a plot of the beverage configuration along with attribute vectors derived through preference mapping. The alcoholic beverages are clustered at the bottom. The juices and carbonated soft drinks are clustered at the left. Grape and Apple juice are above the carbonated and sweet soft drinks and are perceived as more healthy than the other soft drinks. Perhaps sales of these drinks would increase if they were marketed as a healthy alternative to sugary soft drinks. A future analysis, after a marketing campaign, could check to see if their positions in the plot change in the healthy direction.

Water, coffee and tea drinks form a cluster at the right. V8 Juice and Milk form two clusters of one point each. Milk and V8 are perceived as the most healthy, whereas the alcoholic beverages are perceived as least healthy. The juices and carbonated soft drinks were rated as the sweetest. Pepsi and Coke are mapped to coincident points. Postum (a coffee substitute) is near Hot Coffee, Orange Soda is near Orange Crush, and Lemon Koolaid is near Lemonade.

Geometry of the Scatter Plots. It is important that scatter plots displaying perceptual mapping information accurately portray the underlying geometry. All of the scatter plots in this chapter were created with the axes equated so that a centimeter on the y-axis represents the same data range as a centimeter on the x-axis.** *This is important.* Distances, angles between vectors, and projections are evaluated to interpret the plots. When the axes are equated, distances and angles are correctly presented in the plot. When axes are scaled independently, for example to fill the page, then the correct geometry is not presented. This important step of equating the axes is often overlooked in practice.

For MDPREF and PREFMAP, the absolute lengths of the vectors are not important since the goal is to project points on vectors, not look at scalar products of row points and column vectors. It is often necessary to change the lengths of *all* of the vectors to improve the graphical display. If all of the vectors are relatively short with end points clustered near the origin, the display will be difficult to interpret. To avoid this problem in Figure 3, *both* the x-axis and y-axis coordinates were multiplied by the same constant, 2.5, to lengthen all vectors by the same relative amount. The coordinates must *not* be scaled independently.

Conjoint Analysis

Conjoint analysis is used in marketing research to analyze consumer preferences for products and services. See Green and Rao (1971) and Green and Wind (1975) for early introductions to conjoint analysis and Green and Srinivasan (1990) for a recent review article.

Conjoint analysis grew out of the area of *conjoint measurement* in mathematical psychology. In its original form, *conjoint analysis* is a main effects analysis-of-variance problem with an ordinal scale-of-measurement dependent variable. Conjoint analysis decomposes rankings or rating-scale evaluation judgments of products into components based on qualitative attributes of the products. Attributes can include price, color, guarantee, environmental impact, and so on. A numerical *utility* or *part-worth utility* value is computed for each level of each attribute. The goal is to compute utilities such that the rank ordering of the sums of each product's set of utilities is the same as the original rank ordering or violates that ordering as little as possible.

When a monotonic transformation of the judgments is requested, a *nonmetric conjoint analysis* is performed. Nonmetric conjoint analysis models are fit iteratively. When the judgments are not transformed, a *metric conjoint analysis* is performed. Metric conjoint analysis models are fit directly with ordinary least squares. When all of the attributes are nominal, the metric conjoint analysis problem is a simple main-effects ANOVA model. The attributes are the independent variables, the judgments comprise the dependent variable, and the utilities are the parameter estimates from the ANOVA model. The metric conjoint analysis model is more restrictive than the nonmetric model and will generally fit the data less well than the nonmetric model. However, this is not necessarily a disadvantage since over-fitting is less of a problem and the results should be more reproducible with the metric model.

**If the plot axes are not equated in this chapter, it is due to unequal distortions of the axes that occurred during the final printing process.

In both metric and nonmetric conjoint analysis, the respondents are typically not asked to rate all possible combinations of the attributes. For example, with five attributes, three with three levels and two with two levels, there are $3 \times 3 \times 3 \times 2 \times 2 = 108$ possible combinations. Rating that many combinations would be difficult for consumers, so typically only a small fraction of the combinations are rated. It is still possible to compute utilities, even if not all combinations are rated. Typically, combinations are chosen from an *orthogonal array* which is a *fractional-factorial design*. In an orthogonal array, the zero/one indicator variables are uncorrelated for all pairs in which the two indicator variables are not from the same factor. The main effects are orthogonal but are confounded with interactions. These interaction effects are typically assumed to be zero.

Questions that can be addressed with conjoint analysis include: How can I reposition my existing products? What new products should I create? What audience should I target for my new products?

Consider an example in which the effects of four attributes of tea on preference were evaluated. The attributes are temperature (Hot, Warm, and Iced), sweetness (No Sugar, 1 Teaspoon, 2 Teaspoons), strength (Strong, Moderate, Weak), and lemon (With Lemon, No Lemon). There are four factors: three with three levels and one with two levels. Figure 8 contains the results.††

Sweetness was the most important attribute (the importance is 55.795). This consumer preferred two teaspoons of sugar over one teaspoon, and some sugar was preferred over no sugar. The second most important attribute was strength (25.067), with moderate and strong tea preferred over weak tea. This consumer's most preferred temperature was iced, and no lemon was preferred over lemon.

Software

SAS includes software that implements these methods. SAS/STAT software was used to perform the analyses for all of the examples. Perceptual mapping methods are described with more mathematical detail starting on page 817.

Correspondence Analysis. The SAS/STAT procedure CORRESP performs simple and multiple correspondence analysis and outputs the coordinates for plotting. Raw data or tables may be input. Supplementary classes are allowed.

Multidimensional Preference Analysis. The SAS/STAT procedure PRINQUAL performs multidimensional preference analysis and outputs the coordinates for plotting. Nonmetric MDPREF, with transformations of continuous and categorical variables, is also available.

Preference Mapping. The SAS/STAT procedure TRANSREG performs preference mapping and outputs the coordinates. Nonmetric PREFMAP, with transformations of continuous and categorical variables, is also available.

Multidimensional Scaling. The SAS/STAT procedure MDS performs multidimensional scaling and outputs the coordinates. Metric, nonmetric, two-way, and three-way models are available.

††See page 483 for more information on conjoint analysis. Note that the results in Figure 8 have been customized using ODS. See page 485 for more information on customizing conjoint analysis output.

Conjoint Analysis of Tea-Tasting Data

The TRANSREG Procedure

The TRANSREG Procedure Hypothesis Tests for Linear(subj2)

Univariate ANOVA Table Based on the Usual Degrees of Freedom

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	7	617.7222	88.24603	32.95	<.0001
Error	10	26.7778	2.67778		
Corrected Total	17	644.5000			

Root MSE	1.63639	R-Square	0.9585
Dependent Mean	12.16667	Adj R-Sq	0.9294
Coeff Var	13.44979		

Utilities Table Based on the Usual Degrees of Freedom

Label	Utility	Standard Error	Importance (% Utility Range)
Intercept	12.1667	0.38570	
Lemon: No	0.7222	0.38570	7.008
Lemon: Yes	-0.7222	0.38570	
Temperature: Hot	0.5000	0.54546	12.129
Temperature: Iced	1.0000	0.54546	
Temperature: Warm	-1.5000	0.54546	
Sweetness: No Sugar	-7.3333	0.54546	55.795
Sweetness: 1 Teaspoon	3.1667	0.54546	
Sweetness: 2 Teaspoons	4.1667	0.54546	
Strength: Moderate	1.8333	0.54546	25.067
Strength: Strong	1.5000	0.54546	
Strength: Weak	-3.3333	0.54546	

Figure 8. Conjoint Analysis

Scatter Plots. The Base SAS procedure PLOT can plot the results from these analyses and optimally position labels in the scatter plot. PROC PLOT uses an algorithm, developed by Kuhfeld (1991), that uses a heuristic approach to avoid label collisions. Labels up to 200 characters long can be plotted.

The `%PlotIt` macro, was used to create graphical scatter plots of labeled points. There are options to draw vectors to certain symbols and draw circles around other symbols. This macro is in the SAS autocall macro library. Also see page 803.

Conjoint Analysis. The SAS/STAT procedure TRANSREG can perform both metric and nonmetric conjoint analysis. PROC TRANSREG can handle both *holdout* observations and *simulations*. Holdouts are ranked by the consumers but are excluded from contributing to the analysis. They are used to validate the results of the study. Simulation observations are not rated by the consumers and do not contribute to the analysis. They are scored as passive observations. Simulations are *what-if* combinations. They are combinations that are entered to get a prediction of what their utility would have been if they had been rated. Conjoint analysis is described in more detail starting on page 483.

The `%MktEx` macro can generate orthogonal designs for both main-effects models and models with interactions. Nonorthogonal designs—for example, when strictly orthogonal designs require too many observations—can also be generated. Nonorthogonal designs can be used in conjoint analysis studies to minimize the number of stimuli when there are many attributes and levels. This macro is in the SAS autocall macro library and is also available free of charge on the web: http://support.sas.com/techsup/tnote/tnote_stat.html#market. Experimental design and the `%MktEx` macro are described in more detail in starting on pages 47, 99, 121, 141, 483, 597, and 667.

Other Data Analysis Methods. Other procedures that are useful for marketing research include the SAS/STAT procedures for regression, ANOVA, discriminant analysis, principal component analysis, factor analysis, categorical data analysis, covariance analysis (structural equation models), and the SAS/ETS procedures for econometrics, time series, and forcasting. Discrete choice data can be analyzed with multinomial logit models using the PHREG procedure. Discrete choice is described in more detail in starting on page 141.

Conclusions

Marketing research helps you understand your customers and your competition. Correspondence analysis compactly displays survey data to aid in determining what kinds of consumers are buying your products. Multidimensional preference analysis and multidimensional scaling show product positioning, group preferences, and individual preferences. Plots from these methods may suggest how to reposition your product to appeal to a broader audience. They may also suggest new groups of customers to target. Preference mapping is used as an aid in understanding MDPREF and MDS results. PREFMAP displays product attributes in the same plot as the products. Conjoint analysis is used to investigate how consumers trade off product attributes when making a purchasing decision.

The insight gained from perceptual mapping and conjoint analysis can be a valuable asset in marketing decision making. These techniques can help you gain insight into your products, your customers, and your competition. They can give you the edge in gaining a competitive advantage.

Introducing the Market Research Analysis Application

Wayne E. Watson

Abstract

Market research focuses on assessing the preferences and choices of consumers and potential consumers. A new component of SAS/STAT software in Release 6.11 of the SAS System is an application written in SAS/AF that provides statistical and graphical techniques for market research data analysis. The application allows you to employ statistical methods such as conjoint analysis, discrete choice analysis, correspondence analysis, and multidimensional scaling through intuitive point-and-click actions.*

Conjoint Analysis

Conjoint analysis is used to evaluate consumer preference. If products are considered to be composed of attributes, conjoint analysis can be used to determine what attributes are important to product preference and what combinations of attribute levels are most preferred.

Usually, conjoint analysis is a main-effects analysis of variance of ordinally-scaled dependent variables. Preferences are used as dependent variables, and attributes are used as independent variables. Often, a monotone transformation is used with the dependent variables to fit a model with no interactions.

As an example, suppose you have four attributes that you think are related to automobile tire purchase. You want to know how important each attribute is to consumers' stated preferences for a potential tire purchase. The four attributes under investigation are

- brand name
- expected tread mileage
- purchase price
- installation cost

The attributes of brand name, tread mileage, and purchase price have three possible values and installation cost has two values. The values for each attribute are:

*For current documentation on the Market Research Application see SAS Institute Inc, *Getting Started with The Market Research Application*, Cary, NC: SAS Institute Inc., 1997, 56 pp. This paper was written and presented at SUGI 20 (1995) by Wayne E. Watson. This paper was also presented to SUGI-Korea (1995) by Warren F. Kuhfeld. Wayne Watson is a Research Statistician at SAS and wrote the Marketing Research Application which uses procedures and macros written by Warren F. Kuhfeld. Copies of this chapter (TS-722B) and all of the macros are available on the web http://support.sas.com/techsup/tnote/tnote_stat.html#market.

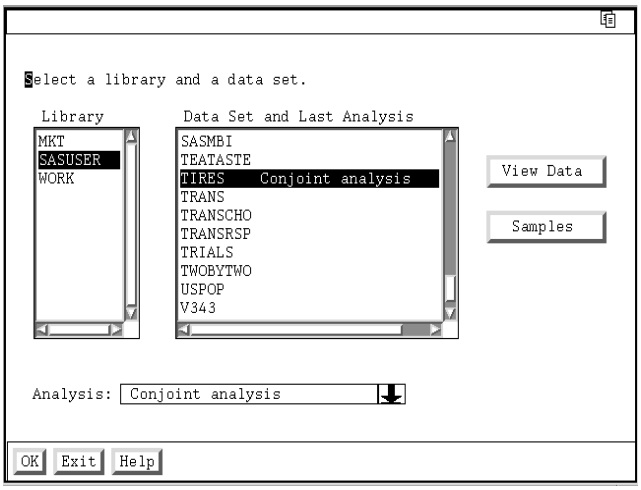


Figure 1. Selecting a Data Set and Analysis

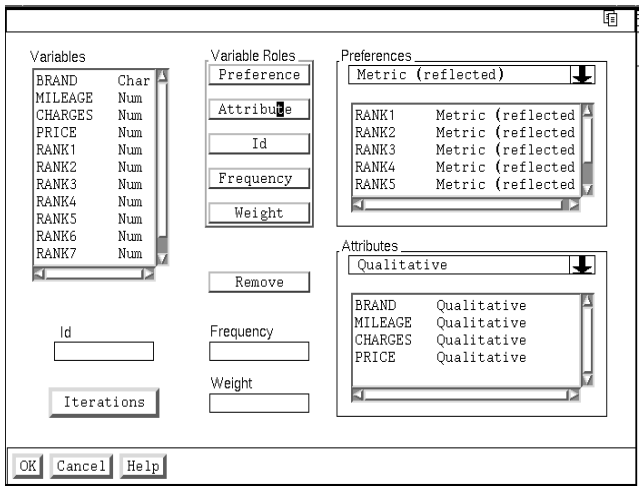


Figure 2. Conjoint Analysis Variable Selection

```
Brand:             Michelin, Goodyear, Firestone
Tread Mileage:     40,000, 60,000, 80,000
Price:             $45.00, $60.00, $75.00
Installation Cost: $0.00, $7.50
```

Seven respondents are asked to rank in order of preference 18 out of the possible 54 combinations. Although rankings are used in this example, preference ratings are frequently used in conjoint analysis.

Invoking the Application. With the data in the SAS data set, SASUSER.TIRES, you can invoke the Market Research application and perform a conjoint analysis. The application is invoked by issuing the "market" command on any command line.

Selecting a Data Set and Analysis. The first window displayed requires you to select a data set and an analysis. Because your data set is SASUSER.TIRES, select SASUSER as the library in the left-hand list box and TIRES as the data set in the right-hand list box. Then, select an analysis by clicking on the down arrow to the right of the analysis name field below the list boxes and select "Conjoint Analysis" from the displayed popup menu. See Figure 1.

View the data by pressing the View Data button and then selecting "Data values." The other selection under the View Data button,"Variable attributes," displays information about each variable.

Selecting Variables. To proceed with the analysis once you have selected a data set and an analysis, press the OK button at the bottom of the window.

The analysis requires preference and attribute variables. The preference variables are the ranks from the seven respondents and the attribute variables are the four factors. See Figure 2.

You can choose to perform a metric or a non-metric conjoint analysis; the metric analysis uses the ranks as they are, while the non-metric analysis performs a monotone transformation on the ranks. To set the measurement type for the preferences, click on the down arrow in the Preferences box at the top right of the window. Select "Metric (reflected)." "Reflected" is used because the lowest rank value, 1, corresponds to the most preferred offering. If the highest preference value corresponded to the most

preferred offering, the "Metric" selection should be used instead.

To select preference variables, select RANK1, RANK2, ... RANK7 in the Variables list box on the left side of the window, and press the Preference button in the Variable Roles box.

Likewise, you must select a measurement type for the attribute variables you want to use. The default measurement type for attributes is Qualitative, which treats the variable as a set of dummy variables with the coefficients of the dummy variables summing to 0. In this way, the utility coefficients † of each attribute sum to 0.

Use this measurement type for all four attribute variables, BRAND, MILEAGE, CHARGES, and PRICE. After selecting these four variables in the Variables list box, press the Attribute button in the Variable Roles box. Alternatively, you could use the "Continuous" measurement type for MILEAGE, CHARGES, or PRICE because these attributes are quantitative in nature.

To delete one or more of the Preference or Attribute variables, either double-click on each one in the appropriate right-hand list box or select them in any of the three list boxes and press the Remove button.

To obtain help about the window, press the Help button at the bottom of the window or click on any of the border titles on the window, for example, "Variables," "Variable Roles," "Preferences."

Once the variables have been selected, press the OK button at the bottom of the window to perform the analysis. To change the analysis, return to the Variable Selection window by pressing the Variables button on the analysis main window.

Results. The first result is a plot of the relative importance of each attribute. Relative importance is a measure of importance of the contribution of each attribute to overall preference; it is calculated by dividing the range of utilities for each attribute by the sum of all ranges and multiplying by 100.

In the example, Tire Mileage is the most important attribute with an average relative importance of 49%. The box-and-whisker plot displays the first and third quartiles as the ends of the box, the maximum and minimum as the whiskers (if they fall outside the box), and the median as a vertical bar in the interior of each box. See Figure 3.

To display a selection of additional results, press the Results button on the window. The first selection, the Utilities Table window, displays the utility coefficients for each level of an attribute for all preferences (the dependent variables). The relative importance of each attribute is displayed separately for each preference variable. This table illustrates that BRAND is the most important attribute for RANK1, the first respondent, and Michelin is the most preferred brand, because it has the highest utility coefficient value. Thus, the first respondent preferred a 80,000 mile, $45 Michelin with no installation charge.

After closing this window, you can view these results in graphical form by pressing the Results button again and selecting "Utilities plots." The plot of the Brand utilities indicates that one respondent clearly prefers Michelin while the other respondents only mildly prefer one brand over another.

To change the plot from the BRAND to the MILEAGE attribute, select MILEAGE in the list box at the right. All but one person prefer longer over shorter mileage tires, and that one prefers the 60,000 mile tire. You can examine plots for the PRICE and CHARGES attributes in the same way.

---

†Utility coefficients are estimates of the value or worth to a subject of each level of an attribute. The most preferred combination of attributes for a subject is the one with the attribute levels having the highest utility coefficient values for each attribute.

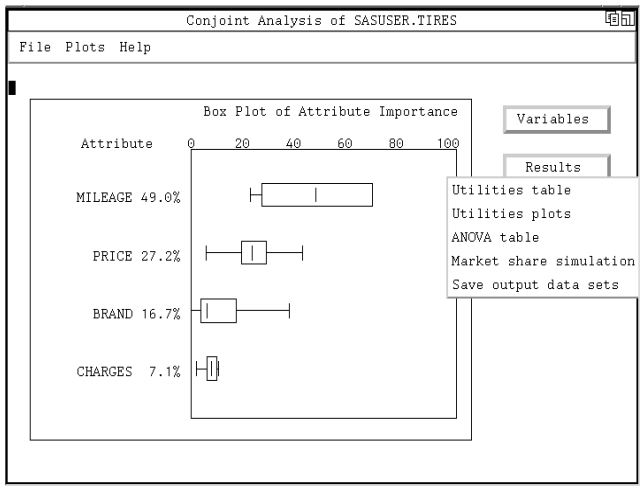


Figure 3. Plot of Relative Importance of Attributes

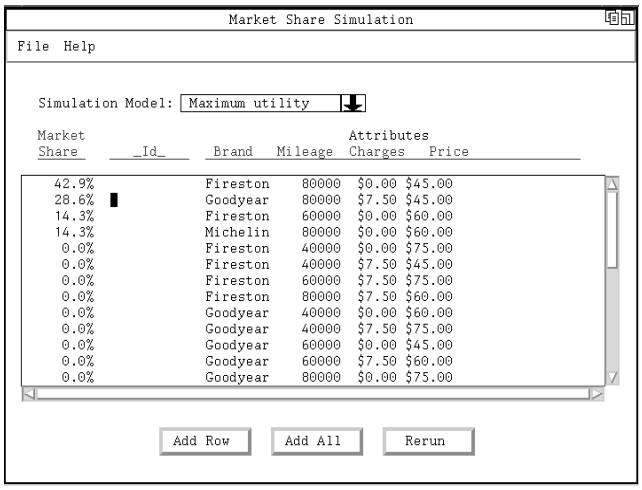


Figure 4. Estimating Market Share

Estimating Market Share. You also can calculate the expected market share for each tire purchase alternative in the sample. To do so, press the Results button and select "Market Share Simulation." The entry in the table with the largest market share is the 80,000 mile, $45 Firestone with no installation charge. It is expected to account for 42.9% of the market. The maximum utility simulation model, the default, was used to calculate the market share. You can choose from two other models: the logit model and the Bradley-Terry-Luce model. Click on the down arrow at the top of the window and select the desired model from the displayed list. See Figure 4.

Only 18 of the 54 possible tire purchase combinations were presented to the respondents. You may want to predict the expected market share of one or more of the combinations that were not present in the sample. To do so, press the Add Row button at the bottom of the window and fill in the observation in the top row of the table. Click on "-Select-" in each attribute column and select the desired level. If the observation that you create is a duplicate, a warning message is displayed. You can modify the contents of the Id column to contain a description of your own choice. After you have added some combinations, you can produce the expected market shares by pressing the Rerun button.

As an example an 80,000 mile, $45 Michelin with no installation charges would be expected to have a 64.3% market share if it was the only combination added to the original sample. Adding combinations may change the estimated market share of the other combinations.

Discrete Choice Analysis

Conjoint analysis is used to examine the preferences of consumers. The rationale for the use of preferences is that they indicate what people will choose to buy. Often in market research, the choices that consumers actually make are the behavior of interest. In these instances, it is appropriate to analyze choices directly using discrete choice analysis.

In discrete choice analysis, the respondent is presented with several choices and selects one of them. As in conjoint analysis, the factors that define the choice possibilities are called attributes. Here, they are called choice attributes to distinguish them from other factors, like demographic variables, that may be of interest but do not contribute to the definition of the choices. Each set of possible choices is called a choice set.

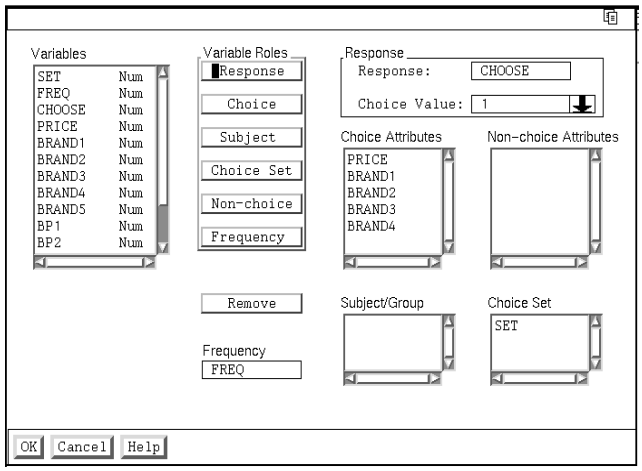


Figure 5. Discrete Choice Analysis Variable Selection

This example has choice possibilities defined by two attributes, price and brand. Five choice alternatives are presented at a time to a respondent, from which one alternative is chosen. Eight of these choice sets are presented, each one with a different set of five combinations of price and brand.

To change to a different data set or analysis, select "File → New dataset/analysis" on the main analysis window. Each time you change the data set or analysis or exit the application, you are asked if you want save the changes that you have made during the session. On the data set selection window, select the PRICE data set in the SASUSER library and then select "Discrete choice analysis." To continue, press the OK button.

With the other analyses in the application, you would be taken directly to the appropriate variable selection window. With discrete choice analysis, a supplementary window is displayed to help you determine if your data are in the appropriate form.

With discrete choice analysis, the structure of the data is important and must be in one of several layouts. After specifying if your data are contained in one or two data sets and whether a frequency variable is used, you can view the appropriate layout by pressing the Examine button. The most important requirement of the data layout is that all choice alternatives must be included, whether chosen or not.

If your data are not in the proper form, they must be rearranged before proceeding with the analysis. If your data are in the proper form, continue with the analysis by pressing the OK button. If not, press the Cancel button.

On the Variable Selection window that appears next, you must select several required variables: a response variable, some choice attribute variables, and a subject variable. Optionally, you can also choose a frequency variable and some non-choice attribute variables. If you select a frequency variable, a subject variable is not necessary.

For this example, select CHOOSE as the response variable. You also must indicate which value of the variable represents a choice. Click on the down arrow to the right of "Choice Value:" and select 1 from the list. In this example the value 1 indicates the chosen alternative and the value 0 indicates the non-chosen alternatives. See Figure 5.

Next, select PRICE and BRAND1, BRAND2, ..., BRAND4 as Choice attributes. BRAND is a nominal variable with five levels. It can be represented as four dummy-coded variables. ‡

Select FREQ as the frequency variable. The frequency variable contains the count of the number of times that a choice alternative was selected.

Because the data include more than one choice set, a Choice Set variable is needed; the choice set variable in this example is SET. After selecting the appropriate variables, press the OK button to perform the analysis.

On the analysis main window, a bar chart is displayed of the significances of each of the choice and non-choice attributes. The chart illustrates that PRICE, BRAND1, BRAND2, and BRAND4 are significant.

You can view other results by pressing the Results button and selecting "Statistics," "Choice probabilities," or "Residual plots" from the ensuing menu. Overall model fit statistics and parameter estimates for the attributes are available from the Statistics window. Probabilities for each choice alternative are available from the Choice Probabilities window. Plots of residual and predicted values are available from the Residual Plots window.

Correspondence Analysis

Categorical data are frequently encountered in the field of market research. Correspondence analysis is a technique that graphically displays relationships among the rows and columns in a contingency table. In the resulting plot there is a point for each row and each column of the table. Rows with similar patterns of counts have points that are close together, and columns with similar patterns of counts have points that are close together.

The CARS data set in the SASUSER library is used as an example (also described in the *SAS/STAT User's Guide*). The CARS data are a sample of individuals who were asked to provide information about themselves and their cars. The pertinent questions for the example are country of origin of their car and their family status.

Simple Correspondence Analysis. Simple correspondence analysis analyzes a contingency table made up of one or more column variables and one or more row variables. To select a data set on which to perform a correspondence analysis, select "File → New dataset/analysis" on the main analysis window. First, select the CARS data set, then select "Correspondence analysis" as the analysis, and then press the OK button.

This example uses raw variables instead of an existing table. The desired type of analysis (simple correspondence analysis) and data layout (raw variables) are default selections on the Variable Selection window. Select ORIGIN, the country of origin of the car, as the column variable and MARITAL, family status, as the row variable to create the desired contingency table. See Figure 6.

‡Each dummy-coded variable has the value of 1 for a different level of the attribute. In this way, each dummy-coded variable represents the presence of that level and the absence of the other levels.

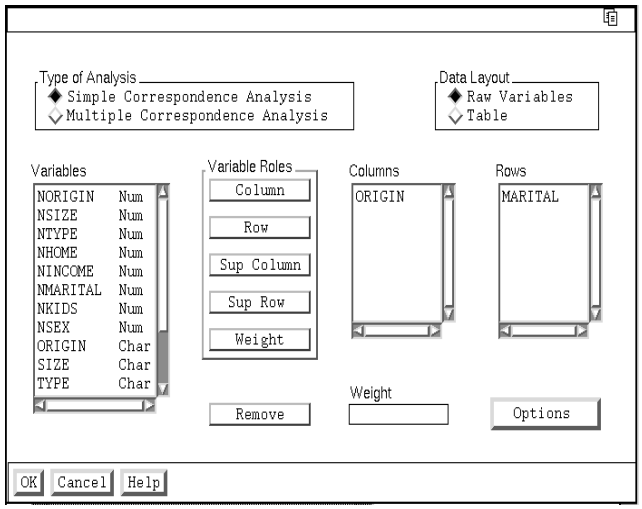


Figure 6. Simple Correspondence Analysis Variable Selection

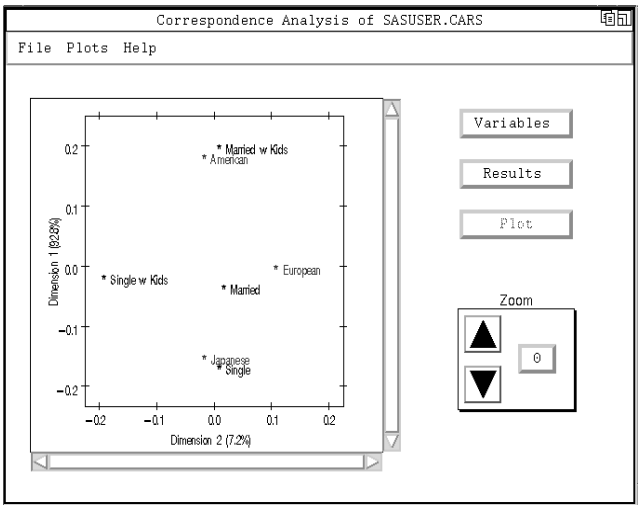


Figure 7. Correspondence Analysis Plot

Plot. The plot displays the column points and row points. The first example in the *SAS/STAT User's Guide* provides an interpretation of the plot. The interpretation has two aspects: what each dimension represents and what the relationship of the points in the dimensional space represents. An interpretation of the vertical dimension is that it represents the country of origin of the cars, with most of the influence coming from whether the car is American or Japanese. The horizontal dimension appears to represent "Single with kids" versus all of the other values. See Figure 7.

Although the row and column points are spread throughout the plot, "married" and "single" appear to be slightly more similar to each other than any of the other points. Keep in mind that distances between row and column points cannot be compared, only distances among row points and distances among column points. However, by treating the country-of-origin points as lines drawn from the 0,0 point and extending off the graph, you can see that the "Married with kids" point is closest to the American car line and the "Single" point is closest to the Japanese car line.

Plot Controls. To enlarge the plot, click on the up arrow in the zoom control box. To return the plot to its zero zoom state, click on the [0] button. If the plot is zoomed, you can move the plot left and right and up and down using the scroll bars.

Results. You can view other results by pressing the Results button and selecting "Inertia table," "Statistics," or "Frequencies." The Inertia Table window lists the singular values and inertias for all possible dimensions in the analysis. The Statistics window displays tables of statistics that aid in the interpretations of the dimensions and the points: the row and column coordinates, the partial contributions to inertia, and the squared cosines. The Frequency Table window displays observed, expected, and deviation contingency tables and row and column profiles.

Multiple Correspondence Analysis. In a multiple correspondence analysis, only column variables are used. They are used to create a Burt table [§] which is then used in the analysis.

The same data set can be used to illustrate multiple correspondence analysis. Return to the Variables Selection window by pressing the Variables button on the main analysis window. See Figure 8. Perform the following steps:

1. Remove the current column and row variables either by double-clicking on them or by selecting them and pressing the Remove button.
2. Select "Multiple Correspondence Analysis" in the Type of Analysis box in the upper left of the window.
3. Select the column variables ORIGIN, TYPE, SIZE, HOME, SEX, INCOME, and MARITAL by clicking on the ORIGIN variable and dragging through the list to the MARITAL variable, then press the Column button.
4. Press the OK button to perform the analysis.

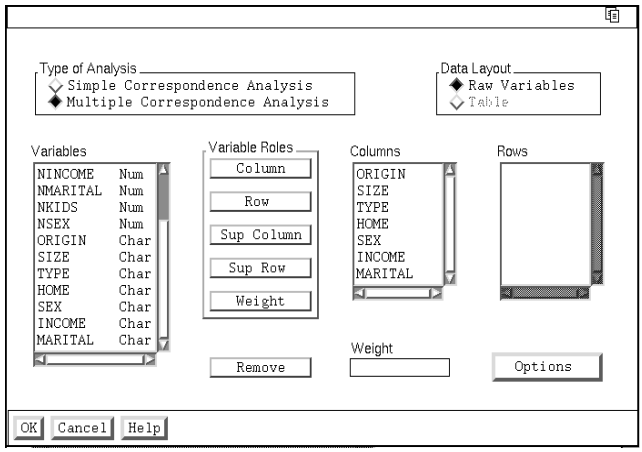


Figure 8. Multiple Correspondence Analysis Variable Selection

The distances between points of different variables can be interpreted in multiple correspondence analysis because they are all column points. However, the multiple correspondence analysis example has more dimensions (12) to interpret and examine than the single correspondence analysis example (2). The total number of dimensions can be examined in the inertia table, which is accessed from the Results button.

By default, a two-dimensional solution is computed. To request a higher dimensional solution, open the Variable Selection window, press the Options button, and select (or enter) the desired number of dimensions.

If you request a three-dimensional (or higher) solution, you can plot the dimensions two at a time by pressing the Plot button and selecting dimensions for the x axis and the y axis.

[§] A Burt table is a partitioned symmetric matrix containing all pairs of crosstabulations among a set of categorical variables. For further explanation, see the *SAS/STAT User's Guide*

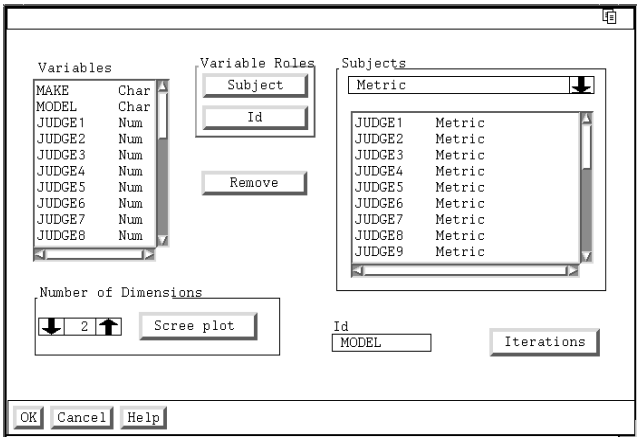


Figure 9. MDPREF Analysis Variable Selection

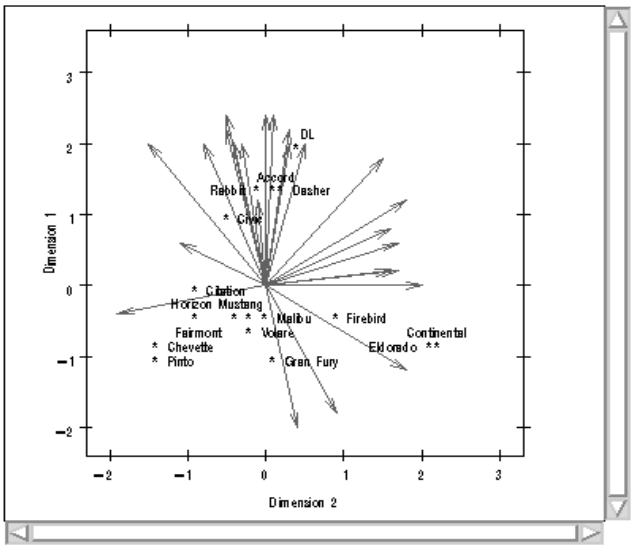


Figure 10. MDPREF Plot

Multidimensional Preference Analysis

With conjoint analysis, respondents indicate their preferences for products that are composed of attributes determined by the experimenter. Sometimes, the data of interest may be preferences of existing products for which relevant attributes are not defined for the respondent. Multidimensional preference analysis (MDPREF) is used to analyze such data.

MDPREF is a principal component analysis of a data matrix whose columns correspond to people and whose rows correspond to objects, the transpose of the usual people by objects multivariate data matrix.

The CARPREF data set in the SASUSER library is used as an example (also described in the *SAS/STAT User's Guide*. It contains data about the preferences of 25 respondents for 17 cars. The preferences are on a scale of 0 to 9 with 0 meaning a very weak preference and 9 meaning a very strong preference. Select the data set and analysis as described in the preceding examples.

As in conjoint analysis, you can choose to perform a metric or non-metric analysis. Choose the measurement type by clicking the arrow in the upper right corner of the window and selecting the desired type. Other, less frequently used, types are available under the "Other" selection. The measurement type is used for all subsequently selected Subject variables. Infrequently, subject variables with different types may be used.

For the example, use the Metric measurement type. Select the preference ratings of each respondent, JUDGE1, JUDGE2, ..., JUDGE25, as Subject variables. Also, select MODEL as the Id variable. See Figure 9.

You also can set the number of dimensions for the analysis; the default is two. A scree plot of the eigenvalues is useful in determining an appropriate number of dimensions. To display the scree plot, press the Scree Plot button. The plot illustrates that the magnitude of the eigenvalues falls off for the first two dimensions; then the plot flattens out for the third and remaining dimensions. From this graph, two dimensions appear appropriate. After closing the Scree Plot window, press the OK button to perform the analysis. See Figure 10.

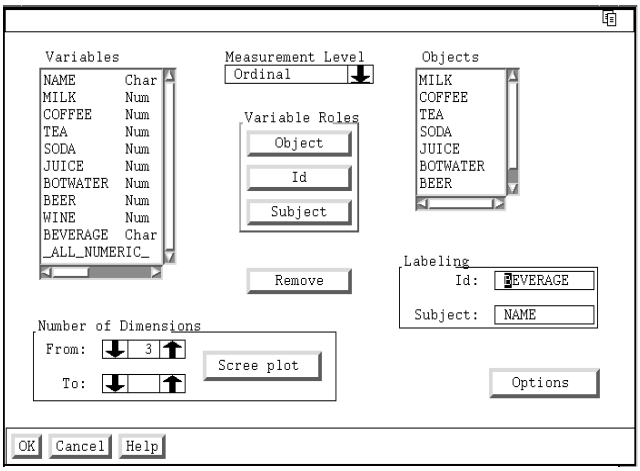


Figure 11. MDS Variable Selection

Results. The plot on the main analysis window contains points for the 17 car models and vectors for the 25 respondents. Interpretations of the two dimensions are 1) the vertical dimension separates foreign and domestic cars in the upper half and lower half, respectively, and 2) the horizontal dimension separates small cars and big cars in the left and right halves, respectively. Respondents prefer cars whose points are closest to their vector. Notice that there are a number of vectors in the upper right quadrant of the plot but there are no cars. This lack of available products to satisfy peoples' preferences indicates a possible niche to fill.

Other results are the "Initial Eigenvalue Plot," "Final Eigenvalue Plot," and "Configuration Table." The Initial Eigenvalue plot is the same as the scree plot on the Variable Selection window. The Final Eigenvalue plot is also a scree plot; it differs from the initial plot only if a measurement type other than Metric is used. The Configuration Table contains the coordinates for the car points.

Multidimensional Scaling

Multidimensional Scaling (MDS) takes subjects' judgments of either similarity or difference of pairs of items and produces a map of the perceived relationship among items.

For example, suppose you ask seven subjects to state their perceived similarity on a 1 to 7 scale for pairs of beverages, with 1 meaning very similar and 7 meaning very different. The beverages are milk, coffee, tea, soda, juice, bottled water, beer, and wine. Someone may state that their perceived similarity between coffee and tea is 3, somewhat similar, or 7, very different. There are 28 possible pairs of these eight beverages.

The data are ordered in an eight observation by eight variable matrix with one matrix (eight observations) for each subject. On the Data Set Selection window, select the BEVERAGE data set in the SASUSER library, then press the OK button. A message window informs you that MDS requires either similarity or distance data. Press the Continue button.

On the Variables Selection window, select the variables MILK, COFFEE, TEA, SODA, JUICE, BOTWATER, BEER, AND WINE as the objects. See Figure 11. It is crucial that the order of the objects is the same as their order in the rows of each matrix. In other words, from the above order, the upper left corner element in the matrix is MILK, MILK (which has a distance of zero) and the element to its

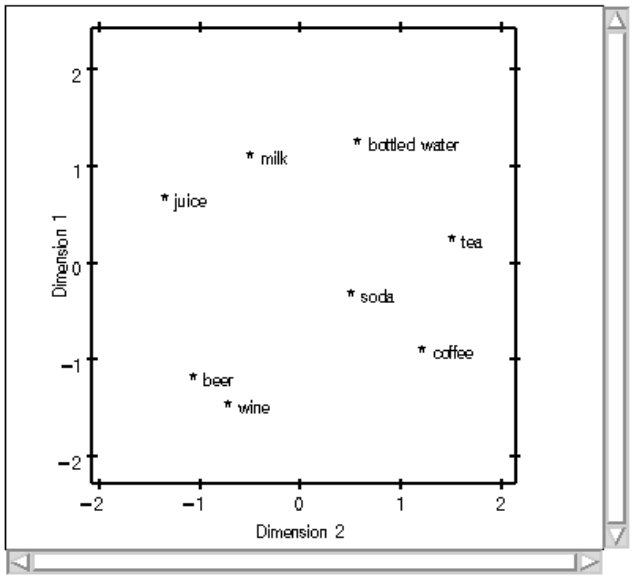


Figure 12. MDS Coordinates Plot

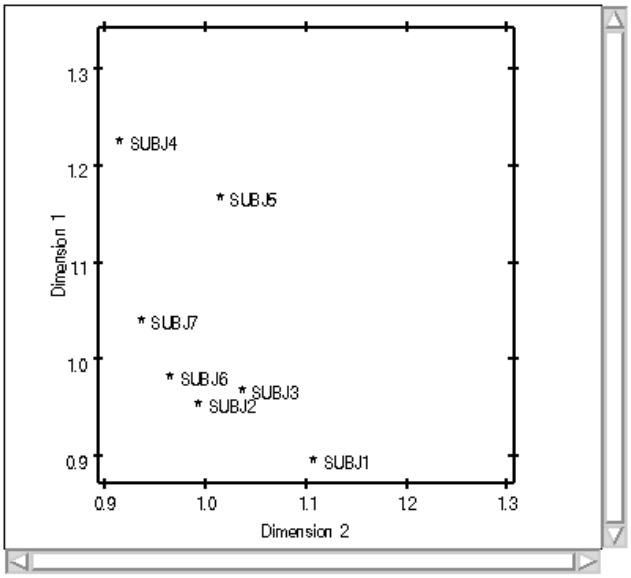


Figure 13. MDS Individual Coefficients Plot

right is MILK, COFFEE.

Also, select BEVERAGE, the beverage names, as the ID variable and NAME, the subject identifiers, as the SUBJECT variable. Because the objects are ordinally-scaled, the ordinal measurement level, the default, is appropriate for this example.

If you think that your subjects may use different perceptual schemes for judging similarity, you can choose to perform an individual differences analysis. Press the Options button and select "Individual Differences Analysis." The data are distances, the default, because larger numbers represent more difference (less similarity). If the data were similarities, you would choose the appropriate selection on the Options window. To close the options window, press the OK button.

As in correspondence analysis and MDPREF analysis, you can set the number of dimensions for the solution. With MDS you have an extra capability; you can solve for several dimensional solutions in one analysis.

Choose a three-dimensional solution by entering a "3" in the input field to the right of the "From:" label or by clicking on the up arrow to its right until the number 3 appears in the input field. As with the other dimensional analyses, a scree plot may be useful in determining the appropriate number of dimensions. You can create the plot by pressing the Scree Plot button.

To continue with the analysis, press the OK button on the Variable Selection window.

Results. As with the correspondence analysis and MDPREF plots, interpreting the MDS plot has two parts: 1) finding a reasonable interpretation for each of the plot dimensions, and 2) finding a reasonable interpretation of the relationship of the points in the plot. See Figure 12.

The presence of bottled water, milk, and juice at the top of the plot and wine, beer, and coffee at the bottom of the plot might indicate a good for you/not so good for you interpretation for the vertical dimension, Dimension 1. The horizontal dimension, Dimension 2, does not have as clear an interpretation. Try to come up with your own interpretation that would have tea, coffee, and water on one side and juice, beer, and wine on the other.

Because you requested a three-dimensional solution, two other plots can be displayed: Dimensions 1 and 3 and Dimensions 2 and 3. To change which dimensions are plotted, press the Plot button and select the desired dimensions. Also, on this window you can choose to display the coefficients of the individual differences analysis instead of the coordinates. To do so, select "Coefficient" at the bottom of the window and press the OK button.

In an individual differences analysis, there is a common perceptual map for all subjects, but different subjects have different weights for each dimension. See Figure 13. SUBJ4 is found to be highest on the vertical axis and lowest on the horizontal axis. In other words, SUBJ4 weights whatever this dimension represents more than do the other subjects and it weights whatever dimension 2 represents less than the other subjects. If the good-for-you interpretation is appropriate for Dimension 1, then it plays a larger role in SUBJ4's perceptual mapping of these beverages than it does for other subjects.

It is possible that SUBJ1, SUBJ2, SUBJ3, SUBJ6, and SUBJ7 may cluster together and SUBJ4 and SUBJ5 may be outliers. Additional subjects may sharpen this possible clustering or eliminate it. MDS is useful in market research for discovering possible perceptual perspectives used by consumers and for revealing possible market segments.

You can display other results by pressing the Results button. These results include Fit statistics, Configuration tables, Residual plots, and the Iteration history. The fit statistics are measures of how well the data fit the model. The Configuration tables contain the coordinates and, optionally, the individual difference coefficients that are used in the plots. The Residual plots allow you to assess the fit of the model graphically. The iteration history contains information about how many iterations were needed and how the criterion changed over the iterations.

Summary

Investigators in the field of market research are interested in how consumers make decisions when they choose to buy products. What attributes are important? Do all people make decisions in the same way? If not, how do they differ? What are the perceptual schemes that people use in their purchasing decisions?

The analyses described in this paper can be used with many different types of data to investigate these questions. The Market Research application makes these analyses easy to use, and it is available in Release 6.11 and subsequent releases with the SAS/STAT product.

Acknowledgements

I would like to thank Greg Goodwin, Warren Kuhfeld, Julie LaBarr, Donna Sawyer, and Maura Stokes for their thoughtful comments on this paper.

Experimental Design, Efficiency, Coding, and Choice Designs

Warren F. Kuhfeld

Abstract

This chapter discusses some of the fundamental concepts in marketing research experimental design including standard factorial designs, orthogonal arrays, nonorthogonal designs, and choice and conjoint designs. Design terminology is introduced, design efficiency is explained, and the process of going from an efficient linear design to a choice design is explained. You should be familiar with the concepts in this chapter before studying the conjoint or discrete choice chapters. After you are comfortable with the material in this chapter, it would be good to also look at the design chapters starting on pages 99 and 121 as well.*

Introduction

Experimental designs are fundamental components of marketing research, conjoint analysis, and choice modeling. An *experimental design* is a plan for running an experiment. The *factors* of an experimental design are the columns or variables that have two or more fixed values, or *levels*. The rows of a design are sometimes called *runs* and correspond to product profiles. Experiments are performed to study the effects of the factor levels on the dependent or response variable. The factors are the attributes of the hypothetical products or services. In a discrete-choice study, the rows of the design correspond to product alternatives, and blocks of several rows comprise a set of products and are called *choice sets*. The dependent variable or response is choice. In a conjoint study, the rows of the design correspond to products, and the dependent variable or response is a rating or a ranking of the products. See page 483 for an introduction to conjoint analysis and page 144 for an introduction to choice models. The next two sections show simple examples of conjoint and choice experiments.

*Copies of this chapter (TS-722C), the other chapters, and all of the macros are available on the web http://support.sas.com/techsup/tnote/tnote_stat.html#market. This chapter is based on the tutorial that Don Anderson and I have given for many years at the American Marketing Association's Advanced Research Techniques Forum. Be forewarned that this chapter still contains some of the occasional silliness that is in the tutorial.

The Basic Conjoint Experiment

A conjoint study uses experimental design to create a list of products, and subjects rate or rank the products. Here is a conjoint design and the layout of a simple conjoint experiment with two factors. In a real experiment, the product descriptions would be more involved and may use art or pictures, but the basic experiment involves people seeing products and rating or ranking them. The brand factor has three levels, Acme, Ajax, and Widget, and the price factor has two levels, $1.99 and $2.99. There are a total of six products.

Conjoint Design

Acme	$1.99
Acme	$2.99
Ajax	$1.99
Ajax	$2.99
Widget	$1.99
Widget	$2.99

Full-Profile Conjoint Experiment

		Rate Your Purchase Interest
Acme	$1.99	
Acme	$2.99	
Ajax	$1.99	
Ajax	$2.99	
Widget	$1.99	
Widget	$2.99	

The Basic Choice Experiment

A discrete choice study uses experimental design to create sets of products, and subjects choose a product from each set. Here is a choice design and the layout of a simple choice experiment. In a real experiment, the product descriptions would be more involved and they may use art or pictures, but the basic experiment involves people seeing sets of products and making choices. This example has four choice sets, each composed of three alternative products, so subjects would make four choices. Each alternative is composed of two factors: brand has three levels, and price has two levels.

Choice Design

1	Acme	$2.99
	Ajax	$1.99
	Widget	$1.99
2	Acme	$2.99
	Ajax	$2.99
	Widget	$2.99
3	Acme	$1.99
	Ajax	$1.99
	Widget	$2.99
4	Acme	$1.99
	Ajax	$2.99
	Widget	$1.99

Discrete Choice Experiment

1	2	3	Choice
Acme $2.99	Ajax $1.99	Widget $1.99	
Acme $2.99	Ajax $2.99	Widget $2.99	
Acme $1.99	Ajax $1.99	Widget $2.99	
Acme $1.99	Ajax $2.99	Widget $1.99	

Experimental Design Terminology

Here again is the conjoint design but presented in three forms. The following tables contain a "raw" experimental design with two factors, the same design with factor names and levels assigned, and a randomized version of the raw design.

Full-Factorial Design	
x1	x2
1	1
1	2
2	1
2	2
3	1
3	2

Full-Profile Conjoint Design	
Brand	Price
Acme	1.99
Acme	2.99
Ajax	1.99
Ajax	2.99
Widget	1.99
Widget	2.99

Randomized Design	
x1	x2
2	2
1	1
1	2
3	1
3	2
2	1

This is an example of a *full-factorial design*. It consists of all possible combinations of the levels of the factors. Full-factorial designs allow you to estimate main effects and interactions. A *main effect* is a simple effect, such as a price or brand effect. In a main effects model, for example, the brand effect is the same at the different prices and the price effect is the same for the different brands. *Interactions* involve two or more factors, such as a brand by price interaction. In a model with interactions, for example, brand preference is different at the different prices and the price effect is different for the different brands. In Figure 1, there is a main effect for price, and utility increases by one when price goes from $2.99 to $1.99 for all brands. Similarly, the change in utility from Acme to Ajax to Widget does not depend on price. In contrast, there are interactions in Figure 2, so the price effect is different depending on brand, and the brand effect is different depending on price.

Before an experimental design is used, it should be *randomized*. This involves sorting the rows into a random order and randomly reassigning all of the factor levels. It is not unusual, for example, for the

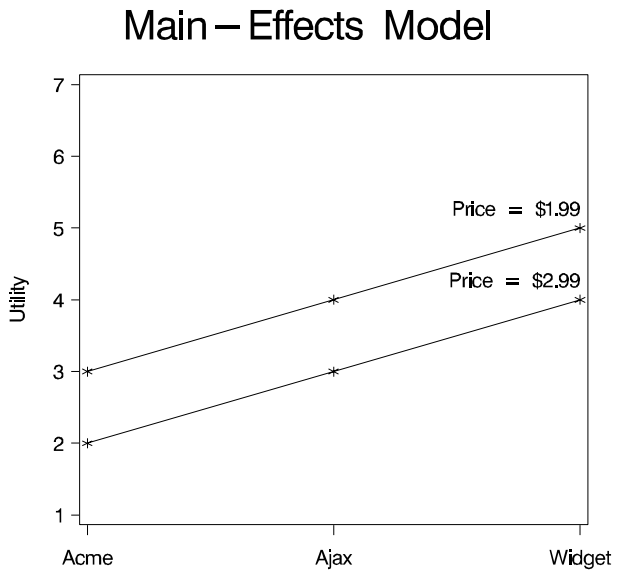


Figure 1

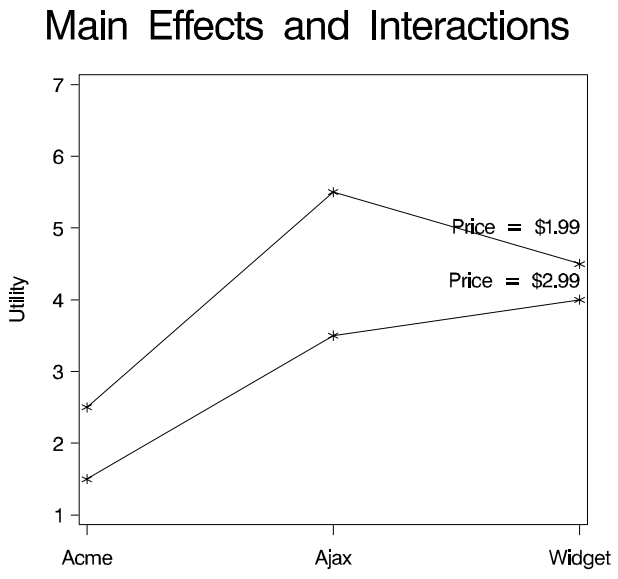


Figure 2

first row of a design to contain all ones, the first level. Randomization takes care of this by mixing things up. It also eliminates showing all of one brand, then all of the next, and so on. Randomization for example will change levels (1 2 3) to one of the following: (1 2 3), (1 3 2), (2 1 3), (2 3 1), (3 1 2), (3 2 1).

In a full-factorial design, all main effects, all two-way interactions, and all higher-order interactions are estimable and uncorrelated. The problem with a full-factorial design is that, for most practical situations, it is too cost-prohibitive and tedious to have subjects consider all possible combinations. For example, with five factors, two at four levels and three at five levels (denoted $4^2 5^3$), there are $4 \times 4 \times 5 \times 5 \times 5 = 2000$ combinations in the full-factorial design. For this reason, researchers often use *fractional-factorial designs*, which have fewer runs than full-factorial designs. The price of having fewer runs is that some effects become confounded. Two effects are *confounded* or *aliased* when they are not distinguishable from each other. This means that lower-order effects such as main effects or two-way interactions may be aliased with higher order interactions in most of our designs. We estimate lower-order effects by assuming that higher-order effects are zero or negligible. See page 306 for an example of aliasing.

Fractional-factorial designs that are both orthogonal and balanced are of particular interest. A design is *balanced* when each level occurs equally often within each factor, which means that the intercept is orthogonal to each effect. When every *pair* of levels occurs equally often across all pairs of factor, the design is said to be *orthogonal*. Another way in which a design can be orthogonal is when the frequencies for level pairs are proportional instead of equal. For example, with 2 two-level factors, an orthogonal design could have pair-wise frequencies proportional to 2, 4, 4, 8. Such a design will not be balanced−one level will occur twice as often as the other. Imbalance is a generalized form of nonorthogonality, hence it increases the variances of the parameter estimates and decreases efficiency.

Fractional-factorial designs are categorized by their *resolution*. The resolution identifies which effects, possibly including interactions, are estimable. For example, for resolution III designs, all main effects are estimable free of each other, but some of them are confounded with two-factor interactions. For resolution IV designs, all main effects are estimable free of each other and free of all two-factor interactions, but some two-factor interactions are confounded with other two-factor interactions. For resolution V designs, all main effects and two-factor interactions are estimable free of each other. More generally, if resolution (r) is odd, then effects of order $e = (r-1)/2$ or less are estimable free of each other. However, at least some of the effects of order e are confounded with interactions of order $e+1$. If r is even, then effects of order $e = (r-2)/2$ are estimable free of each other and are also free of interactions of order $e+1$. Higher resolutions require larger designs. Resolution III fractional-factorial designs are frequently used in marketing research.

A special type of factorial design is the *orthogonal array*. An orthogonal array or orthogonal design is one in which all estimable effects are uncorrelated. Orthogonal arrays come in specific numbers of runs for specific numbers of factors with specific numbers of levels. Here is a list of all orthogonal arrays up to 28 runs. The list shows the number of runs followed by the design, in notation: levels raised to the number-of-factors power.

4	2^3	12	2^{11}	16	2^{15}	18	$2^1 3^7$	21	$3^1 7^1$	24	2^{23}	25	5^6
6	$2^1 3^1$		$2^4 3^1$		$2^{12} 4^1$		$2^1 9^1$	22	$2^1 11^1$		$2^{20} 4^1$	26	$2^1 13^1$
8	2^7		$2^2 6^1$		$2^9 4^2$		$3^6 6^1$				$2^{16} 3^1$	27	3^{13}
	$2^4 4^1$		$3^1 4^1$		$2^8 8^1$	20	2^{19}				$2^{14} 6^1$		$3^9 9^1$
9	3^4	14	$2^1 7^1$		$2^6 4^3$		$2^8 5^1$				$2^{13} 3^1 4^1$	28	2^{27}
10	$2^1 5^1$	15	$3^1 5^1$		$2^3 4^4$		$2^2 10^1$				$2^{12} 12^1$		$2^{12} 7^1$
					4^5		$4^1 5^1$				$2^{11} 4^1 6^1$		$2^2 14^1$
											$3^1 8^1$		$4^1 7^1$

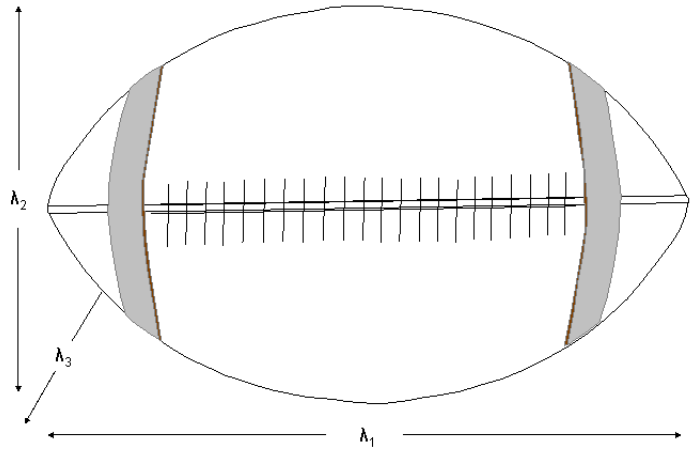
An orthogonal array is both balanced and orthogonal, and hence 100% efficient and optimal. Efficiency, which is explained starting on page 53, is a measure of the goodness of the experimental design. The term "orthogonal array," as it is sometimes used in practice, is imprecise. It is correctly used to refer to designs that are both orthogonal and balanced, and hence optimal. However, the term is sometimes also used to refer to designs that are orthogonal but not balanced, and hence not 100% efficient and sometimes not even optimal. Orthogonal designs are often practical for main-effects models when the number of factors is small and the number of levels of each factor is small. However, there are some situations in which orthogonal designs are not practical, such as when

- not all combinations of factor levels are feasible or make sense
- the desired number of runs is not available in an orthogonal design
- a nonstandard model is being used, such as a model with interactions, polynomials, or splines.

When an orthogonal and balanced design is not practical, you must make a choice. One choice is to change the factors and levels to fit some known orthogonal design. This choice is undesirable for obvious reasons. When a suitable orthogonal and balanced design does not exist, efficient nonorthogonal designs can be used instead. Nonorthogonal designs, where some coefficients may be slightly correlated, can be used in all of the situations listed previously. You do not have to adapt every experiment to fit some known orthogonal array. First you choose the number of runs. You are not restricted by the sizes of orthogonal arrays, which come in specific numbers of runs for specific numbers of factors with specific numbers of levels. Then you specify the levels of each of the factors and the number of runs. Algorithms for generating efficient designs select a set of *design points* from a set of *candidate points*, such as a full factorial, that optimize an efficiency criterion. Throughout this book, we will use the `%MktEx` macro to find good, efficient experimental designs. The `%MktEx` macro is a part of the SAS autocall library. See page 597 for information on installing and using SAS autocall macros.

Eigenvalues, Means, and Footballs

The next section will discuss experimental design efficiency. To fully understand that section, you need some basic understanding of *eigenvalues*, and various types of means or averages. This section explains these and other concepts, but without a high degree of mathematical rigor. An American football provides a nice visual image for understanding the eigenvalues of a matrix.



The rows or columns of a matrix can be thought of as a swarm of points in Euclidean space. Similarly, a football consists of a set of points in a space. Sometimes, it is helpful to get an idea of the size of your group of points. For a football, you might think of three measures because a football is a three-dimensional object: the longest length from end to end, the height in the center and perpendicular to the length, and finally the width, which for a fully-inflated football is the same as the height. One can do similar things for matrices, and that's where eigenvalues come in. For many of us, eigenvalues are most familiar from factor analysis and principal component analysis. In principal component analysis, one rotates a cloud of points to a principal axes orientation, just as this football has been rotated so that its longest dimension is horizontally displayed. The principal components correspond to: the longest squared length, the second longest squared length perpendicular or orthogonal to the first, the third longest squared length orthogonal to the first two, and so on. The eigenvalues are the variances of the principal components and are proportional to squared lengths. The eigenvalues provide a set of measures of the size of a matrix, just as the lengths provide a set of measures of the size of a football.

Here is a small experimental design, the coded design $\mathbf{X}$, the sum of squares and cross products matrix $\mathbf{X}'\mathbf{X}$, the matrix inverse $(\mathbf{X}'\mathbf{X})^{-1}$, and the eigenvalues of the inverse, Λ.

Design		
1	1	1
1	2	2
1	2	2
2	1	2
2	2	1
2	2	1

$\mathbf{X}$			
1	1	1	1
1	1	-1	-1
1	1	-1	-1
1	-1	1	-1
1	-1	-1	1
1	-1	-1	1

$\mathbf{X}'\mathbf{X}$			
6	0	-2	0
0	6	0	-2
-2	0	6	0
0	-2	0	6

$(\mathbf{X}'\mathbf{X})^{-1}$			
0.188	0.000	0.063	0.000
0.000	0.188	0.000	0.063
0.063	0.000	0.188	0.000
0.000	0.063	0.000	0.188

Λ			
1/4	0	0	0
0	1/4	0	0
0	0	1/8	0
0	0	0	1/8

$\mathbf{X}$ is made from the raw design by coding, which in this case simply involves creating an intercept and appending the design, replacing 2 with -1. See page 64 for more on coding. The $\mathbf{X}'\mathbf{X}$ matrix comes from a matrix multiplication of the transpose of $\mathbf{X}$ times $\mathbf{X}$. For example the -2 in the first row comes from $\mathbf{x}_1'\mathbf{x}_3 = (1 \;\; 1 \;\; 1 \;\; 1 \;\; 1 \;\; 1)'(1 \;\; \text{-1} \;\; \text{-1} \;\; 1 \;\; \text{-1} \;\; \text{-1}) = 1\times1+1\times-1+1\times-1+1\times1+1\times-1+1\times-1 = -2$. Explaining the computations involved in finding the matrix inverse and eigenvalues is beyond the scope of this chapter, however, they are explained in many linear algebra and multivariate statistics texts.

The trace is the sum of the diagonal elements of a matrix, which for $(\mathbf{X}'\mathbf{X})^{-1}$ is both the sum of the variances and the sum of the eigenvalues: $\text{trace}\,((\mathbf{X}'\mathbf{X})^{-1}) = \text{trace}\,(\Lambda) = 0.188+0.188+0.188+0.188 = 1/4+1/4+1/8+1/8 = 0.75$. The determinant of $(\mathbf{X}'\mathbf{X})^{-1}$, denoted $|(\mathbf{X}'\mathbf{X})^{-1}|$, is the product of the eigenvalues and is 0.0009766: $|(\mathbf{X}'\mathbf{X})^{-1}| = |\Lambda| = 1/4\times1/4\times1/8\times1/8 = 0.0009766$. The determinant of a matrix is geometrically interpreted in terms of the volume of the space defined by the matrix. The formula for the determinant of a nondiagonal matrix is complicated, so determinants are more conveniently expressed as a function of the eigenvalues.

Given a set of eigenvalues, or any set of numbers, we frequently want to create a single number that summarizes the values in the set. The most obvious way to do this is to compute the average or arithmetic mean. The familiar *arithmetic mean* is found by adding together p numbers and then dividing by p. A trace, divided by p, is an arithmetic mean. The arithmetic mean is an enormously popular and useful statistic, however it is not the only way to average numbers. The less familiar *geometric mean* is found by multiplying p numbers together and then taking the *pth* root of the product. The *pth* root of a determinant is a geometric mean of eigenvalues. To better understand the geometric mean, consider an example. Say your investments increased by 7%, 5%, and 12% over a three year period. The arithmetic mean of these numbers, $(7+5+12)/3 = 8\%$, is *not* the average increase that would have come if the investments had increased by the same amount every year. To find that average, we need the geometric mean: $(1.07\times1.05\times1.12)^{1/3} = 1.0796$. The average increase is 7.96%.

Experimental Design Efficiency

This section discusses precisely what is meant by an efficient design. While this section is important, it is not critical that you understand every mathematical detail. The concepts are explained again in a more intuitive and less mathematical way in the next section. Also, refer to page 99 for more information on efficient experimental designs.

The goodness or *efficiency* of an experimental design can be quantified. Common measures of the efficiency of an $(N_D \times p)$ design matrix $\mathbf{X}$ are based on the *information matrix* $\mathbf{X}'\mathbf{X}$. The variance-covariance matrix of the vector of parameter estimates $\hat{\boldsymbol{\beta}}$ in a least-squares analysis is proportional to $(\mathbf{X}'\mathbf{X})^{-1}$. An efficient design will have a "small" variance matrix, and the eigenvalues of $(\mathbf{X}'\mathbf{X})^{-1}$ provide measures of its "size." The two most prominent efficiency measures are based on quantifying the idea of matrix size by averaging (in some sense) the eigenvalues or variances.

A-efficiency is a function of the arithmetic mean of the eigenvalues, which is also the arithmetic mean of the variances, and is given by $\text{trace}\,((\mathbf{X}'\mathbf{X})^{-1})/p$. *A*-efficiency is perhaps the most obvious measure of efficiency. As the variances get smaller and the arithmetic mean of the variances of the parameter estimates goes down, *A*-efficiency goes up. However, as we learned in the previous section, there are other averages that we might consider. *D-efficiency* is a function of the geometric mean of the eigenvalues, which is given by $|(\mathbf{X}'\mathbf{X})^{-1}|^{1/p}$. Both *D*-efficiency and *A*-efficiency are based on the idea of average variance, but in different senses of the word "average." We will usually use *D*-efficiency for two

reasons. It is the easier and faster of the two for a computer program to optimize. Furthermore, relative *D*-efficiency, the ratio of two *D*-efficiencies for two competing designs, is invariant under different coding schemes. This is not true with *A*-efficiency. A third common efficiency measure, *G-efficiency*, is based on σ_M, the maximum standard error for prediction over the candidate set. All three of these criteria are convex functions of the eigenvalues of $(\mathbf{X}'\mathbf{X})^{-1}$ and hence are usually highly correlated.

For all three criteria, if a balanced and orthogonal design exists, then it has optimum efficiency; conversely, the more efficient a design is, the more it tends toward balance and orthogonality. A design is balanced and orthogonal when $(\mathbf{X}'\mathbf{X})^{-1}$ is diagonal and equals $\frac{1}{N_D}\mathbf{I}$ for a suitably coded $\mathbf{X}$. A design is orthogonal when the submatrix of $(\mathbf{X}'\mathbf{X})^{-1}$, excluding the row and column for the intercept, is diagonal; there may be off-diagonal nonzeros for the intercept. A design is balanced when all off-diagonal elements in the intercept row and column are zero. How we choose $\mathbf{X}$ determines the efficiency of our design. Ideally, we want to choose our $\mathbf{X}$'s so that the design is balanced and orthogonal or at least very nearly so. More precisely, we want to choose $\mathbf{X}$ so that we maximize efficiency.

These measures of efficiency can be scaled to range from 0 to 100 (see pages 64− 66 for the orthogonal coding of $\mathbf{X}$ that must be used with these formulas):

$$A\text{-efficiency} = 100 \times \frac{1}{N_D \text{ trace}\left((\mathbf{X}'\mathbf{X})^{-1}\right)/p}$$

$$D\text{-efficiency} = 100 \times \frac{1}{N_D \left|(\mathbf{X}'\mathbf{X})^{-1}\right|^{1/p}}$$

$$G\text{-efficiency} = 100 \times \frac{\sqrt{p/N_D}}{\sigma_M}$$

These efficiencies measure the goodness of the design relative to hypothetical orthogonal designs that may not exist, so they are not useful as absolute measures of design efficiency. Instead, they should be used relatively, to compare one design to another for the same situation. Efficiencies that are not near 100 may be perfectly satisfactory.

Experimental Design: Rafts, Rulers, Alligators, and Stones

A good physical metaphor for understanding experimental design and design efficiency is a raft. A raft is a flat boat, often supported by flotation devices attached to the corners. The raft in Figure 3 has four Styrofoam blocks under each corner, which provide nice stability and equal support. This raft corresponds to 2 two-level factors from a 16-run design (see Table 1). The four corners correspond to each of the four possible combinations of 2 two-level factors, and the four blocks under the raft form an up-side-down bar chart showing the frequencies for each of the four combinations. Looking at the raft, one can tell that the first factor is balanced (equal support on the left and on the right) as is the second (equal support in the front and the back). The design is also orthogonal (equal support in all four corners). Making a design that supports your research conclusions is like making a raft that you are actually going to use in water full of piranha and alligators. You want good support no matter which portion of the raft you find yourself on. Similarly, you want good support for your research and good information about all of your product attributes and attribute levels.

Now compare the raft in Figure 3 to the one shown in Figure 4. The Figure 4 raft corresponds to the two-level factors in the design shown in Table 2. This design has 18 runs, and since 18 cannot be

divided by 2×2, a design that is both balanced and orthogonal is not possible. Clearly this design is not balanced in either factor. There are twelve blocks on the left and only six on the right, and there are twelve blocks on the back and only six on the front. This design *is* however orthogonal because the corner frequencies are proportional. These two factors can be made from 2 three-level factors in the L_{18} design, which has up to 7 three-level factors. See Table 3. The three-level factors are all orthogonal, and recoding levels, replacing 3 with 1, preserves that orthogonality at the cost of decreased efficiency and a horrendous lack of balance. See Table 3 on page 106 for the information and variance matrices for the Figure 4 raft.

Finally, compare the raft in Figure 4 to the one shown in Figure 5. Both of these correspond to designs with two-level factors in 18 runs. The Figure 5 raft corresponds to a design that is balanced. There are nine blocks on the left and nine on the right, and there are nine blocks on the back and nine on the front. The design is not however orthogonal since the corner frequencies are 4, 5, 4, and 5, which are not equal or even proportional. Ideally, you would like a raft like the one in Figure 3, which corresponds to a design that is both orthogonal and balanced. However, to have both two or more three-level and two or more two-level factors, you need 36 runs. In 18 runs, you can make an optimal design, like the one in Table 4 and Figure 5, that provides good support under all corners but not perfectly equal support. See Tables 3 and 4 in the next chapter on pages 107 and 106 for the information and variance matrices for the Figure 4 and 5 rafts.

Which raft would you rather walk on? The Figure 3 and Figure 5 rafts are going to be pretty stable. The Figure 3 raft is in fact optimal, given exactly 16 Styrofoam blocks, and the Figure 5 raft is also optimal, given exactly 18 Styrofoam blocks. The Figure 4 raft might be fine if you stay in the back left corner, but take one step, and you will be alligator bait.[†] Seriously though, all alligator silliness aside, if you do not provide stable and well supported research, your clients or brand managers *will* find someone else who can. In both raft and design terms, the problem is one of stability and support. In design terms, part of your results will not be stable due to a lack of information about the front right combination in your factorial design. How confident will you be in your results when you have so little information about some of your product attribute levels?

The Table 4 design (Figure 5 raft) brings to mind a cup containing exactly one half cup of water. The optimist sees the cup as half full, and the pessimist sees it as half empty. In the design, the optimist sees a little extra support in the back left and front right corners. The pessimist sees a little less support in the front left and back right corners. Either way, all available resources (design points) are optimally allocated to maximize efficiency and stability. What you would really like is both balance and orthogonality. However, you cannot get both in 18 runs, because 2×2 does not divide 18. Still, you can do pretty well. Mick Jagger and Keith Richards (1969) summed it up best in one of their

[†]The alligator knows where your weakness is, and he is watching! The alligator was in the ART Forum tutorial from the start, and I did not want to leave him out of this chapter, even if this part is a bit silly. Here is the alligator story that started all of this silliness. I live in central North Carolina. There are alligators in the southeast part of NC, but there are not supposed to be any in my part. In fact, on rare occasions, they swim up river in the summer and end up in surprising places like the time a ten-footer ended up in a golf course lake outside of Raleigh. I live not too far from Jordan lake, which contrary to popular belief was *not* named after Michael Jordan, who played his college ball nearby at UNC. Just before our first design tutorial, some fishermen were out bow fishing one night on Jordan lake. One of them shot what he thought was a really big fish. It turned out, it was a small alligator. Shooting an alligator is North Carolina is a very illegal thing to do. Nevertheless, the fishermen decided to have their trophy stuffed. They took it to a taxidermist, which was a very stupid thing to do, because a law-abiding taxidermist must turn you in to the authorities if you bring in a protected animal to be mounted. They were both arrested and slapped with a hefty fine. In their defense, they argued, "But how were we supposed to know it was an alligator? There *are no alligators* in this part of North Carolina!" The authorities agreed that this was true, but they still had to pay the fine. The moral of the story is you always have to watch for alligators, because you never know where they will turn up. The real point is, the alligator is a predator, just like your competition. Just as you would be foolish to float through the gators on an unstable raft, you would be unwise to use anything less than the most efficient design you can find to support your research conclusions.

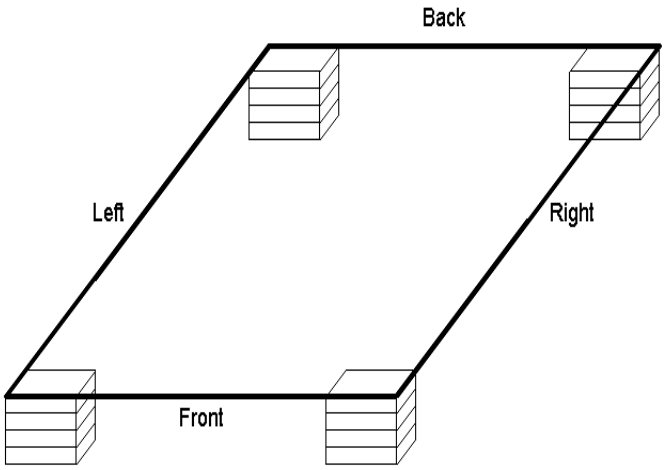


Figure 3 16 Runs, Orthogonal and Balanced

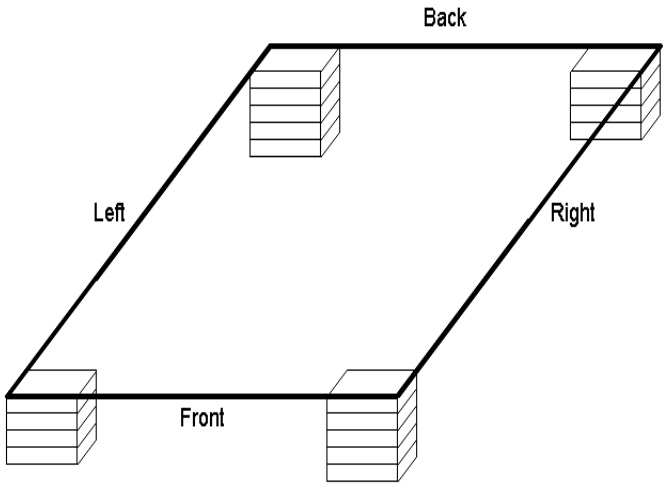


Figure 5 18 Runs, Balanced and Almost Orthogonal

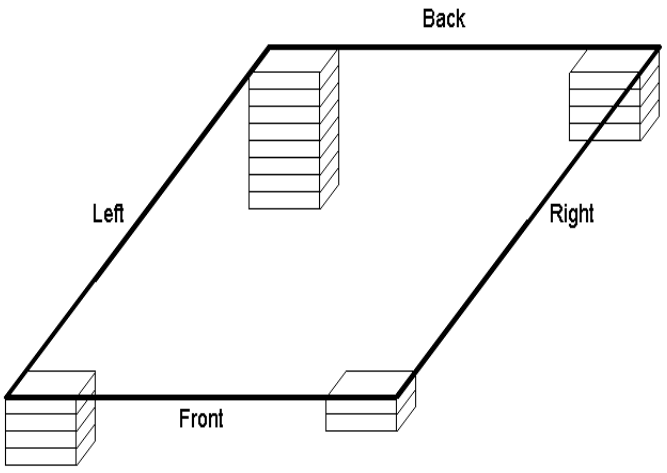


Figure 4 18 Runs, Orthogonal but not Balanced

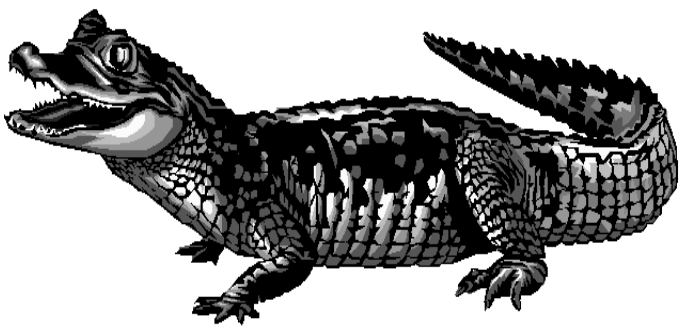


Table 1
Two-Level Factors in 16 Runs

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	2	2	2	2	1	1	2	2	1	2	2
1	1	2	2	1	1	2	2	1	2	1	2	2	1	2
1	1	2	2	2	2	1	1	1	2	2	1	2	2	1
1	2	1	2	1	2	1	2	2	2	2	2	1	1	1
1	2	1	2	2	1	2	1	2	2	1	1	1	2	2
1	2	2	1	1	2	2	1	2	1	2	1	2	1	2
1	2	2	1	2	1	1	2	2	1	1	2	2	2	1
2	1	1	2	1	2	2	1	2	1	1	2	2	2	1
2	1	1	2	2	1	1	2	2	1	2	1	2	1	2
2	1	2	1	1	2	1	2	2	2	1	1	1	2	2
2	1	2	1	2	1	2	1	2	2	2	2	1	1	1
2	2	1	1	1	1	2	2	1	2	2	1	2	2	1
2	2	1	1	2	2	1	1	1	2	1	2	2	1	2
2	2	2	2	1	1	1	1	1	1	2	2	1	2	2
2	2	2	2	2	2	2	2	1	1	1	1	1	1	1

Table 2
Unbalanced
$2^2 3^3$ in 18 Runs

1	1	1	1	1
1	1	2	3	3
1	1	1	3	2
1	1	3	2	3
1	2	2	2	1
1	2	3	1	2
1	1	1	2	3
1	1	3	3	2
1	1	2	2	2
1	1	3	1	1
1	2	1	3	1
1	2	2	1	3
2	1	2	1	2
2	1	3	2	1
2	1	1	1	3
2	1	2	3	1
2	2	1	2	2
2	2	3	3	3

Table 3
Making Twos from Threes

1		1	1		1
1		1	1		1
1		1	2		2
1		1	2		2
1		1	3		1
1		1	3		1
2		2	1		1
2		2	1		1
2	→	2	2	→	2
2		2	2		2
2		2	3		1
2		2	3		1
3		1	1		1
3		1	1		1
3		1	2		2
3		1	2		2
3		1	3		1
3		1	3		1

Table 4
Optimal
$2^2 3^3$ in 18 Runs

1	1	1	2	2
1	1	2	3	2
1	1	3	1	2
1	1	3	2	3
1	2	1	1	3
1	2	1	3	1
1	2	2	1	3
1	2	2	2	1
1	2	3	3	1
2	1	1	2	1
2	1	1	3	3
2	1	2	1	1
2	1	2	3	3
2	1	3	1	1
2	2	1	1	2
2	2	2	2	2
2	2	3	2	3
2	2	3	3	2

most influential commentaries on choice designs: "You can't always get what you want, but if you try sometimes, you just might find, you get what you need!"[‡] What you want is orthogonality and balance. What you need is good stability. Efficient designs can give you what you need.

The Table 2 and Figure 4 design may seem like just a "straw man," something that we build up just so that we can knock it down. However, this design was widely used in the past, and in fact, in spite of the fact that its deficiencies have been known for over 10 years (Kuhfeld, Tobias, and Garratt 1994) it is *still* used in some sources as a text-book example of a good design. In fact, it is a text-book example of how *not* to make designs. It is an example of what can happen when you choose orthogonality as the be-all-end-all design criterion, ignoring both balance and statistical efficiency. It is also an example of what can happen when you construct designs from a small, inferior, and incomplete catalog instead of using a comprehensive designer. Even among orthogonal designs, it is not optimal (see pages 105–107). If we can achieve perfect orthogonality *and* balance, our design will be optimal and have maximum efficiency. The key consideration is that maximizing statistical efficiency minimizes the variability of our parameter estimates, and that is what we want to achieve. Recall that for a linear model, the variance-covariance matrix of the vector of parameter estimates is proportional to $(\mathbf{X}'\mathbf{X})^{-1}$. Maximizing efficiency minimizes those variances, covariances, and hence standard errors. These designs are discussed in more detail, including an examination of their variance matrices, starting on page 105.

How we choose our design, our $\mathbf{X}$ values, affects the variability of our parameter estimates. Previously we talked about eigenvalues and the variance matrix, which provided a mathematical representation of the idea that we choose our $\mathbf{X}$ values so that our parameter estimates will have small standard errors. Now, we will discuss this less mathematically. Imagine that we are going to construct a very simple experiment. We are interested in investigating the purchase interest of a product as a function of its price. So we design an experiment with two prices, \$1.49 and \$1.50 and ask people to rate how interested they are in the products at those two prices. We plot the results with price on the horizontal axis and purchase interest on the vertical axis. We find that the price effect is minimal. See Figure 6. Now imagine that the line is a ruler and the two dots are your fingers. Your fingers are the design points providing support for your research. Your fingers are close together because in our research design, we chose two prices that are close together. Furthermore, imagine that there is a small amount of error in your data, that is error in the reported purchase interest, which is in the vertical direction. To envision this, move your fingers up and down, just a little bit. What happens to your slope and intercept as you do this?[§] They vary a lot! This is not a function of your data; it is a function of your design being inefficient because you did not adequately sample a reasonable price range.

Next, let's design a similar experiment, but this time with prices of \$0.99 and \$1.99. See Figure 7. Imagine again that the line is a ruler and the two dots are your fingers, but this time they are farther apart. Again, move your fingers up and down, just a little bit. What happens to your slope and intercept as you do this? Not very much; they change a little bit. The standard errors for Figure 6 would be much greater than the standard errors for Figure 7. How you choose your design points affects the stability of your parameter estimates. This is the same lesson that the mathematics involving $(\mathbf{X}'\mathbf{X})^{-1}$ gives you. You want to choose your $\mathbf{X}$'s so that efficiency is maximized and the variability of your parameter estimates is minimized. This example does not imply, however, that you should pick prices like \$0.01 and \$1,000,000,000. Your design levels need to make sense for the product.

[‡]Like the alligator, Mick and Keith have been a part of the tutorial for many years. Of course they weren't really writing about choice designs when they wrote this immortal line, but this line is amazingly applicable to the problems of applied design!

[§]I encourage you to actually try this and see what happens! At this point in the tutorial, I am up front demonstrating this. It is a great physical demonstration showing that you you choose $\mathbf{X}$ affects the stability of the parameter estimates.

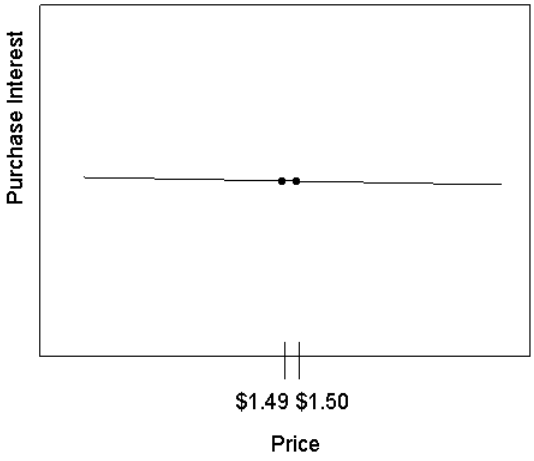


Figure 6 Prices Close Together

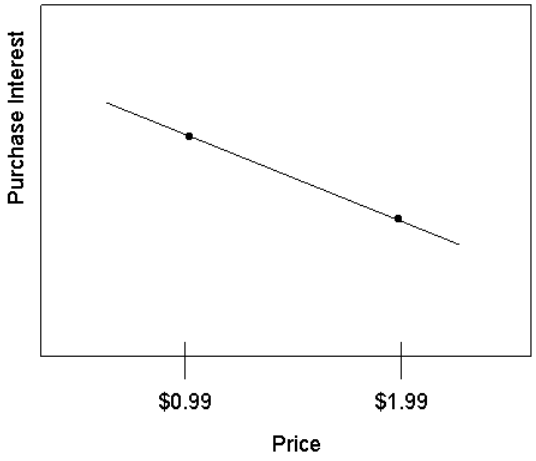


Figure 7 Prices Farther Apart

The Number of Factor Levels

The number of levels of the factors can affect design efficiency. Since two points define a line, it is inefficient to use more than two points to model a linear function. When a quadratic function is used (x and x^2 are included in the model), three points are needed−the two extremes and the midpoint. Similarly, four points are needed for a cubic function. More levels are needed when the functional form is unknown. Extra levels allow for the examination of complicated nonlinear functions, with a cost of decreased efficiency for the simpler functions. When the function is assumed to be linear, experimental points should not be spread throughout the range of experimentation.

We are often tempted to have more levels than we really need, particularly for factors like price. If you expect to model a quadratic price function, you only need three price points. It may make sense to have one or two more price points so that you can test for departures from the quadratic model, but you do not want more than that. You probably would never be interested in a price function more complicated than a cubic function. Creating a design with many price points and then fitting a low-order price function reduces efficiency at analysis time. The more factors you have with more than two or three levels, the harder it is usually going to be to find an orthogonal and balanced design or even a close approximation.

There are times, however, when you can reasonably create factors with more levels than you really need. Say you have a design with two-level and four-level factors and you want to create quadratic price effects, which would mean three evenly-spaced levels. Say you also want the ability to test for departures from a quadratic model. One strategy is to create an eight-level price factor. Then you can recode it as follows: (1 2 3 4 5 6 7 8) → (1 2 3 4 5 1 3 5). Notice that you will end up with twice as many points at the min, middle, and max positions as in the second and fourth positions. This will give you good efficiency for the quadratic effect and some information about higher-order effects. Furthermore, there are many designs with mixtures of (2, 4, and 8)-level factors in 64 runs, which you can easily block. You need 400 runs before you can find a design with mixes of (2, 4, and 5)-level

factors in an orthogonal array. If you are assigning levels with a format, you can assign levels and do the recoding all at the same time.

```
proc format;
   value price  1 = $2.89 2 = $2.99 3 = $3.09 4 = $3.19 5 = $3.29
                6 = $2.89           7 = $3.09           8 = $3.29;
   run;
```

Conjoint, Linear, and Choice Designs

Consider a simple example of three brands each at two prices. We always use linear-model theory to guide us in creating designs for a full-profile conjoint studies. Usually we pick orthogonal arrays for conjoint studies. For choice modeling, the process is somewhat different. We will often use linear-model theory to create a *linear design* from which we could construct a *choice design* to use in a discrete choice study. The conjoint and linear (choice) designs are shown next.

Full-Profile
Conjoint Design

Brand	Price
1	1.99
1	2.99
2	1.99
2	2.99
3	1.99
3	2.99

Linear Design
Used to Make a Choice Design

Brand 1 Price	Brand 2 Price	Brand3 Price
1.99	1.99	1.99
1.99	2.99	2.99
2.99	1.99	2.99
2.99	2.99	1.99

This conjoint design has two factors, brand and price, and six runs or product profiles. Subjects would be shown each combination, such as brand 1 at $1.99 and be asked to report purchase interest through either a rating (for example, on a 1 to 9 scale) or a ranking of the six profiles.

The linear version of the choice design for a pricing study with three brands has three factors (Brand 1 Price, Brand 2 Price, and Brand 3 Price) and one row for each choice set. More generally, the linear design has one factor for each attribute of each alternative (or brand), and brand is not a factor in the linear design. Each brand is a "bin" into which its factors are collected. Subjects would see these sets of products and report which one they would choose (and implicitly, which ones they would not choose). However, before we fit the choice model, we will need to construct a true choice design from the linear design and code the choice design. See Tables 5, 6, and 7.

The linear design has one row per choice set. The choice design has three rows for each choice set, one for each alternative. The linear design and the choice design contain different arrangements of the exact same information. In the linear design, brand is a bin into which its factors are collected (in this case one factor per brand). In the choice design, brand and price are both factors, because the design has been rearranged from one row per choice set to one row per alternative per choice set. For this problem, with only one attribute per brand, the first row of the choice design matrix corresponds to the first value in the linear design matrix, Brand 1 at $1.99. The second row of the choice design matrix corresponds to the second value in the linear design matrix, Brand 2 at $1.99. The third row of the choice design matrix corresponds to the third value in the linear design matrix, Brand 3 at $1.99, and so on.

Table 5

Linear Design

1	2	3
1.99	1.99	1.99
1.99	2.99	2.99
2.99	1.99	2.99
2.99	2.99	1.99

Table 6

Choice Design

Set	Brand	Price
1	1	1.99
	2	1.99
	3	1.99
2	1	1.99
	2	2.99
	3	2.99
3	1	2.99
	2	1.99
	3	2.99
4	1	2.99
	2	2.99
	3	1.99

Table 7

Choice Design Coding

	Brand Effects			Brand by Price		
Set	1	2	3	1	2	3
1	1	0	0	1.99	0	0
	0	1	0	0	1.99	0
	0	0	1	0	0	1.99
2	1	0	0	1.99	0	0
	0	1	0	0	2.99	0
	0	0	1	0	0	2.99
3	1	0	0	2.99	0	0
	0	1	0	0	1.99	0
	0	0	1	0	0	2.99
4	1	0	0	2.99	0	0
	0	1	0	0	2.99	0
	0	0	1	0	0	1.99

A design is coded by replacing each factor with one more columns of *indicator variables* (which are often referred to as "dummy variables") or other codings. In this example, a brand factor is replaced by the three binary variables. We will go through how to construct and code linear and choice designs many times in the examples using a number of different codings. For now, just notice that the conjoint design is different from the linear design, which is different from the choice design. They aren't even the same size! Also note that we *cannot* use linear efficiency criteria to directly construct the choice design bypassing the linear design step. Usually, we will use the `%MktEx` macro to make a linear design, the `%MktRoll` macro to convert it into a choice design, and the TRANSREG procedure to code the choice design.

Here is a slightly more involved illustration of the differences between the linear and final version of a choice design. This example has three brands and three alternatives, one per brand. The category is sports beverages, and they are available in three sizes, at two prices with three different types of tops including a pop up top and two different twist versions. Six choice sets are shown in Table 8.

The linear design has one row per choice set. The full choice design has 36 choice sets. There is one factor for each attribute of each alternative. This experiment has three alternatives, one for each of three brands, and three attributes per alternative. The first goal is to make a linear design where each attribute, both within and between alternatives, is orthogonal and balanced, or at least very nearly so. Brand is the bin into which the linear factors are collected, and it becomes an actual attribute in the choice design. The right partition of the table shows the choice design. The `x1` attribute in the choice design is made from `x1`, `x4`, and `x7`, in the linear design. These are the three size factors. Similarly, `x2` is made from `x2`, `x5`, and `x8`, in the linear design. These are the three price factors. Finally, `x3` is made from the three top factors, `x3`, `x6`, and `x9`.

Table 8

Linear Design									Choice Design			
Brand 1			Brand 2			Brand 3						
x1	x2	x3	x4	x5	x6	x7	x8	x9	Brand	x1	x2	x3
16 oz.	0.89	Twist 1	24 oz.	0.89	Twist 2	20 oz.	0.99	Pop-Up	1	16 oz.	0.89	Twist 1
									2	24 oz.	0.89	Twist 2
									3	20 oz.	0.99	Pop-Up
20 oz.	0.99	Pop-Up	24 oz.	0.89	Twist 1	20 oz.	0.89	Twist 2	1	20 oz.	0.99	Pop-Up
									2	24 oz.	0.89	Twist 1
									3	20 oz.	0.89	Twist 2
20 oz.	0.89	Twist 1	20 oz.	0.99	Twist 2	16 oz.	0.89	Twist 2	1	20 oz.	0.89	Twist 1
									2	20 oz.	0.99	Twist 2
									3	16 oz.	0.89	Twist 2
20 oz.	0.89	Twist 1	16 oz.	0.99	Twist 1	24 oz.	0.99	Pop-Up	1	20 oz.	0.89	Twist 1
									2	16 oz.	0.99	Twist 1
									3	24 oz.	0.99	Pop-Up
16 oz.	0.89	Twist 2	24 oz.	0.99	Pop-Up	16 oz.	0.99	Twist 2	1	16 oz.	0.89	Twist 2
									2	24 oz.	0.99	Pop-Up
									3	16 oz.	0.99	Twist 2
24 oz.	0.99	Twist 2	16 oz.	0.89	Twist 2	16 oz.	0.89	Pop-Up	1	24 oz.	0.99	Twist 2
									2	16 oz.	0.89	Twist 2
									3	16 oz.	0.89	Pop-Up

Blocking the Choice Design

The sports beverage example has 36 choice sets. This may be too many judgments for one subject to make. How many blocks to use depends on the number of choice sets and the complexity of the choice task. For example, 36 choice sets might be small enough that no blocking is necessary, or instead, they may be divided into 2 blocks of size 18, 3 blocks of size 12, 4 blocks of size 9, 6 blocks of size 6, 9 blocks of size 4, 12 blocks of size 3, 18 blocks of size 2, or even 36 blocks of size 1. Technically, subjects *should* each see exactly one choice set. Showing subjects more than one choice set is economical, and in practice, most researchers almost always show multiple choice sets to each subject. The number of sets shown does not change the expected utilities, however, it does affect the covariance structure. Sometimes, attributes will be highly correlated within blocks, particularly with small block sizes, but that is not a problem as long as they are not highly correlated over the entire design.

Efficiency of a Choice Design

All of the efficiency theory discussed so far concerned linear models. In linear models, the parameter estimates $\hat{\boldsymbol{\beta}}$ have variances proportional to $(\mathbf{X}'\mathbf{X})^{-1}$. In contrast, the variances of the parameter estimates in the discrete choice multinomial logit model are given by

$$V(\hat{\beta}) = -\left[\frac{\partial^2 \ell(\beta)}{\partial \beta^2}\right]^{-1} = \left[\Sigma_{k=1}^{n} N \left[\frac{\Sigma_{j=1}^{m} \exp(x_j'\beta) x_j x_j'}{\Sigma_{j=1}^{m} \exp(x_j'\beta)} - \frac{(\Sigma_{j=1}^{m} \exp(x_j'\beta) x_j)(\Sigma_{j=1}^{m} \exp(x_j'\beta) x_j)'}{(\Sigma_{j=1}^{m} \exp(x_j'\beta))^2}\right]\right]^{-1}$$

where

$$\ell(\beta) = \Pi_{k=1}^{n} \frac{\exp((\Sigma_{j=1}^{m} f_j x_j')\beta)}{(\Sigma_{j=1}^{m} \exp(x_j'\beta))^N}$$

m − brands
n − choice sets
N − people

In the choice model, ideally we would like to pick $\mathbf{x}$'s that make this variance matrix "small." Unfortunately, we cannot do this unless we know $\boldsymbol{\beta}$, and if we knew $\boldsymbol{\beta}$, we would not need to do the experiment. However, in the chair example on pages 363−382, we will see how to make an efficient choice design when we are willing to make assumptions about $\boldsymbol{\beta}$.

Because we do not know $\boldsymbol{\beta}$, we will often create experimental designs for choice models using efficiency criteria for linear models. We make a good design for a linear model by picking $\mathbf{x}$'s that minimize a function of $(\mathbf{X}'\mathbf{X})^{-1}$ and then convert our linear design into a choice design. Certain assumptions must be made before applying ordinary general-linear-model theory to problems in marketing research. The usual goal in linear modeling is to estimate parameters and test hypotheses about those parameters. Typically, independence and normality are assumed. In full-profile conjoint analysis, each subject rates all products and separate ordinary-least-squares analyses are run for each subject. This is not a standard general linear model; in particular, observations are not independent and normality cannot be assumed. Discrete choice models, which are nonlinear, are even more removed from the general linear model.

Marketing researchers have always made the critical assumption that designs that are good for general linear models are also good designs for conjoint analysis and discrete choice models. We also make this assumption. We will assume that an efficient design for a linear model is a good design for the multinomial logit model used in discrete choice studies. We assume that if we create the linear design (one row per choice set and all of the attributes of all of the alternatives comprise that row), and if we strive for linear-model efficiency (near balance and orthogonality), then we will have a good design for measuring the utility of each alternative and the contributions of the factors to that utility. When we construct choice designs in this way, our designs will have two nice properties. 1) Each attribute level will occur equally often (or at least nearly equally often) for each attribute of each alternative across all choice sets. 2) Each attribute will be independent of every other attribute (or at least nearly independent), both those in the current alternative and those in all of the other alternatives. The design techniques discussed in this book, based on the assumption that linear design efficiency is a good surrogate for choice design goodness, have been used quite successfully in the field for many years.

In most of the examples, we will use the `%MktEx` macro to create a good linear design, from which we will construct our choice design. This seems to be a good, safe strategy. It is a good strategy because it makes designs where all attributes, both within and between alternatives, are orthogonal or at least nearly so. It is safe in the sense that you have enough choice sets and collect the right information so that very complex models, including models with alternative-specific effects, availability effects, and cross effects, can be fit. However, it is good to remember that when you run the `%MktEx` macro and you get an efficiency value, it corresponds to the linear design, not the choice design. It is a surrogate for the criterion of interest, the efficiency of the choice design, which is unknowable unless you know the parameters.

Coding, Efficiency, Balance, and Orthogonality

We mentioned on page 54 that we use a special orthogonal coding of **X** when computing design efficiency. This section shows that coding and other codings. Even if you gloss over the mathematical details, this section is informative, because it provides insights into coding and the meaning of 100% efficiency and less than 100% efficient designs.

Here are nonorthogonal less-than-full-rank *binary* or *indicator* codings for two-level through five-level factors. There is one column for each level, and the coding contains a 1 when the level matches the column and a zero otherwise. We will use these codings in many places throughout the examples.

Two-Level

a	1	0
b	0	1

Three-Level

a	1	0	0
b	0	1	0
c	0	0	1

Four-Level

a	1	0	0	0
b	0	1	0	0
c	0	0	1	0
d	0	0	0	1

Five-Level

a	1	0	0	0	0
b	0	1	0	0	0
c	0	0	1	0	0
d	0	0	0	1	0
e	0	0	0	0	1

Here are nonorthogonal full-rank binary or indicator codings for two-level through five-level factors. This coding is like the full-rank coding above, except that the column corresponding to the *reference level* has been dropped. Frequently, the reference level is the last level, but it can be any level. We will use these codings in many places throughout the examples.

Two-Level

a	1
b	0

Three-Level

a	1	0
b	0	1
c	0	0

Four-Level

a	1	0	0
b	0	1	0
c	0	0	1
d	0	0	0

Five-Level

a	1	0	0	0
b	0	1	0	0
c	0	0	1	0
d	0	0	0	1
e	0	0	0	0

Here are nonorthogonal *effects coding* for two-level[¶] through five-level factors. The effects coding differs from the full-rank binary coding in that the former always has a -1 to indicate the reference level. The binary and effects codings are explained in more detail in the SAS/STAT manual, PROC TRANSREG, DETAILS, "ANOVA Codings" section. We will use these codings in many places throughout the examples.

Two-Level

a	1
b	-1

Three-Level

a	1	0
b	0	1
c	-1	-1

Four-Level

a	1	0	0
b	0	1	0
c	0	0	1
d	-1	-1	-1

Five-Level

a	1	0	0	0
b	0	1	0	0
c	0	0	1	0
d	0	0	0	1
e	-1	-1	-1	-1

Table 9, using the design in Table 8, shows the less-than-full-rank binary coding (brand, 3 parameters), the full-rank binary coding (size, 2 parameters), and the effects coding (top, 2 parameters). Price (1 parameter) is not coded and instead is entered as is for a linear price effect.

[¶]The two-level effects coding is orthogonal, but the three-level and beyond codings are not.

Table 9

Choice Design				Coding							
Brand	x1	x2	x3	Brand 1	Brand 2	Brand 3	16 oz.	20 oz.	Price	Twist 1	Twist 2
1	16 oz.	0.89	Twist 1	1	0	0	1	0	0.89	1	0
2	24 oz.	0.89	Twist 2	0	1	0	0	0	0.89	0	1
3	20 oz.	0.99	Pop-Up	0	0	1	0	1	0.99	-1	-1
1	20 oz.	0.99	Pop-Up	1	0	0	0	1	0.99	-1	-1
2	24 oz.	0.89	Twist 1	0	1	0	0	0	0.89	1	0
3	20 oz.	0.89	Twist 2	0	0	1	0	1	0.89	0	1
1	20 oz.	0.89	Twist 1	1	0	0	0	1	0.89	1	0
2	20 oz.	0.99	Twist 2	0	1	0	0	1	0.99	0	1
3	16 oz.	0.89	Twist 2	0	0	1	1	0	0.89	0	1
1	20 oz.	0.89	Twist 1	1	0	0	0	1	0.89	1	0
2	16 oz.	0.99	Twist 1	0	1	0	1	0	0.99	1	0
3	24 oz.	0.99	Pop-Up	0	0	1	0	0	0.99	-1	-1
1	16 oz.	0.89	Twist 2	1	0	0	1	0	0.89	0	1
2	24 oz.	0.99	Pop-Up	0	1	0	0	0	0.99	-1	-1
3	16 oz.	0.99	Twist 2	0	0	1	1	0	0.99	0	1
1	24 oz.	0.99	Twist 2	1	0	0	0	0	0.99	0	1
2	16 oz.	0.89	Twist 2	0	1	0	1	0	0.89	0	1
3	16 oz.	0.89	Pop-Up	0	0	1	1	0	0.89	-1	-1

Here is the *orthogonal contrast coding* for two-level through five-level factors. These are the same as the orthogonal codings that will be discussed in detail next, except that this version has been scaled so that all values are integers.

Two-Level

a	1
b	-1

Three-Level

a	1	-1
b	0	2
c	-1	-1

Four-Level

a	1	-1	-1
b	0	2	-1
c	0	0	3
d	-1	-1	-1

Five-Level

a	1	-1	-1	-1
b	0	2	-1	-1
c	0	0	3	-1
d	0	0	0	4
e	-1	-1	-1	-1

Here is the *standardized orthogonal coding* for two-level through five-level factors that the design software uses internally.

Two-Level

a	1.00
b	-1.00

Three-Level

a	1.22	-0.71
b	0	1.41
c	-1.22	-0.71

Four-Level

a	1.41	-0.82	-0.58
b	0	1.63	-0.58
c	0	0	1.73
d	-1.41	-0.82	-0.58

Five-Level

a	1.58	-0.91	-0.65	-0.50
b	0	1.83	-0.65	-0.50
c	0	0	1.94	-0.50
d	0	0	0	2.00
e	-1.58	-0.91	-0.65	-0.50

Notice that the sum of squares for the orthogonal coding of the two-level factor is 2. For both columns of the three-level factor, the sums of squares are 3; for the three columns of the four-level factor, the sums of squares are all 4; and for the four columns of the five-level factor, the sums of squares are all 5. Also notice that each column within a factor is orthogonal to all of the other columns−the sum of cross products is zero. For example, in the last two columns of the five-level factor, $-0.65 \times -0.5 + -0.65 \times -0.5 + 1.94 \times -.05 + 0 \times 2 + -0.65 \times -0.5 = 0$. Finally notice that the codings for each level form a contrast−the *ith* level versus all of the preceding levels and the last level.

Recall that our measures of design efficiency are scaled to range from 0 to 100.

$$\begin{aligned} A\text{-efficiency} &= 100 \times \frac{1}{N_D \text{ trace}\left((\mathbf{X}'\mathbf{X})^{-1}\right)/p} \\ D\text{-efficiency} &= 100 \times \frac{1}{N_D \left|(\mathbf{X}'\mathbf{X})^{-1}\right|^{1/p}} \end{aligned}$$

When computing *D*-efficiency or *A*-efficiency, we code $\mathbf{X}$ so that when the design is orthogonal and balanced, $\mathbf{X}'\mathbf{X} = N_D\mathbf{I}$ where $\mathbf{I}$ is a $p \times p$ identity matrix. When our design is orthogonal and balanced, $(\mathbf{X}'\mathbf{X})^{-1} = \frac{1}{N_D}\mathbf{I}$, and $\text{trace}\left((\mathbf{X}'\mathbf{X})^{-1}\right)/p = |(\mathbf{X}'\mathbf{X})^{-1}|^{1/p} = 1/N_D$. In this case, the two denominator terms cancel and efficiency is 100%. As the average variance increases, efficiency decreases.

This next example shows the coding of a 2×6 full-factorial design in 12 runs using a coding function that requires that the factor levels are consecutive positive integers beginning with one and ending with m for an m-level factor. Note that the IML operator # performs ordinary (scalar) multiplication, and ## performs exponentiation.

```
proc iml; /* orthogonal coding, levels must be 1, 2, ..., m */
   reset fuzz;

   start orthogcode(x);
      levels = max(x);
      xstar  = shape(x, levels - 1, nrow(x))‘;
      j = shape(1 : (levels - 1), nrow(x), levels - 1);
      r = sqrt(levels # (x / (x + 1))) # (j = xstar) -
          sqrt(levels / (j # (j + 1))) # (j > xstar | xstar = levels);
      return(r);
      finish;
   design = (1:2)‘ @ j(6, 1, 1) || {1, 1} @ (1:6)‘;
   x = j(12, 1, 1) || orthogcode(design[,1]) || orthogcode(design[,2]);
   print design[format=1.]  ’  ’ x[format=5.2 colname={’Int’ ’Two’ ’Six’}];

   xpx = x‘ * x;     print xpx[format=best5.];
   inv = inv(xpx);   print inv[format=best5.];
   d_eff = 100 / (nrow(x) #   det(inv) ## (1 / ncol(inv)));
   a_eff = 100 / (nrow(x) # trace(inv)       / ncol(inv));
   print    ’D-efficiency =’ d_eff[format=6.2]
         ’   A-efficiency =’ a_eff[format=6.2];
```

```
                  X
DESIGN         Int   Two   Six

     1 1      1.00  1.00  1.73 -1.00 -0.71 -0.55 -0.45
     1 2      1.00  1.00  0.00  2.00 -0.71 -0.55 -0.45
     1 3      1.00  1.00  0.00  0.00  2.12 -0.55 -0.45
     1 4      1.00  1.00  0.00  0.00  0.00  2.19 -0.45
     1 5      1.00  1.00  0.00  0.00  0.00  0.00  2.24
     1 6      1.00  1.00 -1.73 -1.00 -0.71 -0.55 -0.45
     2 1      1.00 -1.00  1.73 -1.00 -0.71 -0.55 -0.45
     2 2      1.00 -1.00  0.00  2.00 -0.71 -0.55 -0.45
     2 3      1.00 -1.00  0.00  0.00  2.12 -0.55 -0.45
     2 4      1.00 -1.00  0.00  0.00  0.00  2.19 -0.45
     2 5      1.00 -1.00  0.00  0.00  0.00  0.00  2.24
     2 6      1.00 -1.00 -1.73 -1.00 -0.71 -0.55 -0.45

                        XPX

        12     0     0     0     0     0     0
         0    12     0     0     0     0     0
         0     0    12     0     0     0     0
         0     0     0    12     0     0     0
         0     0     0     0    12     0     0
         0     0     0     0     0    12     0
         0     0     0     0     0     0    12

                        INV

     0.083     0     0     0     0     0     0
         0 0.083     0     0     0     0     0
         0     0 0.083     0     0     0     0
         0     0     0 0.083     0     0     0
         0     0     0     0 0.083     0     0
         0     0     0     0     0 0.083     0
         0     0     0     0     0     0 0.083

                    D_EFF                   A_EFF

    D-efficiency = 100.00    A-efficiency = 100.00
```

With this orthogonal and balanced design, $\mathbf{X}'\mathbf{X} = N_D\mathbf{I} = 12\mathbf{I}$, which means $(\mathbf{X}'\mathbf{X})^{-1} = \frac{1}{N_D}\mathbf{I} = \frac{1}{12}\mathbf{I}$, and D-efficiency = 100%. With a nonorthogonal design, for example with the first 10 rows of the 2×6 full-factorial design, D-efficiency and A-efficiency are less than 100%.

```
design = design[1:10,];
x = j(10, 1, 1) || orthogcode(design[,1]) || orthogcode(design[,2]);
inv = inv(x‘ * x);
d_eff = 100 / (nrow(x) #   det(inv) ## (1 / ncol(inv)));
a_eff = 100 / (nrow(x) # trace(inv)       / ncol(inv));
print    ’D-efficiency =’ d_eff[format=6.2]
      ’   A-efficiency =’ a_eff[format=6.2];
quit;
```

```
                                D_EFF                    A_EFF

              D-efficiency =  92.90    A-efficiency =  84.00
```

In this case, $|(\mathbf{X}'\mathbf{X})^{-1}|^{1/p}$ and $\text{trace}\left((\mathbf{X}'\mathbf{X})^{-1}\right)/p$ are multiplied in the denominator of the efficiency formulas by $\frac{1}{N_D} = \frac{1}{10}$. If an orthogonal and balanced design were available for this problem, then $(\mathbf{X}'\mathbf{X})^{-1}$ would equal $\frac{1}{N_D}\mathbf{I} = \frac{1}{10}\mathbf{I}$. Since an orthogonal and balanced design is not possible (6 does not divide 10), both *D*-efficiency and *A*-efficiency will be less than 100%, even with the optimal design. A main-effects, orthogonal and balanced design, with a variance matrix equal to $\frac{1}{N_D}\mathbf{I}$, is the standard by which 100% efficiency is gauged, even when we know such a design cannot exist. The standard is the average variance for the maximally efficient *potentially hypothetical* design, which is knowable, not the average variance for the optimal design, which for many practical problems we have no way of knowing.

For our purposes in this book, we will only consider experimental designs with at least as many runs as parameters. A *saturated* or *tight* design has as many runs as there are parameters. The number of parameters in a main-effects model is the sum of the numbers of levels of all of the factors, minus the number of factors, plus 1 for the intercept. Equivalently, since there are $m - 1$ parameters in an m-level factor, the number of parameters is $1 + \sum_{j=1}^{k}(m_j - 1)$ for k factors, each with m_j levels.

If a main-effects design is orthogonal and balanced, then the design must be at least as large as the saturated design and the number of runs must be divisible by the number of levels of all the factors and by the products of the number of levels of all pairs of factors. For example, a $2 \times 2 \times 3 \times 3 \times 3$ design cannot be orthogonal and balanced unless the number of runs is divisible by 2 (twice because there are two 2's), 3 (three times because there are three 3's), $2 \times 2 = 4$ (once, because there is one pair of 2's), $2 \times 3 = 6$ (six times, two 2's times three 3's), and $3 \times 3 = 9$ (three times, three pairs of 3's). If the design is orthogonal and balanced, then all of the divisions will work without a remainder. However, all of the divisions working is a necessary but not sufficient condition for the existence of an orthogonal and balanced design. For example, 45 is divisible by 3 and $3 \times 3 = 9$, but an orthogonal and balanced saturated design 3^{22} (22 three-level factors) in 45 runs does not exist.

Orthogonally Coding Price and Other Quantitative Attributes

For inherently quantitative factors like price, you may want to use different strategies for coding instead of using indicator variables or effects coding. When we create a design with a quantitative factor such as price, we do not have to do anything special. The orthogonal coding what we use to make qualitative factors is just as applicable when the factor will become quantitative. See page 107 for more information. However, for analysis, we may want a different coding than the binary or effects coding. Imagine, for example, a choice experiment in the SUV category with price as an attribute

and with levels of $27,500, $30,000, and $32,500. You probably will not want to code them as is and just add these prices directly to the model, because these values are considerably larger than the other values in your coded independents, which will usually consist of values like -1, 0, and 1. You might believe that choice is not a linear function of price; it may be nonlinear or quadratic. Hence, you might think about adding a price-squared term, but squaring values this large is almost certain to cause collinearity. When you are dealing with factors like this, you are usually better off recoding them in a "nicer" way. The first column of the next table shows some of the steps in the recoding.

Price	Centered Price	Divide By Increment	Square
27,500	27,500 - 30,000 = -2,500	-2,500 / 2,500 = -1	$-1^2 = 1$
30,000	30,000 - 30,000 = 0	0 / 2,500 = 0	$0^2 = 0$
32,500	32,500 - 30,000 = 2,500	2,500 / 2,500 = 1	$1^2 = 1$

The second column shows the results of centering them−subtracting the mean price of $30,000. The third column shows the results of dividing the centered values by the increment between values, 2,500. The fourth column shows the square of the third column. These last two columns would make much better linear and quadratic price terms than the original price and the original price squared, however, we can do better still. The first part of the next table shows the final steps and the full, orthogonal, quadratic coding.

Code	Centered Quadratic	Multiply Through	Orthogonal Code
1 -1 1	1 - 2/3 = 1/3	3 × 1/3 = 1	1 -1 1
1 0 0	0 - 2/3 = -2/3	3 × −2/3 = -2	1 0 -2
1 1 1	1 - 2/3 = 1/3	3 × 1/3 = 1	1 1 1

The first coding consists of an intercept, a linear term, and a quadratic term. Notice that the sum of the quadratic term is not zero, so the quadratic term is not orthogonal to the intercept. We can correct this by centering (subtracting the mean which is 2/3). After centering, all three columns are orthogonal. We can make the coding nicer still by multiplying the quadratic term by 3 to get rid of the fractions. The full orthogonal coding is shown in the last set of columns. Note however, that only the last two columns would be used. The intercept is just there to more clearly show that all columns are orthogonal. This orthogonal coding will work for any three-level quantitative factor with equal intervals between adjacent levels.

For four equally-spaced levels, and with less detail, the linear and quadratic coding is shown in the last two columns of the next table.

Price	Center	Divide	Make Into Integers	Square	Center	Smallest Integers	Code
27500	-3750	-1.5	-3	9	4	1	-3 1
30000	-1250	-0.5	-1	1	-4	-1	-1 -1
32500	1250	0.5	1	1	-4	-1	1 -1
35000	3750	1.5	3	9	4	1	3 1

Canonical Correlations

We will use canonical correlations to evaluate nonorthogonal designs and the extent to which factors are correlated or are not independent. To illustrate, consider a design with four three-level factors in 9 runs shown next along with its coding.

Linear Design

x1	x2	x3	x4
1	1	1	1
1	2	3	3
1	3	2	2
2	1	2	3
2	2	1	2
2	3	3	1
3	1	3	2
3	2	2	1
3	3	1	3

Coded Linear Design

x1			x2			x3			x4		
1	2	3	1	2	3	1	2	3	1	2	3
1	0	0	1	0	0	1	0	0	1	0	0
1	0	0	0	1	0	0	0	1	0	0	1
1	0	0	0	0	1	0	1	0	0	1	0
0	1	0	1	0	0	0	1	0	0	0	1
0	1	0	0	1	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0	1	1	0	0
0	0	1	1	0	0	0	0	1	0	1	0
0	0	1	0	1	0	0	1	0	1	0	0
0	0	1	0	0	1	1	0	0	0	0	1

Each three-level factor can be coded with three columns that contains the less-than-full-rank binary coding (see page 64). A factor can be recoded by applying a coefficient vector $\boldsymbol{\alpha}' = (\alpha_1\ \alpha_2\ \alpha_3)$ or $\boldsymbol{\beta}' = (\beta_1\ \beta_2\ \beta_3)$ to a coded factor to create a single column. In other words, the original coding of (1 2 3) can be replaced with arbitrary $(\alpha_1\ \alpha_2\ \alpha_3)$ or $(\beta_1\ \beta_2\ \beta_3)$ If two factors are orthogonal, then for all choices of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, the simple correlation between recoded columns is zero. A *canonical correlation* shows the maximum correlation between two recoded factors that can be obtained with the optimal $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. This design, 3^4 in 9 runs is orthogonal so for all pairs of factors and all choices of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, the simple correlations between recoded factors will be zero. The canonical correlation between a factor and itself is 1.0.

For nonorthogonal designs and designs with interactions, the canonical-correlation matrix is not a substitute for looking at the variance matrix discussed on pages 196, 243, and 683. It just provides a quick and more-compact picture of the correlations between the factors. The variance matrix is sensitive to the actual model specified and the actual coding. The canonical-correlation matrix just tells you if there is some correlation between the main effects. A matrix of canonical correlations provides a useful picture of the orthogonality or lack of orthogonality in a design. For example, this canonical-correlation matrix from the vacation example on page 194, shown next, shows a design with 16 factors that is mostly orthogonal. However, `x13-x15` are not orthogonal to each other. Still, with $r^2 = 0.25^2 = 0.0625$, these factors are nearly independent.

The Process of Designing a Choice Experiment

It is important that you understand a number of things in this chapter before you design your first choice experiment. Most of this chapter is fairly straight-forward, but without a clear understanding of it, you will no doubt get confused when you actually design an experiment. You should go back and review if you are not completely comfortable with the meaning of any of these terms: linear design, choice design, generic choice design, factors, attributes, alternatives, choice sets, orthogonality, balance, and efficiency. In particular, the meaning of *linear design* and *choice design* (pages 60–61) and the

Canonical Correlation Matrix

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16
x1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
x4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
x5	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
x6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
x7	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
x8	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
x9	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
x10	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
x11	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
x12	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
x13	0	0	0	0	0	0	0	0	0	0	0	0	1	0.25	0.25	0
x14	0	0	0	0	0	0	0	0	0	0	0	0	0.25	1	0.25	0
x15	0	0	0	0	0	0	0	0	0	0	0	0	0.25	0.25	1	0
x16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

relationship between the two is fundamental and the source of a great deal of confusion when most people start out. Make sure you understand them. You do not have to understand the formula for the variance matrix for a choice model, the orthogonal coding, or the formuulas for efficiency. However, you should be comfortable with the idea of average variability of the parameter estimates and how that is related to efficiency.

This section lists the steps in designing a choice experiment. The next section illustrates these steps with a simple example. All of the page numbers in this section refer to the more complicated examples in the discrete choice chapter. Before consulting them, work through the simple example in the next section.

The first step in designing a choice experiment involves determining:

- Is this a generic study (no brands) or a branded study? Branded studies have a label for each alternative that conveys meaning beyond ordinary attributes. Brand names are the most common example. The destinations in the vaction example (pages 184 and 229) also act like brands. In a generic study, the alternatives are simply bundles of attributes (page 363).
- If it is branded, what are the brands?
- How many alternatives?
- Is there a constant (none, no purchase, delay purchase, or stick with my regular brand) alternative?
- What are the attributes of all of the alternatives, and what are their levels?
- Are any the attributes generic? In other words, are there attributes that you expect to behave the same way across all alternatives?
- Are any the attributes alternative-specific? In other words, are there attributes that you expect to behave differently across all alternatives (brand by attribute interactions)?

Step 1. Write down all of your attributes of all of your alternatives and their levels. See for example pages 184, 229, 284, and 363 .

Step 2. Is this a generic study (like the chair example on page 363) or a branded example (like the vacation examples on pages 184 and 229 and the food example on page 284)?

Step 3. If this is a branded study:

- Use the `%MktRuns` macro to suggest the number of choice sets. See page 740 for documentation and pages 156, 185, 230, 234, 294, 296, and 345 for examples.
- Use the `%MktEx` macro to make a linear design. See page 667 for documentation and pages 158, 158, 178, 188, 196, 232, 233, 235, 243, 287, 292, 297, 297, 300, 346, 383, 385, 386, 389, 404, 410, 413, 420, 420, 423, 430, 435, 438, 449, 452, 452, 455, 458, 460, 461, 464, and 465 for examples.
- Use the `%MktEval` macro to evaluate the linear design. See page 663 for documentation and pages 160, 161, 161, 194, 197, 232, 242, 292, 297, and 348 for examples.
- Print and check the linear design. See for example page 159.
- Use the `%MktKey` and `%MktRoll` macros to make a choice design from the linear design. See page 735 for documentation on the `%MktRoll` macro, page 710 for documentation on the `%MktKey` macro, and pages 165, 200, 223, 247, 317, 353, 363, 378, 416, 425, 432, and 438, for examples.

Step 4. If this is a generic study:

- Use the `%MktRuns` macro to suggest a size for the candidate design. See page 740 for documentation and page 364 for an example.
- Use the `%MktEx` macro to make a candidate design. See page 667 for documentation and pages 363, 365, 374, and 378 for examples.
- Use the `%MktLab` macro to add alternative flags. See page 712 for documentation and page 365 for an example.
- Print and check the candidate design. See for example page 365.
- Use the `%ChoiceEff` macro to find an efficient choice design. See page 600 for documentation and pages 366, 370, 372, 374, 375, 378, 379, 417, 425, 428, 432, 438, 440, 455, 458, 461, 464, and 465 for examples.
- Print and check the choice design. See for example page 369.
- Go back and try the `%MktEx` step with other size choice sets. Stop when you feel comfortable with the results.

Step 5. Continue processing the design:

- Print and check the choice design. See for example page 201.
- Assign formats and labels. See for example page 202.
- Print and check the choice design. See for example page 202.
- Use the `%ChoicEff` macro to evaluate the design. See page 600 for documentation and pages 166, 203, 248, 320, 321, 437, 452, and 453 for examples.

- For larger designs, you will need to block the design. See page 638 for documentation and pages 244 and 308 for examples. Alternatively, with the linear design approach, you can sometimes just add a blocking factor directly to the linear design. See page 197 for an example.

Step 6. Collect data and process the design:

- Print or otherwise generate the choice tasks, and then collect and enter the data. See for example page 207.
- Use the `%MktMerge` macro to merge the data and the design. See page 723 for documentation and pages 171, 208, 223, 255, 331, and 339, for examples.
- Print part of the data and design and check the results. See for example page 208.
- Optionally, particularly for large data sets, you can aggregate the data set using PROC SUMMARY. See for example page 332.
- Use the TRANSREG procedure to code the design. See for example pages 173, 209, 214, 216, 219, 223, 255, 263, 265, 269, 275, 276, 278, 280, 333, 337, 340, and 357.
- Print part of the coded design and check the results. See for example page 209.
- Use the `%PhChoice` macro to customize the output. See page 748 for documentation and page 143 for an example.
- Use the PHREG procedure to fit the multinomial logit model. See for example pages 257, 259, 264, 266, 273, 278, 280, 334, 337, 342, 357, and 360.

There are many variations not covered in this simple outline. For example, you could use the `%ChoicEff` macro even for branded studies. Also, restricted designs and partial profile designs are not mentioned here. See the examples in the discrete choice chapter (pages 141−465) for lots of other possibilities.

A Simple Choice Experiment from A to Z

This section outlines a small and simple choice experiment from start to finish. The examples on pages 141 through 465 tend to be much more involved and have many more nuances. This example shows the basic steps in the context of a simple example with no complications. Understanding this one will help you with the more involved examples that come later.

The category is breakfast bars, and there are three brands, Branolicious, Brantopia, and Brantasia.[‖] The choice sets will consist of three brands and a constant (no purchase) alternative. Each brand has two attributes, a four-level price factor and a two-level number of bars per box attribute. The prices are $2.89, $2.99, $3.09, and $3.19, and the sizes are 6 count and 8 count. The design will consist of the following factors, shown here organized by brand and also organized by attribute. There is only one set of attributes shown here, however it is shown in two different ways.

[‖] Of course real studies would use real brands. Since we have not collected real data, we cannot use real brand names. We picked these silly names so no one would confuse our artificial data with real data.

Factors Organized By Brand

Linear Factor Name	Levels	Brand	Choice Design Attribute
x1	4 levels	Branolicious	Price
x2	2 levels	Branolicious	Count
x3	4 levels	Brantopia	Price
x4	2 levels	Brantopia	Count
x5	4 levels	Brantasia	Price
x6	2 levels	Brantasia	Count

Factors Organized By Attribute

Linear Factor Name	Levels	Brand	Choice Design Attribute
x1	4 levels	Branolicious	Price
x3	4 levels	Brantopia	Price
x5	4 levels	Brantasia	Price
x2	2 levels	Branolicious	Count
x4	2 levels	Brantopia	Count
x6	2 levels	Brantasia	Count

We need a linear design with 6 factors: Branolicious Price, Branolicious Count, Brantopia Price, Brantopia Count, Brantasia Price, and Brantasia Count. From it, we will make a choice design with three attributes, brand, count, and price. We can use the `%MktRuns` macro to suggest the number of choice sets. The input is the number of levels of all of the factors. See for example pages 185 and 230 for more about the syntax of this macro.

```
title 'Cereal Bars';

%mktruns( 4 2  4 2  4 2 )
```

The output from the macro is shown next. It tells us that there are 3 two-level factors and 3 four-level factors. The saturated design has 13 runs or rows, so we need at least 13 choice sets with this approach. The full-factorial design has 512 runs, so there are a maximum of 512 possible choice sets. Sixteen choice sets is ideal since there is an orthogonal-array design that we could use. The macro stops considering larger sizes when it finds a perfect size (in this case 32) that is twice as big as another perfect size (16). The last part of the output lists the orthgonal arrays.

```
                              Cereal Bars

                           Design Summary

                        Number of
                        Levels       Frequency

                             2               3
                             4               3

                              Cereal Bars


                  Saturated      = 13
                  Full Factorial = 512
```

```
   Some Reasonable                        Cannot Be
      Design Sizes       Violations       Divided By

              16 *                0
              32 *                0
              24                  3      16
              20                 12       8 16
              28                 12       8 16
              14                 18       4  8 16
              18                 18       4  8 16
              22                 18       4  8 16
              26                 18       4  8 16
              30                 18       4  8 16

  * - 100% Efficient Design can be made with the MktEx Macro.

                              Cereal Bars

 n                    Design                          Reference

16    2 **  6          4 **  3                       Fractional-Factorial
16    2 **  3          4 **  4                       Fractional-Factorial
32    2 ** 22          4 **  3                       Fractional-Factorial
32    2 ** 19          4 **  4                       Fractional-Factorial
32    2 ** 16          4 **  5                       Fractional-Factorial
32    2 ** 15          4 **  3   8 **  1            Fractional-Factorial
32    2 ** 13          4 **  6                       Fractional-Factorial
32    2 ** 12          4 **  4   8 **  1            Fractional-Factorial
32    2 ** 10          4 **  7                       Fractional-Factorial
32    2 **  9          4 **  5   8 **  1            Fractional-Factorial
32    2 **  7          4 **  8                       Fractional-Factorial
32    2 **  6          4 **  6   8 **  1            Fractional-Factorial
32    2 **  4          4 **  9                       Fractional-Factorial
32    2 **  3          4 **  7   8 **  1            Fractional-Factorial
```

We will create a linear design with 16 choice sets. The `%MktRuns` macro suggests 16 because it can be divided by 2 (we have two-level factors), 4 (we have four-level factors), 2×2 (we have more than one two-level factor), 4×4 (we have more than one four-level factor), and 2×4 (we have both two-level factors and four-level factors). The number of choice sets must be divisible by all of these it the design is going to be orthogonal and balanced. Sixteen is a reasonable number of judgments for people to make, and the other suggestions (24, 20, 28, 14, 18, 22, 26, 30) all cannot be divided by at least one of these numbers.

We will use the `%MktEx` macro to get our linear design. It accepts a factor-level list like `%MktRuns` along with the number of runs or choice sets. We specify a random number seed so that we always get the same design if we rerun the `%MktEx` macro.

```
%mktex( 4 2  4 2  4 2, n=16, seed=17 )
```

The output is next. The `%MktEx` macro found a 100% efficient, orthogonal and balanced design with

3 two-level factors and 3 four-level factors, just as the %MktRuns told us it would. The levels are all positive integers, starting with 1 and continuing to the number of levels.

Cereal Bars

Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
1	Start	100.0000	100.0000	Tab
1	End	100.0000		

Cereal Bars

The OPTEX Procedure

Class Level Information

Class	Levels	Values
x1	4	1 2 3 4
x2	2	1 2
x3	4	1 2 3 4
x4	2	1 2
x5	4	1 2 3 4
x6	2	1 2

Cereal Bars

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	100.0000	100.0000	100.0000	0.9014

Next, we examine some of the properties of the design and we print it. The %MktEval macro tells us which factors are orthogonal and which are correlated. It also tells us how often each level occurs, how often each pair of levels occurs across pairs of factors, and how often each run or choice set occurs.

```
title2 'Examine Correlations and Frequencies';
%mkteval;

title2 'Examine Design';
proc print data=randomized; run;
```

Here is the first part of the output. It tells us that the design is orthogonal, that every factor is uncorrelated with every other factor.

Cereal Bars
Examine Correlations and Frequencies
Canonical Correlations Between the Factors
There are 0 Canonical Correlations Greater Than 0.316

	x1	x2	x3	x4	x5	x6
x1	1	0	0	0	0	0
x2	0	1	0	0	0	0
x3	0	0	1	0	0	0
x4	0	0	0	1	0	0
x5	0	0	0	0	1	0
x6	0	0	0	0	0	1

Here is the next part of the output. It tells us that each level occurs equally often, (4 times in the four-level factors and 8 times in the two-level factors), and each pair of levels occurs equally often. The n-way frequencies tell us that every choice set occurs only once in the design−there are no duplicates.

Cereal Bars
Examine Correlations and Frequencies
Summary of Frequencies
There are 0 Canonical Correlations Greater Than 0.316

Frequencies

x1	4 4 4 4
x2	8 8
x3	4 4 4 4
x4	8 8
x5	4 4 4 4
x6	8 8
x1 x2	2 2 2 2 2 2 2 2
x1 x3	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
x1 x4	2 2 2 2 2 2 2 2
x1 x5	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
x1 x6	2 2 2 2 2 2 2 2
x2 x3	2 2 2 2 2 2 2 2
x2 x4	4 4 4 4
x2 x5	2 2 2 2 2 2 2 2
x2 x6	4 4 4 4
x3 x4	2 2 2 2 2 2 2 2
x3 x5	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
x3 x6	2 2 2 2 2 2 2 2

```
x4 x5    2 2 2 2 2 2 2 2
x4 x6    4 4 4 4
x5 x6    2 2 2 2 2 2 2 2
N-Way    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Here is the randomized design. It has 3 four-level factors with levels 1, 2, 3, 4, followed by 3 two-level factors with levels 1, 2, 3. It has 16 rows. The levels and rows are not sorted, so this is in a good form for use.

Cereal Bars
Examine Design

Obs	x1	x2	x3	x4	x5	x6
1	1	1	2	1	2	1
2	4	1	2	2	4	2
3	3	2	2	1	3	2
4	3	1	4	1	4	1
5	2	2	2	2	1	1
6	4	1	1	1	1	1
7	3	2	1	2	2	1
8	1	2	3	2	4	1
9	2	1	4	2	2	2
10	1	2	4	1	1	2
11	4	2	4	2	3	1
12	2	1	3	1	3	1
13	2	2	1	1	4	2
14	1	1	1	2	3	2
15	3	1	3	2	1	2
16	4	2	3	1	2	2

Next, we need to make a choice design from our linear design. We need to specify in a SAS data set the rules for doing this. We need to specify that the brands are Branolicious, Brantopia, Brantasia, and None. We need to specify that the Branolicious Price is made from x1, the Branolicious Count is made from x2, the Brantopia Price is made from x3, the Brantopia Count is made from x4, the Brantasia Price is made from x5, and the Brantasia Count is made from x6. We also need to specify that the none alternative is not made from any of the attributes. We call this data set the *key* to constructing the choice design. The variables in this data set correspond to the attributes in the choice design, and the values correspond to the brands and to the linear design factors. The `%MktKey` macro gives us the linear design factor names that we can cut and paste into this data set. For large designs, this makes it much easier to construct the design key.

```
%mktkey(3 2)
```

Here are the three rows and two columns of the names `x1-x6` for pasting into our key data set.

```
                                    x1    x2

                                    x1    x2
                                    x3    x4
                                    x5    x6
```

Here is the key data set.

```
title2 'Create the Choice Design Key';

data key;
   input
Brand $ 1-12    Price $    Count $; datalines;
Branolicious    x1         x2
Brantopia       x3         x4
Brantasia       x5         x6
None            .          .
;
```

The `%MktRoll` macro uses the linear design and the information in the key data set to make the choice design.

```
title2 'Create Choice Design from Linear Design';
%mktroll( design=randomized, key=key, alt=brand,
          out=cerealdes )

proc print; id set; by set; run;
```

The Brand variable contains literal names, and it is named on the `alt=` option, which designates the alternative name (brand) attribute. The remaining variables contain factor names from the linear data set. Here is the choice design. It contains the variable `Set` along with the variable names and brands from the key data set and also the information from the linear design all stored in the right places.

```
                        Cereal Bars
              Create Choice Design from Linear Design

 Set    Brand           Price    Count

   1    Branolicious      1        1
        Brantopia         2        1
        Brantasia         2        1
        None              .        .

   2    Branolicious      4        1
        Brantopia         2        2
        Brantasia         4        2
        None              .        .
```

3	Branolicious	3	2
	Brantopia	2	1
	Brantasia	3	2
	None	.	.
4	Branolicious	3	1
	Brantopia	4	1
	Brantasia	4	1
	None	.	.
5	Branolicious	2	2
	Brantopia	2	2
	Brantasia	1	1
	None	.	.
6	Branolicious	4	1
	Brantopia	1	1
	Brantasia	1	1
	None	.	.
7	Branolicious	3	2
	Brantopia	1	2
	Brantasia	2	1
	None	.	.
8	Branolicious	1	2
	Brantopia	3	2
	Brantasia	4	1
	None	.	.
9	Branolicious	2	1
	Brantopia	4	2
	Brantasia	2	2
	None	.	.
10	Branolicious	1	2
	Brantopia	4	1
	Brantasia	1	2
	None	.	.
11	Branolicious	4	2
	Brantopia	4	2
	Brantasia	3	1
	None	.	.
12	Branolicious	2	1
	Brantopia	3	1
	Brantasia	3	1
	None	.	.

```
 13    Branolicious      2        2
       Brantopia         1        1
       Brantasia         4        2
       None              .        .

 14    Branolicious      1        1
       Brantopia         1        2
       Brantasia         3        2
       None              .        .

 15    Branolicious      3        1
       Brantopia         3        2
       Brantasia         1        2
       None              .        .

 16    Branolicious      4        2
       Brantopia         3        1
       Brantasia         2        2
       None              .        .
```

These next steps assign formats to the levels and print the choice sets. In the interest of space, only the first four choice sets are printed. The design is stored in a permanent SAS data set so it will still be available at analysis time.

```
title2 'Final Choice Design';

proc format;
   value price  1 = $2.89 2 = $2.99 3 = $3.09 4 = $3.19 . = ' ';
   value count 1 = 'Six Bars' 2 = 'Eight Bars'  . = ' ';
   run;

data sasuser.cerealdes;
   set cerealdes;
   format price price. count count.;
   run;

proc print data=sasuser.cerealdes(obs=16);
   by set;   id set;
   run;
```

Here are the first four choice sets.

```
                          Cereal Bars
                       Final Choice Design

   Set    Brand           Price      Count

    1     Branolicious    $2.89    Six Bars
          Brantopia       $2.99    Six Bars
          Brantasia       $2.99    Six Bars
          None
```

```
2     Branolicious    $3.19    Six Bars
      Brantopia       $2.99    Eight Bars
      Brantasia       $3.19    Eight Bars
      None

3     Branolicious    $3.09    Eight Bars
      Brantopia       $2.99    Six Bars
      Brantasia       $3.09    Eight Bars
      None

4     Branolicious    $3.09    Six Bars
      Brantopia       $3.19    Six Bars
      Brantasia       $3.19    Six Bars
      None
```

This step evaluates the goodness of the design for a choice model using the `%ChoicEff` macro. This macro can also be used to search for efficient choice designs.

```
title2 'Evaluate Design';

%choiceff(model=class(brand price count),  /* model, expand to dummy vars  */
          nalts=4,                         /* number of alternatives      */
          nsets=16,                        /* number of choice sets       */
          beta=zero,                       /* assumed beta vector, Ho: b=0 */
          intiter=0,                       /* no internal iterations just  */
                                           /* evaluate the input design    */
          data=sasuser.cerealdes,          /* the input design to evaluate */
          init=sasuser.cerealdes(keep=set))/* choice set number from design*/
```

Here is the last output table, which is what we are most interested in seeing. We see three parameters for brand (4 alternatives including none minus 1), three for price (4 - 1), one one for count (2 - 1). All are estimable and all have reasonable standard errors. These results look good.

Cereal Bars
Evaluate Design

n	Variable Name	Label	Variance	DF	Standard Error
1	BrandBranolicious	Brand Branolicious	0.94444	1	0.97183
2	BrandBrantasia	Brand Brantasia	0.94444	1	0.97183
3	BrandBrantopia	Brand Brantopia	0.94444	1	0.97183
4	Price_2_89	Price $2.89	0.88889	1	0.94281
5	Price_2_99	Price $2.99	0.88889	1	0.94281
6	Price_3_09	Price $3.09	0.88889	1	0.94281
7	CountSix_Bars	Count Six Bars	0.44444	1	0.66667
				==	
				7	

Next, the questionaire is designed. Here are two sample choice sets.

Branolicious	Brantopia	Brantasia	No Purchase
$2.89	$2.99	$2.99	
Six Bars	Six Bars	Six Bars	

Branolicious	Brantopia	Brantasia	No Purchase
$3.19	$2.99	$3.19	
Six Bars	Eight Bars	Eight Bars	

In practice, data collection is usually much more elaborate than this. It may involve art work or photographs, and the choice sets may be presented and the data may be collected through personal interview or over the web. However the choice sets are presented and the data are collected, the essential ingredients remain the same. Subjects are shown sets of alternatives and are asked to make a choice, then they go on to the next set. Each subject sees all 16 choice sets and chooses one alternative from each. The data for each subject consist of 16 integers in the range 1 to 4 showing which alternative was chosen. The data are collected and entered into a SAS data set. There is one row for each subject containing the number of the chosen alternatives for each choice set.

```
title2 'Read Data';

data results;
   input Subject (r1-r16) (1.);
   datalines;
  1 1331132331312213
  2 3231322131312233
  3 1233332111132233
  4 1211232111313233
  5 1233122111312233
  6 3231323131212313
  7 3231232131332333
  8 3233332131322233
  9 1223332111333233
 10 1332132111233233
 11 1233222211312333
 12 1221332111213233
 13 1231332131133233
 14 3211333211313233
 15 3313332111122233
 16 3321123231331223
 17 3223332231312233
 18 3211223311112233
 19 1232332111132233
 20 1213233111312413
```

```
 21 1333232131212233
 22 3321322111122231
 23 3231122131312133
 24 1232132111311333
 25 3113332431213233
 26 3213132141331233
 27 3221132111312233
 28 3222333131313231
 29 1221332131312231
 30 3233332111212233
 31 1221332111342233
 32 2233232111111211
 33 2332332131211231
 34 2221132211312411
 35 1232233111332233
 36 1231333131322333
 37 1231332111331333
 38 1223132211233331
 39 1321232131211231
 40 1223132331321233
;
```

The %MktMerge macro merges the data and the design and creates the dependent variable.

```
title2 'Merge Data and Design';

%mktmerge(design=sasuser.cerealdes, /* input design                        */
          data=results,             /* input data set                       */
          out=res2,                 /* output data set with design and data */
          nsets=16,                 /* number of choice sets                */
          nalts=4,                  /* number of alternatives               */
          setvars=r1-r16)           /* variables with the chosen alt nums   */
```

This data set has one row for each alternative of each choice set for each subject (in this case, there are $4 \times 16 \times 40 = 2560$ rows). This step prints the first four choice sets for the first subject.

```
title2 'Design and Data Both';

proc print data=res2(obs=16);
   by set subject;   id set subject;
   run;
```

Here are the first four choice sets for the first subject.

Cereal Bars
Design and Data Both

Set	Subject	Brand	Price	Count	c
1	1	Branolicious	$2.89	Six Bars	1
		Brantopia	$2.99	Six Bars	2
		Brantasia	$2.99	Six Bars	2
		None			2

```
2        1       Branolicious    $3.19    Six Bars      2
                 Brantopia       $2.99    Eight Bars    2
                 Brantasia       $3.19    Eight Bars    1
                 None                                   2

3        1       Branolicious    $3.09    Eight Bars    2
                 Brantopia       $2.99    Six Bars      2
                 Brantasia       $3.09    Eight Bars    1
                 None                                   2

4        1       Branolicious    $3.09    Six Bars      1
                 Brantopia       $3.19    Six Bars      2
                 Brantasia       $3.19    Six Bars      2
                 None                                   2
```

The dependent variable is `c`. A 1 in `c` indicates first choice, and a 2 indicates the alternatives that were not chosen (second or subsequent choice).

This next step codes the design variables for analysis and prints the coded results for the first subject for the first four choice sets. All of the independent variables are names in the `class` specification. PROC TRANSREG options are explained in detail in throughout the other examples, particularly on page 173.

```
title2 'Code the Independent Variables';

proc transreg design norestoremissing data=res2;
   model class(brand price count);
   id subject set c;
   output out=coded(drop=_type_ _name_ intercept) lprefix=0;
   run;
proc print data=coded(obs=16) label;
   title3 'ID Information and the Dependent Variable';
   format price price. count count.;
   var Brand Price Count Subject Set c;
   by set subject;   id set subject;
   run;
proc print data=coded(obs=16) label;
   title3 'ID Information and the Coding of Brand';
   format price price. count count.;
   var brandbranolicious brandbrantasia brandbrantopia brand;
   by set subject;   id set subject;
   run;
proc print data=coded(obs=16) label;
   title3 'ID Information and the Coding of Price and Count';
   format price price. count count.;
   var Price_2_89 Price_2_99 Price_3_09 CountSix_Bars Price Count;
   by set subject;   id set subject;
   run;
```

Here is the coded design for the first four choice sets, shown in three panels.

Cereal Bars
Code the Independent Variables
ID Information and the Dependent Variable

Set	Subject	Brand	Price	Count	Subject	Set	c
1	1	Branolicious	$2.89	Six Bars	1	1	1
		Brantopia	$2.99	Six Bars	1	1	2
		Brantasia	$2.99	Six Bars	1	1	2
		None			1	1	2
2	1	Branolicious	$3.19	Six Bars	1	2	2
		Brantopia	$2.99	Eight Bars	1	2	2
		Brantasia	$3.19	Eight Bars	1	2	1
		None			1	2	2
3	1	Branolicious	$3.09	Eight Bars	1	3	2
		Brantopia	$2.99	Six Bars	1	3	2
		Brantasia	$3.09	Eight Bars	1	3	1
		None			1	3	2
4	1	Branolicious	$3.09	Six Bars	1	4	1
		Brantopia	$3.19	Six Bars	1	4	2
		Brantasia	$3.19	Six Bars	1	4	2
		None			1	4	2

Cereal Bars
Code the Independent Variables
ID Information and the Coding of Brand

Set	Subject	Branolicious	Brantasia	Brantopia	Brand
1	1	1	0	0	Branolicious
		0	0	1	Brantopia
		0	1	0	Brantasia
		0	0	0	None
2	1	1	0	0	Branolicious
		0	0	1	Brantopia
		0	1	0	Brantasia
		0	0	0	None
3	1	1	0	0	Branolicious
		0	0	1	Brantopia
		0	1	0	Brantasia
		0	0	0	None
4	1	1	0	0	Branolicious
		0	0	1	Brantopia
		0	1	0	Brantasia
		0	0	0	None

Cereal Bars
Code the Independent Variables
ID Information and the Coding of Price and Count

Set	Subject	$2.89	$2.99	$3.09	Six Bars	Price	Count
1	1	1	0	0	1	$2.89	Six Bars
		0	1	0	1	$2.99	Six Bars
		0	1	0	1	$2.99	Six Bars
		0	0	0	0		
2	1	0	0	0	1	$3.19	Six Bars
		0	1	0	0	$2.99	Eight Bars
		0	0	0	0	$3.19	Eight Bars
		0	0	0	0		
3	1	0	0	1	0	$3.09	Eight Bars
		0	1	0	1	$2.99	Six Bars
		0	0	1	0	$3.09	Eight Bars
		0	0	0	0		
4	1	0	0	1	1	$3.09	Six Bars
		0	0	0	1	$3.19	Six Bars
		0	0	0	1	$3.19	Six Bars
		0	0	0	0		

This code fits the choice model. Notice that we use the `%PhChoice` macro to customize the output from PROC PHREG so it looks more like discrete choice output and less like survival analysis output. The choice model is a special case of a survival-analysis model. Before the model equals sign, `c` indicates the chosen alternative and also the alternatives that were not chosen, those with values of 2 or greater. After the equal sign is a macro variable that PROC TRANSREG creates with the list of coded variables. Each subject and set combination or stratum makes a contribution to the likelihood function.

```
%phchoice(  on  )

title2 'Multinomial Logit Discrete Choice Model';

proc phreg data=coded brief;
   model c*c(2) = &_trgind / ties=breslow;
   strata subject set;
   run;

%phchoice(  off  )
```

Here are the results.

Cereal Bars
Multinomial Logit Discrete Choice Model

The PHREG Procedure

Model Information

Data Set	WORK.CODED
Dependent Variable	c
Censoring Variable	c
Censoring Value(s)	2
Ties Handling	BRESLOW

Number of Observations Read	2560
Number of Observations Used	2560

Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Pattern	Number of Choices	Number of Alternatives	Chosen Alternatives	Not Chosen
1	640	4	1	3

Convergence Status

Convergence criterion (GCONV=1E-8) satisfied.

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	1774.457	1142.630
AIC	1774.457	1156.630
SBC	1774.457	1187.860

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	631.8271	7	<.0001
Score	518.1014	7	<.0001
Wald	275.0965	7	<.0001

Cereal Bars
Multinomial Logit Discrete Choice Model

The PHREG Procedure

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Branolicious	1	2.64506	0.47268	31.3142	<.0001
Brantasia	1	2.94600	0.47200	38.9571	<.0001
Brantopia	1	2.44876	0.47416	26.6706	<.0001
$2.89	1	2.69907	0.20307	176.6557	<.0001
$2.99	1	1.72036	0.17746	93.9845	<.0001
$3.09	1	0.76407	0.17437	19.2008	<.0001
Six Bars	1	-0.54645	0.11899	21.0912	<.0001

Notice near the top of the output that there was one pattern of results. There were 640 times (16 choice sets times 40 people) that four alternatives were presented and one was chosen. This table provides a check on the data entry. Usually, the number of alternatives is the same in all choice sets, as it is here. Multiple patterns would mean a data entry error had occured. The "Multinomial Logit Parameter Estimates" table is of primary interest. All of the part-worth utilities (parameter estimates) are significant, and the clearest pattern in the results is that the lower prices have the highest utility.

These are the basic steps in designing, processing, and analyzing a choice experiment. Pages 141 through 465 have many more examples, much greater detail, and show how to use other tools.

Optimal Generic Choice Designs

In some situations, particularly for certain generic choice experiments, we can make optimal choice designs under the assumption that $\boldsymbol{\beta} = 0$. A generic choice experiment is one that does not have any brands. The alternatives are simply bundles of attributes. For example, a manufacturer of any electronic product may construct a choice study with potential variations on a new product to see which attributes are the most important drivers of choice. Consider for example a study that involves 4 two-level factors and four choice sets, each with two alternatives. Here is an the optimal generic choice design along with a three fractional-factorial designs. The first fractional-factorial design consists of 3 two-level factors in 4 runs and an intercept. The second fractional-factorial design consists of 3 two-level factors in 4 runs and a "shifted intercept," a column of twos instead of the customary column of ones. The third fractional-factorial design consists of 1 four-level factor and 4 two-level factors in eight runs.

Optimal Generic Choice Design

1	1	1	1
2	2	2	2
1	1	2	2
2	2	1	1
1	2	2	1
2	1	1	2
1	2	1	2
2	1	2	1

Fractional Factorial 2^3 in 4 Runs With Intercept

1	1	1	1
1	1	2	2
1	2	2	1
1	2	1	2

Shifted Fractional Factorial

2	2	2	2
2	2	1	1
2	1	1	2
2	1	2	1

Fractional Factorial $4^1 2^4$ in 8 Runs

1	1	1	1	1
2	1	1	2	2
3	1	2	2	1
4	1	2	1	2
1	2	2	2	2
2	2	2	1	1
3	2	1	1	2
4	2	1	2	1

The first fractional-factorial design exactly matches the two-level factors in the first half of the third fractional-factorial design, and the second fractional-factorial design exactly matches the two-level factors in the second half of the third fractional-factorial design. Sorting the third fractional-factorial design on the four-level factor and using it as the choice set number yields the optimal generic choice design.

The optimal generic choice design can be constructed by creating a fractional-factorial design with an intercept and using it to make the first alternative of each choice set. The second alternative is made from the first by shifting or cycling through the levels (changing 1 to 2 and 2 to 1). The first alternative is shown in the fractional-factorial table, and the second alternative is shown in shifted fractional-factorial table. The plan for the second alternative is a different fractional-factorial plan. Alternatively and equivalently, this design can be made from the orthogonal array $4^1 2^4$ in 8 runs by using the four-level factor as the choice set number. Note that the optimal generic choice design never shows two alternatives with the same levels of any factor. For this reason, some researchers do not use them and consider this class of designs to be more of academic and combinatorial interest than of practical significance.

Here is an optimal generic choice design with 9 three-level attributes, with three alternatives, and nine choice sets, each in a separate box. It is made from the orthogonal design $3^9 9^1$ in 27 runs by using the nine-level factor as the choice set number. Notice that each alternative is made from the previous alternative by adding one to the previous level, mod 3. Similarly, the first alternative is made from the third alternative by adding one to the previous level, mod 3.

1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3

1	2	3	3	1	2	2	3	1
2	3	1	1	2	3	3	1	2
3	1	2	2	3	1	1	2	3

1	1	1	2	2	2	3	3	3
2	2	2	3	3	3	1	1	1
3	3	3	1	1	1	2	2	2

1	3	2	1	3	2	1	3	2
2	1	3	2	1	3	2	1	3
3	2	1	3	2	1	3	2	1

1	1	1	3	3	3	2	2	2
2	2	2	1	1	1	3	3	3
3	3	3	2	2	2	1	1	1

1	3	2	2	1	3	3	2	1
2	1	3	3	2	1	1	3	2
3	2	1	1	3	2	2	1	3

1	2	3	1	2	3	1	2	3
2	3	1	2	3	1	2	3	1
3	1	2	3	1	2	3	1	2

1	3	2	3	2	1	2	1	3
2	1	3	1	3	2	3	2	1
3	2	1	2	1	3	1	3	2

1	2	3	2	3	1	3	1	2
2	3	1	3	1	2	1	2	3
3	1	2	1	2	3	2	3	1

Here is an optimal generic choice design with 8 four-level attributes, with four alternatives, and eight choice sets, each in a separate box. It is made from the fractional-factorial design $4^8 8$ in 32 runs by using the eight-level factor as the choice set number. Notice that every attribute has all four levels in each factor. With four-level factors, the rules that are used to make orthogonal arrays are more complicated than the mod 3 addition that is used with three-level factors, so you do not get the same pattern of shifted results that we saw previously.

1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4

1	3	2	1	4	2	3	4
2	4	1	2	3	1	4	3
3	1	4	3	2	4	1	2
4	2	3	4	1	3	2	1

1	1	3	4	2	2	4	3
2	2	4	3	1	1	3	4
3	3	1	2	4	4	2	1
4	4	2	1	3	3	1	2

1	3	4	4	3	1	2	2
2	4	3	3	4	2	1	1
3	1	2	2	1	3	4	4
4	2	1	1	2	4	3	3

1	2	2	3	3	4	4	1
2	1	1	4	4	3	3	2
3	4	4	1	1	2	2	3
4	3	3	2	2	1	1	4

1	4	1	3	2	3	2	4
2	3	2	4	1	4	1	3
3	2	3	1	4	1	4	2
4	1	4	2	3	2	3	1

1	2	4	2	4	3	1	3
2	1	3	1	3	4	2	4
3	4	2	4	2	1	3	1
4	3	1	3	1	2	4	2

1	4	3	2	1	4	3	2
2	3	4	1	2	3	4	1
3	2	1	4	3	2	1	4
4	1	2	3	4	1	2	3

If you need a generic choice design and you do not have the level of symmetry shown in these examples (all m-level factors with m alternatives) then you can use the `%ChoicEff` macro to find an efficient generic design using the methods shown in the chair example on page 363.

In general, optimal generic designs can be constructed for experiments with p choice sets and m-level factors with m alternatives when there is an orthogonal array $p^1 m^q$ in $p \times m$ runs where $q \leq p$. We can process the design catalog from the `%MktOrth` macro to find these.

```
%mktorth(maxn=100, options=parent);

data x;
   set mktdeslev;
   array x[50];
   c = 0; one = 0; q = 0;
   do i = 1 to 50;
      c + (x[i] > 0); /* how many differing numbers of levels */
      if x[i] > 1 then do; m = i; q = x[i]; end; /* m^q */
      if x[i] = 1 then do; one + 1; p = i;  end; /* p^1 */
      end;
   if c = 2 and one = 1 and p >= q and q > 2 and p * m = n;
   design = compbl(left(design));
   run;

proc print; var n design; run;
```

Here are a few of the smaller designs that work.

Obs	n	Design
1	8	2 ** 4 4 ** 1
2	16	2 ** 8 8 ** 1
3	18	3 ** 6 6 ** 1
4	24	2 ** 12 12 ** 1
5	27	3 ** 9 9 ** 1
6	32	2 ** 16 16 ** 1
7	32	4 ** 8 8 ** 1
8	36	3 ** 12 12 ** 1
9	40	2 ** 20 20 ** 1
10	45	3 ** 9 15 ** 1
11	48	2 ** 24 24 ** 1
12	48	4 ** 12 12 ** 1
13	50	5 ** 10 10 ** 1
14	54	3 ** 18 18 ** 1
15	56	2 ** 28 28 ** 1
16	63	3 ** 12 21 ** 1
17	64	2 ** 32 32 ** 1
18	64	4 ** 16 16 ** 1
19	72	2 ** 36 36 ** 1
20	72	3 ** 24 24 ** 1
21	75	5 ** 8 15 ** 1
22	80	2 ** 40 40 ** 1
23	80	4 ** 10 20 ** 1
24	81	3 ** 27 27 ** 1
25	88	2 ** 44 44 ** 1

26	90	3 ** 30 30 ** 1
27	96	2 ** 48 48 ** 1
28	98	7 ** 14 14 ** 1
29	99	3 ** 13 33 ** 1
30	100	5 ** 20 20 ** 1

For a specification of p, q, and m, assuming the orthogonal array p^1m^q in pm runs exists, this code makes an optimal generic choice design. The `%ChoicEff` macro evaluates the results.

```
%let p = 6;
%let m = 3;
%let q = &p;

%mktex(&p &m ** &q, n=&p * &m)
%mktlab(data=design, vars=Set x1-x&q)

proc print; id set; by set; run;

%choiceff(data=final, init=final(keep=set), model=class(x1-x&q),
          nsets=&p, nalts=&m, beta=zero)
```

Here is an example with $m = 3$ and $p = q = 6$ choice sets.

Set	x1	x2	x3	x4	x5	x6
1	1	1	1	1	1	1
	2	2	2	2	2	2
	3	3	3	3	3	3
2	1	1	2	2	3	3
	2	2	3	3	1	1
	3	3	1	1	2	2
3	1	2	1	3	3	2
	2	3	2	1	1	3
	3	1	3	2	2	1
4	1	2	3	1	2	3
	2	3	1	2	3	1
	3	1	2	3	1	2
5	1	3	2	3	2	1
	2	1	3	1	3	2
	3	2	1	2	1	3
6	1	3	3	2	1	2
	2	1	1	3	2	3
	3	2	2	1	3	1

Here are the parameters names and their variances under the null hypothesis that $\boldsymbol{\beta} = \mathbf{0}$.

n	Variable Name	Label	Variance	DF	Standard Error
1	x11	x1 1	1	1	1
2	x12	x1 2	1	1	1
3	x21	x2 1	1	1	1
4	x22	x2 2	1	1	1
5	x31	x3 1	1	1	1
6	x32	x3 2	1	1	1
7	x41	x4 1	1	1	1
8	x42	x4 2	1	1	1
9	x51	x5 1	1	1	1
10	x52	x5 2	1	1	1
11	x61	x6 1	1	1	1
12	x62	x6 2	1	1	1
				==	
				12	

The rest of this section is optional and can be skipped by readers not interested in the combinatorial details of this construction method. The next section starts on page 96. An orthogonal array $p^1 m^q$ in $p \times m$ runs is made by developing a difference scheme (Wang and Wu 1991). A *difference scheme* is a matrix that is a "building block" used in the construction of many orthogonal arrays. It is called a difference scheme because if you subtract any two columns, all differences occur equally often. Note however, that the meaning of the term "subract" in this context is quite different from its customary meaning in the real number system. Here, subtraction is in a Galois or abelian field. Explaining this fully is beyond the scope of this discussion, but we will provide an example. Here for example are the addition, subtraction, multiplication, and inversion tables that are used in a Galois field of order 5 (GF(5)). These tables are used when constructing factors with five levels (0 1 2 3 4).

Addition

	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

Subtraction

	0	1	2	3	4
0	0	4	3	2	1
1	1	0	4	3	2
2	2	1	0	4	3
3	3	2	1	0	4
4	4	3	2	1	0

Multiplication

	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

Inverse

0	
1	1
2	3
3	2
4	4

The rules for addition and multiplication follow the rules for integer arithmetic mod 5, so for example, $4 + 4 \bmod 5 = 8 \bmod 5 = 3$ and $4 \times 4 \bmod 5 = 16 \bmod 5 = 1$. The results can also be seen by accessing the row 4, column 4 entries of the addition and multiplication tables. The rules for subtraction can easily be derived from the rules for addition, and the rules for inversion can easily be derived from the rules for multiplication. For example, since 4 + 3 = 2 in GF(5), then 4 = 2 - 3, and since $3 \times 2 = 1$, then 2 is the inverse of 3. Note that in many cases, the rules for field arithmetic are not this simple. In some cases, like when the order of the field is a power of a prime (4, 8, 9 ...) or a composite number that contains a power of a prime (12, 18, ...), the rules are much more complicated, and modulo m arithmetic does not work.

In GF(5), the multiplication table is a 5×5 difference scheme, **D**. You can verify that if you subract every column from every other column, the five elements of the field (0 1 2 3 4) all occur exactly once

in all of the difference vectors. An orthogonal array is made by constructing

D + 0 (0,1,2,3,4)
D + 1 (0,1,2,3,4)
D + 2 (0,1,2,3,4)
D + 3 (0,1,2,3,4)
D + 4 (0,1,2,3,4).

For each new block, the difference scheme is shifted, adding 1 each time to the previous matrix. An additional factor can be added which consists of the levels (0, 1, ..., $p-1$). Here is the orthogonal array on the left and the generic choice design made from sorting this orthogonal array on the right.

Orthogonal Array

0	0	0	0	0	0
0	1	2	3	4	1
0	2	4	1	3	2
0	3	1	4	2	3
0	4	3	2	1	4
1	1	1	1	1	0
1	2	3	4	0	1
1	3	0	2	4	2
1	4	2	0	3	3
1	0	4	3	2	4
2	2	2	2	2	0
2	3	4	0	1	1
2	4	1	3	0	2
2	0	3	1	4	3
2	1	0	4	3	4
3	3	3	3	3	0
3	4	0	1	2	1
3	0	2	4	1	2
3	1	4	2	0	3
3	2	1	0	4	4
4	4	4	4	4	0
4	0	1	2	3	1
4	1	3	0	2	2
4	2	0	3	1	3
4	3	2	1	0	4

Generic Choice Design

Set	Factors				
0	0	0	0	0	0
0	1	1	1	1	1
0	2	2	2	2	2
0	3	3	3	3	3
0	4	4	4	4	4
1	0	1	2	3	4
1	1	2	3	4	0
1	2	3	4	0	1
1	3	4	0	1	2
1	4	0	1	2	3
2	0	2	4	1	3
2	1	3	0	2	4
2	2	4	1	3	0
2	3	0	2	4	1
2	4	1	3	0	2
3	0	3	1	4	2
3	1	4	2	0	3
3	2	0	3	1	4
3	3	1	4	2	0
3	4	2	0	3	1
4	0	4	3	2	1
4	1	0	4	3	2
4	2	1	0	4	3
4	3	2	1	0	4
4	4	3	2	1	0

This is the same orthogonal array that `%MktEx` produces except that by default, `%MktEx` uses one-based integers instead of a zero base. For a generic choice design, the difference scheme provides levels for the first alternative, and all other alternatives are made from the previous alternative by adding 1 in the appropriate field.

Conclusions

This chapter introduced some choice design terminology and ideas without going into great detail on how to make designs and process data for analysis. The information in this chapter should provide a good foundation for all of the detailed examples in the discrete choice chapter.

Choice Design Glossary

Choice modeling, like all other areas, has its own vocabulary. This section defines some of those terms. These terms are used and also defined throughout the discrete choice chapter (pages 141−465).

allocation study - An allocation study is a choice study where multiple, not single choices are made. For example, in prescription drug marketing, physicians are asked questions like "For the next ten prescriptions you write for a particular condition, how many would you write for each of these drugs?"

alternative-specific attribute - An alternative-specific attribute is one that is expected to interact with brand. If you expect utility to change in different ways for the different brands, then the attribute is alternative-specific. Otherwise, it is generic. In the analysis, there is a set of alternative-specific attribute parameters for each alternative.

alternative - An alternative is one of the options available to be chosen in a choice set. An alternative might correspond to a particular brand in a branded study or just a bundle of attributes in a generic study.

attribute - An attribute is one of the characteristics of an alternative. Common attributes include price, size, and a variety of other product-specific factors.

availability cross effects - A design may have a varying number of alternatives. When not all alternatives are available in every choice set, availability cross effects, may be of interest. These capture the effects of the presence/absence of one brand on the utility of another.

blocking - Large choice designs need to be broken into blocks. Subjects will just see a subset of the full design. How many blocks depends on the number of choice sets and the complexity of the choice task.

branded design - A branded choice design has one factor that consists of a brand name or other alternative label. The vacation examples on pages 184-260 are examples of branded designs even though the labels, destinations, and not brands. The examples starting on pages 156, 283, and 261 use branded designs and actual brand names.

choice design - A choice design has one column for every different product attribute and one row for every alternative of every choice set. In some cases, different alternatives will have different attributes and different choice sets may have differing numbers of alternatives. See pages 48 and 60−61.

choice set - A choice set consists of two or more alternatives. Subjects see one or more choice sets and choose one alternative from each set.

cross effects - A cross effect represents the effect of one alternative on the utility of another alternative. When the IIA assumption holds, all cross effects will be zero.

generic attribute - A generic attribute is one that is not expected to interact with brand. If you expect utility to change as a function of the levels of the attribute in the same way for every brand, then

the attribute is generic. In contrast, if you expect utility to change in different ways for the different brands, then the attribute is alternative-specific. All attributes in generic designs are generic. In the analysis, there is one set of parameters for generic attributes, regardless of the number of alternatives.

generic design or **generic model** - A generic design is unbranded. The alternatives are simply bundles of attributes. Each alternative may be for example a cell phone or computer all made by the same manufacturer.

IIA - The independence of irrelevant alternatives or IIA property states that utility only depends on an alternative's own attributes. IIA means the odds of choosing alternative c_i over c_j do not depend on the other alternatives in the choice set. Departures from IIA exist when certain subsets of brands are in more direct competition and tend to draw a disproportionate amount of share from each other than from other members in the category.

linear design - A linear design is a factorial design. In the choice model context, it contains one row for each choice set and one column for every attribute of every alternative. The columns are grouped, the first group contains every attribute for the first alternative, ..., and the *jth* group contains every attribute for the *jth* alternative. The linear design is used to construct a choice design. Each of the m blocks for the m alternatives is moved below the preceding block creating a choice design with m times as many rows as previously and approximately $1/m$ times as many columns. See pages 60−61.

mother logit model - The mother logit model is a model with cross effects that can be used to test for violations of IIA.

Efficient Experimental Design with Marketing Research Applications

Warren F. Kuhfeld

Randall D. Tobias

Mark Garratt

Abstract

We suggest using *D*-efficient experimental designs for conjoint and discrete-choice studies, and discuss orthogonal arrays, nonorthogonal designs, relative efficiency, and nonorthogonal design algorithms. We construct designs for a choice study with asymmetry and interactions and for a conjoint study with blocks and aggregate interactions.*

Introduction

The design of experiments is a fundamental part of marketing research. Experimental designs are required in widely used techniques such as preference-based conjoint analysis and discrete-choice studies (e.g., Carmone and Green 1981; Elrod, Louviere, and Davey 1992; Green and Wind 1975; Huber, et al. 1993; Lazari and Anderson 1994; Louviere 1991; Louviere and Woodworth 1983; Wittink and Cattin 1989). Ideally, marketing researchers prefer *orthogonal* designs. When a linear model is fit with an orthogonal design, the parameter estimates are uncorrelated, which means each estimate is independent of the other terms in the model. More importantly, orthogonality usually implies that the coefficients will have minimum variance, though we discuss exceptions to this rule. For these reasons, orthogonal designs are usually quite good. However, for many practical problems, orthogonal designs are simply not available. In those situations, *nonorthogonal* designs must be used.

*This chapter is a revision of a paper that appeared in *Journal of Marketing Research*, November, 1994, pages 545−557. Warren F. Kuhfeld is now Manager, Multivariate Models R&D, SAS. Randall D. Tobias is now Manager, Linear Models R&D, SAS. Mark Garratt was Vice President, Conway | Milliken & Associates when this paper was first published in 1994 and is now with Miller Brewing Company. The authors thank Jordan Louviere, *JMR* editor Barton Weitz, and three anonymous reviewers for their helpful comments on earlier versions of this article. Thanks to Michael Ford for the idea for the second example. The *JMR* article was based on a presentation given to the AMA Advanced Research Techniques Forum, June 14, 1993, Monterey CA.

Our primary message when this paper was published in 1994 was that marketing researchers should use *D*-efficient experimental designs. This message remains as strong as ever, but today, we have much better tools for accomplishing this than we had in 1994. Most of the revisions of the original paper are due to improvements in the tools. Our new design tool, the `%MktEx` SAS macro, is easier to use than our old tools, and it usually makes better designs. Copies of this chapter (TS-722D) and all of the macros are available on the web http://support.sas.com/techsup/tnote/tnote_stat.html#market.

Orthogonal designs are available for only a relatively small number of very specific problems. They may not be available when some combinations of factor levels are infeasible, a nonstandard number of *runs* (factor level combinations or hypothetical products) is desired, or a nonstandard model is being used, such as a model with interaction or polynomial effects. Consider the problem of designing a discrete choice study in which there are alternative specific factors, different numbers of levels within each factor, and interactions within each alternative. Orthogonal designs are not readily available for this situation, particularly when the number of runs must be limited. When an orthogonal design is not available, an alternative must be chosen−the experiment can be modified to fit some known orthogonal design, which is undesirable for obvious reasons, or a known design can be modified to fit the experiment, which may be difficult and inefficient.

Our primary purpose is to explore a third alternative, the use of optimal (or nearly optimal) designs. Such designs are typically nonorthogonal; however they are efficient in the sense that the variances and covariances of the parameter estimates are minimized. Furthermore, they are always available, even for nonstandard situations. Finding these designs usually requires the aid of a computer, but we want to emphasize that we are not advocating a black-box approach to designing experiments. Computerized design algorithms do not supplant traditional design-creation skills. Our examples show that our best designs were usually found when we used our human design skills to guide the computerized search.

First, we will summarize our main points; next, we will review some fundamentals of the design of experiments; then we will discuss computer-generated designs, a discrete-choice example, and a conjoint analysis example.

Summary of Main Points. Our goal is to explain the benefits of using computer-generated designs in marketing research. Our main points follow:

1. The goodness of an experimental design (*efficiency*) can be quantified as a function of the variances and covariances of the parameter estimates. Efficiency increases as the variances decrease. Designs should not be thought of in terms of the dichotomy between orthogonal versus nonorthogonal but rather as varying along the continuous attribute of efficiency. Some orthogonal designs are less efficient than other (orthogonal and nonorthogonal) alternatives.

2. Orthogonality is not the primary goal in design creation. It is a secondary goal, associated with the primary goal of minimizing the variances of the parameter estimates. Degree of orthogonality is an important consideration, but other factors should not be ignored.

3. For complex, nonstandard situations, computerized searches provide the only practical method of design generation for all but the most sophisticated of human designers. These situations do not have to be avoided just because it is extremely difficult to generate a good design manually.

4. The best approach to design creation is to use the computer as a tool along with traditional design skills, not as a substitute for thinking about the problem.

Background and Assumptions. We present an overview of the theory of efficient experimental design, developed for the general linear model. This topic is well known to specialists in statistical experimentation, though it is not typically taught in design classes. Then we will suggest ways in which this theory can be applied to marketing research problems.

Certain assumptions must be made before applying ordinary general linear model theory to problems in marketing research. The usual goals in linear modeling are to estimate parameters and test hypotheses about those parameters. Typically, independence and normality are assumed. In conjoint analysis, each subject rates all products and separate ordinary-least-squares analyses are run for each subject. This is not a standard general linear model; in particular, observations are not independent and normality cannot be assumed. Discrete choice models, which are nonlinear, are even further removed from the general linear model.

Marketing researchers have always made the critical assumption that designs that are good for general linear models are also good for conjoint analysis and discrete choice. We also make this assumption. Specifically, we assume the following:

1. Market share estimates computed from a conjoint analysis model using a more efficient design will be better than estimates using a less efficient design. That is, more efficient designs mean better estimates of the part-worth utilities, which lead to better estimates of product utility and market share.
2. An efficient design for a linear model is a good design for the multinomial logit (MNL) model used in discrete choice studies.

Investigating these standard assumptions is beyond the scope of this article. However, they are supported by Carson and colleagues (1994), our experiences in consumer product goods, and limited simulation results. Much more research is needed on this topic, particularly in the area of discrete choice.

Design of Experiments

Orthogonal Experimental Designs. An experimental design is a plan for running an experiment. The *factors* of an experimental design are variables that have two or more fixed values, or *levels*. Experiments are performed to study the effects of the factor levels on the dependent variable. In a conjoint or discrete-choice study, the factors are the attributes of the hypothetical products or services, and the response is preference or choice.

A simple experimental design is the *full-factorial design*, which consists of all possible combinations of the levels of the factors. For example, with five factors, two at two levels and three at three levels (denoted $2^2 3^3$), there are 108 possible combinations. In a full-factorial design, all main effects, two-way interactions, and higher-order interactions are estimable and uncorrelated. The problem with a full-factorial design is that, for most practical situations, it is too cost-prohibitive and tedious to have subjects rate all possible combinations. For this reason, researchers often use *fractional-factorial designs*, which have fewer runs than full-factorial designs. The price of having fewer runs is that some effects become confounded. Two effects are *confounded* or *aliased* when they are not distinguishable from each other.

A special type of fractional-factorial design is the *orthogonal array*, in which all estimable effects are uncorrelated. Orthogonal arrays are categorized by their *resolution*. The resolution identifies which effects, possibly including interactions, are estimable. For example, for resolution III designs, all main effects are estimable free of each other, but some of them are confounded with two-factor interactions.

For resolution V designs, all main effects and two-factor interactions are estimable free of each other. Higher resolutions require larger designs. Orthogonal arrays come in specific numbers of runs (e.g., 16, 18, 20, 24, 27, 28) for specific numbers of factors with specific numbers of levels.

Resolution III orthogonal arrays are frequently used in marketing research. The term "orthogonal array," as it is sometimes used in practice, is imprecise. It correctly refers to designs that are both orthogonal and balanced, and hence optimal. It is also imprecisely used to refer to designs that are orthogonal but not balanced, and hence potentially nonoptimal. A design is *balanced* when each level occurs equally often within each factor, which means the intercept is orthogonal to each effect. Imbalance is a generalized form of nonorthogonality, which increases the variances of the parameter estimates.

Design Efficiency. Efficiencies are measures of design goodness. Common measures of the efficiency of an $(N_D \times p)$ design matrix $\mathbf{X}$ are based on the *information matrix* $\mathbf{X}'\mathbf{X}$. The variance-covariance matrix of the vector of parameter estimates $\boldsymbol{\beta}$ in a least-squares analysis is proportional to $(\mathbf{X}'\mathbf{X})^{-1}$. An efficient design will have a "small" variance matrix, and the eigenvalues of $(\mathbf{X}'\mathbf{X})^{-1}$ provide measures of its "size." Two common efficiency measures are based on the idea of "average eigenvalue" or "average variance." *A-efficiency* is a function of the arithmetic mean of the eigenvalues, which is given by trace $((\mathbf{X}'\mathbf{X})^{-1})/p$. *D-efficiency* is a function of the geometric mean of the eigenvalues, which is given by $|(\mathbf{X}'\mathbf{X})^{-1}|^{1/p}$. A third common efficiency measure, *G-efficiency*, is based on σ_M, the maximum standard error for prediction over the candidate set. All three of these criteria are convex functions of the eigenvalues of $(\mathbf{X}'\mathbf{X})^{-1}$ and hence are usually highly correlated.

For all three criteria, if a balanced and orthogonal design exists, then it has optimum efficiency; conversely, the more efficient a design is, the more it tends toward balance and orthogonality. A design is balanced and orthogonal when $(\mathbf{X}'\mathbf{X})^{-1}$ is diagonal (for a suitably coded $\mathbf{X}$, see page 64). A design is orthogonal when the submatrix of $(\mathbf{X}'\mathbf{X})^{-1}$, excluding the row and column for the intercept, is diagonal; there may be off-diagonal nonzeros for the intercept. A design is balanced when all off-diagonal elements in the intercept row and column are zero.

These measures of efficiency can be scaled to range from 0 to 100 (for a suitably coded $\mathbf{X}$):

$$
\begin{aligned}
A\text{-efficiency} &= 100 \times \frac{1}{N_D \text{ trace}\left((\mathbf{X}'\mathbf{X})^{-1}\right)/p} \\
D\text{-efficiency} &= 100 \times \frac{1}{N_D \left|(\mathbf{X}'\mathbf{X})^{-1}\right|^{1/p}} \\
G\text{-efficiency} &= 100 \times \frac{\sqrt{p/N_D}}{\sigma_M}
\end{aligned}
$$

These efficiencies measure the goodness of the design relative to hypothetical orthogonal designs that may be far from possible, so they are not useful as absolute measures of design efficiency. Instead, they should be used relatively, to compare one design with another for the same situation. Efficiencies that are not near 100 may be perfectly satisfactory.

Figure 1 shows an optimal design in four runs for a simple example with two factors, using interval-measure scales for both. There are three candidate levels for each factor. The full-factorial design is shown by the nine asterisks, with circles around the optimal four design points. As this example shows,

Figure 1
Candidate Set and Optimal Design

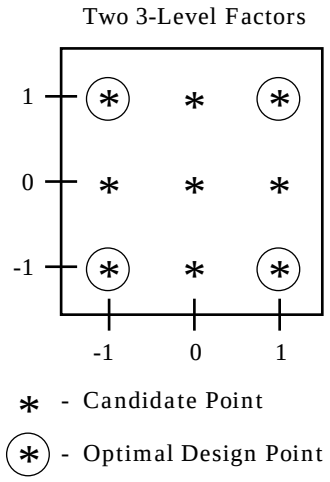


Table 1
Full-Factorial Design

Information Matrix

	Int	X1	X2	X3	-	X4	-	X5	-
Int	108	0	0	0	0	0	0	0	0
X1	0	108	0	0	0	0	0	0	0
X2	0	0	108	0	0	0	0	0	0
X3	0	0	0	108	0	0	0	0	0
-	0	0	0	0	108	0	0	0	0
X4	0	0	0	0	0	108	0	0	0
-	0	0	0	0	0	0	108	0	0
X5	0	0	0	0	0	0	0	108	0
-	0	0	0	0	0	0	0	0	108

100.0000	*D*-efficiency
100.0000	*A*-efficiency
100.0000	*G*-efficiency

efficiency tends to emphasize the corners of the design space. Interestingly, nine different sets of four points form orthogonal designs−every set of four that forms a rectangle or square. Only one of these orthogonal designs is optimal, the one in which the points are spread out as far as possible.

Computer-Generated Design Algorithms. When a suitable orthogonal design does not exist, computer-generated nonorthogonal designs can be used instead. Various algorithms exist for selecting a good set of *design points* from a set of *candidate points*. The candidate points consist of all of the factor-level combinations that can potentially be included in the design−for example the nine points in Figure 1. The number of runs, N_D, is chosen by the researcher. Unlike orthogonal arrays, N_D can be any number as long as $N_D \geq p$.[†] The algorithm searches the candidate points for a set of N_D design points that is optimal in terms of a given efficiency criterion.

It is almost never possible to list all N_D-run designs and choose *the* most efficient or optimal design, because run time is exponential in the number of candidates. For example, with $2^2 3^3$ in 18 runs, there are $108!/(18!(108-18)!) = 1.39 \times 10^{20}$ possible designs. Instead, nonexhaustive search algorithms are used to generate a small number of designs, and the most efficient one is chosen. The algorithms select points for possible inclusion or deletion, then compute rank-one or rank-two updates of some efficiency criterion. The points that most increase efficiency are added to the design. These algorithms invariably find efficient designs, but they may fail to find *the* optimal design, even for the given criterion. For this reason, we prefer to use terms like *information-efficient* and *D-efficiency* over the more common *optimal* and *D-optimal*.

There are many algorithms for generating information-efficient designs. We will begin by describing some of the simpler approaches and then proceed to the more complicated (and more reliable) algo-

[†]In fact, this restriction is not strictly necessary. So called "super-saturated" designs (Booth and Cox, 1962) have more runs than parameters. However, such designs are typically not used in marketing research. The `%MktRuns` SAS macro provides some guidance on the selection of N_D. See page 740.

rithms. Dykstra's (1971) sequential search method starts with an empty design and adds candidate points so that the chosen efficiency criterion is maximized at each step. This algorithm is fast, but it is not very reliable in finding a globally optimal design. Also, it always finds the same design (due to a lack of randomness).

The Mitchell and Miller (1970) simple exchange algorithm is a slower but more reliable method. It improves the initial design by adding a candidate point and then deleting one of the design points, stopping when the chosen criterion ceases to improve. The DETMAX algorithm of Mitchell (1974) generalizes the simple exchange method. Instead of following each addition of a point by a deletion, the algorithm makes excursions in which the size of the design may vary. These three algorithms add and delete points one at a time.

The next two algorithms add and delete points simultaneously, and for this reason, are usually more reliable for finding the truly optimal design; but because each step involves a search over all possible pairs of candidate and design points, they generally run much more slowly (by an order of magnitude). The Fedorov (1972) algorithm simultaneously adds one candidate point and deletes one design point. Cook and Nachtsheim (1980) define a modified Fedorov algorithm that finds the best candidate point to switch with each design point. The resulting procedure is generally as efficient as the simple Fedorov algorithm in finding the optimal design, but it is up to twice as fast. We extensively use one more algorithm, the coordinate exchange algorithm of Meyer and Nachtsheim (1995). This algorithm does not use a candidate set. Instead it refines an initial design by exchanging each level with every other possible level, keeping those exchanges that increase efficiency. In effect, this method uses a virtual candidate set that consists of all possible runs, even when the full-factorial candidate set is too large to generate and store.

Choice of Criterion and Algorithm. Typically, the choice of efficiency criterion is less important than the choice between manual design creation and computerized search. All of the information-efficient designs presented in this article were generated optimizing D-efficiency because it is faster to optimize than A-efficiency and because it is the standard approach. It is also possible to optimize A-efficiency, though the algorithms generally run much more slowly because the rank-one updates are more complicated with A-efficiency. G-efficiency is an interesting ancillary statistic; however, our experience suggests that attempts to maximize G-efficiency with standard algorithms do not work very well.

The candidate set search algorithms, ordered from the fastest and least reliable to the slowest and most reliable, are: sequential, simple exchange, DETMAX, and modified Fedorov. We always use the modified Fedorov and coordinate exchange algorithms even for extremely large problems; we never even try the other algorithms. For small problems in which the full factorial is no more than a few thousand runs, modified Fedorov tends to work best. For larger problems, coordinate exchange tends to be better. Our latest software, the `%MktEx` macro, tries a few iterations with both methods, then picks the best method for that problem and continues on with more iterations using just the chosen method. See page 667 and all of the examples starting on page 141.

Nonlinear Models. The experimental design problem is relatively simple for linear models and much more complicated for nonlinear models. The usual goal when creating a design is to minimize some function of the variance matrix of the parameter estimates, such as the determinant. For linear models, the variance matrix is proportional to $(\mathbf{X}'\mathbf{X})^{-1}$, and so the design optimality problem is well-posed. However, for nonlinear models, such as the multinomial logit model used with discrete-choice data, the variance matrix depends on the true values of the parameters themselves. (See pages 121, 600, and

363 for more on efficient choice designs based on assumptions about the parameters.) Thus in general, there may not exist a design for a discrete-choice experiment that is always optimal. However, Carson and colleagues (1994) and our experience suggest that *D*-efficient designs work well for discrete-choice models.

Lazari and Anderson (1994) provide a catalog of designs for discrete-choice models, which are good for certain specific problems. For those specific situations, they may be as good as or better than computer-generated designs. However, for many real problems, cataloged designs cannot be used without modification, and modification can reduce efficiency. We carry their work one step further by discussing a general computerized approach to design generation.

Design Comparisons

Comparing Orthogonal Designs. All orthogonal designs are not perfectly or even equally efficient. In this section, we compare designs for $2^2 3^3$. Table 1 gives the information matrix, $\mathbf{X}'\mathbf{X}$, for a full-factorial design using an orthogonal coding. The matrix is a diagonal matrix with the number of runs on the diagonal. The three efficiency criteria are printed after the information matrix. Because this is a full-factorial design, all three criteria show that the design is 100% efficient. The variance matrix (not shown) is $(1/108)\mathbf{I} = 0.0093\mathbf{I}$.

Table 2 shows the information matrix, efficiencies, and variance matrix for a classical 18-run orthogonal design for $2^2 3^3$, Chakravarti's (1956) L_{18}, for comparison with information-efficient designs with 18 runs. (The SAS ADX menu system was used to generate the design. Tables A1 and A2 contain the factor levels and the orthogonal coding used in generating Table 2.) Note that although the factors are all orthogonal to each other, X1 is not balanced. Because of this, the main effect of X1 is estimated with a higher variance (0.063) than X2 (0.056).

The precision of the estimates of the parameters critically depends on the efficiency of the experimental design. The parameter estimates in a general linear model are always unbiased (in fact, best linear unbiased [BLUE]) no matter what design is chosen. However, all designs are not equally efficient. In fact, all orthogonal designs are not equally efficient, even when they have the same factors and the same number of runs. Efficiency criteria can be used to help choose among orthogonal designs. For example, the orthogonal design in Tables 3 and A3 (from the Green and Wind 1975 carpet cleaner example) for $2^2 3^3$ is less *D*-efficient than the Chakravarti L_{18} ($97.4166/98.6998 = 0.9870$). The Green and Wind design can be created from a 3^5 balanced orthogonal array by collapsing two of the three-level factors into two-level factors. In contrast, the Chakravarti design is created from a $2^1 3^4$ balanced orthogonal array by collapsing only one of the three-level factors into a two-level factor. The extra imbalance makes the Green and Wind design less efficient. (Note that the off-diagonal 2 in the Green and Wind information matrix does not imply that X1 and X2 are correlated. It is an artifact of the coding scheme. The off-diagonal 0 in the variance matrix shows that X1 and X2 are uncorrelated.)

Orthogonal Versus Nonorthogonal Designs. Orthogonal designs are not always more efficient than nonorthogonal designs. Tables 4 and A4 show the results for an information-efficient, main-effects-only design in 18 runs. The OPTEX procedure of SAS software was used to generate the design, using the modified Fedorov algorithm. The information-efficient design is slightly better than the classical L_{18}, in terms of the three efficiency criteria. In particular, the ratio of the *D*-efficiencies for the classical and information-efficient designs are $99.8621/98.6998 = 1.0118$. In contrast to the L_{18}, this design is

Table 2
Orthogonal Design
Information Matrix

	Int	X1	X2	X3	-	X4	-	X5	-
Int	18	6	0	0	0	0	0	0	0
X1	6	18	0	0	0	0	0	0	0
X2	0	0	18	0	0	0	0	0	0
X3	0	0	0	18	0	0	0	0	0
-	0	0	0	0	18	0	0	0	0
X4	0	0	0	0	0	18	0	0	0
-	0	0	0	0	0	0	18	0	0
X5	0	0	0	0	0	0	0	18	0
-	0	0	0	0	0	0	0	0	18

98.6998 *D*-efficiency
97.2973 *A*-efficiency
94.8683 *G*-efficiency

Variance Matrix

	Int	X1	X2	X3	-	X4	-	X5	-
Int	63	-21	0	0	0	0	0	0	0
X1	-21	63	0	0	0	0	0	0	0
X2	0	0	56	0	0	0	0	0	0
X3	0	0	0	56	0	0	0	0	0
-	0	0	0	0	56	0	0	0	0
X4	0	0	0	0	0	56	0	0	0
-	0	0	0	0	0	0	56	0	0
X5	0	0	0	0	0	0	0	56	0
-	0	0	0	0	0	0	0	0	56

Note: multiply variance matrix values by 0.001.

Table 3
Green & Wind Orthogonal Design
Information Matrix

	Int	X1	X2	X3	-	X4	-	X5	-
Int	18	-6	-6	0	0	0	0	0	0
X1	-6	18	2	0	0	0	0	0	0
X2	-6	2	18	0	0	0	0	0	0
X3	0	0	0	18	0	0	0	0	0
-	0	0	0	0	18	0	0	0	0
X4	0	0	0	0	0	18	0	0	0
-	0	0	0	0	0	0	18	0	0
X5	0	0	0	0	0	0	0	18	0
-	0	0	0	0	0	0	0	0	18

97.4166 *D*-efficiency
94.7368 *A*-efficiency
90.4534 *G*-efficiency

Variance Matrix

	Int	X1	X2	X3	-	X4	-	X5	-
Int	69	21	21	0	0	0	0	0	0
X1	21	63	0	0	0	0	0	0	0
X2	21	0	63	0	0	0	0	0	0
X3	0	0	0	56	0	0	0	0	0
-	0	0	0	0	56	0	0	0	0
X4	0	0	0	0	0	56	0	0	0
-	0	0	0	0	0	0	56	0	0
X5	0	0	0	0	0	0	0	56	0
-	0	0	0	0	0	0	0	0	56

Notes: multiply variance matrix values by 0.001.

balanced in all the factors, but X1 and X2 are slightly correlated, shown by the 2's off the diagonal. There is no *completely* orthogonal (that is, both balanced and orthogonal) $2^2 3^3$ design in 18 runs.[‡] The nonorthogonality in Table 4 has a much smaller effect on the variances of X1 and X2 (1.2%) than the lack of balance in the orthogonal design in Table 2 has on the variance of X2 (12.5%). In optimizing efficiency, the search algorithms effectively optimize both balance and orthogonality. In contrast, in some orthogonal designs, balance and efficiency may be sacrificed to preserve orthogonality.

This example shows that a nonorthogonal design may be more efficient than an unbalanced orthogonal design. We have seen this phenomenon with other orthogonal designs and in other situations as well. *Preserving orthogonality at all costs can lead to decreased efficiency.* Orthogonality was extremely important in the days before general linear model software became widely available. Today, it is more important to consider efficiency when choosing a design. These comparisons are interesting because they illustrate in a simple example how lack of orthogonality and imbalance affect efficiency. Nonorthogonal designs will never be more efficient than balanced orthogonal designs, when they exist. However, nonorthogonal designs may well be more efficient than unbalanced orthogonal designs. Although this point is interesting and important, what is most important is that good nonorthogonal designs exist in

[‡]In order for the design to be both balanced and orthogonal, the number of runs must be divisible by 2, 3, 2×2, 3×3, and 2×3. Since 18 is not divisible by 2×2, orthogonality and balance are not both simultaneously possible for this design.

Table 4
Information-Efficient Orthogonal Design
Information Matrix

	Int	X1	X2	X3	-	X4	-	X5	-
Int	18	0	0	0	0	0	0	0	0
X1	0	18	2	0	0	0	0	0	0
X2	0	2	18	0	0	0	0	0	0
X3	0	0	0	18	0	0	0	0	0
-	0	0	0	0	18	0	0	0	0
X4	0	0	0	0	0	18	0	0	0
-	0	0	0	0	0	0	18	0	0
X5	0	0	0	0	0	0	0	18	0
-	0	0	0	0	0	0	0	0	18

99.8621 *D*-efficiency
99.7230 *A*-efficiency
98.6394 *G*-efficiency

Variance Matrix

	Int	X1	X2	X3	-	X4	-	X5	-
Int	56	0	0	0	0	0	0	0	0
X1	0	56	-6	0	0	0	0	0	0
X2	0	-6	56	0	0	0	0	0	0
X3	0	0	0	56	0	0	0	0	0
-	0	0	0	0	56	0	0	0	0
X4	0	0	0	0	0	56	0	0	0
-	0	0	0	0	0	0	56	0	0
X5	0	0	0	0	0	0	0	56	0
-	0	0	0	0	0	0	0	0	56

Notes: multiply variance matrix values by 0.001.
The diagonal entries for X1 and X2 are slightly larger at 0.0563 than the other diagonal entries of 0.0556.

Table 5
Unrealistic Combinations Excluded
Information Matrix

	Int	X1	X2	X3	-	X4	-	X5	-
Int	18	0	0	0	0	0	0	0	0
X1	0	18	2	0	0	0	0	0	0
X2	0	2	18	0	0	0	0	0	0
X3	0	0	0	18	0	0	0	0	0
-	0	0	0	0	18	0	0	0	0
X4	0	0	0	0	0	18	0	-6	5
-	0	0	0	0	0	0	18	5	0
X5	0	0	0	0	0	-6	5	18	0
-	0	0	0	0	0	5	0	0	18

96.4182 *D*-efficiency
92.3190 *A*-efficiency
91.0765 *G*-efficiency

Variance Matrix

	Int	X1	X2	X3	-	X4	-	X5	-
Int	56	0	0	0	0	0	0	0	0
X1	0	56	-6	0	0	0	0	0	0
X2	0	-6	56	0	0	0	0	0	0
X3	0	0	0	56	0	0	0	0	0
-	0	0	0	0	56	0	0	0	0
X4	0	0	0	0	0	69	-7	25	-20
-	0	0	0	0	0	-7	61	-20	2
X5	0	0	0	0	0	25	-20	69	-7
-	0	0	0	0	0	-20	2	-7	61

Notes: multiply variance matrix values by 0.001.

many situations in which no orthogonal designs exist. These designs are also discussed and at a more basic level starting on page 54.

Design Considerations

Codings and Efficiency. The specific design matrix coding does not affect the relative *D*-efficiency of competing designs. Rank-preserving linear transformations are immaterial, whether they are from full-rank indicator variables to effects coding or to an orthogonal coding such as the one shown in Table A2. Any full-rank coding is equivalent to any other. The absolute *D*-efficiency values will change, but the ratio of two *D*-efficiencies for competing designs is constant. Similarly, scale for quantitative factors does not affect relative efficiency. The proof is simple. If design $\mathbf{X}_1$ is recoded to $\mathbf{X}_1\mathbf{A}$, then $|(\mathbf{X}_1\mathbf{A})'(\mathbf{X}_1\mathbf{A})| = |\mathbf{A}'\mathbf{X}_1'\mathbf{X}_1\mathbf{A}| = |\mathbf{A}\mathbf{A}'||\mathbf{X}_1'\mathbf{X}_1|$. The relative efficiency of design $\mathbf{X}_1$ compared to $\mathbf{X}_2$ is the same as $\mathbf{X}_1\mathbf{A}$ compared to $\mathbf{X}_2\mathbf{A}$, since the $|\mathbf{A}\mathbf{A}'|$'s terms in efficiency ratios

will cancel. We prefer the orthogonal coding because it yields "nicer" information matrices with the number of runs on the diagonal and efficiency values scaled so that 100 means perfect efficiency.

Quantitative Factors. The factors in an experimental design are usually qualitative (nominal), but quantitative factors such as price are also important. With quantitative factors, the choice of levels depends on the function of the original variable that is modeled. To illustrate, consider a pricing study in which price ranges from \$0.99 to \$1.99. If a linear function of price is modeled, only two levels of price should be used−the end points (\$0.99 and \$1.99). Using prices that are closer together is inefficient; the variances of the estimated coefficients will be larger. The efficiency of a given design is affected by the coding of quantitative factors, even though the relative efficiency of competing designs is unaffected by coding. Consider treating the second factor of the Chakravarti L_{18}, $2^2 3^3$ as linear. It is nearly three times more *D*-efficient to use \$0.99 and \$1.99 as levels instead of \$1.49 and \$1.50 ($58.6652/21.0832 = 2.7826$). To visualize this, imagine supporting a yard stick (line) on your two index fingers (with two points). The effect on the slope of the yard stick of small vertical changes in finger locations is much greater when your fingers are closer together than when they are near the ends.

Of course there are other considerations besides the numerical measure of efficiency. It would not make sense to use prices of \$0.01 and \$1,000,000 just because that is more efficient than using \$0.99 and \$1.99. The model is almost certainly not linear over this range. To maximize efficiency, the range of experimentation for quantitative factors should be as large as possible, given that the model is plausible.

The number of levels also affects efficiency. Because two points define a line, it is inefficient to use more than two points to model a linear function. When a quadratic function is used (x and x^2 are included in the model), three points are needed−the two extremes and the midpoint. Similarly, four points are needed for a cubic function. More levels are needed when the functional form is unknown. Extra levels allow for the examination of complicated nonlinear functions, with a cost of decreased efficiency for the simpler functions. When the function is assumed to be linear, experimental points should not be spread throughout the range of experimentation. See page 785 for a discussion of nonlinear functions of quantitative factors in conjoint analysis.

Most of the discussion outside this section has concerned qualitative (nominal) factors, even if that was not always explicitly stated. Quantitative factors complicate general design characterizations. For example, we previously stated that "if a balanced and orthogonal design exists, then it has optimum efficiency." This statement must be qualified to be absolutely correct. The design would not be optimal if, for example, a three-level factor were treated as quantitative and linear.

Nonstandard Algorithms and Criteria. Other researchers have proposed other algorithms and criteria. Steckel, DeSarbo, and Mahajan (SDM) (1991) proposed using computer-generated experimental designs for conjoint analysis to exclude unacceptable combinations from the design. They considered a nonstandard measure of design goodness based on the determinant of the (m-factor × m-factor) correlation matrix ($|\mathbf{R}|$) instead of the customary determinant of the (p-parameter × p-parameter) variance matrix ($|(\mathbf{X}'\mathbf{X})^{-1}|$). The SDM approach represents each factor by a single column rather than as a set of coded indicator variables. Designs generated using nonstandard criteria will not generally be efficient in terms of standard criteria like *A*-efficiency and *D*-efficiency, so the parameter estimates will have larger variances. To illustrate graphically, refer to Figure 1. The criterion $|\mathbf{R}|$ cannot distinguish between any of the nine different four-point designs, constructed from this candidate set, that form a square or a rectangle. All are orthogonal; only one is optimal.

We generated a *D*-efficient design for SDM's example, treating the variables as all quantitative (as they did). The $|\mathbf{R}|$ for the SDM design is 0.9932, whereas the $|\mathbf{R}|$ for the information-efficient design is 0.9498. The SDM approach works quite well in maximizing $|\mathbf{R}|$; hence the SDM design is close to orthogonal. However, efficiency is not always maximized when orthogonality is maximized. The SDM design is approximately 75% as *D*-efficient as a design generated with standard criteria and algorithms ($70.1182/93.3361 = 0.7512$).

Choosing a Design. Computerized search algorithms generate many designs, from which the researcher must choose one. Often, several designs are tied or nearly tied for the best *D*, *A*, and *G* information efficiencies. A design should be chosen after examining the design matrix, its information matrix, its variance matrix, factor correlations, and levels frequencies. *It is important to look at the results and not just routinely choose the design from the top of the list.*

For studies involving human subjects, achieving at least nearly-balanced designs is an important consideration. Consider for example a two-level factor in an 18-run design in which one level occurs 12 times and the other level occurs 6 times versus a design in which each level occurs 9 times. Subjects who see one level more often than the other may try to read something into the study and adjust their responses in some way. Alternatively, subjects who see one level most often may respond differently than those who see the second level most often. These are not concerns with nearly balanced designs. One design selection strategy is to choose the most balanced design from the top few.

Many other strategies can be used. Perhaps correlation and imprecision are tolerable in some variables but not in others. Perhaps imbalance is tolerable, but the correlations between the factors should be minimal. Goals will no doubt change from experiment to experiment. Choosing a suitable design can be part art and part science. Efficiency should always be considered when choosing between alternative designs, even manually created designs, but it is not the only consideration.*

Adding Observations or Variables. These techniques can be extended to augment an existing design. A design with r runs can be created by augmenting m specified combinations (established brands or existing combinations) with $r - m$ combinations chosen by the algorithm. Alternatively, combinations that must be used for certain variables can be specified, and then the algorithm picks the levels for the other variables (Cook and Nachtsheim 1989). This can be used to ensure that some factors are balanced or uncorrelated; another application is blocking factors. Using design algorithms, we are able to establish numbers of runs and blocking patterns that fit into practical fielding schedules.

Designs with Interactions. There is a growing interest in using both main effects and interactions in discrete-choice models, because interaction and cross-effect terms may improve aggregate models (Elrod, Louviere, and Davey 1992). The current standard for choice models is to have all main-effects estimable both within and between alternatives. It is often necessary to estimate interactions within alternatives, such as in modeling separate price elasticities for product forms, sizes or packages. For certain classes of designs, in which a brand appears in only a subset of runs, it is often necessary to have estimable main-effects, own-brand interactions, and cross-effects in the submatrix of the design in which that brand is present. One way to ensure estimability is to include in the model interactions between the alternative-specific variables of interest and the indicator variables that control for presence or absence of the brand in the choice set. Orthogonal designs that allow for estimation of interactions are usually very large, whereas efficient nonorthogonal designs can be generated for any linear model, including models with interactions, and for any (reasonable) number of runs.

*See the `%MktEval` macro, page 663, for a tool that helps evaluate designs.

Unrealistic Combinations. It is sometimes useful to exclude certain combinations from the candidate set. SDM (1991) have also considered this problem. Consider a discrete-choice model for several brands and their line extensions. It may not make sense to have a choice set in which the line extension is present and the "flagship" brand absent. Of course, as we eliminate combinations, we may introduce unavoidable correlation between the parameter estimates. In Tables 5 and A5, the twenty combinations where (X1 = 1 and X2 = 1 and X3 = 1) or (X4 = 1 and X5 = 1) were excluded and an 18-run design was generated with the modified Fedorov algorithm. With these restrictions, all three efficiency criteria dropped, for example 96.4182/99.8621 = 0.9655. This shows that the design with excluded combinations is almost 97% as *D*-efficient as the best (unrestricted) design. The information matrix shows that X1 and X2 are correlated, as are X4 and X5. This is the price paid for obtaining a design with only realistic combinations.

In the "Quantitative Factors" section, we stated "Because two points define a line, it is inefficient to use more than two points to model a linear function." When unrealistic combinations are excluded, this statement may no longer be true. For example, if minimum price with maximum size is excluded, an efficient design may involve the median price and size.

Choosing the Number of Runs. Deciding on a number of runs for the design is a complicated process; it requires balancing statistical concerns of estimability and precision with practical concerns like time and subject fatigue. Optimal design algorithms can generate designs for any number of runs greater than or equal to the number of parameters. The variances of the least-squares estimates of the part-worth utilities will be roughly inversely proportional to both the *D*-efficiency and the number of runs. In particular, for a given number of runs, a *D*-efficient design will give more accurate estimates than would be obtained with a less efficient design. A more precise value for the number of choices depends on the ratio of the inherent variability in subject ratings to the absolute size of utility that is considered important. Subject concerns probably outweigh the statistical concerns, and the best course is to provide as many products as are practical for the subjects to evaluate. In any case, the use of information-efficient designs provides more flexibility than manual methods.

Asymmetry in the Number of Levels of Variables. In many practical applications of discrete-choice modeling, there is asymmetry in the number of factor levels, and interaction and polynomial parameters must be estimated. One common method for generating choice model designs is to create a resolution III orthogonal array and modify it. The starting point is a $q^{\Sigma M_j}$ design, where q represents a fixed number of levels across all attributes and M_j represents the number of attributes for brand j. For example, in the "Consumer Food Product" example in a subsequent section, with five brands with 1, 3, 1, 2, and 1 attributes and with each attribute having at most four levels, the starting point is a 4^8 orthogonal array. Availability cross-effect designs are created by letting one of the M_j variables function as an indicator for presence/absence of each brand or by allowing one level of a common variable (price) to operate as the indicator. These methods are fairly straightforward to implement in designs in which the factor levels are all the same, but they become quite difficult to set up when there are different numbers of levels for some factors or in which specific interactions must be estimable.

Asymmetry in the number of levels of factors may be handled either by using the "coding down" approach (Addelman 1962b) or by expansion. In the coding down approach, designs are created using factors that have numbers of levels equal to the largest number required in the design. Factors that have fewer levels are created by recoding. For example, a five-level factor {1, 2, 3, 4, 5} can be recoded into a three-level factor by duplicating levels {1, 1, 2, 2, 3}. The variables will still be orthogonal because the indicator variables for the recoding are in a subspace of the original space. However, recoding introduces imbalance and inefficiency. The second method is to expand a factor at k-levels

into several variables at some fraction of k-levels. For example, a four-level variable can be expanded into three orthogonal two-level variables. In many cases, both methods must be used to achieve the required design.

These approaches are difficult for a simple main-effect design of resolution III and extremely difficult when interactions between asymmetric factors must be considered. In practical applications, asymmetry is the norm. Consider for example the form of an analgesic product. One brand may have caplet and tablet varieties, another may have tablet, liquid, and chewable forms. In a discrete-choice model, these two brand/forms must be modeled as asymmetric alternative-specific factors. If we furthermore anticipated that the direct price elasticity might vary, depending on the form, we would need to estimate the interaction of a quantitative price variable with the nominal-level form variable.

Computerized search methods are simpler to use by an order of magnitude. They provide asymmetric designs that are usually nearly balanced, as well as providing easy specification for interactions, polynomials and continuous by class effects.

Strategies for Many Variables. Consider generating a 3^{15} design in 36 runs. There are 14,348,907 combinations in the full-factorial design, which is too many to use even for a candidate set. For problems like this, the coordinate exchange algorithm (Meyer and Nachtsheim 1995)] works well. The `%MktEx` macro which uses coordinate exchange with a partial orthogonal array initialization easily finds design over 98.9% D-efficient. Even designs with over 100 variables can be created this way.

Examples

Choice of Consumer Food Products. Consider the problem of using a discrete choice model to study the effect of introducing a retail food product. This may be useful, for example, to refine a marketing plan or to optimize a product prior to test market. A typical brand team will have several concerns such as knowing the potential market share for the product, examining the source of volume, and providing guidance for pricing and promotions. The brand team may also want to know what brand attributes have competitive clout and want to identify competitive attributes to which they are vulnerable.

To develop this further, assume our client wishes to introduce a line extension in the category of frozen entrees. The client has one nationally branded competitor, a regional competitor in each of three regions, and a profusion of private label products at the grocery chain level. The product comes in two different forms: stove-top or microwaveable. The client believes that the private labels are very likely to mimic this line extension and to sell it at a lower price. The client suspects that this strategy on the part of private labels may work for the stove-top version but not for the microwaveable, in which they have the edge on perceived quality. They also want to test the effect of a shelf-talker that will draw attention to their product.

This problem may be set up as a discrete choice model in which a respondent's choice among brands, given choice set C_a of available brands, will correspond to the brand with the highest utility. For each brand i, the utility U_i is the sum of a systematic component V_i and a random component e_i. The probability of choosing brand i from choice set C_a is therefore:

$$P(i|C_a) = P(U_i > \max(U_j)) = P(V_i + e_i > \max(V_j + e_j)) \ \ \forall \ \ (j \neq i) \in C_a$$

Table 6
Factors and Levels

Alternative	Factor	Levels	Brand	Description
1	X1	4	Client	3 prices + absent
2	X2	4	Client Line Extension	3 prices + absent
	X3	2		microwave/stove-top
	X4	2		shelf-talker yes/no
3	X5	3	Regional	2 prices + absent
4	X6	3	Private Label	2 prices + absent
	X7	2		microwave/stove-top
5	X8	3	Competitor	2 prices + absent

Assuming that the e_i follow an extreme value type I distribution, the conditional probabilities $P(i|C_a)$ can be found using the MNL formulation of McFadden (1974)

$$P(i|C_a) = \exp(V_i)/\sum_{j\in C_a}\exp(V_j)$$

One of the consequences of the MNL formulation is the property of independence of irrelevant alternatives (IIA). Under the assumption of IIA, all cross-effects are assumed to be equal, so that if a brand gains in utility, it draws share from all other brands in proportion to their current shares. Departures from IIA exist when certain subsets of brands are in more direct competition and tend to draw a disproportionate amount of share from each other than from other members in the category. One way to capture departures from IIA is to use the mother logit formulation of McFadden (1974). In these models, the utility for brand i is a function of both the attributes of brand i and the attributes of other brands. The effect of one brand's attributes on another is termed a *cross-effect*. In the case of designs in which only subsets C_a of the full shelf set C appear, the effect of the presence or absence of one brand on the utility of another is termed an *availability cross-effect*.

In the frozen entree example, there are five alternatives: the client, the client's line extension, a national branded competitor, a regional brand and a private label brand. Several regional and private labels can be tested in each market, then aggregated for the final model. Note that the line extension is treated as a separate alternative rather than as a "level" of the client brand. This enables us to model the source of volume for the new entry and to quantify any cannibalization that occurs. Each brand is shown at either two or three price points. Additional price points are included so that quadratic models of price elasticity can be tested. The indicator for the presence or absence of any brand in the shelf set is coded using one level of the price variable. The layout of factors and levels is given in Table 6.

In addition to intercepts and main effects, we also require that all two-way interactions within alternatives be estimable: X2*X3, X2*X4, X3*X4 for the line extension and X6*X7 for private labels. This will enable us to test for different price elasticities by form (stove-top versus microwaveable) and to

see if the promotion works better combined with a low price or with different forms. Using a linear model for X1-X8, the total number of parameters including the intercept, all main effects, and two-way interactions with brand is 25. This assumes that price is treated as qualitative. The actual number of parameters in the choice model is larger than this because of the inclusion of cross-effects. Using indicator variables to code availability, the systematic component of utility for brand i can be expressed as:

$$V_i = a_i + \sum_k (b_{ik} \times x_{ik}) + \sum_{j \neq i} z_j (d_{ij} + \sum_l (g_{ijl} \times x_{jl}))$$

where

a_i = intercept for brand i
b_{ik} = effect of attribute k for brand i, where $k = 1, .., K_i$
x_{ik} = level of attribute k for brand i
d_{ij} = availability cross-effect of brand j on brand i
z_j = availability code $= \begin{cases} 1 & \text{if } j \in C_a, \\ 0 & \text{otherwise} \end{cases}$
g_{ijl} = cross-effect of attribute l for brand j on brand i, where $l = 1, .., L_j$
x_{jl} = level of attribute l for brand j.

The x_{ik} and x_{jl} might be expanded to include interaction and polynomial terms. In an availability cross-effects design, each brand is present in only a fraction of choice sets. The size of this fraction or subdesign is a function of the number of levels of the alternative-specific variable that is used to code availability (usually price). For example, if price has three valid levels and a fourth "zero" level to indicate absence, then the brand will appear in only three out of four runs. Following Lazari and Anderson (1994), the size of each subdesign determines how many model equations can be written for each brand in the discrete choice model. If X_i is the subdesign matrix corresponding to V_i, then each X_i must be full rank to ensure that the choice set design provides estimates for all parameters.

To create the design, a full candidate set is generated consisting of 3456 runs. It is then reduced to 2776 runs that contain between two and four brands so that the respondent is never required to compare more than four brands at a time. In the algorithm model specification, we designate all variables as classification variables and require that all main effects and two-way interactions within brands be estimable. The number of runs to use follows from a calculation of the number of parameters that we wish to estimate in the various submatrices $\mathbf{X}_i$ of $\mathbf{X}$. Assuming that there is a category "None" used as a reference cell, the numbers of parameters required for various alternatives are shown in the Table 7 along with the size of submatrices (rounded down) for various numbers of runs. Parameters for quadratic price models are given in parentheses. Note that the effect of private label being in a microwaveable or stove-top form (stove/micro cross-effect) is an explicit parameter under the client line extension.

The number of runs chosen was N=26. This number provides adequate degrees of freedom for the linear price model and will also allow estimation of direct quadratic price effects. To estimate quadratic cross-effects for price would require 32 runs at the very least. Although the technique of using two-way interactions between nominal level variables will usually guarantee that all direct and cross-effects are estimable, it is sometimes necessary and a good practice to check the ranks of the submatrices for more complex models (Lazari and Anderson 1994). Creating designs for cross effects can be difficult, even with the aid of a computer.

Table 7
Parameters

Effect	Client	Client Line Extension	Regional	Private Label	Competitor
intercept	1	1	1	1	1
availability cross-effects	4	4	4	4	4
direct price effect	1 (2)	1 (2)	1	1	1
price cross-effects	4 (8)	4 (8)	4	4	4
stove versus microwave	-	1	-	1	-
stove/micro cross-effects	-	1	-	-	-
shelf-talker	-	1	-	-	-
price*stove/microwave	-	1 (2)	-	1	-
price*shelf-talker	-	1 (2)	-	-	-
stove/micro*shelf-talker	-	1	-	-	-
Total	10 (15)	16 (23)	10	12	10
Subdesign size					
22 runs	16	16	14	14	14
26 runs	19	19	17	17	17
32 runs	24	24	21	21	21

It took approximately 4.5 minutes to generate a design. The final (unrandomized) design in 26 runs is in table A6.† The coded choice sets are presented in Table A7 and the level frequencies are presented in Table A8. Note that the runs have been ordered by the presence/absence of the shelf-talker. This ordering is done because it is unrealistic to think that once the respondent's attention has been drawn in by the promotion, it can just be "undrawn." The two blocks that result may be shown to two groups of people or to the same people sequentially. It would be extremely difficult and time consuming to generate a design for this problem without a computerized algorithm.

Conjoint Analysis with Aggregate Interactions. This example illustrates creating a design for a conjoint analysis study. The goal is to create a 3^6 design in 90 runs. The design consists of five blocks of 18 runs each, so each subject will only have to rate 18 products. Within each block, main-effects must be estimable. In the aggregate, all main-effects and two-way interactions must be estimable. (The utilities from the main-effects models will be used to cluster subjects, then in the aggregate analysis, clusters of subjects will be pooled across blocks and the blocking factor ignored.) Our goal is to create a design that is simultaneously efficient in six ways. Each of the five blocks should be an efficient design for a first-order (main-effects) model, and the aggregate design should be efficient for the second-order (main-effects and two-way interactions) model. The main-effects models for the five blocks have $5(1 + 6(3 - 1)) = 65$ parameters. In addition, there are $(6 \times 5/2)(3 - 1)(3 - 1) = 60$ parameters for interactions in the aggregate model. There are more parameters than runs, but not all

†This is the design that was presented in the original 1994 paper, which due to differences in the random number seeds, is not reproduced by today's tools.

parameters will be simultaneously estimated.

One approach to this problem is the Bayesian regression method of DuMouchell and Jones (1994). Instead of optimizing $|\mathbf{X}'\mathbf{X}|$, we optimized $|\mathbf{X}'\mathbf{X}+\mathbf{P}|$, where $\mathbf{P}$ is a diagonal matrix of prior precisions. This is analogous to ridge regression, in which a diagonal matrix is added to a rank-deficient $\mathbf{X}'\mathbf{X}$ to create a full-rank problem. We specified a model with a blocking variable, main effects for the six factors, block-effect interactions for the six factors, and all two-way interactions. We constructed $\mathbf{P}$ to contain zeros for the blocking variable, main effects, and block-effect interactions, and 45s (the number of runs divided by 2) for the two-way interactions. Then we used the modified Fedorov algorithm to search for good designs.

With an appropriate coding for $\mathbf{X}$, the value of the prior precision for a parameter roughly reflects the number of runs worth of prior information available for that parameter. The larger the prior precision for a parameter, the less information about that parameter is in the final design. Specifying a nonzero prior precision for a parameter reduces the contribution of that parameter to the overall efficiency. For this problem, we wanted maximal efficiency for the within-subject main-effects models, so we gave a nonzero prior precision to the aggregated two-way interactions.

Our best design had a *D*-efficiency for the second-order model of 63.9281 (with a *D*-efficiency for the aggregate main-effects model of 99.4338) and *D*-efficiencies for the main-effects models within each block of 100.0000, 100.0000, 100.0000, 99.0981, and 98.0854. The design is completely balanced within all blocks. We could have specified other values in $\mathbf{P}$ and gotten better efficiency for the aggregate design but less efficiency for the blocks. Choice of $\mathbf{P}$ depends in part on the primary goals of the experiment. It may require some simulation work to determine a good choice of $\mathbf{P}$.

All of the examples in this article so far have been straight-forward applications of computerized design methodology. A set of factors, levels, and estimable effects was specified, and the computer looked for an efficient design for that specification. Simple problems, such as those discussed previously, require only a few minutes of computer time. This problem was much more difficult, so we let a work station generate designs for about 72 hours. (We could have found less efficient but still acceptable designs in much less time.) We were asking the computer to find a good design out of over 9.6×10^{116} possibilities. This is like looking for a needle in a haystack, when the haystack is the size of the entire known universe. With such problems, we may do better if we use our intuition to give the computer "hints," forcing certain structure into the design. To illustrate, we tried this problem again, this time using a different approach.

We used the modified Fedorov algorithm to generate main-effects only 3^6 designs in 18 runs. We stopped when we had ten designs all with 100% efficiency. We then wrote an ad hoc program that randomly selected five of the ten designs, randomly permuted columns within each block, and randomly permuted levels within each block. These operations do not affect the first-order efficiencies but do affect the overall efficiency for the aggregate design. When an operation increased efficiency, the new design was kept. We iterated over the entire design 20 times. We let the program run for about 16 hours, which generated 98 designs, and we found our best design in three hours. Our best design had a *D*-efficiency for the second-order model of 68.0565 (versus 63.9281 previously), and all first-order efficiencies of 100.

Many other variations on this approach could be tried. For example, columns and blocks could be chosen at random, instead of systematically. We performed excursions of up to eight permutations before we reverted to the previous design. This number could be varied. It seemed that permuting the levels helped more than permuting the columns, though this was not thoroughly investigated. Whatever is done, it is important to consider efficiency. For example, just randomly permuting levels can create very inefficient designs.

For this particular problem, the ad hoc algorithm generated better designs than the Bayesian method, and it required less computer time. In fact, 91 out of the 98 ad hoc designs were better than the best Bayesian design. However, the ad hoc method required much more programmer time. It is possible to manually create a design for this situation, but it would be extremely difficult and time consuming to find an efficient design without a computerized algorithm for all but the most sophisticated of human designers. The best designs were found when used both our human design skills and a computerized search. We have frequently found this to be the case.

Conclusions

Computer-generated experimental designs can provide both better and more general designs for discrete-choice and preference-based conjoint studies. Classical designs, obtained from books or computerized tables, can be good options when they exist, but they are not the only option. The time-consuming and potentially error-prone process of finding and manually modifying an existing design can be avoided. When the design is nonstandard and there are restrictions, a computer can generate a design, and it can be done quickly. In most situations, a good design can be generated in a few minutes or hours, though for certain difficult problems more time may be necessary. Furthermore, when the circumstances of the project change, a new design can again be generated quickly.

We do not argue that computerized searches for D-efficient designs are *uniformly* superior to manually generated designs. The human designer, using intuition, experience, and heuristics, can recognize structure that an optimization algorithm cannot. On the other hand, the computerized search usually does a good job, it is easy to use, and it can create a design faster than manual methods, especially for the nonexpert. Computerized search methods and the use of efficiency criteria can benefit expert designers as well. For example, the expert can manually generate a design and then use the computer to evaluate and perhaps improve its efficiency.

In nonstandard situations, simultaneous balance and orthogonality may be unobtainable. Often, the best that can be hoped for is optimal efficiency. Computerized algorithms help by searching for the most efficient designs from a potentially very large set of possible designs. Computerized search algorithms for D-efficient designs do not supplant traditional design-creation skills. Rather, they provide helpful tools for finding good, efficient experimental designs.

Table A1
Chakravarti's L_{18}, Factor Levels

X1	X2	X3	X4	X5
-1	-1	-1	-1	-1
-1	-1	0	0	1
-1	-1	1	1	0
-1	1	-1	1	0
-1	1	0	-1	-1
-1	1	1	0	1
1	-1	-1	0	0
1	-1	-1	1	1
1	-1	0	-1	0
1	-1	0	1	-1
1	-1	1	-1	1
1	-1	1	0	-1
1	1	-1	-1	1
1	1	-1	0	-1
1	1	0	0	0
1	1	0	1	1
1	1	1	-1	0
1	1	1	1	-1

Table A2
Chakravarti's L_{18}, Orthogonal Coding

X1	X2	X3	-	X4	-	X5	-
-1	-1	-1.225	-0.707	-1.225	-0.707	-1.225	-0.707
-1	-1	0.000	1.414	0.000	1.414	1.225	-0.707
-1	-1	1.225	-0.707	1.225	-0.707	0.000	1.414
-1	1	-1.225	-0.707	1.225	-0.707	0.000	1.414
-1	1	0.000	1.414	-1.225	-0.707	-1.225	-0.707
-1	1	1.225	-0.707	0.000	1.414	1.225	-0.707
1	-1	-1.225	-0.707	0.000	1.414	0.000	1.414
1	-1	-1.225	-0.707	1.225	-0.707	1.225	-0.707
1	-1	0.000	1.414	-1.225	-0.707	0.000	1.414
1	-1	0.000	1.414	1.225	-0.707	-1.225	-0.707
1	-1	1.225	-0.707	-1.225	-0.707	1.225	-0.707
1	-1	1.225	-0.707	0.000	1.414	-1.225	-0.707
1	1	-1.225	-0.707	-1.225	-0.707	1.225	-0.707
1	1	-1.225	-0.707	0.000	1.414	-1.225	-0.707
1	1	0.000	1.414	0.000	1.414	0.000	1.414
1	1	0.000	1.414	1.225	-0.707	1.225	-0.707
1	1	1.225	-0.707	-1.225	-0.707	0.000	1.414
1	1	1.225	-0.707	1.225	-0.707	-1.225	-0.707

Table A3
Green & Wind
Orthogonal Design
Example

X1	X2	X3	X4	X5
-1	-1	-1	-1	-1
-1	-1	-1	1	0
-1	-1	0	-1	-1
-1	-1	0	0	1
-1	-1	0	1	-1
-1	-1	1	-1	0
-1	-1	1	0	1
-1	-1	1	1	0
-1	1	-1	1	1
-1	1	-1	-1	1
-1	1	0	0	0
-1	1	1	0	-1
1	-1	-1	0	-1
1	-1	-1	0	0
1	-1	0	1	1
1	-1	1	-1	1
1	1	1	1	-1
1	1	0	-1	0

Table A4
Information-Efficient
Design,
Factor Levels

X1	X2	X3	X4	X5
-1	-1	-1	0	-1
-1	-1	0	-1	0
-1	-1	0	1	-1
-1	-1	1	0	1
-1	-1	1	1	1
-1	1	-1	-1	0
-1	1	-1	0	-1
-1	1	0	-1	1
-1	1	1	1	0
1	-1	-1	-1	1
1	-1	-1	1	0
1	-1	0	0	0
1	-1	1	-1	-1
1	1	-1	1	1
1	1	0	0	1
1	1	0	1	-1
1	1	1	-1	-1
1	1	1	0	0

Table A5
Information-Efficient
Design, Unrealistic
Combinations Excluded

X1	X2	X3	X4	X5
-1	-1	-1	1	0
-1	-1	-1	-1	1
-1	-1	-1	0	-1
-1	-1	0	-1	1
-1	-1	0	0	0
-1	1	1	1	0
-1	1	1	-1	-1
-1	1	1	0	1
-1	1	0	1	-1
1	-1	1	1	-1
1	-1	1	-1	0
1	-1	1	0	1
1	-1	0	1	-1
1	1	-1	1	0
1	1	-1	-1	-1
1	1	-1	0	1
1	1	0	-1	1
1	1	0	0	0

Table A6
Consumer Food Product (Raw) Design

X1	X2	X3	X4	X5	X6	X7	X8
1	1	2	1	1	2	1	3
1	2	2	1	2	3	1	2
1	4	1	1	1	3	1	3
2	2	1	1	3	2	1	1
2	3	2	1	2	2	2	3
2	4	2	1	3	3	2	2
3	1	1	1	3	2	2	2
3	3	2	1	3	1	2	1
3	4	2	1	2	1	1	1
4	1	1	1	2	3	2	1
4	1	2	1	3	3	1	1
4	2	2	1	1	2	2	3
4	3	1	1	1	1	1	2
1	3	1	2	3	2	2	1
1	3	2	2	3	1	1	3
1	4	2	2	1	1	2	1
2	1	1	2	1	3	1	1
2	2	2	2	3	2	1	1
2	3	1	2	2	1	2	3
2	4	1	2	3	1	1	2
3	1	2	2	2	3	2	2
3	2	1	2	1	3	2	3
3	4	2	2	2	3	1	3
4	1	1	2	3	2	1	3
4	2	1	2	2	1	2	2
4	3	2	2	1	2	1	2

Table A7
Consumer Food Product Choice Set

Block 1: Shelf-Talker Absent For Client Line Extension

Choice Set	Client Brand	Client Line Extension	Regional Brand	Private Label	National Competitor
1	$1.29	$1.39/stove	$1.99	$2.29/micro	N/A
2	$1.29	$1.89/stove	$2.49	N/A	$2.39
3	$1.29	N/A	$1.99	N/A	N/A
4	$1.69	$1.89/micro	N/A	$2.29/micro	$1.99
5	$1.69	$2.39/stove	$2.49	$2.29/stove	N/A
6	$1.69	N/A	N/A	N/A	$2.39
7	$2.09	$1.39/micro	N/A	$2.29/stove	$2.39
8	$2.09	$2.39/stove	N/A	$1.49/stove	$1.99
9	$2.09	N/A	$2.49	$1.49/micro	$1.99
10	N/A	$1.39/micro	$2.49	N/A	$1.99
11	N/A	$1.39/stove	N/A	N/A	$1.99
12	N/A	$1.89/stove	$1.99	$2.29/stove	N/A
13	N/A	$2.39/micro	$1.99	$1.49/micro	$2.39

Block 2: Shelf-Talker Present For Client Line Extension

Choice Set	Client Brand	Client Line Extension	Regional Brand	Private Label	National Competitor
14	$1.29	$2.39/micro	N/A	$2.29/stove	$1.99
15	$1.29	$2.39/stove	N/A	$1.49/micro	N/A
16	$1.29	N/A	$1.99	$1.49/stove	$1.99
17	$1.69	$1.39/micro	$1.99	N/A	$1.99
18	$1.69	$1.89/stove	N/A	$2.29/micro	$1.99
19	$1.69	$2.39/micro	$2.49	$1.49/stove	N/A
20	$1.69	N/A	N/A	$1.49/micro	$2.39
21	$2.09	$1.39/stove	$2.49	N/A	$2.39
22	$2.09	$1.89/micro	$1.99	N/A	N/A
23	$2.09	N/A	$2.49	N/A	N/A
24	N/A	$1.39/micro	N/A	$2.29/micro	N/A
25	N/A	$1.89/micro	$2.49	$1.49/stove	$2.39
26	N/A	$2.39/stove	$1.99	$2.29/micro	$2.39

Table A8
Consumer Food Product Design Level Frequencies

Level	X1	X2	X3	X4	X5	X6	X7	X8
1	6	7	12	13	8	8	14	9
2	7	6	14	13	8	9	12	8
3	6	7			10	9		9
4	7	6						

Table A9
Consumer Food Product Design Creation Code

```
*-------------------------------------------------------*
|                Construct the Design.                  |
*-------------------------------------------------------;

%macro bad;
  bad = (x1 < 4) + (x2 < 4) + (x5 < 3) + (x6 < 3) + (x8 < 3);
  bad = abs(bad - 3) * ((bad < 2) | (bad > 4));
  %mend;

%mktex( 4 4 2 2 3 3 2 3, n=26, interact=x2*x3 x2*x4 x3*x4 x6*x7,
        restrictions=bad, outr=sasuser.choicdes )

*-------------------------------------------------------*
|                  Print the Design.                    |
*-------------------------------------------------------;

proc format;
   value yn    1 = 'No'    2 = 'Talker';
   value micro 1 = 'Micro' 2 = 'Stove';
   run;

data key;
   missing N;
   input x1-x8;
   format x1 x2 x5 x6 x8 dollar5.2
          x4 yn. x3 x7 micro.;
   label x1 = 'Client Brand'
         x2 = 'Client Line Extension'
         x3 = 'Client Micro/Stove'
         x4 = 'Shelf Talker'
         x5 = 'Regional Brand'
         x6 = 'Private Label'
         x7 = 'Private Micro/Stove'
         x8 = 'National Competitor';
   datalines;
1.29 1.39 1 1 1.99 1.49 1 1.99
1.69 1.89 2 2 2.49 2.29 2 2.39
2.09 2.39 . . N    N    .    N
N    N    . . .    .    .    .
;

%mktlab(data=sasuser.choicdes, key=key)

proc sort out=sasuser.finchdes; by x4; run;

proc print label; id x4; by x4; run;
```

A General Method for Constructing Efficient Choice Designs

Klaus Zwerina

Joel Huber

Warren F. Kuhfeld

Abstract

Researchers have traditionally built choice designs using extensions of concepts from the general linear design literature. We show that a computerized search strategy can generate efficient choice designs with standard personal computers. This approach holds three important advantages over previous design strategies. First, it allows the incorporation of anticipated model parameters, thereby increasing design efficiency and considerably reducing the number of required choices. Second, complex choice designs can be easily generated, allowing researchers to conduct choice experiments that more closely mirror actual market conditions. Finally, researchers can explore model and design modifications and examine trade-offs between a design's statistical benefits and its operational and behavioral costs.*

Introduction

Discrete choice experiments are becoming increasingly popular in marketing, economics, and transportation. These experiments enable researchers to model choice in an explicit competitive context, thus realistically emulating market decisions. A choice design consists of choice sets composed of several alternatives, each defined as combinations of different attribute levels. A good choice design is efficient, meaning that the parameters of the choice model are estimated with maximum precision.

A number of methods have been suggested for building choice designs (Anderson and Wiley 1992, Bunch, Louviere, and Anderson 1996, Krieger and Green 1991, Kuhfeld 2005 (page 141), Lazari and Anderson 1994, Louviere and Woodworth 1983). Most of the methods use extensions of standard linear experimental designs (Addelman 1962b, Green 1974). However, the use of linear designs in choice experiments may be nonoptimal due to two well-known differences between linear and choice

*Klaus Zwerina is a consultant at BASF AG, Ludwigshafen, Germany. Joel Huber is Professor of Marketing, Fuqua School of Business, Duke University. Warren F. Kuhfeld is Manager, Multivariate Models R&D, Statistical Research and Development, SAS Institute Inc. We would like to thank Jim Bettman and Richard Johnson for their helpful comments on an earlier version of this chapter. Copies of this chapter (TS-722E) and all of the macros are available on the web http://support.sas.com/techsup/tnote/tnote_stat.html#market.

models. First, probabilistic choice models are nonlinear in the parameters, implying that the statistical efficiency of a choice design depends on an (unknown) parameter vector. This property implies the need to bring anticipated parameter values in choice designs. Second, choice design efficiency depends both on the creation of appropriate profiles and properly placing them into several choice sets. For example, in a linear design, the order of the 16 profiles in a conjoint exercise does not affect its formal efficiency, whereas the efficiency of the same 16 profiles broken into four choice sets depends critically on the grouping. Despite its limitations, linear design theory has been used to produce satisfactory choice designs for many years, drawing on readily available tables and processes. Such carefully selected linear designs are reasonable, general-purpose choice designs, but are generally not optimal in a statistical sense.

We present a general strategy for the computerized construction of efficient choice designs. This contribution can be viewed as an extension of the work of Kuhfeld, Tobias, and Garratt (1994) and of Huber and Zwerina (1996). Kuhfeld et al. recommended using a search algorithm to find efficient *linear* designs. Huber and Zwerina show how to modify *choice* designs using anticipated model parameters in order to improve design efficiency. We adapt the optimization procedure outlined in Kuhfeld et al. to the principles of choice design efficiency described by Huber and Zwerina. Our approach holds several important advantages over previous choice design strategies. It (1) optimizes the "correct" criterion of minimizing estimation error rather than following linear design principles, (2) it can generate choice designs that accommodate any anticipated parameter vector, (3) it can accommodate virtually any level of model complexity, and finally (4) it can be built using widely available software. To illustrate, we include a SAS/IML program that generates relatively simple choice designs. This program can be easily generalized to handle far more complex problems.

The chapter begins with a review of criteria for efficient choice designs and illustrates how they can be built with a computer. Then, beginning with simple designs, we illustrate how the algorithm works and how our linear design intuition must be changed when coping with choice designs. Next, we generate more complex choice designs and show how to evaluate the impact on efficiency of different design and model modifications. We conclude with a discussion of the proposed choice design approach and directions for future research.

Criteria For Choice Design Efficiency

Measure Of Choice Design Efficiency. First, we derive a measure of efficiency in choice designs from the well-known multinomial logit model (McFadden 1974). This model assumes that consumers make choices among alternatives that maximize their perceived utility, u, given by

$$u = \mathbf{x}_i\boldsymbol{\beta} + e \tag{1}$$

where $\mathbf{x}_i$ is a row vector of attributes characterizing alternative i, $\boldsymbol{\beta}$ is a column vector of K weights associated with these attributes, and e is an error term that captures unobserved variations in utility. Suppose that there are N choice sets, C_n, indexed by $n = 1, 2, \ldots, N$, where each choice set is characterized by a set of alternatives $C_n = \{x_{1n}, K, x_{J_n n}\}$. If the errors, e, are independently and identically Gumbel distributed, then it can be shown that the probability of choosing an alternative i from a choice set C_n is

$$P_{in}(\mathbf{X_n}, \boldsymbol{\beta}) = \frac{e^{\mathbf{x}_{in}\boldsymbol{\beta}}}{\sum_{j=1}^{J_n} e^{\mathbf{x}_{jn}\boldsymbol{\beta}}} \tag{2}$$

where $\mathbf{X_n}$ is a matrix that consists of J_n row vectors, each describing the characteristics of the alternatives, $\mathbf{x}_{jn}$. The vertical concatenation of the $\mathbf{X_n}$ matrices is called a choice design matrix $\mathbf{X}$.

The task of the analyst is to find a parameter estimate for $\boldsymbol{\beta}$ in Equation (2) that maximizes the likelihood given the data. Under very general conditions, the maximum likelihood estimator is consistent and asymptotically normal with covariance matrix

$$\boldsymbol{\Sigma} = (\mathbf{Z}'\mathbf{P}\mathbf{Z})^{-1} = \left[\sum_{n=1}^{N}\sum_{j=1}^{J_N} \mathbf{z}'_{jn} P_{jn} \mathbf{z}_{jn}\right]^{-1} \tag{3}$$

$$\text{where} \quad \mathbf{z}_{jn} = \mathbf{x}_{jn} - \sum_{i=1}^{J_n} \mathbf{x}_{in} P_{in} \ .$$

Equation (3) reveals some important properties of (nonlinear) choice models. In linear models, centering occurs across all profiles whereas in choice models, centering occurs within choice sets. This shows that in choice designs both the profile selection and the assignment of profiles to choice sets affects the covariance matrix. Moreover, in linear models, the covariance matrix does not depend on the true parameter vector, whereas in choice models the probabilities, P_{jn}, are functions of $\boldsymbol{\beta}$ and hence the covariance matrix. Assuming $\boldsymbol{\beta} = \mathbf{0}$ simplifies the design problem, however Huber and Zwerina (1996) recently demonstrated that this assumption may be costly. They showed that incorrectly assuming that $\boldsymbol{\beta} = \mathbf{0}$ may require from 10% to 50% more respondents than those built from reasonably anticipated parameters.

The goal in choice designs is to define a group of choice sets, given the anticipated $\boldsymbol{\beta}$, that minimizes the "size" of the covariance matrix, $\boldsymbol{\Sigma}$, defined in Equation (3). There are various summary measures of error size that can be derived from the covariance matrix (see, e.g., Raktoe, Hedayat, and Federer 1981). Perhaps the most intuitive summary measure is the average variance around the estimated parameters of a model. This measure is referred to in the literature as *A*-efficiency or its inversely related counterpart,

$$A - error = \text{trace}\,(\boldsymbol{\Sigma})/K \tag{4}$$

where K is the number of parameters. Two problems with this measure limit its suitability as an overall measure of design efficiency. First, relative *A*-error is not invariant under (nonsingular) recodings of the design matrix, i.e., design efficiency depends on the type of coding. Second, it is computationally expensive to update. A related measure,

$$D - error = |\boldsymbol{\Sigma}|^{1/K} \tag{5}$$

is based on the determinant as opposed to the trace of the covariance matrix. *D*-error is computationally efficient to update, and the ratios of *D*-errors are invariant under different codings of the design matrix. Since *A*-error is the arithmetic mean and *D*-error is the geometric mean of the eigenvalues of $\boldsymbol{\Sigma}$, they

are generally highly correlated. *D*-error thereby provides a reasonable way to find designs that are "good" on alternative criteria. For example, if *A*-error is the ultimate criterion, we can first minimize *D*-error and then select the design with minimum *A*-error rather than minimizing *A*-error directly. For these reasons, *D*-error (or its inverse, *D*-efficiency or *D*-optimality) is the most common criterion for evaluating linear designs and we advocate it as a criterion for choice designs.

Next, we discuss four principles of choice design efficiency defined by Huber and Zwerina (1996). Choice designs that satisfy these principles are optimal, however, these principles are only satisfied for a few special cases and under quite restrictive assumptions. The principles of orthogonality and balance that figured so prominently in linear designs remain important in understanding what makes a good choice design, but they will be less useful in generating one.

Four Principles Of Efficient Choice Designs. Huber and Zwerina (1996) identify four principles which when jointly satisfied indicate that a design has minimal *D*-error. These principles are orthogonality, level balance, minimal overlap, and utility balance. *Orthogonality* is satisfied when the levels of each attribute vary independently of one another. *Level balance* is satisfied when the levels of each attribute appear with equal frequency. *Minimal overlap* is satisfied when the alternatives within each choice set have nonoverlapping attribute levels. *Utility balance* is satisfied when the utilities of alternatives within choice sets are the same, i.e., the design will be more efficient as the expected probabilities within a choice set C_n among J_n alternatives approach $1/J_n$.

These principles are useful in understanding what makes a choice design efficient, and improving any principle, holding the others constant, improves efficiency. However, for most combinations of attributes, levels, alternatives, and assumed parameter vectors, it is impossible to create a design that satisfies these principles. The proposed approach does not build choice designs from these formal principles, but instead uses a computer to directly minimize *D*-error. As a result, these principles may only be approximately satisfied in our designs, but they will generally be more efficient than those built directly from the principles.

A General Method For Efficient Choice Designs

Figure 1 provides a flowchart of the proposed design approach. The critical aspect of this approach involves an adaptation of Cook and Nachtsheim's (1980) modification of the Fedorov (1972) algorithm that has successfully been used to generate efficient *linear* designs (e.g., Cook and Nachtsheim 1980, Kuhfeld et al. 1994). We will first describe the proposed choice design approach conceptually and then define the details in a context of a particular search.

The process begins by building a *candidate set*, which is a list of potential alternatives. A random selection of these alternatives is the *starting design*. The algorithm alters the starting design by exchanging its alternatives with the candidate alternatives. The algorithm finds the best exchange (if one exists) for the first alternative in the starting design. The first iteration is completed once the algorithm has sequentially found the best exchanges for all of the alternatives in the starting design. After that, the process moves back to the first alternative and continues until no substantial efficiency improvement is possible. To avoid poor local optima, the whole process can be restarted with different random starting designs and the most efficient design is selected. For example, if there are 300 alternatives in the candidate set and 50 alternatives in the choice design, then each iteration requires testing 15,000 possible exchanges, which is a reasonable problem on today's desktop computers and workstations. While there is no guarantee that it will converge to an optimal design, our experience

with relatively small problems suggests that the algorithm works very well.

To illustrate the process we first generate choice designs for simple models that reveal the characteristics of efficient choice designs. In examining these simple designs, our focus is on the benefits and the insights that derive from using this approach. Then, we apply the approach to more complex design problems, such as alternative-specific designs and designs with constant alternatives. As we illustrate more complex designs, we will focus on the use of the approach, *per se*. We provide illustrative computer code in the appendix.

Choice Design Applications

Generic Models. The simplest choice models involve alternatives described by generic attributes. The utility functions for these models consist of attribute parameters that are the same for all alternatives, for example, a common price slope across all alternatives. Generic designs are appealing because they are simple and analogous to main-effects conjoint experiments. Bunch et al. (1996) evaluate ways to generate generic choice designs and show that shifted or cyclic designs generally have superior efficiency compared with other strategies for generating main effects designs. These shifted designs use an orthogonal fractional factorial to provide the "seed" alternatives for each choice set. Subsequent alternatives within a choice set are cyclically generated. The attribute levels of the new alternatives add one to the level of the previous alternative until it is at its highest level, at which point the assignment re-cycles to the lowest level.

For certain families of plans and assuming that all coefficients are zero, these shifted designs satisfy all four principles, and thus are optimal.[†] For example, consider a choice experiment with three attributes, each at three levels, defining three alternatives in each of nine choice sets. The left-hand panel of Table 1 shows a plan using the Bunch et al. (1996) method.

In this special case, all four efficiency principles are perfectly satisfied. Level balance is satisfied since each level occurs in precisely 1/3 of the cases, and orthogonality can be confirmed by noting the all pairs of attribute levels occur in precisely 1/9 of the attributes (Addelman 1962b). There is perfect minimal overlap since each level occurs exactly once in each choice set, and finally, utility balance is trivially satisfied with the assumption that $\boldsymbol{\beta} = \mathbf{0}$. More formally, it is useful to examine the covariance matrix of the (effects-coded) parameters, reported in the first panel of Table 2. The equal variances across attributes and the zero covariances across attributes both indicate optimality.

A simple design such as this could have been built from our algorithm, although using a standard orthogonal array and cyclic permutations ensured optimality. Our next example, encompassing a model with just one interaction term, illustrates the case when a computerized search is very useful in finding a statistically efficient design.

Estimating an A×B Interaction. For the previous example with nine choice sets, let us assume that the researcher is confident that there are no A×C or B×C interactions, but the A×B interaction must be estimated. The middle panel of Table 1 shows the best design we were able to find which includes this one interaction. Note that in this design, the principle of minimal overlap on attributes A and B is violated, in that attribute levels are frequently repeated within a set. In general, interactions *require* overlap of attribute levels to produce the contrasts necessary to estimate the effects.

[†]We were not able to analytically prove this, but after examining scores of designs, we have never found more efficient designs than those that satisfy all four principles.

Figure 1
Flowchart of Algorithm for Constructing Efficient Choice Designs

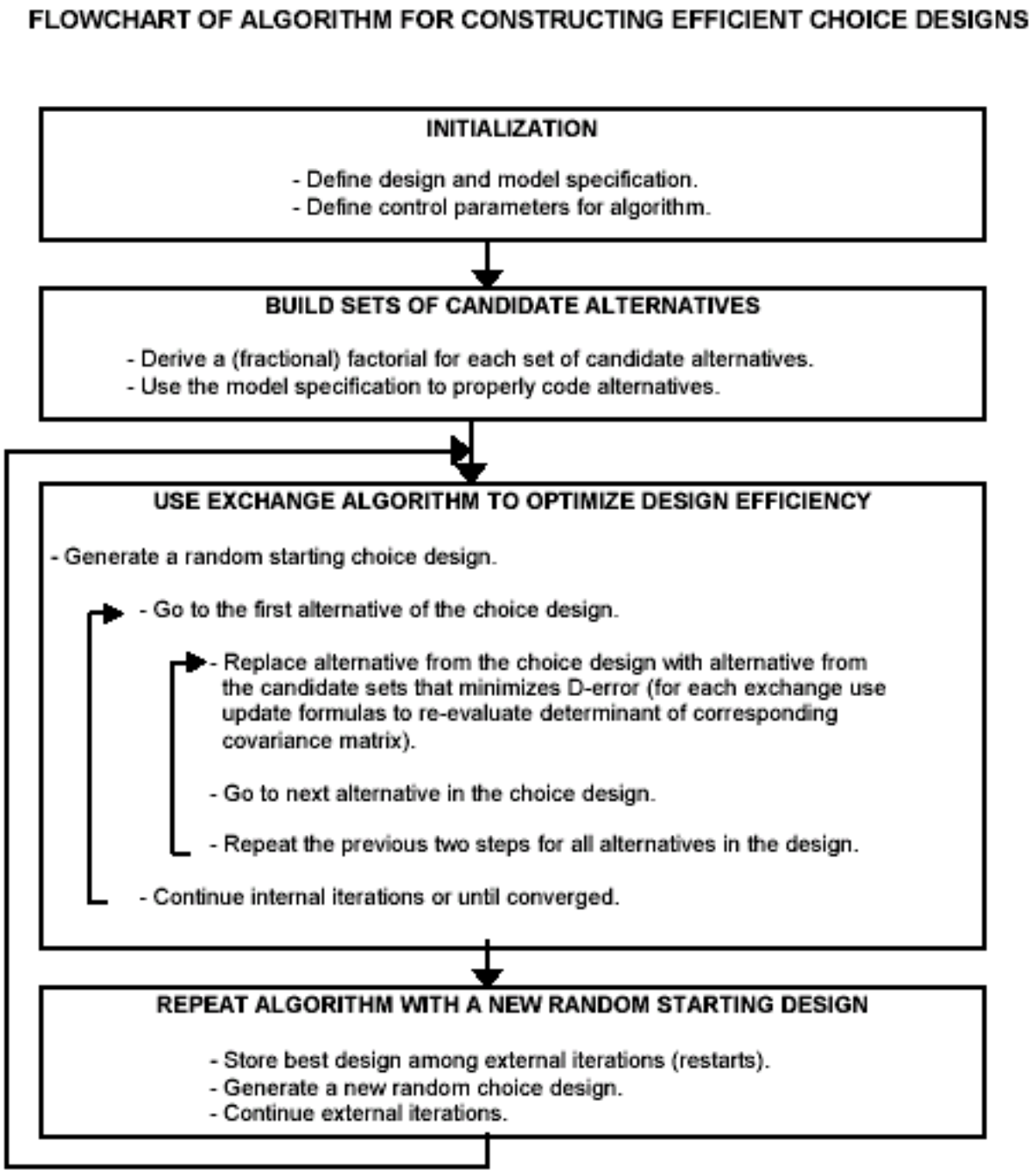


Table 1
Main Effects and A×B-Interaction Effects Choice Design

		β_0-Efficient Main-Effects Design (β_0=0 0 0 0 0 0)			β_0-Efficient Interaction-Effects (β_0=0 0 0 0 0 0 0 0 0)				β_1-Efficient Interaction-Effects (β_1=-1 0 -1 0 -1 0 0 0 0)			
Set	Alt	A	B	C	A	B	C	p(β_1)	A	B	C	p(β_1)
1	I	1	1	1	2	3	2	.495	2	1	3	.422
	II	2	2	2	2	2	3	.495	1	3	2	.422
	III	3	3	3	1	1	1	.009	3	1	1	.155
2	I	1	2	2	3	1	1	.155	3	2	2	.422
	II	2	3	3	2	2	2	.422	2	1	3	.155
	III	3	1	1	1	2	3	.422	3	3	1	.422
3	I	1	3	3	1	1	2	.042	2	2	3	.155
	II	2	1	1	1	3	1	.114	3	3	2	.422
	III	3	2	2	3	1	3	.844	2	3	3	.422
4	I	2	1	3	2	1	2	.018	2	1	2	.422
	II	3	2	1	3	3	3	.965	1	1	3	.422
	III	1	3	2	2	2	1	.018	1	2	1	.155
5	I	2	2	1	1	3	3	.245	3	1	2	.422
	II	3	3	2	3	3	2	.665	1	1	3	.155
	III	1	1	3	2	3	1	.090	3	2	1	.422
6	I	2	3	2	2	1	3	.468	1	3	3	.422
	II	3	1	3	1	2	1	.063	2	3	2	.422
	III	1	2	1	1	3	2	.468	3	2	1	.155
7	I	3	1	2	3	2	3	.665	1	2	1	.212
	II	1	2	3	3	3	1	.245	2	2	1	.576
	III	2	3	1	3	1	2	.090	1	1	2	.212
8	I	3	2	3	1	2	2	.042	1	2	3	.576
	II	1	3	1	2	3	3	.844	1	3	1	.212
	III	2	1	2	3	2	1	.114	3	1	1	.212
9	I	3	3	1	1	1	3	.114	2	2	2	.212
	II	1	1	2	2	1	1	.042	3	1	3	.576
	III	2	2	3	3	2	2	.844	2	3	1	.212
								avemaxp = .690				avemaxp = .474
		D-error(β_0) = .192						D-error(β_0) = .306				D-error(β_0) = .365
								D-error(β_1) = .630				D-error(β_1) = .399

The covariance matrix of this design, depicted in the lower half of Table 2, highlights the effects of incorporating the A×B interaction. Violating minimal overlap permits the estimation of the A×B interaction by sacrificing efficiency on the main effects of attribute A and B, reflected in higher variances of the main effects estimates (a1, a2, b1, and b2). The *D*-error of the main effect estimates increases by 24%, from .192 to .239, and the covariances across attributes A and B are no longer zero. Note also that the errors around attribute C are unchanged−they are unaffected by the A×B interaction, indicating that the algorithm was able to find a design that allowed the A×B interaction to be uncorrelated with C.

Table 2
Covariance Matrix of Main Effects and A×B-Interaction
Effects Choice Design

$\boldsymbol{\beta}_0$-Efficient Main Effects Design

	a1	a2	b1	b2	c1	c2
a1	.222	-.111	.000	.000	.000	.000
a2	-.111	.222	.000	.000	.000	.000
b1	.000	.000	.222	-.111	.000	.000
b2	.000	.000	-.111	.222	.000	.000
c1	.000	.000	.000	.000	.222	-.111
c1	.000	.000	.000	.000	-.111	.222

D-error($\boldsymbol{\beta}_0$)=.192

$\boldsymbol{\beta}_0$-Efficient Interaction Effects Design

	a1	a2	b1	b2	c1	c2	ab11	ab12	ab21	ab22
a1	.296	-.130	.019	-.019	.000	.000	-.037	.000	.000	-.019
a2	-.130	.296	-.019	.019	.000	.000	.037	.000	.000	.019
b1	.019	-.019	.296	-.130	.000	.000	.019	-.056	.000	.037
b2	-.019	.019	-.130	.296	.000	.000	-.019	.056	.000	-.037
c1	.000	.000	.000	.000	.222	-.111	.000	.000	.000	.000
c2	.000	.000	.000	.000	-.111	.222	.000	.000	.000	.000
ab11	-.037	.037	.019	-.019	.000	.000	.630	-.333	-.333	.148
ab12	.000	.000	-.056	.056	.000	.000	-.333	.556	.167	-.278
ab21	.000	.000	.000	.000	.000	.000	-.333	.167	.667	-.333
ab22	-.019	.019	.037	-.037	.000	.000	.148	-.278	-.333	.630

D-error($\boldsymbol{\beta}_0$) of main effects = .239
D-error($\boldsymbol{\beta}_0$) of all effects = .306

There are several important lessons from this simple example. First, it illustrates that a design that is "perfect" for one model may be far from optimal for a slightly different model. Adding one interaction strongly altered the covariance matrix, so efficient designs generally violate the formal principles. Second, the example shows that estimating new interactions is not without cost; being able to estimate one interaction increased by 24% the error on the main effects. Finally, the trade-off of efficiency with estimability demonstrates one of the primary benefits of this approach−it allows the analyst to understand the efficiency implications of changes in the design structure and/or model specification. This use of the approach will be illustrated again in the context of more complex choice designs.

The Impact Of Non-Zero Betas. The preceding discussion has assumed that the true parameters are zero. This assumption is justified when there is very little information about the model parameters; however, typically the analyst has some information on the relative importance of attributes or the relative value of their levels (Huber and Zwerina 1996). To show the potential gain that can come from nonzero parameters, assume that the anticipated partworths of the main effects for the three level attributes discussed previously are not 0, 0, 0, but -1, 0, 1, while the A×B-interaction effect continues to have zero parameters.[‡] Calling the new parameter vector $\boldsymbol{\beta}_1$ to distinguish it from the zero parameter

[‡]We assume for simplicity that the interaction has parameter values of zero. Note, this also produces minimal variance of estimates around zero, implying greatest power of a test in the region of that null hypothesis.

Table 3
Attributes/Levels for an Alternative-Specific Choice Experiment

	Alternative-Specific Levels		
Attributes	Coke	Pepsi	RC Cola
Price per case	$5.69	$5.39	$4.49
	$6.89	$5.99	$5.39
	$7.49	$6.59	$5.99
Container	12 oz cans	12 oz cans	12 oz cans
	10 oz bottle	10 oz bottle	16 oz bottle
	16 oz bottle	18 oz bottle	22 oz bottle
Flavor	Regular	Regular	Regular
	Cherry Coke	Pepsi Lite	Cherry
	Diet Coke	Diet Pepsi	Diet

vector, $\boldsymbol{\beta}_0$, the third panel of Table 1 displays the efficient design using these parameters. This new design has a D-error($\boldsymbol{\beta}_1$) of 0.399. However, if instead we had used the design in the center panel, its error given $\boldsymbol{\beta}_1$ is true would have been .630, implying that 37% (1 - .399/.630) fewer respondents are needed for the "utility balanced" over the "utility neutral" design.

Comparing the last two panels in Table 1 reveals how the algorithm used the anticipated nonzero parameters to produce a more efficient design. As an index of utility balance, we calculated the average of the maximum within-choice-set choice probabilities (avemaxp). The smaller this index the harder is the average choice task and the greater is "utility balance." We can see, by using $\boldsymbol{\beta}_1$, the new design is more utility balanced than the previous design, which results in an average maximum probability of .474 compared with one of .690. We also see that the increase in utility balance sacrifices somewhat the three formal principles, reflected in an increase of D-error($\boldsymbol{\beta}_0$) from .306 to .365. The new design does not have perfect orthogonality, level balance, utility balance, or minimal overlap, but it is more efficient than any design that is perfect on any of those criteria.

More Complex Choice Designs. The proposed algorithm is very general and can be applied to virtually any level of design complexity. We will use it next to generate an alternative-specific choice design, which has a separate set of parameters for each alternative. Suppose, the researcher is interested in simulating the market behavior of three brands, Coke, Pepsi, and RC Cola, with the attribute combinations shown in Table 3.

This kind of choice experiment, which we call a market emulation study, is quite different from the generic choice design presented previously. In a market emulation study, emphasis is on predicting the impact of brand, flavor, and container decisions in the context of a realistic market place offering. What this kind of study gains in realism, it loses in the interpretability of its results. For example, since each brand only occurs at specific prices, it is much harder to disentangle the independent effects of brand and price. These designs are, however, useful in assessing the managerially critical question of the impact of, say, a 60 cent drop in the price of Coke's 16 ounce case in a realistic competitive configuration.

Since we assume that the impact of price depends on the brand to which it is attached, it is important that the impact of price be estimable within each brand.[§] Further, let us assume that the reaction to price additionally depends on the number of ounces, so that it is necessary to estimate the brand×price×container interaction. Using standard ANOVA-coding, these assumptions require four main effects (brand, flavor, container, and price for 8 *df*), four two-way interactions (brand×price, brand×flavor, brand×container, and price×container for 16 *df*), and one three-way interaction (brand×price×container for 8 *df*), resulting in a total of 32 parameters.[¶]

Suppose we want to precisely estimate these effects with a choice design consisting of 27 choice sets each composed of three alternatives.[*] The candidate set of alternatives comprises the $3^4 = 81$ possible alternatives, and the initial design is a random selection from these. The algorithm exchanges alternatives between the candidate set and the starting design until the efficiency gain becomes negligible. In the example with 27 choice sets and 32 parameters, *D*-error is .167. This statistic provides a baseline for evaluating other related designs, which we will generate in the following section.

Evaluating Design Modifications. The proposed approach can be used to evaluate design modifications. Typically, efficiency is meaningful within a relatively narrow family of designs, limited to a particular attribute structure, model specification, and number of alternatives per choice set. For many applications, optimizing a design within such a narrow design family is too restrictive. Most analysts are not tightly bound to a particular number of alternatives per choice set or even particular attributes, but are interested in exploring the impact of changes in these specifications on the precision of the parameter estimates. We will demonstrate how comparing designs across design families allows a reasoned trade-off of design structure against estimation precision.

Consider the following questions an analyst might ask concerning the alternative-specific choice design just presented.

1. How much does efficiency increase if 54 choice sets are used instead of two replications of 27 choice sets?
2. What is the efficiency loss if each of the brands (Coke, Pepsi, RC) must be present in a choice set?
3. What is the gain in efficiency if a fourth alternative is added to each choice set?
4. What happens to efficiency if this fourth alternative is constant (e.g., "keep on shopping")?

The first question assesses the benefit of building a design with 54 choice sets rather than using the original 27 choice sets twice. As Table 4 shows, specifying twice as many choice sets produces a *D*-error of .079 compared with .084 (=.167/2) for two independent runs of the 27 choice set design. This relatively small 6% benefit in efficiency indicates that the original 27 choice set design, while highly fractionated, appears to have suffered little due to this fact.

The second question evaluates the impact of constraints on the choice sets that respondents face. The original design often paired the same brand against itself within a choice set. For example, a choice

[§]The assumption that price has a different impact depending on the brand is testable. The ability to make that test is just one of the advantages of these choice designs.

[¶]We need the fourth two-way interaction, price×container, to be able to estimate the three-way interaction brand×price×container. Of course, there are many other ways of coding a design.

[*]The appendix contains a SAS/IML program that performs the search for this design. Focusing on the principles of the algorithm, the program was deliberately kept simple, specific, and small. A general macro for searching for choice designs, `%ChoicEff`, is documented in Kuhfeld (2005) starting on pages 597 and 600. See page 363 for an example.

Table 4
Impact of Design Modifications on *D*-Error

Design Modification	*D*-error	Efficiency per Choice Set	Comments
27 sets, 3 alternatives per set.	.167	100%	Original design.
Double the number of sets.	.079	106%	Limited benefit from doubling the number of sets.
Require each alternative to contain one of each brand.	.175	95%	Shows minor cost of constraining a design.
Add a fourth alternative.	.144	116%	Diminishing returns from adding additional alternatives.
Fourth alternative is constant.	.195	86%	Design is less efficient because constant alternative is chosen 25% of the time.

set with Coke in a 12 oz bottle for $5.69 per case might include Coke in a 16 oz bottle for $7.49 per case. For managerial reasons it might be desirable to have each brand (Coke, Pepsi, RC) represented in every set of three alternatives. To examine the cost of this constraint, Coke is assigned to the first alternative, Pepsi to the second alternative, and RC to the third alternative within each of the 27 choice sets. With this constraint, the *D*-error is .175. This relatively moderate decrease in efficiency of 5% should be acceptable if there are managerially-based reasons to constrain the choice sets.

The third question investigates the benefits of adding a fourth alternative to each choice set. This change increases by 25% the number of alternatives, although the marginal effect of an additional alternative should not be as great. With this modification, *D*-error becomes .144, producing a 16% efficiency gain over three alternatives per choice set. The decision whether to include a fourth alternative now pits the analyst's appraisal of the trade-off between the value of this 16% efficiency gain and the cost in respondent time and reliability.

What happens if this fourth alternative is common and constrained to be constant in all choice sets? With a constant alternative, respondents are not forced to make a choice among undesirable alternatives. Moreover, a constant alternative permits an estimate of demand volume rather than just market shares (Carson et al. 1994). A constant alternative can take many forms, ranging from the respondent's "current brand," to an indication that "none are acceptable," or simply "keep on shopping." While constant alternatives are often added to choice sets, little is known about the efficiency implications of this practice. To create designs with a constant alternative, this alternative must be added to the candidate set. Also, a model with a constant alternative has one more parameter. Comparing a design with a constant alternative to one without, it is necessary to calculate *D*-error with respect to the original 32 parameters using the corresponding submatrix of $\mathbf{\Sigma}$.

Adding a constant alternative to the original design increases the *D*-error of the original 32 parameters by 17% and is nearly 35% worse than allowing the fourth alternative to be variable. Some part of this loss in efficiency is due to the one additional degree of freedom from the constant alternative. A

larger part is due to the efficiency lost when respondents are assumed to select the constant alternative. Every time it is chosen, one obtains less information about the values of the other parameters. In this case, the assumption that $\boldsymbol{\beta} = \mathbf{0}$ is not benign, as it assumes the constant alternative, along with all others in the four-option choice sets, will be chosen 25% of the time. We can reduce the efficiency cost to the other parameters by using a smaller $\boldsymbol{\beta}$ for the constant alternative, reflecting the assumption that it will be chosen less often.

In summary, the analysis suggests that adding a constant alternative to a three-alternative choice set can degrade the precision of estimates around the original parameters. Two caveats are important. First, this result will not always occur. We have found some highly fractionated designs where a constant alternative adds to the resolution of the original design. Second, there are studies where a major goal is the estimation of the constant alternative; in that case "oversampling" the constant ensures that its coefficient will be known with greater precision.

An important lesson across these four examples is that one cannot rely on heuristics to guide design strategies, ignoring statistical efficiency. It is generally necessary to test specific design strategies, given anticipated model parameters, to find a good choice design.

Evaluating Model Modifications. The proposed approach can be used to assess modifications of the model specification. This allows one, for example, to estimate the cost of "assumption insurance," i.e., building a design that is robust to false assumptions. Often we assume that factors are independent; for example, that the utility of price does not depend on brand or container. In many instances this assumption would be better termed a "presumption" in that if it is wrong, the estimates are biased, but there is no way to know given the design. Assessing the cost of assumption insurance involves four steps:

1. Find the best design for the unrestricted model (possibly including interactions).
2. Find the best design for the restricted model.
3. Evaluate *D*-error for that unrestricted design under the restricted model.
4. Evaluate *D*-error for the best design for the restricted model.

The cost of assumption insurance is the percent difference between steps 3 and 4, reflecting the loss of efficiency of the core parameters for the two designs. We illustrate how to assess this cost for a design that permits the price term to interact with brand and container versus one that assumes they are independent. To simplify the example, we take the same case as before, but assume that price is a linear rather than a categorical variable.[†]

The first step involves finding an efficient design with all price interactions with brand and brand×container estimable. This unrestricted model has 7 *df* for main effects (two for brand, two for container, two for flavor, and one for price), 12 *df* for two-way interactions (brand×price, brand×flavor, brand×container, container×price), and 4 *df* for the three-way interaction (brand×container×price). An efficient design for this unrestricted model has a *D*-error of .148. If this design is used for a restricted model in which price does not interact (7 *df* for main effects and 8 *df* for two-way interactions) then

[†]Substituting a linear price term for a three-level categorical one has two immediate implications. First, any change in coding results in quite different absolute values of *D*-error. Second, in optimizing a linear coding for price, the search routine will try to eliminate alternatives with the middle level of price within brand. This focus on extremes is appropriate given the linear assumption, but, may preclude estimation of quadratic effects.

D-error drops to .118. The critical question is how much better still can one do by searching for the best design in the 15-parameter restricted model. The best design we find has a *D*-error of .110. Thus, assumption insurance in this case imposes a 6% (1 - .110/.118) efficiency loss, a reasonable cost given that prices will often interact with brands and containers.

To summarize, the search routine allows estimates of the cost in efficiency of various design modifications and even changes in the model specification. Again, the important lesson is not the generalizations from the results of these particular examples, but rather an understanding of how these and similar questions can be answered in the context of any research study.

How Good Are These Designs? The preceding discussion has shown that our adaptation of the modified Fedorov algorithm can find estimable choice designs and answer a variety of useful questions. We still need to discuss the question, how close to optimal are these designs? The search is nonexhaustive, and there is no guarantee that the solutions are optimal or even nearly so. For some designs, such as the alternative-specific one shown previously, we can never be completely certain that the search process does not consistently find poor local optima. However, one can achieve some confidence from the pattern of results based on different random restarts; similar efficiencies emerging from different random starts indicate robustness of the resultant designs. An even stronger test is to assess efficiencies of the search process in cases where an optimal solution is known. While this cannot be done generally, we can test the absolute efficiency of certain symmetric designs, where the optimal design can be built using formal methods. We illustrated this kind of design in the three attribute, three level, three alternative, nine choice set ($3^3/3/9$) design discussed earlier, and found that the search routine was not able to find a better design. Now, we ask how good are our generated designs relative to three optimal designs: the design mentioned previously and two corresponding, but bigger designs−$4^4/4/16$ and a $5^5/5/25$ generic design.

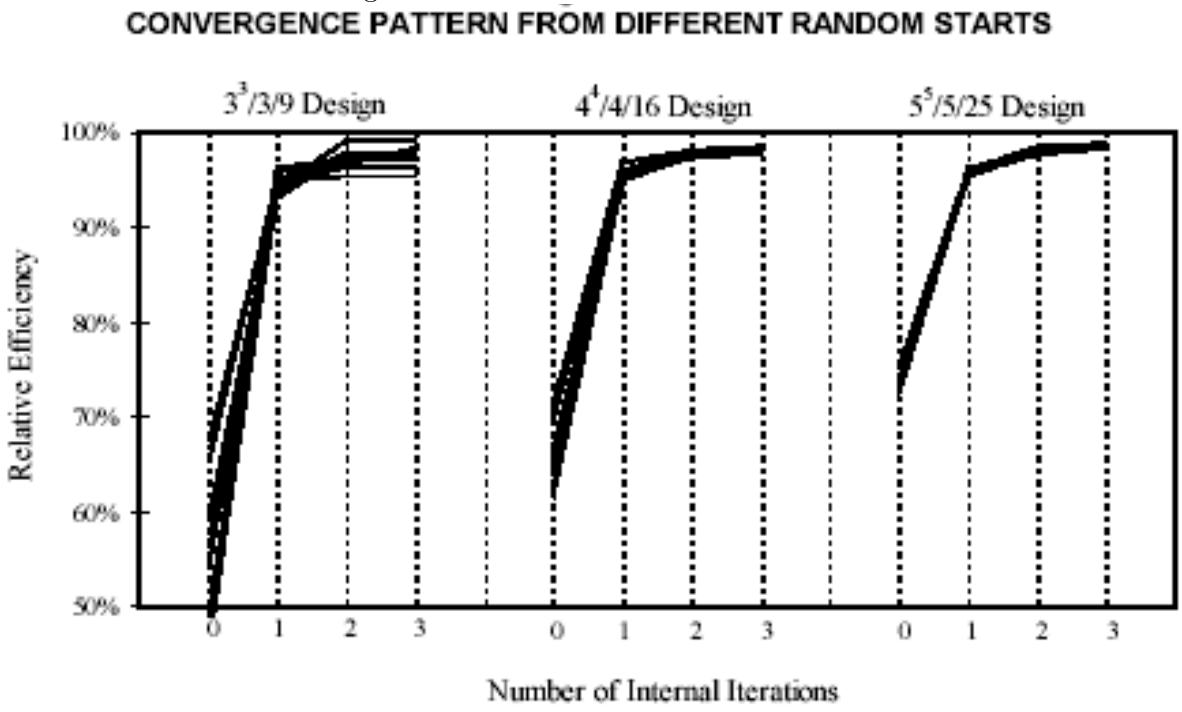
For these types of designs we apply the proposed algorithm and compare our designs with the analytically generated ones. For each design, we used ten different random starts and three internal iterations. Figure 2 displays the impact of efficiency on different starting points and different numbers of internal iterations.

Figure 2 reveals important properties of the proposed algorithm. After the first iteration, the algorithm finds a choice design with about 90% relative efficiency, after a few more iterations, relative efficiencies approach 95%-99%. Further, this property appears to be independent of any initial starting design−the process converges just as quickly from a random start as from a rational one. These encouraging properties suggest important advantages for the practical use of the approach. First, in contrast to Huber and Zwerina (1996), the process does not require a rational starting design (which may be difficult to build). Second, since the process yields very efficient designs after only one or two iterations, most practical problems involving even large choice designs can be accommodated.

Conclusions

We propose an adaptation of the modified Fedorov algorithm to construct statistically efficient choice designs with standard personal computers. The algorithm generates efficient designs quickly and is appropriate for all but the largest choice designs. The approach is illustrated with a SAS/IML program. SAS has the advantage of a general model statement that facilitates the building of choice designs with different model specifications. The cost of using SAS/IML software, however, is that the algorithm generally runs slower than a program developed in, for example, PASCAL or C.

Figure 2
Convergence Pattern From Different Random Starts



There are three major advantages of using a computer to construct choice designs rather than deriving them from formal principles. First, computers are the only way we know to build designs that allow one to incorporate anticipated model parameters. Since the incorporation of this information can increase efficiency by 10% to 50% (see Huber and Zwerina 1996), this benefit alone justifies the use of computer search routines to find efficient choice designs.

The second advantage is that one is less restricted in design selection. Symmetric designs may not reflect the typically asymmetric characteristics of the real market. The adaptability of computerized searches is particularly important in choice studies that simulate consumer choice in a real market (Carson et al. 1994). Moreover, the process we propose allows the analyst to generate choice designs that account for any set of interactions, or alternative-specific effects of interest and critical tests of these assumptions. We illustrated a market emulation design that permits brand to interact with price, container, and flavor and can test the three-way interaction of brand by container by price. This pattern of alternative-specific effects would be very hard to build with standard designs, but it is easy to do with the computerized search routine by simply setting the model statement. The process can handle even more complex models, such as availability and attribute cross effects models (Lazari and Anderson 1994).

Finally, the ability to assess expected errors in parameters permits the researcher to examine the impact of different modifications in a given design, such as adding more choice sets or dropping a level from a factor. Most valuable, perhaps, is the ability to easily test designs for robustness. We provide one example of assumption insurance, but others are straightforward to generate. What happens to the efficiency of a design if there are interactions, but they are not included in the model statement? What kind of model will do a good job given a linear representation of price, but will also permit a test of curvature? What happens to the efficiency of the design if one's estimate of $\boldsymbol{\beta}$ is wrong?

There are several areas in which future research is needed. The first of these involves studies of the search process per se. We chose the modified Fedorov algorithm because it is robust and runs fast enough on today's desktop computers. As computing power increases, more exhaustive searches should be evaluated. For extremely large problems, faster and less reliable algorithms may be appropriate. Furthermore, while the approach builds efficient choice designs for multinomial logit models, efficiency issues with respect to other models, for example, nested logit and probit models, have yet to be explored.

A second area in which research would be fruitful involves the behavioral impact of different choice designs. The evaluations of our designs all implicitly assumed that the error level is constant regardless of the design. Many choice experiments use relatively small set sizes and few attributes reflecting an implicit recognition that "better" information comes from making the choice less complex. However, from a statistical perspective it is easy to show that smaller set sizes reduce statistical efficiency. In one example, we demonstrated that increasing the number of alternatives per choice set from three to four can increase efficiency by 16%. This gain depends on the assumption that respondent's error levels do not change. If they do increase, then that 16% percent gain might be lessened or even reversed. Thus, there is a need for a series of studies measuring respondents' error levels to tasks at different levels of complexity. Also, it is important to measure the degree of correspondence between the experimental tasks and the actual market behavior, choice experiments are intended to simulate. Such information is critical for correct trade-offs between design efficiency, measured here, and survey effectiveness, measured in the marketplace.

The purpose of this article is to demonstrate the important advantages of a flexible computerized search in generating efficient choice designs. The proposed adaptation of the modified Fedorov algorithm solves many of the practical problems involved in building choice designs, thus enabling more researchers to conduct choice experiments. Nevertheless, we want to emphasize that it does not preclude traditional design skills; they remain critical in determining the model specification and in assessing the choice designs produced by the computerized search.

Appendix

SAS/IML Code for the Proposed Choice Design Algorithm

The SAS code shows a simple implementation of the algorithm. In this example, the program finds a design with 27 choice sets and three alternatives per set. There are four attributes (brand, price, container, and flavor) each with three levels. A design is requested in which all main effects, the two-way interactions between brand and the other attributes, the two-way interaction between container and price, and the brand by price by container three-way interaction are estimable. Here, the parameters are assumed to be zero, but could be easily changed by setting other values.

A computer that evaluated all possible ($81^{81}/3!27! = 5 \times 9 \times 10^{125}$) designs would take numerous billion years. Instead, we use the modified Fedorov algorithm, which uses the following heuristic: find the best exchange for each design point given all of the other candidate points. With 81 candidate alternatives, 27 choice sets, 3 alternatives per set, (say) 3 internal iterations, and 2 random starts, $81 \times 27 \times 3 \times 3 \times 2 = 39,366$ exchanges must be evaluated. The algorithm tries to maximize $|\mathbf{X}'\mathbf{X}|$ rather than minimizing $|(\mathbf{X}'\mathbf{X})^{-1}|$ (note that $|(\mathbf{X}'\mathbf{X})^{-1}| = |\mathbf{X}'\mathbf{X}|^{-1}$). Each exchange requires then the evaluation of a matrix determinant, $|\mathbf{X}'\mathbf{X}|$. Fortunately, we do not have to evaluate this determinant from scratch for each exchange since $|\mathbf{X}'\mathbf{X} + \mathbf{x}'\mathbf{x}| = |\mathbf{X}'\mathbf{X}||I + \mathbf{x}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{x}'|$ (Mardia et al. 1979). Each exchange evaluates a quadratic form, and in this example with three alternatives per choice set, the determinant of a 3×3 matrix. It should also be noted that this algorithm can handle a rank-deficient covariance matrix by operating on $|\mathbf{X}'\mathbf{X} + I\epsilon|$, where ϵ is a small number. This eliminates zero determinants so that less-than-full-rank codings and singular starting designs are not a problem. With these short cuts, one iteration required about 30 seconds on an ordinary 486 PC, implying that the algorithm is reasonable for many marketing contexts.

This appendix is provided simply to show the algorithm for those who might wish to implement or better understand it. If you want to use the algorithm, use the `%ChoicEff` autocall SAS macro documented in Kuhfeld (2005) starting on pages 597 and 600. See page 363 for an example. The `%ChoicEff` is much larger and more full-featured than the code shown in this appendix.

```
/*------------------------------Initial Set Up------------------------------*/
%let beta  = 0 0 0 0 0 0 0 0              /* 8 main effects                   */
             0 0 0 0 0 0 0 0              /* brand x price, brand x container,*/
             0 0 0 0                      /* brand x flavor,                  */
             0 0 0 0                      /* price x container interactions   */
             0 0 0 0 0 0 0 0;             /* brand x price x container        */
%let nalts = 3;                           /* Number of alternatives           */
%let nsets = 27;                          /* Number of choice sets            */

proc plan ordered;                        /* Create candidate alternatives    */
   factors brand=3 price=3 contain=3 flavor=3 / noprint;
   output out=candidat;
   run;

proc transreg design data=candidat;       /* Code the candidate alternatives  */
model class(brand price contain flavor brand*price brand*contain
            brand*flavor contain*price brand*contain*price / effects);
   output out=tmp_cand;
   run;

proc contents p data=tmp_cand(keep=&_trgind); run;

/*-----------------------Begin Efficient Design Search----------------------*/
proc iml; file log;
   use tmp_cand(keep=&_trgind);           /* Identify candidate set for input */
   read all into cand;                    /* Read candidate set into IML      */
   utils  = exp(cand * {&beta}`);         /* exp(alternative utilities)       */
   np     = 1 / ncol(cand);               /* Exponent applied to determinant  */
   imat   = i(&nalts);                    /* Identity matrix                  */
   nobs   = &nsets # &nalts;              /* Total n of alts in choice design */
   ncands = nrow(cand);                   /* Number of candidates             */
   fuzz   = i(ncol(cand)) # 1e-8;         /* X`X ridge factor, avoid singular */

   start center(x, exputil);              /* Probability centering subroutine */
      do i = 1 to nrow(x) / &nalts;       /* Do for each choice set           */
         k = (i-1)#&nalts+1 : i#&nalts;   /* Choice set index vector          */
         p = exputil[k,]; p = p / sum(p); /* Probability of choice            */
         z = x[k,];                       /* Get choice set                   */
         x[k,] = (z - j(&nalts,1,1) *     /* Center choice set, absorb p's    */
                  p` * z) # sqrt(p);
         end;
      finish;

   /*---------------Create Designs With Different Random Starts--------------*/
   do desnum = 1 to 2;                    /* Number of designs to create      */

      indvec  = ceil(ncands *             /* Random index vector (indvec)     */
                uniform(j(1, nobs, 0)));  /* into candidates                  */
      des     = cand[indvec,];            /* Initial random design            */
```

```
run center(des, utils[indvec,]);    /* Probability center              */
currdet = det(des‘ * des);          /* Initial determinants, eff’s     */
maxdet  = currdet; oldeff = currdet ## np; fineff = oldeff;
if fineff <= 0 then err = .; else err = 1 / fineff;
put /// +8 ’Design   Iteration  D-Efficiency       D-Error’ /
        +8 ’-----------------------------------------’;
put +6 desnum 6. 0 10. +6 fineff best12. +2 err best12.;

/*-------------------------Internal Iterations-------------------------*/
do iter = 1 to 8 until(converge);   /* Iterate until convergence       */

   /*---------Consider Replacing Each Alternative in the Design---------*/
   do desi = 1 to nobs;             /* Process each alt in design      */
      ind = ceil(desi / &nalts);    /* Choice set number               */
      ind = (ind - 1) # &nalts + 1  /* Choice set index vector         */
            : ind # &nalts;
      besttry = des[ind,];          /* Store current choice set        */
      des[ind,] = 0;                /* Remove current choice set       */
      do i = 0 to 100 until(d ## np > 1e-8);
         xpx = des‘*des + i#i*fuzz; /* X‘X, ridged if necessary        */
         d   = det(xpx);            /* Determinant, if 0 then X‘X will */
         end;                       /* be ridged to make it nonsingular */
      xpxinv = inv(xpx);            /* Inverse (all but current set)   */
      indcan = indvec[,ind];        /* Indvec for this choice set      */
      alt = mod(desi-1, &nalts) + 1;/* Alternative number              */

      /*-----------------Loop Over All of the Candidates----------------*/
      do candi = 1 to ncands;       /* Consider each candidate         */
         indcan[,alt] = candi;      /* Update indvec for this candidate */
         tryit = cand[indcan,];     /* Candidate choice set            */
         run center(tryit,          /* Probability center              */
                    utils[indcan,]);
         currdet = d *              /* Update determinant              */
                   det(imat + tryit * xpxinv * tryit‘);

         /*------------Store Results When Efficiency Improves-----------*/
         if currdet > maxdet then do;
            maxdet        = currdet;/* Best determinant so far         */
            indvec[,desi] = candi;  /* Indvec of best design so far    */
            besttry       = tryit;  /* Best choice set so far          */
            end;
         end;

      des[ind,] = besttry;          /* Update design with new choice set*/
      end;

   /*----------Evaluate Efficiency/Convergence, Report Results----------*/
   neweff = maxdet ## np;           /* Newest efficiency               */
   converge = ((neweff - oldeff) /  /* Less than 1/2 percent           */
```

```
            max(oldeff,1e-8) < 0.005); /* improvement means convergence    */
        oldeff = neweff;               /* Store for use in next iteration  */
        fineff = det(des' * des) ## np;  /* Efficiency at end of iteration   */
        if fineff <= 0 then err = .; else err = 1 / fineff;
        put +12 iter 10. +6 fineff best12. +2 err best12.;
        end;

     /*----Store Efficiency, Index of Efficient Design, Covariance Matrix----*/
     final = final // (shape(desnum || fineff, nobs, 2) || indvec');
     cov = cov // (shape(desnum || fineff, ncol(des), 2) ||
                 sweep(des' * des, 1 : ncol(des)));
     end;

  /*----------------------Write Results to SAS Data Sets---------------------*/
  create cov     var({design effic &_trgind});   append from cov;
  create results var({design effic index   });   append from final;
  quit;

/*-------------Store Actual Design Points, Using Indices from IML-------------*/
data results; set results; i=index; n=_n_; set candidat point=i; run;
proc sort; by descending effic n; run;    /* Put most eff design first        */
proc print; run;                          /* Print designs, best to worst     */
```

Discrete Choice

Warren F. Kuhfeld

Abstract

Discrete choice modeling is a popular technique in marketing research, transportation, and other areas, and is used for understanding people's stated choice among alternatives. We will discuss designing a choice experiment, preparing the questionnaire, inputting and processing the data, performing the analysis, and interpreting the results.*

Introduction

This chapter shows you how to use the multinomial logit model (McFadden, 1974; Manski and McFadden, 1981; Louviere and Woodworth, 1983) to investigate consumer's stated choices. The multinomial logit model is an alternative to full-profile conjoint analysis and is extremely popular in marketing research (Louviere, 1991; Carson et. al., 1994). Discrete choice, using the multinomial logit model, is sometimes referred to as "choice-based conjoint." However, discrete choice uses a different model from full-profile conjoint analysis. Discrete choice applies a nonlinear model to aggregate choice data, whereas full-profile conjoint analysis applies a linear model to individual-level rating or ranking data.

Several examples are discussed.† There is also a very basic introductory example starting on page 73 in the introduction to experimental design chapter, which starts on page 47. Be sure to read the design chapter before proceeding to the examples in this chapter.

- The candy example (page 144) is a first, very simple example that discusses the multinomial logit model, the input data, analysis, results, and computing probability of choice.
- The fabric softener example (page 156) is a small, somewhat more realistic example that discusses designing the choice experiment, randomization, generating the questionnaire, entering and processing the data, analysis, results, probability of choice, and custom questionnaires.
- The first vacation example (page 184) is a larger, symmetric example that discusses designing the choice experiment, blocks, randomization, generating the questionnaire, entering and processing the data, coding, and alternative-specific effects.

*Copies of this chapter (TS-722F) and all of the macros are available on the web http://support.sas.com/techsup/tnote/tnote_stat.html#market. This document would not be possible without the help of Randy Tobias who contributed to the discussion of experimental design and Ying So who contributed to the discussion of analysis. Randy Tobias wrote PROC FACTEX and PROC OPTEX. Ying So wrote PROC PHREG. Warren F. Kuhfeld wrote PROC TRANSREG and the macros.

†All of the example data sets are artificially generated.

- The second vacation example (page 229) is a larger, asymmetric example that discusses designing the choice experiment, blocks, blocking an existing design, interactions, generating the questionnaire, generating artificial data, reading, processing, and analyzing the data, aggregating the data to save time and memory.
- The brand choice example (page 261) is a small example that discusses the processing of aggregate data, the mother logit model, and the likelihood function.
- The food product example (page 283) is a medium sized example that discusses asymmetry, coding, checking the design to ensure that all effects are estimable, price cross effects, availability cross effects, interactions, overnight design searches, modeling subject attributes, and designs when balance is of primary importance.
- The drug allocation example (page 345) is a small example that discusses data processing for studies where respondents potentially make multiple choices.
- The chair example (page 363) is a purely generic-attributes study, and it uses the `%ChoicEff` macro to create experimental designs.
- The next example section (page 383) shows how to improve an existing design and augmenting a design with some choice sets are fixed in advance.
- The last example section (page 397) discusses partial-profile designs and designs with restrictions. Also see page 700 for an example of a choice design with a complicated set of restrictions.

This chapter relies heavily on a number of macros and procedures.

- We use the `%MktRuns` autocall macro to suggest design sizes. See page 740 for documentation.
- We use the `%MktEx` autocall macro to generate most of our experimental designs. See page 667 for documentation.
- We use the `%MktEval` autocall macro to evaluate our designs. See page 663 for documentation.
- We use the `%ChoicEff` autocall macro to generate certain specialized choice designs. We also use it to evaluate our choice designs before collecting data. See page 600 for documentation.
- We use the autocall macros `%MktKey`, `%MktRoll`, `%MktMerge`, and `%MktAllo` to prepare the data and design for analysis. See pages 710, 735, 723, and 632 for documentation.
- We use PROC TRANSREG to do all of our design coding.
- We use the `%PhChoice` autocall macro to customize our printed output. This macro uses PROC TEMPLATE and ODS (Output Delivery System) to customize the output from PROC PHREG, which fits the multinomial logit model. See page 748 for documentation.
- The `%MktBal` macro can be used to make perfectly balanced designs. See page 635 for documentation.
- The `%MktBlock` macro can be used to block a linear or choice design. See page 638 for documentation.
- The `%MktDups` macro can be used to search for duplicate runs or choice sets. See page 655 for documentation.

- The `%MktLab` macro can be used to assign different variable names, labels and levels to experimental designs and to add an intercept. See page 712 for documentation.
- The `%MktOrth` macro can be used to list orthogonal experimental designs that the `%MktEx` macro can produce. See page 725 for documentation.
- The `%MktPPro` macro can be used to make certain partial-profile choice designs. See page 731 for documentation.

All of these macros are distributed with SAS 9.1 as autocall macros (see page 597 for more information on autocall macros), however, you should get the latest versions of the macros from the web http://support.sas.com/techsup/tnote/tnote_stat.html#market.

Experimental Design

Experimental design is a fundamental component of choice modeling. A discrete choice study uses experimental design to create sets of products, and subjects choose a product from each set. Often, the most challenging part of the entire study is making the design. There are many examples of making choice designs in this chapter. Before you read them, be sure to read the design chapter beginning on page 47. As you become more comfortable with the ideas in that chapter, you should also look at the other two design chapters beginning on pages 99 and 121.

Customizing the Multinomial Logit Output

The multinomial logit model for discrete choice experiments is fit using the SAS/STAT procedure PHREG (proportional hazards regression), with the `ties=breslow` option. The likelihood function of the multinomial logit model has the same form as a survival analysis model fit by PROC PHREG. The output from PROC PHREG is primarily designed for survival-analysis studies. Before we fit the multinomial logit model with PROC PHREG, we can customize the output to make it more appropriate for choice experiments. We will use the autocall macro `%PhChoice` macro. See page 597 for information on autocall macros. You can run the following macro to customize PROC PHREG output.

```
%phchoice(on)
```

The macro uses PROC TEMPLATE and ODS (Output Delivery System) to customize the output from PROC PHREG. Running this code edits the templates and stores copies in `sasuser`. These changes will remain in effect until you delete them, so typically, you only have to run this macro once. Note that these changes assume that each effect in the choice model has a variable label associated with it, so there is no need to print variable names. If you are coding with PROC TRANSREG, this will usually be the case. To return to the default output from PROC PHREG, run the following macro.

```
%phchoice(off)
```

See page 748 for more information on the `%PhChoice` macro.

Candy Example

We begin with a very simple introductory example. In this example, we will discuss the multinomial logit model, data input and processing, analysis, results, interpretation, and probability of choice. Many aspects of this example, the experimental design in particular, are simpler than almost all realistic choice studies. Still, it is useful to start with a simple choice study with no experimental design issues to consider. In this example, each of ten subjects was presented with eight different chocolate candies and asked to choose one. The eight candies consist of the 2^3 combinations of dark or milk chocolate, soft or chewy center, and nuts or no nuts. Each subject saw all eight candies and made one choice. Experimental choice data such as these are typically analyzed with a multinomial logit model.

The Multinomial Logit Model

The multinomial logit model assumes that the probability that an individual will choose one of the m alternatives, c_i, from choice set C is

$$p(c_i|C) = \frac{\exp(U(c_i))}{\sum_{j=1}^{m} \exp(U(c_j))} = \frac{\exp(\mathbf{x}_i\boldsymbol{\beta})}{\sum_{j=1}^{m} \exp(\mathbf{x}_j\boldsymbol{\beta})}$$

where $\mathbf{x}_i$ is a vector of alternative attributes and $\boldsymbol{\beta}$ is a vector of unknown parameters. $U(c_i) = \mathbf{x}_i\boldsymbol{\beta}$ is the utility for alternative c_i, which is a linear function of the attributes. The probability that an individual will choose one of the m alternatives, c_i, from choice set C is the exponential of the utility of the alternative divided by the sum of all of the exponentiated utilities.

There are $m = 8$ attribute vectors in this example, one for each alternative. Let $\mathbf{x}$ = (Dark/Milk, Soft/Chewy, Nuts/No Nuts) where Dark/Milk = (1 = Dark, 0 = Milk), Soft/Chewy = (1 = Soft, 0 = Chewy), Nuts/No Nuts = (1 = Nuts, 0 = No Nuts). The eight attribute vectors are

$\mathbf{x}_1 = (0\ 0\ 0)$ (Milk, Chewy, No Nuts)
$\mathbf{x}_2 = (0\ 0\ 1)$ (Milk, Chewy, Nuts)
$\mathbf{x}_3 = (0\ 1\ 0)$ (Milk, Soft, No Nuts)
$\mathbf{x}_4 = (0\ 1\ 1)$ (Milk, Soft, Nuts)
$\mathbf{x}_5 = (1\ 0\ 0)$ (Dark, Chewy, No Nuts)
$\mathbf{x}_6 = (1\ 0\ 1)$ (Dark, Chewy, Nuts)
$\mathbf{x}_7 = (1\ 1\ 0)$ (Dark, Soft, No Nuts)
$\mathbf{x}_8 = (1\ 1\ 1)$ (Dark, Soft, Nuts)

Say, hypothetically that $\boldsymbol{\beta}' = (4\ \ -2\ \ 1)$. That is, the part-worth utility for dark chocolate is 4, the part-worth utility for soft center is -2, and the part-worth utility for nuts is 1. The utility for each of the combinations, $\mathbf{x}_i\boldsymbol{\beta}$, would be as follows.

U(Milk, Chewy, No Nuts)	=	0×4	+	0×-2	+	0×1	=	0
U(Milk, Chewy, Nuts)	=	0×4	+	0×-2	+	1×1	=	1
U(Milk, Soft, No Nuts)	=	0×4	+	1×-2	+	0×1	=	-2
U(Milk, Soft, Nuts)	=	0×4	+	1×-2	+	1×1	=	-1
U(Dark, Chewy, No Nuts)	=	1×4	+	0×-2	+	0×1	=	4
U(Dark, Chewy, Nuts)	=	1×4	+	0×-2	+	1×1	=	5
U(Dark, Soft, No Nuts)	=	1×4	+	1×-2	+	0×1	=	2
U(Dark, Soft, Nuts)	=	1×4	+	1×-2	+	1×1	=	3

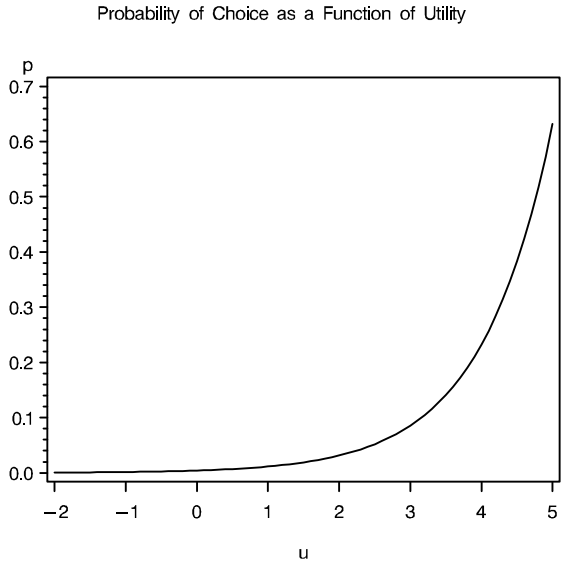
The denominator of the probability formula, $\sum_{j=1}^{m} \exp(\mathbf{x}_j\boldsymbol{\beta})$, is $\exp(0)+\exp(1)+\exp(-2)+\exp(-1)+\exp(4)+\exp(5)+\exp(2)+\exp(3) = 234.707$. The probability that each alternative is chosen, $\exp(\mathbf{x}_i\boldsymbol{\beta})/\sum_{j=1}^{m} \exp(\mathbf{x}_j\boldsymbol{\beta})$, is

p(Milk, Chewy, No Nuts)	=	exp(0)	/ 234.707	=	0.004
p(Milk, Chewy, Nuts)	=	exp(1)	/ 234.707	=	0.012
p(Milk, Soft, No Nuts)	=	exp(-2)	/ 234.707	=	0.001
p(Milk, Soft, Nuts)	=	exp(-1)	/ 234.707	=	0.002
p(Dark, Chewy, No Nuts)	=	exp(4)	/ 234.707	=	0.233
p(Dark, Chewy, Nuts)	=	exp(5)	/ 234.707	=	0.632
p(Dark, Soft, No Nuts)	=	exp(2)	/ 234.707	=	0.031
p(Dark, Soft, Nuts)	=	exp(3)	/ 234.707	=	0.086

Note that even combinations with a negative or zero utility have a nonzero probability of choice. Also note that adding a constant to the utilities will not change the probability of choice, however multiplying by a constant will.

Probability of choice is a nonlinear and increasing function of utility. The following plot shows the relationship between utility and probability of choice for this hypothetical situation.

```
data x;
   do u = -2 to 5 by 0.1;
      p = exp(u) / 234.707;
      output;
      end;
   run;
proc gplot;
   title h=1 'Probability of Choice as a Function of Utility';
   plot p * u;
   symbol1 i=join;
   run; quit;
```



This plot shows the function exp(−2) to exp(5), scaled into the range zero to one, the range of probability values. For the small negative utilities, the probability of choice is essentially zero. As utility increases beyond two, the function starts rapidly increasing.

In this example, the chosen alternatives are $\mathbf{x}_5$, $\mathbf{x}_6$, $\mathbf{x}_7$, $\mathbf{x}_5$, $\mathbf{x}_2$, $\mathbf{x}_6$, $\mathbf{x}_2$, $\mathbf{x}_6$, $\mathbf{x}_6$, $\mathbf{x}_6$. Alternative $\mathbf{x}_2$ was chosen 2 times, $\mathbf{x}_5$ was chosen 2 times, $\mathbf{x}_6$ was chosen 5 times, and $\mathbf{x}_7$ was chosen 1 time. The choice model likelihood for these data is the product of ten terms, one for each choice set for each subject. Each term consists of the probability that the chosen alternative is chosen. For each choice set, the utilities for all of the alternatives enter into the denominator, and the utility for the chosen alternative enters into the numerator. The choice model likelihood for these data is

$$\begin{aligned}
\mathcal{L}_C &= \frac{\exp(\mathbf{x}_5\boldsymbol{\beta})}{\left[\sum_{j=1}^{8}\exp(\mathbf{x}_j\boldsymbol{\beta})\right]} \times \frac{\exp(\mathbf{x}_6\boldsymbol{\beta})}{\left[\sum_{j=1}^{8}\exp(\mathbf{x}_j\boldsymbol{\beta})\right]} \times \frac{\exp(\mathbf{x}_7\boldsymbol{\beta})}{\left[\sum_{j=1}^{8}\exp(\mathbf{x}_j\boldsymbol{\beta})\right]} \times \frac{\exp(\mathbf{x}_5\boldsymbol{\beta})}{\left[\sum_{j=1}^{8}\exp(\mathbf{x}_j\boldsymbol{\beta})\right]} \times \\
&\quad \frac{\exp(\mathbf{x}_2\boldsymbol{\beta})}{\left[\sum_{j=1}^{8}\exp(\mathbf{x}_j\boldsymbol{\beta})\right]} \times \frac{\exp(\mathbf{x}_6\boldsymbol{\beta})}{\left[\sum_{j=1}^{8}\exp(\mathbf{x}_j\boldsymbol{\beta})\right]} \times \frac{\exp(\mathbf{x}_2\boldsymbol{\beta})}{\left[\sum_{j=1}^{8}\exp(\mathbf{x}_j\boldsymbol{\beta})\right]} \times \frac{\exp(\mathbf{x}_6\boldsymbol{\beta})}{\left[\sum_{j=1}^{8}\exp(\mathbf{x}_j\boldsymbol{\beta})\right]} \times \\
&\quad \frac{\exp(\mathbf{x}_6\boldsymbol{\beta})}{\left[\sum_{j=1}^{8}\exp(\mathbf{x}_j\boldsymbol{\beta})\right]} \times \frac{\exp(\mathbf{x}_6\boldsymbol{\beta})}{\left[\sum_{j=1}^{8}\exp(\mathbf{x}_j\boldsymbol{\beta})\right]} \\
&= \frac{\exp((2\mathbf{x}_2 + 2\mathbf{x}_5 + 5\mathbf{x}_6 + \mathbf{x}_7)\boldsymbol{\beta})}{\left[\sum_{j=1}^{8}\exp(\mathbf{x}_j\boldsymbol{\beta})\right]^{10}}
\end{aligned}$$

The Input Data

The data set consists of one observation for each alternative of each choice set for each subject. (A typical choice study has more than one choice set per person. This first example only has one choice set to help keep it simple.) All of the chosen and unchosen alternatives must appear in the data set. The data set must contain variables that identify the subject, the choice set, which alternative was chosen, and the set of alternatives from which it was chosen. In this example, the data set contains $10 \times 1 \times 8 = 80$ observations: 10 subjects each saw 1 choice set with 8 alternatives.

Typically, two variables are used to identify the choice sets, subject ID and choice set within subject. In this simple case where each subject only made one choice, the choice set variable is not necessary. However, we use it here to illustrate the general case. The variable `Subj` is the subject number, and `Set` identifies the choice set within subject. The chosen alternative is indicated by `c=1`, which means first choice. All second and subsequent choices are unobserved, so the unchosen alternatives are indicated by `c=2`, which means that all we know is that they would have been chosen after the first choice. Both the chosen and unchosen alternatives must appear in the input data set since both are needed to construct the likelihood function. The `c=2` observations enter into the denominator of the likelihood function, and the `c=1` observations enter into both the numerator and the denominator of the likelihood function. In this input DATA step, the data for four alternatives appear on one line, and all of the data for a choice set of eight alternatives appear on two lines. The DATA step shows data entry in the way that requires the fewest programming statements. Each execution of the `input` statement reads information about one alternative. The `@@` in the `input` statement specifies that SAS should not automatically go to a new input data set line when it reads the next row of data. This specification is needed here because each line in the input data set contains the data for four output data set rows. The data from the first two subjects is printed.

```
title 'Choice of Chocolate Candies';

data chocs;
   input Subj c Dark Soft Nuts @@;
   Set = 1;
   datalines;
 1 2 0 0 0    1 2 0 0 1    1 2 0 1 0    1 2 0 1 1
 1 1 1 0 0    1 2 1 0 1    1 2 1 1 0    1 2 1 1 1
 2 2 0 0 0    2 2 0 0 1    2 2 0 1 0    2 2 0 1 1
 2 2 1 0 0    2 1 1 0 1    2 2 1 1 0    2 2 1 1 1
 3 2 0 0 0    3 2 0 0 1    3 2 0 1 0    3 2 0 1 1
 3 2 1 0 0    3 2 1 0 1    3 1 1 1 0    3 2 1 1 1
 4 2 0 0 0    4 2 0 0 1    4 2 0 1 0    4 2 0 1 1
 4 1 1 0 0    4 2 1 0 1    4 2 1 1 0    4 2 1 1 1
 5 2 0 0 0    5 1 0 0 1    5 2 0 1 0    5 2 0 1 1
 5 2 1 0 0    5 2 1 0 1    5 2 1 1 0    5 2 1 1 1
 6 2 0 0 0    6 2 0 0 1    6 2 0 1 0    6 2 0 1 1
 6 2 1 0 0    6 1 1 0 1    6 2 1 1 0    6 2 1 1 1
 7 2 0 0 0    7 1 0 0 1    7 2 0 1 0    7 2 0 1 1
 7 2 1 0 0    7 2 1 0 1    7 2 1 1 0    7 2 1 1 1
 8 2 0 0 0    8 2 0 0 1    8 2 0 1 0    8 2 0 1 1
 8 2 1 0 0    8 1 1 0 1    8 2 1 1 0    8 2 1 1 1
 9 2 0 0 0    9 2 0 0 1    9 2 0 1 0    9 2 0 1 1
 9 2 1 0 0    9 1 1 0 1    9 2 1 1 0    9 2 1 1 1
10 2 0 0 0   10 2 0 0 1   10 2 0 1 0   10 2 0 1 1
10 2 1 0 0   10 1 1 0 1   10 2 1 1 0   10 2 1 1 1
;
proc print data=chocs noobs;
   where subj <= 2;
   var subj set c dark soft nuts;
   run;
```

Choice of Chocolate Candies

Subj	Set	c	Dark	Soft	Nuts
1	1	2	0	0	0
1	1	2	0	0	1
1	1	2	0	1	0
1	1	2	0	1	1
1	1	1	1	0	0
1	1	2	1	0	1
1	1	2	1	1	0
1	1	2	1	1	1

```
2     1     2     0     0     0
2     1     2     0     0     1
2     1     2     0     1     0
2     1     2     0     1     1
2     1     2     1     0     0
2     1     1     1     0     1
2     1     2     1     1     0
2     1     2     1     1     1
```

These next steps illustrate a more typical form of data entry. The experimental design is stored in a separate data set from the choices and is merged with the choices as the data are read, which produces the same results as the preceding steps. The process of merging the experimental design and the data is explicitly illustrated with a DATA step program. In practice, and in all of the other examples, we use the `%MktMerge` macro to do this.

```
title 'Choice of Chocolate Candies';

* Alternative Form of Data Entry;

data combos;                      /* Read the design matrix.     */
   input Dark Soft Nuts;
   datalines;
0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1
;
data chocs;                       /* Create the data set.        */
   input Choice @@; drop choice; /* Read the chosen combo num.  */
   Subj = _n_; Set = 1;           /* Store subj, choice set num. */
   do i = 1 to 8;                 /* Loop over alternatives.     */
      c = 2 - (i eq choice);      /* Designate chosen alt.       */
      set combos point=i;         /* Read design matrix.         */
      output;                     /* Output the results.         */
      end;
   datalines;
5 6 7 5 2 6 2 6 6 6
;
```

The variable `Choice` is the number of the chosen alternative. For each choice set, each of the eight observations in the experimental design is read. The `point=` option on the `set` statement is used to read the *ith* observation of the data set `Combos`. When `i` (the alternative index) equals `Choice` (the number of the chosen alternative), the logical expression `(i eq choice)` equals 1; otherwise it is 0. The statement `c = 2 - (i eq choice)` sets `c` to 1 (two minus one) when the alternative is chosen and 2 (two minus zero) otherwise. All eight observations in the `Combos` data set are read 10 times, once per subject. The resulting data set is the same as the one we created previously. As we mentioned

previously, in all of the remaining examples, we will simplify this process by using the `%MktMerge` macro to merge the design and data. The basic logic underlying this macro is shown in the preceding step. The number of a chosen alternative is read, then each alternative of the choice set is read, the chosen alternative is flagged `(c = 1)`, and the unchosen alternatives are flagged `(c = 2)`. One observation per choice set per subject is read from the input data stream, and one observation per alternative per choice set per subject is written.

Choice and Survival Models

In SAS, the multinomial logit model is fit with the SAS/STAT procedure PHREG (proportional hazards regression), with the `ties=breslow` option. The likelihood function of the multinomial logit model has the same form as a survival-analysis model fit by PROC PHREG.

In a discrete choice study, subjects are presented with sets of alternatives and asked to choose the most preferred alternative. The data for one choice set consist of one alternative that was chosen and $m-1$ alternatives that were not chosen. First choice was observed. Second and subsequent choices were not observed; it is only known that the other alternatives would have been chosen after the first choice. In survival analysis, subjects (rats, people, light bulbs, machines, and so on) are followed until a specific event occurs (such as failure or death) or until the experiment ends. The data are event times. The data for subjects who have not experienced the event (such as those who survive past the end of a medical experiment) are *censored*. The exact event time is not known, but it is known to have occurred after the censored time. In a discrete choice study, first choice occurs at time one, and all subsequent choices (second choice, third choice, and so on) are unobserved or censored. The survival and choice models are the same.

Fitting the Multinomial Logit Model

The data are now in the right form for analysis. To fit the multinomial logit model, use PROC PHREG as follows.

```
proc phreg data=chocs outest=betas;
   strata subj set;
   model c*c(2) = dark soft nuts / ties=breslow;
   label dark = 'Dark Chocolate' soft = 'Soft Center'
         nuts = 'With Nuts';
   run;
```

The `data=` option specifies the input data set. The `outest=` option requests an output data set called `Betas` with the parameter estimates. The `strata` statement specifies that each combination of the variables `Set` and `Subj` forms a set from which a choice was made. Each term in the likelihood function is a *stratum*. There is one term or stratum per choice set per subject, and each is composed of information about the chosen and all the unchosen alternatives.

In the left side of the `model` statement, you specify the variables that indicate which alternatives were chosen and not chosen. While this could be two different variables, we will use one variable `c` to provide both pieces of information. The response variable `c` has values 1 (chosen or first choice) and 2 (unchosen or subsequent choices). The first `c` of the `c*c(2)` in the `model` statement specifies that `c` indicates which alternative was chosen. The second `c` specifies that `c` indicates which alternatives

were not chosen, and `(2)` means that observations with values of 2 were not chosen. When `c` is set up such that 1 indicates the chosen alternative and 2 indicates the unchosen alternatives, always specify `c*c(2)` on the left of the equal sign in the `model` statement. The attribute variables are specified after the equal sign. Specify `ties=breslow` after a slash to explicitly specify the likelihood function for the multinomial logit model. (Do not specify any other `ties=` options; `ties=breslow` specifies the most computationally efficient and always appropriate way to fit the multinomial logit model.) The `label` statement is added since we are using a template that assumes each variable has a label.

Note that the `c*c(`n`)` syntax allows second choice `(c=2)` and subsequent choices (`c=3`, `c=4`, ...) to be entered. Just enter in parentheses one plus the number of choices actually made. For example, with first and second choice data specify `c*c(3)`. Note however that some experts believe that second and subsequent choice data are much less reliable than first choice data.

Multinomial Logit Model Results

The output is shown next. Recall that we used `%phchoice(on)` on page 143 to customize the output from PROC PHREG.

Choice of Chocolate Candies

The PHREG Procedure

Model Information

Data Set	WORK.CHOCS
Dependent Variable	c
Censoring Variable	c
Censoring Value(s)	2
Ties Handling	BRESLOW

Number of Observations Read	80
Number of Observations Used	80

Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Stratum	Subj	Set	Number of Alternatives	Chosen Alternatives	Not Chosen
1	1	1	8	1	7
2	2	1	8	1	7
3	3	1	8	1	7
4	4	1	8	1	7
5	5	1	8	1	7

6	6	1	8	1	7
7	7	1	8	1	7
8	8	1	8	1	7
9	9	1	8	1	7
10	10	1	8	1	7
Total			80	10	70

Convergence Status

Convergence criterion (GCONV=1E-8) satisfied.

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	41.589	28.727
AIC	41.589	34.727
SBC	41.589	35.635

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	12.8618	3	0.0049
Score	11.6000	3	0.0089
Wald	8.9275	3	0.0303

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Dark Chocolate	1	1.38629	0.79057	3.0749	0.0795
Soft Center	1	-2.19722	1.05409	4.3450	0.0371
With Nuts	1	0.84730	0.69007	1.5076	0.2195

The first table, `Model Information`, contains the input data set name, dependent variable name, censoring information, and tie handling option.

The `Summary of Subjects, Sets, and Chosen and Unchosen Alternatives` table is printed by default and should be used to check the data entry. In general, there are as many strata as there are combinations of the `Subj` and `Set` variables. In this case, there are ten strata. Each stratum must be composed of m alternatives. In this case, there are eight alternatives. The number of chosen alternatives should be 1, and the number of unchosen alternatives is $m - 1$ (in this case 7). **Always check the summary table to ensure that the data are arrayed correctly.**

The `Convergence Status` table shows that the iterative algorithm successfully converged. The next tables, `Model Fit Statistics` and `Testing Global Null Hypothesis: BETA=0` contain the overall fit of the model. The -2 LOG L statistic under `With Covariates` is 28.727 and the Chi-Square statistic is 12.8618 with 3 *df* (p=0.0049), which is used to test the null hypothesis that the attributes do not influence choice. At common alpha levels such as 0.05 and 0.01, we would reject the null hypothesis of no relationship between choice and the attributes. Note that 41.589 (-2 LOG L Without Covariates, which is -2 LOG L for a model with no explanatory variables) minus 28.727 (-2 LOG L With Covariates, which is -2 LOG L for a model with all explanatory variables) equals 12.8618 (Model Chi-Square, which is used to test the effects of the explanatory variables).

Next is the 'Multinomial Logit Parameter Estimates' table. For each effect, it contains the maximum likelihood parameter estimate, its estimated standard error (the square root of the corresponding diagonal element of the estimated variance matrix), the Wald Chi-Square statistic (the square of the parameter estimate divided by its standard error), the *df* of the Wald Chi-Square statistic (1 unless the corresponding parameter is redundant or infinite, in which case the value is 0), and the *p*-value of the Chi-Squared statistic with respect to a chi-squared distribution with one *df*. The parameter estimate with the smallest *p*-value is for soft center. Since the parameter estimate is negative, chewy is the more preferred level. Dark is preferred over milk, and nuts over no nuts, however only the *p*-value for Soft is less than 0.05.

Fitting the Multinomial Logit Model, All Levels

It is instructive to perform some manipulations on the data set and analyze it again. These steps will perform the same analysis as before, only now, coefficients for both levels of the three attributes are printed. Binary variables for the missing levels are created by subtracting the existing binary variables from 1.

```
data chocs2;
   set chocs;
   Milk = 1 - dark; Chewy = 1 - Soft; NoNuts = 1 - nuts;
   label dark = 'Dark Chocolate' milk   = 'Milk Chocolate'
         soft = 'Soft Center'    chewy  = 'Chewy Center'
         nuts = 'With Nuts'      nonuts = 'No Nuts';
   run;

proc phreg data=chocs2;
   strata subj set;
   model c*c(2) = dark milk soft chewy nuts nonuts / ties=breslow;
   run;
```

```
                    Choice of Chocolate Candies

                        The PHREG Procedure

                         Model Information

Data Set                  WORK.CHOCS2
Dependent Variable        c
```

Censoring Variable	c
Censoring Value(s)	2
Ties Handling	BRESLOW

Number of Observations Read	80
Number of Observations Used	80

Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Stratum	Subj	Set	Number of Alternatives	Chosen Alternatives	Not Chosen
1	1	1	8	1	7
2	2	1	8	1	7
3	3	1	8	1	7
4	4	1	8	1	7
5	5	1	8	1	7
6	6	1	8	1	7
7	7	1	8	1	7
8	8	1	8	1	7
9	9	1	8	1	7
10	10	1	8	1	7
Total			80	10	70

Convergence Status

Convergence criterion (GCONV=1E-8) satisfied.

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	41.589	28.727
AIC	41.589	34.727
SBC	41.589	35.635

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	12.8618	3	0.0049
Score	11.6000	3	0.0089
Wald	8.9275	3	0.0303

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Dark Chocolate	1	1.38629	0.79057	3.0749	0.0795
Milk Chocolate	0	0	.	.	.
Soft Center	1	-2.19722	1.05409	4.3450	0.0371
Chewy Center	0	0	.	.	.
With Nuts	1	0.84730	0.69007	1.5076	0.2195
No Nuts	0	0	.	.	.

Now the zero coefficients for the reference levels, milk, chewy, and no nuts are printed. The part-worth utility for Milk Chocolate is a structural zero, and the part-worth utility for Dark Chocolate is larger at 1.38629. Similarly, the part-worth utility for Chewy Center is a structural zero, and the part-worth utility for Soft Center is smaller at -2.19722. Finally, the part-worth utility for No Nuts is a structural zero, and the part-worth utility for Nuts is larger at 0.84730.

Probability of Choice

The parameter estimates are used next to construct the estimated probability that each alternative will be chosen. The DATA step program uses the following formula to create the choice probabilities.

$$p(c_i|C) = \frac{\exp(\mathbf{x}_i\boldsymbol{\beta})}{\sum_{j=1}^{m}\exp(\mathbf{x}_j\boldsymbol{\beta})}$$

```
* Estimate the probability that each alternative will be chosen;

data p;
   retain sum 0;
   set combos end=eof;

   * On the first pass through the DATA step (_n_ is the pass
     number), get the regression coefficients in B1-B3.
     Note that they are automatically retained so that they
     can be used in all passes through the DATA step.;

   if _n_ = 1 then
      set betas(rename=(dark=b1 soft=b2 nuts=b3));
   keep dark soft nuts p;
   array x[3] dark soft nuts;
   array b[3] b1-b3;
   * For each combination, create x * b;
   p = 0;
   do j = 1 to 3;
      p = p + x[j] * b[j];
      end;
```

```
   * Exponentiate x * b and sum them up;
   p   = exp(p);
   sum = sum + p;

   * Output sum exp(x * b) in the macro variable '&sum';
   if eof then call symput('sum',put(sum,best12.));
   run;

proc format;
   value df 1 = 'Dark' 0 = 'Milk';
   value sf 1 = 'Soft' 0 = 'Chewy';
   value nf 1 = 'Nuts' 0 = 'No Nuts';
   run;
* Divide each exp(x * b) by sum exp(x * b);
data p;
   set p;
   p = p / (&sum);
   format dark df. soft sf. nuts nf.;
   run;

proc sort;
   by descending p;
   run;
proc print;
  run;
```

Choice of Chocolate Candies

Obs	Dark	Soft	Nuts	p
1	Dark	Chewy	Nuts	0.50400
2	Dark	Chewy	No Nuts	0.21600
3	Milk	Chewy	Nuts	0.12600
4	Dark	Soft	Nuts	0.05600
5	Milk	Chewy	No Nuts	0.05400
6	Dark	Soft	No Nuts	0.02400
7	Milk	Soft	Nuts	0.01400
8	Milk	Soft	No Nuts	0.00600

The three most preferred alternatives are Dark/Chewy/Nuts, Dark/Chewy/No Nuts, and Milk/Chewy/Nuts.

Fabric Softener Example

In this example, subjects are asked to choose among fabric softeners. This example shows all of the steps in a discrete choice study, including experimental design creation and evaluation, creating the questionnaire, inputting the raw data, creating the data set for analysis, coding, fitting the discrete choice model, interpretation, and probability of choice. In addition, custom questionnaires are discussed. We assume that the reader is familiar with the experimental design issues that are discussed starting on page 47.

Set Up

The study involves four fictitious fabric softeners *Sploosh*, *Plumbbob*, *Platter*, and *Moosey*.[‡] Each choice set consists of each of these four brands and a constant alternative *Another*. Each of the brands is available at three prices, \$1.49, \$1.99, and \$2.49. *Another* is only offered at \$1.99. There are 50 subjects, each of which will see the same choice sets. We can use the `%MktRuns` autocall macro to help us choose the number of choice sets. All of the autocall macros used in this book are documented starting on page 597. To use this macro, you specify the number of levels for each of the factors. With four brands each with three prices, you specify four 3's (or `3 ** 4`).

```
title 'Choice of Fabric Softener';

%mktruns( 3 3 3 3 )
```

The output first tells us that we specified a design with four factors, each with three levels. The next table reports the size of the saturated design, which is the number of parameters in the linear model based on this design, and suggests design sizes.

```
                    Choice of Fabric Softener

                        Design Summary

                     Number of
                     Levels       Frequency

                          3               4

                    Choice of Fabric Softener

              Saturated      = 9
              Full Factorial = 81
```

[‡]Of course real studies would use real brands. Since we have not collected real data, we cannot use real brand names. We picked these silly names so no one would confuse our artificial data with real data.

```
Some Reasonable                    Cannot Be
   Design Sizes        Violations  Divided By

              9 *              0
             18 *              0
             12                6  9
             15                6  9
             10               10  3 9
             11               10  3 9
             13               10  3 9
             14               10  3 9
             16               10  3 9
             17               10  3 9

* - 100% Efficient Design can be made with the MktEx Macro.

                Choice of Fabric Softener

  n    Design                          Reference

  9           3 **  4                  Fractional-Factorial
 18    2 **  1  3 **  7                Orthogonal Array
 18           3 **  6  6 **  1         Orthogonal Array
```

The output from this macro tells us that the saturated design has nine runs and the full-factorial design has 81 runs. It also tells us that 9 and 18 are optimal design sizes with zero violations. The macro tells us that in nine runs, an orthogonal design with 4 three-level factors is available, and in 18 runs, two orthogonal and balanced designs are available: one with a two-level factor and 7 three-level factors, and one with 6 three-level factors and a six-level factor. There are zero violations with these designs because these sizes can be divided by 3 and $3 \times 3 = 9$. Twelve and 15 are also reported as potential design sizes, but each has 6 violations. Six times (the $4(4-1)/2 = 6$ pairs of the four 3's) 12 and 15 cannot be divided by $3 \times 3 = 9$. Ideally, we would like to have a manageable number of choice sets for people to evaluate and a design that is both orthogonal and balanced. When violations are reported, orthogonal and balanced designs are not possible. While orthogonality and balance are not required, they are nice properties to have. With 4 three-level factors, the number of choice sets in all orthogonal and balanced designs must be divisible by $3 \times 3 = 9$.

Nine choice sets is a bit small. Furthermore, there are no error *df*. We set the number of choice sets to 18 since it is small enough for each person to see all choice sets, large enough to have reasonable error *df*, and an orthogonal and balanced design is available. It is important to understand however that the concept of number of parameters and error *df* discussed here applies to the linear design and not to the choice design.[§] We could use the nine-run design for a discrete choice model and have error *df* in the choice model. If we were to instead use this design for a full-profile conjoint (not recommended), there would be no error *df*.

To make the code easier to modify for future use, the number of choice sets and alternatives are stored in macro variables and the prices are put into a format. Our design, in raw form, will have values for price of 1, 2, and 3. We will use a format to assign the actual prices: \$1.49, \$1.99, and \$2.49. The

[§]See page 60 for an explanation of linear versus choice designs.

format also creates a price of $1.99 for missing, which will be used for the constant alternative.

```
%let n = 18;                    /* n choice sets                    */
%let m = 5;                     /* m alternative including constant */
%let mm1 = %eval(&m - 1);       /* m - 1                            */

proc format;                    /* create a format for the price    */
   value price 1 = '$1.49' 2 = '$1.99' 3 = '$2.49' . = '$1.99';
   run;
```

Designing the Choice Experiment

In the next steps, an efficient experimental design is created. We will use an autocall macro `%MktEx` to create most of our designs. (All of the autocall macros used in this book are documented starting on page 597.) When you invoke the `%MktEx` macro for a simple problem, you only need to specify the numbers of levels, and number of runs. The macro does the rest. Here is the `%MktEx` macro usage for this example:

```
%mktex(3 ** 4, n=&n)
```

The syntax '*n* ** *m*' means *m* factors each at *n* levels. This example has four factors, `x1` through `x4`, all with three levels. A design with 18 runs is requested. The `n=` option specifies the number of runs. These are all the options that are needed for a simple problem such as this one. However, throughout this book, random number seeds are explicitly specified with the `seed=` option so that the results will be reproducible.¶ Here is the macro call with the random number seed specified:

```
%mktex(3 ** 4, n=&n, seed=17)
```

Here are the results.

Choice of Fabric Softener

Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
1	Start	100.0000	100.0000	Tab
1	End	100.0000		

¶By specifying a random number seed, results should be reproducible within a SAS release for a particular operating system and for a particular version of the macro. However, due to machine and macro differences, some results may not be exactly reproducible everywhere. For most orthogonal and balanced designs, the results should be reproducible. When computerized searches are done, it is likely that you will not get the same design as the one in the book, although you would expect the efficiency differences to be slight.

Choice of Fabric Softener

The OPTEX Procedure

Class Level Information

Class	Levels	Values
x1	3	1 2 3
x2	3	1 2 3
x3	3	1 2 3
x4	3	1 2 3

Choice of Fabric Softener

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	100.0000	100.0000	100.0000	0.7071

Here is the design.

```
proc print; run;
```

Choice of Fabric Softener

Obs	x1	x2	x3	x4
1	1	1	1	1
2	1	1	2	2
3	1	2	1	3
4	1	2	3	1
5	1	3	2	3
6	1	3	3	2
7	2	1	1	3
8	2	1	3	1
9	2	2	2	2
10	2	2	3	3
11	2	3	1	2
12	2	3	2	1

13	3	1	2	3
14	3	1	3	2
15	3	2	1	2
16	3	2	2	1
17	3	3	1	1
18	3	3	3	3

The macro found a perfect, orthogonal and balanced, 100% *D*-efficient design consisting of 4 three-level factors, `x1-x4`. The levels are the integers 1 to 3. For this problem, the macro generated the design directly. For other problems, the macro may have to use a computerized search. See page 191 for more information on how the `%MktEx` macro works.

Examining the Design

You should run basic checks on all designs, even orthogonal designs such as this one. You can use the `%MktEval` macro to display information about the design. The macro first prints a matrix of canonical correlations between the factors. We hope to see an identity matrix (a matrix of ones on the diagonal and zeros everywhere else) which means the design is orthogonal. Next, the macro prints all one-way frequencies for all attributes, all two-way frequencies, and all n-way frequencies (in this case four-way frequencies). We hope to see equal or at least nearly equal one-way and two-way frequencies, and we want to see that each combination occurs only once.

```
%mkteval;
```

Choice of Fabric Softener
Canonical Correlations Between the Factors
There are 0 Canonical Correlations Greater Than 0.316

	x1	x2	x3	x4
x1	1	0	0	0
x2	0	1	0	0
x3	0	0	1	0
x4	0	0	0	1

Choice of Fabric Softener
Summary of Frequencies
There are 0 Canonical Correlations Greater Than 0.316

Frequencies

x1	6 6 6
x2	6 6 6
x3	6 6 6
x4	6 6 6

```
x1 x2    2 2 2 2 2 2 2 2 2
x1 x3    2 2 2 2 2 2 2 2 2
x1 x4    2 2 2 2 2 2 2 2 2
x2 x3    2 2 2 2 2 2 2 2 2
x2 x4    2 2 2 2 2 2 2 2 2
x3 x4    2 2 2 2 2 2 2 2 2
N-Way    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

A *canonical correlation* is the maximum correlation between linear combinations of the coded factors (see page 70). All zeros off of the diagonal show that this design is orthogonal for main effects. If any off-diagonal canonical correlations had been greater than 0.316 ($r^2 > 0.1$), the macro would have listed them in a separate table. The last title line tells you that none of them was this large. For nonorthogonal designs and designs with interactions, the canonical-correlation matrix is not a substitute for looking at the variance matrix (with `examine=v`, discussed on pages 196, 243, and 683). It just provides a quick and more-compact picture of the correlations between the factors. The variance matrix is sensitive to the actual model specified and the actual coding. The canonical-correlation matrix just tells you if there is some correlation between the main effects. In this case, there are no correlations.

The equal one-way frequencies show you that this design is balanced. The equal two-way frequencies show you that this design is orthogonal. The n-way frequencies, all equal to one, show there are no duplicate profiles. This is a perfect design for a main-effects model.

You should always check the n-way frequencies to ensure that you do not have duplicates. For this situation for example, a 100% D-efficient design exists where each run appears twice. It consists of two copies of the fractional-factorial design 3^4 in 9 runs. When you get duplicates, specify `options=nodups` in the `%MktEx` macro, or sometimes you can just change the random number seed. Most designs will not have duplicates, so it is better to specify `options=nodups` only after you have found a design with duplicates. The no-duplicates constraint greatly slows down the algorithm.

The `%MktEval` macro produces a very compact summary of the design, hence some information, for example the levels to which the frequencies correspond, is not shown. You can use the `print=freqs` option to get a less compact and more detailed display.

```
%mkteval(data=design, print=freqs)
```

Here are some of the results.

Choice of Fabric Softener
Frequencies

Effects	Frequency	x1	x2	x3	x4
x1	6	1	.	.	.
	6	2	.	.	.
	6	3	.	.	.
x2	6	.	1	.	.
	6	.	2	.	.
	6	.	3	.	.

```
.
.
.
x1 x2          2          1          1          .          .
               2          1          2          .          .
               2          1          3          .          .
               2          2          1          .          .
               2          2          2          .          .
               2          2          3          .          .
               2          3          1          .          .
               2          3          2          .          .
               2          3          3          .          .
.
.
.
x3 x4          2          .          .          1          1
               2          .          .          1          2
               2          .          .          1          3
               2          .          .          2          1
               2          .          .          2          2
               2          .          .          2          3
               2          .          .          3          1
               2          .          .          3          2
               2          .          .          3          3
N-Way          1          1          1          1          1
               1          1          1          2          2
               1          1          2          1          3
               1          1          2          3          1
               1          1          3          2          3
               1          1          3          3          2
               1          2          1          1          3
               1          2          1          3          1
               1          2          2          2          2
               1          2          2          3          3
               1          2          3          1          2
               1          2          3          2          1
               1          3          1          2          3
               1          3          1          3          2
               1          3          2          1          2
               1          3          2          2          1
               1          3          3          1          1
               1          3          3          3          3
```

The Randomized Design and Postprocessing

The design we just looked at and examined was in the default output data set, `Design`. The `Design` data set is sorted, and often the first row consists entirely of ones. For these reasons, you should actually use the *randomized* design. In the randomized design, the choice sets are presented in a random order and the levels have been randomly reassigned. Neither of these operations affects the design *D*-efficiency, balance, or orthogonality. The macro automatically randomizes the design and stores the results in a data set called `Randomized`. The next steps assign formats and labels and store the results in a SAS data set `sasuser.Softener_LinDes`. It is important to store the design in a permanent SAS data set or in some other permanent form so that it will be available for analysis after the data are collected.

Every SAS data set has a two-level name of the form `libref.filename`. You can always reference a file with its two-level name. However, you can also use a one-level name, and then that data set is stored in temporary SAS data library with a libref of `Work`. Temporary data sets are deleted at the end of your SAS session, so any data that must be saved needs to be stored in a permanent SAS data set. The `libref` called `sasuser` is automatically available for permanent storage in most SAS installations. Furthermore, you can make your own `libref` using a `libname` statement. You may wish to create a separate library for each project. The latter approach of using a `libname` statement is usually preferable, but for our purposes, mainly to avoid discussing issues of host-specific paths and file names, we will use `sasuser`. See your BASE SAS documentation and SAS Companion for your operating system for more information on data libraries, `libref`, and `libname`.

```
data sasuser.Softener_LinDes;
   set randomized;
   format x1-x&mm1 price.;
   label x1 = 'Sploosh' x2 = 'Plumbbob' x3 = 'Platter' x4 = 'Moosey';
   run;
```

This is the final design.

```
proc print data=sasuser.Softener_LinDes label; /* print final design */
   title2 'Efficient Design';
   run;
```

Choice of Fabric Softener
Efficient Design

Obs	Sploosh	Plumbbob	Platter	Moosey
1	$1.99	$1.99	$1.99	$2.49
2	$2.49	$1.49	$1.49	$1.99
3	$1.49	$2.49	$2.49	$1.49
4	$2.49	$1.99	$2.49	$1.99
5	$1.49	$1.49	$1.49	$2.49
6	$1.49	$2.49	$1.99	$1.99
7	$2.49	$1.99	$1.99	$1.49
8	$2.49	$2.49	$1.49	$1.49
9	$1.99	$1.49	$2.49	$1.49

10	$1.49	$1.49	$1.99	$1.49
11	$1.99	$2.49	$1.49	$2.49
12	$1.49	$1.99	$1.49	$1.99
13	$1.99	$1.99	$1.49	$1.49
14	$1.49	$1.99	$2.49	$2.49
15	$2.49	$1.49	$2.49	$2.49
16	$1.99	$2.49	$2.49	$1.99
17	$1.99	$1.49	$1.99	$1.99
18	$2.49	$2.49	$1.99	$2.49

From a Linear Design to a Choice Design

The randomized design is now in a useful form for making the questionnaire, which is discussed in the next section. However, it is not in the final choice-design form that is needed for analysis and for the last evaluation that we should perform before collecting data. In this section, we convert our linear design to a choice design and evaluate its goodness for a choice model.

Our linear design, which we stored in a permanent SAS data set, `sasuser.Softener_LinDes`, is arranged with one row per choice set. For analysis, we need a *choice design* with one row for each alternative of each choice set. We call the randomized design a *linear design* (see page 60) because we used the `%MktEx` macro to create it optimizing *D*-efficiency for a linear model. We will use the macro `%MktRoll` to "roll out" the linear design into the choice design, which is in the proper form for analysis. First, we must create a data set that describes how the design will be processed. We call this data set the *design key*.

In this example, we want a choice design with two factors, `Brand` and `Price`. `Brand` has levels *Sploosh*, *Plumbbob*, *Platter*, *Moosey*, and *Another*. `Price` has levels $1.49, $1.99, and $2.49. `Brand` and `Price` are created by different processes. The `Price` factor will be constructed from the factors of the linear design matrix. In contrast, there is no `Brand` factor in the linear design. Each brand is a bin into which its factors are collected. The variable `Brand` will be named on the `alt=` option of the `%MktRoll` macro as the alternative variable, so its values will be read directly out of the `Key` data set. `Price` will not be named on the `alt=` macro option, so its values in the `Key` data set are variable names from the linear design data set. The values of `Price` in the final choice design will be read from the named variables in the linear design data set. The `Price` factor in the choice design is created from the four linear design factors (`x1` for *Sploosh*, `x2` for *Plumbbob*, `x3` for *Platter*, `x4` for *Moosey*, and no attribute for *Another*, the constant alternative). Here is how the `Key` data set is created. The `Brand` factor levels and the `Price` linear design factors are stored in the `Key` data set.

```
title2 'Key Data Set';

data key;
   input Brand $ Price $;
   datalines;
Sploosh   x1
Plumbbob  x2
Platter   x3
Moosey    x4
Another   .
;

proc print; run;
```

Choice of Fabric Softener
Key Data Set

Obs	Brand	Price
1	Sploosh	x1
2	Plumbbob	x2
3	Platter	x3
4	Moosey	x4
5	Another	

Note that the value of `Price` for alternative *Another* in the `Key` data set is blank (character missing). The period in the in-stream data set is simply a place holder, used with list input to read both character and numeric missing data. A period is not stored with the data. Next, we use the `%MktRoll` macro to process the design.

```
%mktroll(design=sasuser.Softener_LinDes, key=key, alt=brand,
         out=sasuser.Softener_ChDes)
```

The `%MktRoll` step processes the `design=sasuser.Softener_LinDes` linear design data set using the rules specified in the `key=key` data set, naming the `alt=brand` variable as the alternative name variable, and creating an output SAS data set called `out=sasuser.Softener_ChDes`, which contains the choice design. The input `design=sasuser.Softener_LinDes` data set has 18 observations, one per choice set, and the output `out=sasuser.Softener_ChDes` data set has $5 \times 18 = 90$ observations, one for each alternative of each choice set. Here are the first three observations of the linear design data set.

```
title2 'Linear Design (First 3 Sets)';

proc print data=sasuser.Softener_LinDes(obs=3); run;
```

Choice of Fabric Softener
Linear Design (First 3 Sets)

Obs	x1	x2	x3	x4
1	$1.99	$1.99	$1.99	$2.49
2	$2.49	$1.49	$1.49	$1.99
3	$1.49	$2.49	$2.49	$1.49

These observations define the first three choice sets. Here are those same observations, arrayed for analysis in the choice design data set.

```
title2 'Choice Design (First 3 Sets)';

proc print data=sasuser.Softener_ChDes(obs=15);
   format price price.;
   id set; by set;
   run;
```

Choice of Fabric Softener
Choice Design (First 3 Sets)

Set	Brand	Price
1	Sploosh	$1.99
	Plumbbob	$1.99
	Platter	$1.99
	Moosey	$2.49
	Another	$1.99
2	Sploosh	$2.49
	Plumbbob	$1.49
	Platter	$1.49
	Moosey	$1.99
	Another	$1.99
3	Sploosh	$1.49
	Plumbbob	$2.49
	Platter	$2.49
	Moosey	$1.49
	Another	$1.99

The choice design data set has a choice set variable `Set`, an alternative name variable `Brand`, and a price variable `Price`. The prices come from the linear design, and the price for *Another* is a constant $1.99. Recall that the prices are assigned by the following format.

```
proc format;                    /* create a format for the price */
   value price 1 = '$1.49' 2 = '$1.99' 3 = '$2.49' . = '$1.99';
   run;
```

Testing the Design Before Data Collection

Collecting data is time consuming and expensive. It is always good practice to make sure that the design will work with the most complicated model that we anticipate fitting. The following code evaluates the choice design.

```
title2 'Evaluate the Choice Design';

%choiceff(data=sasuser.Softener_ChDes, init=sasuser.Softener_ChDes(keep=set),
          nsets=&n, nalts=&m, beta=zero, intiter=0,
          model=class(brand price / zero='Another' '$1.99')
                / lprefix=0 cprefix=0%str(;)
                format price price.)
```

The `%ChoicEff` macro has two uses. You can use it to search for an efficient choice design, or you can use it to evaluate a choice design including designs that were generated using other methods such as the `%MktEx` macro. It is this latter use that is illustrated here.

The way you check a design like this is to first name it on the `data=` option. This will be the candidate

set that contains all of the choice sets that we will consider. In addition, the same design is named on the `init=` option. The full specification is `init=sasuser.Softener_ChDes(keep=set)`. Just the variable `Set` is kept. It will be used to bring in just the indicated choice sets from the `data=` design, which in this case is all of them. The option `nsets=&n` specifies that there are `&n`=18 choice sets, and `nalts=&m` specifies that there are `&m`=5 alternatives. The option `beta=zero` specifies that we are assuming for design evaluation purposes the null hypothesis that all of the betas or part-worth utilities are zero. You can evaluate the design for other parameter vectors by specifying a list of numbers after `beta=`. This will change the variances and standard errors. We also specify `intiter=0` which specifies zero internal iterations. We use zero internal iterations when we want to evaluate an initial design, but not attempt to improve it. The last option specifies the model.

The model specification contains everything that appears on the TRANSREG procedure's `model` statement for coding the design. Some of these options will be familiar from the previous example. The specification `class(brand price / zero='Another' '$1.99')` names the `brand` and `price` variable as a classification variables and asks for coded variables for every level except `'Another'` for `brand` and '$1.99' for `price`. The levels `'Another'` and `'$1.99'` are the reference levels for the two attributes. In a p-level factor, there are at most $p - 1$ nonzero parameters.

The `lprefix=0` option specifies that when labels are created for the binary variables, zero characters of the original variable name should be used as a prefix. This means that the labels are created only from the level values. For example, `'Sploosh'` and `'Plumbbob'` are created as labels not `'Brand Sploosh'` and `'Brand Plumbbob'`. The `cprefix=0` option specifies that when names are created for the binary variables, zero characters of the original variable name should be used as a prefix. This means that the names are created only from the level values. The `c` in `cprefix` stands for `class`.

The code following the `cprefix=` specification is a bit of a trick. The `%ChoicEff` macro generates a `model` statement for PROC TRANSREG using the specified value like this:

```
model &model;
```

By adding a semicolon, enclosed in `%str( )` and a format statement, we can send a format statement to the PROC TRANSREG coding step. The semicolon must be in the `%str( )` macro function so that it is passed into the macro and is not treated as the end of the macro specification. The model specification adds these two statements to PROC TRANSREG in the `%ChoicEff` macro.

```
model class(brand price / zero='Another' '$1.99') / lprefix=0 cprefix=0;
format price price.;
```

Alternatively, we could have just created a separate data set and added the format statement that way.

Here are the results from this step.

Choice of Fabric Softener
Evaluate the Choice Design

n	Name	Beta	Label
1	Moosey	0	Moosey
2	Platter	0	Platter
3	Plumbbob	0	Plumbbob
4	Sploosh	0	Sploosh
5	_1_49	0	$1.49
6	_2_49	0	$2.49

Choice of Fabric Softener
Evaluate the Choice Design

Design	Iteration	D-Efficiency	D-Error
1	0	2.342234	0.426943

Choice of Fabric Softener
Evaluate the Choice Design

Final Results

Design	1
Choice Sets	18
Alternatives	5
D-Efficiency	2.342234
D-Error	0.426943

Choice of Fabric Softener
Evaluate the Choice Design

n	Variable Name	Label	Variance	DF	Standard Error
1	Moosey	Moosey	0.72917	1	0.85391
2	Platter	Platter	0.72917	1	0.85391
3	Plumbbob	Plumbbob	0.72917	1	0.85391
4	Sploosh	Sploosh	0.72917	1	0.85391
5	_1_49	$1.49	0.52083	1	0.72169
6	_2_49	$2.49	0.52083	1	0.72169
				==	
				6	

The first table provides the name, specified value, and label for each parameter. The second table is the iteration history. There is just one line in the table since zero internal iterations were requested. The third table summarizes the design. The first design has 18 choice sets, 5 alternatives, a *D*-efficiency of 2.34 and a *D*-error of 0.43. *D*-error = 1 / *D*-efficiency. Note that *D*-efficiency and *D*-error are computed on a scale with an unknown maximum, so unlike the values that come out of the `%MktEx` macro, are not on a percentage or zero to 100 scale. When the 9.2 release of SAS is available, there will be new options for orthogonal coding in TRANSREG, and this will no longer always be the case. For now, the *D*-efficiency is not what really interests us. We are most interested in the final table. It shows the names and labels for the parameters as well as their variances, standard errors, and *df*. We see that the parameters for all four brands have the same standard errors. Similarly, the standard errors for the two prices are the same. They are different for the two attributes since both have a different number of levels. In both sets, however, they are all approximately the same order of magnitude. Sometimes you will see wildly varying parameters. This is usually a sign of a problematic design, perhaps one with too few choice sets for the number of parameters. This design looks good.

It is a really good idea to perform this step before designing the questionnaire and collecting data. Data collection is expensive, so it is good to make sure that the design can be used for the most complicated

model that you intend to fit. Much more will be said on evaluating the standard errors in the later and more complicated examples.

Generating the Questionnaire

A questionnaire based on the design is printed using the DATA step. The statement `array brands[&m] $ _temporary_ ('Sploosh' 'Plumbbob' 'Platter' 'Moosey' 'Another')` creates a constant array so that `brands[1]` accesses the string `'Sploosh'`, `brands[2]` accesses the string `'Plumbbob'`, and so on. The `_temporary_` specification means that no output data set variables are created for this array. The `linesleft=` specification in the `file` statement creates the variable `ll`, which contains the number of lines left on a page. This ensures that each choice set is not split over two pages.

```
options ls=80 ps=60 nonumber nodate;
title;

data _null_;                        /* print questionnaire */
   array brands[&m] $ _temporary_ ('Sploosh' 'Plumbbob' 'Platter'
                                   'Moosey' 'Another');
   array x[&m] x1-x&m;
   file print linesleft=ll;
   set sasuser.Softener_LinDes;

   x&m = 2;                        /* constant alternative */
   format x&m price.;

   if _n_ = 1 or ll < 12 then do;
      put _page_;
      put @60 'Subject: _________' //;
      end;
   put _n_ 2. ') Circle your choice of '
       'one of the following fabric softeners:' /;
   do brnds = 1 to &m;
      put '    ' brnds 1. ') ' brands[brnds] 'brand at '
          x[brnds] +(-1) '.' /;
      end;
   run;
```

In the interest of space, only the first two choice sets are printed. The questionnaire is printed, copied, and the data are collected.

```
                                                    Subject: _________

   1) Circle your choice of one of the following fabric softeners:

      1) Sploosh brand at $1.99.

      2) Plumbbob brand at $1.99.

      3) Platter brand at $1.99.

      4) Moosey brand at $2.49.

      5) Another brand at $1.99.

   2) Circle your choice of one of the following fabric softeners:

      1) Sploosh brand at $2.49.

      2) Plumbbob brand at $1.49.

      3) Platter brand at $1.49.

      4) Moosey brand at $1.99.

      5) Another brand at $1.99.
```

In practice, data collection will typically be much more elaborate than this. It may involve art work or photographs, and the choice sets may be presented and the data may be collected through personal interview or over the web. However the choice sets are presented and the data are collected, the essential elements remain the same. Subjects are shown a set of alternatives and are asked to make a choice, then they go on to the next set.

Entering the Data

The data consist of a subject number followed by 18 integers in the range 1 to 5. These are the alternatives that were chosen for each choice set. For example, the first subject chose alternative 3 (*Platter* brand at \$1.99) in the first choice set, alternative 3 (*Platter* brand at \$1.49) in the second choice set, and so on. In the interest of space, data from three subjects appear on one line.

```
title 'Choice of Fabric Softener';

data results;                     /* read choice data set */
   input Subj (choose1-choose&n) (1.) @@;
   datalines;
  1 334533434233312433  2 334213442433333325  3 333333333333313333
  4 334431444434412453  5 335431434233512423  6 334433434433312433
  7 334433434433322433  8 334433434433412423  9 334433332353312433
 10 325233435233332433 11 334233434433313333 12 334331334433312353
 13 534333334333312323 14 134421444433412423 15 334333435433312335
 16 334433435333315333 17 534333432453312423 18 334435544433412543
 19 334333335433313433 20 331431434233315533 21 334353534433512323
 22 334333452233312523 23 334333332333312433 24 525221444233322423
 25 354333434433312333 26 334435545233312323 27 334353534233352323
 28 334333332333332333 29 334433534335352423 30 334453434533313433
 31 354333334333312433 32 354331332233332423 33 334424432353312325
 34 334433434433312433 35 334551444453412325 36 334234534433312433
 37 334431434433512423 38 354333334433352523 39 334351334333312533
 40 324433334433412323 41 334433444433412443 42 334433434433312423
 43 334434454433332423 44 334433434233312423 45 334451544433412424
 46 434431435433512423 47 524434534433412433 48 335453334433322453
 49 334533434133312433 50 334433332333312423
;
```

Processing the Data

The next step merges the choice data with the choice design using the `%MktMerge` macro.

```
proc format;
   value price 1 = '$1.49' 2 = '$1.99' 3 = '$2.49' . = '$1.99';
   run;

%mktmerge(design=sasuser.Softener_ChDes, data=results, out=res2,
          nsets=&n, nalts=&m, setvars=choose1-choose&n)
```

This step reads the `design=sasuser.Softener_ChDes` choice design and the `data=results` data set and creates the `out=res2` output data set. The data are from an experiment with `nsets=&n` choice sets, `nalts=&m` alternatives, with variables `setvars=choose1-choose&n` containing the numbers of the chosen alternatives. Here are the first 15 observations.

```
title2 'Choice Design and Data (First 3 Sets)';

proc print data=res2(obs=15);
   id subj set; by subj set;
   run;
```

Choice of Fabric Softener
Choice Design and Data (First 3 Sets)

Subj	Set	Brand	Price	c
1	1	Sploosh	2	2
		Plumbbob	2	2
		Platter	2	1
		Moosey	3	2
		Another	.	2
1	2	Sploosh	3	2
		Plumbbob	1	2
		Platter	1	1
		Moosey	2	2
		Another	.	2
1	3	Sploosh	1	2
		Plumbbob	3	2
		Platter	3	2
		Moosey	1	1
		Another	.	2

The data set contains the subject ID variable `Subj` from the `data=results` data set, the `Set`, `Brand`, and `Price` variables from the `design=sasuser.Softener_ChDes` data set, and the variable `c`, which indicates which alternative was chosen. The variable `c` contains: 1 for first choice and 2 for second or subsequent choice. This subject chose the third alternative, *Platter* in the first choice set, *Platter* in the second, and *Moosey* in the third. This data set has 4500 observations: 50 subjects times 18 choice sets times 5 alternatives.

Since we did not specify a format, we see in the design the raw design values for `Price`: 1, 2, 3 and missing for the constant alternative. If we were going to treat `Price` as a categorical variable for analysis, this would be fine. We would simply assign our price format to `Price` and designate it as a `class` variable. However, in this analysis we are going to treat price as quantitative and use the actual prices in the analysis. Hence, we must convert our design values from 1, 2, 3, and . to 1.49, 1.99, 2.49, and 1.99. We cannot do this by simply assigning a format. Formats create character strings that are printed in place of the original value. We need to convert a numeric variable from one set of numeric values to another. We could use `if` and assignment statements. We could also use the `%MktLab` macro, which is used in later examples. However, instead we will use the `put` function to write the formatted value into a character string, then we read it back using a dollar format and the `input` function. For example, the expression `put(price, price.)` converts a number, say 2, into a string (in this case '$1.99'), then the `input` function reads the string and converts it to a numeric 1.99. This step also assigns a label to the variable `Price`.

```
data res3;   /* Create a numeric actual price */
   set res2;
   price = input(put(price, price.), dollar5.);
   label price = 'Price';
   run;
```

Binary Coding

One more thing must be done to these data before they can be analyzed. The factors must be coded. In this example, we use a *binary* or zero-one coding for the brand effect. This can be done with PROC TRANSREG.

```
proc transreg design=5000 data=res3 nozeroconstant norestoremissing;
   model class(brand / zero=none order=data)
         identity(price) / lprefix=0;
   output out=coded(drop=_type_ _name_ intercept);
   id subj set c;
   run;
```

The `design` option specifies that no model is fit; the procedure is just being used to code a design. When `design` is specified, dependent variables are not required. Optionally, `design` can be followed by "`=`n" where n is the number of observations to process at one time. By default, PROC TRANSREG codes all observations in one big group. For very large data sets, this can consume large amounts of memory and time. Processing blocks of smaller numbers of observations is more computationally efficient. The option `design=5000` processes observations in blocks of 5000. For smaller computers, try something like `design=1000`.

The `nozeroconstant` and `norestoremissing` options are not necessary for this example, but they are included here, because sometimes they are very helpful in coding choice models. The `nozeroconstant` option specifies that if the coding creates a constant variable, it should not be zeroed. The `nozeroconstant` option should always be specified when you specify `design=`n because the last group of observations may be small and may contain constant variables. The `nozeroconstant` option is also important if you do something like coding `by subj set` because sometimes an attribute is constant within a choice set. The `norestoremissing` option specifies that missing values should not be restored when the `out=` data set is created. By default, the coded `class` variable contains a row of missing values for observations in which the `class` variable is missing. When you specify the `norestoremissing` option, these observations contain a row of zeros instead. This option is useful when there is a constant alternative indicated by missing values. Both of these options, like almost all options in PROC TRANSREG, can be abbreviated to three characters (`noz` and `nor`).

The `model` statement names the variables to code and provides information about how they should be coded. The specification `class(brand / ...)` specifies that the variable `Brand` is a classification variable and requests a binary coding. The `zero=none` option creates binary variables for all categories. In contrast, by default, a binary variable is not created for the last category—the parameter for the last category is a structural zero. The `zero=none` option is used when there are no structural zeros or when you want to see the structural zeros in the multinomial logit parameter estimates table. The `order=data` option sorts the levels into the order that they were first encountered in the data set. Alternatively, the levels could be sorted based on the formatted or unformatted values. The specification `identity(price)` specifies that `Price` is a quantitative factor that should be analyzed as is (not expanded into indicator variables).

The `lprefix=0` option specifies that when labels are created for the binary variables, zero characters of the original variable name should be used as a prefix.

An `output` statement names the output data set and drops variables that are not needed. These variables do not have to be dropped. However, since they are variable names that are often found in special data set types, PROC PHREG prints warnings when it finds them. Dropping the variables prevents the warnings. Finally, the `id` statement names the additional variables that we want copied from the input to the output data set. The next steps print the first three coded choice sets.

```
proc print data=coded(obs=15) label;
   title2 'First 15 Observations of Analysis Data Set';
   id subj set c;
   by subj set;
   run;
```

Choice of Fabric Softener
First 15 Observations of Analysis Data Set

Subj	Set	c	Sploosh	Plumbbob	Platter	Moosey	Another	Price	Brand
1	1	2	1	0	0	0	0	1.99	Sploosh
		2	0	1	0	0	0	1.99	Plumbbob
		1	0	0	1	0	0	1.99	Platter
		2	0	0	0	1	0	2.49	Moosey
		2	0	0	0	0	1	1.99	Another
1	2	2	1	0	0	0	0	2.49	Sploosh
		2	0	1	0	0	0	1.49	Plumbbob
		1	0	0	1	0	0	1.49	Platter
		2	0	0	0	1	0	1.99	Moosey
		2	0	0	0	0	1	1.99	Another
1	3	2	1	0	0	0	0	1.49	Sploosh
		2	0	1	0	0	0	2.49	Plumbbob
		2	0	0	1	0	0	2.49	Platter
		1	0	0	0	1	0	1.49	Moosey
		2	0	0	0	0	1	1.99	Another

Fitting the Multinomial Logit Model

The next step fits the discrete choice, multinomial logit model.

```
proc phreg data=coded outest=betas brief;
   title2 'Discrete Choice Model';
   model c*c(2) = &_trgind / ties=breslow;
   strata subj set;
   run;
```

The `brief` option requests a brief summary for the strata. As with the candy example, `c*c(2)` designates the chosen and unchosen alternatives in the `model` statement. We specify the `&_trgind` macro variable for the `model` statement independent variable list. PROC TRANSREG automatically creates this macro variable. It contains the list of coded independent variables generated by the procedure. This is so you do not have to figure out what names TRANSREG created and specify them. In this case, PROC TRANSREG sets `&_trgind` to contain the following list.

```
BrandSploosh BrandPlumbbob BrandPlatter BrandMoosey BrandAnother Price
```

The `ties=breslow` option specifies a PROC PHREG model that has the same likelihood as the multinomial logit model for discrete choice. The `strata` statement specifies that the combinations of `Set` and `Subj` indicate the choice sets. This data set has 4500 observations consisting of $18 \times 50 = 900$ strata and five observations per stratum.

Each subject rated 18 choice sets, but the multinomial logit model assumes each stratum is independent. That is, the multinomial logit model assumes each person makes only one choice. The option of collecting only one datum from each subject is too expensive to consider for many problems, so multiple choices are collected from each subject, and the repeated measures aspect of the problem is ignored. This practice is typical, and it usually works well.

Multinomial Logit Model Results

The output is shown next. (Recall that we used `%phchoice(on)` on page 143 to customize the output from PROC PHREG.)

Choice of Fabric Softener
Discrete Choice Model

The PHREG Procedure

Model Information

Data Set	WORK.CODED
Dependent Variable	c
Censoring Variable	c
Censoring Value(s)	2
Ties Handling	BRESLOW

Number of Observations Read	4500
Number of Observations Used	4500

Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Pattern	Number of Choices	Number of Alternatives	Chosen Alternatives	Not Chosen
1	900	5	1	4

Convergence Status

Convergence criterion (GCONV=1E-8) satisfied.

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	2896.988	1439.457
AIC	2896.988	1449.457
SBC	2896.988	1473.469

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	1457.5310	5	<.0001
Score	1299.7889	5	<.0001
Wald	635.9093	5	<.0001

Choice of Fabric Softener
Discrete Choice Model

The PHREG Procedure

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Sploosh	1	-1.30565	0.21097	38.3017	<.0001
Plumbbob	1	-0.49090	0.18035	7.4091	0.0065
Platter	1	2.08485	0.14592	204.1402	<.0001
Moosey	1	0.62183	0.15503	16.0884	<.0001
Another	0	0	.	.	.
Price	1	-4.60150	0.21608	453.5054	<.0001

The procedure output begins with information about the data set, variables, options, and number of observations read. This is followed by information about the 900 strata. Since the `brief` option was specified, this table contains one row for each stratum pattern. In contrast, the default table would have 900 rows, one for each choice set and subject combination. Each subject and choice set combination consists of a total of five observations, one that was chosen and four that were not chosen. This pattern was observed 900 times. This table provides a check on data entry. Unless we have an availability or allocation study (page 334) or a nonconstant number of alternatives in different choice sets, we would expect to see one pattern of results where one of the m alternatives was chosen for each choice set. If you do not observe this for a study like this, there was probably a mistake in the data entry or processing.

The most to least preferred brands are: *Platter*, *Moosey*, *Another*, *Plumbbob*, and *Sploosh*. Increases in price have a negative utility. For example, the predicted utility of *Platter* brand at \$1.99 is $\mathbf{x}_i\boldsymbol{\beta}$ which is $(0 \quad 0 \quad 1 \quad 0 \quad 0 \quad \$1.99) \quad (-1.31 \quad -0.49 \quad 2.08 \quad 0.62 \quad 0 \quad -4.60)' = 2.08 + 1.99 \times -4.60 = -7.07$ Since `Price` was analyzed as a quantitative factor, we can see for example that the utility of *Platter* at \$1.89, which was not in any choice set, is $2.08 + 1.89 \times -4.60 = -6.61$, which is a $\$0.10 \times 4.60 = 0.46$ increase in utility.

Probability of Choice

These next steps compute the expected probability that each alternative is chosen within each choice set. This code could easily be modified to compute expected market share for hypothetical marketplaces that do not directly correspond to the choice sets. Note however, that a term like "expected market share," while widely used, is a misnomer. Without purchase volume data, it is unlikely that these numbers would mirror true market share.

First, PROC SCORE is used to compute the predicted utility for each alternative.

```
proc score data=coded(where=(subj=1) drop=c)
           score=betas type=parms out=p;
   var &_trgind;
   run;
```

The data set to be scored is named with the `data=` option, and the coefficients are specified in the option `score=beta`. Note that we only need to read all of the choice sets once, since the parameter estimates were computed in an aggregate analysis. This is why we specified `where=(subj=1)`. We do not need $\mathbf{x}_j\hat{\boldsymbol{\beta}}$ for each of the different subjects. We dropped the variable `c` from the `Coded` data set since this name will be used by PROC SCORE for the results ($\mathbf{x}_j\hat{\boldsymbol{\beta}}$). The option `type=parms` specifies that the `score=` data set contains the parameters in `_TYPE_ = 'PARMS'` observations. The output data set with the predicted utilities is named P. Scoring is based on the coded variables from PROC TRANSREG, whose names are contained in the macro variable `&_trgind`. The next step exponentiates $\mathbf{x}_j\hat{\boldsymbol{\beta}}$.

```
data p2;
   set p;
   p = exp(c);
   run;
```

Next, $\exp(\mathbf{x}_j\hat{\boldsymbol{\beta}})$ is summed for each choice set.

```
proc means data=p2 noprint;
   output out=s sum(p) = sp;
   by set;
   run;
```

Finally, each $\mathbf{x}_j\hat{\boldsymbol{\beta}}$ is divided by $\sum_{j=1}^{m} \mathbf{x}_j\hat{\boldsymbol{\beta}}$.

```
data p;
   merge p2 s(keep=set sp);
   by set;
   p = p / sp;
   keep brand set price p;
   run;
```

Here are the results for the first three choice sets.

```
proc print data=p(obs=15);
   title2 'Choice Probabilities for the First 3 Choice Sets';
   run;
```

Choice of Fabric Softener
Choice Probabilities for the First 3 Choice Sets

Obs	Price	Brand	Set	p
1	1.99	Sploosh	1	0.02680
2	1.99	Plumbbob	1	0.06052
3	1.99	Platter	1	0.79535
4	2.49	Moosey	1	0.01845
5	1.99	Another	1	0.09888
6	2.49	Sploosh	2	0.00030
7	1.49	Plumbbob	2	0.06843
8	1.49	Platter	2	0.89921
9	1.99	Moosey	2	0.02086
10	1.99	Another	2	0.01120
11	1.49	Sploosh	3	0.11679
12	2.49	Plumbbob	3	0.00265
13	2.49	Platter	3	0.03479
14	1.49	Moosey	3	0.80260
15	1.99	Another	3	0.04318

Custom Questionnaires

In this part of the example, a custom questionnaire is printed for each person. Previously, each subject saw the same questionnaire, with the same choice sets, each containing the same alternatives, with everything in the same order. In this example, the order of the choice sets and all alternatives within choice sets are randomized for each subject. Randomizing avoids any systematic effects due to the order of the alternatives and choice sets. The constant alternative is always printed last. If you have no interest in custom questionnaires, you can skip ahead to page 184.

First, the macro variable `&forms` is created. It contains the number of separate questionnaires (or forms or subjects, in this case 50). We can use the `%MktEx` macro to create a data set with one observation for each alternative of each choice set for each person. The specification `%mktex(&forms &n &mm1, n=&forms * &n * &mm1)` is `%mktex(50 18 4, n=50 * 18 * 4)` and creates a $50 \times 18 \times 4$ full-factorial design. Note that the `n=` specification allows expressions. The macro `%MktLab` is then used to assign the variable names `Form`, `Set`, and `Alt` instead of the default `x1 - x3`. The data set is sorted by `Form`. Within `Form`, the choice sets are sorted into a random order, and within choice set, the alternatives are sorted into a random order. The 72 observations for each choice set contain 18 blocks of 4 observations−one block per choice set in a random order and the 4 alternatives within each choice set, again in a random order. Note that we store these in a permanent SAS data set so they will be available after the data are collected. See page 163 for more information on permanent SAS data sets.

```
%let forms = 50;
title2 'Create 50 Custom Questionnaires';

*---Make the design---;
%mktex(&forms &n &mm1, n=&forms * &n * &mm1)

*---Assign Factor Names---;
%mktlab(data=design, vars=Form Set Alt)

*---Set up for Random Ordering---;
data sasuser.orders;
   set final;
   by form set;
   retain r1;
   if first.set then r1 = uniform(17);
   r2 = uniform(17);
   run;

*---Random Sort---;
proc sort out=sasuser.orders(drop=r:); by form r1 r2; run;

proc print data=sasuser.orders(obs=16); run;
```

The first 16 observations in this data set are shown next.

Choice of Fabric Softener
Create 50 Custom Questionnaires

Obs	Form	Set	Alt
1	1	4	3
2	1	4	1
3	1	4	2
4	1	4	4
5	1	8	2
6	1	8	3
7	1	8	1
8	1	8	4
9	1	16	1
10	1	16	2
11	1	16	3
12	1	16	4
13	1	1	3
14	1	1	1
15	1	1	4
16	1	1	2

The data set is transposed, so the resulting data set contains $50 \times 18 = 900$ observations, one per subject per choice set. The alternatives are in the variables `Col1-Col4`. The first 18 observations, which contain the ordering of the choice sets for the first subject, are shown next.

```
proc transpose data=sasuser.orders out=sasuser.orders(drop=_name_);
   by form notsorted set;
   run;

proc print data=sasuser.orders(obs=18);
   run;
```

Choice of Fabric Softener
Create 50 Custom Questionnaires

Obs	Form	Set	COL1	COL2	COL3	COL4
1	1	4	3	1	2	4
2	1	8	2	3	1	4
3	1	16	1	2	3	4
4	1	1	3	1	4	2
5	1	6	2	4	1	3
6	1	7	4	1	3	2
7	1	12	3	2	1	4
8	1	2	2	4	1	3
9	1	17	3	4	1	2
10	1	15	4	2	3	1
11	1	14	1	2	3	4
12	1	10	2	4	3	1
13	1	5	1	4	2	3
14	1	9	2	4	1	3
15	1	13	3	2	1	4
16	1	3	3	4	2	1
17	1	18	4	2	1	3
18	1	11	3	1	4	2

The following DATA step prints the 50 custom questionnaires.

```
options ls=80 ps=60 nodate nonumber;
title;

data _null_;
   array brands[&mm1] $ _temporary_
                  ('Sploosh' 'Plumbbob' 'Platter' 'Moosey');
   array x[&mm1] x1-x&mm1;
   array c[&mm1] col1-col&mm1;
   format x1-x&mm1 price.;
   file print linesleft=ll;
```

```
do frms = 1 to &forms;
   do choice = 1 to &n;
      if choice = 1 or ll < 12 then do;
         put _page_;
         put @60 'Subject: ' frms //;
         end;
      put choice 2. ') Circle your choice of '
          'one of the following fabric softeners:' /;
      set sasuser.orders;
      set sasuser.Softener_LinDes point=set;
      do brnds = 1 to &mm1;
         put '    ' brnds 1. ') ' brands[c[brnds]] 'brand at '
             x[c[brnds]] +(-1) '.' /;
         end;
      put '    5) Another brand at $1.99.' /;
      end;
   end;
stop;
run;
```

The loop `do frms = 1 to &forms` creates the 50 questionnaires. The loop `do choice = 1 to &n` creates the alternatives within each choice set. On the first choice set and when there is not enough room for the next choice set, we skip to a new page (`put _page_`) and print the subject (forms) number. The data set `sasuser.Orders` is read and the `Set` variable is used to read the relevant observation from `sasuser.Softener_LinDes` using the `point=` option in the `set` statement. The order of the alternatives is in the `c` array and variables `col1-col&mm1` from the `sasuser.Orders` data set. In the first observation of `sasuser.Orders`, `Set=4`, `Col1=3`, `Col2=1`, `Col3=2`, and `Col4=4`. The first brand, is `c[brnds] = c[1] = col1 = 3`, so `brands[c[brnds]] = brands[c[1]] = brands[3] = 'Platter'`, and the price, from observation `Set=4` of `sasuser.Softener_LinDes`, is `x[c[brnds]] = x[3] = $2.49`. The second brand, is `c[brnds] = c[2] = col2 = 1`, so `brands[c[brnds]] = brands[c[2]] = brands[1] = 'Sploosh'`, and the price, from observation `Set=4` of `sasuser.Softener_LinDes`, is `x[c[brnds]] = x[1] = $2.49`.

In the interest of space, only the first two choice sets are printed. Note that the subject number is printed on the form. This information is needed to restore all data to the original order.

```
                                                    Subject: 1


  1) Circle your choice of one of the following fabric softeners:

     1) Platter brand at $2.49.

     2) Sploosh brand at $2.49.

     3) Plumbbob brand at $1.99.

     4) Moosey brand at $1.99.

     5) Another brand at $1.99.
```

```
 2) Circle your choice of one of the following fabric softeners:

    1) Plumbbob brand at $2.49.

    2) Platter brand at $1.49.

    3) Sploosh brand at $2.49.

    4) Moosey brand at $1.49.

    5) Another brand at $1.99.
```

Processing the Data for Custom Questionnaires

Here are the data. (Actually, these are the data that would have been collected if the same people as in the previous situation made the same choices, without error and uninfluenced by order effects.) Before these data are analyzed, the original order must be restored.

```
title 'Choice of Fabric Softener';

data results;                    /* read choice data set */
   input Subj (choose1-choose&n) (1.) @@;
   datalines;
  1 524141141211421241  2 532234223321321311  3 223413221434144231
  4 424413322222544331  5 123324312534444533  6 233114423441143321
  7 123243224422433312  8 312432241121112412  9 315432222144111124
 10 511432445343442414 11 331244123342421432 12 323234114312123245
 13 312313434224435334 14 143433332142334114 15 234423133531441145
 16 425441421454434414 17 234431535341441432 18 235224352241523311
 19 134331342432542243 20 335331253334232433 21 513453254214134224
 22 212241213544214125 23 133444341431414432 24 453424142151142322
 25 324424431252444221 26 244145452131443415 27 553254131423323121
 28 233423242432231424 29 322454324541433543 30 323433433135133542
 31 412422434342513222 32 243144343352123213 33 441113141133454445
 34 131114113312342312 35 325222443355122522 36 342133254432124342
 37 511322324114234222 38 522153113442344541 39 211542232314512412
 40 244432222212213211 41 241411341323123213 42 314334342111232114
 43 422351321313343332 44 124243444234124432 45 141251113314352121
 46 414215225442424413 47 333452434454311222 48 334325341342552344
 49 335124122444243112 50 244412331342433332
;
```

The data set is transposed, and the original order is restored.

```
proc transpose data=results   /* create one obs per choice set */
               out=res2(rename=(col1=choose) drop=_name_);
   by subj;
   run;
```

```
data res3(keep=subj set choose);
   array c[&mm1] col1-col&mm1;
   merge sasuser.orders res2;
   if choose < 5 then choose = c[choose];
   run;

proc sort; by subj set; run;
```

The actual choice number, stored in `Choose`, indexes the alternative numbers from `sasuser.Orders` to restore the original alternative orders. For example, for the first subject, the first choice was 5, which is the *Another* constant alternative. Since the first subject saw the fourth choice set first, the fourth data value for the first subject in the processed data set will have a value of 5. The choice in the second choice set for the first subject was 2, and the second alternative the subject saw was *Platter*. The data set `sasuser.Orders` shows in the second observation that this choice of 2 corresponds to the third (original) alternative (in the second column variable, `Col2 = 3`) of choice set `Set= 8`. In the original ordering, *Platter* is the third alternative. Hence the eighth data value in the processed data set will have a value of 3. This DATA step writes out the data after the original order has been restored. It matches the data on page 171.

```
data _null_;
   set res3;
   by subj;
   if first.subj then do;
      if mod(subj, 3) eq 1 then put;
      put subj 4. +1 @@;
      end;
   put choose 1. @@;
   run;
```

```
 1 334533434233312433   2 334213442433333325   3 333333333333313333
 4 334431444434412453   5 335431434233512423   6 334433434433312433
 7 334433434433322433   8 334433434433412423   9 334433332353312433
10 325233435233332433  11 334233434433313333  12 334331334433312353
13 534333334333312323  14 134421444433412423  15 334333435433312335
16 334433435333315333  17 534333432453312423  18 334435544433412543
19 334333335433313433  20 331431434233315533  21 334353534433512323
22 334333452233312523  23 334333332333312433  24 525221444233322423
25 354333434433312333  26 334435545233312323  27 334353534233352323
28 334333332333332333  29 334433534335352423  30 334453434533313433
31 354333334333312433  32 354331332233332423  33 334424432353312325
34 334433434433312433  35 334551444453412325  36 334234534433312433
37 334431434433512423  38 354333334433352523  39 334351334333312533
40 324433334433412323  41 334433444433412443  42 334433434433312423
43 334434454433332423  44 334433434233312423  45 334451544433412424
46 434431435433512423  47 524434534433412433  48 335453334433322453
49 334533434133312433  50 334433332333312423
```

The data can be combined with the design and analyzed as in the previous example.

Vacation Example

This example illustrates the design and analysis for a larger choice experiment. We will discuss designing a choice experiment, evaluating the design, generating the questionnaire, processing the data, binary coding, generic attributes, quantitative price effects, quadratic price effects, effects coding, alternative-specific effects, analysis, and interpretation of the results. In this example, a researcher is interested in studying choice of vacation destinations. There are five destinations (alternatives) of interest: Hawaii, Alaska, Mexico, California, and Maine. Here are two summaries of the design, one with factors first grouped by attribute and one grouped by destination.

Factor	Destination	Attribute	Levels
X1	Hawaii	Accommodations	Cabin, Bed & Breakfast, Hotel
X2	Alaska	Accommodations	Cabin, Bed & Breakfast, Hotel
X3	Mexico	Accommodations	Cabin, Bed & Breakfast, Hotel
X4	California	Accommodations	Cabin, Bed & Breakfast, Hotel
X5	Maine	Accommodations	Cabin, Bed & Breakfast, Hotel
X6	Hawaii	Scenery	Mountains, Lake, Beach
X7	Alaska	Scenery	Mountains, Lake, Beach
X8	Mexico	Scenery	Mountains, Lake, Beach
X9	California	Scenery	Mountains, Lake, Beach
X10	Maine	Scenery	Mountains, Lake, Beach
X11	Hawaii	Price	$999, $1249, $1499
X12	Alaska	Price	$999, $1249, $1499
X13	Mexico	Price	$999, $1249, $1499
X14	California	Price	$999, $1249, $1499
X15	Maine	Price	$999, $1249, $1499

Factor	Destination	Attribute	Levels
X1	Hawaii	Accommodations	Cabin, Bed & Breakfast, Hotel
X6		Scenery	Mountains, Lake, Beach
X11		Price	$999, $1249, $1499
X2	Alaska	Accommodations	Cabin, Bed & Breakfast, Hotel
X7		Scenery	Mountains, Lake, Beach
X12		Price	$999, $1249, $1499
X3	Mexico	Accommodations	Cabin, Bed & Breakfast, Hotel
X8		Scenery	Mountains, Lake, Beach
X13		Price	$999, $1249, $1499
X4	California	Accommodations	Cabin, Bed & Breakfast, Hotel
X9		Scenery	Mountains, Lake, Beach
X14		Price	$999, $1249, $1499
X5	Maine	Accommodations	Cabin, Bed & Breakfast, Hotel
X10		Scenery	Mountains, Lake, Beach
X15		Price	$999, $1249, $1499

Each alternative is composed of three factors: package cost ($999, $1,249, $1,499), scenery (mountains, lake, beach), and accommodations (cabin, bed & breakfast, and hotel). There are five destinations, each with three attributes, for a total of 15 factors. This problem requires a design with 15 three-level factors, denoted 3^{15}. Each row of the design matrix contains the description of the five alternatives in one choice set. Note that the levels do not have to be the same for all destinations. For example, the cost for Hawaii and Alaska could be different from the other destinations. However, for this example, each destination will have the same attributes.

Set Up

We can use the `%MktRuns` autocall macro to suggest design sizes. (All of the autocall macros used in this book are documented starting on page 597.) To use this macro, you specify the number of levels for each of the factors. With 15 attributes each with three prices, you specify fifteen 3's (`3 3 3 3 3 3 3 3 3 3 3 3 3 3 3`), or you can use the more compact syntax of `3 ** 15`.

```
title 'Vacation Example';

%mktruns( 3 ** 15 )
```

The output tells us the size of the saturated design, which is the number of parameters in the linear design, and suggests design sizes.

Vacation Example

Design Summary

Number of Levels	Frequency
3	15

Vacation Example

Saturated = 31
Full Factorial = 14,348,907

Some Reasonable Design Sizes	Violations	Cannot Be Divided By
36	0	
45	0	
54 *	0	
63	0	
72 *	0	
33	105	9
39	105	9
42	105	9
48	105	9
51	105	9

* - 100% Efficient Design can be made with the MktEx Macro.

Vacation Example

n	Design									Reference
54	2 **	1	3 **	25						Orthogonal Array
54	2 **	1	3 **	21	9 **	1				Orthogonal Array
54			3 **	24	6 **	1				Orthogonal Array
54			3 **	20	6 **	1	9 **	1		Orthogonal Array
54			3 **	18	18 **	1				Orthogonal Array
72	2 **	23	3 **	24						Orthogonal Array
72	2 **	22	3 **	20	6 **	1				Orthogonal Array
72	2 **	21	3 **	16	6 **	2				Orthogonal Array
72	2 **	20	3 **	24	4 **	1				Orthogonal Array
72	2 **	19	3 **	20	4 **	1	6 **	1		Orthogonal Array
72	2 **	18	3 **	16	4 **	1	6 **	2		Orthogonal Array
72	2 **	16	3 **	25						Orthogonal Array
72	2 **	15	3 **	21	6 **	1				Orthogonal Array
72	2 **	14	3 **	24	6 **	1				Orthogonal Array
72	2 **	14	3 **	17	6 **	2				Orthogonal Array
72	2 **	13	3 **	25	4 **	1				Orthogonal Array
72	2 **	13	3 **	20	6 **	2				Orthogonal Array
72	2 **	12	3 **	24	12 **	1				Orthogonal Array
72	2 **	12	3 **	21	4 **	1	6 **	1		Orthogonal Array
72	2 **	12	3 **	16	6 **	3				Orthogonal Array
72	2 **	11	3 **	24	4 **	1	6 **	1		Orthogonal Array
72	2 **	11	3 **	20	6 **	1	12 **	1		Orthogonal Array
72	2 **	11	3 **	17	4 **	1	6 **	2		Orthogonal Array
72	2 **	10	3 **	20	4 **	1	6 **	2		Orthogonal Array
72	2 **	10	3 **	16	6 **	2	12 **	1		Orthogonal Array
72	2 **	9	3 **	16	4 **	1	6 **	3		Orthogonal Array
72			3 **	25	8 **	1				Orthogonal Array
72			3 **	24	24 **	1				Orthogonal Array

In this design, there are $15 \times (3-1)+1 = 31$ parameters, so at least 31 choice sets must be created. With all three-level factors, the number of choice sets in all orthogonal and balanced designs must be divisible by $3 \times 3 = 9$. Hence, optimal designs for this problem have at least 36 choice sets (the smallest number ≥ 31 and divisible by 9). Note however, that zero violations does not guarantee that a 100% *D*-efficient design exists. It just means that 100% *D*-efficiency is not precluded by unequal cell frequencies. In fact, the `%MktEx` orthogonal design catalog does not include orthogonal designs for this problem in 36, 45, and 63 runs (because they do not exist).

Thirty-six would be a good design size (2 blocks of size 18) as would 54 (3 blocks of size 18). Fifty-four would probably be the best choice, and that is what we would recommend for this study. However, we will instead create a *D*-efficient experimental design with 36 choice sets using the `%MktEx` macro. In practice, with more difficult designs, an orthogonal design is not available, and using 36 choice sets will allow us to see an example of using the `%Mkt` family of macros to get a nonorthogonal design.

We can see what orthogonal designs with three-level factors are available in 36 runs as follows. The `%MktOrth` macro creates a data set with information about the orthogonal designs that the `%MktEx` macro knows how to make. This macro produces a data set called `MktDesLev` that contains variables `n`, the number of runs; `Design`, a description of the design; and `Reference`, which contains the type of the design. In addition, there are variables: `x1`, the number of 1-level factors (which is always zero); `x2`, the number of 2-level factors; `x3`, the number of 3-level factors; and so on. We specify that `%MktOrth` only output `n=36` run designs and sort this list so that designs with the most three-level factors are printed first.

```
%mktorth(range=n=36)

proc sort data=mktdeslev out=list(drop=x:);
   by descending x3;
   where x3;
   run;

proc print; run;
```

Vacation Example

Obs	n	Design	Reference
1	36	2 ** 4 3 ** 13	Orthogonal Array
2	36	3 ** 13 4 ** 1	Orthogonal Array
3	36	2 ** 11 3 ** 12	Orthogonal Array
4	36	2 ** 2 3 ** 12 6 ** 1	Orthogonal Array
5	36	3 ** 12 12 ** 1	Orthogonal Array
6	36	2 ** 3 3 ** 9 6 ** 1	Orthogonal Array
7	36	2 ** 10 3 ** 8 6 ** 1	Orthogonal Array
8	36	2 ** 1 3 ** 8 6 ** 2	Orthogonal Array
9	36	3 ** 7 6 ** 3	Orthogonal Array
10	36	2 ** 2 3 ** 5 6 ** 2	Orthogonal Array
11	36	2 ** 16 3 ** 4	Orthogonal Array
12	36	2 ** 9 3 ** 4 6 ** 2	Orthogonal Array
13	36	2 ** 1 3 ** 3 6 ** 3	Orthogonal Array
14	36	2 ** 20 3 ** 2	Orthogonal Array
15	36	2 ** 11 3 ** 2 6 ** 1	Orthogonal Array
16	36	2 ** 3 3 ** 2 6 ** 3	Orthogonal Array
17	36	2 ** 27 3 ** 1	Orthogonal Array
18	36	2 ** 18 3 ** 1 6 ** 1	Orthogonal Array
19	36	2 ** 10 3 ** 1 6 ** 2	Orthogonal Array
20	36	2 ** 4 3 ** 1 6 ** 3	Orthogonal Array

There are 13 two-level factors available in 36 runs, and we need 15, only two more, so we would expect to make a pretty good nonorthogonal design.

Designing the Choice Experiment

The following code creates a design.

```
%let m   = 6;                    /* m alts including constant        */
%let mm1 = %eval(&m - 1);        /* m - 1                            */
%let n   = 18;                   /* number of choice sets per person */
%let blocks = 2;                 /* number of blocks                 */

%mktex(3 ** 15 2, n=&n * &blocks, seed=151)
```

The specification `3 ** 15` requests a design with 15 factors, `x1`−`x15`, each with three levels. This specification also requests a two-level factor (the `2` following the `3 ** 15`). This is because 36 choice sets may be too many for one person to rate, so we may want to block the design into two blocks, and we can use a two-level factor to do this. A design with $18 \times 2 = 36$ runs is requested, which will mean 36 choice sets. A random number seed is explicitly specified so we will be able to reproduce these exact results.*

Here are some of the log messages.

```
NOTE: Generating the candidate set.
NOTE: Performing 20 searches of 81 candidates, full-factorial=28,697,814.
NOTE: Generating the orthogonal array design, n=36.
```

The macro searches a fractional-factorial candidate set of 81 runs, and it also generates a tabled design in 36 runs to try as part of the design. This will be explained in more detail on page 191.

Here are some of the results from the `%MktEx` macro.

Vacation Example

Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
1	Start	82.2544	82.2544	Can
1	End	82.2544		

*By specifying a random number seed, results should be reproducible within a SAS release for a particular operating system and for a particular version of the macro. However, due to machine and macro differences, some results may not be exactly reproducible everywhere. For most orthogonal and balanced designs, the results should be reproducible. When computerized searches are done, it is likely that you will not get the same design as the one in the book, although you would expect the efficiency differences to be slight.

2	Start	78.8337		Tab,Ran
2	3 15	82.7741	82.7741	
2	4 14	83.2440	83.2440	
2	4 15	83.6041	83.6041	
2	4 15	83.8085	83.8085	
2	6 14	84.0072	84.0072	
.				
.				
.				
2	End	98.8567		
.				
.				
.				
5	Start	78.5222		Tab,Ran
5	11 14	98.8567	98.8567	
5	End	98.8567		
.				
.				
.				
8	Start	77.5829		Tab,Ran
8	10 15	98.9438	98.9438	
8	End	98.9438		
.				
.				
.				
21	Start	48.8411		Ran,Mut,Ann
21	End	93.1010		

Design Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
0	Initial	98.9438	98.9438	Ini
1	Start	77.8094		Tab,Ran
1	End	98.6368		
2	Start	77.9170		Tab,Ran
2	End	98.5516		
.				
.				
.				
78	Start	79.9023		Tab,Ran
78	24 15	98.9438	98.9438	
78	End	98.9438		

```
   .
   .
   .
  87      Start       74.7014                Tab,Ran
  87      4  15       98.9438       98.9438
  87        End       98.9438
   .
   .
   .
 146      Start       78.3794                Tab,Ran
 146     19  15       98.9438       98.9438
 146        End       98.9438
   .
   .
   .
 200      Start       84.1995                Tab,Ran
 200        End       98.6368
```

Vacation Example

Design Refinement History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
0	Initial	98.9438	98.9438	Ini
1	Start	96.5678		Pre,Mut,Ann
1	1 8	98.9438	98.9438	
1	26 14	98.9438	98.9438	
1	30 11	98.9438	98.9438	
1	1 12	98.9438	98.9438	
1	32 5	98.9438	98.9438	
1	18 6	98.9438	98.9438	
1	End	98.9438		
.				
.				
.				
6	Start	97.2440		Pre,Mut,Ann
6	33 7	98.9438	98.9438	
6	4 3	98.9438	98.9438	
6	16 12	98.9438	98.9438	
6	3 14	98.9438	98.9438	
6	20 15	98.9438	98.9438	
6	End	98.6958		

NOTE: Stopping since it appears that no improvement is possible.

Vacation Example

The OPTEX Procedure

Class Level Information

Class	Levels	Values
x1	3	1 2 3
x2	3	1 2 3
x3	3	1 2 3
x4	3	1 2 3
x5	3	1 2 3
x6	3	1 2 3
x7	3	1 2 3
x8	3	1 2 3
x9	3	1 2 3
x10	3	1 2 3
x11	3	1 2 3
x12	3	1 2 3
x13	3	1 2 3
x14	3	1 2 3
x15	3	1 2 3
x16	2	1 2

Vacation Example

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	98.9437	97.9592	98.9743	0.9428

The %MktEx macro used 30 seconds and found a design that is almost 99% *D*-efficient. (Differences in the fourth decimal place between the iteration history and the final table, in this case 98.9438 versus 98.9437, are due to rounding error and differences in ridging strategies between the macro code the generates the design and PROC OPTEX, which evaluates the design, and are nothing to worry about.)

The %MktEx Macro Algorithm

The %MktEx macro creates *D*-efficient linear experimental designs using several approaches. The macro will try to create a tabled design, it will search a set of candidate runs (rows of the design), and it will use a coordinate-exchange algorithm using both random initial designs and also a partial tabled design initialization. The macro stops if at any time it finds a perfect, 100% *D*-efficient, orthogonal and balanced design. This first phase is the algorithm search phase. In it, the macro tries a number of methods for this problem. At the end of this phase, the macro chooses the method that has produced the best design and performs another set of iterations using exclusively the chosen approach. Finally,

the macro performs a third set of iterations, where it takes the best design it found so far and tries to improve it.

The `%MktEx` macro can directly generate, without iterations, well over one-hundred thousand different 100% *D*-efficient, orthogonal and balanced, tabled designs. It does this using its design catalog and many different general and ad hoc algorithms. The closest design that the macro knows how to make for this problem is $2^1 3^{13}$ in 36 runs.

The candidate-set search has two parts. First, either PROC PLAN is run to create a full-factorial design for small problems, or PROC FACTEX is run to create a fractional-factorial design for large problems. Either way, this larger design is a *candidate set* that in the second part is searched by PROC OPTEX using the modified Fedorov algorithm. A design is built from a selection of the rows of the candidate set (Fedorov, 1972; Cook and Nachtsheim, 1980). The modified Fedorov algorithm considers each run in the design and each candidate run. Candidate runs are swapped in and design runs are swapped out if the swap improves *D*-efficiency. In this case, since the full-factorial design is large (over 14 million runs), the candidate-set search step calls PROC FACTEX to make the candidate set and then PROC OPTEX to do the search. The `Can` line of the iteration history shows that this step found a design that was 82.2544% *D*-efficient.

Next, the `%MktEx` macro uses the *coordinate-exchange algorithm*, based on Meyer and Nachtsheim (1995). The coordinate-exchange algorithm considers each level of each factor, and considers the effect on *D*-efficiency of changing a level ($1 \rightarrow 2$, or $1 \rightarrow 3$, or $2 \rightarrow 1$, or $2 \rightarrow 3$, or $3 \rightarrow 1$, or $3 \rightarrow 2$, and so on). Exchanges that increase *D*-efficiency are performed. In this step, the macro first tries to initialize the design with a tabled design (`Tab`) and a random design (`Ran`) both. In this case, 14 of the 16 factors can be initialized with the 13 three-level factors and one two-level factor of $2^4 3^{13}$, and the other two factors are randomly initialized. Levels that are not orthogonally initialized may be exchanged for other levels if the exchange increases *D*-efficiency. The algorithm search and design search iteration histories for this example show that the macro exchanged levels in factor 14 and 15 only, the ones that were randomly initialized.

The initialization may be more complicated in other problems. Say you asked for the design $4^1 5^1 3^4$ in 18 runs. The macro would use the tabled design $3^6 6^1$ in 18 runs to initialize the three-level factors orthogonally, and the five-level factor with the six-level factor coded down to five levels (and hence unbalanced). The four-level factor would be randomly initialized. The macro would also try the same initialization but with a random rather than unbalanced initialization of the five-level factor, as a minor variation on the first initialization. In the next initialization variation, the macro would use a fully-random initialization. If the number of runs requested were smaller than the number or runs in the initial tabled design, the macro would initialize the design with just the first n rows of the tabled design. Similarly, if the number of runs requested were larger than the number or runs in the initial tabled design, the macro would initialize part of the design with the orthogonal tabled design and the remaining rows and columns randomly. The coordinate-exchange algorithm considers each level of each factor that is not orthogonally initialized, and it exchanges a level if the exchange improves *D*-efficiency. When the number or runs in the tabled design does not match the number of runs desired, none of the design is initialized orthogonally.

The coordinate-exchange algorithm is not restricted by having a candidate set and hence can *potentially* consider any possible design. In practice, however, both the candidate-set-based and coordinate-exchange algorithms consider only a tiny fraction of the possible designs. When the number of runs in the full-factorial design is very small (say 100 or 200 runs), the modified Fedorov algorithm and coordinate-exchange algorithms usually work equally well. When the number of runs in the full-factorial design is small (up to several thousand), the modified Fedorov algorithm is sometimes superior to co-

ordinate exchange, particularly for models with interactions. When the full-factorial design is larger, coordinate exchange is usually the superior approach. However, heuristics like these are sometimes wrong, which is why the macro tries both methods to see which one is really best for each problem.

In the first attempt at coordinate exchange (Design 2), the macro found a design that is 98.8567% *D*-efficient (Design 2, `End`). In design 3 and subsequent designs, the macro uses this same approach, but different random initializations of the remaining two factors. In design 8, the `%MktEx` macro finds a design that is 98.9438% *D*-efficient. Designs 12 through 21 use a purely random initialization and simulated annealing and are not as good as previous designs. During these iterations, the macro is considering exchanging every level of every factor with every other level, one one row and one factor at a time. At this point, the `%MktEx` macro determines that the combination of tabled and random initialization is working best and tries more iterations using that approach. It starts by printing the initial (`Ini`) best *D*-efficiency of 98.9438. In designs 78, 87, 146, and 197 the macro finds a design that is 98.9438% *D*-efficient.

Next, the `%MktEx` macro tries to improve the best design it found previously. Using the previous best design as an initialization (`Pre`), and random mutations of the initialization (`Mut`) and simulated annealing (`Ann`), the macro uses the coordinate-exchange algorithm to try to find a better design. This step is important because the best design that the macro found may be an intermediate design, and it may not be the final design at the end of an iteration. Sometimes the iterations deliberately make the designs less *D*-efficient, and sometimes, the macro never finds a design as efficient or more efficient again. Hence it is worthwhile to see if the best design found so far can be improved. In this case, the macro fails to improve the design. After iteration 6, the macro stops since it keeps finding the same design over and over. This does not necessarily mean the macro found *the* optimal design; it means it found a very attractive (perhaps local) optimum, and it is unlikely it will do better using this approach. At the end, PROC OPTEX is called to print the levels of each factor and the final *D*-efficiency.

Random mutations add random noise to the initial design before the iterations start (levels are randomly changed). This may eliminate the perfect balance that will often be in the initial design. By default, random mutations are used with designs with fully-random initializations and in the design refinement step; orthogonal initial designs are not mutated.

Simulated annealing allows the design to get worse occasionally but with decreasing probability as the number of exchanges increases. For design 1, for the first level of the first factor, by default, the macro may execute an exchange (say change a 2 to a 1), that makes the design worse, with probability 0.05. As more and more exchanges occur, this probability decreases so at the end of the processing of design 1, exchanges that decrease *D*-efficiency are hardly ever done. For design 2, this same process is repeated, again starting by default with an annealing probability of 0.05. This often helps the algorithm overcome local efficiency maxima. To envision this, imagine that you are standing on a molehill next to a mountain. The only way you can start going up the mountain is to first step down off the molehill. Once you are on the mountain, you may occasionally hit a dead end, where all you can do is step down and look for a better place to continue going up. Other analogies include cleaning a garage and painting a room. Both have steps where you make things look worse so that in the end they may look better. The solitaire game "Spider," which is available on many PCs, is another example. Sometimes, you need to break apart those suits that you so carefully put together in order to make progress. Simulated annealing, by occasionally stepping down the efficiency function, often allows the macro to go farther up it than it would otherwise. The simulated annealing is why you will sometimes see designs getting worse in the iteration history. However, the macro keeps track of the best design, not the final design in each step. By default, annealing is used with designs with fully-random initializations and in the design refinement step; simulated annealing is not used with orthogonal initial designs.

For this example, the %MktEx macro ran in around 30 seconds. If an orthogonal design had been available, run time would have been a few seconds. If the fully-random initialization method had been the best method, run time might have been on the order of 10 to 45 minutes. Since the tabled initialization worked best, run time was much shorter. While it is possible to construct huge problems that will take much longer, for any design that most marketing researchers are likely to encounter, run time should be less than one hour. One of the macro options, maxtime=, typically ensures this.

Examining the Design

Before you use a design, you should always look at its characteristics. We will use the %MktEval macro.

```
%mkteval;
```

Here are some of the results.

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16
x1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
x4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
x5	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
x6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
x7	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
x8	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
x9	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
x10	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
x11	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
x12	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
x13	0	0	0	0	0	0	0	0	0	0	0	0	1	0.25	0.25	0
x14	0	0	0	0	0	0	0	0	0	0	0	0	0.25	1	0.25	0
x15	0	0	0	0	0	0	0	0	0	0	0	0	0.25	0.25	1	0
x16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Vacation Example
Summary of Frequencies
There are 0 Canonical Correlations Greater Than 0.316
* - Indicates Unequal Frequencies

Frequencies

x1	12 12 12
x2	12 12 12
x3	12 12 12
x4	12 12 12
x5	12 12 12
x6	12 12 12
x7	12 12 12

```
      x8          12 12 12
      x9          12 12 12
      x10         12 12 12
      x11         12 12 12
      x12         12 12 12
      x13         12 12 12
      x14         12 12 12
      x15         12 12 12
      x16         18 18
      x1 x2       4 4 4 4 4 4 4 4 4
      x1 x3       4 4 4 4 4 4 4 4 4
      x1 x4       4 4 4 4 4 4 4 4 4
      x1 x5       4 4 4 4 4 4 4 4 4
      x1 x6       4 4 4 4 4 4 4 4 4
      x1 x7       4 4 4 4 4 4 4 4 4
      x1 x8       4 4 4 4 4 4 4 4 4
      x1 x9       4 4 4 4 4 4 4 4 4
      x1 x10      4 4 4 4 4 4 4 4 4
      x1 x11      4 4 4 4 4 4 4 4 4
      x1 x12      4 4 4 4 4 4 4 4 4
      x1 x13      4 4 4 4 4 4 4 4 4
      x1 x14      4 4 4 4 4 4 4 4 4
      x1 x14      4 4 4 4 4 4 4 4 4
      x1 x15      4 4 4 4 4 4 4 4 4
      x1 x16      6 6 6 6 6 6
      .
      .
      .
*     x13 x14     3 6 3 6 3 3 3 3 6
*     x13 x15     3 3 6 3 6 3 6 3 3
      x13 x16     6 6 6 6 6 6
*     x14 x15     3 6 3 3 3 6 6 3 3
      x14 x16     6 6 6 6 6 6
      x15 x16     6 6 6 6 6 6
      N-Way       1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
                  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

This design looks great! The factors `x1-x13` form an orthogonal design, `x14` and `x15` are slightly correlated with each other and with `x13`. The blocking factor `x16` is orthogonal to all the other factors. All of the factors are perfectly balanced. The N-Way frequencies show that each choice set appears once.

What if there had been some larger canonical correlations? Would this be a problem? That depends. You have to decide this for yourself based on your particular study. You do not want large correlations between your most important factors. If you have high correlations between the wrong factors, you can swap them with other factors with the same number of levels, or try to make a new design with a different seed, or change the number of choice sets, and so on. While this design looks great, we should make one minor adjustment based on these results. Since our correlations are in the factors we originally planned to make price factors, we should change our plans slightly and use those factors for

less important attributes like scenery.

You can run the %MktEx macro to provide additional information about a design, for example asking to examine the information matrix (I) and its inverse (V), which is the variance matrix of the parameter estimates. You hope to see that all of the off-diagonal elements of the variance matrix, the covariances, are small relative to the variances on the diagonal. When options=check is specified, the macro evaluates an initial design instead of generating a design. The option init=randomized names the design to evaluate, and the examine= option displays the information and variance matrices. The blocking variable was dropped.

```
%mktex(3 ** 15,  n=&n * &blocks, init=randomized(drop=x16),
       options=check, examine=i v)
```

Here is a small part of the output.

Vacation Example

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	98.9099	97.8947	98.9418	0.9280

Vacation Example
Information Matrix

	Intercept	x11	x12	x21	x22	x31	x32	x41
Intercept	36.000	0	0	0	0	0	0	0
x11	0	36.000	0	0	0	0	0	0
x12	0	0	36.000	0	0	0	0	0
x21	0	0	0	36.000	0	0	0	0
x22	0	0	0	0	36.000	0	0	0
x31	0	0	0	0	0	36.000	0	0
x32	0	0	0	0	0	0	36.000	0
x41	0	0	0	0	0	0	0	36.000
.								
.								
.								
x122	36.000	0	0	0	0	0	0	0
x131	0	36.000	0	9.000	0	-4.500	-7.794	
x132	0	0	36.000	0	9.000	-7.794	4.500	
x141	0	9.000	0	36.000	0	-4.500	-7.794	
x142	0	0	9.000	0	36.000	-7.794	4.500	
x151	0	-4.500	-7.794	-4.500	-7.794	36.000	0	
x152	0	-7.794	4.500	-7.794	4.500	0	36.000	

Vacation Example
Variance Matrix

	Intercept	x11	x12	x21	x22	x31	x32	x41
Intercept	0.028	0	0	0	0	0	0	0
x11	0	0.028	0	0	0	0	0	0
x12	0	0	0.028	0	0	0	0	0
x21	0	0	0	0.028	0	0	0	0
x22	0	0	0	0	0.028	0	0	0
x31	0	0	0	0	0	0.028	0	0
x32	0	0	0	0	0	0	0.028	0
x41	0	0	0	0	0	0	0	0.028
.								
.								
.								
x122	0.028	0	0	0	0	0	0	0
x131	0	0.031	0	-0.006	0	0.003	0.005	
x132	0	0	0.031	0	-0.006	0.005	-0.003	
x141	0	-0.006	0	0.031	0	0.003	0.005	
x142	0	0	-0.006	0	0.031	0.005	-0.003	
x151	0	0.003	0.005	0.003	0.005	0.031	0	
x152	0	0.005	-0.003	0.005	-0.003	0	0.031	

This design still looks good. The *D*-efficiency for the design excluding the blocking factor is 98.9099%. We can see that the nonorthogonality between `x13-x15` makes their variances larger than the other factors (0.031 versus 0.028).

These next steps use the `%MktLab` macro to reassign the variable names, store the design in a permanent SAS data set, `sasuser.Vacation_LinDesBlckd`, and then use the `%MktEx` macro to check the results. See page 163 for more information on permanent SAS data sets. We need to make the correlated variables correspond to the least important attributes in different alternatives (in this case the scenery factors for Alaska, Mexico, and Maine). The `vars=` option provides the new variable names: the first variable (originally `x1`) becomes `x1` (still), ..., the fifth variable (originally `x5`) becomes `x5` (still), the sixth variable (originally `x6`) becomes `x11`, ... the tenth variable (originally `x10`) becomes `x15`, the eleventh through fifteenth original variables become `x6`, `x9`, `x7`, `x8`, `x10`, and finally the last variable becomes `Block`. The PROC SORT step sorts the design into blocks.

```
%mktlab(data=randomized, vars=x1-x5 x11-x15 x6 x9 x7 x8 x10 Block,
        out=sasuser.Vacation_LinDesBlckd)

proc sort data=sasuser.Vacation_LinDesBlckd; by block; run;

%mkteval(blocks=block)
```

Here is the output from the `%MktLab` macro, which shows the correspondence between the original and new variable names.

```
Variable Mapping:
   x1  : x1
   x2  : x2
   x3  : x3
   x4  : x4
   x5  : x5
   x6  : x11
   x7  : x12
   x8  : x13
   x9  : x14
   x10 : x15
   x11 : x6
   x12 : x9
   x13 : x7
   x14 : x8
   x15 : x10
   x16 : Block
```

Here is some of the output from the `%MktEval` macro.

Vacation Example
Canonical Correlations Between the Factors
There are 0 Canonical Correlations Greater Than 0.316

	Block	x1	x2	x3	x4	x5	x11	x12	x13	x14	x15	x6	x9	x7	x8	x10
Block	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
x3	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
x4	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
x5	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
x11	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
x12	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
x13	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
x14	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
x15	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
x6	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
x9	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
x7	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0.25	0.25
x8	0	0	0	0	0	0	0	0	0	0	0	0	0	0.25	1	0.25
x10	0	0	0	0	0	0	0	0	0	0	0	0	0	0.25	0.25	1

Vacation Example
Summary of Frequencies
There are 0 Canonical Correlations Greater Than 0.316
* - Indicates Unequal Frequencies

Frequencies

Block	18 18
x1	12 12 12
x2	12 12 12
x3	12 12 12
x4	12 12 12
x5	12 12 12
x11	12 12 12
x12	12 12 12
x13	12 12 12
x14	12 12 12
x15	12 12 12
x6	12 12 12
x9	12 12 12
x7	12 12 12
x8	12 12 12
x10	12 12 12
Block x1	6 6 6 6 6 6
Block x2	6 6 6 6 6 6
Block x3	6 6 6 6 6 6
Block x4	6 6 6 6 6 6
Block x5	6 6 6 6 6 6
Block x11	6 6 6 6 6 6
Block x12	6 6 6 6 6 6
Block x13	6 6 6 6 6 6
Block x14	6 6 6 6 6 6
Block x15	6 6 6 6 6 6
Block x6	6 6 6 6 6 6
Block x9	6 6 6 6 6 6
Block x7	6 6 6 6 6 6
Block x8	6 6 6 6 6 6
Block x10	6 6 6 6 6 6
x1 x2	4 4 4 4 4 4 4 4 4
x1 x3	4 4 4 4 4 4 4 4 4
x1 x4	4 4 4 4 4 4 4 4 4
x1 x5	4 4 4 4 4 4 4 4 4

.
.
.

```
*     x7 x8        6 3 3 3 6 3 3 3 6
*     x7 x10       3 3 6 3 6 3 6 3 3
*     x8 x10       3 3 6 3 6 3 6 3 3
      N-Way        1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
                   1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

From a Linear Design to a Choice Design

These next steps prepare the design for analysis and further evaluation. We need to covert our linear design into a choice design.† We need to create a data set `Key` that describes how the factors in our linear design will be used to make the choice design for analysis. The `Key` data set will contain all of the factor names, `x1`, `x2`, `x3`, ... `x15`. We can run the `%MktKey` macro to get these names, for cutting and pasting into the program without typing them. This requests 5 rows, 3 columns and the results transposed so names progress down each column instead of across each row.

```
%mktkey(5 3 t)
```

The `%MktKey` macro produced the following data set.

x1	x2	x3
x1	x6	x11
x2	x7	x12
x3	x8	x13
x4	x9	x14
x5	x10	x15

This code makes the `Key` data set and processes the design.

```
title 'Vacation Example';

data key;
   input Place $ 1-10 (Lodge Scene Price) ($);
   datalines;
Hawaii      x1    x6     x11
Alaska      x2    x7     x12
Mexico      x3    x8     x13
California  x4    x9     x14
Maine       x5    x10    x15
Home        .     .      .
;

%mktroll(design=sasuser.Vacation_LinDesBlckd, key=key, alt=place,
         out=sasuser.Vacation_ChDes)
```

†See page 60 for an explanation of linear versus choice designs.

For analysis, the design will have four factors as shown by the variables in the data set `Key`. `Place` is the alternative name; its values are directly read from the `Key` in-stream data. `Lodge` is an attribute whose values will be constructed from the `sasuser.Vacation_LinDesBlckd` data set. `Lodge` is created from `x1` for Hawaii, `x2` for Alaska, ..., `x5` for Maine, and no attribute for Home. Similarly, `Scene` is created from `x6-x10`, and `Price` is created from `x11-x15`. The macro `%MktRoll` is used to create the data set `sasuser.Vacation_ChDes` from `sasuser.Vacation_LinDesBlckd` using the mapping in `Key` and using the variable `Place` as the alternative ID variable.

The macro warns us:

```
WARNING: The variable BLOCK is in the DESIGN= data set but not
         the KEY= data set.
```

While this message could indicate a problem, in this case it does not. The variable `Block` in the `design=sasuser.Vacation_LinDesBlckd` data set will not appear in the final design. The purpose of the variable `Block` (sorting the design into blocks) has already been achieved. You can specify `options=nowarn` if you want to suppress this warning.

These next steps show the results for the first two choice sets. The data set is converted from a design matrix with one row per choice set to a design matrix with one row per alternative per choice set.

```
proc print data=sasuser.Vacation_LinDesBlckd(obs=2);
   id Block;
   var x1-x15;
   run;

proc print data=sasuser.Vacation_ChDes(obs=12);
   id set; by set;
   run;
```

Vacation Example

Block	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15
1	1	3	3	1	1	3	1	3	2	1	1	1	2	2	2
1	1	2	1	2	3	3	2	2	1	3	1	3	1	1	2

Vacation Example

Set	Place	Lodge	Scene	Price
1	Hawaii	1	3	1
	Alaska	3	1	1
	Mexico	3	3	2
	California	1	2	2
	Maine	1	1	2
	Home	.	.	.

2	Hawaii	1	3	1
	Alaska	2	2	3
	Mexico	1	2	1
	California	2	1	1
	Maine	3	3	2
	Home	.	.	.

The next steps assign formats, convert the variable `Price` to contain actual prices, and recode the constant alternative.

```
proc format;
   value price 1 = ' 999'      2 = '1249'
               3 = '1499'      0 = '   0';
   value scene 1 = 'Mountains' 2 = 'Lake'
               3 = 'Beach'     0 = 'Home';
   value lodge 1 = 'Cabin'     2 = 'Bed & Breakfast'
               3 = 'Hotel'     0 = 'Home';
   run;
data sasuser.Vacation_ChDes;
   set sasuser.Vacation_ChDes;
   if place = 'Home' then do; lodge = 0; scene = 0; price = 0; end;
   price = input(put(price, price.), 5.);
   format scene scene. lodge lodge.;
   run;

proc print data=sasuser.Vacation_ChDes(obs=12);
   id set; by set;
   run;
```

Vacation Example

Set	Place	Lodge	Scene	Price
1	Hawaii	Cabin	Beach	999
	Alaska	Hotel	Mountains	999
	Mexico	Hotel	Beach	1249
	California	Cabin	Lake	1249
	Maine	Cabin	Mountains	1249
	Home	Home	Home	0
2	Hawaii	Cabin	Beach	999
	Alaska	Bed & Breakfast	Lake	1499
	Mexico	Cabin	Lake	999
	California	Bed & Breakfast	Mountains	999
	Maine	Hotel	Beach	1249
	Home	Home	Home	0

It is not necessary to recode the missing values for the constant alternative. In practice, we usually will not do this step. However, for this first analysis, we will want all nonmissing values of the attributes so we can see all levels in the final printed output. We also recode `Price` so that for a later analysis, we can analyze `Price` as a quantitative effect. For example, the expression `put(price, price.)` converts a number, say 2, into a string (in this case '1249'), then the `input` function reads the string and converts it to a numeric 1249.

Testing the Design Before Data Collection

Collecting data is time consuming and expensive. It is always good practice to make sure that the design will work with the most complicated model that we anticipate fitting. The following code evaluates the choice design.

```
title2 'Evaluate the Choice Design';

%choiceff(data=sasuser.Vacation_ChDes, init=sasuser.Vacation_ChDes(keep=set),
          nsets=36, nalts=6, beta=zero, intiter=0,
          model=class(place / zero=none order=data)
                class(place * price place * scene place * lodge /
                      zero='Home' '0' 'Home' 'Home' order=formatted) /
                lprefix=0 cprefix=0 separators=' ' ', ')
```

The `%ChoicEff` macro has two uses. You can use it to search for an efficient choice design, or you can use it to evaluate a choice design including designs that were generated using other methods such as the `%MktEx` macro. It is this latter use that is illustrated here.

The way you check a design like this is to first name it on the `data=` option. This will be the candidate set that contains all of the choice sets that we will consider. In addition, the same design is named on the `init=` option. Just the variable `Set` is kept. It will be used to bring in just the indicated choice sets from the `data=` design, which in this case is all of them. The option `nsets=` specifies that there are 36 choice sets, and `nalts=` specifies that there are 6 alternatives. The option `beta=zero` specifies that we are assuming for design evaluation purposes the null hypothesis that all of the betas or part-worth utilities are zero. You can evaluate the design for other parameter vectors by specifying a list of numbers after `beta=`. This will change the variances and standard errors. We also specify `intiter=0` which specifies zero internal iterations. We use zero internal iterations when we want to evaluate an initial design, but not attempt to improve it. The last option specifies the model.

The model specification contains everything that appears on the TRANSREG procedure's `model` statement for coding the design. Some of these options will be familiar from the previous example. The specification `class(place / zero=none order=data)` names the `place` variable as a classification variables and asks for coded variables for every level including the constant, stay-at-home alternative. The specification `class(place * price place * scene place * lodge / zero='Home' '0' 'Home' 'Home' order=formatted)` asks for alternative-specific effects for price, lodging, and scenery. The alternative-specific effects allow the part-worth utilities to be different for each of the destinations. This is accomplished by requesting interactions between the destination and the attributes. Class levels are sorted by their formatted values, and for all factors, the reference level is the stay-at-home level. The factors, ignoring second and subsequent appearances in the `class` specification, are `place price scene lodge` and the values in the `zero=` option apply to the factors in order: `place` with `'Home'`, `price` with `'0'`, `scene` with `'Home'`, and `lodge` with `'Home'`.

The `lprefix=0` option specifies that when labels are created for the binary variables, zero characters of the original variable name should be used as a prefix. The `cprefix=0` option specifies that when names are created for the binary variables, zero characters of the original variable name should be used as a prefix. The `separators='' ', '` option provides two strings (one null and the other a comma followed by a blank) that allow you to specify label component separators for the main effect and interaction terms. By specifying a comma and a blank for the second value, we request labels for the side trip effects like `'Alaska, 999'` instead of the default `'Alaska * 999'`. This option is explained in more detail on page 265.

Here is the last table from the `%ChoicEff` macro, which is the one in which we are most interested.

Vacation Example
Evaluate the Choice Design

n	Variable Name	Label	Variance	DF	Standard Error
1	Hawaii	Hawaii	1.53333	1	1.23828
2	Alaska	Alaska	1.53545	1	1.23913
3	Mexico	Mexico	1.53545	1	1.23913
4	California	California	1.53333	1	1.23828
5	Maine	Maine	1.53545	1	1.23913
6	Home	Home	.	0	.
7	Alaska_999	Alaska, 999	1.20000	1	1.09545
8	Alaska_1249	Alaska, 1249	1.20000	1	1.09545
9	Alaska_1499	Alaska, 1499	.	0	.
10	California_999	California, 999	1.20000	1	1.09545
11	California_1249	California, 1249	1.20000	1	1.09545
12	California_1499	California, 1499	.	0	.
13	Hawaii_999	Hawaii, 999	1.20000	1	1.09545
14	Hawaii_1249	Hawaii, 1249	1.20000	1	1.09545
15	Hawaii_1499	Hawaii, 1499	.	0	.
16	Maine_999	Maine, 999	1.20000	1	1.09545
17	Maine_1249	Maine, 1249	1.20000	1	1.09545
18	Maine_1499	Maine, 1499	.	0	.
19	Mexico_999	Mexico, 999	1.20000	1	1.09545
20	Mexico_1249	Mexico, 1249	1.20000	1	1.09545
21	Mexico_1499	Mexico, 1499	.	0	.
22	AlaskaBeach	Alaska, Beach	1.20635	1	1.09834
23	AlaskaLake	Alaska, Lake	1.20635	1	1.09834
24	AlaskaMountains	Alaska, Mountains	.	0	.
25	CaliforniaBeach	California, Beach	1.20000	1	1.09545
26	CaliforniaLake	California, Lake	1.20000	1	1.09545
27	CaliforniaMountains	California, Mountains	.	0	.
28	HawaiiBeach	Hawaii, Beach	1.20000	1	1.09545
29	HawaiiLake	Hawaii, Lake	1.20000	1	1.09545
30	HawaiiMountains	Hawaii, Mountains	.	0	.

31	MaineBeach	Maine, Beach	1.20635	1	1.09834
32	MaineLake	Maine, Lake	1.20635	1	1.09834
33	MaineMountains	Maine, Mountains	.	0	.
34	MexicoBeach	Mexico, Beach	1.20635	1	1.09834
35	MexicoLake	Mexico, Lake	1.20635	1	1.09834
36	MexicoMountains	Mexico, Mountains	.	0	.
37	AlaskaBed___Breakfast	Alaska, Bed & Breakfast	1.20000	1	1.09545
38	AlaskaCabin	Alaska, Cabin	1.20000	1	1.09545
39	AlaskaHotel	Alaska, Hotel	.	0	.
40	CaliforniaBed___Breakfast	California, Bed & Breakfast	1.20000	1	1.09545
41	CaliforniaCabin	California, Cabin	1.20000	1	1.09545
42	CaliforniaHotel	California, Hotel	.	0	.
43	HawaiiBed___Breakfast	Hawaii, Bed & Breakfast	1.20000	1	1.09545
44	HawaiiCabin	Hawaii, Cabin	1.20000	1	1.09545
45	HawaiiHotel	Hawaii, Hotel	.	0	.
46	MaineBed___Breakfast	Maine, Bed & Breakfast	1.20000	1	1.09545
47	MaineCabin	Maine, Cabin	1.20000	1	1.09545
48	MaineHotel	Maine, Hotel	.	0	.
49	MexicoBed___Breakfast	Mexico, Bed & Breakfast	1.20000	1	1.09545
50	MexicoCabin	Mexico, Cabin	1.20000	1	1.09545
51	MexicoHotel	Mexico, Hotel	.	0	.
				==	
				35	

We see estimable parameters for the five destinations, but not for the stay at home alternative. For each destination/attribute combination, which are the alternative-specific effects, we see two estimable parameters. In some sense, each `class` variable in a choice model with a constant alternative has two reference levels or two levels that will always have a zero coefficient: the level corresponding to the constant alternative (mostly not shown here) and the level corresponding to the last level. More will be said on this throughout the analysis. The standard errors for most of the alternative-specific effects are 1.09545, but a few are a bit higher. They correspond scenery attributes for Alaska, Maine, and Mexico, which are our nonorthogonal factors. This design looks quite good. Everything that should be estimable in an alternative-specific effects model is estimable, and all of the standard errors are of a similar magnitude.

Generating the Questionnaire

This next DATA step prints the questionnaires. They are then copied and the data are collected.

```
title;
proc sort data=sasuser.Vacation_LinDesBlckd; by block; run;

options ls=80 ps=60 nodate nonumber;

data _null_;
   array dests[&mm1] $ 10 _temporary_ ('Hawaii' 'Alaska' 'Mexico'
                                       'California' 'Maine');
   array prices[3]   $  5 _temporary_ ('$999' '$1249' '$1499');
   array scenes[3]   $ 13 _temporary_
                     ('the Mountains' 'a Lake' 'the Beach');
   array lodging[3]  $ 15 _temporary_
                     ('Cabin' 'Bed & Breakfast' 'Hotel');
   array x[15];
   file print linesleft=ll;
   set sasuser.Vacation_LinDesBlckd;
   by block;

   if first.block then do;
      choice = 0;
      put _page_;
      put @50 'Form: ' block  ' Subject: ________' //;
      end;
   choice + 1;
   if ll < 19 then put _page_;
   put choice 2. ') Circle your choice of '
       'vacation destinations:' /;
   do dest = 1 to &mm1;
      put '    ' dest 1. ') ' dests[dest]
          +(-1) ', staying in a ' lodging[x[dest]]
          'near ' scenes[x[&mm1 + dest]] +(-1) ',' /
          '       with a package cost of '
         prices[x[2 * &mm1 + dest]] +(-1) '.' /;
      end;
   put "    &m) Stay at home this year." /;
   run;
```

In this design, there are five destinations, and each destination has three attributes. Each destination name is accessed from the array `dests`. Note that destination is not a factor in the design; it is a bin into which the attributes are grouped. The factors in the design are named in the statement `array x[15]`, which is a short-hand notation for `array x[15] x1-x15`. The first five factors are used for the lodging attribute of the five destinations. The actual descriptions of lodging are accessed by `lodging[x[dest]]`. The variable `Dest` varies from 1 to 5 destinations, so `x[dest]` extracts the levels for the `Dest` destination. Similarly for scenery, `scenes[x[&mm1 + dest]]` extracts the descriptions of the scenery. The index `&mm1 + dest` accesses factors 6 through 10, and `x[&mm1 + dest]` indexes the `scenes` array. For prices, `prices[x[2 * &mm1 + dest]]`, the index `2 * &mm1 + dest` accesses the

factors 11 through 15. Here are the first two choice sets.

```
                                Vacation Example
                                              Form: 1  Subject: ________


   1) Circle your choice of vacation destinations:

      1) Hawaii, staying in a Cabin near the Beach,
         with a package cost of $999.

      2) Alaska, staying in a Hotel near the Mountains,
         with a package cost of $999.

      3) Mexico, staying in a Hotel near the Beach,
         with a package cost of $1249.

      4) California, staying in a Cabin near a Lake,
         with a package cost of $1249.

      5) Maine, staying in a Cabin near the Mountains,
         with a package cost of $1249.

      6) Stay at home this year.

   2) Circle your choice of vacation destinations:

      1) Hawaii, staying in a Cabin near the Beach,
         with a package cost of $999.

      2) Alaska, staying in a Bed & Breakfast near a Lake,
         with a package cost of $1499.

      3) Mexico, staying in a Cabin near a Lake,
         with a package cost of $999.

      4) California, staying in a Bed & Breakfast near the Mountains,
         with a package cost of $999.

      5) Maine, staying in a Hotel near the Beach,
         with a package cost of $1249.

      6) Stay at home this year.
```

In practice, data collection will typically be much more elaborate than this. It may involve art work or photographs, and the choice sets may be presented and the data may be collected through personal interview or over the web. However the choice sets are presented and the data are collected, the essential elements remain the same. Subjects are shown a set of alternatives and are asked to make a choice, then they go on to the next set.

Entering and Processing the Data

Here are some of the input data. Data from a total of 200 subjects were collected, 100 per form.

```
 1  1 111353313351554151   2  2 344113155513111413   3  1 132353331151534151
 4  2 341133131523331143   5  1 142153111151334143   6  2 344114111543131151
 7  1 141343111311154154   8  2 344113111343121111   9  1 141124131151342155
10  2 344113131523131141  11  1 311423131353524144  12  2 332123151413331151
13  1 311244331352134155  14  2 341114111543131153  15  1 141253111351344151
16  2 344135131323331143  17  1 142123313154132141  18  2 542113151323131141
19  1 145314111311144111  20  2 344111131313431143  21  1 133343131313432145
.
.
.
;
```

Next, we use the macro `%MktMerge` to combine the data and design and create the variable `c`, indicating whether each alternative was a first choice or a subsequent choice.

```
%mktmerge(design=sasuser.Vacation_ChDes, data=results, out=res2, blocks=form,
          nsets=&n, nalts=&m, setvars=choose1-choose&n)

proc print data=res2(obs=12);
   id subj form set; by subj form set;
   run;
```

This macro takes the `design=sasuser.Vacation_ChDes` experimental design, merges it with the `data=result` data set, creating the `out=res2` output data set. The `Results` data set contains the variable `Form` that contains the block number. Since there are two blocks, this variable must have values of 1 and 2. This variable must be specified in the `blocks=` option. The experiment has `nsets=&n` choice sets, `nalts=6` alternatives, and the variables `setvars=choose1-choose&n` contain the numbers of the chosen alternatives. The output data set `Res2` has 21,600 observations (200 subjects who each saw 18 choice sets with 6 alternatives). Here are the first two choice sets.

Vacation Example

Subj	Form	Set	Place	Lodge	Scene	Price	c
1	1	1	Hawaii	Cabin	Beach	999	1
			Alaska	Hotel	Mountains	999	2
			Mexico	Hotel	Beach	1249	2
			California	Cabin	Lake	1249	2
			Maine	Cabin	Mountains	1249	2
			Home	Home	Home	0	2
1	1	2	Hawaii	Cabin	Beach	999	1
			Alaska	Bed & Breakfast	Lake	1499	2
			Mexico	Cabin	Lake	999	2
			California	Bed & Breakfast	Mountains	999	2
			Maine	Hotel	Beach	1249	2
			Home	Home	Home	0	2

Binary Coding

One more thing must be done to these data before they can be analyzed. The binary design matrix is coded for each effect. This can be done with PROC TRANSREG.

```
proc transreg design=5000 data=res2 nozeroconstant norestoremissing;
   model class(place / zero=none order=data)
         class(price scene lodge / zero=none order=formatted) /
         lprefix=0;
   output out=coded(drop=_type_ _name_ intercept);
   id subj set form c;
   run;
```

The `design` option specifies that no model is fit; the procedure is just being used to code a design. When `design` is specified, dependent variables are not required. Optionally, `design` can be followed by "=n" where n is the number of observations to process at one time. By default, PROC TRANSREG codes all observations in one big group. For very large data sets, this can consume large amounts of memory and time. Processing blocks of smaller numbers of observations is more computationally efficient. The option `design=5000` processes observations in blocks of 5000. For smaller computers, try something like `design=1000`.

The `nozeroconstant` and `norestoremissing` options are not necessary for this example but are included here because sometimes they are very helpful in coding choice models. The `nozeroconstant` option specifies that if the coding creates a constant variable, it should not be zeroed. The `nozeroconstant` option should always be specified when you specify `design=`n because the last group of observations may be small and may contain constant variables. The `nozeroconstant` option is also important if you do something like coding `by subj set` because sometimes an attribute is constant within a choice set. The `norestoremissing` option specifies that missing values should not be restored when the `out=` data set is created. By default, the coded `class` variable contains a row of missing values for observations in which the `class` variable is missing. With the `norestoremissing` option, these observations contain a row of zeros instead. This option is useful when there is a constant alternative indicated by missing values. Both of these options, like almost all options in PROC TRANSREG, can be abbreviated to three characters (`noz` and `nor`).

The `model` statement names the variables to code and provides information about how they should be coded. The specification `class(place / ...)` specifies that the variable `Place` is a classification variable and requests a binary coding. The `zero=none` option creates binary variables for all categories. The `order=data` option sorts the levels into the order they were first encountered in the data set. It is specified so `'Home'` will be the last destination in the analysis, as it is in the data set. The `class(price scene lodge / ...)` specification names the variables `Price`, `Scene`, and `Lodge` as categorical variables and creates binary variables for all of the levels of all of the variables. The levels are sorted into order based on their formatted values. The `lprefix=0` option specifies that when labels are created for the binary variables, zero characters of the original variable name should be used as a prefix. This means that the labels are created only from the level values. For example, `'Mountains'` and `'Bed & Breakfast'` are created as labels not `'Scene Mountains'` and `'Lodge Bed & Breakfast'`.

An `output` statement names the output data set and drops variables that are not needed. These variables do not have to be dropped. However, since they are variable names that are often found in special data set types, PROC PHREG prints warnings when it finds them. Dropping the variables prevents the warnings. Finally, the `id` statement names the additional variables that we want copied from the input to the output data set. The next steps print the first coded choice set.

```
proc print data=coded(obs=6);
   id place;
   var subj set form c price scene lodge;
   run;
proc print data=coded(obs=6) label;
   var pl:;
   run;
proc print data=coded(obs=6) label;
   id place;
   var sc:;
   run;
proc print data=coded(obs=6) label;
   id place;
   var lo: pr:;
   run;
```

Vacation Example

Place	Subj	Set	Form	c	Price	Scene	Lodge
Hawaii	1	1	1	1	999	Beach	Cabin
Alaska	1	1	1	2	999	Mountains	Hotel
Mexico	1	1	1	2	1249	Beach	Hotel
California	1	1	1	2	1249	Lake	Cabin
Maine	1	1	1	2	1249	Mountains	Cabin
Home	1	1	1	2	0	Home	Home

Vacation Example

Obs	Hawaii	Alaska	Mexico	California	Maine	Home	Place
1	1	0	0	0	0	0	Hawaii
2	0	1	0	0	0	0	Alaska
3	0	0	1	0	0	0	Mexico
4	0	0	0	1	0	0	California
5	0	0	0	0	1	0	Maine
6	0	0	0	0	0	1	Home

Vacation Example

Place	Beach	Home	Lake	Mountains	Scene
Hawaii	1	0	0	0	Beach
Alaska	0	0	0	1	Mountains
Mexico	1	0	0	0	Beach
California	0	0	1	0	Lake
Maine	0	0	0	1	Mountains
Home	0	1	0	0	Home

Vacation Example

Place	Bed & Breakfast	Cabin	Home	Hotel	Lodge	0	999	1249	1499	Price
Hawaii	0	1	0	0	Cabin	0	1	0	0	999
Alaska	0	0	0	1	Hotel	0	1	0	0	999
Mexico	0	0	0	1	Hotel	0	0	1	0	1249
California	0	1	0	0	Cabin	0	0	1	0	1249
Maine	0	1	0	0	Cabin	0	0	1	0	1249
Home	0	0	1	0	Home	1	0	0	0	0

The coded design consists of binary variables for destinations Hawaii − Home, scenery Beach − Mountains, lodging Bed & Breakfast − Hotel, and price 0 − 1499. For example, in the last printed panel of the first choice set, the Cabin column has a 1 for Hawaii since Hawaii has Cabin lodging in this choice set. The Cabin column has a 0 for Alaska since Alaska does not have Cabin lodging in this choice set. These binary variables will form the independent variables in the analysis.

Note that we are fitting a model with *generic attributes*. Generic attributes are assumed to be the same for all alternatives. For example, our model is structured so that the part-worth utility for being on a lake will be the same for Hawaii, Alaska, and all of the other destinations. Similarly, the part-worth utilities for the different prices will not depend on the destinations. In contrast, on page 222, using the same data, we will code alternative-specific effects where the part-worth utilities are allowed by the model to be different for each of the destinations.

PROC PHREG is run in the usual way to fit the choice model.

```
proc phreg data=coded brief;
   model c*c(2) = &_trgind / ties=breslow;
   strata subj set;
   run;
```

We specify the `&_trgind` macro variable for the `model` statement independent variable list. PROC TRANSREG automatically creates this macro variable. It contains the list of coded independent variables generated by the procedure. This is so you do not have to figure out what names TRANSREG created and specify them. In this case, PROC TRANSREG sets `&_trgind` to contain the following list.

```
PlaceHawaii PlaceAlaska PlaceMexico PlaceCalifornia PlaceMaine PlaceHome
Price0 Price999 Price1249 Price1499 SceneBeach SceneHome SceneLake
SceneMountains LodgeBed___Breakfast LodgeCabin LodgeHome LodgeHotel
```

The analysis is stratified by subject and choice set. Each stratum consists of a set of alternatives from which a subject made one choice. In this example, each stratum consists of six alternatives, one of which was chosen and five of which were not chosen. (Recall that we used `%phchoice(on)` on page 143 to customize the output from PROC PHREG.) The full table of the strata would be quite large with one line for each of the 3600 strata, so the `brief` option was specified on the PROC PHREG statement. This option produces a brief summary of the strata. In this case, we see there were 3600 choice sets that all fit one response pattern. Each consisted of 6 alternatives, 1 of which was chosen and 5 of which were not chosen. There should be one pattern for all choice sets in an example like this one−the number of alternatives, number of chosen alternatives, and the number not chosen should be constant.

Vacation Example

The PHREG Procedure

Model Information

Data Set	WORK.CODED
Dependent Variable	c
Censoring Variable	c
Censoring Value(s)	2
Ties Handling	BRESLOW

Number of Observations Read	21600
Number of Observations Used	21600

Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Pattern	Number of Choices	Number of Alternatives	Chosen Alternatives	Not Chosen
1	3600	6	1	5

Convergence Status

Convergence criterion (GCONV=1E-8) satisfied.

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	12900.668	6257.752
AIC	12900.668	6279.752
SBC	12900.668	6347.827

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	6642.9164	11	<.0001
Score	5858.3798	11	<.0001
Wald	2482.5118	11	<.0001

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Hawaii	1	3.50429	0.45819	58.4934	<.0001
Alaska	1	0.62029	0.46624	1.7699	0.1834
Mexico	1	2.81487	0.45955	37.5193	<.0001
California	1	2.13549	0.46027	21.5263	<.0001
Maine	1	1.53470	0.46220	11.0253	0.0009
Home	0	0	.	.	.
0	0	0	.	.	.
999	1	3.56656	0.08849	1624.2978	<.0001
1249	1	1.40145	0.08293	285.6189	<.0001
1499	0	0	.	.	.
Beach	1	1.34191	0.06410	438.2880	<.0001
Home	0	0	.	.	.
Lake	1	0.67993	0.06981	94.8542	<.0001
Mountains	0	0	.	.	.
Bed & Breakfast	1	0.64972	0.05363	146.7874	<.0001
Cabin	1	-1.41463	0.07581	348.1654	<.0001
Home	0	0	.	.	.
Hotel	0	0	.	.	.

The destinations, from most preferred to least preferred, are Hawaii, Mexico, California, Maine, Alaska, and then stay at home. The utility for lower price is greater than the utility for higher price. The beach is preferred over a lake, which is preferred over the mountains. A bed & breakfast is preferred over a hotel, which is preferred over a cabin. Notice that the coefficients for the constant alternative (home and zero price) are all zero. Also notice that for each factor, destination, price, scenery and accommodations, the coefficient for the last level is always zero. This will always occur when we code with `zero=none`. The last level of each factor is a reference level, and the other coefficients will have values relative to this zero. For example, all of the coefficients for the destination are positive relative to the zero for staying at home. For scenery, all of the coefficients are positive relative to the zero for the mountains. For accommodations, the coefficient for cabin is less than the zero for hotel, which is less than the coefficient for bed & breakfast. In some sense, each `class` variable in a choice model with a constant alternative has two reference levels or two levels that will always have a zero coefficient: the level corresponding to the constant alternative and the level corresponding to the last level. At first, it is reassuring to run the model with all levels represented to see that all the right levels get zeroed. Later, we will see ways to eliminate these levels from the output.

Quantitative Price Effect

These data can also be analyzed in a different way. The `Price` variable can be specified directly as a quantitative variable, instead of with indicator variables for a qualitative price effect. You could print the independent variable list and copy and edit it, removing the `Price` indicator variables and adding `Price`.

```
%put &_trgind;
```

Alternatively, you could run PROC TRANSREG again with the new coding. We use this latter approach, because it is easier, and it will allow us to illustrate other options. In the previous analysis, there were a number of structural-zeros in the parameter estimate results due to the usage of the `zero=none` option in the PROC TRANSREG coding. This is a good thing, particularly for a first attempt at the analysis. It is good to specify `zero=none` and check the results and make sure you have the right pattern of zeros and nonzeros. Later, you can run again excluding some of the structural zeros. This time, we will explicitly specify the `'Home'` level in the `zero=` option as the reference level so it will be omitted from the `&_trgind` variable list. The first `class` specification specifies `zero='Home'` since there is one variable. The second `class` specification specifies `zero='Home' 'Home'` specifying the reference level for each of the two variables. The variable `Price` is designated as an `identity` variable. The `identity` transformation is the no-transformation option, which is used for variables that need to enter the model with no further manipulations. The `identity` variables are simply copied into the output data set and added to the `&_trgind` variable list. The statement `label price = 'Price'` is specified to explicitly set a label for the `identity` variable price. This is because we explicitly modified PROC PHREG output using `%phchoice(on)` so that the rows of the parameter estimate table would be labeled only with variable labels not variable names. A label for `Price` must be explicitly specified in order for the output to contain a label for the price effect.

```
proc transreg design data=res2 nozeroconstant norestoremissing;
   model class(place / zero='Home' order=data) identity(price)
         class(scene lodge / zero='Home' 'Home' order=formatted) /
         lprefix=0;
   output out=coded(drop=_type_ _name_ intercept);
   label price = 'Price';
   id subj set form c;
   run;
proc phreg data=coded brief;
   model c*c(2) = &_trgind / ties=breslow;
   strata subj set;
   run;
```

Here are the results.

Vacation Example

The PHREG Procedure

Model Information

Data Set	WORK.CODED
Dependent Variable	c
Censoring Variable	c
Censoring Value(s)	2
Ties Handling	BRESLOW

Number of Observations Read	21600
Number of Observations Used	21600

Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Pattern	Number of Choices	Number of Alternatives	Chosen Alternatives	Not Chosen
1	3600	6	1	5

Convergence Status

Convergence criterion (GCONV=1E-8) satisfied.

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	12900.668	6295.152
AIC	12900.668	6315.152
SBC	12900.668	6377.039

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	6605.5164	10	<.0001
Score	5750.9220	10	<.0001
Wald	2483.9241	10	<.0001

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Hawaii	1	14.27118	0.50198	808.2623	<.0001
Alaska	1	11.44532	0.49063	544.1855	<.0001
Mexico	1	13.56216	0.49955	737.0457	<.0001
California	1	12.94025	0.49430	685.3359	<.0001
Maine	1	12.36405	0.49553	622.5618	<.0001
Price	1	-0.00740	0.0001770	1747.2333	<.0001
Beach	1	1.33978	0.06458	430.4561	<.0001
Lake	1	0.71161	0.07131	99.5777	<.0001
Mountains	0	0	.	.	.
Bed & Breakfast	1	0.66233	0.05319	155.0604	<.0001
Cabin	1	-1.33467	0.07353	329.4356	<.0001
Hotel	0	0	.	.	.

The results of the two different analyses are similar. The coefficients for the destinations all increase by a nonconstant amount (approximately 10.8) but the pattern is the same. There is still a negative effect for price. Also, the fit of this model is slightly worse, Chi-Square = 6605.5164, compared to the previous value of 6642.9164 (bigger values mean better fit), because price has one fewer parameter.

Quadratic Price Effect

Previously, we saw price treated as a qualitative factor with two parameters and two *df*, then we saw price treated as a quantitative factor with one parameter and one *df*. Alternatively, we could treat price as quantitative and add a *quadratic* price effect (price squared). Like treating price as a qualitative factor, there are two parameters and two *df* for price. First, we create `PriceL`, the linear price term by centering the original price and dividing by the price increment (250). This maps (999, 1249, 1499) to (-1, 0, 1). Next, we run PROC TRANSREG and PROC PHREG with the new price variables.

```
data res3;
   set res2;
   PriceL = price;
   if price then pricel = (price - 1249) / 250;
   run;

proc transreg design=5000 data=res3 nozeroconstant norestoremissing;
   model class(place / zero='Home' order=data)
         pspline(pricel / degree=2)
         class(scene lodge / zero='Home' 'Home' order=formatted) /
         lprefix=0;
   output out=coded(drop=_type_ _name_ intercept);
   label pricel = 'Price';
   id subj set form c;
   run;
```

The `pspline` or polynomial spline expansion with the `degree=2` option replaces the variable `PriceL` with two coded variables, `PriceL_1` (which is the same as the original `PriceL`) and `PriceL_2` (which is `PriceL` squared). A `degree=2` spline with no knots (neither `knots=` nor `nknots=` were specified) simply expands the variable into a quadratic polynomial.

```
proc phreg data=coded brief;
   model c*c(2) = &_trgind / ties=breslow;
   strata subj set;
   run;
```

This step produced the following results.

Vacation Example

The PHREG Procedure

Model Information

Data Set	WORK.CODED
Dependent Variable	c
Censoring Variable	c
Censoring Value(s)	2
Ties Handling	BRESLOW

Number of Observations Read	21600
Number of Observations Used	21600

Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Pattern	Number of Choices	Number of Alternatives	Chosen Alternatives	Not Chosen
1	3600	6	1	5

Convergence Status

Convergence criterion (GCONV=1E-8) satisfied.

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	12900.668	6257.752
AIC	12900.668	6279.752
SBC	12900.668	6347.827

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	6642.9164	11	<.0001
Score	5858.3798	11	<.0001
Wald	2482.5118	11	<.0001

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Hawaii	1	4.90574	0.45379	116.8713	<.0001
Alaska	1	2.02174	0.46010	19.3081	<.0001
Mexico	1	4.21633	0.45427	86.1476	<.0001
California	1	3.53695	0.45507	60.4085	<.0001
Maine	1	2.93615	0.45761	41.1683	<.0001
Price 1	1	-1.78328	0.04425	1624.2978	<.0001
Price 2	1	0.38183	0.06263	37.1732	<.0001
Beach	1	1.34191	0.06410	438.2880	<.0001
Lake	1	0.67993	0.06981	94.8542	<.0001
Mountains	0	0	.	.	.
Bed & Breakfast	1	0.64972	0.05363	146.7874	<.0001
Cabin	1	-1.41463	0.07581	348.1654	<.0001
Hotel	0	0	.	.	.

The fit is exactly the same as when price was treated as qualitative, Chi-Square = 6642.9164. This is because both models are the same except for the different but equivalent 2 *df* codings of price. The coefficients for the destinations in the two models differ by a constant 1.40145. The coefficients for the factors after price are unchanged. The part-worth utility for \$999 is $-1.78328 \times (999 - 1249)/250 + 0.38183 \times ((999 - 1249)/250)^2 = 2.16511$, the part-worth utility for \$1249 is $-1.78328 \times (1249 - 1249)/250 + 0.38183 \times ((1249 - 1249)/250)^2 = 0$, and the part-worth utility for \$1499 is $-1.78328 \times (1499-1249)/250+0.38183\times((1499-1249)/250)^2 = -1.40145$, which differ from the coefficients when price was treated as qualitative, by a constant -1.40145.

Effects Coding

In the previous analyses, *binary* (1, 0) codings were used for the variables. The next analysis illustrates *effects* (1, 0, -1) coding. The two codings differ in how the final reference level is coded. In binary coding, the reference level is coded with zeros. In effects coding, the reference level is coded with minus ones.

	Binary Coding		Effects Coding	
Levels	One	Two	One	Two
1	1	0	1	0
2	0	1	0	1
3	0	0	-1	-1

In this example, we will use a binary coding for the destinations and effects codings for the attributes.

PROC TRANSREG can be used for effects coding. The `effects` option used inside the parentheses after `class` asks for a (0, 1, -1) coding. The `zero=` option specifies the levels that receive the -1's. PROC PHREG is run with almost the same variable list as before, except now the variables for the

reference levels, those whose parameters are structural zeros are omitted. Refer back to the parameter estimates table on page 212, a few select lines of which are reproduced next:

(Some Lines in the)
Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Home	0	0	.	.	.
0	0	0	.	.	.
1499	0	0	.	.	.
Home	0	0	.	.	.
Mountains	0	0	.	.	.
Home	0	0	.	.	.
Hotel	0	0	.	.	.

Notice that the coefficients for the constant alternative (home and zero price) are all zero. Also notice that for each factor, destination, price, scenery and accommodations, the coefficient for the last level is always zero. In some sense, each `class` variable in a choice model with a constant alternative has two reference levels or two levels that will always have a zero coefficient: the level corresponding to the constant alternative and the level corresponding to the last level. In some of the preceding examples, we eliminated the `'Home'` levels by specifying `zero=Home`. Now we will see how to eliminate all of the structural zeros from the parameter estimate table.

First, for each classification variable, we change the level for the constant alternative to missing. (Recall that they were originally missing and we only made them nonmissing to deliberately produce the zero coefficients.) This will cause PROC TRANSREG to ignore those levels when constructing indicator variables. When you use this strategy, you must specify the `norestoremissing` option in the PROC TRANSREG statement. During the first stage of design matrix creation, PROC TRANSREG puts zeros in the indicator variables for observations with missing `class` levels. At the end, it replaces the zeros with missings, "restoring the missing values." When the `norestoremissing` option is specified, missing values are not restored and we get zeros in the indicator variables for missing `class` levels, which is usually what we want. The DATA step `if` statements recode the constant levels to missing. Next, in PROC TRANSREG, the reference levels `'Mountains'` and `'Hotel'` are listed in the `zero=` option in the `class` specification.

```
data res4;
   set res3;
   if scene = 0 then scene = .;
   if lodge = 0 then lodge = .;
   run;
```

```
proc transreg design=5000 data=res4 nozeroconstant norestoremissing;
   model class(place / zero='Home' order=data)
         pspline(pricel / degree=2)
         class(scene lodge /
               effects zero='Mountains' 'Hotel' order=formatted) /
         lprefix=0;
   output out=coded(drop=_type_ _name_ intercept);
   label pricel = 'Price';
   id subj set form c;
   run;
```

Next, the coded data and design matrix are printed for the first choice set. The coded design matrix begins with five binary columns for the destinations, `'Hawaii'` through `'Maine'`. There is not a column for the stay-at-home destination and the row for stay at home has all zeros in the coded variables. Next is the linear price effect, `'Price 1'`, consisting of 0, 1, and -1. It is followed by the quadratic price effect, `'Price 2'`, which is `'Price 1'` squared. Next are the scenery terms, effects coded. `'Beach'` and `'Lake'` have values of 0 and 1; -1's in the fourth row for the reference level, `'Mountains'`; and zeros in the last row for the stay-at-home alternative. Next are the lodging terms, effects coded. `'Bed & Breakfast'` and `'Cabin'` have values of 0 and 1; -1's in the first, third and fourth row for the reference level, `'Hotel'`; and zeros in the last row for the stay-at-home alternative.

```
proc print data=coded(obs=6) label; run;
```

Vacation Example

Obs	Hawaii	Alaska	Mexico	California	Maine	Price 1	Price 2	Beach	Lake	Bed & Breakfast	Cabin
1	1	0	0	0	0	-1	1	1	0	0	1
2	0	1	0	0	0	-1	1	-1	-1	-1	-1
3	0	0	1	0	0	0	0	1	0	-1	-1
4	0	0	0	1	0	0	0	0	1	0	1
5	0	0	0	0	1	0	0	-1	-1	0	1
6	0	0	0	0	0	0	0	0	0	0	0

Obs	Place	Price	Scene	Lodge	Subj	Set	Form	c
1	Hawaii	-1	Beach	Cabin	1	1	1	1
2	Alaska	-1	Mountains	Hotel	1	1	1	2
3	Mexico	0	Beach	Hotel	1	1	1	2
4	California	0	Lake	Cabin	1	1	1	2
5	Maine	0	Mountains	Cabin	1	1	1	2
6	Home	0	.	.	1	1	1	2

```
proc phreg data=coded brief;
   model c*c(2) = &_trgind / ties=breslow;
   strata subj set;
   run;
```

Vacation Example

The PHREG Procedure

Model Information

Data Set	WORK.CODED
Dependent Variable	c
Censoring Variable	c
Censoring Value(s)	2
Ties Handling	BRESLOW

Number of Observations Read	21600
Number of Observations Used	21600

Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Pattern	Number of Choices	Number of Alternatives	Chosen Alternatives	Not Chosen
1	3600	6	1	5

Convergence Status

Convergence criterion (GCONV=1E-8) satisfied.

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	12900.668	6257.752
AIC	12900.668	6279.752
SBC	12900.668	6347.827

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	6642.9164	11	<.0001
Score	5858.3798	11	<.0001
Wald	2482.5118	11	<.0001

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Hawaii	1	5.32472	0.44985	140.1076	<.0001
Alaska	1	2.44072	0.45690	28.5360	<.0001
Mexico	1	4.63531	0.45052	105.8613	<.0001
California	1	3.95593	0.45176	76.6803	<.0001
Maine	1	3.35513	0.45381	54.6610	<.0001
Price 1	1	-1.78328	0.04425	1624.2978	<.0001
Price 2	1	0.38183	0.06263	37.1732	<.0001
Beach	1	0.66796	0.03582	347.7320	<.0001
Lake	1	0.00599	0.03922	0.0233	0.8787
Bed & Breakfast	1	0.90469	0.03471	679.2342	<.0001
Cabin	1	-1.15966	0.04650	621.9367	<.0001

It is instructive to compare the results of this analysis to the previous analysis on page 217. First, the model fit and chi-square statistics are the same indicating the models are equivalent. The coefficients for the destinations differ by a constant -0.41898, the price coefficients are the same, the scenery coefficients differ by a constant 0.67395, and the lodging coefficients differ by a constant -0.25497. Notice that $-0.41898+0+0.67395+-0.25497=0$, so the utility for each alternative is unchanged by the different but equivalent codings.

Alternative-Specific Effects

In all of the analyses presented so far in this example, we have assumed that the effects for price, scenery, and accommodations are generic or constant across the different destinations. Equivalently, we assumed that destination does not interact with the attributes. Next, we show a model with *alternative-specific effects* that does not make this assumption. The alternative-specific model allows for different price, scenery and lodging effects for each destination. The coding can be done with PROC TRANSREG using its syntax for interactions. Before we do the coding, let's go back to the design preparation stage and redo it in a more normal fashion so reference levels will be omitted from the analysis.

We start by creating the data set `Key`. This step differs from the one we saw on page 200 only in that now we have a missing value for `Place` for the constant alternative.

```
data key;
   input Place $ 1-10 (Lodge Scene Price) ($);
   datalines;
Hawaii      x1  x6   x11
Alaska      x2  x7   x12
Mexico      x3  x8   x13
California  x4  x9   x14
Maine       x5  x10  x15
.           .   .    .
;
```

Next, we use the %MktRoll macro to process the design and the %MktMerge macro to merge the design and data.

```
%mktroll(design=sasuser.Vacation_LinDesBlckd, key=key, alt=place,
         out=sasuser.Vacation_ChDes)

%mktmerge(design=sasuser.Vacation_ChDes, data=results, out=res2, blocks=form,
          nsets=&n, nalts=&m, setvars=choose1-choose&n,
          stmts=%str(price = input(put(price, price.), 5.);
                     format scene scene. lodge lodge.;))

proc print data=res2(obs=12); run;
```

The usage of the %MktRoll macro is exactly the same as we saw on page 200. The %MktMerge macro usage differs from page 208 in that instead of assigning labels and recoding price in a separate DATA step, we now do it directly in the macro. The stmts= option is used to add a price = assignment statement and format statement to the DATA step that merges the two data sets. The statements were included in a %str( ) macro since they contain semicolons. Here are the first two choice sets.

Vacation Example

Obs	Subj	Form	Set	Place	Lodge	Scene	Price	c
1	1	1	1	Hawaii	Cabin	Beach	999	1
2	1	1	1	Alaska	Hotel	Mountains	999	2
3	1	1	1	Mexico	Hotel	Beach	1249	2
4	1	1	1	California	Cabin	Lake	1249	2
5	1	1	1	Maine	Cabin	Mountains	1249	2
6	1	1	1		.	.	.	2
7	1	1	2	Hawaii	Cabin	Beach	999	1
8	1	1	2	Alaska	Bed & Breakfast	Lake	1499	2
9	1	1	2	Mexico	Cabin	Lake	999	2
10	1	1	2	California	Bed & Breakfast	Mountains	999	2
11	1	1	2	Maine	Hotel	Beach	1249	2
12	1	1	2		.	.	.	2

Notice that the attributes for the constant alternative are all missing. Next, we code with PROC TRANSREG. Since we are using missing values for the constant alternative, we must specify the norestoremissing option in the PROC TRANSREG statement. With the norestoremissing option, the indicator variables created for missing class variable values contain all zeros instead of all missings. First, we specify the variable Place as a class variable. Next, we interact Place with all of the attributes, Price, Scene, and Lodge, to create the alternative-specific effects.

```
proc transreg design=5000 data=res2 nozeroconstant norestoremissing;
   model class(place / zero=none order=data)
         class(place * price place * scene place * lodge /
               zero=none order=formatted) / lprefix=0 sep=' ' ', ';
   output out=coded(drop=_type_ _name_ intercept);
   id subj set form c;
   run;
```

```
proc print data=coded(obs=6) label noobs; run;
```

The coded design matrix consists of:

- five binary columns, `'Hawaii'` through `'Maine'`, for the five destinations,
- fifteen binary columns (5 destinations times 3 prices), `'Alaska, 999'` through `'Mexico, 1499'`, for the alternative-specific price effects,
- fifteen binary columns (5 destinations times 3 sceneries), `'Alaska, Beach'` through `'Mexico, Mountains'`, for the alternative-specific scenery effects,
- fifteen binary columns (5 destinations times 3 lodgings), `'Alaska, Bed & Breakfast'` through `'Mexico, Hotel'`, for the alternative-specific lodging effects.

The entire sixth row of the coded design matrix, the stay-at-home alternative, consists of zeros.

Vacation Example

Hawaii	Alaska	Mexico	California	Maine	Alaska, 999	Alaska, 1249	Alaska, 1499
1	0	0	0	0	0	0	0
0	1	0	0	0	1	0	0
0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0

California, 999	California, 1249	California, 1499	Hawaii, 999	Hawaii, 1249	Hawaii, 1499
0	0	0	1	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Maine, 999	Maine, 1249	Maine, 1499	Mexico, 999	Mexico, 1249	Mexico, 1499	Alaska, Beach	Alaska, Lake	Alaska, Mountains
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1
0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

California, Beach	California, Lake	California, Mountains	Hawaii, Beach	Hawaii, Lake	Hawaii, Mountains
0	0	0	1	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Maine, Beach	Maine, Lake	Maine, Mountains	Mexico, Beach	Mexico, Lake	Mexico, Mountains	Alaska, Bed & Breakfast	Alaska, Cabin	Alaska, Hotel
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

California, Bed & Breakfast	California, Cabin	California, Hotel	Hawaii, Bed & Breakfast	Hawaii, Cabin	Hawaii, Hotel
0	0	0	0	1	0
0	0	0	0	0	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Maine, Bed & Breakfast	Maine, Cabin	Maine, Hotel	Mexico, Bed & Breakfast	Mexico, Cabin	Mexico, Hotel	Place
0	0	0	0	0	0	Hawaii
0	0	0	0	0	0	Alaska
0	0	0	0	0	1	Mexico
0	0	0	0	0	0	California
0	1	0	0	0	0	Maine
0	0	0	0	0	0	

Price	Scene	Lodge	Subj	Set	Form	c
999	Beach	Cabin	1	1	1	1
999	Mountains	Hotel	1	1	1	2
1249	Beach	Hotel	1	1	1	2
1249	Lake	Cabin	1	1	1	2
1249	Mountains	Cabin	1	1	1	2
.	.	.	1	1	1	2

Analysis proceeds by running PROC PHREG as before.

```
proc phreg data=coded brief;
   model c*c(2) = &_trgind / ties=breslow;
   strata subj set;
   run;
```

Vacation Example

The PHREG Procedure

Model Information

Data Set	WORK.CODED
Dependent Variable	c
Censoring Variable	c
Censoring Value(s)	2
Ties Handling	BRESLOW

Number of Observations Read	21600
Number of Observations Used	21600

Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Pattern	Number of Choices	Number of Alternatives	Chosen Alternatives	Not Chosen
1	3600	6	1	5

Convergence Status

Convergence criterion (GCONV=1E-8) satisfied.

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	12900.668	6239.870
AIC	12900.668	6309.870
SBC	12900.668	6526.474

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	6660.7982	35	<.0001
Score	6601.7928	35	<.0001
Wald	2448.1475	35	<.0001

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Hawaii	1	3.49208	0.47222	54.6856	<.0001
Alaska	1	0.17527	0.67139	0.0682	0.7940
Mexico	1	2.93013	0.47932	37.3706	<.0001
California	1	2.17915	0.49725	19.2058	<.0001
Maine	1	1.27770	0.54587	5.4787	0.0192
Alaska, 999	1	4.02423	0.46534	74.7858	<.0001
Alaska, 1249	1	1.81200	0.49473	13.4147	0.0002
Alaska, 1499	0	0	.	.	.
California, 999	1	3.38438	0.19965	287.3498	<.0001
California, 1249	1	1.22372	0.22445	29.7251	<.0001
California, 1499	0	0	.	.	.
Hawaii, 999	1	3.61016	0.14157	650.2879	<.0001
Hawaii, 1249	1	1.45415	0.13050	124.1725	<.0001
Hawaii, 1499	0	0	.	.	.
Maine, 999	1	3.80918	0.26060	213.6577	<.0001
Maine, 1249	1	1.53370	0.27050	32.1475	<.0001
Maine, 1499	0	0	.	.	.
Mexico, 999	1	3.45924	0.15495	498.4209	<.0001
Mexico, 1249	1	1.41406	0.15693	81.1907	<.0001
Mexico, 1499	0	0	.	.	.
Alaska, Beach	1	1.01542	0.21881	21.5355	<.0001
Alaska, Lake	1	0.48168	0.22639	4.5269	0.0334
Alaska, Mountains	0	0	.	.	.
California, Beach	1	1.47681	0.15536	90.3528	<.0001
California, Lake	1	0.84358	0.16138	27.3244	<.0001
California, Mountains	0	0	.	.	.
Hawaii, Beach	1	1.29573	0.12493	107.5692	<.0001
Hawaii, Lake	1	0.61301	0.12299	24.8444	<.0001
Hawaii, Mountains	0	0	.	.	.
Maine, Beach	1	1.59739	0.20874	58.5584	<.0001
Maine, Lake	1	0.64984	0.20203	10.3468	0.0013
Maine, Mountains	0	0	.	.	.
Mexico, Beach	1	1.26780	0.13744	85.0857	<.0001
Mexico, Lake	1	0.67632	0.13589	24.7716	<.0001
Mexico, Mountains	0	0	.	.	.

Alaska, Bed & Breakfast	1	1.00195	0.18862	28.2169	<.0001
Alaska, Cabin	1	-1.33747	0.31958	17.5146	<.0001
Alaska, Hotel	0	0	.	.	.
California, Bed & Breakfast	1	0.67004	0.13195	25.7875	<.0001
California, Cabin	1	-1.50239	0.16734	80.6060	<.0001
California, Hotel	0	0	.	.	.
Hawaii, Bed & Breakfast	1	0.63585	0.11523	30.4508	<.0001
Hawaii, Cabin	1	-1.41004	0.13462	109.7155	<.0001
Hawaii, Hotel	0	0	.	.	.
Maine, Bed & Breakfast	1	0.58532	0.15999	13.3848	0.0003
Maine, Cabin	1	-1.50967	0.22377	45.5166	<.0001
Maine, Hotel	0	0	.	.	.
Mexico, Bed & Breakfast	1	0.54835	0.11802	21.5891	<.0001
Mexico, Cabin	1	-1.40762	0.15033	87.6707	<.0001
Mexico, Hotel	0	0	.	.	.

There are zero coefficients for the reference level. Do we need this more complicated model instead of the simpler model? To answer this, first look at the coefficients. Are they similar across different destinations? In this case, they seem to be. This suggests that the simpler model may be sufficient.

More formally, the two models can be statistically compared. You can test the null hypothesis that the two models are not significantly different by comparing their likelihoods. The difference between two $-2\log(\mathcal{L}_C)$'s (the number reported under 'With Covariates' in the output) has a chi-square distribution. We can get the *df* for the test by subtracting the two *df* for the two likelihoods. The difference $6257.752 - 6239.870 = 17.882$ is distributed χ^2 with $35 - 11 = 24$ *df* ($p < 0.80869$). This more complicated model does not account for significantly more variance than the simpler model.

Vacation Example with Alternative-Specific Attributes

This example discusses choosing the number of choice sets, designing the choice experiment, ensuring that certain key interactions are estimable, examining the design, blocking an existing design, evaluating the design, generating the questionnaire, generating artificial data, reading, processing, and analyzing the data, binary coding, generic attributes, alternative-specific effects, aggregating the data, analysis, and interpretation of the results. In this example, a researcher is interested in studying choice of vacation destinations. This page and the next page contain two summaries of the design, one with factors grouped by attribute and one grouped by destination.

This example is a modification of the previous example. Now, all alternatives do not have the same factors, and all factors do not have the same numbers of levels. There are still five destinations of interest: Hawaii, Alaska, Mexico, California, and Maine. Each alternative is composed of three factors like before: package cost, scenery, and accommodations, only now they do not all have the same levels, and the Hawaii and Mexico alternatives are composed of one additional attribute. For Hawaii and Alaska, the costs are \$1,249, \$1,499, and \$1,749; for California, the prices are \$999, \$1,249, \$1,499, and \$1,749; and for Mexico and Maine, the prices are \$999, \$1,249, and \$1,499. Scenery (mountains, lake, beach) and accommodations (cabin, bed & breakfast, and hotel) are the same as before. The Mexico trip now has the option of a side trip to sites of archaeological significance, via bus, for an additional cost of \$100. The Hawaii trip has the option of a side trip to an active volcano, via helicopter, for an additional cost of \$200. This is typical of the problems that marketing researchers face. We have lots of factors and *asymmetry*—each alternative is not composed of the same factors, and the factors do not all have the same numbers of levels.

Factor	Destination	Attribute	Levels
X1	Hawaii	Accommodations	Cabin, Bed & Breakfast, Hotel
X2	Alaska	Accommodations	Cabin, Bed & Breakfast, Hotel
X3	Mexico	Accommodations	Cabin, Bed & Breakfast, Hotel
X4	California	Accommodations	Cabin, Bed & Breakfast, Hotel
X5	Maine	Accommodations	Cabin, Bed & Breakfast, Hotel
X6	Hawaii	Scenery	Mountains, Lake, Beach
X7	Alaska	Scenery	Mountains, Lake, Beach
X8	Mexico	Scenery	Mountains, Lake, Beach
X9	California	Scenery	Mountains, Lake, Beach
X10	Maine	Scenery	Mountains, Lake, Beach
X11	Hawaii	Price	\$1249, \$1499, \$1749
X12	Alaska	Price	\$1249, \$1499, \$1749
X13	Mexico	Price	\$999, \$1249, \$1499
X14	California	Price	\$999, \$1249, \$1499, \$1749
X15	Maine	Price	\$999, \$1249, \$1499
X16	Hawaii	Side Trip	Yes, No
X17	Mexico	Side Trip	Yes, No

Factor	Destination	Attribute	Levels
X1	Hawaii	Accommodations	Cabin, Bed & Breakfast, Hotel
X6		Scenery	Mountains, Lake, Beach
X11		Price	$1249, $1499, $1749
X16		Side Trip	Yes, No
X2	Alaska	Accommodations	Cabin, Bed & Breakfast, Hotel
X7		Scenery	Mountains, Lake, Beach
X12		Price	$1249, $1499, $1749
X3	Mexico	Accommodations	Cabin, Bed & Breakfast, Hotel
X8		Scenery	Mountains, Lake, Beach
X13		Price	$999, $1249, $1499
X17		Side Trip	Yes, No
X4	California	Accommodations	Cabin, Bed & Breakfast, Hotel
X9		Scenery	Mountains, Lake, Beach
X14		Price	$999, $1249, $1499, $1749
X5	Maine	Accommodations	Cabin, Bed & Breakfast, Hotel
X10		Scenery	Mountains, Lake, Beach
X15		Price	$999, $1249, $1499

Choosing the Number of Choice Sets

We can use the `%MktRuns` autocall macro to suggest experimental design sizes. (All of the autocall macros used in this book are documented starting on page 597.) As before, we specify a list containing the number of levels of each factor.

```
title 'Vacation Example with Asymmetry';

%mktruns( 3 ** 14 4 2 2 )
```

The output tells us the size of the saturated design, which is the number of parameters in the linear design, and suggests design sizes.

Vacation Example with Asymmetry

Design Summary

Number of Levels	Frequency
2	2
3	14
4	1

Vacation Example with Asymmetry

Saturated = 34
Full Factorial = 76,527,504

Some Reasonable Design Sizes	Violations	Cannot Be Divided By
72 *	0	
144 *	0	
36	2	8
108	2	8
54	18	4 8 12
90	18	4 8 12
126	18	4 8 12
45	48	2 4 6 8 12
63	48	2 4 6 8 12
81	48	2 4 6 8 12

* - 100% Efficient Design can be made with the MktEx Macro.

Vacation Example with Asymmetry

n	Design	Reference
72	2 ** 20 3 ** 24 4 ** 1	Orthogonal Array
72	2 ** 19 3 ** 20 4 ** 1 6 ** 1	Orthogonal Array
72	2 ** 18 3 ** 16 4 ** 1 6 ** 2	Orthogonal Array
72	2 ** 13 3 ** 25 4 ** 1	Orthogonal Array
72	2 ** 12 3 ** 21 4 ** 1 6 ** 1	Orthogonal Array
72	2 ** 11 3 ** 24 4 ** 1 6 ** 1	Orthogonal Array
72	2 ** 11 3 ** 17 4 ** 1 6 ** 2	Orthogonal Array
72	2 ** 10 3 ** 20 4 ** 1 6 ** 2	Orthogonal Array
72	2 ** 9 3 ** 16 4 ** 1 6 ** 3	Orthogonal Array
144	2 ** 92 3 ** 24 4 ** 1	Orthogonal Array
144	2 ** 91 3 ** 20 4 ** 1 6 ** 1	Orthogonal Array
144	2 ** 90 3 ** 16 4 ** 1 6 ** 2	Orthogonal Array
144	2 ** 85 3 ** 25 4 ** 1	Orthogonal Array
144	2 ** 84 3 ** 21 4 ** 1 6 ** 1	Orthogonal Array
144	2 ** 83 3 ** 24 4 ** 1 6 ** 1	Orthogonal Array
144	2 ** 83 3 ** 17 4 ** 1 6 ** 2	Orthogonal Array
144	2 ** 82 3 ** 20 4 ** 1 6 ** 2	Orthogonal Array
144	2 ** 81 3 ** 16 4 ** 1 6 ** 3	Orthogonal Array
.		
.		
.		

We need at least 34 choice sets, as shown by `(Saturated=34)` in the listing. Any size that is a multiple of 72 would be optimal. We would recommend 72 choice sets, four blocks of size 18. However, like the previous vacation example, we will use fewer choice sets so that we can illustrate getting an efficient but nonorthogonal design. A design with 36 choice sets is pretty good. Thirty-six is not divisible by $8 = 2 \times 4$, so we cannot have equal frequencies in the California price and Mexico and Hawaii side trip combinations. This should not pose any problem. This leaves only 2 error *df* for the linear model, but in the choice model, we will have adequate error *df*.

Designing the Choice Experiment

This problem requires a design with 1 four-level factor for price and 4 three-level factors for price. There are 10 three-level factors for scenery and accommodations as before. Also, we need 2 two-level factors for the two side trips. Note that we do not need a factor for the price or mode of transportation of the side trips since they are constant within each trip. With the `%MktEx` macro, making an asymmetric design is no more difficult than making a symmetric design.[‡]

```
%mktex(3 ** 13 4 3 2 2, n=36, seed=205)
%mkteval;
```

Here is the last part of the results.

Vacation Example with Asymmetry

The OPTEX Procedure

Class Level Information

Class	Levels	Values
x1	3	1 2 3
x2	3	1 2 3
x3	3	1 2 3
x4	3	1 2 3
x5	3	1 2 3
x6	3	1 2 3
x7	3	1 2 3
x8	3	1 2 3
x9	3	1 2 3
x10	3	1 2 3
x11	3	1 2 3
x12	3	1 2 3
x13	3	1 2 3
x14	4	1 2 3 4
x15	3	1 2 3
x16	2	1 2
x17	2	1 2

[‡]Due to machine, SAS release, and macro differences, you may not get exactly the same design as was used in this book, but the differences should be slight.

Vacation Example with Asymmetry

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	98.8874	97.5943	97.4925	0.9718

Vacation Example with Asymmetry
Canonical Correlations Between the Factors
There are 2 Canonical Correlations Greater Than 0.316

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16	x17
x1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
x5	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
x6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
x7	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
x8	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
x9	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
x10	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
x11	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
x12	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
x13	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0.25	0	0
x14	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0.33	0.33
x15	0	0	0	0	0	0	0	0	0	0	0	0	0.25	0	1	0	0
x16	0	0	0	0	0	0	0	0	0	0	0	0	0	0.33	0	1	0
x17	0	0	0	0	0	0	0	0	0	0	0	0	0	0.33	0	0	1

The macro found a very nice, almost orthogonal and almost 99% *D*-efficient design in 40 seconds. However, we will not use this design. Instead, we will make a larger design with interactions.

Ensuring that Certain Key Interactions are Estimable

Next, we will ensure that certain key interactions are estimable. Specifically, it would be good if in the aggregate, the interactions between price and accommodations were estimable for each destination. We would like the following interactions to be estimable: `x1*x11 x2*x12 x3*x13 x4*x15 x5*x15`. We will again use the `%MktEx` macro.

```
%mktex(3 ** 13 4 3 2 2, n=36,
       interact=x1*x11 x2*x12 x3*x13 x4*x14 x5*x15,
       seed=205)
```

We immediately get this message.

```
ERROR: More parameters than runs.
       If you really want to do this, specify RIDGE=.
       There are 36 runs with 56 parameters.
ERROR: The MKTEX macro ended abnormally.
```

If we want interactions to be estimable, we will need more choice sets. The number of parameters is 1 for the intercept, $14 \times (3-1) + (4-1) + 2 \times (2-1) = 33$ for main effects, and $4 \times (3-1) \times (3-1) + (4-1) \times (3-1) = 22$ for interactions for a total of $1 + 33 + 22 = 56$ parameters. This means we need at least 56 choice sets, and ideally for this design with 2, 3, and 4 level factors, we would like the number of sets to be divisible by 2×2, 2×3, 2×4, 3×3, and 3×4. Sixty is divisible by 2, 3, 4, 6, and 12 so is a reasonable design size. Sixty choice sets could be divided into three blocks of size 20, four blocks of size 15, or five blocks of size 12. Seventy-two choice sets would be better, since unlike 60, 72 can be divided by 9. Unfortunately, 72 would require one more block.

We can use the `%MktRuns` macro to help us choose the number of choice sets. We also specified a keyword option `max=` to consider only the 45 design sizes from the minimum of 56 up to 100.

```
title 'Vacation Example with Asymmetry';
%mktruns(3 ** 13 4 3 2 2, interact=x1*x11 x2*x12 x3*x13 x4*x14 x5*x15, max=45)
```

Vacation Example with Asymmetry

Design Summary

Number of Levels	Frequency
2	2
3	14
4	1

Vacation Example with Asymmetry

Saturated = 56
Full Factorial = 76,527,504

Some Reasonable Design Sizes	Violations	Cannot Be Divided By
72	58	27 81 108
81	79	2 4 6 8 12 18 24 36 108
90	95	4 8 12 24 27 36 81 108
63	133	2 4 6 8 12 18 24 27 36 81 108
99	133	2 4 6 8 12 18 24 27 36 81 108
96	174	9 18 27 36 81 108
60	178	8 9 18 24 27 36 81 108
84	178	8 9 18 24 27 36 81 108
66	194	4 8 9 12 18 24 27 36 81 108
78	194	4 8 9 12 18 24 27 36 81 108

We see that 72 cannot be divided by $27 = 9 \times 3$ so for example the Maine accommodation/price combinations cannot occur with equal frequency with each of the three-level factors. We see that 72 cannot be divided by $81 = 9 \times 9$ so for example the Mexico accommodation/price combinations cannot occur with equal frequency with each of the Hawaii accommodation/price combinations. We see that 72 cannot be divided by $108 = 9 \times 12$ so for example the California accommodation/price combinations cannot occur with equal frequency with each of the Maine accommodation/price combinations. With interactions, there are many higher-order opportunities for nonorthogonality. However, usually we will not be overly concerned about potential unequal cell frequencies on combinations of attributes in different alternatives.

The smallest number of runs in the table is 60. While 72 is better in that it can be divided by more numbers, either 72 or 60 should work fine. We will pick the larger number and run the `%MktEx` macro again with `n=72` specified.

```
%mktex(3 ** 13 4 3 2 2, n=72, seed=368,
       interact=x1*x11 x2*x12 x3*x13 x4*x14 x5*x15)
```

Here is the final *D*-efficiency table.

Vacation Example with Asymmetry

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	89.8309	79.7751	94.4393	0.8819

Sometimes, particularly in models with two-way interactions, we can do better by having `%MktEx` do pair-wise exchanges in the coordinate-exchange algorithm instead of working on a single column at a time. You can always specify `exchange=2` and `order=sequential` to get all possible pairs, but this can be very time consuming. Alternatively, you can use the `order=matrix`=*SAS-data-set* option and tell `%MktEx` exactly which pairs of columns to work on. That approach is illustrated in the next steps.

```
data mat;
   do a = 1 to 17;
      b = .;
      output;
      end;
   do a = 1 to 5;
      b = 10 + a;
      output;
      end;
   run;

proc print; run;

%mktex(3 ** 13 4 3 2 2, n=72, seed=368, order=matrix=mat,
       interact=x1*x11 x2*x12 x3*x13 x4*x14 x5*x15)
```

Here is the data set `Mat`.

Vacation Example with Asymmetry

Obs	a	b
1	1	.
2	2	.
3	3	.
4	4	.
5	5	.
6	6	.
7	7	.
8	8	.
9	9	.
10	10	.
11	11	.
12	12	.
13	13	.
14	14	.
15	15	.
16	16	.
17	17	.
18	1	11
19	2	12
20	3	13
21	4	14
22	5	15

It has two columns. The values in the data set indicate the pairs of columns that `%MktEx` should work on together. Missing values are replaced by a random column number for every row and for every pass through the design. This data set instructs `%MktEx` to sequentially go through every column each time paired with some other random column, then work through all of the interaction pairs, `x1*x11`, `x2*x12`, and so on. This performs 22 pair-wise exchanges in every row, which is many fewer exchanges than the $17 \times 16/2 = 136$ that would be required with `exchange=2` and `order=sequential`. There are many other combinations that you might consider. Here are a few examples.

One		Two		Three		Four		Five		
1	.	1	11	1	1	1	11	1	.	.
2	.	2	12	2	2	2	12	2	.	.
3	.	3	13	3	3	3	13	3	.	.
4	.	4	14	4	4	4	14	4	.	.
5	.	5	15	5	5	5	15	5	.	.
6	.	6	.	6	6	6	6	6	.	.
7	.	7	.	7	7	7	7	7	.	.
8	.	8	.	8	8	8	8	8	.	.
9	.	9	.	9	9	9	9	9	.	.
10	.	10	.	10	10	10	10	10	.	.
11	.			11	11			11	.	.
12	.			12	12			12	.	.
13	.			13	13			13	.	.
14	.			14	14			14	.	.
15	.			15	15			15	.	.
16	.			16	16			16	.	.
17	.			17	17			17	.	.
1	11			1	11			1	11	.
2	12			2	12			2	12	.
3	13			3	13			3	13	.
4	14			4	14			4	14	.
5	15			5	15			5	15	.

Set one is the set we just used. Each column is paired with a random column and every interaction pair is mentioned. Set two is like set one except it consists of only 10 pairs and the interaction columns are only paired with the other columns in its interaction term. Set three names each factor twice and then has the usual pairs for interactions. This requests 17 single-column exchanges followed by 5 pair-wise exchanges. When a column is repeated, all but the first instance is ignored. `%MktEx` does not consider all pairs of a factor with itself. Set four is similar to set 3 but the interaction columns are only paired with the other columns in its interaction term. Set five is like set one except three-way exchanges are performed and a random column is added to each exchange. There are many other possibilities. It is impossible to know what will work best, but often, expending some effort to consider exchanges in pairs for two-way interactions or in triples for three-way interactions is rewarded with a small gain in D-efficiency.

The macro printed these notes to the log.

```
NOTE: Generating the candidate set.
NOTE: Performing 20 searches of 243 candidates, full-factorial=76,527,504.
NOTE: Generating the orthogonal array design, n=72.
```

The candidate-set search is using a fractional-factorial candidate set with $3^5 = 243$ candidates. The two-level factors in the candidate set are made from three-level factors by coding down. *Coding down* replaces an m-level factor with a factor with fewer than m levels, for example a two-level factor could be created from a three-level factor: ((1 2 3) ⇒ (1 2 1)). The four-level factor in the candidate set is made from 2 three-level factors and coding down. ((1 2 3)×(1 2 3) ⇒ (1 2 3 4 5 6 7 8 9) ⇒ (1 2 3 4 1 2 3 4 1)). The tabled design used for the partial initialization in the coordinate-exchange steps has 72 runs. Here are some of the results.

Vacation Example with Asymmetry

Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
1	Start	84.8774	84.8774	Can
1	End	84.8774		
2	Start	41.1012		Tab
2	8 5	84.9292	84.9292	
2	16 2	84.9450	84.9450	
2	17 15	85.0008	85.0008	
2	18 3	85.0423	85.0423	
.				
.				
.				
2	42 14	87.3823	87.3823	
2	66 5	87.4076	87.4076	
2	2 13	87.4113	87.4113	
2	End	87.4113		
.				
.				
.				
11	Start	41.1012		Tab
11	End	87.2914		
12	Start	55.7719		Ran,Mut,Ann
12	53 16	87.4195	87.4195	
12	48 9	87.4355	87.4355	
12	49 6	87.4688	87.4688	
12	50 1	87.4688	87.4688	
.				
.				
.				
12	9 4	90.3157	90.3157	
12	End	90.3157		
.				
.				
.				
17	Start	58.3436		Ran,Mut,Ann
17	End	90.5685		

NOTE: Quitting the algorithm search step after 10.03 minutes and 17 designs.

Design Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
0	Initial	90.7877	90.7877	Ini
1	Start	59.2298		Ran,Mut,Ann
1	End	90.3158		
.				
.				
.				
14	Start	58.8515		Ran,Mut,Ann
14	End	90.3433		

NOTE: Quitting the design search step after 20.17 minutes and 14 designs.

Vacation Example with Asymmetry

Design Refinement History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
0	Initial	90.7877	90.7877	Ini
1	Start	88.9486		Pre,Mut,Ann
1	End	90.6106		
2	Start	87.6249		Pre,Mut,Ann
2	64 4	90.7886	90.7886	
2	End	90.7803		
.				
.				
.				
5	Start	89.3771		Pre,Mut,Ann
5	End	90.5049		

NOTE: Quitting the refinement step after 5.60 minutes and 5 designs.

Vacation Example with Asymmetry

The OPTEX Procedure

Class Level Information

Class	Levels	Values
x1	3	1 2 3
x2	3	1 2 3
x3	3	1 2 3
x4	3	1 2 3
x5	3	1 2 3
x6	3	1 2 3
x7	3	1 2 3
x8	3	1 2 3
x9	3	1 2 3
x10	3	1 2 3
x11	3	1 2 3
x12	3	1 2 3
x13	3	1 2 3
x14	4	1 2 3 4
x15	3	1 2 3
x16	2	1 2
x17	2	1 2

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	90.7886	81.5474	93.3553	0.8819

The macro ran in approximately 36 minutes. The algorithm search history shows that the candidate-set approach (`Can`) used in design 1 found a design that was 84.8774% D-efficient. The macro makes no attempt to improve on this design, unless there are restriction on the design, until the end in the design refinement step, and only if it is the best design found.

Designs 2 through 11 used the coordinate-exchange algorithm with a tabled design initialization (`Tab`). For this problem, the tabled design initialization initializes all 72 rows; For other problems, when the number of runs in the design is greater than the number of runs in the nearest tabled design, the remaining rows would be randomly initialized. The tabled design initialization usually works very well when all but at most a very few rows and columns are left uninitialized and there are no interactions or restrictions. That is not the case in this problem, and when the algorithm switches to a fully-random initialization in design 12, it immediately does better.

The algorithm search phase picked the coordinate-exchange algorithm with a random initialization, random mutations, and simulated annealing as the algorithm to use in the next step, the design search step. The design search history is initialized with the best design (D-efficiency = 90.7877) found so far. The design search phase starts out with the initial design (`Ini`) found in the algorithm search phase.

The macro finds a design with D-efficiency = 90.3433.

The final set of iterations tries to improve the best design found so far. Random mutations (`Ran`), simulated annealing (`Ann`), and level exchanges are used on the previous best (`Pre`) design. The random mutations are responsible for making the D-efficiency of the starting design worse than the previous best D-efficiency. In this case, the design refinement step found a very slight improvement on the best design found by the design search step.

Each stage ended before the maximum number of iterations and printed a note. All three notes appear next.

```
NOTE: Quitting the algorithm search step after 10.03 minutes and 17 designs.
NOTE: Quitting the design search step after 20.17 minutes and 14 designs.
NOTE: Quitting the refinement step after 5.60 minutes and 5 designs.
```

The default values for `maxtime=10 20 5` constrain the three steps to run in an approximate maximum time of 10, 20, and 5 minutes. Fewer iterations are performed with `order=matrix` than with the default single-column exchanges because each pair of exchanges takes longer than a single exchange. For example, with two three-level factors, a pair-wise exchange considers $3 \times 3 = 9$ exchanges, whereas a single exchange considers 3 exchanges. However, a single design with a random initialization and annealing, would have been faster and better than the full `%MktEx` run with a single-column exchange. This could be requested as follows.

```
data mat;
   do a = 1 to 17;
      b = .;
      output;
      end;
   do a = 1 to 5;
      b = 10 + a;
      output;
      end;
   run;

proc print; run;

%mktex(3 ** 13 4 3 2 2, n=72, seed=368, order=matrix=mat,
       optiter=0, tabiter=0, maxdesigns=1,
       interact=x1*x11 x2*x12 x3*x13 x4*x14 x5*x15)
```

These steps were not run, and we will use the design created with the previous steps.

Examining the Design

We can use the %MktEval macro to evaluate the goodness of this design.

```
%mkteval(data=design);
```

Here are some of the results.

Vacation Example with Asymmetry
Canonical Correlations Between the Factors
There are 0 Canonical Correlations Greater Than 0.316

	x1	x2	x3	x4	x5	x6	x7	x8	x9
x1	1	0.11	0.09	0.17	0.14	0.03	0.09	0.11	0.09
x2	0.11	1	0.18	0.13	0.09	0.14	0.09	0.07	0.15
x3	0.09	0.18	1	0.20	0.11	0.09	0.09	0.13	0.09
x4	0.17	0.13	0.20	1	0.13	0.11	0.07	0.09	0.15
x5	0.14	0.09	0.11	0.13	1	0.03	0.12	0.05	0.13
x6	0.03	0.14	0.09	0.11	0.03	1	0.10	0.04	0.11
x7	0.09	0.09	0.09	0.07	0.12	0.10	1	0.08	0.09
x8	0.11	0.07	0.13	0.09	0.05	0.04	0.08	1	0.07
x9	0.09	0.15	0.09	0.15	0.13	0.11	0.09	0.07	1

.
.
.

Vacation Example with Asymmetry
Summary of Frequencies
There are 0 Canonical Correlations Greater Than 0.316
* - Indicates Unequal Frequencies

		Frequencies
*	x1	25 24 23
*	x2	24 25 23
	x3	24 24 24
*	x4	25 24 23
*	x5	25 23 24
*	x6	26 22 24
*	x7	22 26 24
*	x8	23 26 23
*	x9	23 27 22
*	x10	24 26 22
*	x11	23 24 25
*	x12	23 25 24
*	x13	25 23 24
*	x14	19 17 19 17
	x15	24 24 24
	x16	36 36
*	x17	37 35

```
*    x1 x2      7 9 9 8 8 8 9 8 6
*    x1 x3      9 9 7 7 8 9 8 7 8
*    x1 x4      7 9 9 11 7 6 7 8 8
*    x1 x5      8 8 9 10 8 6 7 7 9
*    x1 x6      9 8 8 9 7 8 8 7 8
*    x1 x7      8 10 7 6 9 9 8 7 8
*    x1 x8      9 8 8 8 8 8 6 10 7
*    x1 x9      7 9 9 8 9 7 8 9 6
*    x1 x10     9 8 8 8 10 6 7 8 8
*    x1 x11     8 8 9 7 8 9 8 8 7
*    x1 x12     7 8 10 9 8 7 7 9 7
*    x1 x13     9 9 7 7 8 9 9 6 8
*    x1 x14     7 6 7 5 7 4 6 7 5 7 6 5
*    x1 x15     10 7 8 8 8 8 6 9 8
*    x1 x16     12 13 12 12 12 11
*    x1 x17     12 13 14 10 11 12
.
.
.
*    x12 x13    6 9 8 10 6 9 9 8 7
*    x12 x14    5 5 6 7 8 6 6 5 6 6 7 5
*    x12 x15    7 9 7 9 7 9 8 8 8
*    x12 x16    12 11 13 12 11 13
*    x12 x17    12 11 13 12 12 12
.
.
.
     N-Way      1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
                1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
                1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
                1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

We can use the `%MktEx` macro to check the design and print the information matrix and variance matrix.

```
%mktex(3 ** 13 4 3 2 2, n=72, examine=i v, options=check, init=randomized,
       interact=x1*x11 x2*x12 x3*x13 x4*x14 x5*x15)
```

Here is a small part of the results.

Vacation Example with Asymmetry
Information Matrix

	Intercept	x11	x12	x21	x22	x31	x32	x41
Intercept	72.000	2.121	-1.225	2.121	1.225	0	0	2.121
x11	2.121	73.500	0.866	1.500	-6.062	-6.000	0	-7.500
x12	-1.225	0.866	70.500	0.866	4.500	-1.732	-3.000	-9.526
x21	2.121	1.500	0.866	73.500	-0.866	-6.000	0	-3.000
x22	1.225	-6.062	4.500	-0.866	70.500	-3.464	-12.000	1.732
x31	0	-6.000	-1.732	-6.000	-3.464	72.000	0	-1.500
x32	0	0	-3.000	0	-12.000	0	72.000	2.598
x41	2.121	-7.500	-9.526	-3.000	1.732	-1.500	2.598	73.500

.
.
.

Vacation Example with Asymmetry
Variance Matrix

	Intercept	x11	x12	x21	x22	x31	x32	x41
Intercept	0.015	-0.001	0.000	-0.001	-0.001	-0.000	-0.000	-0.001
x11	-0.001	0.017	-0.001	0.001	0.002	0.001	0.001	0.002
x12	0.000	-0.001	0.017	-0.001	-0.001	0.001	0.000	0.002
x21	-0.001	0.001	-0.001	0.017	0.000	0.002	0.000	0.001
x22	-0.001	0.002	-0.001	0.000	0.018	0.001	0.003	0.000
x31	-0.000	0.001	0.001	0.002	0.001	0.016	0.000	-0.000
x32	-0.000	0.001	0.000	0.000	0.003	0.000	0.018	-0.001
x41	-0.001	0.002	0.002	0.001	0.000	-0.000	-0.001	0.017

.
.
.

Blocking an Existing Design

An existing design is blocked using the `%MktBlock` macro. The macro takes the observations in an existing design and optimally sorts them into blocks. Here, we are seeing how to block the linear version of the choice design, but the macro can also be used directly on the choice design.

```
%mktblock(data=randomized, nblocks=4, out=sasuser.AsymVac_LinDesBlckd, seed=114)
```

This step took 2 seconds. Here are some of the results including the one-way frequencies within blocks. They should be examined to ensure that each level is well represented in each block. The design is nearly balanced in most of the factors and blocks.

Vacation Example with Asymmetry
Canonical Correlations Between the Factors
There are 0 Canonical Correlations Greater Than 0.316

	Block	x1	x2	x3	x4	x5	x6	x7	x8
Block	1	0.08	0.06	0.10	0.06	0.08	0.08	0.08	0.06
x1	0.08	1	0.11	0.09	0.17	0.14	0.03	0.09	0.11
x2	0.06	0.11	1	0.18	0.13	0.09	0.14	0.09	0.07
x3	0.10	0.09	0.18	1	0.20	0.11	0.09	0.09	0.13
x4	0.06	0.17	0.13	0.20	1	0.13	0.11	0.07	0.09
x5	0.08	0.14	0.09	0.11	0.13	1	0.03	0.12	0.05
x6	0.08	0.03	0.14	0.09	0.11	0.03	1	0.10	0.04
x7	0.08	0.09	0.09	0.09	0.07	0.12	0.10	1	0.08
x8	0.06	0.11	0.07	0.13	0.09	0.05	0.04	0.08	1

.
.
.

Vacation Example with Asymmetry
Summary of Frequencies
There are 0 Canonical Correlations Greater Than 0.316
* - Indicates Unequal Frequencies

Frequencies

	Block	18 18 18 18
*	x1	25 23 24
*	x2	25 24 23
	x3	24 24 24
*	x4	25 24 23
*	x5	25 24 23
*	x6	22 26 24
*	x7	24 26 22
*	x8	26 23 23
*	x9	23 22 27
*	x10	24 26 22
*	x11	23 24 25
*	x12	24 23 25
*	x13	24 23 25
*	x14	17 19 19 17
	x15	24 24 24
	x16	36 36
*	x17	37 35

```
        *    Block x1     6 6 6 6 5 7 6 6 6 7 6 5
        *    Block x2     6 6 6 6 6 6 7 6 5 6 6 6
        *    Block x3     6 6 6 6 6 6 6 7 5 6 5 7
        *    Block x4     7 6 5 6 6 6 6 6 6 6 6 6
        *    Block x5     6 6 6 7 5 6 6 7 5 6 6 6
        *    Block x6     6 7 5 6 6 6 5 7 6 5 6 7
        *    Block x7     6 7 5 5 7 6 7 6 5 6 6 6
        *    Block x8     6 6 6 7 5 6 7 6 5 6 6 6
        *    Block x9     7 5 6 5 5 8 6 6 6 5 6 7
        *    Block x10    5 7 6 6 6 6 6 7 5 7 6 5
        *    Block x11    4 7 7 7 5 6 5 6 7 7 6 5
        *    Block x12    5 6 7 7 6 5 5 6 7 7 5 6
        *    Block x13    6 6 6 6 5 7 6 6 6 6 6 6
        *    Block x14    3 5 4 6 5 5 5 3 4 5 6 3 5 4 4 5
        *    Block x15    6 6 6 6 5 7 5 7 6 7 6 5
             Block x16    9 9 9 9 9 9 9 9
        *    Block x17    9 9 9 9 10 8 9 9
        .
        .
        .
             N-Way        1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
                          1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
                          1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
                          1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Collecting data is time consuming and expensive. Before collecting data, it is a good practice to convert your linear design into a choice design and evaluate it in the context of a choice model. We start by creating some formats for the factor levels and the key to converting the linear design into a choice design.[§]

```
proc format;
   value price 1 = ' 999'      2 = '1249' 3 = '1499' 4 = '1749'  . = ' ';
   value scene 1 = 'Mountains' 2 = 'Lake'            3 = 'Beach' . = ' ';
   value lodge 1 = 'Cabin'     2 = 'Bed & Breakfast' 3 = 'Hotel' . = ' ';
   value side  1 = 'Side Trip' 2 = 'No'                          . = ' ';
   run;

data key;
   input Place $ 1-10 (Lodge Scene Price Side) ($);
   datalines;
Hawaii      x1  x6   x11  x16
Alaska      x2  x7   x12  .
Mexico      x3  x8   x13  x17
California  x4  x9   x14  .
Maine       x5  x10  x15  .
.           .   .    .    .
;
```

[§]See page 60 for an explanation of linear versus choice designs.

For analysis, the design will have five attributes. `Place` is the alternative name. `Lodge`, `Scene`, `Price` and `Side` are created from the design using the indicated factors. See page 200 for more information on creating the design key. Notice that `Side` only applies to some of the alternatives and hence has missing values for the others. Processing the design and merging it with the data are similar to what was done on pages 200 and 208. One difference is now there are asymmetries in `Price`. For Hawaii's price, `x11`, we need to change 1, 2, and 3 to \$1249, \$1499, and \$1749. For Alaska's price, `x12`, we need to change 1, 2, and 3 to \$1249, \$1499, and \$1749. For Mexico's price, `x13`, we need to change 1, 2, and 3 to \$999, \$1249, and \$1499. For California's price, `x14`, we need to change 1, 2, 3, and 4 to \$999, \$1249, \$1499, and \$1749. For Maine's price, `x11`, we need to change 1, 2, and 3 to \$999, \$1249, and \$1499. We can simplify the problem by adding 1 to `x11` and `x12`, which are the factors that start at \$1249 instead of \$999. This will allow us to use a common format to set the price. See pages 311 and 622 for examples of handling more complicated asymmetries.

```
data temp;
   set sasuser.AsymVac_LinDesBlckd(rename=(block=Form));
   x11 + 1;
   x12 + 1;
   run;

%mktroll(design=temp, key=key, alt=place, out=sasuser.AsymVac_ChDes,
         options=nowarn, keep=form)

data sasuser.AsymVac_ChDes;
   set sasuser.AsymVac_ChDes;
   format scene scene. lodge lodge. side side. price price.;
   run;

proc print data=sasuser.AsymVac_ChDes(obs=12);
   by form set; id form set;
   run;
```

Here are the first two choice sets. Notice that each has six alternatives, one of which is printing in this format as all blank.

Vacation Example with Asymmetry

Form	Set	Place	Lodge	Scene	Price	Side
1	1	Hawaii	Bed & Breakfast	Lake	1499	No
		Alaska	Bed & Breakfast	Mountains	1249	
		Mexico	Cabin	Lake	999	No
		California	Cabin	Lake	1249	
		Maine	Hotel	Beach	1249	

```
  1      2     Hawaii        Hotel             Lake         1749     Side Trip
               Alaska        Hotel             Lake         1499
               Mexico        Hotel             Beach        1249     No
               California    Cabin             Beach         999
               Maine         Cabin             Lake         1249
```

Testing the Design Before Data Collection

Collecting data is time consuming and expensive. It is always good practice to make sure that the design will work with the most complicated model that we anticipate fitting. The following code evaluates the choice design.

```
title2 'Evaluate the Choice Design';

%choiceff(data=sasuser.AsymVac_ChDes, init=sasuser.AsymVac_ChDes(keep=set),
          nsets=72, nalts=6, beta=zero, intiter=0,
          model=class(place / zero=none order=data)
                class(place * price place * scene place * lodge /
                      zero=none order=formatted separators='' ' ')
                class(place * side / zero=' ' 'No' separators='' ' ') /
                lprefix=0 cprefix=0)
```

We use the `%ChoicEff` macro to evaluate our choice design. Normally, you would use this macro to search a candidate set for an efficient choice design. You can also use it to evaluate a design created by other means. The way you check a design like this is to first name it on the `data=` option. This will be the candidate set that contains all of the choice sets that we will consider. In addition, the same design is named on the `init=` option. The full specification is `init=sasuser.AsymVac_ChDes(keep=set)`. Just the variable `Set` is kept. It will be used to bring in just the indicated choice sets from the `data=` design, which in this case is all of them. The option `nsets=72` specifies the number of choice sets, and `nalts=6` specifies the number of alternatives. The option `beta=zero` specifies that we are assuming for design evaluation purpose that all of the betas or part-worth utilities are zero. You can evaluate the design for other parameter vectors by specifying a list of numbers after `beta=`. This will change the variances and standard errors. We also specify `intiter=0` which specifies zero internal iterations. We use zero internal iterations when we want to evaluate an initial design, but not attempt to improve it. The last option specifies the model.

The model specification contains everything that appears on the TRANSREG procedure's `model` statement for coding the design. Many of these options should be familiar from previous examples. The specification `class(place / zero=none order=data)` names the `place` variable as a classification variable and asks for coded variables for every nonmissing level (`zero=none`). The order of the levels on output matches the order that the levels are first encountered in the input data set. This specification creates the alternative effects or alternative-specific intercepts.

The next specification, `class(place * price place * scene place * lodge / zero=none order=formatted separators='' ' ')` requests alternative-specific effects for all of the attributes except the side trip. The alternative-specific effects are requested by interacting the alternative-specific intercepts, in this case the destination, with the attributes. The `zero=none` option creates binary variables for all categories. In contrast, by default, a variable is not created for the last category−the parameter

for the last category is a structural zero. The `zero=none` option is used when you want to see the structural zeros in the results. The `separators='' ' '` option (`separators=` quote quote space quote space quote, which provides two strings (one null and the other blank), allows you to specify two label component separators for the main effect and interaction terms, respectively. By specifying a blank for the second value, we request labels for the side trip effects like `'Mexico Side Trip'` instead of the default `'Mexico * Side Trip'`. This option is explained in more detail on page 265.

The last part of the model specification consists of `class(place * side / zero=' ' 'No' separators ='' ' ')` and creates the alternative-specific side trip effects with all levels for `place` and `'No'` as the reference level for the side trip factor. The last part of the model specification is followed by a slash and some options: `/ lprefix=0 cprefix=0)`. The `cprefix=0` option specifies that when names are created for the binary variables, zero characters of the original variable name should be used as a prefix. This means that the names are created only from the level values. The `lprefix=0` option specifies that when labels are created for the binary variables, zero characters of the original variable name should be used as a prefix. This means that the labels are created only from the level values.

Here is the last part of the output.

Vacation Example with Asymmetry
Evaluate the Choice Design

n	Variable Name	Label	Variance	DF	Standard Error
1	Hawaii	Hawaii	0.91061	1	0.95426
2	Alaska	Alaska	0.70276	1	0.83831
3	Mexico	Mexico	0.79649	1	0.89246
4	California	California	0.89577	1	0.94645
5	Maine	Maine	0.82172	1	0.90649
6	Alaska_999	Alaska 999	.	0	.
7	Alaska_1249	Alaska 1249	0.59635	1	0.77223
8	Alaska_1499	Alaska 1499	0.60551	1	0.77814
9	Alaska_1749	Alaska 1749	.	0	.
10	California_999	California 999	0.85492	1	0.92462
11	California_1249	California 1249	0.81130	1	0.90072
12	California_1499	California 1499	0.82552	1	0.90858
13	California_1749	California 1749	.	0	.
14	Hawaii_999	Hawaii 999	.	0	.
15	Hawaii_1249	Hawaii 1249	0.60792	1	0.77969
16	Hawaii_1499	Hawaii 1499	0.59679	1	0.77252
17	Hawaii_1749	Hawaii 1749	.	0	.
18	Maine_999	Maine 999	0.60676	1	0.77894
19	Maine_1249	Maine 1249	0.61109	1	0.78172
20	Maine_1499	Maine 1499	.	0	.
21	Maine_1749	Maine 1749	.	0	.
22	Mexico_999	Mexico 999	0.59178	1	0.76927
23	Mexico_1249	Mexico 1249	0.60604	1	0.77849
24	Mexico_1499	Mexico 1499	.	0	.
25	Mexico_1749	Mexico 1749	.	0	.

26	AlaskaBeach	Alaska Beach	0.63778	1	0.79861
27	AlaskaLake	Alaska Lake	0.58330	1	0.76374
28	AlaskaMountains	Alaska Mountains	.	0	.
29	CaliforniaBeach	California Beach	0.59453	1	0.77106
30	CaliforniaLake	California Lake	0.67196	1	0.81973
31	CaliforniaMountains	California Mountains	.	0	.
32	HawaiiBeach	Hawaii Beach	0.63923	1	0.79952
33	HawaiiLake	Hawaii Lake	0.61115	1	0.78176
34	HawaiiMountains	Hawaii Mountains	.	0	.
35	MaineBeach	Maine Beach	0.63688	1	0.79805
36	MaineLake	Maine Lake	0.58479	1	0.76471
37	MaineMountains	Maine Mountains	.	0	.
38	MexicoBeach	Mexico Beach	0.59462	1	0.77111
39	MexicoLake	Mexico Lake	0.59710	1	0.77272
40	MexicoMountains	Mexico Mountains	.	0	.
41	AlaskaBed___Breakfast	Alaska Bed & Breakfast	0.62130	1	0.78823
42	AlaskaCabin	Alaska Cabin	0.61012	1	0.78110
43	AlaskaHotel	Alaska Hotel	.	0	.
44	CaliforniaBed___Breakfast	California Bed & Breakfast	0.62122	1	0.78817
45	CaliforniaCabin	California Cabin	0.60866	1	0.78016
46	CaliforniaHotel	California Hotel	.	0	.
47	HawaiiBed___Breakfast	Hawaii Bed & Breakfast	0.61876	1	0.78661
48	HawaiiCabin	Hawaii Cabin	0.59145	1	0.76906
49	HawaiiHotel	Hawaii Hotel	.	0	.
50	MaineBed___Breakfast	Maine Bed & Breakfast	0.61592	1	0.78480
51	MaineCabin	Maine Cabin	0.60681	1	0.77898
52	MaineHotel	Maine Hotel	.	0	.
53	MexicoBed___Breakfast	Mexico Bed & Breakfast	0.61050	1	0.78134
54	MexicoCabin	Mexico Cabin	0.61670	1	0.78530
55	MexicoHotel	Mexico Hotel	.	0	.
56	AlaskaSide_Trip	Alaska Side Trip	.	0	.
57	CaliforniaSide_Trip	California Side Trip	.	0	.
58	HawaiiSide_Trip	Hawaii Side Trip	0.40413	1	0.63572
59	MaineSide_Trip	Maine Side Trip	.	0	.
60	MexicoSide_Trip	Mexico Side Trip	0.40622	1	0.63735
				==	
				38	

It consists of a table with the name and label for each parameter along with its variance, *df*, and standard error. It needs to be carefully evaluated to see if the zeros and nonzeros are in all of the right places. We see one parameter for five of the six destinations, with the constant stay-at-home alternative in all cases excluded from the table. This is followed by four terms for the Alaska price effect. The Alaska at $999 parameter is zero since $999 does not apply to Alaska. The Alaska at $1749 parameter is the reference level and hence is zero. The other two Alaska price parameters are nonzero. Similary, each of the alternative-specific price effects have two or three parameters (the number of applicable prices minus one). For the scenery and accomodations attributes, each alternative has two nonzero parameters and a reference level. There are two nonzero parameters for the side trips for the two applicable destinations. The pattern of zeros and nonzeros looks perfect. There are 38 parameters in the alternative-specific model.

You should also note that the variances and standard errors. They are all apprioximately the same order of magnitude. Sometimes you will see wildly varying parameters. This is usually a sign of a problematic design, perhaps one with too few choice sets for the number of parameters. This design looks good. Note one difference between these results and the results that we see in the previous example on page 204. Here, our standard errors are not constant within an attribute, although they are similar. This is because none of our factors are orthogonal, although they are close.

Generating the Questionnaire

These next steps print the questionnaire.

```
%let m   = 6;                    /* m alts including constant  */
%let mm1 = %eval(&m - 1);        /* m - 1                      */
%let n   = 18;                   /* number of choice sets      */
%let blocks = 4;                 /* number of blocks           */

title;
options ls=80 ps=60 nonumber nodate;

data _null_;
   array dests[&mm1] $ 10 _temporary_ ('Hawaii' 'Alaska' 'Mexico'
                                       'California' 'Maine');
   array scenes[3]   $ 13 _temporary_
                     ('the Mountains' 'a Lake' 'the Beach');
   array lodging[3]  $ 15 _temporary_
                     ('Cabin' 'Bed & Breakfast' 'Hotel');
   array x[15];
   array p[&mm1];
   length price $ 6;
   file print linesleft=ll;

   set sasuser.AsymVac_LinDesBlckd;
   by block;

   p1 = 1499 + (x[11] - 2) * 250;
   p2 = 1499 + (x[12] - 2) * 250;
   p3 = 1249 + (x[13] - 2) * 250;
   p4 = 1374 + (x[14] - 2.5) * 250;
   p5 = 1249 + (x[15] - 2) * 250;

   if first.block then do;
      choice = 0;
      put _page_;
      put @50 'Form: ' block  ' Subject: ________' //;
      end;
   choice + 1;

   if ll < (19 + (x16 = 1) + (x17 = 1)) then put _page_;
   put choice 2. ') Circle your choice of '
       'vacation destinations:' /;
```

```
do dest = 1 to &mm1;
   price = left(put(p[dest], dollar6.));
   put '    ' dest 1. ') ' dests[dest]
       +(-1) ', staying in a ' lodging[x[dest]]
       'near ' scenes[x[&mm1 + dest]] +(-1) ',' /
       +7 'with a package cost of ' price +(-1) @@;
   if dest = 3 and x16 = 1 then
      put ', and an optional visit' / +7
          'to archaeological sites for an additional $100' @@;
   else if dest = 1 and x17 = 1 then
      put ', and an optional helicopter' / +7
          'flight to an active volcano for an additional $200' @@;
   put '.' /;
   end;
put "    &m) Stay at home this year." /;
run;
```

Here are the first two choice sets for the first subject.

```
                                          Form: 1  Subject: ________


1) Circle your choice of vacation destinations:

   1) Hawaii, staying in a Bed & Breakfast near a Lake,
      with a package cost of $1,499.

   2) Alaska, staying in a Bed & Breakfast near the Mountains,
      with a package cost of $1,249.

   3) Mexico, staying in a Cabin near a Lake,
      with a package cost of $999.

   4) California, staying in a Cabin near a Lake,
      with a package cost of $1,249.

   5) Maine, staying in a Hotel near the Beach,
      with a package cost of $1,249.

   6) Stay at home this year.
```

```
2) Circle your choice of vacation destinations:

   1) Hawaii, staying in a Hotel near a Lake,
      with a package cost of $1,749.

   2) Alaska, staying in a Hotel near a Lake,
      with a package cost of $1,499.

   3) Mexico, staying in a Hotel near the Beach,
      with a package cost of $1,249, and an optional visit
      to archaeological sites for an additional $100.

   4) California, staying in a Cabin near the Beach,
      with a package cost of $999.

   5) Maine, staying in a Cabin near a Lake,
      with a package cost of $1,249.

   6) Stay at home this year.
```

In practice, data collection will typically be much more elaborate than this. It may involve art work or photographs, and the choice sets may be presented and the data may be collected through personal interview or over the web. However the choice sets are presented and the data are collected, the essential elements remain the same. Subjects are shown a set of alternatives and are asked to make a choice, then they go on to the next set.

Generating Artificial Data

This next step generates an artificial set of data. Collecting data is time consuming and expensive. Generating some artificial data before the data are collected to test your code and make sure the analysis will run is a good idea. It helps avoid the "How am I going to analyze this?" question from occurring after the data have already been collected. This step generates data for 400 subjects, 100 per block.

```
data _null_;
   array dests[&mm1] _temporary_ (5 -1 4 3 2);
   array scenes[3]   _temporary_ (-1 0 1);
   array lodging[3]  _temporary_ (0 3 2);
   array u[&m];
   array x[15];
   do rep = 1 to 100;
      n = 0;
      do i = 1 to &blocks;
         k + 1;
         if mod(k,3) = 1 then put;
         put k 3. +1 i 1. +2 @@;
```

```
            do j = 1 to &n; n + 1;
               set sasuser.AsymVac_LinDesBlckd point=n;
               do dest = 1 to &mm1;
                  u[dest] = dests[dest] + lodging[x[dest]] +
                            scenes[x[&mm1 + dest]] -
                            x[2 * &mm1 + dest] +
                            2 * normal(17);
               end;
               u[1] = u[1] + (x16 = 1);
               u[3] = u[3] + (x17 = 1);
               u&m  = -3 + 3 * normal(17);
               m = max(of u1-u&m);
               if      abs(u1 - m) < 1e-4 then c = 1;
               else if abs(u2 - m) < 1e-4 then c = 2;
               else if abs(u3 - m) < 1e-4 then c = 3;
               else if abs(u4 - m) < 1e-4 then c = 4;
               else if abs(u5 - m) < 1e-4 then c = 5;
               else                            c = 6;
               put +(-1) c @@;
               end;
            end;
         end;
      stop;
      run;
```

The `dests`, `scenes`, and `lodging` arrays are initialized with part-worth utilities for each level. The utilities for each of the destinations are computed and stored in the array `u` in the statement `u[dest] = ...`, which includes an error term `2 * normal(17)`. The utilities for the side trips are added in separately with `u[1] = u[1] + (x16 = 1)` and `u[3] = u[3] + (x17 = 1)`. The utility for the stay-at-home alternative is `-3 + 3 * normal(17)`. The maximum utility is computed, `m = max(of u1-u&m)` and the alternative with the maximum utility is chosen. The `put` statement writes out the results to the log.

Reading, Processing, and Analyzing the Data

The results from the previous step are pasted into a DATA step and run to mimic reading real input data.

```
title 'Vacation Example with Asymmetry';

data results;
   input Subj Form (choose1-choose&n) (1.) @@;
   datalines;
  1 1 413414111315351335   2 2 115311141441134121   3 3 331451344433513341
  4 4 113111143133311314   5 1 113413531545431313   6 2 145131111414331511
  7 3 313413113111313331   8 4 415143311133541321   9 1 133314111133431113
  .
  .
  .
;
```

The analysis proceeds in a fashion similar to before as in the simpler vacation example on page 208.

```
%mktmerge(design=sasuser.AsymVac_ChDes, data=results, out=res2, blocks=form,
          nsets=&n, nalts=&m, setvars=choose1-choose&n,
          stmts=%str(price = input(put(price, price.), 5.);
                     format scene scene. lodge lodge. side side.;))

proc print data=res2(obs=18);
   id form subj set; by form subj set;
   run;
```

Here are the first three choice sets for the first subject.

Vacation Example with Asymmetry

Form	Subj	Set	Place	Lodge	Scene	Price	Side	c
1	1	1	Hawaii	Bed & Breakfast	Lake	1499	No	2
			Alaska	Bed & Breakfast	Mountains	1249		2
			Mexico	Cabin	Lake	999	No	2
			California	Cabin	Lake	1249		1
			Maine	Hotel	Beach	1249		2
								2
1	1	2	Hawaii	Hotel	Lake	1749	Side Trip	1
			Alaska	Hotel	Lake	1499		2
			Mexico	Hotel	Beach	1249	No	2
			California	Cabin	Beach	999		2
			Maine	Cabin	Lake	1249		2
								2
1	1	3	Hawaii	Hotel	Mountains	1749	Side Trip	2
			Alaska	Hotel	Mountains	1749		2
			Mexico	Bed & Breakfast	Beach	1249	Side Trip	1
			California	Cabin	Lake	1249		2
			Maine	Bed & Breakfast	Mountains	999		2
								2

Indicator variables and labels are created using PROC TRANSREG like before.

```
proc transreg design=5000 data=res2 nozeroconstant norestoremissing;
   model class(place / zero=none order=data)
         class(price scene lodge / zero=none order=formatted)
         class(place * side / zero=' ' 'No' separators='' ' ') /
         lprefix=0;
   output out=coded(drop=_type_ _name_ intercept);
   id subj set form c;
   run;

proc print data=coded(obs=6) label;
   run;
```

The `design=5000` option specifies that no model is fit; the procedure is just being used to code a design in blocks of 5000 observations at a time. The `nozeroconstant` option specifies that if the coding creates

a constant variable, it should not be zeroed. The `norestoremissing` option specifies that missing values should not be restored when the `out=` data set is created. The `model` statement names the variables to code and provides information about how they should be coded. The specification `class(place / ...)` specifies that the variable `Place` is a classification variable and requests a binary coding. The `zero=none` option creates binary variables for all categories. The `order=data` option sorts the levels into the order they were first encountered in the data set. Similarly, the variables `Price`, `Scene`, and `Lodge` are classification variables. The specification `class(place * side / ...)` creates alternative-specific side trip effects. The option `zero=' ' 'No'` specifies that indicator variables should be created for all levels of `Place` except blank, and all levels of `Side` except `'No'`. The specification `zero=' '` is almost the same as `zero=none`. The `zero=' '` specification names a missing level as the reference level creating indicator variables for all nonmissing levels of the `class` variables, just like `zero=none`. The difference is `zero=none` applies to all of the variables named in the `class` specification. When you want `zero=none` to apply to only some variables, then you must use `zero=' '`, as in `zero=' ' 'No'` instead. In this case, `zero=none` applies to the first variable and `zero='No'` applies to the second. With `zero=' '`, TRANSREG prints the following warning, which can be safely ignored.

```
WARNING: Reference level ZERO='' was not found for variable Place.
```

The `separators='' ' '` option (`separators=` quote quote space quote space quote) allows you to specify two label component separators for the main effect and interaction terms, respectively. By specifying a blank for the second value, we request labels for the side trip effects like `'Mexico Side Trip'` instead of the default `'Mexico * Side Trip'`. This option is explained in more detail on page 265.

The `lprefix=0` option specifies that when labels are created for the binary variables, zero characters of the original variable name should be used as a prefix. This means that the labels are created only from the level values. An `output` statement names the output data set and drops variables that are not needed. Finally, the `id` statement names the additional variables that we want copied from the input to the output data set.

Vacation Example with Asymmetry

Obs	Hawaii	Alaska	Mexico	California	Maine	999	1249	1499	1749	Beach	Lake
1	1	0	0	0	0	0	0	1	0	0	1
2	0	1	0	0	0	0	1	0	0	0	0
3	0	0	1	0	0	1	0	0	0	0	1
4	0	0	0	1	0	0	1	0	0	0	1
5	0	0	0	0	1	0	1	0	0	1	0
6	0	0	0	0	0	0	0	0	0	0	0

Obs	Mountains	Bed & Breakfast	Cabin	Hotel	Alaska Side Trip	California Side Trip	Hawaii Side Trip	Maine Side Trip	Mexico Side Trip
1	0	1	0	0	0	0	0	0	0
2	1	1	0	0	0	0	0	0	0
3	0	0	1	0	0	0	0	0	0
4	0	0	1	0	0	0	0	0	0
5	0	0	0	1	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0

Obs	Place	Price	Scene	Lodge	Side	Subj	Set	Form	c
1	Hawaii	1499	Lake	Bed & Breakfast	No	1	1	1	2
2	Alaska	1249	Mountains	Bed & Breakfast		1	1	1	2
3	Mexico	999	Lake	Cabin	No	1	1	1	2
4	California	1249	Lake	Cabin		1	1	1	1
5	Maine	1249	Beach	Hotel		1	1	1	2
6						1	1	1	2

The PROC PHREG specification is the same as we have used before. (Recall that we used `%phchoice(on)` on page 143 to customize the output from PROC PHREG.)

```
proc phreg data=coded brief;
   model c*c(2) = &_trgind / ties=breslow;
   strata subj set;
   run;
```

Here are the results.

Vacation Example with Asymmetry

The PHREG Procedure

Model Information

Data Set	WORK.CODED
Dependent Variable	c
Censoring Variable	c
Censoring Value(s)	2
Ties Handling	BRESLOW

Number of Observations Read	43200
Number of Observations Used	43200

Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Pattern	Number of Choices	Number of Alternatives	Chosen Alternatives	Not Chosen
1	7200	6	1	5

Convergence Status

Convergence criterion (GCONV=1E-8) satisfied.

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	25801.336	12603.247
AIC	25801.336	12631.247
SBC	25801.336	12727.593

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	13198.0891	14	<.0001
Score	12223.0125	14	<.0001
Wald	5081.3295	14	<.0001

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Hawaii	1	3.61141	0.22224	264.0701	<.0001
Alaska	1	-0.94997	0.26364	12.9836	0.0003
Mexico	1	2.26877	0.22776	99.2247	<.0001
California	1	1.54548	0.22760	46.1102	<.0001
Maine	1	0.74153	0.23210	10.2074	0.0014
999	1	2.10214	0.07298	829.7619	<.0001
1249	1	1.44298	0.06078	563.6949	<.0001
1499	1	0.72311	0.05936	148.4188	<.0001
1749	0	0	.	.	.
Beach	1	1.42021	0.04635	938.8384	<.0001
Lake	1	0.72019	0.04472	259.3676	<.0001
Mountains	0	0	.	.	.
Bed & Breakfast	1	0.65045	0.04079	254.3369	<.0001
Cabin	1	-1.42317	0.04809	875.8795	<.0001
Hotel	0	0	.	.	.
Alaska Side Trip	0	0	.	.	.
California Side Trip	0	0	.	.	.
Hawaii Side Trip	1	0.71850	0.05753	155.9801	<.0001
Maine Side Trip	0	0	.	.	.
Mexico Side Trip	1	0.65550	0.06293	108.4863	<.0001

You would not expect the part-worth utilities to match those that were used to generate the data, but you would expect a similar ordering within each factor, and in fact that does occur. These data can also be analyzed with quantitative price effects and destination by attribute interactions, as in the previous vacation example.

Aggregating the Data

This data set is rather large with 43,200 observations. You can make the analysis run faster and with less memory by aggregating. Instead of stratifying on each choice set and subject combination, you can stratify just on choice set and specify the number of times each alternative was chosen and the number of times it was not chosen. First, use PROC SUMMARY to count the number of times each observation occurs. Specify all the analysis variables, and in this example, also specify `Form`. The variable `Form` was added to the list because `Set` designates choice set within form. It is the `Form` and `Set` combinations that identify the choice sets. (In the previous PROC PHREG step, since the `Subj * Set` combinations uniquely identified each stratum, `Form` was not needed.) PROC SUMMARY stores the number of times each unique observation appears in the variable `_freq_`. PROC PHREG is then run with a `freq` statement. Now, instead of analyzing a data set with 43,200 observations and 7200 strata, we analyze a data set with at most $2 \times 6 \times 72 = 864$ observations and 72 strata. For each of the 6 alternatives and 72 choice sets, there are typically 2 observations in the aggregate data set: one that contains the number of times it was chosen and one that contains the number of times it was not chosen. When one of those counts is zero, there will be one observation. In this case, the aggregate data set has 724 observations.

```
proc summary data=coded nway;
   class form set c &_trgind;
   output out=agg(drop=_type_);
   run;

proc phreg data=agg;
   model c*c(2) = &_trgind / ties=breslow;
   freq _freq_;
   strata form set;
   run;
```

PROC SUMMARY ran in three seconds, and PROC PHREG ran in less than one second. The parameter estimates and Chi-Square statistics (not shown) are the same as before. The summary table shows the results of the aggregation, 100 out of 600 alternatives were chosen in each stratum. The log likelihood statistics are different, but that does not matter since the Chi-Square statistics are the same. Page 282 provides more information about this.

Vacation Example with Asymmetry

The PHREG Procedure

Model Information

Data Set	WORK.AGG
Dependent Variable	c
Censoring Variable	c
Censoring Value(s)	2
Frequency Variable	_FREQ_
Ties Handling	BRESLOW

Number of Observations Read	724
Number of Observations Used	724
Sum of Frequencies Read	43200
Sum of Frequencies Used	43200

Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Stratum	Form	Set	Number of Alternatives	Chosen Alternatives	Not Chosen
1	1	1	600	100	500
2	1	2	600	100	500
3	1	3	600	100	500
4	1	4	600	100	500
.					
.					
.					
71	4	71	600	100	500
72	4	72	600	100	500
Total			43200	7200	36000

Brand Choice Example with Aggregate Data

In this next example, subjects were presented with brands of a product at different prices. There were four brands and a constant alternative, eight choice sets, and 100 subjects. This example shows how to handle data that come to you already aggregated. It also illustrates comparing the fits of two competing models, the mother logit model, cross effects, IIA, and techniques for handling large data sets. The choice sets, with the price of each alternative and the number of times it was chosen, are shown next.

Set	Brand 1		Brand 2		Brand 3		Brand 4		Other	
1	$3.99	4	$5.99	29	$3.99	16	$5.99	42	$4.99	9
2	$5.99	12	$5.99	19	$5.99	22	$5.99	33	$4.99	14
3	$5.99	34	$5.99	26	$3.99	8	$3.99	27	$4.99	5
4	$5.99	13	$3.99	37	$5.99	15	$3.99	27	$4.99	8
5	$5.99	49	$3.99	1	$3.99	9	$5.99	37	$4.99	4
6	$3.99	31	$5.99	12	$5.99	6	$3.99	18	$4.99	33
7	$3.99	37	$3.99	10	$5.99	5	$5.99	35	$4.99	13
8	$3.99	16	$3.99	14	$3.99	5	$3.99	51	$4.99	14

The first choice set consists of Brand 1 at $3.99, Brand 2 at $5.99, Brand 3 at $3.99, Brand 4 at $5.99, and Other at $4.99. From this choice set, Brand 1 was chosen 4 times, Brand 2 was chosen 29 times, Brand 3 was chosen 16 times, Brand 4 was chosen 42 times, and Other was chosen 9 times.

Processing the Data

As in the previous examples, we will process the data to create a data set with one stratum for each choice set within each subject and m alternatives per stratum. This example will have 100 people times 5 alternatives times 8 choice sets equals 4000 observations. The first five observations are for the first subject and the first choice set, the next five observations are for the second subject and the first choice set, ..., the next five observations are for the one-hundredth subject and the first choice set, the next five observations are for the first subject and the second choice set, and so on. Subject 1 in the first choice set is almost certainly not the same as subject 1 in subsequent choice sets since we were given aggregate data. However, that is not important. What is important is that we have a subject and choice set variable whose unique combinations identify each choice set within each subject. In previous examples, we specified `strata Subj Set` with PROC PHREG, and our data were sorted by choice set within subject. We can still use the same specification even though our data are now sorted by subject within choice set. This next step reads and prepares the data.

```
%let m = 5;  /* Number of Brands in Each Choice Set */
             /* (including Other)                   */

title 'Brand Choice Example, Multinomial Logit Model';

proc format;
   value brand 1 = 'Brand 1' 2 = 'Brand 2' 3 = 'Brand 3'
               4 = 'Brand 4' 5 = 'Other';
   run;
```

```
data price;
   array p[&m] p1-p&m; /* Prices for the Brands */
   array f[&m] f1-f&m; /* Frequency of Choice   */

   input p1-p&m f1-f&m;
   keep subj set brand price c p1-p&m;
   * Store choice set and subject number to stratify;
   Set = _n_; Subj = 0;
   do i = 1 to &m;           /* Loop over the &m frequencies    */
      do ci = 1 to f[i];     /* Loop frequency of choice times  */
         subj + 1;           /* Subject within choice set       */
         do Brand = 1 to &m; /* Alternatives within choice set  */

            Price = p[brand];

            * Output first choice: c=1, unchosen: c=2;
            c = 2 - (i eq brand); output;
            end;
         end;
      end;

format brand brand.;
datalines;
3.99 5.99 3.99 5.99 4.99   4 29 16 42  9
5.99 5.99 5.99 5.99 4.99  12 19 22 33 14
5.99 5.99 3.99 3.99 4.99  34 26  8 27  5
5.99 3.99 5.99 3.99 4.99  13 37 15 27  8
5.99 3.99 3.99 5.99 4.99  49  1  9 37  4
3.99 5.99 5.99 3.99 4.99  31 12  6 18 33
3.99 3.99 5.99 5.99 4.99  37 10  5 35 13
3.99 3.99 3.99 3.99 4.99  16 14  5 51 14
;

proc print data=price(obs=15);
   var subj set c price brand;
   run;
```

The inner loop `do Brand = 1 to &m` creates all of the observations for the m alternatives within a person/choice set combination. Within a choice set (row of input data), the outer two loops, `do i = 1 to &m` and `do ci = 1 to f[i]` execute the code inside 100 times, the variable `Subj` goes from 1 to 100. In the first choice set, they first create the data for the four subjects that chose Brand 1, then the data for the 29 subjects that chose Brand 2, and so on. Here are the first 15 observations of the data set.

Brand Choice Example, Multinomial Logit Model

Obs	Subj	Set	c	Price	Brand
1	1	1	1	3.99	Brand 1
2	1	1	2	5.99	Brand 2
3	1	1	2	3.99	Brand 3
4	1	1	2	5.99	Brand 4
5	1	1	2	4.99	Other
6	2	1	1	3.99	Brand 1
7	2	1	2	5.99	Brand 2
8	2	1	2	3.99	Brand 3
9	2	1	2	5.99	Brand 4
10	2	1	2	4.99	Other
11	3	1	1	3.99	Brand 1
12	3	1	2	5.99	Brand 2
13	3	1	2	3.99	Brand 3
14	3	1	2	5.99	Brand 4
15	3	1	2	4.99	Other

Note that the data set also contains the variables `p1-p5` which contain the prices of each of the alternatives. These variables, which are used in constructing the cross effects, will be discussed in more detail on page 268.

```
proc print data=price(obs=5); run;
```

Brand Choice Example, Multinomial Logit Model

Obs	p1	p2	p3	p4	p5	Set	Subj	Brand	Price	c
1	3.99	5.99	3.99	5.99	4.99	1	1	Brand 1	3.99	1
2	3.99	5.99	3.99	5.99	4.99	1	1	Brand 2	5.99	2
3	3.99	5.99	3.99	5.99	4.99	1	1	Brand 3	3.99	2
4	3.99	5.99	3.99	5.99	4.99	1	1	Brand 4	5.99	2
5	3.99	5.99	3.99	5.99	4.99	1	1	Other	4.99	2

Simple Price Effects

The data are coded using PROC TRANSREG.

```
proc transreg design data=price nozeroconstant norestoremissing;
   model class(brand / zero=none) identity(price) / lprefix=0;
   output out=coded(drop=_type_ _name_ intercept);
   label price = 'Price';
   id subj set c;
   run;
```

The `design` option specifies that no model is fit; the procedure is just being used to code a design. The `nozeroconstant` option specifies that if the coding creates a constant variable, it should not be zeroed. The `norestoremissing` option specifies that missing values should not be restored when the `out=` data set is created. The `model` statement names the variables to code and provides information about how they should be coded. The specification `class(brand / zero=none)` specifies that the variable `Brand` is a classification variable and requests a binary coding. The `zero=none` option creates binary variables for all categories. The specification `identity(price)` specifies that the variable `Price` is quantitative and hence should directly enter the model without coding. The `lprefix=0` option specifies that when labels are created for the binary variables, zero characters of the original variable name should be used as a prefix. This means that the labels are created only from the level values. An `output` statement names the output data set and drops variables that are not needed. Finally, the `id` statement names the additional variables that we want copied from the input to the output data set.

```
proc phreg data=coded brief;
   title2 'Discrete Choice with Common Price Effect';
   model c*c(2) = &_trgind / ties=breslow;
   strata subj set;
   run;
```

Here are the results. (Recall that we used `%phchoice(on)` on page 143 to customize the output from PROC PHREG.)

Brand Choice Example, Multinomial Logit Model
Discrete Choice with Common Price Effect

The PHREG Procedure

Model Information

Data Set	WORK.CODED
Dependent Variable	c
Censoring Variable	c
Censoring Value(s)	2
Ties Handling	BRESLOW

Number of Observations Read	4000
Number of Observations Used	4000

Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Pattern	Number of Choices	Number of Alternatives	Chosen Alternatives	Not Chosen
1	800	5	1	4

Convergence Status

Convergence criterion (GCONV=1E-8) satisfied.

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	2575.101	2425.214
AIC	2575.101	2435.214
SBC	2575.101	2458.637

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	149.8868	5	<.0001
Score	153.2328	5	<.0001
Wald	142.9002	5	<.0001

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Brand 1	1	0.66727	0.12305	29.4065	<.0001
Brand 2	1	0.38503	0.12962	8.8235	0.0030
Brand 3	1	-0.15955	0.14725	1.1740	0.2786
Brand 4	1	0.98964	0.11720	71.2993	<.0001
Other	0	0	.	.	.
Price	1	0.14966	0.04406	11.5379	0.0007

Alternative-Specific Price Effects

In the next step, the data are coded for fitting a multinomial logit model with brand by price effects.

```
proc transreg design data=price nozeroconstant norestoremissing;
   model class(brand / zero=none separators='' ' ') |
         identity(price) / lprefix=0;
   output out=coded(drop=_type_ _name_ intercept);
   label price = 'Price';
   id subj set c;
   run;
```

The PROC TRANSREG `model` statement has a vertical bar, "`|`", between the `class` specification and the `identity` specification. Since the `zero=none` option is specified with `class`, the vertical bar creates two sets of variables: five indicator variables for the brand effects and five more variables for the brand by price interactions. The `separators=` option allows you to specify two label component separators as quoted strings. The specification `separators='' ' '` (`separators=` quote quote space quote space quote) specifies a null string (quote quote) and a blank (quote space quote). The `separators='' ' '` option in the `class` specification specifies the separators that are used to construct the labels for the main effect and interaction terms, respectively. By default, the alternative-specific price effects—the

brand by price interactions−would have labels like 'Brand 1 * Price' since the default second value for separators= is ' * ' (a quoted space asterisk space). Specifying ' ' (a quoted space) as the second value creates labels of the form 'Brand 1 Price'. Since lprefix=0, the main-effects separator, which is the first separators= value, '' (quote quote), is ignored. Zero name or input variable label characters are used to construct the label. The label is simply the formatted value of the class variable. The next steps print the first two coded choice sets and perform the analysis.

```
proc print data=coded(obs=10) label;
   title2 'Discrete Choice with Brand by Price Effects';
   var subj set c brand price &_trgind;
   run;
proc phreg data=coded brief;
   model c*c(2) = &_trgind / ties=breslow;
   strata subj set;
   run;

title2;
```

Brand Choice Example, Multinomial Logit Model
Discrete Choice with Brand by Price Effects

Obs	Subj	Set	c	Brand	Price	Brand 1	Brand 2	Brand 3	Brand 4
1	1	1	1	Brand 1	3.99	1	0	0	0
2	1	1	2	Brand 2	5.99	0	1	0	0
3	1	1	2	Brand 3	3.99	0	0	1	0
4	1	1	2	Brand 4	5.99	0	0	0	1
5	1	1	2	Other	4.99	0	0	0	0
6	2	1	1	Brand 1	3.99	1	0	0	0
7	2	1	2	Brand 2	5.99	0	1	0	0
8	2	1	2	Brand 3	3.99	0	0	1	0
9	2	1	2	Brand 4	5.99	0	0	0	1
10	2	1	2	Other	4.99	0	0	0	0

Obs	Other	Brand 1 Price	Brand 2 Price	Brand 3 Price	Brand 4 Price	Other Price
1	0	3.99	0.00	0.00	0.00	0.00
2	0	0.00	5.99	0.00	0.00	0.00
3	0	0.00	0.00	3.99	0.00	0.00
4	0	0.00	0.00	0.00	5.99	0.00
5	1	0.00	0.00	0.00	0.00	4.99
6	0	3.99	0.00	0.00	0.00	0.00
7	0	0.00	5.99	0.00	0.00	0.00
8	0	0.00	0.00	3.99	0.00	0.00
9	0	0.00	0.00	0.00	5.99	0.00
10	1	0.00	0.00	0.00	0.00	4.99

Brand Choice Example, Multinomial Logit Model
Discrete Choice with Brand by Price Effects

The PHREG Procedure

Model Information

Data Set	WORK.CODED
Dependent Variable	c
Censoring Variable	c
Censoring Value(s)	2
Ties Handling	BRESLOW

Number of Observations Read	4000
Number of Observations Used	4000

Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Pattern	Number of Choices	Number of Alternatives	Chosen Alternatives	Not Chosen
1	800	5	1	4

Convergence Status

Convergence criterion (GCONV=1E-8) satisfied.

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	2575.101	2424.812
AIC	2575.101	2440.812
SBC	2575.101	2478.288

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	150.2891	8	<.0001
Score	154.2563	8	<.0001
Wald	143.1425	8	<.0001

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Brand 1	1	-0.00972	0.43555	0.0005	0.9822
Brand 2	1	-0.62230	0.48866	1.6217	0.2028
Brand 3	1	-0.81250	0.60318	1.8145	0.1780
Brand 4	1	0.31778	0.39549	0.6456	0.4217
Other	0	0	.	.	.
Brand 1 Price	1	0.13587	0.08259	2.7063	0.1000
Brand 2 Price	1	0.20074	0.09210	4.7512	0.0293
Brand 3 Price	1	0.13126	0.11487	1.3057	0.2532
Brand 4 Price	1	0.13478	0.07504	3.2255	0.0725
Other Price	0	0	.	.	.

The likelihood for this model is essentially the same as for the simpler, common-price-slope model fit previously, $-2\log(\mathcal{L}_C) = 2425.214$ compared to 2424.812. You can test the null hypothesis that the two models are not significantly different by comparing their likelihoods. The difference between two $-2\log(\mathcal{L}_C)$'s (the number reported under 'With Covariates' in the output) has a chi-square distribution. We can get the *df* for the test by subtracting the two *df* for the two likelihoods. The difference $2425.214 - 2424.812 = 0.402$ is distributed χ^2 with $8 - 5 = 3$ *df* and is not statistically significant.

Mother Logit Model

This next step fits the so-called "mother logit" model. This step creates the full design matrix, including the brand, price, and cross effects. A cross effect represents the effect of one alternative on the utility of another alternative. First, let's look at the input data set for the first choice set.

```
proc print data=price(obs=5) label;
   run;
```

Brand Choice Example, Multinomial Logit Model

Obs	p1	p2	p3	p4	p5	Set	Subj	Brand	Price	c
1	3.99	5.99	3.99	5.99	4.99	1	1	Brand 1	3.99	1
2	3.99	5.99	3.99	5.99	4.99	1	1	Brand 2	5.99	2
3	3.99	5.99	3.99	5.99	4.99	1	1	Brand 3	3.99	2
4	3.99	5.99	3.99	5.99	4.99	1	1	Brand 4	5.99	2
5	3.99	5.99	3.99	5.99	4.99	1	1	Other	4.99	2

The input consists of `Set`, `Subj`, `Brand`, `Price`, and a choice time variable `c`. In addition, it contains five variables `p1` through `p5`. The first observation of the `Price` variable shows us that the first alternative costs \$3.99; `p1` contains the cost of alternative 1, \$3.99, which is the same for all alternatives. It does not matter which alternative you are looking at, `p1` shows that alternative 1 costs \$3.99. Similarly, the

second observation of the `Price` variable shows us that the second alternative costs \$5.99; `p2` contains the cost of alternative 2, \$5.99, which is the same for all alternatives. There is one price variable, `p1` through `p5`, for each of the five alternatives.

In all of the previous examples, we have used models that were coded so that the utility of an alternative only depended on the attributes of that alternative. For example, the utility of Brand 1 would only depend on the Brand 1 name and its price. In contrast, `p1-p5` contain information about each of the *other* alternatives' attributes. We will construct cross effects using the interaction of `p1-p5` and the `Brand` variable. In a model with cross effects, the utility for an alternative depends on both that alternative's attributes *and* the other alternatives' attributes. The IIA (independence from irrelevant alternatives) property states that utility only depends on an alternative's own attributes. Cross effects add other alternative's attributes to the model, so they can be used to test for violations of IIA. (See pages 275, 283, 476, and 480 for other discussions of IIA.) Here is the PROC TRANSREG code for the cross-effects model.

```
proc transreg design data=price nozeroconstant norestoremissing;
   model class(brand / zero=none separators='' ' ') | identity(price)
         identity(p1-p&m) *
            class(brand / zero=none lprefix=0 separators='' ' on ') /
         lprefix=0;
   output out=coded(drop=_type_ _name_ intercept);
   label price = 'Price'
         p1 = 'Brand 1' p2 = 'Brand 2' p3 = 'Brand 3'
         p4 = 'Brand 4' p5 = 'Other';
   id subj set c;
   run;
```

The `class(brand / ...) | identity(price)` specification in the `model` statement is the same as the previous analysis. The additional terms, `identity(p1-p&m) * class(brand / ...)` create the cross effects. The second value of the `separators=` option, `' on'` is used to create labels like `'Brand 1 on Brand 2'` instead of the default `'Brand 1 * Brand 2'`. It is important to note that you must specify the cross effect by specifying `identity` with the price factors, followed by the asterisk, followed by `class` and the brand effect, *in that order*. The order of the specification determines the order in which brand names are added to the labels. Do not specify the brand variable first; doing so will create incorrect labels.

With m alternatives, there are $m \times m$ cross effects, but as we will see, many of them are zero. The first coded choice set is printed with the following PROC PRINT steps. Multiple steps are used to facilitate explaining the coding.

```
title2 'Discrete Choice with Cross Effects, Mother Logit';
proc format; value zer 0 = '  0' 1 = '  1'; run;
proc print data=coded(obs=5) label; var subj set c brand price; run;
proc print data=coded(obs=5) label; var Brand:;
   format brand: zer5.2 brand brand.; run;
proc print data=coded(obs=5) label; var p1B:; format p: zer5.2; id brand; run;
proc print data=coded(obs=5) label; var p2B:; format p: zer5.2; id brand; run;
proc print data=coded(obs=5) label; var p3B:; format p: zer5.2; id brand; run;
proc print data=coded(obs=5) label; var p4B:; format p: zer5.2; id brand; run;
proc print data=coded(obs=5) label; var p5B:; format p: zer5.2; id brand; run;
```

The coded data set contains the strata variable `Subj` and `Set`, choice time variable `c`, and `Brand` and `Price`. `Brand` and `Price` were used to create the coded independent variables but they are not used

in the analysis with PROC PHREG.

Brand Choice Example, Multinomial Logit Model
Discrete Choice with Cross Effects, Mother Logit

Obs	Subj	Set	c	Brand	Price
1	1	1	1	Brand 1	3.99
2	1	1	2	Brand 2	5.99
3	1	1	2	Brand 3	3.99
4	1	1	2	Brand 4	5.99
5	1	1	2	Other	4.99

The effects 'Brand 1' through 'Other' in the next output are the binary brand effect variables. They indicate the brand for each alternative. The effects 'Brand 1 Price' through 'Other Price' are alternative-specific price effects. They indicate the price for each alternative. All ten of these variables are independent variables in the analysis, and their names are part of the &_trgind macro variable list, as are all of the cross effects that are described next.

Brand Choice Example, Multinomial Logit Model
Discrete Choice with Cross Effects, Mother Logit

Obs	Brand 1	Brand 2	Brand 3	Brand 4	Other	Brand 1 Price	Brand 2 Price	Brand 3 Price	Brand 4 Price	Other Price	Brand
1	1	0	0	0	0	3.99	0	0	0	0	Brand 1
2	0	1	0	0	0	0	5.99	0	0	0	Brand 2
3	0	0	1	0	0	0	0	3.99	0	0	Brand 3
4	0	0	0	1	0	0	0	0	5.99	0	Brand 4
5	0	0	0	0	1	0	0	0	0	4.99	Other

The effects 'Brand 1 on Brand 1' through 'Brand 1 on Other' in the next output are the first five cross effects.

Brand Choice Example, Multinomial Logit Model
Discrete Choice with Cross Effects, Mother Logit

Brand	Brand 1 on Brand 1	Brand 1 on Brand 2	Brand 1 on Brand 3	Brand 1 on Brand 4	Brand 1 on Other
Brand 1	3.99	0	0	0	0
Brand 2	0	3.99	0	0	0
Brand 3	0	0	3.99	0	0
Brand 4	0	0	0	3.99	0
Other	0	0	0	0	3.99

They represent the effect of Brand 1 at its price on the utility of each alternative. The label `'Brand` n `on Brand` m`'` is read as 'the effect of Brand n at its price on the utility of Brand m.' For the first choice set, these first five cross effects consist entirely of zeros and \$3.99's, where \$3.99 is the price of Brand 1 in this choice set. The nonzero value is constant across all of the alternatives in each choice set since Brand 1 has only one price in each choice set. Notice the `'Brand 1 on Brand 1'` term, which is the effect of Brand 1 at its price on the utility of Brand 1. Also notice the `'Brand 1 Price'` effect, which is shown in the previous output. The description "the effect of Brand 1 at its price on the utility of Brand 1" is just a convoluted way of describing the Brand 1 price effect. The `'Brand 1 on Brand 1'` cross effect is the same as the Brand 1 price effect, hence when we do the analysis, we will see that the coefficient for the `'Brand 1 on Brand 1'` cross effect is zero.

The effects `'Brand 2 on Brand 1'` through `'Brand 2 on Other'` in the next output are the next five cross effects.

Brand Choice Example, Multinomial Logit Model
Discrete Choice with Cross Effects, Mother Logit

Brand	Brand 2 on Brand 1	Brand 2 on Brand 2	Brand 2 on Brand 3	Brand 2 on Brand 4	Brand 2 on Other
Brand 1	5.99	0	0	0	0
Brand 2	0	5.99	0	0	0
Brand 3	0	0	5.99	0	0
Brand 4	0	0	0	5.99	0
Other	0	0	0	0	5.99

They represent the effect of Brand 2 at its price on the utility of each alternative. For the first choice set, these five cross effects consist entirely of zeros and \$5.99's, where \$5.99 is the price of Brand 2 in this choice set. The nonzero value is constant across all of the alternatives in each choice set since Brand 2 has only one price in each choice set. Notice the `'Brand 2 on Brand 2'` term, which is the effect of Brand 2 at its price on the utility of Brand 2. The description "the effect of Brand 2 at its price on the utility of Brand 2" is just a convoluted way of describing the Brand 2 price effect. The `'Brand 2 on Brand 2'` cross effect is the same as the Brand 2 price effect, hence when we do the analysis, we will see that the coefficient for the `'Brand 2 on Brand 2'` cross effect is zero.

The effects `'Brand 3 on Brand 1'` through `'Brand 3 on Other'` in the next output are the next five cross effects.

Brand Choice Example, Multinomial Logit Model
Discrete Choice with Cross Effects, Mother Logit

Brand	Brand 3 on Brand 1	Brand 3 on Brand 2	Brand 3 on Brand 3	Brand 3 on Brand 4	Brand 3 on Other
Brand 1	3.99	0	0	0	0
Brand 2	0	3.99	0	0	0
Brand 3	0	0	3.99	0	0
Brand 4	0	0	0	3.99	0
Other	0	0	0	0	3.99

They represent the effect of Brand 3 at its price on the utility of each alternative. For the first choice set, these five cross effects consist entirely of zeros and \$3.99's, where \$3.99 is the price of Brand 3 in this choice set. Notice that the `'Brand 3 on Brand 3'` term is the same as the Brand 3 price effect, hence when we do the analysis, we will see that the coefficient for the `'Brand 3 on Brand 3'` cross effect is zero.

Here are the remaining cross effects. They follow the same pattern that was described for the previous cross effects.

Brand Choice Example, Multinomial Logit Model
Discrete Choice with Cross Effects, Mother Logit

Brand	Brand 4 on Brand 1	Brand 4 on Brand 2	Brand 4 on Brand 3	Brand 4 on Brand 4	Brand 4 on Other
Brand 1	5.99	0	0	0	0
Brand 2	0	5.99	0	0	0
Brand 3	0	0	5.99	0	0
Brand 4	0	0	0	5.99	0
Other	0	0	0	0	5.99

Brand Choice Example, Multinomial Logit Model
Discrete Choice with Cross Effects, Mother Logit

Brand	Other on Brand 1	Other on Brand 2	Other on Brand 3	Other on Brand 4	Other on Other
Brand 1	4.99	0	0	0	0
Brand 2	0	4.99	0	0	0
Brand 3	0	0	4.99	0	0
Brand 4	0	0	0	4.99	0
Other	0	0	0	0	4.99

We have been describing variables by their labels. While it is not necessary to look at it, the `&_trgind` macro variable name list that PROC TRANSREG creates for this problem is as follows:

```
%put &_trgind;
```

```
BrandBrand_1 BrandBrand_2 BrandBrand_3 BrandBrand_4 BrandOther
BrandBrand_1Price BrandBrand_2Price BrandBrand_3Price BrandBrand_4Price
BrandOtherPrice p1BrandBrand_1 p1BrandBrand_2 p1BrandBrand_3 p1BrandBrand_4
p1BrandOther p2BrandBrand_1 p2BrandBrand_2 p2BrandBrand_3 p2BrandBrand_4
p2BrandOther p3BrandBrand_1 p3BrandBrand_2 p3BrandBrand_3 p3BrandBrand_4
p3BrandOther p4BrandBrand_1 p4BrandBrand_2 p4BrandBrand_3 p4BrandBrand_4
p4BrandOther p5BrandBrand_1 p5BrandBrand_2 p5BrandBrand_3 p5BrandBrand_4
p5BrandOther
```

The analysis proceeds in exactly the same manner as before.

```
proc phreg data=coded brief;
   model c*c(2) = &_trgind / ties=breslow;
   strata subj set;
   run;
```

Brand Choice Example, Multinomial Logit Model
Discrete Choice with Cross Effects, Mother Logit

The PHREG Procedure

Model Information

Data Set	WORK.CODED
Dependent Variable	c
Censoring Variable	c
Censoring Value(s)	2
Ties Handling	BRESLOW

Number of Observations Read	4000
Number of Observations Used	4000

Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Pattern	Number of Choices	Number of Alternatives	Chosen Alternatives	Not Chosen
1	800	5	1	4

Convergence Status

Convergence criterion (GCONV=1E-8) satisfied.

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	2575.101	2349.325
AIC	2575.101	2389.325
SBC	2575.101	2483.018

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	225.7752	20	<.0001
Score	218.4500	20	<.0001
Wald	190.0257	20	<.0001

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Brand 1	1	1.24963	1.31259	0.9064	0.3411
Brand 2	1	-0.16269	1.38579	0.0138	0.9065
Brand 3	1	-3.90179	1.56511	6.2150	0.0127
Brand 4	1	2.49435	1.25537	3.9480	0.0469
Other	0	0	.	.	.
Brand 1 Price	1	0.51056	0.13178	15.0096	0.0001
Brand 2 Price	1	-0.04920	0.13411	0.1346	0.7137
Brand 3 Price	1	-0.27594	0.15517	3.1623	0.0754
Brand 4 Price	1	0.28951	0.12192	5.6389	0.0176
Other Price	0	0	.	.	.
Brand 1 on Brand 1	0	0	.	.	.
Brand 1 on Brand 2	1	0.51651	0.13675	14.2653	0.0002
Brand 1 on Brand 3	1	0.66122	0.15655	17.8397	<.0001
Brand 1 on Brand 4	1	0.32806	0.12664	6.7105	0.0096
Brand 1 on Other	0	0	.	.	.
Brand 2 on Brand 1	1	-0.39876	0.12832	9.6561	0.0019
Brand 2 on Brand 2	0	0	.	.	.
Brand 2 on Brand 3	1	-0.01755	0.15349	0.0131	0.9090
Brand 2 on Brand 4	1	-0.33802	0.12220	7.6512	0.0057
Brand 2 on Other	0	0	.	.	.
Brand 3 on Brand 1	1	-0.43868	0.13119	11.1823	0.0008
Brand 3 on Brand 2	1	-0.31541	0.13655	5.3356	0.0209
Brand 3 on Brand 3	0	0	.	.	.
Brand 3 on Brand 4	1	-0.54854	0.12528	19.1723	<.0001
Brand 3 on Other	0	0	.	.	.

Brand 4 on Brand 1	1	0.24398	0.12781	3.6443	0.0563
Brand 4 on Brand 2	1	-0.01214	0.13416	0.0082	0.9279
Brand 4 on Brand 3	1	0.40500	0.15285	7.0211	0.0081
Brand 4 on Brand 4	0	0	.	.	.
Brand 4 on Other	0	0	.	.	.
Other on Brand 1	0	0	.	.	.
Other on Brand 2	0	0	.	.	.
Other on Brand 3	0	0	.	.	.
Other on Brand 4	0	0	.	.	.
Other on Other	0	0	.	.	.

The results consist of:

- four nonzero brand effects and a zero for the constant alternative
- four nonzero alternative-specific price effects and a zero for the constant alternative
- $5 \times 5 = 25$ cross effects, the number of alternatives squared, but only $(5-1) \times (5-2) = 12$ of them are nonzero (four brands not counting Other affecting each of the remaining three brands).
  - There are three cross effects for the effect of Brand 1 on Brands 2, 3, and 4.
  - There are three cross effects for the effect of Brand 2 on Brands 1, 3, and 4.
  - There are three cross effects for the effect of Brand 3 on Brands 1, 2, and 4.
  - There are three cross effects for the effect of Brand 4 on Brands 1, 2, and 3.

All coefficients for the constant (other) alternative are zero as are the cross effects of a brand on itself.

The mother logit model is used to test for violations of IIA (independence from irrelevant alternatives). IIA means the odds of choosing alternative c_i over c_j do not depend on the other alternatives in the choice set. Ideally, this more general model will not significantly explain more variation in choice than the restricted models. Also, if IIA is satisfied, few if any of the cross-effect terms should be significantly different from zero. (See pages 269, 283, 476, and 480 for other discussions of IIA.) In this case, it appears that IIA is *not* satisfied (the data are artificial), so the more general mother logit model is needed. The chi-square statistic is $2424.812 - 2349.325 = 75.487$ with $20 - 8 = 12$ *df* ($p < 0.0001$).

You could eliminate some of the zero parameters by changing `zero=none` to `zero='Other'` and eliminating `p5` (`p&m`) from the model.

```
proc transreg design data=price nozeroconstant norestoremissing;
   model class(brand / zero='Other' separators='' ' ') | identity(price)
         identity(p1-p4) * class(brand / zero='Other' separators='' ' on ') /
         lprefix=0;
   output out=coded(drop=_type_ _name_ intercept);
   label price = 'Price'
         p1 = 'Brand 1' p2 = 'Brand 2' p3 = 'Brand 3'
         p4 = 'Brand 4';
   id subj set c;
   run;
```

You could also eliminate the brand by price effects and instead capture brand by price effects as the cross effect of a variable on itself.

```
proc transreg design data=price nozeroconstant norestoremissing;
   model class(brand / zero='Other' separators='' ' ')
         identity(p1-p4) * class(brand / zero='Other' separators='' ' on ') /
         lprefix=0;
   output out=coded(drop=_type_ _name_ intercept);
   label price = 'Price'
         p1 = 'Brand 1' p2 = 'Brand 2' p3 = 'Brand 3'
         p4 = 'Brand 4';
   id subj set c;
   run;
```

In both cases, the analysis (not shown) would be run in the usual manner. Except for the elimination of zero terms, and in the second case, the change to capture the price effects in the cross effects, the results are identical.

Aggregating the Data

In all examples so far (except the last part of the last vacation example), the data set has been created for analysis with one stratum for each choice set and subject combination. Such data sets can be large. The data can also be arrayed with a frequency variable and each choice set forming a separate stratum. This example illustrates how.

```
title 'Brand Choice Example, Multinomial Logit Model';
title2 'Aggregate Data';

%let m = 5;  /* Number of Brands in Each Choice Set */
             /* (including Other)                   */

proc format;
   value brand 1 = 'Brand 1' 2 = 'Brand 2' 3 = 'Brand 3'
               4 = 'Brand 4' 5 = 'Other';
   run;

data price2;
   array p[&m] p1-p&m; /* Prices for the Brands */
   array f[&m] f1-f&m; /* Frequency of Choice   */
   input p1-p&m f1-f&m;
   keep set price brand freq c p1-p&m;

   * Store choice set number to stratify;
   Set = _n_;
```

```
      do Brand = 1 to &m;

         Price = p[brand];

         * Output first choice: c=1, unchosen: c=2;
         Freq = f[brand]; c = 1; output;

         * Output number of times brand was not chosen.;
         freq = sum(of f1-f&m) - freq; c = 2; output;

         end;

   format brand brand.;
   datalines;
   3.99 5.99 3.99 5.99 4.99   4 29 16 42  9
   5.99 5.99 5.99 5.99 4.99  12 19 22 33 14
   5.99 5.99 3.99 3.99 4.99  34 26  8 27  5
   5.99 3.99 5.99 3.99 4.99  13 37 15 27  8
   5.99 3.99 3.99 5.99 4.99  49  1  9 37  4
   3.99 5.99 5.99 3.99 4.99  31 12  6 18 33
   3.99 3.99 5.99 5.99 4.99  37 10  5 35 13
   3.99 3.99 3.99 3.99 4.99  16 14  5 51 14
   ;
   proc print data=price2(obs=10);
      var set c freq price brand;
      run;
```

Brand Choice Example, Multinomial Logit Model
Aggregate Data

Obs	Set	c	Freq	Price	Brand
1	1	1	4	3.99	Brand 1
2	1	2	96	3.99	Brand 1
3	1	1	29	5.99	Brand 2
4	1	2	71	5.99	Brand 2
5	1	1	16	3.99	Brand 3
6	1	2	84	3.99	Brand 3
7	1	1	42	5.99	Brand 4
8	1	2	58	5.99	Brand 4
9	1	1	9	4.99	Other
10	1	2	91	4.99	Other

This data set has 5 brands times 2 observations times 8 choice sets for a total of 80 observations, compared to $100 \times 5 \times 8 = 4000$ using the standard method. Two observations are created for each alternative within each choice set. The first contains the number of people who chose the alternative, and the second contains the number of people who did not choose the alternative.

To analyze the data, specify `strata Set` and `freq Freq`.

```
proc transreg design data=price2 nozeroconstant norestoremissing;
   model class(brand / zero=none) identity(price) / lprefix=0;
   output out=coded(drop=_type_ _name_ intercept);
   label price = 'Price';
   id freq set c;
   run;

proc phreg data=coded;
   title2 'Discrete Choice with Common Price Effect, Aggregate Data';
   model c*c(2) = &_trgind / ties=breslow;
   strata set;
   freq freq;
   run;
```

These steps produced the following results.

Brand Choice Example, Multinomial Logit Model
Discrete Choice with Common Price Effect, Aggregate Data

The PHREG Procedure

Model Information

Data Set	WORK.CODED
Dependent Variable	c
Censoring Variable	c
Censoring Value(s)	2
Frequency Variable	Freq
Ties Handling	BRESLOW

Number of Observations Read	80
Number of Observations Used	80
Sum of Frequencies Read	4000
Sum of Frequencies Used	4000

Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Stratum	Set	Number of Alternatives	Chosen Alternatives	Not Chosen
1	1	500	100	400
2	2	500	100	400
3	3	500	100	400
4	4	500	100	400

5	5	500	100	400
6	6	500	100	400
7	7	500	100	400
8	8	500	100	400
Total		4000	800	3200

Convergence Status

Convergence criterion (GCONV=1E-8) satisfied.

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	9943.373	9793.486
AIC	9943.373	9803.486
SBC	9943.373	9826.909

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	149.8868	5	<.0001
Score	153.2328	5	<.0001
Wald	142.9002	5	<.0001

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Brand 1	1	0.66727	0.12305	29.4065	<.0001
Brand 2	1	0.38503	0.12962	8.8235	0.0030
Brand 3	1	-0.15955	0.14725	1.1740	0.2786
Brand 4	1	0.98964	0.11720	71.2993	<.0001
Other	0	0	.	.	.
Price	1	0.14966	0.04406	11.5379	0.0007

The summary table is small with eight rows, one row per choice set. Each row represents 100 chosen alternatives and 400 unchosen. The 'Analysis of Maximum Likelihood Estimates' table exactly matches the one produced by the standard analysis. The -2 LOG L statistics are different than before: 9793.486 now compared to 2425.214 previously. This is because the data are arrayed in this example so that the partial likelihood of the proportional hazards model fit by PROC PHREG with the `ties=breslow` option is now proportional to−not identical to−the likelihood for the choice model. However, the Model Chi-Square statistics, *df*, and *p*-values are the same as before. The two corresponding pairs of -2 LOG L's differ by a constant $9943.373 - 2575.101 = 9793.486 - 2425.214 = 7368.272 = 2 \times 800 \times \log(100)$. Since the χ^2 is the -2 LOG L without covariates minus -2 LOG L with covariates, the constants cancel and the χ^2 test is correct for both methods.

The technique of aggregating the data and using a frequency variable can be used for other models as well, for example with brand by price effects.

```
proc transreg design data=price2 nozeroconstant norestoremissing;
   model class(brand / zero=none separators='' ' ') |
         identity(price) / lprefix=0;
   output out=coded(drop=_type_ _name_ intercept);
   label price = 'Price';
   id freq set c;
   run;

proc phreg data=coded;
   title2 'Discrete Choice with Brand by Price Effects, Aggregate Data';
   model c*c(2) = &_trgind / ties=breslow;
   strata set;
   freq freq;
   run;
```

This step produced the following results. The only thing that changes from the analysis with one stratum for each subject and choice set combination is the likelihood.

Brand Choice Example, Multinomial Logit Model
Discrete Choice with Brand by Price Effects, Aggregate Data

The PHREG Procedure

Model Information

Data Set	WORK.CODED
Dependent Variable	c
Censoring Variable	c
Censoring Value(s)	2
Frequency Variable	Freq
Ties Handling	BRESLOW

Number of Observations Read	80
Number of Observations Used	80
Sum of Frequencies Read	4000
Sum of Frequencies Used	4000

Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Stratum	Set	Number of Alternatives	Chosen Alternatives	Not Chosen
1	1	500	100	400
2	2	500	100	400
3	3	500	100	400
4	4	500	100	400

5	5	500	100	400
6	6	500	100	400
7	7	500	100	400
8	8	500	100	400
Total		4000	800	3200

Convergence Status

Convergence criterion (GCONV=1E-8) satisfied.

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	9943.373	9793.084
AIC	9943.373	9809.084
SBC	9943.373	9846.561

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	150.2891	8	<.0001
Score	154.2562	8	<.0001
Wald	143.1425	8	<.0001

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Brand 1	1	-0.00972	0.43555	0.0005	0.9822
Brand 2	1	-0.62230	0.48866	1.6217	0.2028
Brand 3	1	-0.81250	0.60318	1.8145	0.1780
Brand 4	1	0.31778	0.39549	0.6456	0.4217
Other	0	0	.	.	.
Brand 1 Price	1	0.13587	0.08259	2.7063	0.1000
Brand 2 Price	1	0.20074	0.09210	4.7512	0.0293
Brand 3 Price	1	0.13126	0.11487	1.3057	0.2532
Brand 4 Price	1	0.13478	0.07504	3.2255	0.0725
Other Price	0	0	.	.	.

Previously, with one stratum per choice set within subject, we compared these models as follows: "The difference $2425.214 - 2424.812 = 0.402$ is distributed χ^2 with $8 - 5 = 3$ *df* and is not statistically significant." The difference between two $-2\log(\mathcal{L}_C)$'s equals the difference between two $-2\log(\mathcal{L}_B)$'s, since the constant terms $(800 \times \log(100))$ cancel, $9793.486 - 9793.084 = 2425.214 - 2424.812 = 0.402$.

Choice and Breslow Likelihood Comparison

This section explains why the -2 LOG L values differ by a constant with aggregate data versus individual data. It may be skipped by all but the most dedicated readers.

Consider the choice model with a common price slope. Let x_0 represent the price of the brand. Let x_1, x_2, x_3, and x_4 be indicator variables representing the choice of brands. Let $\mathbf{x} = (x_0\ x_1\ x_2\ x_3\ x_4)$ be the vector of alternative attributes. (A sixth element for 'Other' is omitted, since its parameter is always zero given the other brands.)

Consider the first choice set. There are five distinct vectors of alternative attributes
$\mathbf{x}_1 = (3.99\ 1\ 0\ 0\ 0)$ $\mathbf{x}_2 = (5.99\ 0\ 1\ 0\ 0)$ $\mathbf{x}_3 = (3.99\ 0\ 0\ 1\ 0)$ $\mathbf{x}_4 = (5.99\ 0\ 0\ 0\ 1)$
$\mathbf{x}_5 = (4.99\ 0\ 0\ 0\ 0)$

The vector $\mathbf{x}_2$, for example, represents choice of Brand 2, and $\mathbf{x}_5$ represents the choice of Other. One hundred individuals were asked to choose one of the $m = 5$ brands from each of the eight sets. Let f_1, f_2, f_3, f_4, and f_5 be the number of times each brand was chosen. For the first choice set, $f_1 = 4$, $f_2 = 29$, $f_3 = 16$, $f_4 = 42$, and $f_5 = 9$. Let N be the total frequency for each choice set, $N = \sum_{j=1}^{5} f_j = 100$. The likelihood L_1^C for the first choice set data is

$$L_1^C = \frac{\exp\left(\left(\sum_{j=1}^{5} f_j \mathbf{x}_j\right)\boldsymbol{\beta}\right)}{\left[\sum_{j=1}^{5} \exp(\mathbf{x}_j\boldsymbol{\beta})\right]^N}$$

The joint likelihood for all eight choice sets is the product of the likelihoods

$$\mathcal{L}_C = \prod_{k=1}^{8} L_k^C$$

The Breslow likelihood for this example, L_k^B, for the *kth* choice set, is the same as the likelihood for the choice model, except for a multiplicative constant.

$$L_k^C = N^N L_k^B = 100^{100} L_k^B$$

Therefore, the Breslow likelihood for all eight choice sets is

$$\mathcal{L}_B = \prod_{k=1}^{8} L_k^B = N^{-8N}\mathcal{L}_C = 100^{-800}\mathcal{L}_C$$

The two likelihoods are not exactly the same, because each choice set is designated as a separate stratum, instead of each choice set within each subject.

The log likelihood for the choice model is

$$\begin{aligned}
\log(\mathcal{L}_C) &= 800 \times \log(100) + \log(\mathcal{L}_B), \\
\log(\mathcal{L}_C) &= 800 \times \log(100) + (-0.5) \times 9793.486, \\
\log(\mathcal{L}_C) &= -1212.607
\end{aligned}$$

and $-2\log(\mathcal{L}_C) = 2425.214$, which matches the earlier output. However, it is usually not necessary to obtain this value.

Food Product Example with Asymmetry and Availability Cross Effects

This example is based on the the choice example from page 111. This example discusses the multinomial logit model, number of parameters, choosing the number of choice sets, designing the choice experiment, long design searches, examining the design, examining the subdesigns, examining the aliasing structure, blocking the design, testing the design before data collection, generating artificial data, processing the data, coding, cross effects, availability, multinomial logit model results, modeling subject attributes, results, and interpretation.

Consider the problem of using a discrete choice model to study the effect of introducing a retail food product. This may be useful, for instance, to refine a marketing plan or to optimize a product prior to test market. A typical brand team will have several concerns such as knowing the potential market share for the product, examining the source of volume, and providing guidance for pricing and promotions. The brand team may also want to know what brand attributes have competitive clout and want to identify competitive attributes to which they are vulnerable.

To develop this further, assume our client wishes to introduce a line extension in the category of frozen entrées. The client has one nationally branded competitor, a regional competitor in each of three regions, and a profusion of private label products at the grocery chain level. The product may come in two different forms: stove-top or microwaveable. The client believes that the private labels are very likely to mimic this line extension and to sell it at a lower price. The client suspects that this strategy on the part of private labels may work for the stove-top version but not for the microwaveable, where they have the edge on perceived quality. They also want to test the effect of a shelf talker that will draw attention to their product.

The Multinomial Logit Model

This problem can be set up as a discrete choice model in which a respondent's choice among brands, given choice set C_a of available brands, will correspond to the brand with the highest utility. For each brand i, the utility U_i is the sum of a systematic component V_i and a random component e_i. The probability of choosing brand i from choice set C_a is therefore:

$$P(i|C_a) = P(U_i > \max(U_j)) = P(V_i + e_i > \max(V_j + e_j)) \quad \forall \quad (j \neq i) \in C_a$$

Assuming that the e_i follow an extreme value type I distribution, the conditional probabilities $P(i|C_a)$ can be found using the multinomial logit (MNL) formulation of McFadden (1974).

$$P(i|C_a) = \exp(V_i) / \sum_{j \in C_a} \exp(V_j)$$

One of the consequences of the MNL formulation is the property of independence from irrelevant alternatives (IIA). Under the assumption of IIA, all cross effects are assumed to be equal, so that if a brand gains in utility, it draws share from all other brands in proportion to their current shares. Departures from IIA exist when certain subsets of brands are in more direct competition and tend to draw a disproportionate amount of share from each other than from other members in the category.

IIA is frequently described using a transportation example. Say you have three alternatives for getting to work: bicycle, car, or a blue bus. If a fourth alternative became available, a red bus, then according to IIA the red bus should draw riders from the other alternatives in proportion to their current usage. However, in this case, IIA would be violated, and instead the red bus would draw more riders from the blue bus than from car drivers and bicycle riders.

The mother logit formulation of McFadden (1974) can be used to capture departures from IIA. In a mother logit model, the utility for brand i is a function of both the attributes of brand i and the attributes of other brands. The effect of one brand's attributes on another is termed a cross effect. In the case of designs in which only subsets C_a of the full shelf set C appear, the effect of the presence/absence of one brand on the utility of another is termed an *availability cross effect*. (See pages 269, 275, 476, and 480 for other discussions of IIA.)

Set Up

In the frozen entrée example, there are five alternatives: the client's brand, the client's line extension, a national branded competitor, a regional brand and a private label brand. Several regional and private labels can be tested in each market, then aggregated for the final model. Note that the line extension is treated as a separate alternative rather than as a level of the client brand. This enables us to model the source of volume for the new entry and to quantify any cannibalization that occurs. Each brand is shown at either two or three price points. Additional price points are included so that quadratic models of price elasticity can be tested. The indicator for the presence or absence of a brand in the shelf set is coded using one level of the `Price` variable. The layout of factors and levels is given in the following table.

Factors and Levels

Alternative	Factor	Levels	Brand	Description
1	X1	4	Client	1.29, 1.69, 2.09 + absent
2	X2	4	Client Line Extension	1.39, 1.89, 2.39, + absent
	X3	2		microwave/stove-top
	X4	2		shelf talker yes/no
3	X5	3	Regional	1.99, 2.49 + absent
4	X6	3	Private Label	1.49, 2.29 absent
	X7	2		microwave/stove-top
5	X8	3	National	1.99 + 2.39 + absent

In addition to intercepts and main effects, we also require that all two-way interactions within alternatives be estimable: `x2*x3, x2*x4, x3*x4` for the line extension and `x6*x7` for private labels. This will enable us to test for different price elasticities by form (stove-top versus microwaveable) and to see if the promotion works better combined with a low price or with different forms. Using a linear model for `x1-x8`, the total number of parameters including the intercept, all main effects, and two-way interactions with brand is 25. This assumes that price is treated as qualitative. The actual number of parameters in the choice model is larger than this because of the inclusion of cross effects. Using indicator variables to code availability, the systematic component of utility for brand i can be expressed as:

$$V_i = a_i + \sum_k (b_{ik} \times x_{ik}) + \sum_{j \neq i} z_j (d_{ij} + \sum_l (g_{ijl} \times x_{jl}))$$

where

a_i = intercept for brand i
b_{ik} = effect of attribute k for brand i, where $k = 1, .., K_i$
x_{ik} = level of attribute k for brand i
d_{ij} = availability cross effect of brand j on brand i
z_j = availability code $= \begin{cases} 1 & \text{if } j \in C_a, \\ 0 & \text{otherwise} \end{cases}$
g_{ijl} = cross effect of attribute l for brand j on brand i, where $l = 1, .., L_j$
x_{jl} = level of attribute l for brand j.

The x_{ik} and x_{jl} could be expanded to include interaction and polynomial terms. In an availability design, each brand is present in only a fraction of the choice sets. The size of this fraction or subdesign is a function of the number of levels of the alternative-specific variable that is used to code availability (usually price). For instance, if price has three valid levels and a fourth zero level to indicate absence, then the brand will appear in only three out of four runs. Following Lazari and Anderson (1994), the size of each subdesign determines how many model equations can be written for each brand in the discrete choice model. If X_i is the subdesign matrix corresponding to V_i, then each X_i must be full rank to ensure that the choice set design provides estimates for all parameters.

To create the design, a full-factorial candidate set is generated consisting of 3456 runs. It is then reduced to 2776 runs that contain between two and four brands so that the respondent is never required to compare more than four brands at a time. In the model specification, we designate all variables as classification variables and require that all main effects and two-way interactions within brands be estimable. The number of runs calculations are based on the number of parameters that we wish to estimate in the various subdesigns $\mathbf{X}_i$ of $\mathbf{X}$. Assuming that there is a None alternative used as a reference level, the numbers of parameters required for various alternatives are shown in the next table along with the sizes of the subdesigns (rounded down) for various numbers of runs. Parameters for quadratic price models are given in parentheses. Note that the effect of private label being in a microwaveable or stove-top form (stove/micro cross effect) is an explicit parameter under the client line extension.

The subdesign sizes are computed by taking the floor of the number of runs from the marginal times the expected proportion of runs in which the alternative will appear. For example, for the client brand which has three prices and not available and 22 runs, floor($22 \times 3/4$) = 16; for the competitor and 32 runs, floor($32 \times 2/3$) = 21. The number of runs chosen was `n=26`. This number provides adequate degrees of freedom for the linear price model and will also allow estimation of direct quadratic price effects. To estimate quadratic cross effects for price would require 32 runs at the very least. Although the technique of using two-way interactions between nominal level variables will usually guarantee that all direct and cross effects are estimable, it is sometimes necessary and good practice to check the ranks of the subdesigns for more complex models (Lazari and Anderson 1994).

Parameters

Effect	Client	Client Line Extension	Regional	Private Label	Competitor
intercept	1	1	1	1	1
availability cross effects	4	4	4	4	4
direct price effect	1 (2)	1 (2)	1	1	1
price cross effects	4 (8)	4 (8)	4	4	4
stove versus microwave	-	1	-	1	-
stove/micro cross effects	-	1	-	-	-
shelf talker	-	1	-	-	-
price*stove/microwave	-	1 (2)	-	1	-
price*shelf talker	-	1 (2)	-	-	-
stove/micro*shelf talker	-	1	-	-	-
Total	10 (15)	16 (23)	10	12	10
Subdesign size					
22 runs	16	16	14	14	14
26 runs	19	19	17	17	17
32 runs	24	24	21	21	21

Designing the Choice Experiment

This example originated with Kuhfeld, Tobias, and Garratt (1994), long before the `%MktRuns` macro existed. At least for now, we will skip the customary step of running the `%MktRuns` macro to suggest a design size and instead use the original size of 26 choice sets.

We will use the `%MktEx` autocall macro to create the design. (All of the autocall macros used in this book are documented starting on page 597.) To recap, we want to make the design $2^3 3^3 4^2$ in 26 runs, and we want the following interactions to be estimable: `x2*x3 x2*x4 x3*x4 x6*x7`. Furthermore, there are restrictions on the design. Each of the price variables, `x1`, `x2`, `x5`, `x6`, and `x8`, has one level−the maximum level−that indicates the alternative is not available in the choice set. We use this to create choice sets with 2, 3, or 4 alternatives available. If `(x1 < 4)` then the first alternative is available, if `(x2 < 4)` then the second alternative is available, if `(x5 < 3)` then the third alternative is available, and so on. A Boolean term such as `(x1 < 4)` is one when true and zero otherwise. Hence,

```
((x1 < 4) + (x2 < 4) + (x5 < 3) + (x6 < 3) + (x8 < 3))
```

is the number of available alternatives. It is simply the sum of some 1's if available and 0's if not available.

We impose restrictions with the `%MktEx` macro by writing a macro, with IML statements, that quantifies the badness of each run (or in this case, each choice set). We do this so `bad` = 0 is good and values larger than zero are increasingly worse. We write our restrictions using an IML row vector `x` that contains the levels (integers beginning with 1) of each of the factors in the *ith* choice set, the one the macro is currently seeking to improve. The *jth* factor is `x[j]`, or we may also use the factor names (for example, `x1`, `x2`). (See pages 403 and 700 for other examples of restrictions.)

We must use IML logical operators, which do not have all of the same syntax alternatives as DATA step operators:

Specify	For	Do Not Specify
=	equals	EQ
∧ = or ¬ =	not equals	NE
<	less than	LT
<=	less than or equal to	LE
>	greater than	GT
>=	greater than or equal to	GE
&	and	AND
|	or	OR
∧ or ¬	not	NOT

To restrict the design, we must specify `restrictions=`*macro-name*, in this case `restrictions=resmac`, that names the macro that quantifies badness. The first statement counts up the number of available alternatives. The next two set the actual badness value. Note that the `else bad = 0` statement is not necessary since`bad` is automatically initialized to zero by the `%MktEx` macro. If the number available is less than two or greater than 4, then `bad` gets set to the absolute difference between the number available and 3. Hence, zero available corresponds to `bad = 3`, one available corresponds to `bad = 2`, two through four available corresponds to `bad = 0`, and five available corresponds to `bad = 2`. Do not just set `bad` to zero when everything is fine and one otherwise, but the macro needs to know that when it switches from zero available to one available, it is going in the right direction. For simple restrictions like this, it does not matter very much. However, for complicated sets of restrictions, it is critical that the *bad* variable is set to a count of the number of current restriction violations. Here is the code.¶

```
title 'Consumer Food Product Example';

%macro resmac;
  navail = (x1 < 4) + (x2 < 4) + (x5 < 3) + (x6 < 3) + (x8 < 3);
  if (navail < 2) | (navail > 4) then bad = abs(navail - 3);
  else                               bad = 0;
  %mend;

%mktex( 4 4 2 2 3 3 2 3, n=26, interact=x2*x3 x2*x4 x3*x4 x6*x7,
        restrictions=resmac, seed=377, outr=sasuser.Entree_LinDes1 )
```

Here are the initial messages the macro prints.

```
NOTE: Generating the fractional-factorial design, n=27.
NOTE: Generating the candidate set.
NOTE: Performing 60 searches of 2,776 candidates, full-factorial=3,456.
```

The tabled design initialization part of the coordinate-exchange algorithm iterations will be initialized with the first 26 rows of a 27 run fractional-factorial design. This design has 13 three-level factors, ten of which are used to make $2^3 3^3 4^2$. The initial design will be unbalanced and one row short of orthogonal, so we would expect that other methods would be better for this problem. The macro also tells us that it is performing 60 PROC OPTEX searches of 2776 candidates, and that the full-factorial

¶Due to machine, SAS release, and macro differences, you may not get exactly the same design as was used in this book, but the differences should be slight.

design has 3456 runs. The macro is searching the full-factorial design minus the excluded choice sets. Since the full-factorial design is not too large (less than 5000), and since there is no tabled design that is very good for this problem, this is the kind of problem where we would expect the PROC OPTEX modified Fedorov algorithm (Fedorov, 1972; Cook and Nachtsheim, 1980) algorithm to work best. The macro chose 60 OPTEX iterations. In the fabric softener example, the macro did not try any OPTEX iterations, because it knew it could directly make a 100% *D*-efficient design. In the vacation examples, it ran the default minimum of 20 OPTEX iterations because the macro's heuristics concluded that OPTEX would probably not be the best approach for those problems. In this example, the macro's heuristics tried more iterations, since this is the kind of example where OPTEX works best.

Here is some of the output.

Consumer Food Product Example

Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
1	Start	84.3176		Can
1	2 1	84.3176	84.3176	Conforms
1	End	84.3176		
2	Start	27.8626		Tab,Unb,Ran
2	1 1	76.5332		Conforms
2	End	80.4628		
.				
.				
.				
11	Start	24.5507		Tab,Ran
11	26 1	78.6100		Conforms
11	End	81.8604		
12	Start	26.3898		Ran,Mut,Ann
12	1 1	67.0450		Conforms
12	End	83.0114		
.				
.				
.				
21	Start	45.9310		Ran,Mut,Ann
21	15 1	67.1046		Conforms
21	End	82.1657		

NOTE: Performing 600 searches of 2,776 candidates.

Consumer Food Product Example

Design Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
0	Initial	84.3176	84.3176	Ini
1	Start	84.7548		Can
1	2 1	84.7548	84.7548	Conforms
1	End	84.7548		

Consumer Food Product Example

Design Refinement History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
0	Initial	84.7548	84.7548	Ini
1	Start	84.7548		Pre,Mut,Ann
1	2 1	82.6737		Conforms
1	14 1	84.7548	84.7548	
1	End	82.6386		
.				
.				
.				
8	Start	84.7548		Pre,Mut,Ann
8	2 1	84.7548	84.7548	Conforms
8	14 1	84.7548	84.7548	
8	21 2	84.7548	84.7548	
8	12 3	84.7548	84.7548	
8	12 6	84.7548	84.7548	
8	18 1	84.7548	84.7548	
8	2 2	84.7548	84.7548	
8	End	84.7548		

NOTE: Stopping since it appears that no improvement is possible.

Consumer Food Product Example

The OPTEX Procedure

Class Level Information

Class	Levels	-Values-
x1	4	1 2 3 4
x2	4	1 2 3 4
x3	2	1 2
x4	2	1 2
x5	3	1 2 3
x6	3	1 2 3
x7	2	1 2

Consumer Food Product Example

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	84.7548	71.1686	98.0583	0.9806

Design 1 (`Can`), which was created by the candidate-set search (using PROC OPTEX), had *D*-efficiency or 84.3176, and the macro confirms that the design conforms to our restrictions. The tabled, unbalanced, and random initializations do not work as well. For each design, the macro iteration history states the *D*-efficiency for the initial design (27.8626 in design 2), the *D*-efficiency when the restrictions are met (76.5332, `Conforms`), and the *D*-efficiency for the final design (80.4628). The fully-random initialization tends to work a little better than the tabled initialization for this problem, but not as well as PROC OPTEX. At the end of the algorithm search phase, the macro decides to use PROC OPTEX and performs 600 more searches, and it finds a design with 84.7548% *D*-efficiency. The design refinement step fails to improve on the best design. This step took 3.5 minutes.

When You Have a Long Time to Search for an Efficient Design

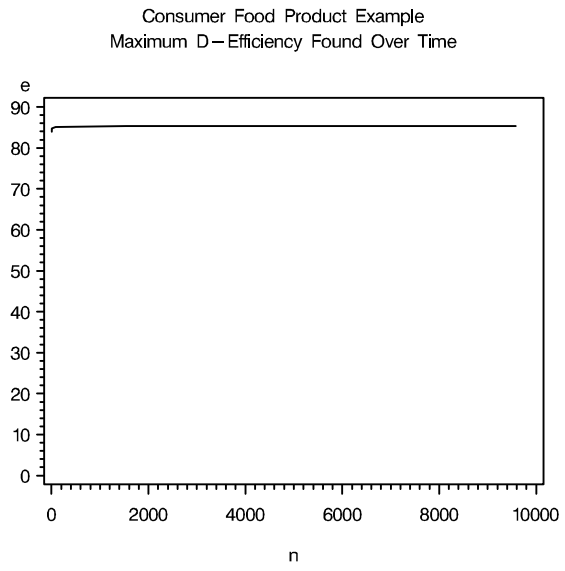
With a moderate sized candidate set such as this one (2000 to 6000 runs), we might be able to do better with more iterations. To test this, PROC OPTEX was run 10,000 times over the winter holiday vacation, from December 22 through January 2, creating a total of 200,000 designs, 20 designs on each try. (This was many years ago on computers that were much slower than the ones we have today.) Here is a summary of the results.

PROC OPTEX Run	*D*-Efficiency	Percent Improvement
1	83.8959	
2	83.9890	0.11%
3	84.3763	0.46%
6	84.7548	0.45%
84	85.1561	0.47%
1535	85.3298	0.20%
9576	85.3985	0.08%

This example is interesting, because it shows the diminishing value of increasing the number of iterations. Six minutes into the search, in the first six passes through PROC OPTEX ($6 \times 20 = 120$ total iterations), we found a design with reasonably good *D*-efficiency=84.7548. Over an hour into the search, with $(84 - 6) \times 20 = 1560$ more iterations, we get a small 0.47% increase in *D*-efficiency to 85.1561. About one day into the search, with $(1535 - 84) \times 20 = 29,020$ more iterations, we get another small 0.20% increase in *D*-efficiency, 85.3298. Finally, almost a week into the search, with $(9576 - 1535) \times 20 = 160,820$ more iterations, we get another small 0.08% increase in *D*-efficiency to 85.3985. Our overall improvement over the best design found in 120 iterations was 0.75952%, about three-quarters of a percent. These numbers will change with other problems and other seeds. However, as these results show, usually the first few iterations will give you a good, efficient design, and usually, subsequent iterations will give you slight improvements but with a cost of much greater run times. Next, we will construct a plot of this table.

```
data; input n e; datalines;
    1  83.8959
    2  83.9890
    3  84.3763
    6  84.7548
   84  85.1561
 1535  85.3298
 9576  85.3985
;
proc gplot;
   title  h=1 'Consumer Food Product Example';
   title2 h=1 'Maximum D-Efficiency Found Over Time';
   plot e * n / vaxis=axis1;
   symbol i=join;
   axis1 order=(0 to 90 by 10);
   run; quit;
```

The plot of maximum *D*-efficiency as a function of PROC OPTEX run number clearly shows that the gain in *D*-efficiency that comes from a large number of iterations is very slight.



If you have a lot of time to search for a good design, you can specify some of the time and maximum number of iteration parameters. Sometimes you will get lucky and find a better design. In this next example, `maxtime=300 300 60` was specified. This gives the macro up to 300 minutes for the algorithm search step, 300 minutes for the design search step, and 60 minutes for the refinement step. The option `maxiter=` increases the number iterations to 10000 for each of the three steps (or the maximum time). With this specification, you would expect the macro to run overnight. See the macro documentation (starting on page 667) for more iteration options. Note that you must increase the number of iterations and the maximum amount of time if you want the macro to run longer. With this specification, the macro performs 1800 OPTEX iterations initially (compared to 60 by default).

```
title 'Consumer Food Product Example';

%macro resmac;
  navail = (x1 < 4) + (x2 < 4) + (x5 < 3) + (x6 < 3) + (x8 < 3);
  if (navail < 2) | (navail > 4) then bad = abs(navail - 3);
  else                               bad = 0;
  %mend;

%mktex( 4 4 2 2 3 3 2 3, n=26, interact=x2*x3 x2*x4 x3*x4 x6*x7,
        restrictions=resmac, seed=151,
        maxtime=300 300 60, maxiter=10000 )
```

The results from this step are not shown.

Examining the Design

We can use the `%MktEval` macro to start to evaluate the design.

```
%mkteval(data=sasuser.Entree_LinDes1);
```

Here are the results.

Consumer Food Product Example
Canonical Correlations Between the Factors
There are 4 Canonical Correlations Greater Than 0.316

	x1	x2	x3	x4	x5	x6	x7	x8
x1	1	0.30	0.20	0.11	0.42	0.26	0.09	0.33
x2	0.30	1	0.10	0.10	0.13	0.17	0.51	0.18
x3	0.20	0.10	1	0.08	0.09	0.30	0	0.10
x4	0.11	0.10	0.08	1	0.09	0.10	0	0.10
x5	0.42	0.13	0.09	0.09	1	0.24	0.05	0.43
x6	0.26	0.17	0.30	0.10	0.24	1	0.14	0.13
x7	0.09	0.51	0	0	0.05	0.14	1	0.14
x8	0.33	0.18	0.10	0.10	0.43	0.13	0.14	1

Consumer Food Product Example
Canonical Correlations > 0.316 Between the Factors
There are 4 Canonical Correlations Greater Than 0.316

		r	r Square
x2	x7	0.51	0.26
x5	x8	0.43	0.18
x1	x5	0.42	0.18
x1	x8	0.33	0.11

Consumer Food Product Example
Summary of Frequencies
There are 4 Canonical Correlations Greater Than 0.316
* - Indicates Unequal Frequencies

		Frequencies
*	x1	7 8 6 5
*	x2	6 7 7 6
	x3	13 13
	x4	13 13
*	x5	9 8 9
*	x6	7 10 9
*	x7	12 14
*	x8	7 9 10

```
*    x1 x2    2 2 1 2 2 2 2 2 1 1 2 2 1 2 2 0
*    x1 x3    3 4 4 4 4 2 2 3
*    x1 x4    4 3 4 4 3 3 2 3
*    x1 x5    4 2 1 2 1 5 2 2 2 1 3 1
*    x1 x6    2 3 2 2 4 2 2 1 3 1 2 2
*    x1 x7    3 4 4 4 3 3 2 3
*    x1 x8    1 2 4 2 4 2 2 1 3 2 2 1
*    x2 x3    3 3 3 4 4 3 3 3
*    x2 x4    3 3 3 4 4 3 3 3
*    x2 x5    2 2 2 3 2 2 2 2 3 2 2 2
*    x2 x6    2 2 2 2 3 2 2 2 3 1 3 2
*    x2 x7    1 5 4 3 2 5 5 1
*    x2 x8    2 2 2 1 3 3 2 2 3 2 2 2
*    x3 x4    7 6 6 7
*    x3 x5    5 4 4 4 4 5
*    x3 x6    2 5 6 5 5 3
*    x3 x7    6 7 6 7
*    x3 x8    4 4 5 3 5 5
*    x4 x5    4 4 5 5 4 4
*    x4 x6    4 5 4 3 5 5
*    x4 x7    6 7 6 7
*    x4 x8    4 4 5 3 5 5
*    x5 x6    2 4 3 2 2 4 3 4 2
*    x5 x7    4 5 4 4 4 5
*    x5 x8    1 2 6 4 2 2 2 5 2
*    x6 x7    3 4 4 6 5 4
*    x6 x8    2 2 3 2 4 4 3 3 3
*    x7 x8    4 4 4 3 5 6
     N-Way    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
              1 1 1 1 1 1 1
```

Some of the canonical correlations are bigger than we would like. They all involve attributes in different alternatives, so they should not pose huge problems. Still, they are large enough to make some researchers uncomfortable. The frequencies are pretty close to balanced. Perfect balance is not possible with 26 choice sets and this design. If we were willing to consider blocking the design, we might do better with more choice sets.

Designing the Choice Experiment, More Choice Sets

Let's run the `%MktRuns` macro to see what design size looks good. For now, we will ignore the interactions.

```
%mktruns( 4 4 2 2 3 3 2 3 )
```

Consumer Food Product Example

Design Summary

Number of Levels	Frequency
2	3
3	3
4	2

Consumer Food Product Example

Saturated = 16
Full Factorial = 3,456

Some Reasonable Design Sizes	Violations	Cannot Be Divided By
144 *	0	
72	1	16
48	3	9
96	3	9
192	3	9
24	4	9 16
120	4	9 16
168	4	9 16
36	7	8 16
108	7	8 16

* - 100% Efficient Design can be made with the MktEx Macro.

Consumer Food Product Example

n	Design	Reference
144	2 ** 48 3 ** 3 4 ** 2	Orthogonal Array
144	2 ** 44 3 ** 3 4 ** 3	Orthogonal Array
144	2 ** 41 3 ** 4 4 ** 2	Orthogonal Array
144	2 ** 39 3 ** 3 4 ** 2 6 ** 1	Orthogonal Array
144	2 ** 37 3 ** 4 4 ** 3	Orthogonal Array
144	2 ** 37 3 ** 3 4 ** 2 12 ** 1	Orthogonal Array
144	2 ** 35 3 ** 3 4 ** 3 6 ** 1	Orthogonal Array
.		
.		
.		

The smallest suggestion larger than 26 is 36. With this mix of factor levels, we would have to have 144 runs to get an orthogonal design (ignoring interactions), so we will definitely want to stick with a nonorthogonal design. Balance will be possible in 36 runs, but 36 cannot be divided by $2 \times 4 = 8$ and $4 \times 4 = 16$. With 36 runs, a blocking factor will be required (2 blocks of 18 or 3 blocks of 12). We would like the shelf talker to appear in half of the choice sets within block, so with two blocks, we will want the number of choice sets to be divisible by $2 \times 2 = 4$, and 36 can be divided by 4. Now let's specify the interactions.

```
%mktruns( 4 4 2 2 3 3 2 3, interact=x2*x3 x2*x4 x3*x4 x6*x7 )
```

Here is the output.

Consumer Food Product Example

Design Summary

Number of Levels	Frequency
2	3
3	3
4	2

Consumer Food Product Example

Saturated = 25
Full Factorial = 3,456

Some Reasonable Design Sizes	Violations	Cannot Be Divided By
144	2	32
96	5	9 18
192	5	9 18
48	7	9 18 32
72	9	16 32 48
216	9	16 32 48
120	14	9 16 18 32 48
168	14	9 16 18 32 48
36	25	8 16 24 32 48
108	25	8 16 24 32 48

Thirty-six runs is still in our list of possibilities, but now we see that not only can it not be divided by 8 and 16, it also cannot be divided by 24, 32, 48. We will try making a design in 36 runs, and see how it looks.

In the previous try in 26 runs, the PROC OPTEX modified Fedorov algorithm worked best. There are two reasons why this probably happened. First, the full-factorial design was small enough to use as a candidate set. After imposing restrictions, the candidate set had 2,776 runs, and any size under 5000 or 10,000 is very manageable. Second, the design has interactions. The coordinate exchange

algorithm by default considers only a single factor at a time, which is just one part of an interaction term. Modified Fedorov in contrast, considers exchanges involving all of the factors. For this problem, Modified Fedorov is invariably superior to the default coordinate-exchange algorithm. However, we can make coordinate exchange better, by having it perform multiple-column exchanges taking into account the interactions, just as we did in the vacation example on page 235. We will use `order=matrix=`*SAS-data-set* approach to looping over the columns of the design with the coordinate-exchange algorithm. In this case, coordinate exchange will pair columns 1, 5, and 8 with a randomly chosen column, it will consider every possible triple in columns 2, 3, and 4, and it will pair columns 6 and 7 with a randomly chosen column.

```
title 'Consumer Food Product Example';

%macro resmac;
  navail = (x1 < 4) + (x2 < 4) + (x5 < 3) + (x6 < 3) + (x8 < 3);
  if (navail < 2) | (navail > 4) then bad = abs(navail - 3);
  else                               bad = 0;
  %mend;

data mat;
   input a b c;
   datalines;
1 1 .
2 3 4
5 5 .
6 7 .
8 8 .
;

%mktex( 4 4 2 2 3 3 2 3, n=36, order=matrix=mat,
        interact=x2*x3 x2*x4 x3*x4 x6*x7,
        restrictions=resmac, seed=377, outr=sasuser.Entree_LinDes2 )

%mkteval;
```

Here is a small part of the output from the `%MktEx` macro.

```
                    Consumer Food Product Example                    1

                       Algorithm Search History

                                 Current          Best
Design    Row,Col  D-Efficiency  D-Efficiency  Notes
-----------------------------------------------------------
     1      Start       94.0517                Can
     1      2   1       94.0517       94.0517  Conforms
     1        End       94.0517
     .
     .
     .
```

12	Start		71.6955		Ran,Mut,Ann
12	1	1	78.5418		Conforms
12	30	5	94.1433	94.1433	
12	33	5	94.1507	94.1507	
12	31	1	94.1532	94.1532	
12	23	6	94.1553	94.1553	
12	End		94.1553		

Design Search History

Design	Row,Col		Current D-Efficiency	Best D-Efficiency	Notes
0	Initial		94.1553	94.1553	Ini
.					
.					
.					
3	Start		68.5288		Ran,Mut,Ann
3	29	1	75.9029		Conforms
3	22	5	94.1682	94.1682	
3	34	5	94.1683	94.1683	
3	35	6	94.2926	94.2926	
3	16	8	94.3718	94.3718	
3	24	6	94.3718	94.3718	
3	9	1	94.4572	94.4572	
3	End		94.2846		
.					
.					
.					

Consumer Food Product Example

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	94.4571	88.7104	94.0740	0.8333

The `order=matrix=` option apparently helped. The coordinate exchange algorithm was in fact chosen over the modified Fedorov algorithm.

D-efficiency at 94.46% looks good. Here is part of the `%MktEval` results.

Consumer Food Product Example
Canonical Correlations Between the Factors
There is 1 Canonical Correlation Greater Than 0.316

	x1	x2	x3	x4	x5	x6	x7	x8
x1	1	0.13	0.10	0.11	0.11	0.17	0.10	0.12
x2	0.13	1	0.12	0.08	0.23	0.39	0.06	0.18
x3	0.10	0.12	1	0.06	0.10	0.04	0.00	0.10
x4	0.11	0.08	0.06	1	0.07	0.07	0.06	0.18
x5	0.11	0.23	0.10	0.07	1	0.13	0.04	0.15
x6	0.17	0.39	0.04	0.07	0.13	1	0.04	0.13
x7	0.10	0.06	0.00	0.06	0.04	0.04	1	0.04
x8	0.12	0.18	0.10	0.18	0.15	0.13	0.04	1

Consumer Food Product Example
Canonical Correlations > 0.316 Between the Factors
There is 1 Canonical Correlation Greater Than 0.316

		r	r Square
x2	x6	0.39	0.15

Consumer Food Product Example
Summary of Frequencies
There is 1 Canonical Correlation Greater Than 0.316
* - Indicates Unequal Frequencies

		Frequencies
	x1	9 9 9 9
*	x2	8 9 10 9
*	x3	19 17
	x4	18 18
*	x5	11 11 14
*	x6	12 13 11
*	x7	17 19
*	x8	11 12 13

The correlations are better, although one is still not as good as we would like. The balance looks pretty good, however it would be nice if the balance, for example, in `x5` were better. It is often the case that improving balance requires some sacrifice of *D*-efficiency. We can run the macro again, this time specifying `balance=2`, which forces better balance. The specification of 2 allows the maximum frequency for a level in a factor to be no more than two greater than the minimum frequency. You should always specify `mintry=` with `balance=`. This allows `%MktEx` to at first increase *D*-efficiency while ignoring the balance restrictions. Then, after `mintry=`m rows have been processed, the balance restrictions are considered. Typically you will specify an expression that is a function of the number of

rows for `mintry=`, for example, `mintry=5 * n`. The `balance=` option works best when its restrictions are imposed on a reasonably efficient design not an inefficient initial design.

This example also uses a somewhat more involved `order=matrix` data set. To understand why, you need to understand how the `balance=` option works. Here is some of the code that `%MktEx` uses to impose balance.

```
__bbad = 1;
if try > &balancetry & j1 then do;
   acol = xmat[,j1];
   acol[i,] = x[,j1];
   acol = design(acol)[+,];
   __bbad = max(0, max(acol) - min(acol) - &balance);
   end;
```

It checks the balance restrictions based on the first column index, `j1`. If we are doing multiple exchanges, the exchanges in the second or subsequent columns could degrade the balance without it registering as a violation in the code above. For example, in the `order=matrix=mat` data set used previously, the last line is: `8 8 .`. The column index `j3` could change any of the columns and it would not register in the balance-checking code, because it is only looking at column 8. For this reason, we add eight more lines so the last thing the restrictions macro does in each row is check every column for the balance constraints.

```
data mat;
   input a b c;
   datalines;
1 1 .
2 3 4
5 5 .
6 7 .
8 8 .
1 1 1
2 2 2
3 3 3
4 4 4
5 5 5
6 6 6
7 7 7
8 8 8
;
%mktex( 4 4 2 2 3 3 2 3, n=36, order=matrix=mat,
        interact=x2*x3 x2*x4 x3*x4 x6*x7,
        restrictions=resmac, seed=368, outr=sasuser.Entree_LinDes3,
        balance=2, mintry=5 * n )
```

Here is the last part of the output from the `%MktEx` macro.

Consumer Food Product Example

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	93.9552	87.8357	92.9627	0.8333

The *D*-efficiency looks good. It is a little lower than before, but not much. Next, we will look at the canonical correlations and frequencies.

```
%mkteval;
```

Here is the first part of the output from the %MktEval macro.

Consumer Food Product Example
Canonical Correlations Between the Factors
There are 0 Canonical Correlations Greater Than 0.316

	x1	x2	x3	x4	x5	x6	x7	x8
x1	1	0.17	0.08	0.08	0.16	0.12	0.18	0.16
x2	0.17	1	0.08	0.08	0.16	0.31	0.27	0.16
x3	0.08	0.08	1	0.11	0.12	0.07	0	0.12
x4	0.08	0.08	0.11	1	0.12	0.07	0	0.07
x5	0.16	0.16	0.12	0.12	1	0.13	0.07	0.10
x6	0.12	0.31	0.07	0.07	0.13	1	0.07	0.19
x7	0.18	0.27	0	0	0.07	0.07	1	0.12
x8	0.16	0.16	0.12	0.07	0.10	0.19	0.12	1

The canonical correlations look good. Here is the last part of the output from the %MktEval macro.

```
                 Consumer Food Product Example
                    Summary of Frequencies
There are 0 Canonical Correlations Greater Than 0.316
               * - Indicates Unequal Frequencies

                     Frequencies

*    x1        9 8 9 10
*    x2        8 9 10 9
     x3        18 18
     x4        18 18
*    x5        12 11 13
*    x6        11 12 13
     x7        18 18
*    x8        11 12 13
*    x1 x2     2 2 2 3 2 2 2 2 2 3 2 2 2 2 4 2
*    x1 x3     5 4 4 4 4 5 5 5
*    x1 x4     4 5 4 4 5 4 5 5
*    x1 x5     3 3 3 2 2 4 3 3 3 4 3 3
*    x1 x6     3 3 3 3 2 3 2 3 4 3 4 3
*    x1 x7     4 5 5 3 5 4 4 6
*    x1 x8     3 3 3 2 2 4 2 4 3 4 3 3
*    x2 x3     4 4 4 5 5 5 5 4
*    x2 x4     4 4 5 4 5 5 4 5
*    x2 x5     3 3 2 3 2 4 3 3 4 3 3 3
*    x2 x6     2 2 4 3 2 4 2 4 4 4 4 1
*    x2 x7     2 6 5 4 6 4 5 4
*    x2 x8     2 2 4 3 3 3 4 3 3 2 4 3
*    x3 x4     8 10 10 8
*    x3 x5     7 5 6 5 6 7
*    x3 x6     5 6 7 6 6 6
     x3 x7     9 9 9 9
*    x3 x8     6 5 7 5 7 6
*    x4 x5     5 6 7 7 5 6
*    x4 x6     6 6 6 5 6 7
     x4 x7     9 9 9 9
*    x4 x8     6 6 6 5 6 7
*    x5 x6     4 4 4 3 3 5 4 5 4
*    x5 x7     6 6 6 5 6 7
*    x5 x8     3 4 5 4 3 4 4 5 4
*    x6 x7     5 6 6 6 7 6
*    x6 x8     2 4 5 4 4 4 5 4 4
*    x7 x8     5 7 6 6 5 7
     N-Way     1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
               1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

This design looks much better. It is possible to get designs with better balance by specifying `balance=1`, however, since this gives `%MktEx` much less freedom, the `balance=1` option may cause *D*-efficiency to go down. Because `balance=1` is a tough restriction, we will try this without `order=matrix`.

```
%mktex( 4 4 2 2 3 3 2 3, n=36,
        interact=x2*x3 x2*x4 x3*x4 x6*x7,
        restrictions=resmac, seed=472, outr=sasuser.Entree_LinDes4,
        balance=1, mintry=5 * n )

%mkteval;
```

Here is the *D*-efficiency, which is a lower than we saw previously.

Consumer Food Product Example

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	90.4983	79.9621	87.0176	0.8333

More troubling is the fact that the balance restrictions have increased the correlations between factors.

Consumer Food Product Example
Canonical Correlations Between the Factors
There are 2 Canonical Correlations Greater Than 0.316

	x1	x2	x3	x4	x5	x6	x7	x8
x1	1	0.22	0.11	0.11	0.19	0.33	0.11	0.30
x2	0.22	1	0.11	0.11	0.44	0.29	0.11	0
x3	0.11	0.11	1	0	0.14	0	0	0.14
x4	0.11	0.11	0	1	0.14	0	0.11	0.14
x5	0.19	0.44	0.14	0.14	1	0.14	0	0.17
x6	0.33	0.29	0	0	0.14	1	0.14	0
x7	0.11	0.11	0	0.11	0	0.14	1	0.14
x8	0.30	0	0.14	0.14	0.17	0	0.14	1

```
                Consumer Food Product Example
          Canonical Correlations > 0.316 Between the Factors
        There are 2 Canonical Correlations Greater Than 0.316


                                    r    r Square

                      x2    x5    0.44      0.20
                      x1    x6    0.33      0.11
```

The balance, however, is perfect.

```
                Consumer Food Product Example
                    Summary of Frequencies
        There are 2 Canonical Correlations Greater Than 0.316
                 * - Indicates Unequal Frequencies

                       Frequencies

              x1       9 9 9 9
              x2       9 9 9 9
              x3       18 18
              x4       18 18
              x5       12 12 12
              x6       12 12 12
              x7       18 18
              x8       12 12 12
```

Having balance in all of the factors is nice, but for this design, we only need to ensure that `x4`, the shelf-talker factor is balanced, since we will be dividing the design into two parts depending on whether the shelf talker is there or not. All things considered, it looks like the design that was created with `balance=2` is the best design for our situation. It is balanced in `x4`, it is either balanced or reasonably close to balanced in the other factors, and it has good D-efficiency and is reasonably close to orthogonal. If our design had not been balanced in `x4`, we could have tried again with a different seed, or we could have tried again with different values for `mintry=`. If the interactions had not been requested, we also could have switched it with another two-level factor, or added it after the fact by blocking (running the `%MktBlock` macro as if we were adding a blocking factor), or we could have used the `init=` option to constrain the factor to be balanced.

The `balance=` option in the `%MktEx` macro works by adding restrictions to the design. The approach it uses often works quite well, but sometimes it does not. Forcing balance gives the macro much less freedom in its search, and makes it easy for the macro to get stuck in suboptimal designs. If perfect balance is critical and there are no interactions or restrictions, you can also try the `%MktBal` macro.

Examining the Subdesigns

As we mentioned previously, "it is sometimes necessary and good practice to check the ranks of the subdesigns for more complex models (Lazari and Anderson 1994)." Here is a way to do that with PROC OPTEX. This is the only usage of PROC OPTEX in this book that is too specialized to be run from one of the `%Mkt` macros (because not all variables are designated as `class` variables). For convenience, we call PROC OPTEX from an ad hoc macro, since it must be run five times, once per alternative, with only a change in the `where` statement. We need to evaluate the design when the client's alternative is available `(x1 ne 4)`, when the client line extension alternative is available `(x2 ne 4)`, when the regional competitor is available `(x5 ne 3)`, when the private label competitor is available `(x6 ne 3)`, and when the national competitor is available `(x8 ne 3)`. We need to use a `model` statement that lists all of the main effects and interactions. We do not designate all of the variables on the `class` statement because we only have enough runs to consider linear price effects within each availability group. The statement `generate method=sequential initdesign=desv` specifies that we will be evaluating the initial design `desv`, using the sequential algorithm, which ensures no swaps between the candidate set and the initial design. The other option of note here appears on the `class` statement, and that is `param=orthref`. This specifies an orthogonal parameterization of the effects that gives us a nice 0 to 100 scale for the *D*-efficiencies.

```
%macro evaleff(where);
data desv / view=desv; set sasuser.Entree_LinDes3(where=(&where)); run;

proc optex data=desv;
   class x3 x4 x7 / param=orthref;
   model x1-x8 x2*x3 x2*x4 x3*x4 x6*x7;
   generate method=sequential initdesign=desv;
   run; quit;

%mkteval(data=desv)
%mend;

%evaleff(x1 ne 4)
%evaleff(x2 ne 4)
%evaleff(x5 ne 3)
%evaleff(x6 ne 3)
%evaleff(x8 ne 3)
```

Each step took just over two seconds. We hope to not see any efficiencies of zero, and we hope to not get the message `WARNING: Can't estimate model parameters in the final design.` Here are some of the results.

Consumer Food Product Example

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	69.7007	61.6709	80.8872	0.7071

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	72.7841	64.9939	87.5576	0.6939

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	66.1876	50.8651	81.2554	0.7518

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	71.8655	59.8208	86.6281	0.7518

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	65.2313	50.1059	84.1610	0.7518

Examining the Aliasing Structure

It is also good to look at the aliasing structure of the design. We use PROC GLM to do this, so we must create a dependent variable. We will use a constant `y=1`. The first PROC GLM step just checks the model to make sure none of the specified effects are aliased with each other. This step is not necessary since our *D*-efficiency value greater than zero already guarantees this.

```
data temp;
   set sasuser.Entree_LinDes3;
   y = 1;
   run;
proc glm data=temp;
   model y = x1-x8 x2*x3 x2*x4 x3*x4 x6*x7 / e aliasing;
   run; quit;
```

Here are the results, ignoring the ANOVA and regression tables, which are not of interest. Each of these lines is a linear combination that is estimable. It is simply a list of the effects.

```
Intercept
x1
x2
x3
x4
x5
x6
x7
x8
x2*x3
x2*x4
x3*x4
x6*x7
```

Contrast this with a specification that includes all simple effects and two-way and three-way interactions. We specify the model of interest first, `x1-x8 x2*x3 x2*x4 x3*x4 x6*x7`, so all of those terms will be listed first, then we specify all main effects and two-way and three-way interactions using the notation `x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8@3`. It is not a problem that some of the terms were both explicitly specified and also generated by the `x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8@3` list since PROC GLM automatically eliminates duplicate terms.

```
proc glm data=temp;
   model y = x1-x8 x2*x3 x2*x4 x3*x4 x6*x7
             x1|x2|x3|x4|x5|x6|x7|x8@3 / e aliasing;
   run; quit;
```

```
Intercept - 20.008*x4*x6 - 9.8483*x1*x4*x6 - 42.279*x2*x4*x6 - 9.0597*x3*x4*x6 +
57.417*x5*x6 + 151.23*x1*x5*x6 + 186.61*x2*x5*x6 + 80.158*x3*x5*x6 +
90.545*x4*x5*x6 - 50.89*x1*x7 + 4.2117*x2*x7 - 159.53*x1*x2*x7 + 12.566*x3*x7 -
52.475*x1*x3*x7 + 43.269*x2*x3*x7 + 0.3801*x4*x7 - 71.5*x1*x4*x7 +
36.725*x2*x4*x7 + 24.297*x3*x4*x7 + 21.563*x5*x7 - 27.16*x1*x5*x7 +
75.528*x2*x5*x7 + 62.984*x3*x5*x7 + 39.224*x4*x5*x7 - 85.333*x1*x6*x7 -
10.566*x2*x6*x7 + 15.818*x3*x6*x7 - 31.415*x4*x6*x7 + 123.51*x5*x6*x7 -
24.144*x1*x8 + 6.6197*x2*x8 - 12.153*x1*x2*x8 - 38.1*x3*x8 - 133.06*x1*x3*x8 -
135.02*x2*x3*x8 + 39.148*x4*x8 + 101.08*x1*x4*x8 + 149.27*x2*x4*x8 -
15.467*x3*x4*x8 - 30.981*x5*x8 - 157.71*x1*x5*x8 - 130.69*x2*x5*x8 -
107.69*x3*x5*x8 + 19.478*x4*x5*x8 - 40.116*x6*x8 - 116.84*x1*x6*x8 -
61.852*x2*x6*x8 - 97.721*x3*x6*x8 - 23.772*x4*x6*x8 + 44.985*x5*x6*x8 -
5.0186*x7*x8 - 171.5*x1*x7*x8 + 12.071*x2*x7*x8 - 2.9687*x3*x7*x8 +
44.468*x4*x7*x8 + 8.5765*x5*x7*x8 - 52.648*x6*x7*x8
```

```
x1 + 9.1371*x4*x6 + 7.8312*x1*x4*x6 + 17.618*x2*x4*x6 + 9.7563*x3*x4*x6 -
21.745*x5*x6 - 69.803*x1*x5*x6 - 73.705*x2*x5*x6 - 39.359*x3*x5*x6 -
25.304*x4*x5*x6 + 22.962*x1*x7 - 2.9296*x2*x7 + 71.792*x1*x2*x7 - 4.9586*x3*x7 +
26.888*x1*x3*x7 - 12.562*x2*x3*x7 - 7.8969*x4*x7 + 11.379*x1*x4*x7 -
35.377*x2*x4*x7 - 21.468*x3*x4*x7 - 12.723*x5*x7 + 10.604*x1*x5*x7 -
43.808*x2*x5*x7 - 32.655*x3*x5*x7 - 32.497*x4*x5*x7 + 31.754*x1*x6*x7 +
6.8554*x2*x6*x7 - 4.0467*x3*x6*x7 + 1.6149*x4*x6*x7 - 46.784*x5*x6*x7 -
1.133*x1*x8 + 7.3858*x2*x8 + 2.0538*x1*x2*x8 + 4.336*x3*x8 + 3.3233*x1*x3*x8 +
39.854*x2*x3*x8 - 5.3094*x4*x8 - 28.994*x1*x4*x8 - 5.5582*x2*x4*x8 +
7.6916*x3*x4*x8 + 6.3495*x5*x8 + 15.979*x1*x5*x8 + 58.815*x2*x5*x8 +
16.519*x3*x5*x8 + 11.175*x4*x5*x8 + 7.3054*x6*x8 + 13.278*x1*x6*x8 +
29.443*x2*x6*x8 + 14.09*x3*x6*x8 + 18.767*x4*x6*x8 - 34.202*x5*x6*x8 +
5.8152*x7*x8 + 65.231*x1*x7*x8 + 14.788*x2*x7*x8 - 3.885*x3*x7*x8 -
15.536*x4*x7*x8 - 6.816*x5*x7*x8 + 18.202*x6*x7*x8
.
.
.
```

Again, we have a list of linear combinations that are estimable. This shows that the `Intercept` cannot be estimated independently of the `x4*x6` interaction and a bunch of others including four-way though eight-way interactions which were not specified and hence not shown. Similarly, `x1` is confounded with a bunch of interactions, and so on. This is why we want to be estimable the two-way interactions between factors that are combined to create an alternative. We did not want something like `x2*x3`, the client-line extension's price and microwave/stove top interaction to be confounded with say another brand's price.

Blocking the Design

At 36 choice sets, this design is a bit large, so we will block it into two blocks of 18 choice sets. Within each block we will want the shelf talker to be on half the time.

```
%mktblock(data=sasuser.Entree_LinDes3, out=sasuser.Entree_LinDes,
          nblocks=2, seed=448)
```

The first attempt (not shown) produced a design where `x4`, the shelf talker did not occur equally often within each block. Changing the seed took care of the problem. Here are the canonical correlations.

Consumer Food Product Example
Canonical Correlations Between the Factors
There are 0 Canonical Correlations Greater Than 0.316

	Block	x1	x2	x3	x4	x5	x6	x7	x8
Block	1	0.08	0.08	0	0	0.07	0.07	0	0.07
x1	0.08	1	0.17	0.08	0.08	0.16	0.12	0.18	0.16
x2	0.08	0.17	1	0.08	0.08	0.16	0.31	0.27	0.16
x3	0	0.08	0.08	1	0.11	0.12	0.07	0	0.12
x4	0	0.08	0.08	0.11	1	0.12	0.07	0	0.07
x5	0.07	0.16	0.16	0.12	0.12	1	0.13	0.07	0.10
x6	0.07	0.12	0.31	0.07	0.07	0.13	1	0.07	0.19
x7	0	0.18	0.27	0	0	0.07	0.07	1	0.12
x8	0.07	0.16	0.16	0.12	0.07	0.10	0.19	0.12	1

The blocking variable is not highly correlated with any of the factors. Here are some of the frequencies.

Consumer Food Product Example
Summary of Frequencies
There are 0 Canonical Correlations Greater Than 0.316
* - Indicates Unequal Frequencies

Frequencies

	Block	18 18
*	x1	9 8 9 10
*	x2	8 9 10 9
	x3	18 18
	x4	18 18
*	x5	12 11 13
*	x6	11 12 13
	x7	18 18
*	x8	11 12 13
*	Block x1	5 5 4 4 4 4 5 5
*	Block x2	4 5 4 5 4 4 5 5
*	Block x3	8 10 9 9
	Block x4	9 9 9 9
*	Block x1	5 4 4 5 4 4 5 5
*	Block x2	4 4 5 5 4 5 5 4
	Block x3	9 9 9 9
	Block x4	9 9 9 9
*	Block x5	6 6 6 6 5 7
*	Block x6	5 6 7 6 6 6
	Block x7	9 9 9 9
*	Block x8	5 6 7 6 6 6

.
.
.

The blocking variable is perfectly balanced, as it is guaranteed to be if the number of blocks divides the number of runs. Balance within blocks, that is the cross-tabulations of the factors with the blocking variable, looks good. The macro also prints canonical correlations within blocking variables. These can sometimes be quite high, even 1.0, but that is *not* a problem.[‖] Here is the design, as it is printed by the `%MktBlock` macro.

Consumer Food Product Example

Block	Run	x1	x2	x3	x4	x5	x6	x7	x8
1	1	1	3	1	1	1	3	1	1
	2	3	1	2	2	1	3	2	1
	3	2	4	2	2	3	1	1	3
	4	4	3	1	2	2	2	1	1
	5	1	2	2	1	3	3	2	3
	6	4	3	1	1	3	2	2	3
	7	2	3	1	2	1	1	1	3
	8	1	1	2	1	2	2	2	3
	9	4	2	1	2	1	3	2	3
	10	3	1	1	1	1	1	2	3
	11	4	4	2	1	3	2	1	1
	12	4	3	2	2	3	3	1	2
	13	1	4	1	1	2	1	1	2
	14	2	2	1	1	2	3	1	1
	15	1	4	2	2	2	1	2	2
	16	3	4	2	1	1	2	2	2
	17	2	1	1	2	3	2	2	2
	18	3	2	2	2	2	3	1	2

[‖] Ideally, each subject would only make one choice, since the choice model is based on this assumption (which is almost always ignored). As the number of blocks increases, the correlations will mostly go to one, and ultimately be undefined when there is only one choice set per block.

Consumer Food Product Example

Block	Run	x1	x2	x3	x4	x5	x6	x7	x8
2	1	4	2	2	1	1	1	2	2
	2	1	3	2	2	3	2	2	1
	3	1	2	1	2	3	1	2	1
	4	4	1	1	1	2	3	2	1
	5	3	4	1	2	2	3	1	3
	6	2	4	1	1	3	1	2	3
	7	2	2	2	2	1	2	1	3
	8	2	1	2	1	3	3	1	2
	9	3	2	1	1	3	2	1	2
	10	1	4	1	2	1	2	1	2
	11	1	1	1	2	1	3	1	3
	12	3	2	2	1	3	1	1	1
	13	4	3	2	1	1	1	1	2
	14	3	3	2	1	2	2	1	3
	15	4	4	1	2	1	2	2	1
	16	2	3	2	1	2	3	2	1
	17	4	1	2	2	2	1	2	3
	18	3	3	1	2	3	3	2	2

The Final Design

The next steps create the final choice design, stored in `sasuser.Entree_ChDes`, sorted by the blocking and shelf-talker variable. We will use the `%MktLab` macro to assign values, formats, and labels to the design. Previously, we have used the `%MktLab` macro to reassign factor names when we wanted something more descriptive than the default, `x1`, `x2`, and so on, and when we wanted to reassign the names of two m-level factors to minimize the problems associated with correlated factors. This time, we will use the `%MktLab` macro primarily to deal with the asymmetry in the price factors. Recall our factor levels.

Factors and Levels

Alternative	Factor	Levels	Brand	Description
1	X1	4	Client	1.29, 1.69, 2.09, absent
2	X2	4	Client Line Extension	1.39, 1.89, 2.39, absent
	X3	2		microwave/stove-top
	X4	2		shelf-talker yes/no
3	X5	3	Regional	1.99, 2.49, absent
4	X6	3	Private Label	1.49, 2.29, absent
	X7	2		microwave/stove-top
5	X8	3	National	1.99 + 2.39, absent

The choice design will need a quantitative price factor, made from all five of the linear price factors, that contains the prices of each of the alternatives. At this point, our factor `x1` contains 1, 2, 3, 4, and not 1.29, 1.69, 2.09, and absent, which is different from `x2` and from all of the other factors. A 1 in `x1` will need to become a price of 1.29 in the choice design, a 1 in `x2` will need to become a price of 1.39 in the choice design, a 1 in `x3` will need to become a price of 1.99 in the choice design, and so on. Before we use the `%MktRoll` macro to turn the linear design into a choice design, we need to use the `%MktLab` macro to assign the actual prices to the price factors.

The `%MktLab` macro is like the `%MktRoll` macro in the sense that it can use as input a `key=` data set that contains the rules for customizing a design for our particular usage. In the `%MktRoll` macro, the `key=` data set provides the rules for turning a linear design into a choice design. In contrast, in the `%MktLab` macro, the `key=` data set contains the rules for turning a linear design into another linear design, changing one or more of the following: factor names, factor levels, factor types (numeric to character), level formats, and factor labels.

We could use the `%MktLab` macro to change the names of the variables and their types, but we will not do that for this example. Ultimately, we will use the `%MktRoll` macro to assign all of the price factors to a variable called `Price` and similarly provide meaningful names for all of the factors in the choice design, just as we have in previous examples. We could also change a variable like `x3` with values of 1 and 2 to something like `Stove` with values `'Stove'` and `'Micro'`. We will not do that because we want to make a design with a simple list of numeric factors, with simple names like `x1-x8` that we can run through the `%MktRoll` macro to get the final choice design. We will assign formats and labels, so we can print the design in a meaningful way, but ultimately, our only goal at this step is to handle the price asymmetries by assigning the actual price values to the factors.

The `key=` data set contains the rules for customizing our design. The data set has as many rows as the maximum number of levels, in this case four. Each variable is one of the factors in the design, and the values are the factor levels that we want in the final design. The first factor, `x1`, is the price factor for the client brand. Its levels are 1.29, 1.69, and 2.09. In addition, one level is 'not available', which is flagged by the SAS special missing value `.N`. In order to read special missing values in an input data set, you must use the `missing` statement and name the expected missing values. The factor `x2` has the same structure as `x1`, but with different levels. The factor `x3` has two levels, hence the `key=` data set has missing values in the third and fourth row. Since the design has only 1's and 2's for `x3`, this missing values will never be used. Notice that we are keeping `x3` as a numeric variable with values 1 and 2 using a format to supply the character levels 'micro' and 'stove'. The other factors are created in a similar fashion. By default, ordinary missing values `'.'` are not permitted as levels. By default, you may only use ordinary missing values as place holders for factors that have fewer levels than the maximum. If you want missing values in the levels, you must use one of the special missing values `.A`, `.B`, ..., `.Z`, and `._`[**] or the `cfill=` or `nfill=` options.

The `%MktLab` macro specification names the input SAS data set with the design and the key data set. By default, it creates an output SAS data set called `Final`. The data set is sorted by block and shelf talker and printed.

```
proc format;
   value yn    1 = 'No'    2 = 'Talker';
   value micro 1 = 'Micro' 2 = 'Stove';
   run;
```

[**]Note that the `'.'` in `'.N'` is not typed in the data, nor is it typed in the `missing` statement. Furthermore, it does not appear in the printed output. However, you need to type it if you ever refer to a special missing value in code: `if x1 eq .N then ...`.

```
data key;
   missing N;
   input x1-x8;
   format x1 x2 x5 x6 x8 dollar5.2
          x4 yn. x3 x7 micro.;

   label x1 = 'Client Brand'
         x2 = 'Client Line Extension'
         x3 = 'Client Micro/Stove'
         x4 = 'Shelf Talker'

         x5 = 'Regional Brand'
         x6 = 'Private Label'
         x7 = 'Private Micro/Stove'
         x8 = 'National Competitor';

   datalines;
1.29 1.39 1 1 1.99 1.49 1 1.99
1.69 1.89 2 2 2.49 2.29 2 2.39
2.09 2.39 . . N    N    .    N
N    N    . . .    .    .    .
;

%mktlab(data=sasuser.Entree_LinDes, key=key)

proc sort out=sasuser.Entree_LinDesLab(drop=run); by block x4; run;

proc print label; id block x4; by block x4; run;
```

The %MktLab macro prints the variable mapping that it uses, old names followed by new names. In this case, none of the names change, but it is good to make sure that you have the expected correspondence.

```
Variable Mapping:
   x1 : x1
   x2 : x2
   x3 : x3
   x4 : x4
   x5 : x5
   x6 : x6
   x7 : x7
   x8 : x8
```

Here is the design.

Consumer Food Product Example

Block	Shelf Talker	Client Brand	Client Line Extension	Client Micro/ Stove	Regional Brand	Private Label	Private Micro/ Stove	National Competitor
1	No	$1.29	$2.39	Micro	$1.99	N	Micro	$1.99
		$1.29	$1.89	Stove	N	N	Stove	N
		N	$2.39	Micro	N	$2.29	Stove	N
		$1.29	$1.39	Stove	$2.49	$2.29	Stove	N
		$2.09	$1.39	Micro	$1.99	$1.49	Stove	N
		N	N	Stove	N	$2.29	Micro	$1.99
		$1.29	N	Micro	$2.49	$1.49	Micro	$2.39
		$1.69	$1.89	Micro	$2.49	N	Micro	$1.99
		$2.09	N	Stove	$1.99	$2.29	Stove	$2.39
1	Talker	$2.09	$1.39	Stove	$1.99	N	Stove	$1.99
		$1.69	N	Stove	N	$1.49	Micro	N
		N	$2.39	Micro	$2.49	$2.29	Micro	$1.99
		$1.69	$2.39	Micro	$1.99	$1.49	Micro	N
		N	$1.89	Micro	$1.99	N	Stove	N
		N	$2.39	Stove	N	N	Micro	$2.39
		$1.29	N	Stove	$2.49	$1.49	Stove	$2.39
		$1.69	$1.39	Micro	N	$2.29	Stove	$2.39
		$2.09	$1.89	Stove	$2.49	N	Micro	$2.39
2	No	N	$1.89	Stove	$1.99	$1.49	Stove	$2.39
		N	$1.39	Micro	$2.49	N	Stove	$1.99
		$1.69	N	Micro	N	$1.49	Stove	N
		$1.69	$1.39	Stove	N	N	Micro	$2.39
		$2.09	$1.89	Micro	N	$2.29	Micro	$2.39
		$2.09	$1.89	Stove	N	$1.49	Micro	$1.99
		N	$2.39	Stove	$1.99	$1.49	Micro	$2.39
		$2.09	$2.39	Stove	$2.49	$2.29	Micro	N
		$1.69	$2.39	Stove	$2.49	N	Stove	$1.99
2	Talker	$1.29	$2.39	Stove	N	$2.29	Stove	$1.99
		$1.29	$1.89	Micro	N	$1.49	Stove	$1.99
		$2.09	N	Micro	$2.49	N	Micro	N
		$1.69	$1.89	Stove	$1.99	$2.29	Micro	N
		$1.29	N	Micro	$1.99	$2.29	Micro	$2.39
		$1.29	$1.39	Micro	$1.99	N	Micro	N
		N	N	Micro	$1.99	$2.29	Stove	$1.99
		N	$1.39	Stove	$2.49	$1.49	Stove	N
		$2.09	$2.39	Micro	N	N	Stove	$2.39

In contrast, here are the actual values without formats and labels.

```
proc print data=sasuser.Entree_LinDesLab; format _numeric_; run;
```

Consumer Food Product Example

Obs	x1	x2	x3	x4	x5	x6	x7	x8	Block
1	1.29	2.39	1	1	1.99	N	1	1.99	1
2	1.29	1.89	2	1	N	N	2	N	1
3	N	2.39	1	1	N	2.29	2	N	1
4	1.29	1.39	2	1	2.49	2.29	2	N	1
5	2.09	1.39	1	1	1.99	1.49	2	N	1
6	N	N	2	1	N	2.29	1	1.99	1
7	1.29	N	1	1	2.49	1.49	1	2.39	1
8	1.69	1.89	1	1	2.49	N	1	1.99	1
9	2.09	N	2	1	1.99	2.29	2	2.39	1
10	2.09	1.39	2	2	1.99	N	2	1.99	1
11	1.69	N	2	2	N	1.49	1	N	1
12	N	2.39	1	2	2.49	2.29	1	1.99	1
13	1.69	2.39	1	2	1.99	1.49	1	N	1
14	N	1.89	1	2	1.99	N	2	N	1
15	N	2.39	2	2	N	N	1	2.39	1
16	1.29	N	2	2	2.49	1.49	2	2.39	1
17	1.69	1.39	1	2	N	2.29	2	2.39	1
18	2.09	1.89	2	2	2.49	N	1	2.39	1
19	N	1.89	2	1	1.99	1.49	2	2.39	2
20	N	1.39	1	1	2.49	N	2	1.99	2
21	1.69	N	1	1	N	1.49	2	N	2
22	1.69	1.39	2	1	N	N	1	2.39	2
23	2.09	1.89	1	1	N	2.29	1	2.39	2
24	2.09	1.89	2	1	N	1.49	1	1.99	2
25	N	2.39	2	1	1.99	1.49	1	2.39	2
26	2.09	2.39	2	1	2.49	2.29	1	N	2
27	1.69	2.39	2	1	2.49	N	2	1.99	2
28	1.29	2.39	2	2	N	2.29	2	1.99	2
29	1.29	1.89	1	2	N	1.49	2	1.99	2
30	2.09	N	1	2	2.49	N	1	N	2
31	1.69	1.89	2	2	1.99	2.29	1	N	2
32	1.29	N	1	2	1.99	2.29	1	2.39	2
33	1.29	1.39	1	2	1.99	N	1	N	2
34	N	N	1	2	1.99	2.29	2	1.99	2
35	N	1.39	2	2	2.49	1.49	2	N	2
36	2.09	2.39	1	2	N	N	2	2.39	2

One issue remains to be resolved regarding this design, and that concerns the role of the shelf talker when the client line extension is not available. The second part of each block of the design consists of choice sets in which the shelf talker is present and calls attention to the client line extension. However, in some of those choice sets, the client line extension is unavailable. This problem can be handled in several ways. Here are a few:

- Rerun the design creation and evaluation programs excluding all choice sets with shelf talker present and client line extension unavailable. However, this requires changing the model because the excluded cell will make inestimable the interaction between client-line-extension price and shelf talker. Furthermore, the shelf-talker variable will almost certainly no longer be balanced.
- Move the choice sets with client line extension unavailable to the no-shelf-talker block and rerandomize. The shelf talker is then on for all of the last nine choice sets.
- Let the shelf talker go on and off as needed.
- Let the shelf talker call attention to a brand that happens to be out of stock. It is easy to imagine this happening in a real store.

Other options are available as well. No one approach is obviously superior to the alternatives. For this example, we will take the latter approach and allow the shelf talker to be on even when the client line extension is not available. Note that if the shelf talker is turned off when the client line extension is not available then the design must be manually modified to reflect this fact.

Testing the Design Before Data Collection

This is a complicated design that will be used to fit a complicated model with alternative-specific effects, price cross effects, and availability cross effects. Collecting data is time consuming and expensive. It is always good practice, and particularly when there are cross effects, to make sure that the design will work with the most complicated model that we anticipate fitting. Before we collect any data, we will convert the linear design to a choice design* and use the `%ChoicEff` macro to evaluate its efficiency for a multinomial logit model with both price and availability cross effects.

For analysis, the design will have four factors, `Brand`, `Price`, `Micro`, `Shelf`. We will use the `%MktRoll` macro and a `key=` data set (although not the same one as before) to make the choice design. `Brand` is the alternative name; its values are directly read from the `key=Key` in-stream data. `Price` is an attribute whose values will be constructed from the factors `x1`, `x2`, `x5`, `x6`, and `x8` in `sasuser.Entree_LinDesLab` data set. `Micro`, the microwave factor, is constructed from `x3` for the client line extension and `x7` for the private label. `Shelf`, the shelf-talker factor, is created from `x4` for the extension. The `keep=` option on the `%MktRoll` macro is used to keep the original price factors in the design, since we will need them for the price cross effects. Normally, they would be dropped.

*See page 60 for an explanation of linear versus choice designs.

```
data key;
   input Brand $ 1-10 (Price Micro Shelf) ($);
   datalines;
Client     x1 .  .
Extension  x2 x3 x4
Regional   x5 .  .
Private    x6 x7 .
National   x8 .  .
None       .  .  .
;

%mktroll(design=sasuser.Entree_LinDesLab, key=key, alt=brand, out=rolled,
         keep=x1 x2 x5 x6 x8)

proc print data=sasuser.Entree_LinDesLab(obs=2); run;

proc print data=rolled(obs=12);
   format price dollar5.2 shelf yn. micro micro.;
   id set; by set;
   run;
```

Consider the first two choice sets in the linear design.

Consumer Food Product Example

Obs	x1	x2	x3	x4	x5	x6	x7	x8	Block
1	$1.29	$2.39	Micro	No	$1.99	N	Micro	$1.99	1
2	$1.29	$1.89	Stove	No	N	N	Stove	N	1

Here they are in the rolled out choice design.

Consumer Food Product Example

Set	Brand	Price	Micro	Shelf	x1	x2	x5	x6	x8
1	Client	$1.29	.	.	$1.29	$2.39	$1.99	N	$1.99
	Extension	$2.39	Micro	No	$1.29	$2.39	$1.99	N	$1.99
	Regional	$1.99	.	.	$1.29	$2.39	$1.99	N	$1.99
	Private	N	Micro	.	$1.29	$2.39	$1.99	N	$1.99
	National	$1.99	.	.	$1.29	$2.39	$1.99	N	$1.99
	None	.	.	.	$1.29	$2.39	$1.99	N	$1.99

2	Client	$1.29	.	.	$1.29	$1.89	N	N	N
	Extension	$1.89	Stove	No	$1.29	$1.89	N	N	N
	Regional	N	.	.	$1.29	$1.89	N	N	N
	Private	N	Stove	.	$1.29	$1.89	N	N	N
	National	N	.	.	$1.29	$1.89	N	N	N
	None	.	.	.	$1.29	$1.89	N	N	N

Set 1, Alternative 1

`Brand`	=	'Client'	the brand for this alternative
`Price`	=	`x1` = $1.29	the price of this alternative
`Micro`	=	.	does not apply to this brand
`Shelf`	=	.	does not apply to this brand
`x1`	=	$1.29	the price of the client brand in this choice set
`x2`	=	$2.39	the price of the extension in this choice set
`x5`	=	$1.99	the price of the regional competitor in this choice set
`x6`	=	N	the private label unavailable in this choice set
`x8`	=	$1.99	national competitor unavailable in this choice set

Set 1, Alternative 2

`Brand`	=	'Extension'	the brand for this alternative
`Price`	=	`x2` = $2.39	the price of this alternative
`Micro`	=	Micro	Microwave version
`Shelf`	=	No	Shelf Talker, No
`x1`	=	$1.29	the price of the client brand in this choice set
`x2`	=	$2.39	the price of the extension in this choice set
`x5`	=	$1.99	the price of the regional competitor in this choice set
`x6`	=	N	the private label unavailable in this choice set
`x8`	=	$1.99	national competitor unavailable in this choice set

The factors `x1` through `x8` will be used to make the price cross effects. Notice that `x1` through `x8` are constant within each choice set. The variable `x1` is the price of alternative one, which is the same no matter which alternative it is stored with. The factors `x1` through `x8` will also be used to make five other factors that will be used to make the availability cross effects. Here is how the prices will be recoded for those factors.

x1	→	a1	x2	→	a2	x5	→	a5	x6	→	a6	x8	→	a8
1.29		1	1.39		1	1.99		1	1.49		1	1.99		1
1.69		1	1.89		1	2.49		1	2.29		1	2.39		1
2.09		1	2.39		1	N		-2	N		-2	N		-2
N		-3	N		-3									

This is a contrast coding. Within each factor, the coding sums to zero. Each availability factor has a coding that contrasts unavailable with the remaining available prices. When an alternative is unavailable, the `a` variable is set to minus the number of available price points. The coding for available alternatives is 1. A -3 is used for the first two alternatives that have three prices, and a -2 is used for the remaining alternatives that have two prices.

We need to do a few more things to this design before we are ready to use it. We need to convert the missings for when `Micro` and `Shelf` do not apply to 2 for 'Stove' and 1 for 'No'. We need to do the contrast coding for making the availability cross effects. More will be said about this after the code is shown. Since we will be treating all of the price factors as a quantitative (not as `class` variables), we need to convert the missing prices to zero. Eventually, we will also need to output just the alternatives that are available (those with a nonzero price and also the none alternative). For now, we will just make a variable `w` that flags the available alternatives (`w` = 1). We can do this using a weight or flag variable: `w` = 1 means available and `w` = 0 means not available. We also need to assign labels and formats.

```
data sasuser.Entree_ChDes(drop=i);
   set rolled;
   array x[6] price x1 -- x8;
   array a[5] a1 a2 a5 a6 a8;
   if nmiss(micro) then micro = 2; /* stove if not a factor in alt      */
   if nmiss(shelf) then shelf = 1; /* not talker if not a factor in alt */
   a1 = -3 * nmiss(x1) + n(x1);    /* alt1: -3 - not avail, 1 - avail */
   a2 = -3 * nmiss(x2) + n(x2);    /* alt2: -3 - not avail, 1 - avail */
   a5 = -2 * nmiss(x5) + n(x5);    /* alt3: -2 - not avail, 1 - avail */
   a6 = -2 * nmiss(x6) + n(x6);    /* alt4: -2 - not avail, 1 - avail */
   a8 = -2 * nmiss(x8) + n(x8);    /* alt5: -2 - not avail, 1 - avail */
   i  = mod(_n_ - 1, 6) + 1;       /* alternative number              */
   if i le 5 then a[i] = 0;        /* 0 effect of an alt on itself    */
   do i = 1 to 6; if nmiss(x[i]) then x[i] = 0; end; /* missing price -> 0    */
   w = brand eq 'None' or price ne 0;                /* 1 - avail, 0 not avail*/
   format price dollar5.2 shelf yn. micro micro.;
   label x1 = 'CE, Client'         a1 = 'AE, Client'
         x2 = 'CE, Extension'      a2 = 'AE, Extension'
         x5 = 'CE, Regional'       a5 = 'AE, Regional'
         x6 = 'CE, Private'        a6 = 'AE, Private'
         x8 = 'CE, National'       a8 = 'AE, National';
   run;

proc print data=sasuser.Entree_ChDes(obs=18); by set; id set; run;
```

The statements in the middle of the data step, from `a1 = ...` through `if i ...` create the variables that will be used to make the availability effects. When alternative 1 is unavailable (`x1` is missing), `a1` is set to -3, otherwise `a1` is set to 1; when alternative 2 is unavailable (`x2` is missing), `a2` is set to -3, otherwise `a2` is set to 1; when alternative 3 is unavailable (`x5` is missing), `a5` is set to -3, otherwise `a5` is set to 1; and so on. Each of these statements could have been written in `if else` form. Here for example is the first assignment statement rewritten: `if nmiss(x1) then a1 = -3; else a1 = 1;`. The variables `x1, x2, x5, x6, and x8` are the five price factors, and the "`a`" factors use the same numbering scheme, although this is not a requirement. The `if i` and `i =` statements then set the variable to zero when the variable will be used to construct the effect of an alternative on itself. For example, the first alternative is the client brand, so `a1` in the first alternative is set to zero. Here are the first three choice sets.

Consumer Food Product Example

Set	Brand	Price	Micro	Shelf	x1	x2	x5	x6	x8	a1	a2	a5	a6	a8	w
1	Client	$1.29	Stove	No	$1.29	$2.39	$1.99	$0.00	$1.99	0	1	1	-2	1	1
	Extension	$2.39	Micro	No	$1.29	$2.39	$1.99	$0.00	$1.99	1	0	1	-2	1	1
	Regional	$1.99	Stove	No	$1.29	$2.39	$1.99	$0.00	$1.99	1	1	0	-2	1	1
	Private	$0.00	Micro	No	$1.29	$2.39	$1.99	$0.00	$1.99	1	1	1	0	1	0
	National	$1.99	Stove	No	$1.29	$2.39	$1.99	$0.00	$1.99	1	1	1	-2	0	1
	None	$0.00	Stove	No	$1.29	$2.39	$1.99	$0.00	$1.99	1	1	1	-2	1	1
2	Client	$1.29	Stove	No	$1.29	$1.89	$0.00	$0.00	$0.00	0	1	-2	-2	-2	1
	Extension	$1.89	Stove	No	$1.29	$1.89	$0.00	$0.00	$0.00	1	0	-2	-2	-2	1
	Regional	$0.00	Stove	No	$1.29	$1.89	$0.00	$0.00	$0.00	1	1	0	-2	-2	0
	Private	$0.00	Stove	No	$1.29	$1.89	$0.00	$0.00	$0.00	1	1	-2	0	-2	0
	National	$0.00	Stove	No	$1.29	$1.89	$0.00	$0.00	$0.00	1	1	-2	-2	0	0
	None	$0.00	Stove	No	$1.29	$1.89	$0.00	$0.00	$0.00	1	1	-2	-2	-2	1
3	Client	$0.00	Stove	No	$0.00	$2.39	$0.00	$2.29	$0.00	0	1	-2	1	-2	0
	Extension	$2.39	Micro	No	$0.00	$2.39	$0.00	$2.29	$0.00	-3	0	-2	1	-2	1
	Regional	$0.00	Stove	No	$0.00	$2.39	$0.00	$2.29	$0.00	-3	1	0	1	-2	0
	Private	$2.29	Stove	No	$0.00	$2.39	$0.00	$2.29	$0.00	-3	1	-2	0	-2	1
	National	$0.00	Stove	No	$0.00	$2.39	$0.00	$2.29	$0.00	-3	1	-2	1	0	0
	None	$0.00	Stove	No	$0.00	$2.39	$0.00	$2.29	$0.00	-3	1	-2	1	-2	1

In the first choice set, for example, since alternative 1 is available, `a1` is 1, for all alternatives except the first, where `a1` is 0. Also in the first choice set, since alternative 4 is not available and there are two price levels, `a6` is -2 for all alternatives except the fourth, where `a6` is 0. In the third choice set, since alternative 1 is not available and there are three price levels, `a1` is -3, for all alternatives except the first, where `a2` is 0. In general, the coding stores a zero in the *ith* effect for the *ith* alternative, otherwise a 1 if the alternative is available, otherwise -(the number of price levels) if the alternative is unavailable.

Now our choice design is done except for the final coding for the analysis. We can now use the `%ChoicEff` macro to evaluate our choice design. Here is some sample code, omitting for now the details of the model (indicated by `model= ...`). The complicated part of this is the model due to the alternative-specific price effects and the cross effects. For now, let's concentrate on everything else.

```
%choiceff(data=sasuser.Entree_ChDes,
          model= ..., /* model specification skipped for now */
          nsets=36, nalts=6, weight=w,
          beta=zero, init=sasuser.Entree_ChDes(keep=set),
          intiter=0)
```

The way you check the efficiency of a design like this is to first name it on the `data=` option. This will be the candidate set that contains all of the choice sets that we will consider. In addition, the same design is named on the `init=` option. The full specification is `init=sasuser.Entree_ChDes(keep=set)`. Just the variable `Set` is kept. It will be used to bring in just the indicated choice sets from the `data=` design, which in this case is all of them. The option `nsets=36` specifies the number of choice sets, and `nalts=6` specifies the number of alternatives. This macro requires a constant number of alternatives in each

choice set for ease of data management. However, not all of the alternatives have to be used. In this case, we have an availability study. We need to keep the unavailable alternatives in the design for this step, but we do not want them to contribute to the analysis, so we specify a weight variable with `weight=w` and flag the available alternatives with `w=1` and the unavailable alternatives with `w=0`. The option `beta=zero` specifies that we are assuming for design evaluation purpose all zero betas. We can specify other values and get other results for the variances and standard errors. Finally, we specify `intiter=0` which specifies zero internal iterations. We use zero internal iterations when we want to evaluate an initial design, but not attempt to improve it. Here is the actual specification we will use, complete with the model specification.

```
%choiceff(data=sasuser.Entree_ChDes,
          model=class(brand / zero='None')
                class(brand / zero='None' separators='' ' ') *
                   identity(price)
                class(shelf micro / lprefix=5 0 zero='No' 'Stove')
                identity(x1 x2 x5 x6 x8) *
                   class(brand / zero='None' separators=' ' ' on ')
                identity(a1 a2 a5 a6 a8) *
                   class(brand / zero='None' separators=' ' ' on ') /
                lprefix=0 order=data,
          nsets=36, nalts=6, weight=w,
          beta=zero, init=sasuser.Entree_ChDes(keep=set),
          intiter=0)
```

The specification `class(brand / zero='None')` specifies the brand effects. This specification will create indicator variables for brand with the constant alternative being the reference brand. The option `zero='None'` ensures that the reference level will be `'None'` instead of the default last sorted level (`'Regional'`). Indicator variables will be created for the brands Client, Extension, Regional, Private, and National, but not None. The `zero='None'` option, like `zero='Home'` and other `zero='literal-string'` options we have used in previous examples, names the actual formatted value of the `class` variable that should be excluded from the coded variables because the coefficient will be zero. Do not confuse `zero=none` and `zero='None'`. The `zero=none` option specifies that you want all indicator variables to be created, even including one for the last level. In contrast, the option `zero='None'` (or `zero=` any quoted string) names a specific formatted value, in this case `'None'`, for which indicator variables are not to be created.

The specification `class(brand / ...) * identity(price)` creates the alternative-specific price effects. They are specified as an interaction between a categorical variable `Brand` and a quantitative factor `Price`. The `separators='' ' '` option in the `class` specification specifies the separators that are used to construct the labels for the main effect and interaction terms. The main-effects separator, which is the first `separators=` value, `''`, is ignored since `lprefix=0`. Specifying `' '` as the second value creates labels of the form *brand-blank-price* instead of the default *brand-blank-asterisk-blank-price*.

The specification `class(shelf micro / ...)` names the shelf-talker and microwave variables as categorical variables and creates indicator variables for the `'Talker'` category, not the `'No'` category and the `'Micro'` category not the `'Stove'` category. In `zero='No' 'Stove'`, the `'No'` applies to the first variable, `Shelf` and the second value, `'Stove'`, applies to second variable, `Micro`.

The specification `identity(x1 x2 x5 x6 x8) * class(brand / ...)` creates the linear price cross effects. The `separators=` option is specified with a second value of `' on '` to create cross effect labels like `'Client on Extension'`. The specification `identity(a1 a2 a5 a6 a8) * class(brand / ...)` creates the availability cross effects. Note that the order of the transformation specification is

important. Make sure you specify `identity` followed by `class` in order to get the right labels. More will be said on the cross effects when we look at the actual coded values in the next few pages.

Note that PROC TRANSREG produces the following warning.

```
WARNING: This usage of * sets one group's slope to zero.  Specify |
         to allow all slopes and intercepts to vary.  Alternatively,
         specify CLASS(vars) * identity(vars) identity(vars) for
         separate within group functions and a common intercept.
         This is a change from Version 6.
```

This is because `class` was interacted with `identity` using the asterisk instead of the vertical bar. In a linear model, this may be a sign of a coding error, so the procedure prints a warning. If you get this warning while coding a choice model specifying `zero=`*'constant-alternative-level'*, you can safely ignore it. Still, it is always good to print out one or more coded choice sets to check the coding as we will do later. Here is the last part of the output from the `%ChoicEff` macro.

Consumer Food Product Example

n	Variable Name	Label	Variance	DF	Standard Error
1	BrandClient	Client	69.807	1	8.3551
2	BrandExtension	Extension	75.688	1	8.6999
3	BrandRegional	Regional	121.147	1	11.0067
4	BrandPrivate	Private	104.058	1	10.2009
5	BrandNational	National	110.456	1	10.5098
6	BrandClientPrice	Client Price	3.255	1	1.8042
7	BrandExtensionPrice	Extension Price	2.233	1	1.4942
8	BrandRegionalPrice	Regional Price	6.599	1	2.5688
9	BrandPrivatePrice	Private Price	2.604	1	1.6138
10	BrandNationalPrice	National Price	11.071	1	3.3273
11	ShelfTalker	Shelf Talker	0.928	1	0.9636
12	MicroMicro	Micro	0.562	1	0.7493
13	x1BrandClient	CE, Client on Client	.	0	.
14	x1BrandExtension	CE, Client on Extension	4.689	1	2.1655
15	x1BrandRegional	CE, Client on Regional	4.462	1	2.1124
16	x1BrandPrivate	CE, Client on Private	5.627	1	2.3720
17	x1BrandNational	CE, Client on National	5.374	1	2.3182
18	x2BrandClient	CE, Extension on Client	3.040	1	1.7435
19	x2BrandExtension	CE, Extension on Extension	.	0	.
20	x2BrandRegional	CE, Extension on Regional	3.038	1	1.7431
21	x2BrandPrivate	CE, Extension on Private	3.666	1	1.9146
22	x2BrandNational	CE, Extension on National	3.130	1	1.7691
23	x5BrandClient	CE, Regional on Client	8.961	1	2.9935
24	x5BrandExtension	CE, Regional on Extension	9.824	1	3.1343
25	x5BrandRegional	CE, Regional on Regional	.	0	.
26	x5BrandPrivate	CE, Regional on Private	10.496	1	3.2398
27	x5BrandNational	CE, Regional on National	10.360	1	3.2188

28	x6BrandClient	CE, Private on Client	3.965	1	1.9912
29	x6BrandExtension	CE, Private on Extension	4.195	1	2.0482
30	x6BrandRegional	CE, Private on Regional	4.429	1	2.1046
31	x6BrandPrivate	CE, Private on Private	.	0	.
32	x6BrandNational	CE, Private on National	4.453	1	2.1102
33	x8BrandClient	CE, National on Client	18.098	1	4.2541
34	x8BrandExtension	CE, National on Extension	16.311	1	4.0387
35	x8BrandRegional	CE, National on Regional	22.372	1	4.7299
36	x8BrandPrivate	CE, National on Private	18.271	1	4.2745
37	x8BrandNational	CE, National on National	.	0	.
38	a1BrandClient	AE, Client on Client	.	0	.
39	a1BrandExtension	AE, Client on Extension	0.981	1	0.9904
40	a1BrandRegional	AE, Client on Regional	0.892	1	0.9447
41	a1BrandPrivate	AE, Client on Private	1.071	1	1.0347
42	a1BrandNational	AE, Client on National	1.031	1	1.0155
43	a2BrandClient	AE, Extension on Client	0.766	1	0.8755
44	a2BrandExtension	AE, Extension on Extension	.	0	.
45	a2BrandRegional	AE, Extension on Regional	0.880	1	0.9381
46	a2BrandPrivate	AE, Extension on Private	0.990	1	0.9952
47	a2BrandNational	AE, Extension on National	0.999	1	0.9995
48	a5BrandClient	AE, Regional on Client	5.128	1	2.2644
49	a5BrandExtension	AE, Regional on Extension	5.530	1	2.3516
50	a5BrandRegional	AE, Regional on Regional	.	0	.
51	a5BrandPrivate	AE, Regional on Private	5.860	1	2.4208
52	a5BrandNational	AE, Regional on National	5.887	1	2.4263
53	a6BrandClient	AE, Private on Client	1.796	1	1.3402
54	a6BrandExtension	AE, Private on Extension	1.843	1	1.3577
55	a6BrandRegional	AE, Private on Regional	2.116	1	1.4547
56	a6BrandPrivate	AE, Private on Private	.	0	.
57	a6BrandNational	AE, Private on National	1.964	1	1.4015
58	a8BrandClient	AE, National on Client	10.135	1	3.1836
59	a8BrandExtension	AE, National on Extension	8.720	1	2.9529
60	a8BrandRegional	AE, National on Regional	12.021	1	3.4671
61	a8BrandPrivate	AE, National on Private	10.188	1	3.1919
62	a8BrandNational	AE, National on National	.	0	.
				==	
				52	

First we see estimable brand effects for each of the five brands, excluding the constant alternative `'None'`. Next, we see quantitative alternative-specific price effects for each of the brands. The next two effects that are single *df* effects for the shelf-talker and the microwave option. Then we see five sets of linear price cross effects (those whose label begins with "CE"), each consisting of four effects of a brand on another brand, plus one more zero *df* cross effect of a brand on itself. The zero *df* and missing variances and standard errors are correct since the cross effect of an alternative on itself is perfectly aliased with its alternative-specific price effect. After that we see five sets of availability cross effects (those whose label begins with "AE"), each consisting of four effects of a brand on another brand, plus one more zero *df* cross effect of a brand on itself. The zero *df* and missing variances and standard errors are correct since the cross effect of an alternative on itself is zero. These results look fine. Everything

that should be estimable is estimable, and everything that should not be estimable is not.

Next, we will run some further checks by looking at the coded design. Before we look at the coded design, recall that the design for the first five choice sets is as follows.

Consumer Food Product Example

Block	Shelf Talker	Client Brand	Client Line Extension	Client Micro/ Stove	Regional Brand	Private Label	Private Micro/ Stove	National Competitor
1	No	$1.29	$2.39	Micro	$1.99	N	Micro	$1.99
		$1.29	$1.89	Stove	N	N	Stove	N
		N	$2.39	Micro	N	$2.29	Stove	N
		$1.29	$1.39	Stove	$2.49	$2.29	Stove	N
		$2.09	$1.39	Micro	$1.99	$1.49	Stove	N

The coded design that the `%ChoiceEff` macro creates is called TMP_CAND. We will look at the coded data set in several ways. First, here are the `Brand`, `Price`, microwave and shelf-talker factors, for just the available alternatives for the first five choice sets.

```
proc print data=tmp_cand(obs=24) label;
   var Brand Price Shelf Micro;
   where w;
   run;
```

Consumer Food Product Example

Obs	Brand	Price	Shelf	Micro
1	Client	$1.29	No	Stove
2	Extension	$2.39	No	Micro
3	Regional	$1.99	No	Stove
5	National	$1.99	No	Stove
6	None	$0.00	No	Stove
7	Client	$1.29	No	Stove
8	Extension	$1.89	No	Stove
12	None	$0.00	No	Stove
14	Extension	$2.39	No	Micro
16	Private	$2.29	No	Stove
18	None	$0.00	No	Stove
19	Client	$1.29	No	Stove
20	Extension	$1.39	No	Stove
21	Regional	$2.49	No	Stove
22	Private	$2.29	No	Stove
24	None	$0.00	No	Stove

25	Client	$2.09	No	Stove
26	Extension	$1.39	No	Micro
27	Regional	$1.99	No	Stove
28	Private	$1.49	No	Stove
30	None	$0.00	No	Stove
34	Private	$2.29	No	Micro
35	National	$1.99	No	Stove
36	None	$0.00	No	Stove

Unlike all previous examples, the number of alternatives is not the same in all of the choice sets due to differing subsets of brands being unavailable in each choice set.

Here are the coded factors for the brand effects and alternative-specific price effects for the first choice set.

```
proc print data=tmp_cand(obs=5) label;
   id Brand;
   var BrandClient -- BrandNational;
   where w;
   run;
proc format; value zer 0 = '  0'; run;

proc print data=tmp_cand(obs=5) label;
   id Brand Price;
   var BrandClientPrice -- BrandNationalPrice;
   format BrandClientPrice -- BrandNationalPrice zer5.2;
   where w;
   run;
```

Consumer Food Product Example

Brand	Client	Extension	Regional	Private	National
Client	1	0	0	0	0
Extension	0	1	0	0	0
Regional	0	0	1	0	0
National	0	0	0	0	1
None	0	0	0	0	0

Consumer Food Product Example

Brand	Price	Client Price	Extension Price	Regional Price	Private Price	National Price
Client	$1.29	1.29	0	0	0	0
Extension	$2.39	0	2.39	0	0	0
Regional	$1.99	0	0	1.99	0	0
National	$1.99	0	0	0	0	1.99
None	$0.00	0	0	0	0	0

The brand effects and alternative-specific price effect codings are similar to those we have used previously. The difference is the presence of all zero columns for unavailable alternatives, in this case the private label and national brands. Note that `Brand Price` are just ID variables and do not enter into the analysis.

Here are the shelf-talker and microwave coded factors (along with the `Brand`, `Price`, `Shelf`, and `Micro` factors).

```
proc print data=tmp_cand(obs=5) label;
   id Brand Price Shelf Micro;
   var shelftalker micromicro;
   where w;
   run;
```

Consumer Food Product Example

Brand	Price	Shelf	Micro	Shelf Talker	Micro
Client	$1.29	No	Stove	0	0
Extension	$2.39	No	Micro	0	1
Regional	$1.99	No	Stove	0	0
National	$1.99	No	Stove	0	0
None	$0.00	No	Stove	0	0

The following code prints the price cross effects along with `Brand` and `Price` for the first choice set.

```
proc print data=tmp_cand(obs=4) label;
   id Brand Price;
   var x1Brand:; format x1Brand: zer5.2;
   where w;
   run;
proc print data=tmp_cand(obs=4) label;
   id Brand Price;
   var x2Brand:; format x2Brand: zer5.2;
   where w;
   run;
```

```
proc print data=tmp_cand(obs=4) label;
   id Brand Price;
   var x5Brand:; format x5Brand: zer5.2;
   where w;
   run;
proc print data=tmp_cand(obs=4) label;
   id Brand Price;
   var x6Brand:; format x6Brand: zer5.2;
   where w;
   run;
proc print data=tmp_cand(obs=4) label;
   id Brand Price;
   var x8Brand:; format x8Brand: zer5.2;
   where w;
   run;
```

The cross effects are printed in panels. This first panel shows the terms that capture the effect of the client brand on the utility of the other brands. The second panel shows the terms that capture the effect of the line extension on the other alternatives, and so on. An unavailable brand has no effect on any other brand's utility in that choice set.

Consumer Food Product Example

Brand	Price	CE, Client on Client	CE, Client on Extension	CE, Client on Regional	CE, Client on Private	CE, Client on National
Client	$1.29	1.29	0	0	0	0
Extension	$2.39	0	1.29	0	0	0
Regional	$1.99	0	0	1.29	0	0
National	$1.99	0	0	0	0	1.29

Brand	Price	CE, Extension on Client	CE, Extension on Extension	CE, Extension on Regional	CE, Extension on Private	CE, Extension on National
Client	$1.29	2.39	0	0	0	0
Extension	$2.39	0	2.39	0	0	0
Regional	$1.99	0	0	2.39	0	0
National	$1.99	0	0	0	0	2.39

Brand	Price	CE, Regional on Client	CE, Regional on Extension	CE, Regional on Regional	CE, Regional on Private	CE, Regional on National
Client	$1.29	1.99	0	0	0	0
Extension	$2.39	0	1.99	0	0	0
Regional	$1.99	0	0	1.99	0	0
National	$1.99	0	0	0	0	1.99

Brand	Price	CE, Private on Client	CE, Private on Extension	CE, Private on Regional	CE, Private on Private	CE, Private on National
Client	$1.29	0	0	0	0	0
Extension	$2.39	0	0	0	0	0
Regional	$1.99	0	0	0	0	0
National	$1.99	0	0	0	0	0

Brand	Price	CE, National on Client	CE, National on Extension	CE, National on Regional	CE, National on Private	CE, National on National
Client	$1.29	1.99	0	0	0	0
Extension	$2.39	0	1.99	0	0	0
Regional	$1.99	0	0	1.99	0	0
National	$1.99	0	0	0	0	1.99

A column like `'CE, Client on Extension'` in the first panel, for example, captures the effect of the client brand at $1.29 on the utility of the extension. In the next panel, `'CE, Extension on Client'` captures the effect of the extension at $2.39 on the utility of the client brand.

The following steps prints the availability cross effects along with `Brand` and `Price` for the first choice set.

```
proc print data=tmp_cand(obs=4) label;
   id Brand Price;
   var a1Brand:;
   where w;
   run;

proc print data=tmp_cand(obs=4) label;
   id Brand Price;
   var a2Brand:;
   where w;
   run;
```

```
proc print data=tmp_cand(obs=4) label;
   id Brand Price;
   var a5Brand:;
   where w;
   run;
proc print data=tmp_cand(obs=4) label;
   id Brand Price;
   var a6Brand:;
   where w;
   run;
proc print data=tmp_cand(obs=4) label;
   id Brand Price;
   var a8Brand:;
   where w;
   run;
```

The availability cross effects are printed in panels. The first panel shows the terms that capture the effect of the client brand which on the other available alternatives, and so on. Panels with 1's in them show the effects of the available brands and panels with negative numbers show the effects of the unavailable brands.

Consumer Food Product Example

Brand	Price	AE, Client on Client	AE, Client on Extension	AE, Client on Regional	AE, Client on Private	AE, Client on National
Client	$1.29	0	0	0	0	0
Extension	$2.39	0	1	0	0	0
Regional	$1.99	0	0	1	0	0
National	$1.99	0	0	0	0	1

Brand	Price	AE, Extension on Client	AE, Extension on Extension	AE, Extension on Regional	AE, Extension on Private	AE, Extension on National
Client	$1.29	1	0	0	0	0
Extension	$2.39	0	0	0	0	0
Regional	$1.99	0	0	1	0	0
National	$1.99	0	0	0	0	1

Brand	Price	AE, Regional on Client	AE, Regional on Extension	AE, Regional on Regional	AE, Regional on Private	AE, Regional on National
Client	$1.29	1	0	0	0	0
Extension	$2.39	0	1	0	0	0
Regional	$1.99	0	0	0	0	0
National	$1.99	0	0	0	0	1

Brand	Price	AE, Private on Client	AE, Private on Extension	AE, Private on Regional	AE, Private on Private	AE, Private on National
Client	$1.29	-2	0	0	0	0
Extension	$2.39	0	-2	0	0	0
Regional	$1.99	0	0	-2	0	0
National	$1.99	0	0	0	0	-2

Brand	Price	AE, National on Client	AE, National on Extension	AE, National on Regional	AE, National on Private	AE, National on National
Client	$1.29	1	0	0	0	0
Extension	$2.39	0	1	0	0	0
Regional	$1.99	0	0	1	0	0
National	$1.99	0	0	0	0	0

The design looks good, it has reasonably good balance and correlations, it can be used to estimate all of the effects of interest, and we have shown that we know how to code all of the factors for a model with cross effects and availability cross effects. We are ready to collect data.

Generating Artificial Data

We will not illustrate questionnaire generation for this example since we have done it several times before in previous examples. Instead we will go straight to data processing and analysis. This DATA step generates some artificial data. Creating artificial data and trying the analysis before collecting real data is another way to test the design before going to the expense of data collection.

```
%let m   = 6;
%let mm1 = %eval(&m - 1);
%let n   = 36;
proc format;
   value yn    1 = 'No'    2 = 'Talker';
   value micro 1 = 'Micro' 2 = 'Stove';
   run;
```

```
data _null_;
   array brands[&m] _temporary_ (5 7 1 2 3 -2);
   array u[&m];
   array x[&mm1] x1 x2 x5 x6 x8;
   do rep = 1 to 300;
      if mod(rep, 2) then put;
      put rep 3. +2 @@;
      do j = 1 to &n;
         set sasuser.Entree_LinDesLab point=j;
         do brand = 1 to &m; u[brand] = brands[brand] + 2 * normal(17); end;
         do brand = 1 to &mm1;
            if n(x[brand]) then u[brand] + -x[brand]; else u[brand] = .;
            end;
         if n(u2) and x4 = 2 then u2 + 1; /* shelf talker */
         if n(u2) and x3 = 1 then u2 + 1; /* microwave    */
         if n(u4) and x7 = 1 then u4 + 1; /* microwave    */
         * Choose the most preferred alternative.;
         m = max(of u1-u&m);
         do brand = 1 to &m;
            if n(u[brand]) then if abs(u[brand] - m) < 1e-4 then c = brand;
            end;
         put +(-1) c @@;
         end;
      end;
   stop;
   run;
```

This DATA step reads the data.

```
data results;
   input Subj (choose1-choose&n) (1.) @@;
   datalines;
  1 22222415521222252222122122222121252 2 22222542124222212222121222222121322
  3 21222452154522212222122222121121252 4 21212512121222212222122242122121252
  5 22222512521222212252121222222121242 6 22212552311222212222422121211242522
  .
  .
  .
297 22222552121222242222422252212115162 298 22222412124212242222122251222121252
299 11222512121212212152122221221121262 300 12222512324212212222121122212321232
;
```

Processing the Data

The analysis proceeds in a fashion similar to before. We have already made the choice design, so we just have to merge it with the data. The data and design are merged in the usual way using the `%MktMerge` macro. Notice at this point that the unavailable alternatives are still in the design. The `%MktMerge` macro has an `nalts=` option and expects a constant number of alternatives in each choice set.

```
%mktmerge(design=sasuser.Entree_ChDes, data=results, out=res2,
          nsets=&n, nalts=&m, setvars=choose1-choose&n)

proc print data=res2(obs=12); id subj set; by subj set; run;
```

Here are the data and design for the first two choice sets for the first subject, including the unavailable alternatives.

Consumer Food Product Example

Subj	Set	Brand	Price	Shelf	Micro	x1	x2	x5	x6	x8	a1	a2	a5	a6	a8	w	c
1	1	Client	$1.29	Stove	No	$1.29	$2.39	$1.99	$0.00	$1.99	0	1	1	-2	1	1	2
		Extension	$2.39	Micro	No	$1.29	$2.39	$1.99	$0.00	$1.99	1	0	1	-2	1	1	1
		Regional	$1.99	Stove	No	$1.29	$2.39	$1.99	$0.00	$1.99	1	1	0	-2	1	1	2
		Private	$0.00	Micro	No	$1.29	$2.39	$1.99	$0.00	$1.99	1	1	1	0	1	0	2
		National	$1.99	Stove	No	$1.29	$2.39	$1.99	$0.00	$1.99	1	1	1	-2	0	1	2
		None	$0.00	Stove	No	$1.29	$2.39	$1.99	$0.00	$1.99	1	1	1	-2	1	1	2
1	2	Client	$1.29	Stove	No	$1.29	$1.89	$0.00	$0.00	$0.00	0	1	-2	-2	-2	1	2
		Extension	$1.89	Stove	No	$1.29	$1.89	$0.00	$0.00	$0.00	1	0	-2	-2	-2	1	1
		Regional	$0.00	Stove	No	$1.29	$1.89	$0.00	$0.00	$0.00	1	1	0	-2	-2	0	2
		Private	$0.00	Stove	No	$1.29	$1.89	$0.00	$0.00	$0.00	1	1	-2	0	-2	0	2
		National	$0.00	Stove	No	$1.29	$1.89	$0.00	$0.00	$0.00	1	1	-2	-2	0	0	2
		None	$0.00	Stove	No	$1.29	$1.89	$0.00	$0.00	$0.00	1	1	-2	-2	-2	1	2

These next steps aggregate the data. The data set is fairly large at 64,800 observations, and aggregating greatly reduces its size, which makes both the TRANSREG and the PHREG steps run in just a few seconds. This step also excludes the unavailable alternatives. When `w` is 1 (true) the alternative is available and counted, otherwise when `w` is 0 (false) the alternative is unavailable and excluded by the `where` clause and not counted. There is nothing in subsequent steps that assumes a fixed number of alternatives.

```
proc summary data=res2 nway;
   class set brand price shelf micro x1 x2 x5 x6 x8 a1 a2 a5 a6 a8 c;
   output out=agg(drop=_type_);
   where w; /* exclude unavailable, w = 0 */
   run;

proc print; where set = 1; run;
```

All of the variables used in the analysis are named as `class` variables in PROC SUMMARY, which reduces the data set from 64,800 observations to 286. Here are the aggregated data for the first choice set.

Consumer Food Product Example

Obs	Set	Brand	Price	Shelf	Micro	x1	x2	x5	x6	x8	a1	a2	a5	a6	a8	c	_FREQ_
1	1	Client	$1.29	No	Stove	$1.29	$2.39	$1.99	$0.00	$1.99	0	1	1	-2	1	1	74
2	1	Client	$1.29	No	Stove	$1.29	$2.39	$1.99	$0.00	$1.99	0	1	1	-2	1	2	226
3	1	Extension	$2.39	No	Micro	$1.29	$2.39	$1.99	$0.00	$1.99	1	0	1	-2	1	1	220
4	1	Extension	$2.39	No	Micro	$1.29	$2.39	$1.99	$0.00	$1.99	1	0	1	-2	1	2	80
5	1	National	$1.99	No	Stove	$1.29	$2.39	$1.99	$0.00	$1.99	1	1	1	-2	0	1	6
6	1	National	$1.99	No	Stove	$1.29	$2.39	$1.99	$0.00	$1.99	1	1	1	-2	0	2	294
7	1	None	$0.00	No	Stove	$1.29	$2.39	$1.99	$0.00	$1.99	1	1	1	-2	1	2	300
8	1	Regional	$1.99	No	Stove	$1.29	$2.39	$1.99	$0.00	$1.99	1	1	0	-2	1	2	300

In the first choice set, the client brand was chosen (`c` = 1) a total of `_freq_` = 74 times and not chosen (`c` = 2) a total of `_freq_` = 226 times. Each alternative was chosen and not chosen a total of 300 times, which is the number of subjects. These next steps code and run the analysis.

Cross Effects

This next step codes the design for analysis. This coding was discussed on page 321. PROC TRANSREG is run like before, except now the data set `Agg` is specified and the ID variable includes `_freq_` (the frequency variable) but not `Subj` (the subject number variable).

```
proc transreg data=agg design=5000 nozeroconstant norestoremissing;
   model class(brand / zero='None')
         class(brand / zero='None' separators='' ' ') * identity(price)
         class(shelf micro / lprefix=5 0 zero='No' 'Stove')
         identity(x1 x2 x5 x6 x8) *
            class(brand / zero='None' separators=' ' ' on ')
         identity(a1 a2 a5 a6 a8) *
            class(brand / zero='None' separators=' ' ' on ') /
         lprefix=0;
   output out=coded(drop=_type_ _name_ intercept);
   id set c _freq_;
   label shelf = 'Shelf Talker'
         micro = 'Microwave';
   run;
```

Note that like we saw in the `%ChoicEff` macro, PROC TRANSREG produces the following warning.

```
WARNING: This usage of * sets one group's slope to zero.  Specify |
         to allow all slopes and intercepts to vary.  Alternatively,
         specify CLASS(vars) * identity(vars) identity(vars) for
         separate within group functions and a common intercept.
         This is a change from Version 6.
```

This is because `class` was interacted with `identity` using the asterisk instead of the vertical bar. In a linear model, this may be a sign of a coding error, so the procedure prints a warning. If you get this warning while coding a choice model specifying `zero=`*'constant-alternative-level'*, you can safely ignore it.

The analysis is the same as we have done previously with aggregate data. PROC PHREG is run to fit the mother logit model, complete with availability cross effects.

```
proc phreg data=coded;
   strata set;
   model c*c(2) = &_trgind / ties=breslow;
   freq _freq_;
   run;
```

Multinomial Logit Model Results

These steps produced the following results. (Recall that we used `%phchoice(on)` on page 143 to customize the output from PROC PHREG.)

Consumer Food Product Example

The PHREG Procedure

Model Information

Data Set	WORK.CODED
Dependent Variable	c
Censoring Variable	c
Censoring Value(s)	2
Frequency Variable	_FREQ_
Ties Handling	BRESLOW

Number of Observations Read	284
Number of Observations Used	284
Sum of Frequencies Read	47400
Sum of Frequencies Used	47400

Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Stratum	Set	Number of Alternatives	Chosen Alternatives	Not Chosen
1	1	1500	300	1200
2	2	900	300	600
3	3	900	300	600
4	4	1500	300	1200
5	5	1500	300	1200
6	6	900	300	600
7	7	1500	300	1200
8	8	1500	300	1200
9	9	1500	300	1200
10	10	1500	300	1200
11	11	900	300	600
12	12	1500	300	1200
13	13	1500	300	1200
14	14	900	300	600
15	15	900	300	600
16	16	1500	300	1200
17	17	1500	300	1200
18	18	1500	300	1200
19	19	1500	300	1200
20	20	1200	300	900
21	21	900	300	600
22	22	1200	300	900
23	23	1500	300	1200
24	24	1500	300	1200
25	25	1500	300	1200
26	26	1500	300	1200
27	27	1500	300	1200
28	28	1500	300	1200
29	29	1500	300	1200
30	30	900	300	600
31	31	1500	300	1200
32	32	1500	300	1200
33	33	1200	300	900
34	34	1200	300	900
35	35	1200	300	900
36	36	1200	300	900
Total		47400	10800	36600

Convergence Status

Convergence criterion (GCONV=1E-8) satisfied.

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	154710.28	134305.63
AIC	154710.28	134409.63
SBC	154710.28	134788.57

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	20404.6461	52	<.0001
Score	22883.7078	52	<.0001
Wald	6444.0844	52	<.0001

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Client	1	8.16629	3.96395	4.2442	0.0394
Extension	1	10.30298	4.13379	6.2119	0.0127
National	1	5.41386	4.90468	1.2184	0.2697
Private	1	4.90773	4.06749	1.4558	0.2276
Regional	1	4.96459	5.93423	0.6999	0.4028
Client Price	1	-1.11653	0.77149	2.0945	0.1478
Extension Price	1	-0.99948	1.21987	0.6713	0.4126
National Price	1	1.25938	1.74132	0.5231	0.4695
Private Price	1	-1.33471	0.76283	3.0614	0.0802
Regional Price	1	-1.22852	1.48246	0.6867	0.4073
Shelf Talker	1	0.66941	0.07828	73.1204	<.0001
Micro	1	0.59645	0.06746	78.1706	<.0001
CE, Client on Client	0	0	.	.	.
CE, Client on Extension	1	-0.31640	0.78240	0.1635	0.6859
CE, Client on National	1	-0.50555	0.80031	0.3990	0.5276
CE, Client on Private	1	-0.25802	0.82061	0.0989	0.7532
CE, Client on Regional	1	1.15121	1.03011	1.2489	0.2638
CE, Extension on Client	1	-0.52993	1.22364	0.1876	0.6650
CE, Extension on Extension	0	0	.	.	.
CE, Extension on National	1	-0.55507	1.24852	0.1977	0.6566
CE, Extension on Private	1	0.20613	1.25789	0.0269	0.8698
CE, Extension on Regional	1	-0.54337	1.43547	0.1433	0.7050
CE, Regional on Client	1	-1.14955	1.07675	1.1398	0.2857
CE, Regional on Extension	1	-1.43726	1.08276	1.7620	0.1844
CE, Regional on National	1	-1.81230	1.13204	2.5629	0.1094
CE, Regional on Private	1	-1.20206	1.09592	1.2031	0.2727
CE, Regional on Regional	0	0	.	.	.

CE, Private on Client	1	-0.42457	0.75836	0.3134	0.5756
CE, Private on Extension	1	-0.35800	0.75937	0.2223	0.6373
CE, Private on National	1	-0.68966	0.79742	0.7480	0.3871
CE, Private on Private	0	0	.	.	.
CE, Private on Regional	1	-1.11543	1.08771	1.0516	0.3051
CE, National on Client	1	1.42556	1.75683	0.6584	0.4171
CE, National on Extension	1	1.03538	1.75043	0.3499	0.5542
CE, National on National	0	0	.	.	.
CE, National on Private	1	1.46740	1.78874	0.6730	0.4120
CE, National on Regional	1	-0.28269	2.28193	0.0153	0.9014
AE, Client on Client	0	0	.	.	.
AE, Client on Extension	1	0.12477	0.38019	0.1077	0.7428
AE, Client on National	1	0.10606	0.38579	0.0756	0.7834
AE, Client on Private	1	-0.04026	0.39633	0.0103	0.9191
AE, Client on Regional	1	-0.57219	0.48525	1.3904	0.2383
AE, Extension on Client	1	0.77428	0.65342	1.4041	0.2360
AE, Extension on Extension	0	0	.	.	.
AE, Extension on National	1	0.61514	0.67002	0.8429	0.3586
AE, Extension on Private	1	0.18324	0.67377	0.0740	0.7856
AE, Extension on Regional	1	0.38862	0.76269	0.2596	0.6104
AE, Regional on Client	1	0.87692	0.77389	1.2840	0.2572
AE, Regional on Extension	1	1.05490	0.77497	1.8529	0.1734
AE, Regional on National	1	1.29670	0.79530	2.6584	0.1030
AE, Regional on Private	1	0.98393	0.77581	1.6085	0.2047
AE, Regional on Regional	0	0	.	.	.
AE, Private on Client	1	0.29125	0.48172	0.3655	0.5454
AE, Private on Extension	1	0.26656	0.48436	0.3029	0.5821
AE, Private on National	1	0.49015	0.50341	0.9480	0.3302
AE, Private on Private	0	0	.	.	.
AE, Private on Regional	1	0.81907	0.74339	1.2140	0.2705
AE, National on Client	1	-1.15849	1.35844	0.7273	0.3938
AE, National on Extension	1	-0.88510	1.35042	0.4296	0.5122
AE, National on National	0	0	.	.	.
AE, National on Private	1	-1.32585	1.38251	0.9197	0.3376
AE, National on Regional	1	0.31543	1.74206	0.0328	0.8563

Since the number of alternatives is not constant within each choice set, the summary table has non-constant numbers of alternatives and numbers of alternatives not chosen. The number chosen, 300 (or one per subject per choice set), is constant, since each subject always chooses one alternative from each choice set regardless of the number of alternatives. The total number of alternatives ranges from 900 with three alternatives to 1500 with five alternatives.

The cross effects are mostly nonsignificant. Since most of the cross effects are nonsignificant, we can rerun the analysis with a simpler model.

```
proc transreg data=agg design=5000 nozeroconstant norestoremissing;
   model class(brand / zero='None')
         class(brand / zero='None' separators='' ' ') * identity(price)
         class(shelf micro / lprefix=5 0 zero='No' 'Stove') /
         lprefix=0;
   output out=coded(drop=_type_ _name_ intercept);
   id set c _freq_;
   label shelf = 'Shelf Talker'
         micro = 'Microwave';
   run;

proc phreg data=coded;
   strata set;
   model c*c(2) = &_trgind / ties=breslow;
   freq _freq_;
   run;
```

Here are the parameter estimates for the simpler model.

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Client	1	5.65644	0.20231	781.6872	<.0001
Extension	1	6.42043	0.21086	927.1112	<.0001
National	1	0.77822	0.63713	1.4919	0.2219
Private	1	4.51101	0.30491	218.8745	<.0001
Regional	1	1.71388	0.99673	2.9567	0.0855
Client Price	1	-0.76985	0.09446	66.4224	<.0001
Extension Price	1	-0.50666	0.08350	36.8162	<.0001
National Price	1	0.74444	0.29078	6.5542	0.0105
Private Price	1	-1.33357	0.14151	88.8068	<.0001
Regional Price	1	-0.42010	0.46037	0.8327	0.3615
Shelf Talker	1	0.71984	0.06941	107.5588	<.0001
Micro	1	0.58407	0.05632	107.5642	<.0001

The most to least preferred brands are: client line extension, client brand, private label, the regional competitor, the national brand, and the none alternative (with an implicit part-worth utility of zero). The price effects are mostly negative, and the positive effects are only marginally significant. Both the shelf-talker and the microwaveable option have positive utility.

Modeling Subject Attributes

This example uses the same design and data as we just saw, but this time we have some demographic information about our respondents that we wish to model. The following DATA step reads a subject number, the choices, and the respondent age and income (in thousands of dollars).

```
data results;
   input Subj (choose1-choose&n) (1.) age income;
   datalines;
  1 22222415521222252222122122222121252 2  33 109
  2 22222542124222212222212122222221211322  56 117
  3 21222452154522212222212222221121112522  56  78
  4 21212512121222212222212224212212122522  57 107
  .
  .
  .
299 11222512121212212152122222122112122622  41  89
300 12222512324212212222121122212321232322  38  95
;
```

Merging the data and design is no different from what we saw previously. To make this analysis simpler, we will not fit any cross effects or availability cross effects, although we certainly could.

```
%mktmerge(design=sasuser.Entree_ChDes(drop=x1--x8 a1--a8), data=results,
          out=res2, nsets=&n, nalts=&m, setvars=choose1-choose&n)

proc print data=res2;
   by subj set; id subj set;
   where (subj = 1 and set = 1) or
         (subj = 2 and set = 2) or
         (subj = 3 and set = 3) or
         (subj = 300 and set = 36);
   run;
```

Here is a small sample of the data. Note that like before, the unavailable alternatives are required for the merge step.

Consumer Food Product Example

Subj	Set	Age	Income	Brand	Price	Micro	Shelf	w	c
1	1	33	109	Client	$1.29	Stove	No	1	2
		33	109	Extension	$2.39	Micro	No	1	1
		33	109	Regional	$1.99	Stove	No	1	2
		33	109	Private	$0.00	Micro	No	0	2
		33	109	National	$1.99	Stove	No	1	2
		33	109	None	$0.00	Stove	No	1	2
2	2	56	117	Client	$1.29	Stove	No	1	2
		56	117	Extension	$1.89	Stove	No	1	1
		56	117	Regional	$0.00	Stove	No	0	2
		56	117	Private	$0.00	Stove	No	0	2
		56	117	National	$0.00	Stove	No	0	2
		56	117	None	$0.00	Stove	No	1	2

3	3	56	78	Client	$0.00	Stove	No	0	2
		56	78	Extension	$2.39	Micro	No	1	1
		56	78	Regional	$0.00	Stove	No	0	2
		56	78	Private	$2.29	Stove	No	1	2
		56	78	National	$0.00	Stove	No	0	2
		56	78	None	$0.00	Stove	No	1	2
300	36	38	95	Client	$2.09	Stove	No	1	2
		38	95	Extension	$2.39	Micro	Talker	1	1
		38	95	Regional	$0.00	Stove	No	0	2
		38	95	Private	$0.00	Stove	No	0	2
		38	95	National	$2.39	Stove	No	1	2
		38	95	None	$0.00	Stove	No	1	2

You can see that the demographic information matches the raw data and is constant within each subject. The rest of the data processing is virtually the same as well. Since we have demographic information, we will not aggregate. There would have to be ties in both the demographics and choice for aggregation to have any effect.

We use PROC TRANSREG to code, adding `Age` and `Income` to the analysis.

```
proc transreg data=res2 design=5000 nozeroconstant norestoremissing;
   model class(brand / zero='None')
         identity(age income) * class(brand / zero='None' separators='' ', ')
         class(brand / zero='None' separators='' ' ') * identity(price)
         class(shelf micro / lprefix=5 0 zero='No' 'Stove') /
         lprefix=0 order=data;
   output out=code(drop=_type_ _name_ intercept);
   id subj set c w;
   label shelf = 'Shelf Talker'
         micro = 'Microwave';
   run;

data coded(drop=w); set code; where w; run; /* exclude unavailable */
```

The `Age` and `Income` variables are incorporated into the analysis by interacting them with `Brand`. Demographic variables must be interacted with product attributes to have any effect. If `identity(age income)` had been specified instead of `identity(age income) * class(brand / ...)` the coefficients for age and income would be zero. This is because age and income are constant within each choice set and subject combination, which means they are constant within each stratum. The second separator `', '` is used to create names for the brand/demographic interaction terms like `'Age, Client'`.

These next steps print the first coded choice set.

```
proc print data=coded(obs=4) label;
   id brand price;
   var BrandClient -- BrandPrivate Shelf Micro c;
   run;
```

```
proc print data=coded(obs=4 drop=Age) label;
   id brand price;
   var Age:;
   run;
proc print data=coded(obs=4 drop=Income) label;
   id brand price;
   var Income:;
   run;
proc print data=coded(obs=4) label;
   id brand price;
   var BrandClientPrice -- BrandPrivatePrice;
   format BrandClientPrice -- BrandPrivatePrice best4.;
   run;
```

Here is the coded data set for the first subject and choice set. The part that is new is the second and third panel, which will be used to capture the brand by age and brand by income effects.

Here are the attributes and the brand effects.

Consumer Food Product Example

Brand	Price	Client	Extension	Regional	Private	Shelf Talker	Microwave	c
Client	$1.29	1	0	0	0	No	Stove	2
Extension	$2.39	0	1	0	0	No	Micro	1
Regional	$1.99	0	0	1	0	No	Stove	2
National	$1.99	0	0	0	0	No	Stove	2

Here are the age by brand effects.

Consumer Food Product Example

Brand	Price	Age, Client	Age, Extension	Age, Regional	Age, Private	Age, National
Client	$1.29	33	0	0	0	0
Extension	$2.39	0	33	0	0	0
Regional	$1.99	0	0	33	0	0
National	$1.99	0	0	0	0	33

Here are the income by brand effects.

Consumer Food Product Example

Brand	Price	Income, Client	Income, Extension	Income, Regional	Income, Private	Income, National
Client	$1.29	109	0	0	0	0
Extension	$2.39	0	109	0	0	0
Regional	$1.99	0	0	109	0	0
National	$1.99	0	0	0	0	109

Here are the alternative-specific price effects.

Consumer Food Product Example

Brand	Price	Client Price	Extension Price	Regional Price	Private Price
Client	$1.29	1.29	0	0	0
Extension	$2.39	0	2.39	0	0
Regional	$1.99	0	0	1.99	0
National	$1.99	0	0	0	0

The PROC PHREG specification is the same as we have used before with nonaggregated data.

```
proc phreg data=coded brief;
   model c*c(2) = &_trgind / ties=breslow;
   strata subj set;
   run;
```

This step took just about one minute and produced the following results.

Consumer Food Product Example

The PHREG Procedure

Model Information

Data Set	WORK.CODED
Dependent Variable	c
Censoring Variable	c
Censoring Value(s)	2
Ties Handling	BRESLOW

Number of Observations Read	47400
Number of Observations Used	47400

Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Pattern	Number of Choices	Number of Alternatives	Chosen Alternatives	Not Chosen
1	2400	3	1	2
2	1800	4	1	3
3	6600	5	1	4

Convergence Status

Convergence criterion (GCONV=1E-8) satisfied.

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	31508.579	10939.139
AIC	31508.579	10983.139
SBC	31508.579	11143.460

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	20569.4401	22	<.0001
Score	21088.4411	22	<.0001
Wald	6947.1766	22	<.0001

Consumer Food Product Example

The PHREG Procedure

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Client	1	2.04997	0.81287	6.3599	0.0117
Extension	1	0.47882	0.81714	0.3434	0.5579
Regional	1	-1.49923	1.51428	0.9802	0.3221
Private	1	3.33861	0.85485	15.2528	<.0001
National	1	-0.62955	1.06206	0.3514	0.5533
Age, Client	1	0.01142	0.00944	1.4655	0.2261
Age, Extension	1	0.00855	0.00949	0.8111	0.3678
Age, Regional	1	0.01114	0.01205	0.8548	0.3552
Age, Private	1	0.00826	0.00970	0.7247	0.3946
Age, National	1	0.00809	0.00964	0.7042	0.4014

Income, Client	1	0.03227	0.00771	17.4954	<.0001
Income, Extension	1	0.05717	0.00776	54.2991	<.0001
Income, Regional	1	0.02496	0.01014	6.0642	0.0138
Income, Private	1	0.00957	0.00794	1.4521	0.2282
Income, National	1	0.00929	0.00789	1.3858	0.2391
Client Price	1	-0.76510	0.09598	63.5410	<.0001
Extension Price	1	-0.51364	0.08465	36.8207	<.0001
Regional Price	1	-0.27824	0.46406	0.3595	0.5488
Private Price	1	-1.37957	0.14286	93.2538	<.0001
National Price	1	0.82684	0.29260	7.9852	0.0047
Shelf Talker	1	0.74026	0.07033	110.7751	<.0001
Micro	1	0.59312	0.05692	108.6006	<.0001

In previous examples, when we used the `brief` option to produce a brief summary of the strata, the table had only one line. In this case, since our choice sets have 3, 4, or 5 alternatives, we have three rows, one for each choice set size. The coefficients for the age and income variables are generally not very significant in this analysis except an effect for income on the client brand and particularly on the extension.

Allocation of Prescription Drugs

This example discusses an allocation study, which is a technique often used in the area of prescription drug marketing research. This example discusses designing the allocation experiment, processing the data, analyzing frequencies, analyzing proportions, coding, analysis, and results. The principles of designing an allocation study are the same as for designing a first-choice experiment, as is the coding and final analysis. However, processing the data before analysis is different.

The previous examples have all modeled simple choice. However, sometimes the response of interest is not simple first choice. For example, in prescription drug marketing, researchers often use allocation studies where multiple, not single choices are made. Physicians are asked questions like "For the next ten prescriptions you write for a particular condition, how many would you write for each of these drugs?" The response, for example, could be "5 for drug 1, none for drug 2, 3 for drug 3, and 2 for drug 4."

Designing the Allocation Experiment

In this study, physicians were asked to specify which of ten drugs they would prescribe to their next ten patients. In this study, ten drugs, Drug 1 − Drug 10, were available each at three different prices, \$50, \$75, and \$100. In real studies, real brand names would be used and there would probably be more attributes. Since experimental design has been covered in some detail in other examples, we chose a simple design for this experiment so that we could concentrate on data processing. First, we use the `%MktRuns` autocall macro to suggest a design size. (All of the autocall macros used in this book are documented starting on page 597.) We specify `3 ** 10` for the 10 three-level factors.

```
title 'Allocation of Prescription Drugs';

%mktruns( 3 ** 10 )
```

```
                Allocation of Prescription Drugs

                      Design Summary

                    Number of
                    Levels       Frequency

                         3          10

                Allocation of Prescription Drugs

Saturated      = 21
Full Factorial = 59,049

Some Reasonable                        Cannot Be
   Design Sizes       Violations       Divided By

             27 *                0
             36 *                0
             45 *                0
```

```
54 *              0
21                45     9
24                45     9
30                45     9
33                45     9
39                45     9
42                45     9

* - 100% Efficient Design can be made with the MktEx Macro.

                Allocation of Prescription Drugs

 n    Design                                          Reference

27              3 ** 13                             Fractional-Factorial
36    2 ** 11   3 ** 12                             Orthogonal Array
36    2 **  4   3 ** 13                             Orthogonal Array
36    2 **  2   3 ** 12    6 **  1                   Orthogonal Array
36              3 ** 13    4 **  1                   Orthogonal Array
36              3 ** 12   12 **  1                   Orthogonal Array
45              3 ** 10    5 **  1                   Orthogonal Array
54    2 **  1   3 ** 25                             Orthogonal Array
54    2 **  1   3 ** 21    9 **  1                   Orthogonal Array
54              3 ** 24    6 **  1                   Orthogonal Array
54              3 ** 20    6 **  1    9 **  1        Orthogonal Array
54              3 ** 18   18 **  1                   Orthogonal Array
```

We need at least 21 choice sets and we see the optimal sizes are all divisible by nine. We will use 27 choice sets, which can give us up to 13 three-level factors.

Next, we use the `%MktEx` macro to create the design.[†] In addition, one more factor is added to the design. This factor will be used to block the design into three blocks of size 9.

```
%let nalts = 10;

%mktex(3 ** &nalts 3, n=27, seed=396)
```

The macro finds a 100% *D*-efficient design.

```
                Allocation of Prescription Drugs

                    Algorithm Search History

                              Current          Best
 Design    Row,Col  D-Efficiency  D-Efficiency  Notes
 ------------------------------------------------------
      1      Start      100.0000      100.0000  Tab
      1        End      100.0000
```

[†]Due to machine, SAS release, and macro differences, you may not get exactly the same design as was used in this book, but the differences should be slight.

Allocation of Prescription Drugs

The OPTEX Procedure

Class Level Information

Class	Levels	-Values-
x1	3	1 2 3
x2	3	1 2 3
x3	3	1 2 3
x4	3	1 2 3
x5	3	1 2 3
x6	3	1 2 3
x7	3	1 2 3
x8	3	1 2 3
x9	3	1 2 3
x10	3	1 2 3
x11	3	1 2 3

Allocation of Prescription Drugs

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	100.0000	100.0000	100.0000	0.9230

The `%MktEx` macro always creates factor names of `x1`, `x2`, and so on with values of 1, 2, ... You can create a data set with the names and values you want and use it to rename the factors and reset the levels. This first step creates a data set with 11 variables, `Block` and `Brand1 - Brand10`. `Block` has values 1, 2, and 3, and the brand variables have values of 50, 75, and 100 with a `dollar` format. The `%MktLab` macro takes the `data=Randomized` design data set and uses the names, values, and formats in the `key=Key` data set to make the `out=Final` data set. This data set is sorted by block and printed. The `%MktEval` macro is called to check the results.

```
data key(drop=i);
   input Block Brand1;
   array Brand[10];
   do i = 2 to 10; brand[i] = brand1; end;
   format brand: dollar4.;
   datalines;
1  50
2  75
3 100
;
```

```
proc print; run;

%mktlab(key=key);

proc sort out=sasuser.DrugAllo_LinDes; by block; run;

proc print; id block; by block; run;

%mkteval(blocks=block)
```

Here is the `key=` data set.

Allocation of Prescription Drugs

Obs	Block	Brand1	Brand2	Brand3	Brand4	Brand5	Brand6	Brand7	Brand8	Brand9	Brand10
1	1	$50	$50	$50	$50	$50	$50	$50	$50	$50	$50
2	2	$75	$75	$75	$75	$75	$75	$75	$75	$75	$75
3	3	$100	$100	$100	$100	$100	$100	$100	$100	$100	$100

The `%MktLab` macro prints the following mapping information.

```
Variable Mapping:
   x1  : Block
   x2  : Brand1
   x3  : Brand2
   x4  : Brand3
   x5  : Brand4
   x6  : Brand5
   x7  : Brand6
   x8  : Brand7
   x9  : Brand8
   x10 : Brand9
   x11 : Brand10
```

Here is the design.

Allocation of Prescription Drugs

Block	Brand1	Brand2	Brand3	Brand4	Brand5	Brand6	Brand7	Brand8	Brand9	Brand10
1	$50	$75	$50	$75	$100	$100	$100	$100	$50	$100
	$100	$50	$100	$75	$75	$75	$100	$50	$100	$75
	$50	$50	$75	$100	$50	$75	$50	$75	$50	$75
	$75	$50	$50	$50	$100	$75	$75	$100	$75	$75
	$75	$75	$100	$100	$75	$100	$50	$50	$75	$100
	$50	$100	$100	$50	$75	$50	$75	$50	$50	$50
	$100	$75	$75	$50	$50	$100	$75	$75	$100	$100
	$100	$100	$50	$100	$100	$50	$50	$100	$100	$50
	$75	$100	$75	$75	$50	$50	$100	$75	$75	$50
2	$100	$75	$50	$100	$75	$50	$100	$75	$75	$75
	$100	$100	$100	$75	$50	$75	$75	$100	$75	$100
	$50	$75	$100	$50	$50	$50	$50	$100	$100	$75
	$75	$50	$100	$100	$50	$100	$100	$100	$50	$50
	$50	$100	$75	$100	$100	$75	$100	$50	$100	$100
	$100	$50	$75	$50	$100	$100	$50	$50	$75	$50
	$50	$50	$50	$75	$75	$100	$75	$75	$100	$50
	$75	$75	$75	$75	$100	$50	$75	$50	$50	$75
	$75	$100	$50	$50	$75	$75	$50	$75	$50	$100
3	$100	$75	$100	$75	$100	$75	$50	$75	$50	$50
	$75	$75	$50	$50	$50	$75	$100	$50	$100	$50
	$50	$75	$75	$100	$75	$75	$75	$100	$75	$50
	$50	$100	$50	$75	$50	$100	$50	$50	$75	$75
	$50	$50	$100	$50	$100	$50	$100	$75	$75	$100
	$75	$50	$75	$75	$75	$50	$50	$100	$100	$100
	$75	$100	$100	$100	$100	$100	$75	$75	$100	$75
	$100	$50	$50	$100	$50	$50	$75	$50	$50	$100
	$100	$100	$75	$50	$75	$100	$100	$100	$50	$75

Here are some of the evaluation results.

Allocation of Prescription Drugs
Canonical Correlations Between the Factors
There are 0 Canonical Correlations Greater Than 0.316

	Block	Brand1	Brand2	Brand3	Brand4	Brand5	Brand6	Brand7	Brand8	Brand9	Brand10
Block	1	0	0	0	0	0	0	0	0	0	0
Brand1	0	1	0	0	0	0	0	0	0	0	0
Brand2	0	0	1	0	0	0	0	0	0	0	0
Brand3	0	0	0	1	0	0	0	0	0	0	0
Brand4	0	0	0	0	1	0	0	0	0	0	0
Brand5	0	0	0	0	0	1	0	0	0	0	0
Brand6	0	0	0	0	0	0	1	0	0	0	0
Brand7	0	0	0	0	0	0	0	1	0	0	0
Brand8	0	0	0	0	0	0	0	0	1	0	0
Brand9	0	0	0	0	0	0	0	0	0	1	0
Brand10	0	0	0	0	0	0	0	0	0	0	1

Allocation of Prescription Drugs
Summary of Frequencies
There are 0 Canonical Correlations Greater Than 0.316

Frequencies

Block	9 9 9
Brand1	9 9 9
Brand2	9 9 9
Brand3	9 9 9
Brand4	9 9 9
Brand5	9 9 9
Brand6	9 9 9
Brand7	9 9 9
Brand8	9 9 9
Brand9	9 9 9
Brand10	9 9 9
Block Brand1	3 3 3 3 3 3 3 3 3
Block Brand2	3 3 3 3 3 3 3 3 3
Block Brand3	3 3 3 3 3 3 3 3 3
Block Brand4	3 3 3 3 3 3 3 3 3
Block Brand5	3 3 3 3 3 3 3 3 3
Block Brand6	3 3 3 3 3 3 3 3 3
Block Brand7	3 3 3 3 3 3 3 3 3
Block Brand8	3 3 3 3 3 3 3 3 3
Block Brand9	3 3 3 3 3 3 3 3 3
Block Brand10	3 3 3 3 3 3 3 3 3

.
.
.

```
N-Way            1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
                 1 1 1 1 1 1 1 1
```

Processing the Data

Questionnaires are generated and data collected using a minor modification of the methods discussed in earlier examples. The difference is instead of asking for first choice data, allocation data are collected instead. Each row of the input data set contains a block, subject, and set number, followed by the number of times each of the ten alternatives was chosen. If all of the choice frequencies are zero, then the constant alternative was chosen. The `if` statement is used to check data entry. For convenience, choice set number is recoded to run from 1 to 27 instead of consisting of three blocks of nine sets. This gives us one fewer variable on which to stratify.

```
data results;
   input Block Subject Set @9 (freq1-freq&nalts) (2.);
   if not (sum(of freq:) in (0, &nalts)) then put _all_;
   set = (block - 1) * 9 + set;
   datalines;
1   1 1  0 0 8 0 2 0 0 0 0 0
1   1 2  0 0 8 0 0 0 2 0 0 0
1   1 3  0 0 0 0 0 0 0 010 0
1   1 4  1 0 0 1 3 3 0 0 2 0
1   1 5  2 0 8 0 0 0 0 0 0 0
1   1 6  0 1 3 1 0 0 0 0 1 4
1   1 7  0 1 3 1 1 2 0 0 2 0
1   1 8  0 0 3 0 0 2 1 0 0 4
1   1 9  0 2 5 0 0 0 0 0 3 0
2   210  1 1 0 2 0 3 0 1 1 1
2   211  1 0 3 1 0 1 1 0 2 1
.
.
.
;
```

In the first step, in creating an analysis data set for an allocation study, we reformat the data from one row per choice set per block per subject ($9 \times 3 \times 100 = 2700$ observations) to one per alternative (including the constant) per choice set per block per subject ($(10+1)\times 9\times 3\times 100 = 29,700$ observations). For each choice set, 11 observations are written storing the choice frequency in the variable `Count` and the brand in the variable `Brand`. If no alternative is chosen, then the constant alternative is chosen ten times, otherwise it is chosen zero times.

```
data allocs(keep=block set brand count);
   set results;

   array freq[&nalts];
```

```
   * Handle the &nalts alternatives;
   do b = 1 to &nalts;
      Brand = 'Brand ' || put(b, 2.);
      Count = freq[b];
      output;
      end;
   * Constant alt choice is implied if nothing else is chosen.
     brand = ' ' is used to flag the constant alternative.;

   brand = ' ';
   count = 10 * (sum(of freq:) = 0);
   output;
   run;
proc print data=results(obs=3) label noobs; run;
proc print data=allocs(obs=33); id block set; by block set; run;
```

The PROC PRINT steps show how the first three observations of the `Results` data set are transposed into the first 33 observations of the `Allocs` data set.

Allocation of Prescription Drugs

Block	Subject	Set	Freq1	Freq2	Freq3	Freq4	Freq5	Freq6	Freq7	Freq8	Freq9	Freq10
1	1	1	0	0	8	0	2	0	0	0	0	0
1	1	2	0	0	8	0	0	0	2	0	0	0
1	1	3	0	0	0	0	0	0	0	0	10	0

Allocation of Prescription Drugs

Block	Set	Brand	Count
1	1	Brand 1	0
		Brand 2	0
		Brand 3	8
		Brand 4	0
		Brand 5	2
		Brand 6	0
		Brand 7	0
		Brand 8	0
		Brand 9	0
		Brand 10	0
			0

```
1       2     Brand  1       0
                     Brand  2       0
                     Brand  3       8
                     Brand  4       0
                     Brand  5       0
                     Brand  6       0
                     Brand  7       2
                     Brand  8       0
                     Brand  9       0
                     Brand 10       0
                                    0

1       3     Brand  1       0
                     Brand  2       0
                     Brand  3       0
                     Brand  4       0
                     Brand  5       0
                     Brand  6       0
                     Brand  7       0
                     Brand  8       0
                     Brand  9      10
                     Brand 10       0
                                    0
```

The next step aggregates the data. It stores in the variable `Count` the number of times each alternative of each choice set was chosen. This creates a data set with 297 observations (3 blocks $\times$ 9 sets $\times$ 11 alternatives = 297).

```
* Aggregate, store the results back in count.;

proc summary data=allocs nway missing;
   class set brand;
   output sum(count)=Count out=allocs(drop=_type_ _freq_);
   run;
```

These next steps prepare the design for analysis. We need to create a data set `Key` that describes how the factors in our design will be used for analysis. It will contain all of the factor names, `Brand1`, `Brand2`, ..., `Brand10`. We can run the `%MktKey` macro to get these names for cutting and pasting into the program without typing them.

```
%mktkey(Brand1-Brand10)
```

The `%MktKey` macro produced the following line.

```
Brand1 Brand2 Brand3 Brand4 Brand5 Brand6 Brand7 Brand8 Brand9 Brand10
```

The next step rolls out the experimental design data set to match the choice allocations data set. The data set is transposed from one row per choice set to one row per alternative per choice set. This data set also has 297 observations. As we saw in many previous examples, the `%MktRoll` macro can be used to process the design.

```
data key(keep=Brand Price);
   input Brand $ 1-8 Price $;
   datalines;
Brand  1    Brand1
Brand  2    Brand2
Brand  3    Brand3
Brand  4    Brand4
Brand  5    Brand5
Brand  6    Brand6
Brand  7    Brand7
Brand  8    Brand8
Brand  9    Brand9
Brand 10    Brand10
.           .
;
%mktroll(design=sasuser.DrugAllo_LinDes, key=key, alt=brand, out=rolled,
         options=nowarn)

proc print data=rolled(obs=11); format price dollar4.; run;
```

Allocation of Prescription Drugs

Obs	Set	Brand	Price
1	1	Brand 1	$50
2	1	Brand 2	$75
3	1	Brand 3	$50
4	1	Brand 4	$75
5	1	Brand 5	$100
6	1	Brand 6	$100
7	1	Brand 7	$100
8	1	Brand 8	$100
9	1	Brand 9	$50
10	1	Brand 10	$100
11	1		.

Both data sets must be sorted the same way before they can be merged. The constant alternative, indicated by a missing brand, is last in the design choice set and hence is out of order. Missing must come before nonmissing for the merge. The order is correct in the `Allocs` data set since it was created by PROC SUMMARY with `Brand` as a `class` variable.

```
proc sort data=rolled; by set brand; run;
```

The data are merged along with error checking to ensure that the merge proceeded properly. Both data sets should have the same observations and `Set` and `Brand` variables, so the merge should be one to one.

```
data allocs2;
   merge allocs(in=flag1) rolled(in=flag2);
   by set brand;
   if flag1 ne flag2 then put 'ERROR: Merge is not 1 to 1.';
   format price dollar4.;
   run;
proc print data=allocs2(obs=22);
   var brand price count;
   sum count;
   by notsorted set;
   id set;
   run;
```

In the aggregate and combined data set, we see how often each alternative was chosen for each choice set. For example, in the first choice set, the constant alternative was chosen zero times, Brand 1 at \$100 was chosen 103 times, and so on. The 11 alternatives were chosen a total of 1000 times, 100 subjects times 10 choices each.

Allocation of Prescription Drugs

Set	Brand	Price	Count
1		.	0
	Brand 1	\$50	103
	Brand 2	\$75	58
	Brand 3	\$50	318
	Brand 4	\$75	99
	Brand 5	\$100	54
	Brand 6	\$100	83
	Brand 7	\$100	71
	Brand 8	\$100	58
	Brand 9	\$50	100
	Brand 10	\$100	56
---			-----
1			1000

```
 2                          .        10
       Brand  1     $100          73
       Brand  2      $50          76
       Brand  3     $100         342
       Brand  4      $75          55
       Brand  5      $75          50
       Brand  6      $75          77
       Brand  7     $100          95
       Brand  8      $50          71
       Brand  9     $100          72
       Brand 10      $75          79
---                               -----
 2                                1000
```

At this point, the data set contains 297 observations (27 choice sets times 11 alternatives) showing the number of times each alternative was chosen. This data set must be augmented to also include the number of times each alternative was not chosen. For example, in the first choice set, brand 1 was chosen 103 times, which means it was not chosen $0+58+318+99+54+83+71+58+100+56 = 897$ times. We use a macro, `%MktAllo` for "marketing allocation study" to process the data. We specify the input `data=allocs2` data set, the output `out=allocs3` data set, the number of alternatives including the constant (`nalts=%eval(&nalts + 1)`), the variables in the data set except the frequency variable (`vars=set brand price`), and the frequency variable (`freq=Count`). The macro counts how many times each alternative was chosen and not chosen and writes the results to the `out=` data set along with the usual `c` = 1 for chosen and `c` = 2 for unchosen.

```
%mktallo(data=allocs2, out=allocs3, nalts=%eval(&nalts + 1),
         vars=set brand price, freq=Count)

proc print data=allocs3(obs=22);
   var set brand price count c;
   run;
```

The first 22 records of the allocation data set are shown next.

Allocation of Prescription Drugs

Obs	Set	Brand	Price	Count	c
1	1		.	0	1
2	1		.	1000	2
3	1	Brand 1	$50	103	1
4	1	Brand 1	$50	897	2
5	1	Brand 2	$75	58	1
6	1	Brand 2	$75	942	2

7	1	Brand	3	$50	318	1
8	1	Brand	3	$50	682	2
9	1	Brand	4	$75	99	1
10	1	Brand	4	$75	901	2
11	1	Brand	5	$100	54	1
12	1	Brand	5	$100	946	2
13	1	Brand	6	$100	83	1
14	1	Brand	6	$100	917	2
15	1	Brand	7	$100	71	1
16	1	Brand	7	$100	929	2
17	1	Brand	8	$100	58	1
18	1	Brand	8	$100	942	2
19	1	Brand	9	$50	100	1
20	1	Brand	9	$50	900	2
21	1	Brand	10	$100	56	1
22	1	Brand	10	$100	944	2

In the first choice set, the constant alternative is chosen zero times and not chosen 1000 times, Brand 1 is chosen 103 times and not chosen $1000 - 103 = 897$ times, Brand 2 is chosen 58 times and not chosen $1000 - 58 = 942$ times, and so on. Note that allocation studies do not always have fixed sums, so it is important to use the `%MktAllo` macro or some other approach that actually counts the number of times each alternative was not chosen. It is not always sufficient to simply subtract from a fixed constant (in this case, 1000).

Coding and Analysis

The next step codes the design for analysis. Indicator variables are created for `Brand` and `Price`. All of the PROC TRANSREG options have been discussed in other examples.

```
proc transreg design data=allocs3 nozeroconstant norestoremissing;
   model class(brand price / zero=none) / lprefix=0;
   output out=coded(drop=_type_ _name_ intercept);
   id set c count;
   run;
```

Analysis proceeds like it has in all other examples. We stratify by choice set number. We do not need to stratify by `Block` since choice set number does not repeat within block.

```
proc phreg data=coded;
   where count > 0;
   model c*c(2) = &_trgind / ties=breslow;
   freq count;
   strata set;
   run;
```

We used the `where` statement to exclude observations with zero frequency; otherwise PROC PHREG complains about them.

Multinomial Logit Model Results

Here are the results. Recall that we used `%phchoice(on)` on page 143 to customize the output from PROC PHREG.

Allocation of Prescription Drugs

The PHREG Procedure

Model Information

Data Set	WORK.CODED
Dependent Variable	c
Censoring Variable	c
Censoring Value(s)	2
Frequency Variable	Count
Ties Handling	BRESLOW

Number of Observations Read	583
Number of Observations Used	583
Sum of Frequencies Read	297000
Sum of Frequencies Used	297000

Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Stratum	Set	Number of Alternatives	Chosen Alternatives	Not Chosen
1	1	11000	1000	10000
2	2	11000	1000	10000
3	3	11000	1000	10000
4	4	11000	1000	10000
5	5	11000	1000	10000
6	6	11000	1000	10000
7	7	11000	1000	10000
8	8	11000	1000	10000
9	9	11000	1000	10000
10	10	11000	1000	10000
11	11	11000	1000	10000
12	12	11000	1000	10000
13	13	11000	1000	10000
14	14	11000	1000	10000
15	15	11000	1000	10000
16	16	11000	1000	10000
17	17	11000	1000	10000
18	18	11000	1000	10000

19	19	11000	1000	10000
20	20	11000	1000	10000
21	21	11000	1000	10000
22	22	11000	1000	10000
23	23	11000	1000	10000
24	24	11000	1000	10000
25	25	11000	1000	10000
26	26	11000	1000	10000
27	27	11000	1000	10000
Total		297000	27000	270000

Convergence Status

Convergence criterion (GCONV=1E-8) satisfied.

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	502505.13	489062.66
AIC	502505.13	489086.66
SBC	502505.13	489185.11

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	13442.4676	12	<.0001
Score	18340.8415	12	<.0001
Wald	14087.6778	12	<.0001

Multinomial Logit Parameter Estimates

		DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Brand	1	1	2.09906	0.06766	962.5297	<.0001
Brand	2	1	2.09118	0.06769	954.5113	<.0001
Brand	3	1	3.54204	0.06484	2984.4698	<.0001
Brand	4	1	2.09710	0.06766	960.5277	<.0001
Brand	5	1	2.08523	0.06771	948.4791	<.0001
Brand	6	1	2.03530	0.06790	898.6218	<.0001
Brand	7	1	2.06920	0.06777	932.3154	<.0001
Brand	8	1	2.08573	0.06771	948.9824	<.0001
Brand	9	1	2.11705	0.06759	980.9640	<.0001
Brand	10	1	2.06363	0.06779	926.7331	<.0001

$50	1	0.00529	0.01628	0.1058	0.7450
$75	1	0.0005304	0.01629	0.0011	0.9740
$100	0	0	.	.	.

The output shows that there are 27 strata, one per choice set, each consisting of 1000 chosen alternatives (10 choices by 100 subjects) and 10,000 unchosen alternatives. All of the brand coefficients are "significant," with the Brand 3 effect being by far the strongest. (We will soon see that statistical significance should be ignored with allocation studies.) There is no price effect.

Analyzing Proportions

Recall that we collected data by asking physicians to report which brands they would prescribe the next ten times they write prescriptions. Alternatively, we could ask them to report the *proportion* of time they would prescribe each brand. We can simulate having proportion data by dividing our count data by 10. This means our frequency variable will no longer contain integers, so we need to specify the `notruncate` option on PROC PHREG `freq` statement to allow "noninteger frequencies."

```
data coded2;
   set coded;
   count = count / 10;
   run;

proc phreg data=coded2;
   where count > 0;
   model c*c(2) = &_trgind / ties=breslow;
   freq count / notruncate;
   strata set;
   run;
```

When we do this, we see the number of alternatives and the number chosen and not chosen decrease by a factor of 10 as do all of the Chi-Square tests. The coefficients are unchanged. This implies that market share calculations are invariant to the different scalings of the frequencies. However, the p-values are not invariant. The sample size is artificially inflated when counts are used so p-values are not interpretable in an allocation study. When proportions are used, each subject is contributing 1 to the number chosen instead of 10, just like a normal choice study, so p-values have meaning.

```
Allocation of Prescription Drugs

The PHREG Procedure

Model Information

Data Set                  WORK.CODED2
Dependent Variable        c
Censoring Variable        c
Censoring Value(s)        2
Frequency Variable        Count
Ties Handling             BRESLOW
```

Number of Observations Read	583
Number of Observations Used	583
Sum of Frequencies Read	29700
Sum of Frequencies Used	29700

Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Stratum	Set	Number of Alternatives	Chosen Alternatives	Not Chosen
1	1	1100.0	100.0	1000.0
2	2	1100.0	100.0	1000.0
3	3	1100.0	100.0	1000.0
4	4	1100.0	100.0	1000.0
5	5	1100.0	100.0	1000.0
6	6	1100.0	100.0	1000.0
7	7	1100.0	100.0	1000.0
8	8	1100.0	100.0	1000.0
9	9	1100.0	100.0	1000.0
10	10	1100.0	100.0	1000.0
11	11	1100.0	100.0	1000.0
12	12	1100.0	100.0	1000.0
13	13	1100.0	100.0	1000.0
14	14	1100.0	100.0	1000.0
15	15	1100.0	100.0	1000.0
16	16	1100.0	100.0	1000.0
17	17	1100.0	100.0	1000.0
18	18	1100.0	100.0	1000.0
19	19	1100.0	100.0	1000.0
20	20	1100.0	100.0	1000.0
21	21	1100.0	100.0	1000.0
22	22	1100.0	100.0	1000.0
23	23	1100.0	100.0	1000.0
24	24	1100.0	100.0	1000.0
25	25	1100.0	100.0	1000.0
26	26	1100.0	100.0	1000.0
27	27	1100.0	100.0	1000.0
Total		29700.0	2700.0	27000.0

Convergence Status

Convergence criterion (GCONV=1E-8) satisfied.

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	37816.553	36472.307
AIC	37816.553	36496.307
SBC	37816.553	36567.119

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	1344.2468	12	<.0001
Score	1834.0841	12	<.0001
Wald	1408.7678	12	<.0001

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Brand 1	1	2.09906	0.21395	96.2530	<.0001
Brand 2	1	2.09118	0.21404	95.4511	<.0001
Brand 3	1	3.54204	0.20503	298.4470	<.0001
Brand 4	1	2.09710	0.21398	96.0528	<.0001
Brand 5	1	2.08523	0.21411	94.8479	<.0001
Brand 6	1	2.03530	0.21470	89.8622	<.0001
Brand 7	1	2.06920	0.21430	93.2315	<.0001
Brand 8	1	2.08573	0.21411	94.8982	<.0001
Brand 9	1	2.11705	0.21375	98.0964	<.0001
Brand 10	1	2.06363	0.21436	92.6733	<.0001
$50	1	0.00529	0.05148	0.0106	0.9181
$75	1	0.0005304	0.05152	0.0001	0.9918
$100	0	0	.	.	.

Chair Design with Generic Attributes

This study illustrates creating an experimental design for a purely generic choice model. This example discusses generic attributes, alternative swapping, choice set swapping, and constant alternatives. In a purely generic study, there are no brands, just bundles of attributes. Also see page 89 in the experimental design chapter for examples of how to combinatorially construct optimal generic choice designs for certain problems.

Say a manufacturer is interested in designing one or more new chairs. The manufacturer can vary the attributes of the chairs, present subjects with competing chair designs, and model the effects of the attributes on choice. Here are the attributes of interest.

Factor	Attribute	Levels
X1	Color	3 Colors
X2	Back	3 Styles
X3	Seat	3 Styles
X4	Arm Rest	3 Styles
X5	Material	3 Materials

Since seeing descriptions of chairs is not the same as seeing and sitting in the actual chairs, the manufacturer is going to actually make sample chairs for people to try and choose from. Subjects will be shown groups of three chairs at a time. If we were to make our design using the approach discussed in previous examples, we would use the `%MktEx` autocall macro to create a design with 15 factors, five for the first chair, five for the second chair, and five for the third chair. This design would have to have at least $15 \times (3-1)+1 = 31$ runs and 93 sample chairs. Here is how we could have made the design.[‡]

```
title 'Generic Chair Attributes';

* This design will not be used;
%mktex(3 ** 15, n=36, seed=238)

%mktkey(3 5)

%mktroll(design=randomized, key=key, out=cand)
```

The `%MktEx` approach to designing an experiment like this allows you to fit very general models including models with alternative-specific effects and even mother logit models. However, at analysis time for this purely generic model, we will fit a model with 10 parameters, two for each of the five factors, `class(x1-x5)`. Creating a design with over $31 \times 3 = 93$ chairs is way too expensive. In ordinary linear designs, we need at least as many runs as parameters. In choice designs, we need to count the total number of alternatives across all choice sets, subtract the number of choice sets, and this number must be at least as large as the number of parameters. Equivalently, each choice set allows us to estimate $m-1$ parameters, where m is the number of alternatives in that choice set. In this case, we could fit our purely generic model with as few as $10/(3-1) = 5$ choice sets.

Since we only need a simple generic model for this example, and since our chair manufacturing for our research will be expensive, we will not use the `%MktEx` approach for designing our choice experiment. Instead, we will use a different approach that will allow us to get a smaller design that is adequate for our model and budget. Recall the discussion of linear design efficiency, choice model design efficiency, and using linear design efficiency as a surrogate for choice design goodness starting on page 53. Instead

[‡]Due to machine, SAS release, and macro differences, you may not get exactly the same design as was used in this book, but the differences should be slight.

of using linear design efficiency as a surrogate for choice design goodness, we can directly optimize choice design efficiency given an assumed model and parameter vector $\boldsymbol{\beta}$. This approach uses the `%ChoicEff` macro.

Generic Attributes, Alternative Swapping, Large Candidate Set

This part of the example illustrates using the `%ChoicEff` macro for efficient choice designs, using its algorithm that builds a design from candidate alternatives (as opposed to candidates consisting of entire choice sets). First, we will use the `%MktRuns` macro to suggest a candidate-set size.

```
%mktruns(3 ** 5)
```

Here are some of the results.

Generic Chair Attributes

Design Summary

Number of Levels	Frequency
3	5

Saturated = 11
Full Factorial = 243

Some Reasonable Design Sizes	Violations	Cannot Be Divided By
18 *	0	
27 *	0	
36 *	0	
12	10	9
15	10	9
21	10	9
24	10	9
30	10	9
33	10	9
11	15	3 9

* - 100% Efficient Design can be made with the MktEx Macro.

Generic Chair Attributes

n	Design										Reference
18	2	**	1	3	**	7					Orthogonal Array
18				3	**	6	6	**	1		Orthogonal Array
27				3	**	13					Fractional-Factorial
27				3	**	9	9	**	1		Fractional-Factorial
36	2	**	11	3	**	12					Orthogonal Array
36	2	**	10	3	**	8	6	**	1		Orthogonal Array
36	2	**	4	3	**	13					Orthogonal Array
36	2	**	3	3	**	9	6	**	1		Orthogonal Array
36	2	**	2	3	**	12	6	**	1		Orthogonal Array
36	2	**	2	3	**	5	6	**	2		Orthogonal Array
36	2	**	1	3	**	8	6	**	2		Orthogonal Array
36				3	**	13	4	**	1		Orthogonal Array
36				3	**	12	12	**	1		Orthogonal Array
36				3	**	7	6	**	3		Orthogonal Array

We could use candidate sets of size: 18, 27 or 36. Additionally, since this problem is small, we could try an 81-run fractional-factorial design or the 243-run full-factorial design. We will choose the 243-run full-factorial design, since it is reasonably small and it will give the macro the most freedom to find a good design.[§]

We will use the `%MktEx` macro to create a candidate set. The candidate set will consist of 5 three-level factors, one for each of the five generic attributes. We will add three flag variables to the candidate set, `f1-f3`, one for each alternative. Since there are three alternatives, the candidate set must contain those observations that may be used for alternative 1, those observations that may be used for alternative 2, and those observations that may be used for alternative 3. The flag variable for each alternative consists of ones for those candidates that may be included for that alternative and zeros or missings for those candidates that may not be included for that alternative. The candidates for the different alternatives may be all different, all the same, or something in between depending on the problem. For example, the candidate set may contain one observation that is only used for the last, constant alternative. In this purely generic case, each flag variable consists entirely of ones indicating that any candidate can appear in any alternative. The `%MktEx` macro will not allow you to create constant or one-level factors. We can instead use the `%MktLab` macro to add the flag variables, essentially by specifying that we have multiple intercepts. The option `int=f1-f3` creates three variables with values all one. The default output data set is called `Final`. The following code creates the candidates.

```
%mktex(3 ** 5, n=243)
%mktlab(data=design, int=f1-f3)

proc print data=final(obs=27); run;
```

[§]Later, we will see we could have chosen 18.

The columns `f1-f3` are the flags, and `x1-x5` are the generic attributes. Here is part of the candidate set.

Generic Chair Attributes

Obs	f1	f2	f3	x1	x2	x3	x4	x5
1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	2
3	1	1	1	1	1	1	1	3
4	1	1	1	1	1	1	2	1
5	1	1	1	1	1	1	2	2
6	1	1	1	1	1	1	2	3
7	1	1	1	1	1	1	3	1
8	1	1	1	1	1	1	3	2
9	1	1	1	1	1	1	3	3
10	1	1	1	1	1	2	1	1
11	1	1	1	1	1	2	1	2
12	1	1	1	1	1	2	1	3
13	1	1	1	1	1	2	2	1
14	1	1	1	1	1	2	2	2
15	1	1	1	1	1	2	2	3
16	1	1	1	1	1	2	3	1
17	1	1	1	1	1	2	3	2
18	1	1	1	1	1	2	3	3
19	1	1	1	1	1	3	1	1
20	1	1	1	1	1	3	1	2
21	1	1	1	1	1	3	1	3
22	1	1	1	1	1	3	2	1
23	1	1	1	1	1	3	2	2
24	1	1	1	1	1	3	2	3
25	1	1	1	1	1	3	3	1
26	1	1	1	1	1	3	3	2
27	1	1	1	1	1	3	3	3

Next, we will search that candidate set for an efficient design for the model specification `class(x1-x5)` and the assumption $\boldsymbol{\beta} = \mathbf{0}$. We will use the `%ChoicEff` autocall macro to do this. (All of the autocall macros used in this book are documented starting on page 597.) This approach is based on the work of Huber and Zwerina (1996) who proposed constructing efficient experimental designs for choice experiments under an assumed model and $\boldsymbol{\beta}$. The `%ChoicEff` macro uses a modified Fedorov algorithm (Fedorov, 1972; Cook and Nachtsheim, 1980) to optimize the choice model variance matrix. We will be using the largest possible candidate set for this problem, the full-factorial design, and we will ask for more than the default number of iterations, so run time will be slower than it could be. However, we will be requesting a very small number of choice sets. Building the chairs will be expensive, so we want to get a really good but small design. This specification requests a generic design with six choice sets each consisting of three alternatives.

```
%choiceff(data=final, model=class(x1-x5), nsets=6, maxiter=100,
          seed=121, flags=f1-f3, beta=zero)
```

The `data=final` option names the input data set of candidates. The `model=class(x1-x5)` option specifies the most general model that will be considered at analysis time. The `nsets=6` option specifies the number of choice sets. Note that this is considerably smaller than the minimum of 31 that would be required if we were just using the `%MktEx` linear-design approach ($6 \times 3 = 18$ chairs instead of $31 \times 3 = 93$ chairs). The `maxiter=100` option requests 100 designs based on 100 random initial designs (by default, `maxiter=2`). The `seed=121` option specifies the random number seed. The `flags=f1-f3` specifies the flag variables for alternatives 1 to 3. Implicitly, this option also specifies the fact that there are three alternatives since three flag variables were specified. The `beta=zero` option specifies the assumption $\boldsymbol{\beta} = \mathbf{0}$. A vector of numbers like `beta=-1 0 -1 0 -1 0 -1 0 -1 0 -1 0` could be specified. (See page 609 for an example of this.) When you wish to assume all parameters are zero, you can specify `beta=zero` instead of typing a vector of the zeros. You can also omit the `beta=` option if you just want the macro to list the parameters. You can use this list to ensure that you specify the parameters in the right order.

The first part of the output from the macro is a list of all of the effects generated and the assumed values of $\boldsymbol{\beta}$. It is very important to check this list and make sure it is correct. In particular, when you are explicitly specifying the $\boldsymbol{\beta}$ vector, you need to make sure you specified all of the values in the right order.

Generic Chair Attributes

n	Name	Beta	Label
1	x11	0	x1 1
2	x12	0	x1 2
3	x21	0	x2 1
4	x22	0	x2 2
5	x31	0	x3 1
6	x32	0	x3 2
7	x41	0	x4 1
8	x42	0	x4 2
9	x51	0	x5 1
10	x52	0	x5 2

Next, the macro produces the iteration history, which is different from the iteration histories we are used to seeing in the `%MktEx` macro. The `%ChoicEff` macro uses PROC IML and a modified Fedorov algorithm to iteratively improve the efficiency of the choice design given the specified candidates, model, and $\boldsymbol{\beta}$. Note that these efficiencies are *not* on a 0 to 100 scale. This step took about 12 minutes. Here are some of the results.

Generic Chair Attributes

Design	Iteration	D-Efficiency	D-Error
1	0	0.352304	2.838455
	1	0.946001	1.057081
	2	1.001164	0.998838
	3	1.041130	0.960494
	4	1.044343	0.957540

.
.
.

Design	Iteration	D-Efficiency	D-Error
34	0	0.469771	2.128698
	1	0.919074	1.088051
	2	1.058235	0.944970
	3	1.154701	0.866025
	4	1.154701	0.866025

.
.
.

Design	Iteration	D-Efficiency	D-Error
100	0	0.456308	2.191501
	1	1.006320	0.993719
	2	1.042702	0.959046
	3	1.042702	0.959046

Next, the macro shows which design it chose and the final *D*-efficiency and *D*-error (*D*-efficiency = 1 / *D*-error).

Final Results

Design	34
Choice Sets	6
Alternatives	3
D-Efficiency	1.154701
D-Error	0.866025

Next, it shows the variance, standard error, and *df* for each effect. It is important to ensure that each effect is estimable: ($df = 1$). Usually, when all of the variances are constant, like we see in this table, it means that the macro has found the optimal design.

Generic Chair Attributes

n	Variable Name	Label	Variance	DF	Standard Error
1	x11	x1 1	1	1	1
2	x12	x1 2	1	1	1
3	x21	x2 1	1	1	1
4	x22	x2 2	1	1	1
5	x31	x3 1	1	1	1
6	x32	x3 2	1	1	1
7	x41	x4 1	1	1	1
8	x42	x4 2	1	1	1
9	x51	x5 1	1	1	1
10	x52	x5 2	1	1	1
				==	
				10	

The data set `Best` contains the final, best design found.

```
proc print; by set; id set; run;
```

The data set contains: `Design` - the number of the design with the maximum D-efficiency, `Efficiency` - the D-efficiency of this design, `Index` - the candidate set observation number, `Set` - the choice set number, `Prob` - the probability that this alternative will be chosen given $\boldsymbol{\beta}$, `n` - the observation number, `x1-x5` - the design, and `f1-f3` - the flags.

Generic Chair Attributes

Set	Design	Efficiency	Index	Prob	n	f1	f2	f3	x1	x2	x3	x4	x5
1	34	1.15470	183	0.33333	595	1	1	1	3	1	3	1	3
	34	1.15470	62	0.33333	596	1	1	1	1	3	1	3	2
	34	1.15470	121	0.33333	597	1	1	1	2	2	2	2	1
2	34	1.15470	217	0.33333	598	1	1	1	3	3	1	1	1
	34	1.15470	45	0.33333	599	1	1	1	1	2	2	3	3
	34	1.15470	104	0.33333	600	1	1	1	2	1	3	2	2
3	34	1.15470	215	0.33333	601	1	1	1	3	2	3	3	2
	34	1.15470	147	0.33333	602	1	1	1	2	3	2	1	3
	34	1.15470	4	0.33333	603	1	1	1	1	1	1	2	1
4	34	1.15470	78	0.33333	604	1	1	1	1	3	3	2	3
	34	1.15470	178	0.33333	605	1	1	1	3	1	2	3	1
	34	1.15470	110	0.33333	606	1	1	1	2	2	1	1	2
5	34	1.15470	90	0.33333	607	1	1	1	2	1	1	3	3
	34	1.15470	46	0.33333	608	1	1	1	1	2	3	1	1
	34	1.15470	230	0.33333	609	1	1	1	3	3	2	2	2

6	34	1.15470	195	0.33333	610	1	1	1	3	2	1	2	3
	34	1.15470	11	0.33333	611	1	1	1	1	1	2	1	2
	34	1.15470	160	0.33333	612	1	1	1	2	3	3	3	1

This design has 18 runs (6 choice sets × 3 alternatives). Notice that in this design, each level occurs exactly once in each factor and each choice set. To use this design for analysis, you would only need the variables `Set` and `x1-x5`. Since it is already in choice design format, it would not need to be processed using the `%MktRoll` macro. Since data collection, processing, and analysis have already been covered in detail in other examples, this example will concentrate solely on experimental design.

Generic Attributes, Alternative Swapping, Small Candidate Set

In this part of this example, we will try to make an equivalent design to the one we just made, only this time using a smaller candidate set. Here is the code.

```
%mktex(3 ** 5, n=18)

%mktlab(data=design, int=f1-f3)

%choiceff(data=final, model=class(x1-x5), nsets=6, maxiter=20,
          seed=121, flags=f1-f3, beta=zero)

proc print; run;
```

This time, instead of creating a full-factorial candidate set, we asked for 5 three-level factors from the L_{18}, an orthogonal tabled design in 18 runs. We also asked for fewer iterations in the `%ChoicEff` macro. Since the candidate set is much smaller, the macro should be able to find the best design available in this candidate set fairly easily. Here are some of the results.

Generic Chair Attributes

n	Name	Beta	Label
1	x11	0	x1 1
2	x12	0	x1 2
3	x21	0	x2 1
4	x22	0	x2 2
5	x31	0	x3 1
6	x32	0	x3 2
7	x41	0	x4 1
8	x42	0	x4 2
9	x51	0	x5 1
10	x52	0	x5 2

Generic Chair Attributes

Design	Iteration	D-Efficiency	D-Error
1	0	0	.
	1	0.913290	1.094943
	2	1.008888	0.991191
	3	1.042878	0.958885
	4	1.154701	0.866025
	5	1.154701	0.866025

.
.
.

Design	Iteration	D-Efficiency	D-Error
20	0	0.364703	2.741954
	1	0.851038	1.175036
	2	1.008888	0.991191
	3	1.042878	0.958885
	4	1.154701	0.866025
	5	1.154701	0.866025

Final Results

Design	1
Choice Sets	6
Alternatives	3
D-Efficiency	1.154701
D-Error	0.866025

Generic Chair Attributes

n	Variable Name	Label	Variance	DF	Standard Error
1	x11	x1 1	1	1	1
2	x12	x1 2	1	1	1
3	x21	x2 1	1	1	1
4	x22	x2 2	1	1	1
5	x31	x3 1	1	1	1
6	x32	x3 2	1	1	1
7	x41	x4 1	1	1	1
8	x42	x4 2	1	1	1
9	x51	x5 1	1	1	1
10	x52	x5 2	1	1	1
				==	
				10	

Generic Chair Attributes

Obs	Design	Efficiency	Index	Set	Prob	n	f1	f2	f3	x1	x2	x3	x4	x5
1	1	1.15470	11	1	0.33333	1	1	1	1	2	3	1	3	1
2	1	1.15470	13	1	0.33333	2	1	1	1	3	1	2	1	2
3	1	1.15470	4	1	0.33333	3	1	1	1	1	2	3	2	3
4	1	1.15470	3	2	0.33333	4	1	1	1	1	2	1	3	2
5	1	1.15470	12	2	0.33333	5	1	1	1	2	3	2	1	3
6	1	1.15470	14	2	0.33333	6	1	1	1	3	1	3	2	1
7	1	1.15470	5	3	0.33333	7	1	1	1	1	3	2	2	1
8	1	1.15470	8	3	0.33333	8	1	1	1	2	1	3	3	2
9	1	1.15470	15	3	0.33333	9	1	1	1	3	2	1	1	3
10	1	1.15470	9	4	0.33333	10	1	1	1	2	2	2	2	2
11	1	1.15470	1	4	0.33333	11	1	1	1	1	1	1	1	1
12	1	1.15470	18	4	0.33333	12	1	1	1	3	3	3	3	3
13	1	1.15470	10	5	0.33333	13	1	1	1	2	2	3	1	1
14	1	1.15470	17	5	0.33333	14	1	1	1	3	3	1	2	2
15	1	1.15470	2	5	0.33333	15	1	1	1	1	1	2	3	3
16	1	1.15470	6	6	0.33333	16	1	1	1	1	3	3	1	2
17	1	1.15470	7	6	0.33333	17	1	1	1	2	1	1	2	3
18	1	1.15470	16	6	0.33333	18	1	1	1	3	2	2	3	1

Notice that we got the same D-efficiency and variances as before (D-efficiency = 1.1547005384 and all variances 1). Also notice the `Index` variable in the design (which is the candidate set row number). Each candidate appears in the design exactly once. As is shown in the experimental design chaper starting on page 89, for problems like this (all generic attributes, no brands, no constant alternative, total number of alternatives equal to the number of runs in an orthogonal design, all factors available in that orthogonal design, and an assumed $\boldsymbol{\beta}$ vector of zero) that the optimal design can be created by optimally sorting the rows of an orthogonal design into choice sets, and the `%ChoicEff` macro can do this quite well. More directly, this design could be made from the orthogonal array $3^6 6^1$ in 18 runs by using the six-level factor as the choice set number.

Six choice sets is a bit small. If you can afford a larger number, it would be good to try a larger design. In this case, nine choice sets are requested using a fractional-factorial candidate set in 27 runs. Notice that like before, the number of runs in the candidate set was chosen to be the product of the number of choice sets and the number of alternatives in each choice set.

```
%mktex(3 ** 5, n=27, seed=382)

%mktlab(data=design, int=f1-f3)

%choiceff(data=final, model=class(x1-x5), nsets=9, maxiter=20,
          seed=121, flags=f1-f3, beta=zero)

proc print; id set; by set; var index prob x:; run;
```

Here are the variances and the design.

Generic Chair Attributes

n	Variable Name	Label	Variance	DF	Standard Error
1	x11	x1 1	0.66667	1	0.81650
2	x12	x1 2	0.66667	1	0.81650
3	x21	x2 1	0.66667	1	0.81650
4	x22	x2 2	0.66667	1	0.81650
5	x31	x3 1	0.66667	1	0.81650
6	x32	x3 2	0.66667	1	0.81650
7	x41	x4 1	0.66667	1	0.81650
8	x42	x4 2	0.66667	1	0.81650
9	x51	x5 1	0.66667	1	0.81650
10	x52	x5 2	0.66667	1	0.81650
				==	
				10	

Generic Chair Attributes

Set	Index	Prob	x1	x2	x3	x4	x5
1	9	0.33333	1	3	3	1	2
	13	0.33333	2	2	1	3	3
	20	0.33333	3	1	2	2	1
2	25	0.33333	3	3	1	2	2
	5	0.33333	1	2	2	1	3
	12	0.33333	2	1	3	3	1
3	6	0.33333	1	2	3	3	1
	26	0.33333	3	3	2	1	3
	10	0.33333	2	1	1	2	2
4	22	0.33333	3	2	1	1	1
	2	0.33333	1	1	2	3	2
	18	0.33333	2	3	3	2	3
5	11	0.33333	2	1	2	1	3
	4	0.33333	1	2	1	2	2
	27	0.33333	3	3	3	3	1
6	8	0.33333	1	3	2	2	1
	19	0.33333	3	1	1	3	3
	15	0.33333	2	2	3	1	2
7	3	0.33333	1	1	3	2	3
	23	0.33333	3	2	2	3	2
	16	0.33333	2	3	1	1	1

8	17	0.33333	2	3	2	3	2
	24	0.33333	3	2	3	2	3
	1	0.33333	1	1	1	1	1
9	7	0.33333	1	3	1	3	3
	14	0.33333	2	2	2	2	1
	21	0.33333	3	1	3	1	2

Notice that like before, the variances are constant, but in this case smaller at 2/3, and each candidate appears once. This is an optimal design in 9 choice sets. More directly, this design could be made from the orthogonal array $3^9 9^1$ in 27 runs by using the nine-level factor as the choice set number.

Generic Attributes, a Constant Alternative, and Alternative Swapping

Now let's make a design for the same problem but this time with a constant alternative. We will first use the `%MktEx` macro just like before to make a design for the nonconstant alternatives. We will then use a DATA step to add the flags and a constant alternative.

```
title 'Generic Chair Attributes';

%mktex(3 ** 5, n=243, seed=306)

data final(drop=i);
   set design end=eof;
   retain f1-f3 1 f4 0;
   output;
   if eof then do;
      array x[9] x1-x5 f1-f4;
      do i = 1 to 9; x[i] = i le 5 or i eq 9; end;
      output;
      end;
   run;

proc print data=final(where=(x1 eq x3 and x2 eq x4 and x3 eq x5 or f4)); run;
```

Here is a sample of the observations in the candidate set.

Generic Chair Attributes

Obs	x1	x2	x3	x4	x5	f1	f2	f3	f4
1	1	1	1	1	1	1	1	1	0
31	1	2	1	2	1	1	1	1	0
61	1	3	1	3	1	1	1	1	0
92	2	1	2	1	2	1	1	1	0
122	2	2	2	2	2	1	1	1	0
152	2	3	2	3	2	1	1	1	0
183	3	1	3	1	3	1	1	1	0

```
213     3     2     3     2     3     1     1     1     0
243     3     3     3     3     3     1     1     1     0
244     1     1     1     1     1     0     0     0     1
```

The first 243 observations may be used for any of the first three alternatives and the 244*th* observation may only be used for fourth or constant alternative. In this example, the constant alternative is composed solely from the first level of each factor. Of course this could be changed depending on the situation. The `%ChoicEff` macro invocation is the same as before, except now we have four flags.

```
%choiceff(data=final, model=class(x1-x5), nsets=6, maxiter=100,
          seed=121, flags=f1-f4, beta=zero)

proc print; by set; id set; run;
```

You can see in the final design that there are now four alternatives and the last alternative in each choice set is constant and is always flagged by `f4=1`. In the interest of space, most of the iteration histories are omitted.

Generic Chair Attributes

n	Name	Beta	Label
1	x11	0	x1 1
2	x12	0	x1 2
3	x21	0	x2 1
4	x22	0	x2 2
5	x31	0	x3 1
6	x32	0	x3 2
7	x41	0	x4 1
8	x42	0	x4 2
9	x51	0	x5 1
10	x52	0	x5 2

Generic Chair Attributes

Design	Iteration	D-Efficiency	D-Error
1	0	0.424723	2.354476
	1	0.900662	1.110294
	2	0.939090	1.064861
	3	0.943548	1.059830

.
.
.

Design	Iteration	D-Efficiency	D-Error
13	0	0.494007	2.024263
	1	0.873818	1.144404
	2	0.915135	1.092735
	3	0.960392	1.041241
	4	0.999769	1.000231
	5	1.003398	0.996614

.
.
.

Design	Iteration	D-Efficiency	D-Error
100	0	0.528399	1.892509
	1	0.883854	1.131408
	2	0.924346	1.081846
	3	0.939811	1.064044
	4	0.942047	1.061518

Generic Chair Attributes

Final Results

Design	13
Choice Sets	6
Alternatives	4
D-Efficiency	1.003398
D-Error	0.996614

Generic Chair Attributes

n	Variable Name	Label	Variance	DF	Standard Error
1	x11	x1 1	1.14695	1	1.07096
2	x12	x1 2	1.33333	1	1.15470
3	x21	x2 1	1.14695	1	1.07096
4	x22	x2 2	1.33333	1	1.15470
5	x31	x3 1	1.19793	1	1.09450
6	x32	x3 2	1.27439	1	1.12889
7	x41	x4 1	1.19793	1	1.09450
8	x42	x4 2	1.27439	1	1.12889
9	x51	x5 1	1.13102	1	1.06350
10	x52	x5 2	1.27439	1	1.12889
				==	
				10	

Generic Chair Attributes

Set	Design	Efficiency	Index	Prob	n	x1	x2	x3	x4	x5	f1	f2	f3	f4
1	13	1.00340	152	0.25	289	2	3	2	3	2	1	1	1	0
	13	1.00340	213	0.25	290	3	2	3	2	3	1	1	1	0
	13	1.00340	15	0.25	291	1	1	2	2	3	1	1	1	0
	13	1.00340	244	0.25	292	1	1	1	1	1	0	0	0	1
2	13	1.00340	154	0.25	293	2	3	3	1	1	1	1	1	0
	13	1.00340	15	0.25	294	1	1	2	2	3	1	1	1	0
	13	1.00340	197	0.25	295	3	2	1	3	2	1	1	1	0
	13	1.00340	244	0.25	296	1	1	1	1	1	0	0	0	1
3	13	1.00340	108	0.25	297	2	1	3	3	3	1	1	1	0
	13	1.00340	220	0.25	298	3	3	1	2	1	1	1	1	0
	13	1.00340	38	0.25	299	1	2	2	1	2	1	1	1	0
	13	1.00340	244	0.25	300	1	1	1	1	1	0	0	0	1
4	13	1.00340	121	0.25	301	2	2	2	2	1	1	1	1	0
	13	1.00340	182	0.25	302	3	1	3	1	2	1	1	1	0
	13	1.00340	63	0.25	303	1	3	1	3	3	1	1	1	0
	13	1.00340	244	0.25	304	1	1	1	1	1	0	0	0	1
5	13	1.00340	111	0.25	305	2	2	1	1	3	1	1	1	0
	13	1.00340	77	0.25	306	1	3	3	2	2	1	1	1	0
	13	1.00340	178	0.25	307	3	1	2	3	1	1	1	1	0
	13	1.00340	244	0.25	308	1	1	1	1	1	0	0	0	1
6	13	1.00340	228	0.25	309	3	3	2	1	3	1	1	1	0
	13	1.00340	52	0.25	310	1	2	3	3	1	1	1	1	0
	13	1.00340	86	0.25	311	2	1	1	2	2	1	1	1	0
	13	1.00340	244	0.25	312	1	1	1	1	1	0	0	0	1

When there were three alternatives, each alternative had a probability of choice of 1/3, and now with four alternatives, the probability is 1/4. They are all equal because of the assumption $\boldsymbol{\beta} = \mathbf{0}$. With other assumptions about $\boldsymbol{\beta}$, typically the probabilities will not all be equal. To use this design for analysis, you would only need the variables `Set` and `x1-x5`. Since it is already in choice design format (one row per alternative), it would not need to be processed using the `%MktRoll` macro. Note that when you make designs with the `%ChoicEff` macro, the `model` statement in PROC TRANSREG should match or be no more complicated than the `model` specification that generated the design:

```
model class(x1-x5);
```

A model with fewer degrees of freedom is safe, although the design will be suboptimal. For example, if `x1-x5` are quantitative attributes, this would be safe:

```
model identity(x1-x5);
```

However, specifying interactions, or using this design in a branded study and specifying alternative-specific effects like this could lead to quite a few inestimable parameters.

```
* Bad idea for this design!!;
model class(x1-x5 x1*x2 x4*x5);

* Another bad idea for this design!!;
model class(brand)
      class(brand * x1 brand * x2 brand * x3 brand * x4 brand * x5);
```

Generic Attributes, a Constant Alternative, and Choice Set Swapping

The %ChoicEff macro can be used in a very different way. Instead of providing a candidate set of alternatives to swap in and out of the design, you can provide a candidate set of entire choice sets. For this particular example, swapping alternatives will almost certainly be better (see page 381). However, sometimes, if you need to impose restrictions on which alternative can appear with which other alternative, then you must use the set-swapping options. We will start by using the %MktEx macro to make a candidate design, with one run per choice set and one factor for each attribute of each alternative (just like we did in the vacation, fabric softener, and food examples). We will then process the candidates from one row per choice set to one row per alternative per choice set using the %MktRoll macro.

```
%mktex(3 ** 15, n=81 * 81, seed=522)

%mktkey(3 5)

data key;
   input (x1-x5) ($);
   datalines;
x1     x2     x3     x4     x5
x6     x7     x8     x9     x10
x11    x12    x13    x14    x15
.      .      .      .      .
;

%mktroll(design=randomized, key=key, out=rolled)

* Code the constant alternative;
data final;
   set rolled;
   if _alt_ = '4' then do; x1 = 1; x2 = 1; x3 = 1; x4 = 1; x5 = 1; end;
   run;

proc print; by set; id set; where set in (1, 100, 1000, 5000, 6561); run;
```

The `%MktKey` macro produced the following data set, which we copied, pasted, and augmented to make the `Key` data set.

```
x1     x2     x3     x4     x5

x1     x2     x3     x4     x5
x6     x7     x8     x9     x10
x11    x12    x13    x14    x15
```

Here are a few of the candidate choice sets.

Generic Chair Attributes

Set	_Alt_	x1	x2	x3	x4	x5
1	1	2	1	1	2	3
	2	3	3	1	1	1
	3	2	2	1	1	3
	4	1	1	1	1	1
100	1	1	2	1	2	1
	2	2	3	3	2	1
	3	2	2	3	3	2
	4	1	1	1	1	1
1000	1	2	1	2	1	3
	2	2	1	2	1	2
	3	1	3	2	2	2
	4	1	1	1	1	1
5000	1	3	1	3	2	3
	2	3	3	3	3	2
	3	1	2	1	2	3
	4	1	1	1	1	1
6561	1	1	3	1	2	2
	2	3	2	2	2	2
	3	1	3	3	1	3
	4	1	1	1	1	1

Next, we will then run the `%ChoicEff` macro, only this time we will specify `nalts=4` instead of `flags=f1-f4`. Since there are no alternative flag variables to count, we have to tell the macro how many alternatives are in each choice set. We will also ask for fewer iterations since the candidate set is large.

```
%choiceff(data=final, model=class(x1-x5), nsets=6, nalts=4, maxiter=10,
          beta=zero, seed=109)
```

Generic Chair Attributes

n	Name	Beta	Label
1	x11	0	x1 1
2	x12	0	x1 2
3	x21	0	x2 1
4	x22	0	x2 2
5	x31	0	x3 1
6	x32	0	x3 2
7	x41	0	x4 1
8	x42	0	x4 2
9	x51	0	x5 1
10	x52	0	x5 2

Generic Chair Attributes

Design	Iteration	D-Efficiency	D-Error
1	0	0.536166	1.865093
	1	0.848201	1.178966
	2	0.872298	1.146398
	3	0.872298	1.146398

.
.
.

Design	Iteration	D-Efficiency	D-Error
5	0	0.529592	1.888245
	1	0.836422	1.195568
	2	0.861051	1.161372
	3	0.898936	1.112426
	4	0.904411	1.105692
	5	0.904411	1.105692

.
.
.

Design	Iteration	D-Efficiency	D-Error
10	0	0.539774	1.852627
	1	0.820582	1.218648
	2	0.846874	1.180814
	3	0.869219	1.150458
	4	0.869219	1.150458

Generic Chair Attributes

Final Results

Design	5
Choice Sets	6
Alternatives	4
D-Efficiency	0.904411
D-Error	1.105692

Generic Chair Attributes

n	Variable Name	Label	Variance	DF	Standard Error
1	x11	x1 1	1.14609	1	1.07056
2	x12	x1 2	2.32530	1	1.52489
3	x21	x2 1	1.48741	1	1.21959
4	x22	x2 2	1.95354	1	1.39769
5	x31	x3 1	1.16334	1	1.07858
6	x32	x3 2	1.50116	1	1.22522
7	x41	x4 1	1.34713	1	1.16066
8	x42	x4 2	1.35845	1	1.16552
9	x51	x5 1	1.27405	1	1.12874
10	x52	x5 2	1.54939	1	1.24475
				==	
				10	

This design is less *D*-efficient than we found using the alternative-swapping algorithm, so we will not use it.

Design Algorithm Comparisons

It is instructive to compare the three approaches outlined in this chapter in the context of this problem. There are $3^{3\times5} = 14,348,907$ choice sets for this problem (three-level factors and 3 alternatives times 5 factors per alternative). If we were to use the pure linear design approach using the `%MktEx` macro, we could never begin to consider all possible candidate choice sets. Similarly, with the choice-set-swapping algorithm of the `%ChoicEff` macro, we could never begin to consider all possible candidate choice sets. Furthermore, with the linear design approach, we could not create a design with six choice sets since the minimum size is $2 \times 15 + 1 = 31$. Now consider the alternative-swapping algorithm. It uses at most a candidate set with only 244 observations ($3^5 + 1$). From it, every possible choice set can potentially be constructed, although the macro will only consider a tiny fraction of the possibilities. Hence, the alternative swapping will usually find a better design, because the candidate set does not limit it.

Both uses of the `%ChoicEff` macro have the advantage that they are explicitly minimizing the variances of the parameter estimates given a model and a $\boldsymbol{\beta}$ vector. They can be used to produce smaller, more specialized, and better designs. However, if the $\boldsymbol{\beta}$ vector or model is badly misspecified, the designs could be horrible. How badly do things have to be misspecified before you will have problems? Who

knows. More research is needed. In contrast, the linear model `%MktEx` approach is very conservative and safe in that it should let you specify a very general model and still produce estimable parameters. The cost is you may be using many more choice sets than you need, particularly for nonbranded generic attributes. If you really have some information about your parameters, you should use them to produce a smaller and better design. However, if you have little or no information about parameters and if you anticipate specifying very general models like mother logit, then you probably want to use the linear design approach.

Initial Designs

This section illustrates some design strategies that involve improving on or augmenting initial designs. We will not actually use any designs from this section.

Improving an Existing Design

Sometimes, it is useful to try to improve an existing design. In this example, we use the `%MktEx` macro to create a design in 80 runs for 25 four-level factors. In the next step, we specify `init=`, and the macro goes straight into the design refinement history seeking to refine the input design. You might want to do this for example whenever you have a good, but not 100% *D*-efficient design, and you are willing to wait a few minutes to see if the macro can make it any better.

```
title 'Try to Improve an Existing Design';

%mktex(4 ** 25, n=80, seed=368)
%mktex(4 ** 25, n=80, seed=306, init=design, maxtime=20)
```

Here is the *D*-efficiency of the final design from the first step.

Try to Improve an Existing Design

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	91.4106	83.9583	97.6073	0.9747

This is a large problem. One in which the `maxtime=` option may cause the macro to stop before it reaches the maximum number of iterations. Running a second refinement step might help improve the design by adding a few more iterations. Here are the results from the second step.

Design Refinement History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
0	Initial	91.4106	91.4106	Ini
1	Start	90.0771		Pre,Mut,Ann
1	End	91.3476		

```
2      Start       88.8927                Pre,Mut,Ann
2     36  12       91.4181       91.4181
2     56   6       91.4285       91.4285
2      7  17       91.4372       91.4372
2     13  10       91.4373       91.4373
2     23  18       91.4404       91.4404
2     17  16       91.4445       91.4445
2     34   6       91.4572       91.4572
2     56  19       91.4673       91.4673
2     56  21       91.4768       91.4768
2     77   1       91.4821       91.4821
2     23  18       91.4827       91.4827
2     48   3       91.4848       91.4848
2     48   9       91.4863       91.4863
2     40  18       91.4863       91.4863
2        End       91.4863
.
.
.
6      Start       90.2194                Pre,Mut,Ann
6     63  19       91.5811       91.5811
6     68  18       91.5835       91.5835
6        End       91.5751


7      Start       89.4607                Pre,Mut,Ann
7     25   4       91.5851       91.5851
7     34   7       91.5902       91.5902
7     47   2       91.5913       91.5913
7     48  14       91.5930       91.5930
7     56   4       91.5955       91.5955
7     56  15       91.5999       91.5999
7     60   6       91.6142       91.6142
7     68   7       91.6172       91.6172
7     78   5       91.6172       91.6172
7     13  21       91.6249       91.6249
7     18  19       91.6249       91.6249
7     43  10       91.6249       91.6249
7     48  14       91.6282       91.6282
7     50  22       91.6408       91.6408
7     61   4       91.6417       91.6417
7     80  15       91.6430       91.6430
7     46  12       91.6430       91.6430
7     48   6       91.6430       91.6430
7        End       91.6430
.
.
.
```

```
10      Start       89.8707                Pre,Mut,Ann
10        End       91.6629
 .
 .
 .
```

Try to Improve an Existing Design

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	91.7082	83.9853	97.5951	0.9747

The macro skips the normal first steps, algorithm search and design search, and goes straight into the design refinement search. In this example a small improvement was found, although often, no improvement is found.

When Some Choice Sets are Fixed in Advance

Sometimes certain runs or choice sets are fixed in advance and must be included in the design. The `%MktEx` macro can be used to efficiently augment a starting design with other choice sets. Suppose that you can make a choice design from the L_{36} ($2^{11}3^{12}$). In addition, you want to optimally add four more choice sets to use as holdouts. First we will look at how to do this using the `fixed=` option. This option can be used for fairly general design augmentation and refinement problems. On page 389, we will see an easier way to handle this particular problem using the `holdouts=` option.

You can create the design in 36 runs as before. Next, a DATA step is used to add a flag variable `f` that has values of 1 for the original 36 runs. In addition, four more runs are added (just copies of the last run) but with a flag value of missing. When this variable is specified on the `fixed=f` option, it indicates that the first 36 runs of the `init=init` design are *fixed*—they may not change. The remaining 4 runs are to be randomly initialized and optimally refined to maximize the *D*-efficiency of the overall 40-run design. We specified `options=nosort` so that the additional runs would stay at the end of the design.

```
title 'Augment a Design';

%mktex(n=36, seed=292)
data init;
   set randomized end = eof;
   f = 1;
   output;
```

```
      if eof then do;
         f = .;
         do i = 1 to 4; output; end;
         drop i;
         end;
      run;
   proc print; run;

   %mktex(2 ** 11 3 ** 12, n=40, init=init, fixed=f, seed=513, options=nosort)

   proc print; run;
```

Here is the initial design.

Augment a Design

0										x	x	x	x	x	x	x	x	x	x	x	x	x	
b	x	x	x	x	x	x	x	x	x	1	1	1	1	1	1	1	1	1	1	2	2	2	2	
s	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	f
1	2	1	2	2	2	2	1	1	1	2	1	2	3	1	1	1	2	3	2	3	1	3	2	1
2	2	1	1	2	2	1	2	1	2	1	2	3	1	2	1	3	2	3	1	2	2	3	1	1
3	2	2	1	1	1	1	1	1	1	2	2	1	2	3	1	1	2	2	3	2	3	3	3	1
4	1	1	1	1	2	1	1	2	2	2	1	1	2	3	2	3	2	3	2	1	2	1	2	1
5	1	2	1	2	2	2	1	2	1	1	2	1	1	2	2	2	1	3	3	3	3	3	2	1
6	2	1	1	2	2	1	2	1	2	1	2	1	2	1	3	2	1	2	2	3	1	1	3	1
7	1	2	2	1	2	1	2	1	1	1	1	3	3	2	2	1	1	2	2	1	2	3	3	1
8	2	2	1	1	1	1	1	1	1	2	2	2	3	2	3	3	1	1	1	3	2	1	2	1
9	2	2	2	2	1	1	1	2	2	1	1	2	2	2	3	3	3	3	3	1	1	3	3	1
10	2	2	2	2	1	1	1	2	2	1	1	1	1	3	1	1	1	1	2	3	2	2	1	1
11	1	1	2	1	1	2	1	1	2	1	2	3	2	2	2	1	2	1	3	3	1	1	1	1
12	2	2	2	1	2	2	2	2	2	2	2	1	3	3	2	3	3	2	1	3	1	3	1	1
13	1	2	1	2	2	2	1	2	1	1	2	3	3	3	3	3	2	1	2	2	1	2	3	1
14	1	1	2	1	1	2	1	1	2	1	2	1	3	1	1	3	1	3	1	1	3	2	3	1
15	2	1	1	1	1	2	2	2	1	1	1	1	3	2	3	1	3	3	2	2	3	1	1	1
16	1	2	2	1	2	1	2	1	1	1	1	2	2	3	3	2	2	3	1	3	3	2	1	1
17	2	1	1	1	1	2	2	2	1	1	1	3	2	3	1	2	1	1	1	1	1	3	2	1
18	1	1	1	1	2	1	1	2	2	2	1	3	1	1	3	1	3	1	1	3	3	3	3	1
19	2	1	2	2	2	2	1	1	1	2	1	1	2	2	2	2	3	1	1	2	2	2	3	1
20	2	1	1	1	1	2	2	2	1	1	1	2	1	1	2	3	2	2	3	3	2	2	3	1
21	1	2	1	2	1	2	2	1	2	2	1	1	3	1	3	2	2	1	3	1	2	3	1	1
22	1	1	2	1	1	2	1	1	2	1	2	2	1	3	3	2	3	2	2	2	2	3	2	1
23	2	2	2	1	2	2	2	2	2	2	2	3	2	1	3	1	1	3	3	2	2	2	2	1
24	1	2	2	1	2	1	2	1	1	1	1	1	1	1	1	3	3	1	3	2	1	1	2	1
25	1	1	2	2	1	1	2	2	1	2	2	1	1	2	3	1	2	2	1	1	1	2	2	1
26	1	1	2	2	1	1	2	2	1	2	2	2	2	1	2	3	1	1	2	2	3	3	1	1
27	1	1	2	2	1	1	2	2	1	2	2	3	3	3	1	2	3	3	3	3	2	1	3	1
28	1	2	1	2	2	2	1	2	1	1	2	2	2	1	1	1	3	2	1	1	2	1	1	1
29	2	1	2	2	2	2	1	1	1	2	1	3	1	3	3	3	1	2	3	1	3	1	1	1
30	2	2	1	1	1	1	1	1	1	2	2	3	1	1	2	2	3	3	2	1	1	2	1	1

31	1	2	1	2	1	2	2	1	2	2	1	2	1	3	2	1	1	3	1	2	1	1	3	1
32	2	1	1	2	2	1	2	1	2	1	2	2	3	3	2	1	3	1	3	1	3	2	2	1
33	2	2	2	2	1	1	1	2	2	1	1	3	3	1	2	2	2	2	1	2	3	1	2	1
34	1	2	1	2	1	2	2	1	2	2	1	3	2	2	1	3	3	2	2	3	3	2	2	1
35	2	2	2	1	2	2	2	2	2	2	2	2	1	2	1	2	2	1	2	1	3	1	3	1
36	1	1	1	1	2	1	1	2	2	2	1	2	3	2	1	2	1	2	3	2	1	2	1	1
37	1	1	1	1	2	1	1	2	2	2	1	2	3	2	1	2	1	2	3	2	1	2	1	.
38	1	1	1	1	2	1	1	2	2	2	1	2	3	2	1	2	1	2	3	2	1	2	1	.
39	1	1	1	1	2	1	1	2	2	2	1	2	3	2	1	2	1	2	3	2	1	2	1	.
40	1	1	1	1	2	1	1	2	2	2	1	2	3	2	1	2	1	2	3	2	1	2	1	.

Here is the iteration history for the augmentation.

Augment a Design

Design Refinement History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
0	Initial	97.0559	97.0559	Ini
1	Start	97.0788	97.0788	Pre,Mut,Ann
1	37 5	97.0805	97.0805	
1	37 8	97.0816	97.0816	
1	37 9	97.1049	97.1049	
1	37 11	97.1133	97.1133	
1	37 13	97.1177	97.1177	
1	38 1	97.1410	97.1410	
1	38 2	97.1605	97.1605	
1	38 3	97.1729	97.1729	
1	38 4	97.1822	97.1822	
1	38 6	97.1905	97.1905	
1	38 8	97.1944	97.1944	
1	39 2	97.1991	97.1991	
1	39 13	97.2007	97.2007	
1	39 19	97.2007	97.2007	
1	40 9	97.2007	97.2007	
1	40 10	97.2007	97.2007	
1	37 18	97.2023	97.2023	
1	37 3	97.2028	97.2028	
1	37 4	97.2028	97.2028	
1	37 23	97.2043	97.2043	
1	End	97.2043		

```
2      Start       97.2043       97.2043  Pre,Mut,Ann
2     40  21       97.2043       97.2043
2     38   2       97.2043       97.2043
2     40   9       97.2043       97.2043
2     40  21       97.2043       97.2043
2        End       97.2043

3      Start       97.2043       97.2043  Pre,Mut,Ann
3        End       97.2043

4      Start       97.2002                Pre,Mut,Ann
4     39  12       97.2043       97.2043
4     39  23       97.2043       97.2043
4     39  16       97.2043       97.2043
4        End       97.2043

5      Start       97.2043       97.2043  Pre,Mut,Ann
5     37   3       97.2043       97.2043
5     37  15       97.2043       97.2043
5        End       97.2043

6      Start       97.2043       97.2043  Pre,Mut,Ann
6     40   1       97.2043       97.2043
6     39  16       97.2043       97.2043
6     38  16       97.2043       97.2043
6        End       97.2043
NOTE: Stopping since it appears that no improvement is possible.
```

Notice that the macro goes straight into the design refinement stage. Also notice that in the iteration history, only rows 37 through 40 are changed. Here is the design. The last four rows are the holdouts.

Augment a Design

O										x	x	x	x	x	x	x	x	x	x	x	x	x	
b	x	x	x	x	x	x	x	x	x	1	1	1	1	1	1	1	1	1	1	2	2	2	2	
s	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	f
1	2	1	2	2	2	2	1	1	1	2	1	2	3	1	1	1	2	3	2	3	1	3	2	1
2	2	1	1	2	2	1	2	1	2	1	2	3	1	2	1	3	2	3	1	2	2	3	1	1
3	2	2	1	1	1	1	1	1	1	2	2	1	2	3	1	1	2	2	3	2	3	3	3	1
4	1	1	1	1	2	1	1	2	2	2	1	1	2	3	2	3	2	3	2	1	2	1	2	1
5	1	2	1	2	2	2	1	2	1	1	2	1	1	2	2	2	1	3	3	3	3	3	2	1
6	2	1	1	2	2	1	2	1	2	1	2	1	2	1	3	2	1	2	2	3	1	1	3	1
7	1	2	2	1	2	1	2	1	1	1	1	3	3	2	2	1	1	2	2	1	2	3	3	1
8	2	2	1	1	1	1	1	1	1	2	2	2	3	2	3	3	1	1	1	3	2	1	2	1
9	2	2	2	2	1	1	1	2	2	1	1	2	2	2	3	3	3	3	3	1	1	3	3	1
10	2	2	2	2	1	1	1	2	2	1	1	1	1	3	1	1	1	1	2	3	2	2	1	1

11	1	1	2	1	1	2	1	1	2	1	2	3	2	2	2	1	2	1	3	3	1	1	1	1
12	2	2	2	1	2	2	2	2	2	2	2	1	3	3	2	3	3	2	1	3	1	3	1	1
13	1	2	1	2	2	2	1	2	1	1	2	3	3	3	3	3	2	1	2	2	1	2	3	1
14	1	1	2	1	1	2	1	1	2	1	2	1	3	1	1	3	1	3	1	1	3	2	3	1
15	2	1	1	1	1	2	2	2	1	1	1	1	3	2	3	1	3	3	2	2	3	1	1	1
16	1	2	2	1	2	1	2	1	1	1	1	2	2	3	3	2	2	3	1	3	3	2	1	1
17	2	1	1	1	1	2	2	2	1	1	1	3	2	3	1	2	1	1	1	1	1	3	2	1
18	1	1	1	1	2	1	1	2	2	2	1	3	1	1	3	1	3	1	1	3	3	3	3	1
19	2	1	2	2	2	2	1	1	1	2	1	1	2	2	2	2	3	1	1	2	2	2	3	1
20	2	1	1	1	1	2	2	2	1	1	1	2	1	1	2	3	2	2	3	3	2	2	3	1
21	1	2	1	2	1	2	2	1	2	2	1	1	3	1	3	2	2	1	3	1	2	3	1	1
22	1	1	2	1	1	2	1	1	2	1	2	2	1	3	3	2	3	2	2	2	2	3	2	1
23	2	2	2	1	2	2	2	2	2	2	2	3	2	1	3	1	1	3	3	2	2	2	2	1
24	1	2	2	1	2	1	2	1	1	1	1	1	1	1	1	3	3	1	3	2	1	1	2	1
25	1	1	2	2	1	1	2	2	1	2	2	1	1	2	3	1	2	2	1	1	1	2	2	1
26	1	1	2	2	1	1	2	2	1	2	2	2	2	1	2	3	1	1	2	2	3	3	1	1
27	1	1	2	2	1	1	2	2	1	2	2	3	3	3	1	2	3	3	3	3	2	1	3	1
28	1	2	1	2	2	2	1	2	1	1	2	2	2	1	1	1	3	2	1	1	2	1	1	1
29	2	1	2	2	2	2	1	1	1	2	1	3	1	3	3	3	1	2	3	1	3	1	1	1
30	2	2	1	1	1	1	1	1	1	2	2	3	1	1	2	2	3	3	2	1	1	2	1	1
31	1	2	1	2	1	2	2	1	2	2	1	2	1	3	2	1	1	3	1	2	1	1	3	1
32	2	1	1	2	2	1	2	1	2	1	2	2	3	3	2	1	3	1	3	1	3	2	2	1
33	2	2	2	2	1	1	1	2	2	1	1	3	3	1	2	2	2	2	1	2	3	1	2	1
34	1	2	1	2	1	2	2	1	2	2	1	3	2	2	1	3	3	2	2	3	3	2	2	1
35	2	2	2	1	2	2	2	2	2	2	2	2	1	2	1	2	2	1	2	1	3	1	3	1
36	1	1	1	1	2	1	1	2	2	2	1	2	3	2	1	2	1	2	3	2	1	2	1	1
37	1	2	2	2	2	1	1	2	1	1	1	3	2	3	3	2	1	2	1	3	1	3	1	.
38	2	2	2	1	2	1	2	2	1	1	1	2	2	1	2	3	3	3	1	1	3	2	3	.
39	1	2	2	2	2	2	2	1	2	2	2	3	3	3	1	2	1	3	2	1	3	2	3	.
40	2	1	1	1	1	1	1	2	2	2	1	2	1	2	3	2	1	3	2	1	1	2	1	.

This code does the same thing only using the `holdouts=4` option instead.

```
title 'Augment a Design';

%mktex(n=36, seed=292)
%mktex(2 ** 11 3 ** 12, n=40, init=randomized,
       holdouts=4, seed=513, options=nosort)

proc print data=design(firstobs=37); run;
```

Here are the holdout observations, which are the same as we saw previously.

Augment a Design

O																							
b	x	x	x	x	x	x	x	x	x	1	1	1	1	1	1	1	1	1	1	2	2	2	2	
s	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	w
37	1	2	2	2	2	1	1	2	1	1	1	3	2	3	3	2	1	2	1	3	1	3	1	.
38	2	2	2	1	2	1	2	2	1	1	1	2	2	1	2	3	3	3	1	1	3	2	3	.
39	1	2	2	2	2	2	2	1	2	2	2	3	3	3	1	2	1	3	2	1	3	2	3	.
40	2	1	1	1	1	1	1	2	2	2	1	2	1	2	3	2	1	3	2	1	1	2	1	.

The %MktEx macro provides another way to use initial designs. The initial design may indicate that part of the design is fixed and may not change and a different part should be randomly initialized and may change. The initial design can have three types of values:

- positive integers are fixed and constant and will not change throughout the course of the iterations.
- zero and missing values are replaced by random values at the start of each new design search and can change throughout the course of the iterations.
- negative values are replaced by their absolute value at the start of each new design attempt and can change throughout the course of the iterations.

Returning to the example of making the design 4^{25} in 80 runs, we could do it in two steps. The maximum number of four-level factors in 80 runs is 11. If it is important that some factors be orthogonal, we could first make an orthogonal array with 11 four-level factors and then append 14 more nonorthogonal-factors. Here is the code.

```
title 'Differential Design Initialization';

%mktex(4 ** 11, n=80)

data init;
   set design;
   retain x12-x25 .;
   run;

%mktex(4 ** 25, n=80, init=init, seed=472)

%mkteval;
```

The initial design consists of the orthogonal array with 11 columns followed by 14 more columns that are all missing. When %MktEx sees missing values in the initial design, it holds all the nonmissing values fixed. Then it randomly replaces the missing values and uses the coordinate-exchange algorithm to refine the last columns. The final design is slightly less D-efficient than we saw previously, but the first 11 columns are orthogonal. The remaining columns are all slightly correlated with themselves and with the first columns. Here is the final efficiency table.

Differential Design Initialization

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	89.4216	78.9807	97.7767	0.9747

Here are the canonical correlations.

Differential Design Initialization
Canonical Correlations Between the Factors
There are 0 Canonical Correlations Greater Than 0.316

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13
x1	1	0	0	0	0	0	0	0	0	0	0	0.15	0.17
x2	0	1	0	0	0	0	0	0	0	0	0	0.17	0.11
x3	0	0	1	0	0	0	0	0	0	0	0	0.20	0.13
x4	0	0	0	1	0	0	0	0	0	0	0	0.16	0.12
x5	0	0	0	0	1	0	0	0	0	0	0	0.16	0.18
x6	0	0	0	0	0	1	0	0	0	0	0	0.12	0.12
x7	0	0	0	0	0	0	1	0	0	0	0	0.18	0.18
x8	0	0	0	0	0	0	0	1	0	0	0	0.16	0.12
x9	0	0	0	0	0	0	0	0	1	0	0	0.15	0.14
x10	0	0	0	0	0	0	0	0	0	1	0	0.12	0.18
x11	0	0	0	0	0	0	0	0	0	0	1	0.16	0.12
x12	0.15	0.17	0.20	0.16	0.16	0.12	0.18	0.16	0.15	0.12	0.16	1	0.10
x13	0.17	0.11	0.13	0.12	0.18	0.12	0.18	0.12	0.14	0.18	0.12	0.10	1
x14	0.18	0.14	0.10	0.14	0.19	0.20	0.18	0.19	0.14	0.15	0.11	0.16	0.15
x15	0.16	0.25	0.15	0.18	0.17	0.17	0.15	0.12	0.24	0.13	0.16	0.14	0.13
x16	0.10	0.23	0.14	0.15	0.17	0.16	0.15	0.13	0.10	0.16	0.21	0.12	0.11
x17	0.20	0.20	0.15	0.07	0.12	0.17	0.16	0.12	0.13	0.20	0.14	0.12	0.11
x18	0.09	0.18	0.10	0.16	0.15	0.13	0.13	0.17	0.10	0.15	0.14	0.15	0.09
x19	0.17	0.18	0.17	0.16	0.13	0.17	0.19	0.21	0.15	0.15	0.16	0.14	0.15
x20	0.23	0.09	0.11	0.14	0.17	0.21	0.25	0.27	0.06	0.17	0.15	0.14	0.11
x21	0.12	0.09	0.15	0.17	0.17	0.15	0.20	0.14	0.20	0.17	0.16	0.18	0.12
x22	0.17	0.14	0.07	0.18	0.15	0.12	0.11	0.16	0.17	0.09	0.17	0.12	0.16
x23	0.19	0.15	0.15	0.18	0.09	0.10	0.16	0.16	0.12	0.16	0.16	0.13	0.12
x24	0.18	0.16	0.14	0.20	0.09	0.22	0.13	0.13	0.18	0.13	0.18	0.14	0.11
x25	0.14	0.18	0.09	0.13	0.19	0.18	0.17	0.19	0.16	0.20	0.14	0.15	0.12

	x14	x15	x16	x17	x18	x19	x20	x21	x22	x23	x24	x25
x1	0.18	0.16	0.10	0.20	0.09	0.17	0.23	0.12	0.17	0.19	0.18	0.14
x2	0.14	0.25	0.23	0.20	0.18	0.18	0.09	0.09	0.14	0.15	0.16	0.18
x3	0.10	0.15	0.14	0.15	0.10	0.17	0.11	0.15	0.07	0.15	0.14	0.09
x4	0.14	0.18	0.15	0.07	0.16	0.16	0.14	0.17	0.18	0.18	0.20	0.13
x5	0.19	0.17	0.17	0.12	0.15	0.13	0.17	0.17	0.15	0.09	0.09	0.19
x6	0.20	0.17	0.16	0.17	0.13	0.17	0.21	0.15	0.12	0.10	0.22	0.18
x7	0.18	0.15	0.15	0.16	0.13	0.19	0.25	0.20	0.11	0.16	0.13	0.17
x8	0.19	0.12	0.13	0.12	0.17	0.21	0.27	0.14	0.16	0.16	0.13	0.19
x9	0.14	0.24	0.10	0.13	0.10	0.15	0.06	0.20	0.17	0.12	0.18	0.16
x10	0.15	0.13	0.16	0.20	0.15	0.15	0.17	0.17	0.09	0.16	0.13	0.20
x11	0.11	0.16	0.21	0.14	0.14	0.16	0.15	0.16	0.17	0.16	0.18	0.14
x12	0.16	0.14	0.12	0.12	0.15	0.14	0.14	0.18	0.12	0.13	0.14	0.15
x13	0.15	0.13	0.11	0.11	0.09	0.15	0.11	0.12	0.16	0.12	0.11	0.12
x14	1	0.22	0.14	0.12	0.12	0.21	0.13	0.13	0.13	0.20	0.18	0.12
x15	0.22	1	0.12	0.09	0.19	0.09	0.09	0.13	0.10	0.15	0.16	0.13
x16	0.14	0.12	1	0.15	0.14	0.14	0.18	0.13	0.12	0.10	0.17	0.18
x17	0.12	0.09	0.15	1	0.09	0.16	0.18	0.09	0.15	0.12	0.18	0.12
x18	0.12	0.19	0.14	0.09	1	0.14	0.13	0.20	0.15	0.13	0.14	0.17
x19	0.21	0.09	0.14	0.16	0.14	1	0.13	0.11	0.10	0.11	0.16	0.12
x20	0.13	0.09	0.18	0.18	0.13	0.13	1	0.10	0.11	0.11	0.18	0.11
x21	0.13	0.13	0.13	0.09	0.20	0.11	0.10	1	0.21	0.15	0.23	0.09
x22	0.13	0.10	0.12	0.15	0.15	0.10	0.11	0.21	1	0.19	0.18	0.22
x23	0.20	0.15	0.10	0.12	0.13	0.11	0.11	0.15	0.19	1	0.13	0.13
x24	0.18	0.16	0.17	0.18	0.14	0.16	0.18	0.23	0.18	0.13	1	0.12
x25	0.12	0.13	0.18	0.12	0.17	0.12	0.11	0.09	0.22	0.13	0.12	1

Here are the one-way frequencies.

```
                Differential Design Initialization
                      Summary of Frequencies
          There are 0 Canonical Correlations Greater Than 0.316
                    * - Indicates Unequal Frequencies

                           Frequencies

                x1         20 20 20 20
                x2         20 20 20 20
                x3         20 20 20 20
                x4         20 20 20 20
                x5         20 20 20 20
                x6         20 20 20 20
                x7         20 20 20 20
                x8         20 20 20 20
                x9         20 20 20 20
                x10        20 20 20 20
                x11        20 20 20 20
```

```
*    x12        19 22 19 20
*    x13        21 19 20 20
*    x14        18 20 21 21
*    x15        18 21 22 19
*    x16        19 18 22 21
*    x17        21 19 18 22
*    x18        21 20 20 19
*    x19        21 18 18 23
*    x20        20 20 21 19
*    x21        18 20 23 19
*    x22        20 18 22 20
*    x23        19 20 21 20
*    x24        23 17 20 20
*    x25        21 19 19 21
```

We could run one more refinement on this design and force `x12-x25` to be balanced. We will need to make a new initial design using our current design. This time, we will make `x12-x25` negative. Then `x1-x11` will not change, the absolute values of `x12-x25` will be used as initial values, but `x12-x25` will still be allowed to change. Then we can use the `balance=1` option with `%MktEx` to make a design that is better balanced. Usually, when you are creating a design, you should specify `mintry=` with `balance=`, however, since we are refining not creating a design it is not necessary. Here is the code.

```
data init(drop=j);
   set design;
   array x[25];
   do j = 12 to 25; x[j] = -x[j]; end;
   run;

%mktex(4 ** 25, n=80, init=init, seed=472, balance=1)

%mkteval;
```

Here is the final *D*-efficiency, which again, is a bit lower than we say previously.

Differential Design Initialization

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	86.1917	71.1301	97.9379	0.9747

Here are the canonical correlations.

Differential Design Initialization
Canonical Correlations Between the Factors
There are 0 Canonical Correlations Greater Than 0.316

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13
x1	1	0	0	0	0	0	0	0	0	0	0	0.20	0.15
x2	0	1	0	0	0	0	0	0	0	0	0	0.15	0.13
x3	0	0	1	0	0	0	0	0	0	0	0	0.15	0.12
x4	0	0	0	1	0	0	0	0	0	0	0	0.20	0.15
x5	0	0	0	0	1	0	0	0	0	0	0	0.20	0.24
x6	0	0	0	0	0	1	0	0	0	0	0	0.13	0.15
x7	0	0	0	0	0	0	1	0	0	0	0	0.17	0.19
x8	0	0	0	0	0	0	0	1	0	0	0	0.13	0.15
x9	0	0	0	0	0	0	0	0	1	0	0	0.12	0.16
x10	0	0	0	0	0	0	0	0	0	1	0	0.13	0.17
x11	0	0	0	0	0	0	0	0	0	0	1	0.20	0.13
x12	0.20	0.15	0.15	0.20	0.20	0.13	0.17	0.13	0.12	0.13	0.20	1	0.14
x13	0.15	0.13	0.12	0.15	0.24	0.15	0.19	0.15	0.16	0.17	0.13	0.14	1
x14	0.20	0.10	0.09	0.13	0.18	0.22	0.17	0.16	0.13	0.16	0.09	0.20	0.15
x15	0.21	0.23	0.22	0.25	0.20	0.24	0.17	0.15	0.22	0.13	0.22	0.18	0.16
x16	0.09	0.20	0.17	0.18	0.12	0.18	0.17	0.17	0.10	0.16	0.20	0.09	0.14
x17	0.20	0.23	0.15	0.15	0.09	0.22	0.20	0.10	0.15	0.17	0.15	0.15	0.15
x18	0.10	0.23	0.10	0.16	0.15	0.15	0.18	0.17	0.10	0.15	0.15	0.15	0.12
x19	0.17	0.20	0.18	0.14	0.17	0.15	0.23	0.16	0.12	0.19	0.17	0.16	0.25
x20	0.26	0.10	0.10	0.17	0.17	0.17	0.24	0.23	0.09	0.17	0.19	0.12	0.10
x21	0.20	0.13	0.20	0.19	0.19	0.19	0.20	0.20	0.15	0.14	0.17	0.14	0.14
x22	0.17	0.16	0.10	0.23	0.14	0.16	0.10	0.16	0.15	0.10	0.14	0.15	0.15
x23	0.24	0.18	0.14	0.18	0.09	0.10	0.15	0.20	0.13	0.22	0.19	0.19	0.12
x24	0.24	0.20	0.27	0.18	0.15	0.29	0.12	0.12	0.18	0.20	0.17	0.20	0.14
x25	0.20	0.16	0.09	0.15	0.23	0.19	0.17	0.18	0.17	0.22	0.15	0.16	0.12

	x14	x15	x16	x17	x18	x19	x20	x21	x22	x23	x24	x25
x1	0.20	0.21	0.09	0.20	0.10	0.17	0.26	0.20	0.17	0.24	0.24	0.20
x2	0.10	0.23	0.20	0.23	0.23	0.20	0.10	0.13	0.16	0.18	0.20	0.16
x3	0.09	0.22	0.17	0.15	0.10	0.18	0.10	0.20	0.10	0.14	0.27	0.09
x4	0.13	0.25	0.18	0.15	0.16	0.14	0.17	0.19	0.23	0.18	0.18	0.15
x5	0.18	0.20	0.12	0.09	0.15	0.17	0.17	0.19	0.14	0.09	0.15	0.23
x6	0.22	0.24	0.18	0.22	0.15	0.15	0.17	0.19	0.16	0.10	0.29	0.19
x7	0.17	0.17	0.17	0.20	0.18	0.23	0.24	0.20	0.10	0.15	0.12	0.17
x8	0.16	0.15	0.17	0.10	0.17	0.16	0.23	0.20	0.16	0.20	0.12	0.18
x9	0.13	0.22	0.10	0.15	0.10	0.12	0.09	0.15	0.15	0.13	0.18	0.17
x10	0.16	0.13	0.16	0.17	0.15	0.19	0.17	0.14	0.10	0.22	0.20	0.22
x11	0.09	0.22	0.20	0.15	0.15	0.17	0.19	0.17	0.14	0.19	0.17	0.15
x12	0.20	0.18	0.09	0.15	0.15	0.16	0.12	0.14	0.15	0.19	0.20	0.16
x13	0.15	0.16	0.14	0.15	0.12	0.25	0.10	0.14	0.15	0.12	0.14	0.12
x14	1	0.20	0.20	0.12	0.18	0.24	0.10	0.25	0.14	0.18	0.17	0.13
x15	0.20	1	0.15	0.15	0.15	0.12	0.09	0.15	0.14	0.12	0.21	0.18
x16	0.20	0.15	1	0.15	0.15	0.12	0.14	0.13	0.09	0.15	0.23	0.17
x17	0.12	0.15	0.15	1	0.20	0.15	0.19	0.24	0.17	0.14	0.26	0.19
x18	0.18	0.15	0.15	0.20	1	0.17	0.17	0.22	0.15	0.09	0.17	0.22
x19	0.24	0.12	0.12	0.15	0.17	1	0.09	0.15	0.17	0.14	0.15	0.13
x20	0.10	0.09	0.14	0.19	0.17	0.09	1	0.12	0.20	0.18	0.10	0.09
x21	0.25	0.15	0.13	0.24	0.22	0.15	0.12	1	0.15	0.12	0.28	0.12
x22	0.14	0.14	0.09	0.17	0.15	0.17	0.20	0.15	1	0.22	0.17	0.27
x23	0.18	0.12	0.15	0.14	0.09	0.14	0.18	0.12	0.22	1	0.21	0.18
x24	0.17	0.21	0.23	0.26	0.17	0.15	0.10	0.28	0.17	0.21	1	0.14
x25	0.13	0.18	0.17	0.19	0.22	0.13	0.09	0.12	0.27	0.18	0.14	1

Here are the one-way frequencies, which are now all perfect.

Differential Design Initialization

Summary of Frequencies
There are 0 Canonical Correlations Greater Than 0.316
* - Indicates Unequal Frequencies

Frequencies

x1	20 20 20 20
x2	20 20 20 20
x3	20 20 20 20
x4	20 20 20 20
x5	20 20 20 20
x6	20 20 20 20
x7	20 20 20 20
x8	20 20 20 20
x9	20 20 20 20
x10	20 20 20 20
x11	20 20 20 20
x12	20 20 20 20
x13	20 20 20 20
x14	20 20 20 20
x15	20 20 20 20
x16	20 20 20 20
x17	20 20 20 20
x18	20 20 20 20
x19	20 20 20 20
x20	20 20 20 20
x21	20 20 20 20
x22	20 20 20 20
x23	20 20 20 20
x24	20 20 20 20
x25	20 20 20 20

You can initialize any part of the design to positive integers (fixed), any other part to zero or missing (randomly initialize and change), and any other part to negative (do not reinitialize but change is allowed). This capability gives you very flexible control over the components of your design.

Partial Profiles and Restrictions

Partial-profile designs (Chrzan and Elrod, 1995) are used when there are many attributes but no more than a few of them are allowed to vary at any one time. Chrzan and Elrod show an example where respondents must choose between vacuum cleaners that vary along 20 different attributes: Brand, Price, Warranty, Horsepower, and so on. It is difficult for respondents to simultaneously evaluate that many attributes, so it is better if they are only exposed to a few at a time. Partial-profile designs have become very popular among some researchers.

Pair-wise Partial-Profile Choice Design

Here for example is a partial-profile design for 20 two-level factors, with 5 varying at a time, with the factors that are not shown printed with an ordinary missing value.

```
. . . 2 . . . 1 1 1 . . . . 1 . . . . .
2 . . . 2 . . . . . 1 . . . . . 1 . . 1
2 . 1 . . . . 1 . . . . . 1 . . . . . 2
. . . . . . . 2 . . . . . 2 2 . . 2 2 .
. . . . 2 . . . . 2 . 2 . . . 2 . 2 . .
. . . . . . . . 2 . . . . 1 . . . 2 1 1
. . . . . . . . 1 . 2 . 2 . . . 2 . . 2
1 . . . . 1 . . . . . . . . 2 . 2 . . . 1
. . . . . . 2 . . . . 2 1 . 2 . . . . 1
. . 1 . 1 . . . . 2 . . . . . 1 . . . 1
1 . 2 . . . 2 . . . . . . . . . 1 2 . .
. 1 . . . . . . . . . 2 . 1 . 1 1 . . . .
2 . . . . . . . . 2 . . 2 . . . 1 2 . . .
. . 1 . . . . . . . . . 2 2 . 1 . . . 1 .
. . . . 2 . 2 . . . . . . 2 . 1 . . 1 .
. 2 . 1 . . . 1 . . . . . . . . 2 . 2 .
2 2 . . . . 1 . 1 . . 1 . . . . . . . .
. 2 . . . . . . . . 1 1 . . . . . . 1 . 2
. 1 . 1 . 2 . 2 . . . . . . . . . . 1 . .
. . . 1 1 . . . . . 2 2 . 1 . . . . . .
1 . . . 2 2 1 . . . . . . . . 2 . . . . .
. . . . . 1 1 2 . 1 . . . . . . 1 . . .
. . . 2 . . . . . . . . 1 2 1 . 1 . . . .
. . . 1 . . . . 2 . . 1 . . 1 . . . . 2
. . 2 . . 1 . . 1 . 1 . . . 2 . . . . .
. . 2 . 1 . . 2 . . . . 1 . . 2 . . . .
. . 2 . . . 1 1 . . 2 . . . . . 1 . . .
. 1 . . 1 . . 1 2 . . . 2 . . . . . .
1 . . . . . . . . 1 . . . 1 . . . . 1 2 .
2 . . 2 1 1 . . . . . . . . . . . . 2 . .
```

```
. . . . . . . . . . . . . . . . . . . . 2 2 1 1 1 .
. 1 2 2 . . . . . . . . . . . . . . . . . . . . 2 2
. . 1 1 . . 1 . . . 1 . . . . 2 . . . . . . . . . .
. . . . . 2 . . . 2 . . 1 2 . . 1 . . . . . . . . .
. . . . . . 2 . . 2 . . . 1 1 . 2 . . . . . . . . .
1 . . . . . . 2 . 2 2 . . . . . . . 1 . . . . . . .
2 . 2 1 . . . . . 1 . . 2 . . . . . . . . . . . . .
. 1 1 . . . . . . . 1 . 1 . . . . 2 . . . . . . . .
. 2 1 2 2 . . . 2 . . . . . . . . . . . . . . . . .
. . . . . 2 2 . . . 2 . . . . 2 . . 2 . . . . . . .
```

A design like this could be used to make a binary choice experiment. For example, the last run has factors 6, 7, 11, 16, and 19 varying. Assume they are all yes-no factors (1 yes, 2 no). Subjects could be offered a choice between these two profiles:

```
x6 = no,   x7 = no,   x11 = no,   x16 = yes,  x19 = no
x6 = yes,  x7 = yes,  x11 = yes,  x16 = no,   x19 = yes
```

The first profile came directly from the design and the second came from shifting the design: yes → no, and no → yes.

Here is the code that generated and printed the partial-profile design.¶

```
title 'Partial Profiles';

%mktex(3 ** 20, n=41, partial=5, seed=292, maxdesigns=1)

%mktlab(values=. 1 2, nfill=99)

data _null_; set final(firstobs=2); put (x1-x20) (2.); run;
```

A 3^{20} design is requested in 41 runs. The three levels are yes, no, and not shown. Forty-one runs will give us 40 partial profiles and one more run with all attributes not shown (all ones in the original design before reassigning levels). When we ask for partial profiles, in this case `partial=5`, we are imposing a constraint that the number of 2's and 3's in each run equals 5 and the number of 1's equals 15. This makes the sum of the coded variables constant in each run and hence introduces a linear dependency (the sum of the coded variables is proportional to the intercept). The way we avoid having the linear dependency is by adding this additional row where all attributes are set to the not-shown level. The sum of the coded variables for this row will be different than the constant sum for the other rows and hence will eliminate the linear dependency we would otherwise have.

The `%MktLab` macro reassigns the levels (1, 2, 3) to (., 1, 2) where "." will mean not shown. Normally, the `%MktLab` macro complains about using missing values for levels, because missing values are used in the `key=` data set as fillers when some factors have more levels than others. Encountering missing levels normally indicates an error. We can allow for missing levels by specifying `nfill=99`. Then the macro considers levels of 99 to be invalid, not missing. A DATA step prints the design excluding the constant (all not shown) first row.

¶Due to machine, SAS release, and macro differences, you may not get exactly the same design as was used in this book, but the differences should be slight.

This next section of code takes this design and turns it into a partial-profile choice design. It reads each profile in the design, and outputs it. If the level is not missing, the code changes 1 to 2 and 2 to 1 and outputs the new profile. The next step uses the `%ChoicEff` macro to evaluate the design. We specified `zero=none` for now to see exactly which parameters we can estimate and which ones we cannot. This usage of the `%ChoicEff` macro is similar to what we saw in the food product example on page 321. Our choice design is specified on the `data=` option and the same data set, with just the `Set` variable kept, is specified on the `init=` option. The number of choice sets, 40 (we drop the constant choice set), number of alternatives, 2, and assumed betas, a vector of zeros, are also specified. Zero internal iterations are requested since we want a design evaluation, not an attempt to improve the design.

```
data des(drop=i);
   Set = _n_;
   set final(firstobs=2);
   array x[20];
   output;
   do i = 1 to 20;
      if n(x[i]) then do; if x[i] = 1 then x[i] = 2; else x[i] = 1; end;
      end;
   output;
   run;

%choiceff(data=des,
          model=class(x1-x20 / zero=none),
          nsets=40, nalts=2,
          beta=zero, init=des(keep=set),
          intiter=0)
```

Here is the last part of the output.

Partial Profiles

n	Variable Name	Label	Variance	DF	Standard Error
1	x11	x1 1	0.46962	1	0.68529
2	x12	x1 2	.	0	.
3	x21	x2 1	0.52778	1	0.72648
4	x22	x2 2	.	0	.
5	x31	x3 1	0.42989	1	0.65566
6	x32	x3 2	.	0	.
7	x41	x4 1	0.46230	1	0.67993
8	x42	x4 2	.	0	.
9	x51	x5 1	0.54615	1	0.73902
10	x52	x5 2	.	0	.

11	x61	x6 1	0.81069	1	0.90038
12	x62	x6 2	.	0	.
13	x71	x7 1	0.50135	1	0.70806
14	x72	x7 2	.	0	.
15	x81	x8 1	0.49753	1	0.70536
16	x82	x8 2	.	0	.
17	x91	x9 1	0.48632	1	0.69737
18	x92	x9 2	.	0	.
19	x101	x10 1	0.54529	1	0.73844
20	x102	x10 2	.	0	.
21	x111	x11 1	0.56975	1	0.75482
22	x112	x11 2	.	0	.
23	x121	x12 1	0.54158	1	0.73592
24	x122	x12 2	.	0	.
25	x131	x13 1	0.54817	1	0.74039
26	x132	x13 2	.	0	.
27	x141	x14 1	0.55059	1	0.74201
28	x142	x14 2	.	0	.
29	x151	x15 1	0.52638	1	0.72552
30	x152	x15 2	.	0	.
31	x161	x16 1	0.44403	1	0.66636
32	x162	x16 2	.	0	.
33	x171	x17 1	0.57751	1	0.75994
34	x172	x17 2	.	0	.
35	x181	x18 1	0.56915	1	0.75442
36	x182	x18 2	.	0	.
37	x191	x19 1	0.58340	1	0.76381
38	x192	x19 2	.	0	.
39	x201	x20 1	0.54343	1	0.73718
40	x202	x20 2	.	0	.
				==	
				20	

We see that one parameter is estimable for each factor and that is the parameter for the 1 or yes level. The `%ChoicEff` macro prints a list of all redundant variables.

```
Redundant Variables:

x12 x22 x32 x42 x52 x62 x72 x82 x92 x102 x112 x122 x132 x142 x152 x162 x172 x182
x192 x202
```

We can cut and paste this list into our program and drop those terms.

```
%choiceff(data=des,
          model=class(x1-x20 / zero=none),
          nsets=40, nalts=2,
          beta=zero, init=des(keep=set),
          intiter=0, drop=x12 x22 x32 x42 x52 x62 x72 x82 x92 x102
          x112 x122 x132 x142 x152 x162 x172 x182 x192 x202)
```

Here is the last part of the output.

Partial Profiles

n	Variable Name	Label	Variance	DF	Standard Error
1	x11	x1 1	0.46962	1	0.68529
2	x21	x2 1	0.52778	1	0.72648
3	x31	x3 1	0.42989	1	0.65566
4	x41	x4 1	0.46230	1	0.67993
5	x51	x5 1	0.54615	1	0.73902
6	x61	x6 1	0.81069	1	0.90038
7	x71	x7 1	0.50135	1	0.70806
8	x81	x8 1	0.49753	1	0.70536
9	x91	x9 1	0.48632	1	0.69737
10	x101	x10 1	0.54529	1	0.73844
11	x111	x11 1	0.56975	1	0.75482
12	x121	x12 1	0.54158	1	0.73592
13	x131	x13 1	0.54817	1	0.74039
14	x141	x14 1	0.55059	1	0.74201
15	x151	x15 1	0.52638	1	0.72552
16	x161	x16 1	0.44403	1	0.66636
17	x171	x17 1	0.57751	1	0.75994
18	x181	x18 1	0.56915	1	0.75442
19	x191	x19 1	0.58340	1	0.76381
20	x201	x20 1	0.54343	1	0.73718
				==	
				20	

Linear Partial-Profile Design

Here is another example. Say you would like to make a design in 36 runs with 12 three-level factors, but you want only four of them to be considered at a time. You would need to create four-level factors with one of the levels meaning not shown. You also need to ask for a design in 37 runs, because with partial profiles, one run must be all-constant. Here is a partial-profile request with the `%MktEx` macro using the `partial=` option.

```
title 'Partial Profiles';

%mktex(4 ** 12, n=37, partial=4, seed=462, maxdesigns=1)
%mktlab(values=. 1 2 3, nfill=99)

proc print; run;
```

The iteration history will proceed like before, so we will not discuss it. Here is the final *D*-efficiency.

Partial Profiles

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	49.4048	22.4346	100.0000	1.0000

With partial-profile designs, *D*-efficiency will typically be much less than we are accustomed to seeing with other types of designs. Here is the design.

Partial Profiles

Obs	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12
1	.	.	.	.	.	.	.	.	.	.	.	.
2	.	.	1	1	.	.	.	.	.	1	.	1
3	.	.	1	.	.	.	2	.	1	.	2	.
4	.	2	2	.	.	.	.	.	.	3	.	3
5	.	.	3	3	3	.	3	.	.	.	.	.
6	.	.	.	.	2	.	1	.	.	.	3	3
7	1	.	.	.	2	.	.	.	.	3	1	.
8	.	1	.	.	1	.	.	.	1	2	.	.
9	2	.	.	.	.	.	.	3	.	.	1	3
10	.	2	2	.	.	.	.	.	3	.	1	.
11	2	2	.	2	.	.	1	.	.	.	.	.
12	.	.	.	.	3	3	.	.	.	2	2	.
13	1	.	2	.	.	2	1	.	.	.	.	.
14	3	3	.	1	3	.	.	.	.	.	.	.
15	3	.	3	.	.	.	.	1	.	.	.	2
16	.	1	1	3	.	3	.	.	.	.	.	.
17	2	.	.	.	.	2	.	.	.	3	3	.
18	.	.	.	.	.	.	1	3	2	3	.	.
19	.	.	.	.	3	1	.	1	.	.	.	1
20	1	2	.	.	.	.	.	.	2	.	3	.
21	.	.	.	3	1	.	.	1	.	.	2	.
22	.	.	.	.	1	3	2	.	.	.	.	1
23	3	.	.	.	.	1	3	.	.	1	.	.
24	.	.	1	.	1	.	.	2	.	.	.	2
25	.	1	.	.	.	.	2	2	.	1	.	.
26	.	3	.	.	.	.	3	1	.	2	.	.
27	.	.	.	1	.	1	2	.	.	2	.	.

28	.	1	.	.	.	.	3	.	.	.	2	2
29	2	.	2	.	2	.	.	.	2	.	.	.
30	.	.	.	3	.	.	.	.	1	1	.	2
31	.	.	.	2	.	2	.	.	2	.	1	.
32	3	.	.	3	.	.	.	2	.	.	.	1
33	.	2	.	.	2	2	.	3	.	.	.	.
34	.	3	3	.	.	1	.	.	.	.	2	.
35	.	.	.	1	.	3	.	2	1	.	.	.
36	1	.	.	2	.	.	.	.	3	.	.	3
37	.	.	2	2	.	.	.	3	.	.	3	.

Notice that the first run is constant. For all other runs, exactly four factors vary and have levels not missing.

Choice from Triples; Partial Profiles Constructed Using Restrictions

The approach we just saw, constructing partial profiles using the `partial=` option, would be fine for a full-profile conjoint study or a pair-wise choice study with level shifts. However, it would not be good for a more general choice experiment with more alternatives. For a choice experiment, you would have to have partial-profile restrictions on each alternative, and you must have the same attributes varying in every alternative within each choice set. There is currently no automatic way to request this in the `%MktEx` macro, so you have to program the restrictions yourself. To specify restrictions for choice designs, you need to take into consideration the number of attributes that may vary within each alternative, which ones, and which attributes go with which alternatives. Fortunately, that is not too difficult. See page 286 for another example of restrictions.

In this section, we will construct a partial-profile design for a purely generic (unbranded) study, with ten attributes and three alternatives. Each attribute will have three levels, and each alternative will be a bundle of attributes. Partial-profile designs have the advantage that subjects do not have to consider all attributes at once. However, this is also a bit of a disadvantage as well in the sense that the subjects must constantly shift from considering one set of attributes to considering a different set. For this reason, it can be helpful to get more information out of each choice, and having more than two alternatives per choice set accomplishes this.

This example will have several parts. As we mentioned in the chair study, we will usually not directly use the `%MktEx` macro to generate designs for generic studies. Instead, we will use the `%MktEx` macro to generate a candidate set of partial-profile choice sets. Next, the design will be checked and turned into a candidate set of generic choice sets. Next, the `%MktDups` macro will be called to ensure there are no duplicate choice sets. Finally, the `%ChoicEff` macro will be used to create an efficient generic partial-profile choice design.

Before we go into any more detail on making this design, let's skip ahead and look at a couple of potential choice sets so it will be clear what we are trying to accomplish and why. Here are two potential choice sets, still in linear design format.

2 2 1 3 1 2 1 2 2 2	2 1 3 2 1 1 2 1 2 2	2 3 2 1 1 3 3 3 2 2
2 2 1 3 2 2 1 3 2 3	3 2 2 3 1 2 1 1 1 1	1 2 3 3 3 2 1 2 3 2

Here are the same two potential choice sets, but now arrayed in choice design format.

Partial Profiles

Set	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
1	2	2	1	3	1	2	1	2	2	2
	2	1	3	2	1	1	2	1	2	2
	2	3	2	1	1	3	3	3	2	2
2	2	2	1	3	2	2	1	3	2	3
	3	2	2	3	1	2	1	1	1	1
	1	2	3	3	3	2	1	2	3	2

Each choice set has 10 three-level factors and three alternatives. Four attributes are constant in each choice set: `x1`, `x5`, `x9`, and `x10` in the first choice set, and `x2`, `x4`, `x6`, and `x7` in the second choice set. We do not need an all-constant choice set like we saw in our earlier partial-profile designs, nor do we need an extra level for not varying. In this approach, we will simply construct choice sets for four constant attributes (they may be constant at 1, 2, or 3) and six varying attributes (with levels: 1, 2, and 3). Respondents will be given a choice task along the lines of "Given a set of products that differ on these attributes but are identical in all other respects, which one would you choose?". They would then be shown a list of differences.

Here is the code for making the candidate set.

```
title 'Partial Profiles';

%macro partprof;
   sum = 0;
   do k = 1 to 10;
      sum = sum + (x[k] = x[k+10]   &   x[k] = x[k+20]);
      end;
   bad = abs(sum - 4);
   %mend;

%mktex(3 ** 30, n=198, optiter=0, tabiter=0, maxtime=0, order=random,
       out=sasuser.cand, restrictions=partprof, seed=382)
```

We requested a design in 198 runs with 30 three-level factors. The 198 was chosen arbitrarily as a number divisible by $3 \times 3 = 9$ that would give us approximately 200 candidate sets. The first ten factors, `x1-x10`, will make the first alternative, the next ten, `x11-x20`, will make the second alternative, and the last ten, `x21-x30`, will make the third alternative. We will want six attributes to be nonconstant at a time. The `PartProf` macro will count the number of constant attributes: `x1 = x11 = x21`, `x2 = x12 = x22`, ..., and `x10 = x20 = x30`. If the number of constant attributes is four, our choice set conforms. If it is more or less than four, our choice set is in violation of the restrictions. The badness is the absolute difference between four and the number of constant attributes.

We specified `order=random`, which specifies that the columns are to be looped over in a random order in the coordinate-exchange algorithm. When `partial=` is specified, as it was in the previous partial-profile examples, `order=random` is the default. Whenever you are imposing partial-profile restrictions without using the `partial=` option, you should specify `order=random`. Without `order=random`, `%MktEx` will tend to put the nonconstant levels close together in each row.

Our goal in this step is to make a candidate set of potential partial-profile choice sets, not to make a final experimental design. Ideally, it would be nice if we had more than random candidates—it would be nice if our candidate generation code at least made some attempt to ensure that our attributes are approximately orthogonal and balanced across attributes both between and within alternatives. This is a big problem (30 factors and 198 runs) with restrictions, so `%MktEx` macro will run slowly by default. It is not critical that we allow the macro to spend a great deal of time optimizing linear model *D*-efficiency. For this reason, we use some of the more esoteric number-of-iterations options. We specify `optiter=0`, which specifies no OPTEX iterations, since with large partial-profile studies, we will never have a good candidate set for PROC OPTEX to search. We also specify `tabiter=0` since a tabled initial design will be horrible for this problem. We specified the `maxtime=0` option so that the macro will just create two candidate designs using the coordinate-exchange algorithm with a random initialization and make one attempt to refine the best one.

Partial Profiles

Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
1	Start	85.1531		Ran,Mut,Ann
1	169 1	93.3863	93.3863	Conforms
1	169 8	93.3892	93.3892	
1	169 22	93.3892	93.3892	
1	169 12	93.3911	93.3911	
1	170 20	93.3954	93.3954	
.				
.				
.				
1	123 1	96.5805	96.5805	
1	38 21	96.5806	96.5806	
1	47 1	96.5811	96.5811	
1	End	96.5811		

NOTE: Quitting the algorithm search step after 2.07 minutes and 1 designs.

.
.
.

```
                          Partial Profiles

                      Design Refinement History

                                 Current          Best
Design    Row,Col  D-Efficiency  D-Efficiency  Notes
-------------------------------------------------------
     0    Initial       96.6394       96.6394  Ini

     1      Start       96.0428                Pre,Mut,Ann
     1    180   1       95.9063                Conforms
     1        End       96.6112

NOTE: Quitting the refinement step after 1.61 minutes and 1 designs.

                          Partial Profiles

                        The OPTEX Procedure

                                                                  Average
                                                                Prediction
Design                                                          Standard
Number     D-Efficiency     A-Efficiency     G-Efficiency          Error
-------------------------------------------------------------------------
     1         96.6394          93.6339          94.9469         0.5551
```

The macro finds a design that conforms to the restrictions (shown by the `Conforms` note). This step took approximately 7 minutes.

Here is the rest of the code for making the partial-profile choice design.

```
%mktkey(3 10)

%mktroll(design=sasuser.cand, key=key, out=rolled)

%mktdups(generic, data=rolled, out=nodups, factors=x1-x10, nalts=3)

proc print data=nodups(obs=9); id set; by set; run;

%choiceff(data=nodups, model=class(x1-x10), seed=495,
          iter=10, nsets=27, nalts=3, options=nodups, beta=zero)

proc print data=best; id set; by notsorted set; var x1-x10; run;
```

The `%MktKey` macro is run to generate a `Key` data set with 3 rows, 10 columns and the variable names `x1 - x30`. Here is the `Key` data set.

Partial Profiles

x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
x11	x12	x13	x14	x15	x16	x17	x18	x19	x20
x21	x22	x23	x24	x25	x26	x27	x28	x29	x30

Then the `%MktRoll` macro is run to create a generic choice design from the linear candidate design.

The next step runs the `%MktDups` macro, which we have not used in previous examples. The `%MktDups` macro can check a design to see if there are any duplicate runs and output just the unique ones. For a generic study like this, it can also check to make sure there are no duplicate choice sets taking into account the fact that two choice sets can be duplicates even if the alternatives are not in the same order. The `%MktDups` step names in a positional parameter the type of design as a `generic` choice design. It names the input data set and the output data set that will contain the design with any duplicates removed. It names the factors in the choice design `x1-x10` and the number of alternatives. The result is a data set called `NoDups`. Here are the first 3 candidate choice sets.

Partial Profiles

Set	_Alt_	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
1	1	1	1	1	1	2	3	2	3	2	2
	2	1	3	3	2	2	3	2	1	1	2
	3	2	2	2	3	2	3	2	1	3	2
2	1	1	1	1	1	3	2	3	3	2	2
	2	2	1	2	1	2	2	2	2	2	1
	3	3	1	1	1	2	2	1	1	2	3
3	1	1	1	1	2	2	3	3	2	1	3
	2	2	3	1	1	3	3	3	1	1	2
	3	3	2	1	3	1	3	3	3	1	1

The `%ChoiceEff` macro is called to search for an efficient choice design. The model specification `class(x1-x10)` specifies a generic model with 10 attributes. The option `iter=10` specifies more than the default number of iterations (the default is 2 designs). We ask for a design with 27 sets and 3 alternatives. Furthermore, we ask for no duplicate choice sets and specify an assumed beta vector of zero. Here are some of the results from the `%ChoiceEff` macro.

Partial Profiles

Design	Iteration	D-Efficiency	D-Error
1	0	2.368953	0.422127
	1	2.904996	0.344235
	2	2.912352	0.343365

.
.
.

Design	Iteration	D-Efficiency	D-Error
10	0	2.444770	0.409036
	1	2.842701	0.351778
	2	2.879135	0.347326
	3	2.903779	0.344379
	4	2.905142	0.344217

Partial Profiles

Final Results

Design	8
Choice Sets	27
Alternatives	3
D-Efficiency	2.930997
D-Error	0.341181

Partial Profiles

n	Variable Name	Label	Variance	DF	Standard Error
1	x11	x1 1	0.44087	1	0.66398
2	x12	x1 2	0.40529	1	0.63662
3	x21	x2 1	0.39964	1	0.63217
4	x22	x2 2	0.40091	1	0.63318
5	x31	x3 1	0.39525	1	0.62869
6	x32	x3 2	0.42408	1	0.65122
7	x41	x4 1	0.41602	1	0.64500
8	x42	x4 2	0.44004	1	0.66335
9	x51	x5 1	0.42818	1	0.65436
10	x52	x5 2	0.42669	1	0.65321

```
11      x61       x6 1      0.39914     1      0.63178
12      x62       x6 2      0.39934     1      0.63194
13      x71       x7 1      0.40924     1      0.63972
14      x72       x7 2      0.41546     1      0.64456
15      x81       x8 1      0.45180     1      0.67216
16      x82       x8 2      0.41896     1      0.64727
17      x91       x9 1      0.42344     1      0.65072
18      x92       x9 2      0.41993     1      0.64802
19      x101      x10 1     0.39293     1      0.62684
20      x102      x10 2     0.39998     1      0.63244
                                        ==
                                        20
```

Here is part of the design.

Partial Profiles

Set	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
54	1	3	2	2	2	2	1	1	2	3
	2	1	2	2	1	3	1	3	2	1
	3	2	2	2	3	1	1	2	2	2
30	1	2	2	1	3	1	2	2	1	1
	3	3	2	2	3	3	3	2	1	3
	2	1	2	3	3	2	1	2	1	2
.										
.										
.										
163	3	2	1	3	1	3	3	1	1	2
	1	2	1	1	2	2	3	1	2	3
	2	2	1	2	3	1	3	1	3	1

The design has 27 choice sets. The choice set numbers shown in this output correspond to the *original* set numbers in the candidate design not the choice set numbers in the final design.

Six Alternatives; Partial Profiles Constructed Using Restrictions

In this next example, we will construct a partial-profile design with 20 binary attributes and six alternatives with 15 attributes fixed at the base-line level of 1 for each alternative. Our partial-profile restriction macro is an obvious modification of the one used in the previous example. Our linear design will need $6 \times 20 = 120$ factors. The first attribute will be made from `x1`, `x21`, `x41`, `x61`, `x81`, and `x101`; the second attribute will be made from `x2`, `x22`, `x42`, `x62`, `x82`, and `x102`; and so on. The `do` loop counts the number of times all of the linear factors within an attribute are equal to one and our badness function increases as the number of constant attributes deviates from 15.

```
%macro partprof;
   sum = 0;
   do k = 1 to 20;
      sum = sum + (x[k]    = 1 & x[k+20] = 1 & x[k+40]  = 1 &
                   x[k+60] = 1 & x[k+80] = 1 & x[k+100] = 1);
      end;
   bad = abs(sum - 15);
   %mend;
```

The %MktEx macro is run requesting 120 binary factors and 300 runs. We specify optiter=0 and tabiter=0 so that only the coordinate-exchange algorithm will be used. With maxtime=0 and maxstages=1, only one design will be created. Several options are specified with options=. The largedesign option allows %MktEx to stop as soon as it has imposed all restrictions. The resrep option reports on the progress of imposing restrictions. We specified order=random, which specifies that the columns are to be looped over in a random order in the coordinate-exchange algorithm. You should always specify order=random with partial-profile designs. With a sequential order you will tend to get nonvarying attributes paired with only nearby attributes.

```
%mktex(2 ** 120, n=300, optiter=0, tabiter=0, maxtime=0, order=random,
       out=cand, restrictions=partprof, seed=424,
       maxstages=1, options=largedesign nosort resrep)
```

Here is the first part of the output.

Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
1	Start	79.5699		Ran,Mut,Ann
1	1	79.8384		14 Violations
1	2	80.0541		12 Violations
1	3	80.3047		14 Violations
1	4	80.5649		15 Violations
1	5	80.7840		13 Violations
1	6	81.0383		12 Violations
.				
.				
.				

At the end of the first pass through the design we see the following warning.

```
WARNING: It may be impossible to meet all restrictions.
```

The second pass begins like this:

Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
1	1	95.9591		13 Violations
1	2	95.9655		12 Violations
1	3	95.9772		13 Violations
1	4	95.9897		15 Violations
1	5	95.9684		12 Violations
.				
.				
.				

The macro never gets anywhere in imposing restrictions. Every row of the design has lots of restrictions violations as shown by the output from the `resrep` (restrictions report) option. Always specify `options=resrep` with complicated restrictions so you can look for things like this. When the macro is never succeeding in imposing restrictions, there is something wrong with the restrictions macro. Here is the macro again.

```
%macro partprof;
   sum = 0;
   do k = 1 to 20;
      sum = sum + (x[k]    = 1 & x[k+20] = 1 & x[k+40]  = 1 &
                   x[k+60] = 1 & x[k+80] = 1 & x[k+100] = 1);
      end;
   bad = abs(sum - 15);
   %mend;
```

It correctly evaluates in the Boolean expression to determine whether an attribute is all one, and it correctly counts the number of such attributes. It also correctly sets `bad` to the absolute difference between the sum and 15, the desired number of constant attributes. Sometimes you get results like we just saw when you made a logical mistake in programming the restrictions. For example, you may have written a set of restrictions that are impossible to satisfy. That is not the problem in this case. The problem in this case is the quantification of badness is not fine enough. You need to tell the macro whenever it does something that moves it closer to an acceptable solution. Consider a potential choice set that almost conforms to the restrictions.

Set	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	x18	x19	x20
1	2	1	1	1	2	2	1	1	1	1	1	1	2	1	1	1	1	1	1	1
	2	1	2	1	2	2	1	1	1	1	1	1	2	1	1	1	1	1	1	1
	1	1	2	1	1	2	2	1	1	1	1	1	2	1	1	1	1	1	1	1
	1	1	2	1	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2	1	1	1	1	1	2	1	1	1	1	1	1	1	1	1	1	1	1	1
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

The problem with this choice set is it has six nonconstant attributes instead of five. So `bad = abs(14 - 15) = 1`. Now consider what happens when the macro changes the first attribute of the first alternative from 2 to 1. This is a change in the right direction because it moves the choice set closer to having one more constant attribute. However, this change has no effect on the badness criterion. It is still 1, because we still have six nonconstant attributes. We are not giving `%MktEx` enough information about when it is heading in the right direction. The `%MktEx` macro, without your restrictions macro to otherwise guide it, is guided by the goal of maximizing *D*-efficiency. It does not particularly want to make partial-profile designs, because imposing all of those ties decreases statistical *D*-efficiency. Using the hill climbing analogy, `%MktEx` wants to climb Mount Everest; your restrictions macro needs to tell it to find the top of an island in the middle of a river valley. This is not where `%MktEx` would normally look. If your restrictions macro is going to overcome the `%MktEx` macro's normal goal, you have to train it and more explicitly tell it where to look.

Training the `%MktEx` macro to find highly restricted designs is like training a dog. You have to be persistent and consistent, and you need to watch it every second. You have to reward it whenever it does the right thing and you have to punish it for even thinking about doing the wrong thing. When it tears up your favorite slippers, you need to whack it over the head with a rolled up newspaper.[‖] It is eager to please, but it is easily tempted, and it is not smart enough to figure out what to do unless you very explicitly tell it. It will run wherever it wants unless you keep it on a short leash. With that in mind, here is a revised partial-profile restrictions macro.

```
%macro partprof;
   sum = 0;
   do k = 1 to 20;
      sum = sum + (x[k]    = 1 & x[k+20] = 1 & x[k+40]  = 1 &
                   x[k+60] = 1 & x[k+80] = 1 & x[k+100] = 1);
      end;
   bad = abs(sum - 15);
   if sum < 15 & x[j1] = 2 then do;
      k = mod(j1 - 1, 20) + 1;
      c = (x[k]    = 1) + (x[k+20] = 1) + (x[k+40]  = 1) +
          (x[k+60] = 1) + (x[k+80] = 1) + (x[k+100] = 1);
      if c >= 3 then bad = bad + (6 - c) / 6;
      end;
   %mend;
```

It starts the same way as the old one, but it has some additional fine tuning. Like before, this macro counts the number of times that all six alternatives equal 1 and stores the result in `sum`. When `sum` is less than 15, we need more constant attributes. When the current level of the current factor, `x[j1]`, is 2, the macro next considers whether it should change the 2 to a 1. First, we compute `k`, the attribute number and evaluate the number of ones in that attribute, and store that in `c`. (For example when `j1`, the linear factor number, equals 2, 22, 42, 62, 82, or 102, we are working with attribute `k` = 2.) If it looks like this factor is going to be constant (three or more ones), we add the proportion of twos to the badness function (more twos is worse).

Consider again the first row of the sample choice set shown previously. Badness starts out as 1 since `sum` is 14. Since badness is nonzero and since we are looking at a 2 in the first column, badness is increased by 1/2, which is the proportion of twos. Now consider changing that first two to a one. Now badness is 1 + 1/3, which is smaller than 1 + 1/2 so changing 2 to 1 moves badness in the right direction.

[‖] No actual dogs were harmed in the process of developing any of this software.

There are many other ways that you could write a restrictions macro for this problem. However you do it, you need to provide a quantification of badness that guides the macro toward acceptable designs. Most of the time, for complicated restrictions, you will not be able to sit down and write a good restrictions macro off of the top of your head.** It will require some trial and error to get something that works. For this reason, at least at first, you should specify `maxtime=0, maxstages=1, options=largedesign resrep` so you can see if the macro is succeeding in imposing restrictions, and so the macro stops quickly when it has succeeded. Then you need to carefully check your design. It is not unusual for the macro to succeed splendidly only to find you gave it the wrong set of restrictions. We can run the `%MktEx` macro exactly as before to test our new macro.

```
%mktex(2 ** 120, n=300, optiter=0, tabiter=0, maxtime=0, order=random,
       out=cand, restrictions=partprof, seed=424,
       maxstages=1, options=largedesign nosort resrep)
```

Here is some of the output from the first pass through the design.

Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
1	Start	79.5699		Ran,Mut,Ann
1	1	79.5414		0 Violations
1	2	79.6570		1 Violations
1	3	79.7998		4 Violations
1	4	79.7799		2 Violations
1	5	79.7653		0 Violations
.				
.				
.				
1	100	71.3142		0 Violations
1	101	71.2115		3 Violations
1	102	71.1284		4 Violations
.				
.				
.				
1	192	55.9495		2.167 Violations
1	193	55.7292		2 Violations
1	194	55.5926		4 Violations
1	195	55.4194		0 Violations
1	196	55.2477		2 Violations
1	197	55.0915		4.333 Violations
1	198	54.8782		0 Violations
1	199	54.7156		1 Violations
1	200	54.6435		6 Violations
.				
.				
.				

**This one may look simple once you see it, but it took me several tries to get it right. I had previous versions that worked quite well in spite of some logical flaws that caused them to quantify badness in not quite the way that I intended.

```
1    294          37.3778              4.167 Violations
1    295          37.1935              2 Violations
1    296          37.1475              6 Violations
1    297          36.9941              3 Violations
1    298          36.7927              0 Violations
1    299          36.5551              1 Violations
1    300          36.3781              5 Violations
```

This iteration history looks better. At least some choice sets conform. Most do not however. Notice the fractional number of violations in some rows. Throughout most of this iteration history, *D*-efficiency is steadily going down as more and more restrictions are imposed. Here is part of the iteration history for the second pass through the design.

Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
1	1	36.4174		0 Violations
1	2	36.3885		0 Violations
1	3	36.2906		0 Violations
1	4	36.2632		1 Violations
1	4	36.2683		0 Violations
1	5	36.3141		0 Violations
.				
.				
.				
1	100	34.2618		0 Violations
1	101	34.2198		0 Violations
1	102	34.1519		0 Violations
.				
.				
.				
1	200	32.4079		0 Violations
1	201	32.3571		0 Violations
1	202	32.2518		0 Violations
.				
.				
.				
1	298	30.8243		0 Violations
1	299	30.8499		0 Violations
1	300	30.7717		0 Violations

This part of the iteration history looks much better. The table contains a number of rows like what we see in row four. The `%MktEx` macro attempts to impose restrictions and it does not quite succeed, so it immediately tries again, one or more times, until it succeeds or gives up. In this case, it always succeeds. *D*-efficiency is still mostly decreasing. Note that the "0 Violations" that we see in this table tells us that there are no violations remaining in the indicated row. There may very well still be violations in other rows. However by the end of this pass, the restrictions are almost certainly completely imposed,

so we should see *D*-efficiency start back up. Here is some of the output from the next pass through the design.

Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
1	1	30.7725		0 Violations
1	2	30.8039		0 Violations
1	3	30.8289		0 Violations
.				
.				
.				
1	100	32.1841		0 Violations
1	101	32.2091		0 Violations
1	102	32.2166		0 Violations
.				
.				
.				
1	200	33.1859		0 Violations
1	201	33.2199		0 Violations
1	202	33.2383		0 Violations
.				
.				
.				
1	298	33.8693		0 Violations
1	299	33.8782		0 Violations
1	300	33.8869		0 Violations

Now *D*-efficiency is increasing. Here is the rest of the iteration history.

Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
1	1 1	33.8869	33.8869	Conforms
1	1 45	33.8873	33.8873	
1	1 85	33.8909	33.8909	
1	1 113	33.8921	33.8921	
1	End	33.8872		

It is followed by these messages.

```
NOTE: Stopping early, possibly before convergence, with a large design.
NOTE: Quitting the algorithm search step after 2.25 minutes and 22 designs.
```

Due to `maxtime=0, maxstages=1, options=largedesign`, the macro stops after it has completed one pass through the design without encountering any restriction violations. Our final *D*-efficiency =

33.8872 is not very high, but this will just be a candidate set. Furthermore, this is a *highly* restricted design, so it is not surprising that *D*-efficiency is not very high. We may want to iterate more and see if we can do better, but for now, let's test the rest of our code. The `%MktEx` step took under 2.5 minutes. When the macro completes a full pass through the design without detecting any violations, it prints "Conforms" and switches from the `options=resrep` style to the normal iteration history style. These next steps turn the linear design into a choice design and eliminate duplicate choice sets.

```
%mktkey(6 20)

%mktroll(design=cand, key=key, out=rolled)

proc print; by set; id set; var x:; where set le 5; run;

%mktdups(generic, data=rolled, out=nodups, factors=x1-x20, nalts=6)

proc print data=nodups(obs=18); id set; by set; run;
```

The `%MktKey` macro is run to generate a `Key` data set with 6 rows, 20 columns and the variable names `x1 - x120`. Here is the `Key` data set, printed in two panels.

x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
x21	x22	x23	x24	x25	x26	x27	x28	x29	x30
x41	x42	x43	x44	x45	x46	x47	x48	x49	x50
x61	x62	x63	x64	x65	x66	x67	x68	x69	x70
x81	x82	x83	x84	x85	x86	x87	x88	x89	x90
x101	x102	x103	x104	x105	x106	x107	x108	x109	x110
x11	x12	x13	x14	x15	x16	x17	x18	x19	x20

x11	x12	x13	x14	x15	x16	x17	x18	x19	x20
x31	x32	x33	x34	x35	x36	x37	x38	x39	x40
x51	x52	x53	x54	x55	x56	x57	x58	x59	x60
x71	x72	x73	x74	x75	x76	x77	x78	x79	x80
x91	x92	x93	x94	x95	x96	x97	x98	x99	x100
x111	x112	x113	x114	x115	x116	x117	x118	x119	x120

The `%MktDups` macro eliminated 70 choice sets with duplicate alternatives resulting in a candidate set with 230 choice sets. Here are the results of the last PROC PRINT, with the first three candidate choice sets.

Set	_Alt_	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	x18	x19	x20
1	1	1	1	1	1	2	1	1	1	2	1	1	1	2	1	1	1	1	1	1	2
	2	1	1	1	1	2	1	1	1	2	1	1	1	2	1	1	1	1	1	1	1
	3	1	1	1	1	1	2	1	1	1	1	1	1	2	1	1	1	1	1	1	2
	4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2
	5	1	1	1	1	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	6	1	1	1	1	2	2	1	1	2	1	1	1	2	1	1	1	1	1	1	2
2	1	1	2	1	2	1	1	1	1	1	1	1	1	2	1	1	1	1	1	1	1
	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	1	1	2	1
	3	1	2	1	1	1	1	1	1	1	1	1	1	2	1	1	1	1	1	1	1
	4	1	2	1	1	1	1	1	1	1	1	1	1	1	1	1	2	1	1	2	1
	5	1	1	1	2	1	1	1	1	1	1	1	1	2	1	1	1	1	1	2	1
	6	1	1	1	2	1	1	1	1	1	1	1	1	2	1	1	2	1	1	1	1
3	1	2	1	1	1	1	1	1	1	2	1	1	1	1	1	1	1	1	1	2	1
	2	2	1	1	1	1	1	2	1	1	1	1	1	1	1	1	1	1	1	2	1
	3	2	1	1	1	1	1	1	1	2	1	1	1	1	1	1	2	1	1	1	1
	4	1	1	1	1	1	1	2	1	2	1	1	1	1	1	1	2	1	1	2	1
	5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	1	1	2	1
	6	1	1	1	1	1	1	2	1	1	1	1	1	1	1	1	2	1	1	1	1

Each choice set has the correct number of nonconstant attributes. This next step runs the `%ChoicEff` macro to find a choice design. We set the reference level to the the first level of each factor, which is the base-line level of 1.

```
%choiceff(data=nodups, model=class(x1-x20 / zero=first), seed=495,
          iter=10, nsets=18, nalts=6, options=nodups, beta=zero)

proc print data=best; id set; by notsorted set; var x1-x20; run;
```

Here is some of the output.

n	Name	Beta	Label
1	x12	0	x1 2
2	x22	0	x2 2
3	x32	0	x3 2
4	x42	0	x4 2
5	x52	0	x5 2
6	x62	0	x6 2
7	x72	0	x7 2
8	x82	0	x8 2
9	x92	0	x9 2
10	x102	0	x10 2
11	x112	0	x11 2
12	x122	0	x12 2
13	x132	0	x13 2
14	x142	0	x14 2
15	x152	0	x15 2
16	x162	0	x16 2
17	x172	0	x17 2
18	x182	0	x18 2
19	x192	0	x19 2
20	x202	0	x20 2

Design	Iteration	D-Efficiency	D-Error
1	0	0.909928	1.098988
	1	1.018467	0.981868
	2	1.028127	0.972643
	3	1.036325	0.964948
	4	1.042916	0.958850
	5	1.043576	0.958243

Design	Iteration	D-Efficiency	D-Error
2	0	0.955735	1.046315
	1	1.026372	0.974306
	2	1.044285	0.957593
	3	1.050147	0.952248
	4	1.050147	0.952248

.
.
.

Design	Iteration	D-Efficiency	D-Error
10	0	0	.
	1	1.021022	0.979411
	2	1.033626	0.967468
	3	1.038226	0.963181

Final Results

Design	2
Choice Sets	18
Alternatives	6
D-Efficiency	1.050147
D-Error	0.952248

n	Variable Name	Label	Variance	DF	Standard Error
1	x12	x1 2	0.88262	1	0.93948
2	x22	x2 2	0.91343	1	0.95573
3	x32	x3 2	1.12396	1	1.06017
4	x42	x4 2	0.84310	1	0.91821
5	x52	x5 2	1.08252	1	1.04044
6	x62	x6 2	1.11615	1	1.05648
7	x72	x7 2	0.84372	1	0.91854
8	x82	x8 2	0.89847	1	0.94787
9	x92	x9 2	1.06525	1	1.03211
10	x102	x10 2	0.88000	1	0.93808
11	x112	x11 2	0.94155	1	0.97033
12	x122	x12 2	1.09926	1	1.04845
13	x132	x13 2	1.09443	1	1.04615
14	x142	x14 2	0.87119	1	0.93337
15	x152	x15 2	1.13624	1	1.06594
16	x162	x16 2	0.93743	1	0.96821
17	x172	x17 2	1.05015	1	1.02477
18	x182	x18 2	0.89569	1	0.94641
19	x192	x19 2	1.12411	1	1.06024
20	x202	x20 2	1.08256	1	1.04046
				==	
				20	

Set	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	x18	x19	x20
223	1	1	1	1	1	2	1	1	1	1	1	1	2	1	1	1	1	1	1	1
	1	1	1	1	1	1	1	1	1	1	1	2	1	1	1	1	2	1	1	1
	1	1	1	1	1	2	1	1	1	2	1	1	1	1	1	1	2	1	1	1
	1	1	1	1	1	1	1	1	1	2	1	2	2	1	1	1	1	1	1	1
	1	1	1	1	1	1	1	1	1	2	1	1	1	1	1	1	1	1	1	1
	1	1	1	1	1	2	1	1	1	2	1	2	2	1	1	1	2	1	1	1

```
 128  1  1  1  1  1  2  1  1  1  1  1  1  2  1  1  1  1  2  1  1
      1  2  2  1  1  1  1  1  1  1  1  1  2  1  1  1  1  1  1  1
      1  1  2  1  1  2  1  1  1  1  1  1  1  1  1  1  1  2  1  1
      1  2  2  1  1  2  1  1  1  1  1  1  1  1  1  1  1  2  1  1
      1  1  1  1  1  1  1  1  1  1  1  1  2  1  1  1  1  2  1  1
      1  2  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
   .
   .
   .
  29  1  1  1  1  1  1  2  1  1  1  1  1  1  2  1  1  1  1  1  1
      2  1  1  2  1  1  2  1  1  1  1  1  1  1  1  2  1  1  1  1
      1  1  1  1  1  1  2  1  1  1  1  1  1  1  1  2  1  1  1  1
      2  1  1  1  1  1  1  1  1  1  1  1  1  2  1  1  1  1  1  1
      2  1  1  2  1  1  1  1  1  1  1  1  1  2  1  2  1  1  1  1
      1  1  1  2  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
```

Since the `%ChoicEff` macro did not have any problems, and the results look reasonable, it would appear that we did everything right. Now we could try again, perhaps with more choice sets (say 400), and we could iterate longer (say one hour), then we could make our final partial-profile choice design. The results of this step are not shown.

```
%mktex(2 ** 120, n=400, optiter=0, tabiter=0, maxtime=60, order=random,
       out=cand, restrictions=partprof, seed=424,
       maxstages=1, options=largedesign nosort)
```

To recap, the first restrictions macro correctly differentiated between acceptable and unacceptable choice sets, but it provided `%MktEx` with no guidance or direction on how to find acceptable choice sets. Hence, the first macro did not work. The second macro corrected this problem by "nudging" `%MktEx` in the right direction. The restrictions macro looked for attributes that appear to be heading toward constant and created a penalty function that encouraged `%MktEx` to make those attributes constant. Next, we will look at another way of writing a restrictions macro for this problem. There is nothing subtle about this next approach. This next macro uses the whack-it-over-the-head-with-a-rolled-up-newspaper approach. Sometimes you need to tell `%MktEx` that a restriction is really important by strongly eliminating that source of badness. In this case, our macro strongly eliminates twos from the design until the restriction violations go away.

```
%macro partprof;
   sum = 0;
   do k = 1 to 20;
      sum = sum + (x[k]    = 1 & x[k+20] = 1 & x[k+40]  = 1 &
                   x[k+60] = 1 & x[k+80] = 1 & x[k+100] = 1);
      end;
   bad = abs(sum - 15);
   if sum < 15 & x[j1] = 2 then bad = bad + 1000;
   %mend;

%mktex(2 ** 120, n=300, optiter=0, tabiter=0, maxtime=0, order=random,
       out=cand, restrictions=partprof, seed=424,
       maxstages=1, options=largedesign nosort resrep)
```

Recall that we specified `order=random` so within each choice set, the columns are traversed in a different random order. As long as there are violations, within each choice set, this macro turns random twos into ones until there are so few twos left that the violations go away. Then once all of the violations go away, it allows twos to be changed back to ones to increase *D*-efficiency.

Here is the part of the iteration history.

Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
1	Start	79.5699		Ran,Mut,Ann
1	1	79.5701		0 Violations
1	2	79.5230		0 Violations
1	3	79.4393		0 Violations
1	4	79.3154		0 Violations
1	5	79.1462		0 Violations
.				
.				
.				
1	100	57.6919		0 Violations
1	101	57.4785		0 Violations
1	102	57.2048		0 Violations
.				
.				
.				
1	200	34.9889		0 Violations
1	201	34.7612		0 Violations
1	202	34.5789		0 Violations
.				
.				
.				
1	298	17.9407		0 Violations
1	299	17.7334		0 Violations
1	300	17.4295		0 Violations
1	1	17.5535		0 Violations
1	2	17.6737		0 Violations
1	3	17.7828		0 Violations
.				
.				
.				
1	100	23.6202		0 Violations
1	101	23.6901		0 Violations
1	102	23.7532		0 Violations

```
.
.
.

1    200          28.2418              0 Violations
1    201          28.2855              0 Violations
1    202          28.3235              0 Violations

.
.
.

1    298          31.8558              0 Violations
1    299          31.8751              0 Violations
1    300          31.9007              0 Violations
1      1   1      31.9007      31.9007 Conforms
1      1  27      31.9113      31.9113
1      1   5      31.9119      31.9119
1      1  86      31.9143      31.9143
1      1   6      31.9149      31.9149
1      1 107      31.9188      31.9188
1      1  67      31.9210      31.9210
1      1  73      31.9314      31.9314
1      1  65      31.9321      31.9321
1      1  33      31.9385      31.9385
1      1  23      31.9389      31.9389
1      1 113      31.9397      31.9397
1      1  43      31.9426      31.9426
1        End      31.9426
```

With this approach, all violations in each row are eliminated in the first pass through the design. The macro quits after the end of the second pass, when it has completed an entire pass without encountering any restriction violations.

Call the first approach the "nudge" approach and the second approach the "whack" approach. The nudge approach starts with *D*-efficiency on the order of 80% for the random design. After one complete pass, imposing many but not all restrictions, *D*-efficiency is down around 36%. After another pass and imposing all restrictions, it is down to around 31%. Then it creeps back up to 34%. *D*-efficiency for the choice design is approximately 1.05. The whack approach starts with the same random design (due to the same random number seed) and with the same *D*-efficiency on the order of 80%. Then after one pass of severe restriction imposition, *D*-efficiency drops to 17%. The nudge approach asks "are you a good two or a bad two?", then it acts accordingly. In contrast, when the whack approach sees a two, it whacks it–no questions asked. There is no subtle nudging in the whack approach, and initially, it over corrects to impose restrictions. Hence, the design it makes at the end of the first pass is not very good, but once all of the restrictions are in place, *D*-efficiency quickly recovers to 32%. This is not quite as high as the nudge approach, but the nudge approach had one more complete pass through the design. *D*-efficiency for the choice design (not shown) was similar at 1.02.

It is natural to ask, which approach is better? There are lots of ways to get `%MktEx` to impose the restrictions. It is not clear that one is better than the other. Our goal here is to use `%MktEx` to create a set of candidates for the `%ChoicEff` macro to search. Any restrictions macro that accomplishes this

should be fine. Writing a macro that uses the whack approach is probably a bit easier. However, there is always some worry that the initial over correction may not be a good thing. In contrast, subtle nudging takes longer to get to the point of all restrictions being met, and you have to be a bit creative sometimes to write nudges that actually work. Sometimes you need to combine both approaches – whack it to take care of one set of restrictions then nudge it in the right direction for a secondary set of restrictions. This is all part of the art of sophisticated experimental design. Note that it is not the fact that we used a large penalty of 1000 that makes this approach an example of the whack approach. It is the whack approach because we strongly overcorrected every violation.

Five-Level Factors; Partial Profiles Constructed Using Restrictions

This next example extends what we discussed in the previous example and constructs a somewhat different style of partial-profile design. This design will have five alternatives and 15 five-level factors, five of which will vary in each choice set. Unlike the previous example, however, the constant factors will not all be at the base-line level. The constant factors can have any of the levels, and we use the first factor within each attribute when we check the restrictions. Here is the partial-profile restrictions macro along with `%MktEx` code, which uses the same basic option set that we used in the previous example.

```
%macro partprof;
   sum = 0;
   do k = 1 to 15;
      sum = sum + (x[k+15] = x[k] & x[k+30] = x[k] &
                   x[k+45] = x[k] & x[k+60] = x[k]);
      end;
   bad = abs(sum - 10);
   if sum < 10 & x[j1] ^= x[mod(j1 - 1, 15) + 1] then bad = bad + 1000;
   %mend;

%mktex(5 ** 75, n=400, optiter=0, tabiter=0, maxtime=0, order=random,
       out=cand, restrictions=partprof, seed=472,
       maxstages=1, options=largedesign nosort resrep)
```

The macro counts the number of times all of the linear factors in a choice set attribute are constant within choice set, that is they equal the level of the first linear factor in the attribute, then it computes badness in the customary way. Also like before, when not all restrictions are met, any level that is not equal to the first factor within its attribute is heavily penalized. The macro uses the "whack" approach to impose constant attributes. Here is some of the iteration history.

Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
1	Start	58.6611		Ran,Mut,Ann
1	1	58.6885		0 Violations
1	2	58.7260		0 Violations

```
.
.
.
1    100          55.7959          0 Violations
1    101          55.7391          2 Violations
1    102          55.7100          3 Violations
.
.
.
1    200          47.9268          1 Violations
1    201          47.8219          3 Violations
1    202          47.7466          1 Violations
.
.
.
1    300          38.9361          6 Violations
1    301          38.8744          4 Violations
1    302          38.7843          0 Violations
.
.
.
1    398          30.4707          4 Violations
1    399          30.3975          3 Violations
1    400          30.3355          1 Violations
1      1          30.4083          0 Violations
1      2          30.4790          0 Violations
1      3          30.5447          0 Violations
.
.
.
1     13          30.9147          2 Violations
1     13          30.9579          0 Violations
1     14          30.9680          0 Violations
1     15          31.0077          0 Violations
1     16          31.0868          0 Violations
1     17          31.0344          2 Violations
1     17          31.0535          0 Violations
.
.
.
```

```
1    398          35.1051              0 Violations
1    399          35.1020              0 Violations
1    400          35.1316              0 Violations
1      1          35.1692              0 Violations
1      2          35.1895              0 Violations
1      3          35.2101              0 Violations
.
.
.
1    398          43.3264              0 Violations
1    399          43.3426              0 Violations
1    400          43.3394              0 Violations
1      1   1      43.3394      43.3394 Conforms
1      1  67      43.3422      43.3422
1      1  19      43.3423      43.3423
1      1  56      43.3425      43.3425
1      1  57      43.3450      43.3450
1        End      43.3417
```

This iteration history has a pattern very similar to what we saw in the previous example with the nudge approach. In the first pass through the design, not all restrictions are met. Even the whack approach does not guarantee that all restrictions will be met right away. In the second pass, some rows are processed more than once until all restrictions are met. We can see that in choice sets 13 and 17. By the end of the second pass, all restrictions are met (efficiency starts back up), `%MktEx` realizes all restrictions are met at the end of the third pass, and then it stops. The `%MktEx` macro would have iterated longer if we had not specified `options=largedesign`, but for now when you are testing a new restrictions macro, it is good to check the results before `%MktEx` spends a long time iterating. This next bit of code creates a choice design from this linear candidate set in the familiar way.

```
%mktkey(5 15)

%mktroll(design=cand, key=key, out=rolled)

%mktdups(generic, data=rolled, out=nodups, factors=x1-x15, nalts=5)

%choiceff(data=nodups, model=class(x1-x15), seed=513,
          iter=10, nsets=15, nalts=5, options=nodups, beta=zero)

proc print data=best(obs=15); id set; by notsorted set; var x1-x15; run;
```

Here is the key with 5 rows, 15 columns and the variable names `x1 - x75`.

```
x1   x2   x3   x4   x5   x6   x7   x8   x9   x10  x11  x12  x13  x14  x15

x1   x2   x3   x4   x5   x6   x7   x8   x9   x10  x11  x12  x13  x14  x15
x16  x17  x18  x19  x20  x21  x22  x23  x24  x25  x26  x27  x28  x29  x30
x31  x32  x33  x34  x35  x36  x37  x38  x39  x40  x41  x42  x43  x44  x45
x46  x47  x48  x49  x50  x51  x52  x53  x54  x55  x56  x57  x58  x59  x60
x61  x62  x63  x64  x65  x66  x67  x68  x69  x70  x71  x72  x73  x74  x75
```

Skipping the `%ChoicEff` macro output for a moment, here is part of the choice design.

```
 Set  x1  x2  x3  x4  x5  x6  x7  x8  x9  x10  x11  x12  x13  x14  x15

  73   3   5   1   5   4   2   2   4   1   2    3    3    3    2    2
       3   1   2   5   4   5   2   4   1   2    3    3    1    2    2
       3   3   4   5   4   5   2   4   1   1    3    3    2    2    2
       3   4   4   5   4   4   2   4   1   1    3    3    1    2    2
       3   2   5   5   4   1   2   4   1   4    3    3    5    2    2

 131   4   1   4   1   4   5   2   5   1   3    3    1    5    5    5
       4   1   1   5   4   5   4   5   1   3    3    1    5    5    3
       4   1   2   3   4   5   4   5   1   3    3    1    5    3    1
       4   1   5   2   4   5   5   5   1   3    3    1    5    1    4
       4   1   3   5   4   5   3   5   1   3    3    1    5    2    2

  69   3   3   3   3   5   4   2   2   2   5    4    4    4    1    3
       2   3   3   3   5   2   4   2   2   5    4    3    4    1    4
       5   3   3   3   5   4   3   2   2   5    4    1    4    1    2
       1   3   3   3   5   3   5   2   2   5    4    5    4    1    4
       4   3   3   3   5   5   1   2   2   5    4    2    4    1    5
```

The pattern of constant and nonconstant attributes looks correct: 10 constant and 5 nonconstant attributes per choice set. Furthermore, the constant attributes have the full range of levels. Here is the last table from the output from the `%ChoicEff` macro.

n	Variable Name	Label	Variance	DF	Standard Error
1	x11	x1 1	30.2557	1	5.50052
2	x12	x1 2	21.1270	1	4.59642
3	x13	x1 3	35.2941	1	5.94089
4	x14	x1 4	9.1106	1	3.01838
5	x21	x2 1	16.9166	1	4.11298
6	x22	x2 2	12.3298	1	3.51139
7	x23	x2 3	9.3354	1	3.05539
8	x24	x2 4	7.8348	1	2.79907
9	x31	x3 1	6.0168	1	2.45291
10	x32	x3 2	13.3475	1	3.65342
11	x33	x3 3	15.2546	1	3.90572
12	x34	x3 4	17.5629	1	4.19081
13	x41	x4 1	11.8263	1	3.43894
14	x42	x4 2	8.8253	1	2.97074
15	x43	x4 3	17.6600	1	4.20239
16	x44	x4 4	17.6338	1	4.19927
17	x51	x5 1	19.3639	1	4.40044
18	x52	x5 2	38.2315	1	6.18316
19	x53	x5 3	32.0957	1	5.66530
20	x54	x5 4	34.3440	1	5.86037
21	x61	x6 1	31.6916	1	5.62953
22	x62	x6 2	7.3134	1	2.70432
23	x63	x6 3	27.6641	1	5.25967
24	x64	x6 4	11.6611	1	3.41484
25	x71	x7 1	30.1417	1	5.49015
26	x72	x7 2	18.7806	1	4.33366
27	x73	x7 3	15.5209	1	3.93966
28	x74	x7 4	32.2130	1	5.67565
29	x81	x8 1	42.3192	1	6.50532
30	x82	x8 2	8.8859	1	2.98093
31	x83	x8 3	36.0899	1	6.00749
32	x84	x8 4	23.2898	1	4.82595
33	x91	x9 1	28.2001	1	5.31038
34	x92	x9 2	34.2759	1	5.85457
35	x93	x9 3	21.7287	1	4.66140
36	x94	x9 4	25.2429	1	5.02424
37	x101	x10 1	40.3210	1	6.34989
38	x102	x10 2	25.3427	1	5.03415
39	x103	x10 3	29.0168	1	5.38673
40	x104	x10 4	30.7894	1	5.54882

41	x111	x11 1	19.4994	1	4.41582
42	x112	x11 2	18.6905	1	4.32325
43	x113	x11 3	26.7441	1	5.17147
44	x114	x11 4	26.6769	1	5.16497
45	x121	x12 1	12.7185	1	3.56629
46	x122	x12 2	15.6341	1	3.95400
47	x123	x12 3	30.6642	1	5.53753
48	x124	x12 4	17.0459	1	4.12867
49	x131	x13 1	28.7573	1	5.36258
50	x132	x13 2	33.5044	1	5.78830
51	x133	x13 3	39.3545	1	6.27332
52	x134	x13 4	25.4582	1	5.04561
53	x141	x14 1	32.6395	1	5.71310
54	x142	x14 2	17.8742	1	4.22779
55	x143	x14 3	13.8981	1	3.72802
56	x144	x14 4	14.8683	1	3.85594
57	x151	x15 1	14.6865	1	3.83229
58	x152	x15 2	12.3377	1	3.51250
59	x153	x15 3	9.8278	1	3.13494
60	x154	x15 4	21.0609	1	4.58922
				==	
				60	

Choice experiment designers frequently ask the questions: "How good is this choice experiment? Is it efficient enough?" One of the challenges of designing a choice experiment is determining the answers to these questions. Our *D*-efficiency value is essentially scale-less. In the linear model experiment, we have a hypothetical maximum *D*-efficiency that we use to scale *D*-efficiency to a 0 to 100 scale. We do not usually have that in choice experiments (see page 440 for an exception). One way to assess the quality of the design is to look at the parameter variances. In this table they seem large and variable. This is usually not a good sign. This run of the `%ChoicEff` macro requested a design with only 15 choice sets, which is not a lot. Let's try again, this time with 30 choice sets.

```
%choiceff(data=nodups, model=class(x1-x15), seed=513,
          iter=10, nsets=30, nalts=5, options=nodups, beta=zero)
```

Here is the new parameter variance table.

---

n	Variable Name	Label	Variance	DF	Standard Error
1	x11	x1 1	1.82750	1	1.35185
2	x12	x1 2	1.52927	1	1.23663
3	x13	x1 3	1.50894	1	1.22839
4	x14	x1 4	1.28186	1	1.13219
5	x21	x2 1	1.67064	1	1.29253
6	x22	x2 2	2.25926	1	1.50308
7	x23	x2 3	1.76088	1	1.32698
8	x24	x2 4	1.69763	1	1.30293
9	x31	x3 1	1.59737	1	1.26387
10	x32	x3 2	2.02124	1	1.42170

11	x33	x3 3	1.67137	1	1.29281
12	x34	x3 4	1.87473	1	1.36921
13	x41	x4 1	1.40549	1	1.18553
14	x42	x4 2	1.22398	1	1.10634
15	x43	x4 3	1.16179	1	1.07786
16	x44	x4 4	1.41207	1	1.18831
17	x51	x5 1	1.54385	1	1.24252
18	x52	x5 2	1.60084	1	1.26524
19	x53	x5 3	1.70746	1	1.30670
20	x54	x5 4	1.42809	1	1.19503
21	x61	x6 1	1.74385	1	1.32055
22	x62	x6 2	1.76563	1	1.32877
23	x63	x6 3	1.49271	1	1.22177
24	x64	x6 4	1.68579	1	1.29838
25	x71	x7 1	2.07147	1	1.43926
26	x72	x7 2	2.13145	1	1.45995
27	x73	x7 3	1.95383	1	1.39779
28	x74	x7 4	1.81710	1	1.34800
29	x81	x8 1	1.42543	1	1.19391
30	x82	x8 2	1.71120	1	1.30813
31	x83	x8 3	1.72912	1	1.31496
32	x84	x8 4	1.65668	1	1.28712
33	x91	x9 1	1.66802	1	1.29152
34	x92	x9 2	1.66456	1	1.29018
35	x93	x9 3	1.84660	1	1.35890
36	x94	x9 4	2.06666	1	1.43759
37	x101	x10 1	1.69240	1	1.30092
38	x102	x10 2	1.50033	1	1.22488
39	x103	x10 3	1.56721	1	1.25188
40	x104	x10 4	1.50211	1	1.22561
41	x111	x11 1	1.36458	1	1.16815
42	x112	x11 2	1.51382	1	1.23037
43	x113	x11 3	1.79314	1	1.33908
44	x114	x11 4	1.60204	1	1.26572
45	x121	x12 1	1.61251	1	1.26985
46	x122	x12 2	1.45537	1	1.20639
47	x123	x12 3	1.58978	1	1.26087
48	x124	x12 4	1.43593	1	1.19830
49	x131	x13 1	1.50986	1	1.22876
50	x132	x13 2	1.64192	1	1.28138

51	x133	x13 3	1.69662	1	1.30254
52	x134	x13 4	1.47618	1	1.21498
53	x141	x14 1	1.42358	1	1.19314
54	x142	x14 2	1.42641	1	1.19433
55	x143	x14 3	1.22545	1	1.10700
56	x144	x14 4	1.24266	1	1.11474
57	x151	x15 1	1.88147	1	1.37167
58	x152	x15 2	2.12868	1	1.45900
59	x153	x15 3	1.61916	1	1.27246
60	x154	x15 4	2.44415	1	1.56338
				==	
				60	

This looks much better. The variances are smaller and more uniform. Note that it would be good to run the `%MktEx` macro again without `options=largedesign` and allow it to iterate more before making the final choice design.

Next, we will investigate another thing you can try. Typically, the `%MktEx` macro is run so that it loops over all of the columns in a row, and then it goes on to the next row. Alternatively, it can work with *pairs* of columns at one time using the `exchange=2` option. Working with pairs of columns instead of single columns is always much slower, but sometimes it can make better designs. Here is the code.

```
%macro partprof;
   sum = 0;
   do k = 1 to 15;
      sum = sum + (x[k+15] = x[k] & x[k+30] = x[k] &
                   x[k+45] = x[k] & x[k+60] = x[k]);
      end;
   bad = abs(sum - 10);
   if sum < 10 then do;
      if x[j1] ^= x[mod(j1 - 1, 15) + 1] then bad = bad + 1000;
      if x[j2] ^= x[mod(j2 - 1, 15) + 1] then bad = bad + 1000;
      end;
   %mend;

%mktex(5 ** 75, n=400, optiter=0, tabiter=0, maxtime=720, order=random=15,
       out=sasuser.cand, restrictions=partprof, seed=472, exchange=2,
       maxstages=1, options=largedesign nosort resrep)
```

The initial quantification of badness is the same. Like before, nonconforming levels are whacked. This time however, they are whacked in two ways−when `j1`, the primary column index points to a nonconstant level, and when `j2`, the secondary column index for the pair-wise exchange indexes a nonconstant level. In the `%MktEx` invocation, we now see `maxtime=720` so that `%MktEx` can run over night for 12 hours (or 720 minutes). We also see `exchange=2` for pair-wise exchanges. The output data set is stored as a permanent SAS data set in the `sasuser` library. If we search for 12 hours for a design, we want to make sure it is there for us if we accidentally trip over the power cord in the morning before we have had our coffee. See page 163 for more information on permanent SAS data sets.

There is one more option that we have not used previously, `order=random=15`. This is a special variation on `order=random` for pair-wise exchanges in partial-profile designs. Before this option is explained, here is a bit of background. Sequential pair-wise exchanges for m factors works like this:

`%MktEx` sets `j1` = 1, 2, 3, ..., m and `j2` = `j1` + 1, `j1` + 2, ..., m. Together, the variables `j1` and `j2` loop over all pairs of columns. Random pair-wise exchanges works like this: this: `%MktEx` sets `j1` = random_permutation(1, 2, 3, ..., m) and `j2` = random_permutation(`j1` + 1, `j1` + 2, ..., m). Together, the variables `j1` and `j2` loop over all pairs of columns but in a random order. For partial profiles, pair-wise exchanges are appealing, because sometimes there is a lot to be gained by having two values change at once. However, it does not make sense to consider simultaneously changing the level of a nonconstant attribute and the level of a constant attribute, nor does it make sense to consider pair-wise exchanges within constant attributes. Random exchange with a value of 15 specified works like this: `%MktEx` sets `j1` = random_permutation(1, 2, 3, ..., m) and `j2` is set to a sequential list of the other factors in the same attribute as `j1`. For example, when `j1` = 18, which means `j1` is indexing the second alternative (18 is in the second block of 15 factors) for the third attribute (18 is 3 beyond the fifteenth factor, which is the end of the first block), then `j2` = 3, 18, 33, 48, and 63 for a nonconstant third attribute (which index the 5 factors that make up the third attribute) and `j2` = `j1` for constant attributes. This does pair-wise exchanges but only within nonconstant attributes. This eliminates a lot of uninteresting pairs from consideration.

Here is a small part of the iteration history.

Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
1	Start	58.6611		Ran,Mut,Ann
1	1	58.6806		0 Violations
1	2	58.7175		0 Violations
1	3	58.7108		1 Violations
1	4	58.7007		0 Violations
1	5	58.6681		1 Violations
1	6	58.6568		2 Violations
.				
.				
.				
1	398	31.4906		4 Violations
1	399	31.3437		0 Violations
1	400	31.2368		0 Violations
1	1	31.3070		0 Violations
1	2	31.3869		0 Violations
1	3	31.4504		0 Violations
1	4	31.5318		0 Violations
1	5	31.5681		0 Violations
1	6	31.6076		0 Violations
1	7	31.6189		1 Violations
1	7	31.6396		0 Violations
1	8	31.6282		1 Violations
1	8	31.6907		0 Violations

```
.
.
.
1    398          34.7965              0 Violations
1    399          34.8517              0 Violations
1    400          34.8992              0 Violations
1      1          34.9343              0 Violations
1      2          34.9673              0 Violations
1      3          34.9941              0 Violations
.
.
.
1    397          43.2570              0 Violations
1    398          43.2617              0 Violations
1    399   1      43.2617      43.2617 Conforms
1    399   7      43.2617      43.2617
1    399  69      43.2665      43.2665
1    399  69      43.2665      43.2665
1    399  48      43.2674      43.2674
1    399  35      43.2701      43.2701
.
.
.
1    271  29      49.0081      49.0081
1    271  59      49.0081      49.0081
1    271  33      49.0081      49.0081
1        End     49.0061
```

It is followed by these messages.

```
NOTE: Stopping early, possibly before convergence, with a large design.
NOTE: Quitting the algorithm search step after 720.07 minutes and 22 designs.
```

These next steps make the choice design.

```
%mktkey(5 15)

%mktroll(design=sasuser.cand, key=key, out=rolled)

%mktdups(generic, data=rolled, out=nodups, factors=x1-x15, nalts=5)

%choiceff(data=nodups, model=class(x1-x15), seed=513,
          iter=10, nsets=30, nalts=5, options=nodups, beta=zero)

proc print data=best(obs=15); id set; by notsorted set; var x1-x15; run;
```

Here is the variance table.

n	Variable Name	Label	Variance	DF	Standard Error
1	x11	x1 1	1.55680	1	1.24772
2	x12	x1 2	1.76682	1	1.32922
3	x13	x1 3	1.50404	1	1.22639
4	x14	x1 4	1.43339	1	1.19724
5	x21	x2 1	1.37856	1	1.17412
6	x22	x2 2	1.62243	1	1.27375
7	x23	x2 3	1.52808	1	1.23615
8	x24	x2 4	1.55253	1	1.24601
9	x31	x3 1	1.79120	1	1.33836
10	x32	x3 2	1.93540	1	1.39119
11	x33	x3 3	1.75466	1	1.32463
12	x34	x3 4	1.81796	1	1.34832
13	x41	x4 1	1.63862	1	1.28009
14	x42	x4 2	1.59684	1	1.26366
15	x43	x4 3	1.58467	1	1.25884
16	x44	x4 4	1.65078	1	1.28483
17	x51	x5 1	1.50650	1	1.22740
18	x52	x5 2	1.63131	1	1.27723
19	x53	x5 3	1.52405	1	1.23452
20	x54	x5 4	1.71268	1	1.30870
21	x61	x6 1	1.37164	1	1.17117
22	x62	x6 2	1.59728	1	1.26384
23	x63	x6 3	1.27588	1	1.12955
24	x64	x6 4	1.31074	1	1.14488
25	x71	x7 1	1.67753	1	1.29519
26	x72	x7 2	1.86514	1	1.36570
27	x73	x7 3	1.34253	1	1.15868
28	x74	x7 4	1.53091	1	1.23730
29	x81	x8 1	1.36565	1	1.16861
30	x82	x8 2	1.43040	1	1.19599
31	x83	x8 3	1.42614	1	1.19421
32	x84	x8 4	1.48790	1	1.21980
33	x91	x9 1	1.77503	1	1.33230
34	x92	x9 2	1.29440	1	1.13772
35	x93	x9 3	1.54699	1	1.24378
36	x94	x9 4	1.39202	1	1.17984
37	x101	x10 1	1.26205	1	1.12341
38	x102	x10 2	1.19272	1	1.09212
39	x103	x10 3	1.36649	1	1.16897
40	x104	x10 4	1.54890	1	1.24455

41	x111	x11 1	1.81033	1	1.34548
42	x112	x11 2	1.41572	1	1.18984
43	x113	x11 3	1.53332	1	1.23827
44	x114	x11 4	1.61646	1	1.27140
45	x121	x12 1	1.55783	1	1.24813
46	x122	x12 2	1.68065	1	1.29640
47	x123	x12 3	1.60454	1	1.26670
48	x124	x12 4	1.36580	1	1.16867
49	x131	x13 1	1.70258	1	1.30483
50	x132	x13 2	1.70672	1	1.30641
51	x133	x13 3	1.56914	1	1.25265
52	x134	x13 4	1.60295	1	1.26608
53	x141	x14 1	1.64414	1	1.28224
54	x142	x14 2	1.43348	1	1.19728
55	x143	x14 3	1.29226	1	1.13678
56	x144	x14 4	1.45448	1	1.20602
57	x151	x15 1	1.62811	1	1.27597
58	x152	x15 2	1.99746	1	1.41332
59	x153	x15 3	1.63405	1	1.27830
60	x154	x15 4	1.76144	1	1.32719
				==	
				60	

Here are the first few choice sets.

Set	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15
257	2	3	1	1	2	4	4	5	1	1	2	1	4	3	3
	2	3	1	4	4	4	4	2	1	4	2	1	4	2	3
	2	3	1	3	1	4	4	1	1	5	2	1	4	5	3
	2	3	1	2	3	4	4	4	1	3	2	1	4	4	3
	2	3	1	4	5	4	4	3	1	2	2	1	4	1	3
290	5	5	5	5	3	4	2	2	5	5	3	5	5	3	3
	5	5	5	5	2	4	2	1	5	2	3	5	5	3	2
	5	5	5	5	4	4	2	5	5	2	3	5	4	3	5
	5	5	5	5	1	4	2	3	5	4	3	5	1	3	4
	5	5	5	5	5	4	2	4	5	1	3	5	3	3	1
261	3	3	1	2	5	1	1	4	2	5	1	4	5	3	1
	1	2	2	2	5	1	1	3	2	5	1	4	3	3	1
	2	4	3	2	5	1	1	2	2	5	1	4	1	3	1
	4	5	4	2	5	1	1	1	2	5	1	4	4	3	1
	4	1	1	2	5	1	1	5	2	5	1	4	2	3	1

The example starting on page 412 and the choice design shown on page 416 creates a partial-profile design with all constant attributes equal to one. In contrast, this design uses the full range of values for the constant attributes. In terms of fitting the choice model, it does not matter. An attribute that is

constant within a choice set does not contribute to the likelihood function for that choice set, and this is true no matter what the constant value is. It typically will not matter for data collection either, since data collection is typically phrased in terms like "everything else being equal" without any specifics about what the equal levels are. About the only difference is in the `%MktEx` macro. *D*-efficiency should be higher with the varying-constant approach than with the all-one approach.

Partial Profiles and Incomplete Blocks Designs

This next example makes a partial-profile design from a balanced incomplete blocks design (BIBD) and a small orthogonal array. See page 443 for more information on BIBD.

80 Choice Sets, 16 Binary Attributes, Four Varying. In this example, we create a partial-profile design with 16 binary attributes and four varying at one time. We create 80 choice sets of two alternatives each. The resulting design is optimal under the assumption $\boldsymbol{\beta} = \mathbf{0}$ (Anderson, 2003).

The following code creates and displays the design, using ad hoc code. This code is provided for those who want to better understand what is going on.

```
proc iml;
   d = { 1 1 1 1 ,
         1 1 2 2 ,
         1 2 1 2 ,
         1 2 2 1 };
   b = { 1  1  1  1  1  2  2  2  2  3  3  3  3  4  4  4  4  5  6  7 ,
         2  5  8 11 14  5  6  7 10  5  6  7  9  5  6  7  8  9 10  8 ,
         3  6  9 12 15  8 12  9 13 13  8 10 11 10  9 11 12 12 11 13 ,
         4  7 10 13 16 11 14 15 16 15 16 12 14 14 13 16 15 16 15 14 }‘;
   m = max(b); p = nrow(d);
   sets = nrow(b) # p;
   d2 = mod(d, 2) + 1;
   x  = j(sets, 2 # m, 1);
   do i = 1 to nrow(b);
      j = ((i-1) # p : i # p - 1) + 1;
      x[j, b[i,]    ] = d;
      x[j, b[i,] + m] = d2;
      end;
   x = (1:sets)‘ @ {1 1}‘ || shape(x, 2 # sets);
   m = ncol(x) - 1;
   vname = ’Set’ || (’x1’ : rowcatc(’x’ || char(m)));
   create design from x[colname=vname];
   append from x;
   quit;

proc print; by set; id set; var x:; where set le 8 or set ge 77; run;
```

Here is an easier but much less explicit way using the `%MktPPro` macro and the `ibd=` option. The `%MktPPro` macro is described in more detail on page 453.

```
%mktex(2 4 2 2 2, n=8)

proc print; run;

data design; set design; drop x2; run;

proc iml;
   b = { 1  1  1  1  1  2  2  2  2  3  3  3  3  4  4  4  4  5  6  7 ,
         2  5  8 11 14  5  6  7 10  5  6  7  9  5  6  7  8  9 10  8 ,
         3  6  9 12 15  8 12  9 13 13  8 10 11 10  9 11 12 12 11 13 ,
         4  7 10 13 16 11 14 15 16 15 16 12 14 14 13 16 15 16 15 14 }‘;
   create b from b; append from b;
   quit;

%mktppro(ibd=b)
```

Returning to the ad hoc code, the matrix `d` contains one half of an orthogonal array with 4 two-level factors in 8 runs. It will be used to make the first alternative in each choice set. The second alternative is the second half of the orthogonal array which is constructed from the first half using the statement `d2 = mod(d, 2) + 1`. Recall the chair example on page 370 where the optimal generic choice design was created by the `%ChoicEff` macro by sorting the rows of an orthogonal array into choice sets. Here, an optimal generic design for four choice sets, each with two alternatives, and four binary attributes is made by pairing each of the four alternatives in `d` with one of the remaining runs in the orthogonal array (the run that can be made by shifting the first alternative: $0 \rightarrow 1$ and $1 \rightarrow 0$). Here is the optimal generic design for four choice sets and four binary attributes.

```
Set
  1  1 1 1 1
     2 2 2 2

  2  1 1 2 2
     2 2 1 1

  3  1 2 1 2
     2 1 2 1

  4  1 2 2 1
     2 1 1 2
```

The BIBD is stored in the matrix `b`, which is transposed, so it has 20 rows and four columns, and the first row is `(1 2 3 4)`. This first row dictates that the first four choice sets are to be made from the first four attributes and the optimal generic design above. The second set of four choice sets is made by storing the four-factor design in attributes 1, 5, 6, and 7. The last four choice sets are made varying attributes 7, 8, 13, and 14 in the same way. Here are the first eight and last four choice sets.

Set	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1

2	1	1	2	2	1	1	1	1	1	1	1	1	1	1	1	1
	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3	1	2	1	2	1	1	1	1	1	1	1	1	1	1	1	1
	2	1	2	1	1	1	1	1	1	1	1	1	1	1	1	1
4	1	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1
	2	1	1	2	1	1	1	1	1	1	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2	1	1	1	2	2	2	1	1	1	1	1	1	1	1	1
6	1	1	1	1	1	2	2	1	1	1	1	1	1	1	1	1
	2	1	1	1	2	1	1	1	1	1	1	1	1	1	1	1
7	1	1	1	1	2	1	2	1	1	1	1	1	1	1	1	1
	2	1	1	1	1	2	1	1	1	1	1	1	1	1	1	1
8	1	1	1	1	2	2	1	1	1	1	1	1	1	1	1	1
	2	1	1	1	1	1	2	1	1	1	1	1	1	1	1	1
77	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	1	1	1	1	1	1	2	2	1	1	1	1	2	2	1	1
78	1	1	1	1	1	1	1	1	1	1	1	1	2	2	1	1
	1	1	1	1	1	1	2	2	1	1	1	1	1	1	1	1
79	1	1	1	1	1	1	1	2	1	1	1	1	1	2	1	1
	1	1	1	1	1	1	2	1	1	1	1	1	2	1	1	1
80	1	1	1	1	1	1	1	2	1	1	1	1	2	1	1	1
	1	1	1	1	1	1	2	1	1	1	1	1	1	2	1	1

Notice that in the first four choice sets, the first four attributes vary and the last 12 are constant. In the last four choice sets, only attributes 7, 8, 13, and 14 vary.

Let's return to the IML code. The first alternatives are stored in `d`, and the second in `d2`, which is made from `d` by shifting. The matrix `b` contains the BIBD, and `x` will contain the choice design. Like when we make linear designs, we will start with one row per choice set and then convert to one row for each alternative of each choice set, so `x` starts out as 80 rows or choice sets and 2×16 attributes for 32 columns. For the first row in `b`, rows `j = (1:4)` of `x` are filled in, for the second row in `b`, rows `j = (5:8)` of `x` are filled in, and so on. The first 16 columns of `x` are filled in using the rows of `d` using the entries in `b[1]` as column indices, and the second 16 columns of `x` are filled in using the rows of `d2` and again using the entries in `b[1]` as column indices. The statement `x = (1:sets)‘ @ {1 1}‘ || shape(x, 2 # sets)` adds the choice set number and rolls out each row of 32 attributes into two rows of 16 attributes. The remaining IML statements output the design to a SAS data set with variable names `Set x1-x16`. These statements evaluate the design.

```
%choiceff(data=chdes, model=class(x1-x16), nsets=80, nalts=2,
          beta=zero, init=chdes, initvars=x1-x16)
```

Here is the last part of the output.

Design	Iteration	D-Efficiency	D-Error
1	0	5.000000	0.200000
	1	5.000000	0.200000

Final Results

Design	1
Choice Sets	80
Alternatives	2
D-Efficiency	5.000000
D-Error	0.200000

n	Variable Name	Label	Variance	DF	Standard Error
1	x11	x1 1	0.2	1	0.44721
2	x21	x2 1	0.2	1	0.44721
3	x31	x3 1	0.2	1	0.44721
4	x41	x4 1	0.2	1	0.44721
5	x51	x5 1	0.2	1	0.44721
6	x61	x6 1	0.2	1	0.44721
7	x71	x7 1	0.2	1	0.44721
8	x81	x8 1	0.2	1	0.44721
9	x91	x9 1	0.2	1	0.44721
10	x101	x10 1	0.2	1	0.44721
11	x111	x11 1	0.2	1	0.44721
12	x121	x12 1	0.2	1	0.44721
13	x131	x13 1	0.2	1	0.44721
14	x141	x14 1	0.2	1	0.44721
15	x151	x15 1	0.2	1	0.44721
16	x161	x16 1	0.2	1	0.44721
				==	
				16	

The `%ChoiceEff` macro cannot improve on the design, since it is optimal, and the variances are all constant. On page 428 we talk about not having a scale for D-efficiency in choice models. While that is usually true, it is not true for special cases like this where the optimal design is known. This design has 100% D-efficiency for a partial-profile design with 80 choice sets, two alternatives, 16 binary attributes with four varying, and $\boldsymbol{\beta} = \mathbf{0}$. We can also use methods like we have seen in previous examples to construct a design for this situation and compare the results. Here is the code. Since we only have two alternatives, our restrictions are fairly simple.

```
%macro partprof;
   sum = 0;
   do k = 1 to 16;
      sum = sum + (x[k] = x[k+16]);
      end;
   bad = abs(sum - 12);
   %mend;
%mktex(2 ** 32, n=1000, optiter=0, tabiter=0, order=random,
       out=sasuser.cand, restrictions=partprof, seed=382)
%mktkey(2 16)

%mktroll(design=sasuser.cand, key=key, out=rolled)

%mktdups(generic, data=rolled, out=nodups, factors=x1-x16, nalts=2)

%choiceff(data=nodups, model=class(x1-x16), seed=495,
          iter=20, nsets=80, nalts=2, options=nodups, beta=zero)

proc print data=best; id set; by notsorted set; var x:; run;
```

The `%MktEx` step ran in about 26 minutes and created 1000 candidate choice sets with a *D*-efficiency of 85.8857%. The `%MktDups` macro detected no duplicate choice sets or alternatives, and the `%ChoicEff` macro ran in just over one minute. Here is the last part of the output.

Final Results

Design	15
Choice Sets	80
Alternatives	2
D-Efficiency	4.977221
D-Error	0.200915

n	Variable Name	Label	Variance	DF	Standard Error
1	x11	x1 1	0.20203	1	0.44947
2	x21	x2 1	0.20104	1	0.44837
3	x31	x3 1	0.20094	1	0.44827
4	x41	x4 1	0.21109	1	0.45944
5	x51	x5 1	0.21214	1	0.46059
6	x61	x6 1	0.20099	1	0.44832
7	x71	x7 1	0.20194	1	0.44938
8	x81	x8 1	0.20203	1	0.44948
9	x91	x9 1	0.20097	1	0.44829
10	x101	x10 1	0.20285	1	0.45039
11	x111	x11 1	0.21104	1	0.45939
12	x121	x12 1	0.20102	1	0.44835
13	x131	x13 1	0.18431	1	0.42931
14	x141	x14 1	0.20206	1	0.44951
15	x151	x15 1	0.19272	1	0.43900
16	x161	x16 1	0.20183	1	0.44926
				==	
				16	

If this design were optimal, *D*-efficiency would be 5 and all of the variances would be 0.2. Instead, our *D*-efficiency is 4.997 and the variances are slightly larger on the average. Computing $100 \times 4.977221/5$ we see that our iteratively-derived partial-profile design is 99.54% *D*-efficient. These next three steps make successively smaller designs.

```
%choiceff(data=nodups, model=class(x1-x16), seed=495,
          iter=20, nsets=60, nalts=2, options=nodups, beta=zero)

%choiceff(data=nodups, model=class(x1-x16), seed=495,
          iter=20, nsets=40, nalts=2, options=nodups, beta=zero)

%choiceff(data=nodups, model=class(x1-x16), seed=495,
          iter=20, nsets=20, nalts=2, options=nodups, beta=zero)
```

Here are the "Final Results" tables for the 80-run design above and the 60, 40, and 20-run designs created here.

Final Results

Design	15
Choice Sets	80
Alternatives	2
D-Efficiency	4.977221
D-Error	0.200915

```
      Final Results

Design                10
Choice Sets           60
Alternatives           2
D-Efficiency     3.716947
D-Error          0.269038

      Final Results

Design                16
Choice Sets           40
Alternatives           2
D-Efficiency     2.452105
D-Error          0.407813

      Final Results

Design                12
Choice Sets           20
Alternatives           2
D-Efficiency     1.150469
D-Error          0.869211
```

Computing $100 \times (80/n) \times (d/5)$, where there are 80 choice sets in the optimal design with D-efficiency 5 and n choice sets in the iteratively derived design with D-efficiency d, we see that our design relative D-efficiencies are: 99.544% in 80 sets, 99.119% in 60 sets, 98.084% in 40 sets, and 92.038% in 20 sets, all relative to a (real in 80 sets and hypothetical in 60, 40, and 20 sets) optimal design in n sets. For most problems, we will never be able to derive numbers like this. However it is reassuring to see them when it is possible, and to see that they are this high. Here are the four tables with the variances for 80, 60, 40, and 20 sets.

n	Variable Name	Label	Variance	DF	Standard Error
1	x11	x1 1	0.20203	1	0.44947
2	x21	x2 1	0.20104	1	0.44837
3	x31	x3 1	0.20094	1	0.44827
4	x41	x4 1	0.21109	1	0.45944
5	x51	x5 1	0.21214	1	0.46059
6	x61	x6 1	0.20099	1	0.44832
7	x71	x7 1	0.20194	1	0.44938
8	x81	x8 1	0.20203	1	0.44948
9	x91	x9 1	0.20097	1	0.44829
10	x101	x10 1	0.20285	1	0.45039
11	x111	x11 1	0.21104	1	0.45939
12	x121	x12 1	0.20102	1	0.44835
13	x131	x13 1	0.18431	1	0.42931
14	x141	x14 1	0.20206	1	0.44951
15	x151	x15 1	0.19272	1	0.43900
16	x161	x16 1	0.20183	1	0.44926
				==	
				16	

n	Variable Name	Label	Variance	DF	Standard Error
1	x11	x1 1	0.26788	1	0.51757
2	x21	x2 1	0.27031	1	0.51992
3	x31	x3 1	0.26788	1	0.51757
4	x41	x4 1	0.29097	1	0.53941
5	x51	x5 1	0.27508	1	0.52449
6	x61	x6 1	0.26789	1	0.51758
7	x71	x7 1	0.27268	1	0.52218
8	x81	x8 1	0.27469	1	0.52411
9	x91	x9 1	0.27032	1	0.51992
10	x101	x10 1	0.27253	1	0.52205
11	x111	x11 1	0.25213	1	0.50212
12	x121	x12 1	0.27496	1	0.52436
13	x131	x13 1	0.25414	1	0.50413
14	x141	x14 1	0.26787	1	0.51756
15	x151	x15 1	0.29131	1	0.53973
16	x161	x16 1	0.27221	1	0.52174
				==	
				16	

n	Variable Name	Label	Variance	DF	Standard Error
1	x11	x1 1	0.40769	1	0.63851
2	x21	x2 1	0.40883	1	0.63940
3	x31	x3 1	0.41634	1	0.64524
4	x41	x4 1	0.41625	1	0.64517
5	x51	x5 1	0.41603	1	0.64501
6	x61	x6 1	0.45934	1	0.67775
7	x71	x7 1	0.40876	1	0.63934
8	x81	x8 1	0.44949	1	0.67044
9	x91	x9 1	0.35011	1	0.59170
10	x101	x10 1	0.42390	1	0.65107
11	x111	x11 1	0.45768	1	0.67652
12	x121	x12 1	0.42318	1	0.65052
13	x131	x13 1	0.41484	1	0.64408
14	x141	x14 1	0.40841	1	0.63907
15	x151	x15 1	0.41452	1	0.64383
16	x161	x16 1	0.37300	1	0.61073
				==	
				16	

n	Variable Name	Label	Variance	DF	Standard Error
1	x11	x1 1	0.72680	1	0.85252
2	x21	x2 1	0.92729	1	0.96296
3	x31	x3 1	0.72155	1	0.84944
4	x41	x4 1	0.94524	1	0.97223
5	x51	x5 1	0.98455	1	0.99224
6	x61	x6 1	0.66667	1	0.81650
7	x71	x7 1	0.79685	1	0.89266
8	x81	x8 1	0.99932	1	0.99966
9	x91	x9 1	0.93214	1	0.96547
10	x101	x10 1	1.32156	1	1.14959
11	x111	x11 1	1.03494	1	1.01732
12	x121	x12 1	0.93766	1	0.96833
13	x131	x13 1	0.83709	1	0.91493
14	x141	x14 1	1.28216	1	1.13232
15	x151	x15 1	1.11837	1	1.05753
16	x161	x16 1	1.12348	1	1.05994
				==	
				16	

At 20 choice sets, we see an increase in the variability of the variances, which is often associated with having too few choice sets.

Let's look in a little more detail at our BIBD. This code tabulates how often attribute j will be paired with attribute k over all $i = 1, ..., 20$ rows in the BIBD.

```
proc iml;
   b  = { 1  1  1  1  1  2  2  2  2  3  3  3  3  4  4  4  4  5  6  7 ,
          2  5  8 11 14  5  6  7 10  5  6  7  9  5  6  7  8  9 10  8 ,
          3  6  9 12 15  8 12  9 13 13  8 10 11 10  9 11 12 12 11 13 ,
          4  7 10 13 16 11 14 15 16 15 16 12 14 14 13 16 15 16 15 14 }‘;
   m = max(b); p = nrow(b); q = ncol(b);
   f = j(m, m, 0);
   do i = 1 to p;
      do j = 1 to q;
         do k = j to q;
            f[b[i,j], b[i,k]] = f[b[i,j], b[i,k]] + 1;
            end;
         end;
      end;
   print f[format=1.];
   x = j(p, m, 0);
   do i = 1 to p; x[i, b[i,]] = 1; end;
   print b[format=2.] ’  ’ x[format=1.];
   quit;
```

Each of the 16 attributes is paired with each of the other 15 attributes exactly once. This is what makes this IBD a balanced IBD or BIBD. Each attribute appears five times (which in our case means that each attribute appears in five blocks of four choice sets for a total of 20 choice sets). The five times comes from the fact that in each row, each attribute is paired with three of the remaining 15, so each attribute must appear $15/3 = 5$ times. In summary, our BIBD consists of 20 quadruples of attributes arranged such that each attribute is paired with every other attribute once, and each attribute appears in the design five times.

F

```
5 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 5 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 5 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 5 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 5 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 5 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 5 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 5 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 5 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 5 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 5 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 5 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 5 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 5 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 5 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 5
```

The PROC IML step also converts the BIBD from 4 columns with attribute numbers into a binary matrix with 16 columns, one for each attribute, that shows which attributes are used and which are not. Here is both the BIBD and the binary matrix.

```
B              X

 1  2  3  4    1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
 1  5  6  7    1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0
 1  8  9 10    1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0
 1 11 12 13    1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0
 1 14 15 16    1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
 2  5  8 11    0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0
 2  6 12 14    0 1 0 0 0 1 0 0 0 0 0 1 0 1 0 0
 2  7  9 15    0 1 0 0 0 0 1 0 1 0 0 0 0 0 1 0
 2 10 13 16    0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 1
 3  5 13 15    0 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0
 3  6  8 16    0 0 1 0 0 1 0 1 0 0 0 0 0 0 0 1
 3  7 10 12    0 0 1 0 0 0 1 0 0 1 0 1 0 0 0 0
 3  9 11 14    0 0 1 0 0 0 0 0 1 0 1 0 0 1 0 0
 4  5 10 14    0 0 0 1 1 0 0 0 0 1 0 0 0 1 0 0
 4  6  9 13    0 0 0 1 0 1 0 0 1 0 0 0 1 0 0 0
 4  7 11 16    0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 1
 4  8 12 15    0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 0
 5  9 12 16    0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 1
 6 10 11 15    0 0 0 0 0 1 0 0 0 1 1 0 0 0 1 0
 7  8 13 14    0 0 0 0 0 0 1 1 0 0 0 0 1 1 0 0
```

80 Choice Sets, 16 Binary Attributes, Four Varying, Part 2. The `%MktEx` macro cannot make a BIBD. However, with a little work and an appropriate restrictions macro, `%MktEx` can be trained to make something close. It can make an IBD, where every pair does not occur equally often, and these can be used to make partial-profile designs. First note that `%MktEx` cannot make an IBD that looks like the left matrix just displayed using a direct approach like specifying 4 sixteen-level factors. In 20 runs, there would be 61 parameters. Instead, we will make a binary matrix, more like the one displayed on the right. We need to make a design in 20 runs with 16 two-level factors and have restrictions so that exactly four factors are two (or not at the base-line level of one). We could do that with a macro like this.

```
%macro res;
   bad = abs(sum(x = 2) - 4);
  %mend;
```

However, our restrictions macro needs to be more complicated. Having four attributes vary is only part of what we need to accomplish. We also would like each attribute to appear in five rows (as shown by the diagonal of the `F` matrix on page 444). We would also like each attribute to be paired with each other attribute exactly once (as shown by the ones in the upper triangle of the `F` matrix. Restrictions macros get called a lot, so it is good to make them as computationally efficient as possible. One way to increase computational efficiency in IML is to use matrix operations instead of do loops as much as is possible. Another way to increase computational efficiency in any language is to move computations

out of loops as much as possible. In other words, if a quantity does no change inside a loop, compute it once outside the loop instead of recomputing it over and over again. This next example does both of these things. The %MktEx macro allows you to call a second restrictions macro once that creates one or more constant matrices that are used in the normal restrictions macro. This second macro is named on the resmac= option and the constant matrices are named on the reslist= option. Here are our restrictions macros.

```
%macro con;
   _f = ((1:&m) @ j(&m, 1, 1) > j(1, &m, 1) @ (1:&m)') + 5 # i(&m);
   _p = (_f = 5) + 10 # (_f = 1);
   %mend;

%macro res;
   bad = 1000 # abs(sum(x = 2) - 4);
   f = j(&m, &m, 0);
   do ii = 1 to &n;
      if ii = i then l = loc(x = 2);
      else l = loc(xmat[ii,] = 2);
      if ncol(l) then f[l, l] = f[l, l] + 1;
      end;
   bad = bad + sum(abs(f - _f ) # _p);
   %mend;
```

We will discuss what these macros do before we discuss how they do it. Our first goal is to whack the design hard when it has the wrong number of varying attributes. Our second goal is to whack the design not quite so hard for not having ones in the frequency matrix f (on page 444) above the diagonal. Our third goal is to nudge the design gently for not having fives in f on the diagonal. We whacked the off diagonals and nudged the diagonal because, if the off diagonals are going to get fixed, the diagonals need some freedom to go up and down for a while.

Our macros both use the %MktEx macro's variables &n and &m, which are the number of rows and columns in the design. The macro con will make two constant matrices for our restrictions macro to use. Both matrices begin with a single underscore, because you must use matrix names that do not conflict with any of the scores of internal matrix names that %MktEx uses. If you create intermediate results in this macro use names that begin with a single underscore for those matrices as well. The first one, _f is exactly equal to the f matrix shown on page 444. The second one, _p contains badness penalties for corresponding cells in _f. There is no penalty associated with any values in the design's f matrix below the diagonal, there is a penalty of ten for each f value that does not match the _f above the diagonal, and there is a penalty of one for each f value that does not match the _f on the diagonal. Here are _f and _p.

```
_F                                  _P

 5 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  1 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
 0 5 1 1 1 1 1 1 1 1 1 1 1 1 1 1  0  1 10 10 10 10 10 10 10 10 10 10 10 10 10 10
 0 0 5 1 1 1 1 1 1 1 1 1 1 1 1 1  0  0  1 10 10 10 10 10 10 10 10 10 10 10 10 10
 0 0 0 5 1 1 1 1 1 1 1 1 1 1 1 1  0  0  0  1 10 10 10 10 10 10 10 10 10 10 10 10
 0 0 0 0 5 1 1 1 1 1 1 1 1 1 1 1  0  0  0  0  1 10 10 10 10 10 10 10 10 10 10 10
 0 0 0 0 0 5 1 1 1 1 1 1 1 1 1 1  0  0  0  0  0  1 10 10 10 10 10 10 10 10 10 10
 0 0 0 0 0 0 5 1 1 1 1 1 1 1 1 1  0  0  0  0  0  0  1 10 10 10 10 10 10 10 10 10
 0 0 0 0 0 0 0 5 1 1 1 1 1 1 1 1  0  0  0  0  0  0  0  1 10 10 10 10 10 10 10 10
 0 0 0 0 0 0 0 0 5 1 1 1 1 1 1 1  0  0  0  0  0  0  0  0  1 10 10 10 10 10 10 10
 0 0 0 0 0 0 0 0 0 5 1 1 1 1 1 1  0  0  0  0  0  0  0  0  0  1 10 10 10 10 10 10
 0 0 0 0 0 0 0 0 0 0 5 1 1 1 1 1  0  0  0  0  0  0  0  0  0  0  1 10 10 10 10 10
 0 0 0 0 0 0 0 0 0 0 0 5 1 1 1 1  0  0  0  0  0  0  0  0  0  0  0  1 10 10 10 10
 0 0 0 0 0 0 0 0 0 0 0 0 5 1 1 1  0  0  0  0  0  0  0  0  0  0  0  0  1 10 10 10
 0 0 0 0 0 0 0 0 0 0 0 0 0 5 1 1  0  0  0  0  0  0  0  0  0  0  0  0  0  1 10 10
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 5 1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1 10
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 5  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1
```

Here are the pieces that go into making _f.

```
%let n = 20;
%let m = 16;
proc iml;
   a = (1:&m);
   b = j(1, &m, 1);
   c = j(&m, 1, 1);
   d = (1:&m)`;
   e = (1:&m) @ j(&m, 1, 1);
   f = j(1, &m, 1) @ (1:&m)`;
   g = (1:&m) @ j(&m, 1, 1) > j(1, &m, 1) @ (1:&m)`;
   h = 5 # i(&m);
   _f = ((1:&m) @ j(&m, 1, 1) > j(1, &m, 1) @ (1:&m)`) + 5 # i(&m);
   print a[format=2.] b[format=2.];
   print c[format=1.] '   ' d[format=2.] '   ' e[format=2.];
   print f[format=2.] g[format=1.] h[format=1.] _f[format=1.];
   quit;
```

```
                                        A

                    1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16

                                        B

                    1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1

          C         D         E

          1         1         1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
          1         2         1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
          1         3         1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
          1         4         1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
          1         5         1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
          1         6         1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
          1         7         1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
          1         8         1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
          1         9         1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
          1        10         1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
          1        11         1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
          1        12         1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
          1        13         1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
          1        14         1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
          1        15         1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
          1        16         1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16

 F                                                                  G

 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1
 4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1
 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
 6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
 7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1
 8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
 9  9  9  9  9  9  9  9  9  9  9  9  9  9  9  9  9 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1
11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1
12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
H                                                _F

5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  5 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 5 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 5 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 5 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 5 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 5 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 5 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 5 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 5 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 5 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 5 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 5 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 5 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 5 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 5 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 5 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 5 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 5 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 5 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 5 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 5 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 5 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 5 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 5  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 5
```

The "`@`" operator specifies Kronecker or direct product. The Kronecker product `A = B @ C` of an $n \times m$ and $p \times q$ matrix is an $np \times mq$ matrix that consists of $n \times m$ submatrices of the form $a_{ij} \times B$. The matrix `e` is the Kronecker product of `a`, an index row vector going from 1 to 16, and `c`, a column vector of 16 ones, which creates a matrix of 16 row index vectors going from 1 to 16. The matrix `f` is the Kronecker product of `b`, a row vector of 16 ones, and `d`, an index column vector going from 1 to 16 which creates a matrix of 16 column index vectors going from 1 to 16. The matrix `f` is the transpose of `e`. The matrix `g` is a Boolean matrix which is upper triangular, and above the diagonal are ones since `e` is greater than `f` above the diagonal. The matrix `h` is an identity matrix times 5, which when added to `g` gives the desired result. The matrix `_p` is constructed from `_f` so that it has zeros where `_f` has zeros, tens where `_f` has ones, and ones where `_f` has fives. Both of these matrices could be constructed in a much more straight-forward way using do loops, and since they are only constructed once, it really would not matter, but it is always good to learn matrix operations. Besides, this way, with a single statement is a lot more fun.

In the `res` restrictions macro, badness is initialized based on the number of varying factors. The next statements compute the `f` matrix of frequencies. Notice that the macro is looking at the full design in `xmat` except when processing the *ith* row, and then it looks in `x`. The `loc` function creates an index vector of all of the columns in its argument vector that are nonzero, so `loc(x = 2)` creates an index vector of all the columns that have twos. For the *ith* row, the rows and columns of `f` for the design columns that have twos are incremented. The statement `if ncol(1) then f[1, 1] = f[1, 1] + 1` adds values both above and below the diagonal, but the below diagonal values are ignored since the penalties are all zero below the diagonal. We could write another `do` loop that just incremented above and on diagonal frequencies, but that would probably be slower. The last statement increments the badness criterion. The multiplication by 1000 in the first line, whacks the design hard for having the wrong number of varying attributes. The tens in `_p` whack the design less hard for not having ones off the diagonal, and the ones in `_p` nudge the design in the direction of having fives on the diagonal. Here is our `%MktEx` call.

```
%mktex(2 ** 16, n=20, tabiter=0, optiter=0,
       restrictions=res, resmac=con, reslist=%str(_f, _p),
       order=random, options=resrep accept, exchange=2, seed=205,
       maxdesigns=1, out=sasuser.ibd2)
```

The macro specifies `restrictions=res, resmac=con, reslist=%str(_f, _p)`. The `resmac=` option names the constant matrix definition macro. The `reslist=` option names the constant matrices. The list `_f, _p` is included in a `%str( )` function since the matrix names must be separated by commas. Another new option is `options=acceptaccept`. This tells `%MktEx` to accept designs that do not meet the restrictions. For this problem, we would be extremely lucky if we found an actual BIBD, and we are willing to accept something close. Here is some of the iteration history.

Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
1	Start	46.3945		Ran,Mut,Ann
1	1	50.9919		4373 Violations
1	2	52.0060		4149 Violations
1	3	54.6365		4149 Violations
1	4	57.6414		3996 Violations
1	5	55.9297		3267 Violations
1	6	57.3744		3043 Violations
1	7	53.0821		2890 Violations
1	8	55.7085		2686 Violations
1	9	54.2112		2441 Violations
1	10	52.6384		2291 Violations
1	11	51.5465		2045 Violations
1	12	53.5948		1912 Violations
1	13	50.6521		1629 Violations
1	14	48.0705		1427 Violations
1	15	51.7720		1264 Violations
1	16	53.0066		1024 Violations
1	17	51.2140		950 Violations
1	18	51.0421		839 Violations
1	19	43.8809		712 Violations
1	20	24.9443		682 Violations
1	1	26.9593		618 Violations
.				
.				
.				
1	10	32.1882		180 Violations
1	11	32.1882		180 Violations
1	12	32.1882		180 Violations
1	13 1	32.1882	32.1882	Acceptable
1	End	32.1882		Violations

One thing that is different from what we have seen previously, is now the number of violations is always evaluated over the entire design, so values start out pretty big. In this example, the %MktEx macro prints some warnings and notes.

```
WARNING: It may be impossible to meet all restrictions.
NOTE: The restrictions were not met.
WARNING: The final efficiency and levels table will not be printed.
WARNING: It appears that the design has zero or near-zero efficiency.
WARNING: Values in the iteration history are ridged.
NOTE: The data set WORK.RANDOMIZED has 20 observations and 16 variables.
NOTE: The data set SASUSER.IBD2 has 20 observations and 16 variables.
NOTE: The final   ridged D-efficiency criterion is 32.1882.
NOTE: The final unridged D-efficiency criterion is 0.0000.
```

When %MktEx finishes with the first observation in the second pass through the design, it prints the warning that it may be impossible to meet all restrictions. This is because %MktEx did not succeed in finding zero violations in any row. We will see later that sometimes it is possible to meet all restrictions even when this message comes out, but in this case, when %MktEx finished, there were still restriction violations. The %MktEx macro skips printing the final efficiencies with PROC OPTEX because there are too many model parameters. For this application, that is not a problem for us, since we are not using this design as is, it is just an intermediate step. We can turn off these warnings with options=nofinal. Here is the design.

```
proc print data=sasuser.ibd2; run;
```

Obs	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16
1	1	1	1	2	1	1	1	2	1	1	2	1	1	2	1	1
2	1	1	1	1	1	2	2	2	1	1	1	1	2	1	1	1
3	2	1	1	1	2	1	1	1	1	1	1	1	2	1	2	1
4	1	2	1	1	2	1	2	1	1	1	1	2	1	1	1	1
5	2	1	1	1	1	1	2	1	1	1	2	1	1	1	1	2
6	1	2	1	1	1	1	1	2	1	1	1	1	1	1	2	2
7	2	2	1	2	1	1	1	1	1	2	1	1	1	1	1	1
8	1	2	1	1	2	1	1	1	2	1	1	1	1	2	1	1
9	1	1	1	1	1	1	1	1	2	1	2	2	1	1	2	1
10	1	1	2	1	1	2	1	1	1	2	2	1	1	1	1	1
11	1	1	1	1	1	1	1	1	1	2	1	1	2	2	1	2
12	1	1	1	2	1	2	1	1	1	1	1	2	1	1	1	2
13	1	2	2	1	1	1	1	1	1	1	2	1	2	1	1	1
14	2	1	2	1	1	1	1	2	2	1	1	1	1	1	1	1
15	1	1	2	2	2	1	1	1	1	1	1	1	1	1	1	2
16	1	1	1	1	2	2	1	1	1	2	1	1	1	1	2	1
17	1	1	1	2	1	1	2	1	2	1	1	1	2	1	1	1
18	1	1	1	1	1	1	1	2	2	2	1	2	1	1	1	1
19	1	1	2	1	1	1	2	1	1	1	1	1	1	2	2	1
20	2	1	1	1	1	2	1	1	1	1	1	2	1	2	1	1

We see four varying attributes in each row. These next steps create the orthogonal array, the actual IBD, the IBD frequency matrix `f` like before, and a choice design from the IBD and the orthogonal array. They also evaluate the design.

```
%mktex(2 4 2 2 2, n=8)

data design; set design; drop x2; run;

%mktppro(x=sasuser.ibd2)

%choiceff(data=chdes, model=class(x1-x16), nsets=80, nalts=2,
          beta=zero, init=chdes, initvars=x1-x16)
```

Recall that on page 435 we used for our orthogonal array, the hard-coded matrix:

```
1 1 1 1
1 1 2 2
1 2 1 2
1 2 2 1
```

and the second half of the orthogonal array:

```
2 2 2 2
2 2 1 1
2 1 2 1
2 1 1 2
```

was made from the first by shifting ($1 \rightarrow 2$ and $2 \rightarrow 1$). In this example, we directly made the orthogonal array with the `%MktEx` macro, and we did it in a particular way. We are interested in 4 two-level factors, and we request a two-level factor, followed by a four-level factor, followed by three two-level factors. More generally, we will request the p^m subset of the design $p^m m^1$ in $p \times m$ runs. Examples: 2^4 in 8 runs, selected from $2^4 4^1$ in 8 runs, 3^3 in 9 runs, selected from $3^3 3^1$ in 9 runs, 4^4 in 16 runs, selected from $4^4 4^1$ in 16 runs. The rows of the orthogonal-array design must be sorted into the right order. The easiest way to do this is to first request one of the m p-level factors, then request the m-level factor, then request the remaining $(m-1)$ p-level factors. Then *after* the design is created, discard `x2`, the m-level factor. The goal is to create an orthogonal array with p blocks of m rows. When a design is created this way, the design factor `x1` will contain the block number and `x2`, which is discarded, will contain the order in which the rows need to be sorted within block. When the design is sorted by `x1` and `x2`, everything is in the right order. These are the steps that did that.

```
%mktex(2 4 2 2 2, n=8)

data design; set design; drop x2; run;
```

All but the most interested readers may skip this paragraph. Each block is a difference scheme, and blocks 2 through p are obtained from the preceding block by adding 1 (in the appropriate Galois field). For example, when p is 2, add 1 modulo 2; and when p is 3, add 1 modulo 3. You will not get optimal results if you stick in any other kind of orthogonal array. Just as many orthogonal arrays are created by developing a difference scheme, our partial-profile design will be created by developing the difference scheme that `%MktEx` will output in the first m rows of the experimental design.

Now our orthogonal array is in the data set `Design`, and our binary representation of an IBD is in the data set `sasuser.IBD2`. We can use the `%MktPPro` macro to develop the orthogonal array into a partial-profile design using the rules in the IBD. This step takes by default the `design=design` data set and uses the binary matrix created by the the `%MktEx` macro, to create `out=` choice design, which by default is called `ChDes`. If we had an actual IBD like the one created on page 435, we would have specified the `ibd=` option instead.

```
%mktppro(x=sasuser.ibd2)
```

Here are the frequencies.

F

5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	5	1	1	2	0	1	1	1	1	1	1	1	1	1	1
0	0	5	1	1	1	1	1	1	1	2	0	1	1	1	1
0	0	0	5	1	1	1	1	1	1	1	1	1	1	0	2
0	0	0	0	5	1	1	0	1	1	0	1	1	1	2	1
0	0	0	0	0	5	1	1	0	2	1	2	1	1	1	1
0	0	0	0	0	0	5	1	1	0	1	1	2	1	1	1
0	0	0	0	0	0	0	5	2	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	5	1	1	2	1	1	1	0
0	0	0	0	0	0	0	0	0	5	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	5	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	5	0	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	5	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	5	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5

The frequencies look pretty good with all fives on the diagonal and mostly ones in the upper triangle. You need to ensure that the constant-diagonal restrictions were met. This next step evaluates the choice design.

```
%choiceff(data=chdes, model=class(x1-x16), nsets=80, nalts=2,
          beta=zero, init=chdes, initvars=x1-x16)
```

Here are the results.

n	Name	Beta	Label
1	x11	0	x1 1
2	x21	0	x2 1
3	x31	0	x3 1
4	x41	0	x4 1
5	x51	0	x5 1

```
 6    x61       0     x6 1
 7    x71       0     x7 1
 8    x81       0     x8 1
 9    x91       0     x9 1
10    x101      0     x10 1
11    x111      0     x11 1
12    x121      0     x12 1
13    x131      0     x13 1
14    x141      0     x14 1
15    x151      0     x15 1
16    x161      0     x16 1
```

Design	Iteration	D-Efficiency	D-Error
1	0	5.000000	0.200000
	1	5.000000	0.200000

Final Results

Design	1
Choice Sets	80
Alternatives	2
D-Efficiency	5.000000
D-Error	0.200000

n	Variable Name	Label	Variance	DF	Standard Error
1	x11	x1 1	0.2	1	0.44721
2	x21	x2 1	0.2	1	0.44721
3	x31	x3 1	0.2	1	0.44721
4	x41	x4 1	0.2	1	0.44721
5	x51	x5 1	0.2	1	0.44721
6	x61	x6 1	0.2	1	0.44721
7	x71	x7 1	0.2	1	0.44721
8	x81	x8 1	0.2	1	0.44721
9	x91	x9 1	0.2	1	0.44721
10	x101	x10 1	0.2	1	0.44721
11	x111	x11 1	0.2	1	0.44721
12	x121	x12 1	0.2	1	0.44721
13	x131	x13 1	0.2	1	0.44721
14	x141	x14 1	0.2	1	0.44721
15	x151	x15 1	0.2	1	0.44721
16	x161	x16 1	0.2	1	0.44721
				==	
				16	

This design is optimal! It is optimal even though we did not use a real BIBD in its construction. We did however, use a real orthogonal array, and the optimality depends a lot more on the orthogonal

array than it does on the BIBD.

80 Choice Sets, 16 Binary Attributes, Four Varying, Part 3. This next example tries basically the same thing we just tried, only this time we use `%MktEx` to find 12 (out of 20) rows from a BIBD to use, which will make $12 \times 4 = 48$ choice sets. Our restrictions macro now loops over 12 rows instead of 20, it looks for threes on the diagonal instead of fives, and it looks for either zeros or ones in the upper triangle. If parts of our computed matrix looks like `_f`, (ones above) we are okay, or if parts of it looks like `_d` (three time and identity matrix and hence zeros above) we are okay. So we create two matrices `abs(f - _f)` and `abs(f - _d)` and take the element-wise minimum using the "`><`" operator and element-wise multiply that times the penalty matrix. Since we now have fewer runs than parameters, we specify `ridge=0.01`. The `%MktEx` macro prints a number of warnings when there are more runs than parameters like we have here, but it does make the design if you specify a *ridging* value to add to the diagonal of the information matrix to make it nonsingular.

```
%macro con;
   _d = 3 # i(&m);
   _f = ((1:&m) @ j(&m, 1, 1) > j(1, &m, 1) @ (1:&m)`) + _d;
   _p = (_f = 3) + 10 # (_f = 1);
   %mend;

%macro res;
   bad = 1000 # abs(sum(x = 2) - 4);
   f = j(&m, &m, 0);
   do ii = 1 to &n;
      if ii = i then l = loc(x = 2);
      else l = loc(xmat[ii,] = 2);
      if ncol(l) then f[l, l] = f[l, l] + 1;
      end;
   bad = bad + sum((abs(f - _f) >< abs(f - _d)) # _p);
   %mend;

%mktex(2 ** 16, n=12, tabiter=0, optiter=0,
       restrictions=res, resmac=con, reslist=%str(_f, _p, _d),
       order=random, options=resrep nofinal, exchange=2, seed=151,
       maxdesigns=1, out=sasuser.ibd3, ridge=0.01)

%mktex(2 4 2 2 2, n=8)

data design; set design; drop x2; run;

%mktppro(x=sasuser.ibd3)

%choiceff(data=chdes, model=class(x1-x16), nsets=48, nalts=2,
          beta=zero, init=chdes, initvars=x1-x16)
```

See page 732 for an easier-to-modify version of this code. Here is some of the output.

Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
1	Start	24.2793		Ran,Mut,Ann
1	1	25.1008		2397 Violations
1	2	25.7906		2173 Violations
1	3	25.8098		1949 Violations
1	4	25.3443		1563 Violations
1	5	25.0900		1268 Violations
1	6	25.2672		1135 Violations
1	7	25.4814		911 Violations
1	8	24.2675		707 Violations
1	9	24.7043		615 Violations
1	10	25.0944		270 Violations
1	11	25.3510		249 Violations
1	12	25.3595		70 Violations
1	1	24.6445		58 Violations
WARNING: It may be impossible to meet all restrictions.				
1	2	25.1399		56 Violations
1	3	25.5988		44 Violations
1	4	25.5988		44 Violations
1	5	26.3537		34 Violations
1	6	27.1408		24 Violations
1	7	27.6591		16 Violations
1	8	27.6843		14 Violations
1	9	27.7351		2 Violations
1	10	27.7351		2 Violations
1	11	27.7351		2 Violations
1	12	27.7351		2 Violations
1	1	27.7351		2 Violations
1	2	27.7351		2 Violations
1	3	27.6546		0 Violations
1	4	27.6546		0 Violations
1	5	27.6546		0 Violations
1	6	27.6546		0 Violations
1	7	27.6546		0 Violations
1	8	27.6546		0 Violations
1	9	27.6546		0 Violations
1	10	27.6546		0 Violations
1	11	27.6546		0 Violations
1	12	27.6546		0 Violations
1	1	27.6546		0 Violations
1	2	27.6546		0 Violations
1	3	27.6546		0 Violations
1	4 1	27.6546	27.6546	Conforms
1	End	27.6546		

F

```
3 1 0 0 1 1 1 0 1 0 1 1 0 1 0 1
0 3 0 1 0 0 0 1 1 1 0 1 1 0 1 1
0 0 3 1 1 0 0 1 1 1 1 1 0 1 0 1
0 0 0 3 0 1 1 0 0 1 0 1 1 1 1 0
0 0 0 0 3 1 0 0 1 0 1 1 0 1 1 1
0 0 0 0 0 3 1 1 0 0 1 1 1 0 1 0
0 0 0 0 0 0 3 0 1 1 1 0 1 1 1 0
0 0 0 0 0 0 0 3 0 1 1 1 1 0 1 1
0 0 0 0 0 0 0 0 3 1 1 0 1 0 0 1
0 0 0 0 0 0 0 0 0 3 0 0 1 1 1 0
0 0 0 0 0 0 0 0 0 0 3 0 1 1 0 0
0 0 0 0 0 0 0 0 0 0 0 3 1 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 3 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3
```

.
.
.

Final Results

Design	1
Choice Sets	48
Alternatives	2
D-Efficiency	3.000000
D-Error	0.333333

n	Variable Name	Label	Variance	DF	Standard Error
1	x11	x1 1	0.33333	1	0.57735
2	x21	x2 1	0.33333	1	0.57735
3	x31	x3 1	0.33333	1	0.57735
4	x41	x4 1	0.33333	1	0.57735
5	x51	x5 1	0.33333	1	0.57735
6	x61	x6 1	0.33333	1	0.57735
7	x71	x7 1	0.33333	1	0.57735
8	x81	x8 1	0.33333	1	0.57735
9	x91	x9 1	0.33333	1	0.57735

```
10      x101      x10 1     0.33333     1     0.57735
11      x111      x11 1     0.33333     1     0.57735
12      x121      x12 1     0.33333     1     0.57735
13      x131      x13 1     0.33333     1     0.57735
14      x141      x14 1     0.33333     1     0.57735
15      x151      x15 1     0.33333     1     0.57735
16      x161      x16 1     0.33333     1     0.57735
                                        ==
                                        16
```

This design is optimal as well. The frequencies look good with 3's on the diagonal.

80 Choice Sets, 16 Binary Attributes, Four Varying, Part 4. We could also ask for 8 runs and 32 choice sets. This time we look for two on the diagonal and zeros or ones above the diagonal. Notice that our IBD conforms to all restrictions.

```
%macro con;
   _d = 2 # i(&m);
   _f = ((1:&m) @ j(&m, 1, 1) > j(1, &m, 1) @ (1:&m)‘) + _d;
   _p = (_f = 2) + 10 # (_f = 1);
   %mend;

%macro res;
   bad = 1000 # abs(sum(x = 2) - 4);
   f = j(&m, &m, 0);
   do ii = 1 to &n;
      if ii = i then l = loc(x = 2);
      else l = loc(xmat[ii,] = 2);
      if ncol(l) then f[l, l] = f[l, l] + 1;
      end;
   bad = bad + sum((abs(f - _f) >< abs(f - _d)) # _p);
   %mend;

%mktex(2 ** 16, n=8, tabiter=0, optiter=0,
       restrictions=res, resmac=con, reslist=%str(_f, _p, _d),
       order=random, options=resrep nofinal, exchange=2, seed=17,
       maxdesigns=1, out=sasuser.ibd4, ridge=0.01)

%mktex(2 4 2 2 2, n=8)

data design; set design; drop x2; run;

%mktppro(x=sasuser.ibd4)

%choiceff(data=chdes, model=class(x1-x16), nsets=32, nalts=2,
          beta=zero, init=chdes, initvars=x1-x16)
```

See page 732 for an easier-to-modify version of this code. Here is just the last part of the output. This design is optimal as well.

Final Results

Design	1
Choice Sets	32
Alternatives	2
D-Efficiency	2.000000
D-Error	0.500000

n	Variable Name	Label	Variance	DF	Standard Error
1	x11	x1 1	0.5	1	0.70711
2	x21	x2 1	0.5	1	0.70711
3	x31	x3 1	0.5	1	0.70711
4	x41	x4 1	0.5	1	0.70711
5	x51	x5 1	0.5	1	0.70711
6	x61	x6 1	0.5	1	0.70711
7	x71	x7 1	0.5	1	0.70711
8	x81	x8 1	0.5	1	0.70711
9	x91	x9 1	0.5	1	0.70711
10	x101	x10 1	0.5	1	0.70711
11	x111	x11 1	0.5	1	0.70711
12	x121	x12 1	0.5	1	0.70711
13	x131	x13 1	0.5	1	0.70711
14	x141	x14 1	0.5	1	0.70711
15	x151	x15 1	0.5	1	0.70711
16	x161	x16 1	0.5	1	0.70711
				==	
				16	

Making the (what should be by now) obvious changes (not shown) asking for 4 runs and 16 choice sets, you can even get an optimal design in 16 runs, although again, we would not recommend this. Here is the last part of the output.

Final Results

Design	1
Choice Sets	16
Alternatives	2
D-Efficiency	1.000000
D-Error	1.000000

n	Variable Name	Label	Variance	DF	Standard Error
1	x11	x1 1	1	1	1
2	x21	x2 1	1	1	1
3	x31	x3 1	1	1	1
4	x41	x4 1	1	1	1
5	x51	x5 1	1	1	1
6	x61	x6 1	1	1	1
7	x71	x7 1	1	1	1
8	x81	x8 1	1	1	1
9	x91	x9 1	1	1	1
10	x101	x10 1	1	1	1
11	x111	x11 1	1	1	1
12	x121	x12 1	1	1	1
13	x131	x13 1	1	1	1
14	x141	x14 1	1	1	1
15	x151	x15 1	1	1	1
16	x161	x16 1	1	1	1
				==	
				16	

---

48 Choice Sets, 24 Three-Level Attributes, Six Varying. This next example creates 48 partial-profile choice sets each containing three alternatives and 24 attributes, six of which vary. It uses a slight variation on the code we used previously. Our `con` macro is exactly the same as our last one. We want each attribute to appear twice. Our `res` macro differs from the last one only in the first line, because now we want 6 attributes to vary instead of four. The `%MktEx` call asks for 24 factors, since we have 24 attributes. It specifies `n=8` to request an eight row IBD.

```
%macro con;
   _d = 2 # i(&m);
   _f = ((1:&m) @ j(&m, 1, 1) > j(1, &m, 1) @ (1:&m)`) + _d;
   _p = (_f = 2) + 10 # (_f = 1);
   %mend;

%macro res;
   bad = 1000 # abs(sum(x = 2) - 6);
   f = j(&m, &m, 0);
   do ii = 1 to &n;
      if ii = i then l = loc(x = 2);
      else l = loc(xmat[ii,] = 2);
      if ncol(l) then f[l, l] = f[l, l] + 1;
      end;
   bad = bad + sum((abs(f - _f) >< abs(f - _d)) # _p);
   %mend;

%mktex(2 ** 24, n=8, tabiter=0, optiter=0,
       restrictions=res, resmac=con, reslist=%str(_f, _p, _d),
       order=random, options=resrep nofinal, exchange=2, seed=114,
       maxdesigns=1, out=sasuser.ibd5, ridge=0.01)
```

This next part is new. It requests 6 three-level factors (and an additional six-level factor) in 18 runs. The order of the factors is important! See page 452. See page 732 for an-easier-to modify version of this code.

```
%mktex(3 6 3 ** 5, n=18)

data design; set design; drop x2; run;

proc print; run;

%mktppro(x=sasuser.ibd5)

%choiceff(data=chdes, model=class(x1-x24), nsets=48, nalts=3,
          beta=zero, init=chdes, initvars=x1-x24)

proc print; by set; id set; var x:; where set le 6 or set gt 42; run;
```

Obs	x1	x3	x4	x5	x6	x7
1	1	1	1	1	1	1
2	1	2	3	2	3	1
3	1	3	2	2	1	3
4	1	3	3	1	2	2
5	1	1	2	3	3	2
6	1	2	1	3	2	3
7	2	2	2	2	2	2
8	2	3	1	3	1	2
9	2	1	3	3	2	1
10	2	1	1	2	3	3
11	2	2	3	1	1	3
12	2	3	2	1	3	1
13	3	3	3	3	3	3
14	3	1	2	1	2	3
15	3	2	1	1	3	2
16	3	2	2	3	1	1
17	3	3	1	2	2	1
18	3	1	3	2	1	2

Here is the last part of the output, including the first six and last six choice sets. This design is optimal.

Final Results

Design	1
Choice Sets	48
Alternatives	3
D-Efficiency	2.309401
D-Error	0.433013

n	Variable Name	Label	Variance	DF	Standard Error
1	x11	x1 1	0.5	1	0.70711
2	x12	x1 2	0.5	1	0.70711
3	x21	x2 1	0.5	1	0.70711
4	x22	x2 2	0.5	1	0.70711
5	x31	x3 1	0.5	1	0.70711
6	x32	x3 2	0.5	1	0.70711
7	x41	x4 1	0.5	1	0.70711
8	x42	x4 2	0.5	1	0.70711
9	x51	x5 1	0.5	1	0.70711
10	x52	x5 2	0.5	1	0.70711
11	x61	x6 1	0.5	1	0.70711
12	x62	x6 2	0.5	1	0.70711
13	x71	x7 1	0.5	1	0.70711
14	x72	x7 2	0.5	1	0.70711
15	x81	x8 1	0.5	1	0.70711
16	x82	x8 2	0.5	1	0.70711
17	x91	x9 1	0.5	1	0.70711
18	x92	x9 2	0.5	1	0.70711
19	x101	x10 1	0.5	1	0.70711
20	x102	x10 2	0.5	1	0.70711
21	x111	x11 1	0.5	1	0.70711
22	x112	x11 2	0.5	1	0.70711
23	x121	x12 1	0.5	1	0.70711
24	x122	x12 2	0.5	1	0.70711
25	x131	x13 1	0.5	1	0.70711
26	x132	x13 2	0.5	1	0.70711
27	x141	x14 1	0.5	1	0.70711
28	x142	x14 2	0.5	1	0.70711
29	x151	x15 1	0.5	1	0.70711
30	x152	x15 2	0.5	1	0.70711
31	x161	x16 1	0.5	1	0.70711
32	x162	x16 2	0.5	1	0.70711
33	x171	x17 1	0.5	1	0.70711
34	x172	x17 2	0.5	1	0.70711
35	x181	x18 1	0.5	1	0.70711
36	x182	x18 2	0.5	1	0.70711
37	x191	x19 1	0.5	1	0.70711
38	x192	x19 2	0.5	1	0.70711
39	x201	x20 1	0.5	1	0.70711
40	x202	x20 2	0.5	1	0.70711

```
          41      x211      x21 1       0.5       1     0.70711
          42      x212      x21 2       0.5       1     0.70711
          43      x221      x22 1       0.5       1     0.70711
          44      x222      x22 2       0.5       1     0.70711
          45      x231      x23 1       0.5       1     0.70711
          46      x232      x23 2       0.5       1     0.70711
          47      x241      x24 1       0.5       1     0.70711
          48      x242      x24 2       0.5       1     0.70711
                                                ==
                                                48
```

Set	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	x18	x19	x20	x21	x22	x23	x24
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	1	1	1	1	1	1	2	1	1	1	2	1	1	1	2	1	1	1	1	1	2	2	1	2
	1	1	1	1	1	1	3	1	1	1	3	1	1	1	3	1	1	1	1	1	3	3	1	3
2	1	1	1	1	1	1	1	1	1	1	2	1	1	1	3	1	1	1	1	1	2	3	1	1
	1	1	1	1	1	1	2	1	1	1	3	1	1	1	1	1	1	1	1	1	3	1	1	2
	1	1	1	1	1	1	3	1	1	1	1	1	1	1	2	1	1	1	1	1	1	2	1	3
3	1	1	1	1	1	1	1	1	1	1	3	1	1	1	2	1	1	1	1	1	2	1	1	3
	1	1	1	1	1	1	2	1	1	1	1	1	1	1	3	1	1	1	1	1	3	2	1	1
	1	1	1	1	1	1	3	1	1	1	2	1	1	1	1	1	1	1	1	1	1	3	1	2
4	1	1	1	1	1	1	1	1	1	1	3	1	1	1	3	1	1	1	1	1	1	2	1	2
	1	1	1	1	1	1	2	1	1	1	1	1	1	1	1	1	1	1	1	1	2	3	1	3
	1	1	1	1	1	1	3	1	1	1	2	1	1	1	2	1	1	1	1	1	3	1	1	1
5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	1	1	1	1	1	3	3	1	2
	1	1	1	1	1	1	2	1	1	1	2	1	1	1	3	1	1	1	1	1	1	1	1	3
	1	1	1	1	1	1	3	1	1	1	3	1	1	1	1	1	1	1	1	1	2	2	1	1
6	1	1	1	1	1	1	1	1	1	1	2	1	1	1	1	1	1	1	1	1	3	2	1	3
	1	1	1	1	1	1	2	1	1	1	3	1	1	1	2	1	1	1	1	1	1	3	1	1
	1	1	1	1	1	1	3	1	1	1	1	1	1	1	3	1	1	1	1	1	2	1	1	2
43	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2	1	2	2	1	1	1	1	1	1	1	1	2	1	2	1	1	1	1	1	1	1	2	1
	3	1	3	3	1	1	1	1	1	1	1	1	3	1	3	1	1	1	1	1	1	1	3	1
44	1	1	2	3	1	1	1	1	1	1	1	1	2	1	3	1	1	1	1	1	1	1	1	1
	2	1	3	1	1	1	1	1	1	1	1	1	3	1	1	1	1	1	1	1	1	1	2	1
	3	1	1	2	1	1	1	1	1	1	1	1	1	1	2	1	1	1	1	1	1	1	3	1
45	1	1	3	2	1	1	1	1	1	1	1	1	2	1	1	1	1	1	1	1	1	1	3	1
	2	1	1	3	1	1	1	1	1	1	1	1	3	1	2	1	1	1	1	1	1	1	1	1
	3	1	2	1	1	1	1	1	1	1	1	1	1	1	3	1	1	1	1	1	1	1	2	1

```
46  1  1  3  3  1  1  1  1  1  1  1  1  1  1  2  1  1  1  1  1  1  1  2  1
    2  1  1  1  1  1  1  1  1  1  1  1  1  2  1  3  1  1  1  1  1  1  1  3  1
    3  1  2  2  1  1  1  1  1  1  1  1  1  3  1  1  1  1  1  1  1  1  1  1  1

47  1  1  1  2  1  1  1  1  1  1  1  1  3  1  3  1  1  1  1  1  1  1  2  1
    2  1  2  3  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  3  1
    3  1  3  1  1  1  1  1  1  1  1  1  2  1  2  1  1  1  1  1  1  1  1  1

48  1  1  2  1  1  1  1  1  1  1  1  1  3  1  2  1  1  1  1  1  1  1  3  1
    2  1  3  2  1  1  1  1  1  1  1  1  1  1  3  1  1  1  1  1  1  1  1  1
    3  1  1  3  1  1  1  1  1  1  1  1  2  1  1  1  1  1  1  1  1  1  2  1
```

---

32 Choice Sets, 16 Four-level Attributes, Four Varying. This example creates 32 choice sets, with 16 four-level attributes, four of which vary in each choice set. See page 732 for an easier-to-modify version of this code.

```
%macro con;
   _d = 2 # i(&m);
   _f = ((1:&m) @ j(&m, 1, 1) > j(1, &m, 1) @ (1:&m)`) + _d;
   _p = (_f = 2) + 10 # (_f = 1);
   %mend;

%macro res;
   bad = 1000 # abs(sum(x = 2) - 4);
   f = j(&m, &m, 0);
   do ii = 1 to &n;
      if ii = i then l = loc(x = 2);
      else l = loc(xmat[ii,] = 2);
      if ncol(l) then f[l, l] = f[l, l] + 1;
      end;
   bad = bad + sum((abs(f - _f) >< abs(f - _d)) # _p);
   %mend;

%mktex(2 ** 16, n=8, tabiter=0, optiter=0,
       restrictions=res, resmac=con, reslist=%str(_f, _p, _d),
       order=random, options=resrep nofinal, exchange=2, seed=114,
       maxdesigns=1, out=sasuser.ibd6, ridge=0.01)

%mktex(4 ** 5, n=16)

data design; set design; drop x2; run;

%mktppro(x=sasuser.ibd6)

%choiceff(data=chdes, model=class(x1-x16), nsets=32, nalts=4,
          beta=zero, init=chdes, initvars=x1-x16)

proc print; by set; id set; var x:; where set le 6 or set gt 42; run;
```

64 Choice Sets, 32 Four-level Attributes, Four Varying. This example creates 64 choice sets, with 32 four-level attributes, four of which vary in each choice set. See page 732 for an easier-to-modify version of this code.

```
%macro con;
   _d = 2 # i(&m);
   _f = ((1:&m) @ j(&m, 1, 1) > j(1, &m, 1) @ (1:&m)`) + _d;
   _p = (_f = 2) + 10 # (_f = 1);
   %mend;

%macro res;
   bad = 1000 # abs(sum(x = 2) - 4);
   f = j(&m, &m, 0);
   do ii = 1 to &n;
      if ii = i then l = loc(x = 2);
      else l = loc(xmat[ii,] = 2);
      if ncol(l) then f[l, l] = f[l, l] + 1;
      end;
   bad = bad + sum((abs(f - _f) >< abs(f - _d)) # _p);
   %mend;

%mktex(2 ** 32, n=16, tabiter=0, optiter=0,
       restrictions=res, resmac=con, reslist=%str(_f, _p, _d),
       order=random, options=resrep nofinal, exchange=2, seed=420,
       maxdesigns=1, out=sasuser.ibd7, ridge=0.01)

%mktex(4 ** 5, n=16)

data design; set design; drop x2; run;

%mktppro(x=sasuser.ibd7)

%choiceff(data=chdes, model=class(x1-x32), nsets=64, nalts=4,
          beta=zero, init=chdes, initvars=x1-x32)
```

Multinomial Logit Models

Ying So

Warren F. Kuhfeld

Abstract

Multinomial logit models are used to model relationships between a polytomous response variable and a set of regressor variables. The term "multinomial logit model" includes, in a broad sense, a variety of models. The cumulative logit model is used when the response of an individual unit is restricted to one of a finite number of ordinal values. Generalized logit and conditional logit models are used to model consumer choices. This article focuses on the statistical techniques for analyzing discrete choice data and discusses fitting these models using SAS/STAT software.*

Introduction

Multinomial logit models are used to model relationships between a polytomous response variable and a set of regressor variables. These polytomous response models can be classified into two distinct types, depending on whether the response variable has an ordered or unordered structure.

In an ordered model, the response Y of an individual unit is restricted to one of m ordered values. For example, the severity of a medical condition may be: none, mild, and severe. The cumulative logit model assumes that the ordinal nature of the observed response is due to methodological limitations in collecting the data that results in lumping together values of an otherwise continuous response variable (McKelvey and Zavoina 1975). Suppose Y takes values $y_1, y_2, \ldots, y_m$ on some scale, where $y_1 < y_2 < \ldots < y_m$. It is assumed that the observable variable is a categorized version of a continuous latent variable U such that

$$Y = y_i \Leftrightarrow \alpha_{i-1} < U \leq \alpha_i, i = 1, \ldots, m$$

where $-\infty = \alpha_0 < \alpha_1 < \ldots < \alpha_m = \infty$. It is further assumed that the latent variable U is determined by the explanatory variable vector $\mathbf{x}$ in the linear form $U = -\boldsymbol{\beta}'\mathbf{x} + \epsilon$, where $\boldsymbol{\beta}$ is a vector of regression coefficients and ϵ is a random variable with a distribution function F. It follows that

$$\Pr\{Y \leq y_i | \mathbf{x}\} = F(\alpha_i + \boldsymbol{\beta}'\mathbf{x})$$

If F is the logistic distribution function, the cumulative model is also known as the proportional odds model. You can use PROC LOGISTIC or PROC PROBIT directly to fit the cumulative logit models. Although the cumulative model is the most widely used model for ordinal response data, other useful models include the adjacent-categories logit model and the continuation-ratio model (Agresti 1990).

*This chapter was presented at SUGI 20 by Ying So and can also be found in the SUGI 20 proceedings. Copies of this article (TS-722G) and all of the macros are available on the web http://support.sas.com/techsup/tnote/tnote_stat.html#market.

In an unordered model, the polytomous response variable does not have an ordered structure. Two classes of models, the generalized logit models and the conditional logit models, can be used with nominal response data. The generalized logit model consists of a combination of several binary logits estimated simultaneously. For example, the response variable of interest is the occurrence or nonoccurrence of infection after a Caesarean section with two types of (I,II) infection. Two binary logits are considered: one for type I infection versus no infection and the other for type II infection versus no infection. The conditional logit model has been used in biomedical research to estimate relative risks in matched case-control studies. The nuisance parameters that correspond to the matched sets in an unconditional analysis are eliminated by using a conditional likelihood that contains only the relative risk parameters (Breslow and Day 1980). The conditional logit model was also introduced by McFadden (1974) in the context of econometrics.

In studying consumer behavior, an individual is presented with a set of alternatives and asked to choose the most preferred alternative. Both the generalized logit and conditional logit models are used in the analysis of discrete choice data. In a conditional logit model, a choice among alternatives is treated as a function of the characteristics of the alternatives, whereas in a generalized logit model, the choice is a function of the characteristics of the individual making the choice. In many situations, a mixed model that includes both the characteristics of the alternatives and the individual is needed for investigating consumer choice.

Consider an example of travel demand. People are asked to choose between travel by auto, plane or public transit (bus or train). The following SAS statements create the data set TRAVEL. The variables `AutoTime`, `PlanTime`, and `TranTime` represent the total travel time required to get to a destination by using auto, plane, or transit, respectively. The variable `Age` represents the age of the individual being surveyed, and the variable `Chosen` contains the individual's choice of travel mode.

```
data travel;
   input AutoTime PlanTime TranTime Age Chosen $;
   datalines;
10.0      4.5     10.5    32  Plane
 5.5      4.0      7.5    13  Auto
 4.5      6.0      5.5    41  Transit
 3.5      2.0      5.0    41  Transit
 1.5      4.5      4.0    47  Auto
10.5      3.0     10.5    24  Plane
 7.0      3.0      9.0    27  Auto
 9.0      3.5      9.0    21  Plane
 4.0      5.0      5.5    23  Auto
22.0      4.5     22.5    30  Plane
 7.5      5.5     10.0    58  Plane
11.5      3.5     11.5    36  Transit
 3.5      4.5      4.5    43  Auto
12.0      3.0     11.0    33  Plane
18.0      5.5     20.0    30  Plane
23.0      5.5     21.5    28  Plane
 4.0      3.0      4.5    44  Plane
 5.0      2.5      7.0    37  Transit
 3.5      2.0      7.0    45  Auto
12.5      3.5     15.5    35  Plane
 1.5      4.0      2.0    22  Auto
;
```

In this example, `AutoTime`, `PlanTime`, and `TranTime` are alternative-specific variables, whereas `Age` is a characteristic of the individual. You use a generalized logit model to investigate the relationship between the choice of transportation and `Age`, and you use a conditional logit model to investigate how travel time affects the choice. To study how the choice depends on both the travel time and age of the individual, you need to use a mixed model that incorporates both types of variables.

A survey of the literature reveals a confusion in the terminology for the nominal response models. The term "multinomial logit model" is often used to describe the generalized logit model. The mixed logit is sometimes referred to as the multinomial logit model in which the generalized logit and the conditional logit models are special cases.

The following sections describe discrete choice models, illustrate how to use SAS/STAT software to fit these models, and discuss cross-alternative effects.

Modeling Discrete Choice Data

Consider an individual choosing among m alternatives in a choice set. Let Π_{jk} denote the probability that individual j chooses alternative k, let $\mathbf{X}_j$ represent the characteristics of individual j, and let $\mathbf{Z}_{jk}$ be the characteristics of the kth alternative for individual j. For example, $\mathbf{X}_j$ may be an age and each $\mathbf{Z}_{jk}$ a travel time.

The generalized logit model focuses on the individual as the unit of analysis and uses individual characteristics as explanatory variables. The explanatory variables, being characteristics of an individual, are constant over the alternatives. For example, for each of the m travel modes, $\mathbf{X}_j = (1\ age)'$, and for the first subject, $\mathbf{X}_1 = (1\ 32)'$. The probability that individual j chooses alternative k is

$$\Pi_{jk} = \frac{\exp(\boldsymbol{\beta}_k'\mathbf{X}_j)}{\sum_{l=1}^{m}\exp(\boldsymbol{\beta}_l'\mathbf{X}_j)} = \frac{1}{\sum_{l=1}^{m}\exp[(\boldsymbol{\beta}_l - \boldsymbol{\beta}_k)'\mathbf{X}_j)]}$$

$\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_m$ are m vectors of unknown regression parameters (each of which is different, even though $\mathbf{X}_j$ is constant across alternatives). Since $\sum_{k=1}^{m}\Pi_{jk} = 1$, the m sets of parameters are not unique. By setting the last set of coefficients to null (that is, $\boldsymbol{\beta}_m = \mathbf{0}$), the coefficients $\boldsymbol{\beta}_k$ represent the effects of the $\mathbf{X}$ variables on the probability of choosing the kth alternative over the last alternative. In fitting such a model, you estimate $m - 1$ sets of regression coefficients.

In the conditional logit model, the explanatory variables $\mathbf{Z}$ assume different values for each alternative and the impact of a unit of $\mathbf{Z}$ is assumed to be constant across alternatives. For example, for each of the m travel modes, $\mathbf{Z}_{jk} = (time)'$, and for the first subject, $\mathbf{Z}_{11} = (10)'$, $\mathbf{Z}_{12} = (4.5)'$, and $\mathbf{Z}_{13} = (10.5)'$. The probability that the individual j chooses alternative k is

$$\Pi_{jk} = \frac{\exp(\boldsymbol{\theta}'\mathbf{Z}_{jk})}{\sum_{l=1}^{m}\exp(\boldsymbol{\theta}'\mathbf{Z}_{jl})} = \frac{1}{\sum_{l=1}^{m}\exp[\boldsymbol{\theta}'(\mathbf{Z}_{jl} - \mathbf{Z}_{jk})]}$$

$\boldsymbol{\theta}$ is a single vector of regression coefficients. The impact of a variable on the choice probabilities derives from the difference of its values across the alternatives.

For the mixed logit model that includes both characteristics of the individual and the alternatives, the choice probabilities are

$$\Pi_{jk} = \frac{\exp(\boldsymbol{\beta}_k'\mathbf{X}_j + \boldsymbol{\theta}'\mathbf{Z}_{jk})}{\sum_{l=1}^{m}\exp(\boldsymbol{\beta}_l'\mathbf{X}_j + \boldsymbol{\theta}'\mathbf{Z}_{jl})}$$

$\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_{m-1}$ and $\boldsymbol{\beta}_m \equiv \mathbf{0}$ are the alternative-specific coefficients, and $\boldsymbol{\theta}$ is the set of global coefficients.

Fitting Discrete Choice Models

The CATMOD procedure in SAS/STAT software directly fits the generalized logit model. SAS/STAT software does not yet have a procedure that is specially designed to fit the conditional or mixed logit models. However, with some preliminary data processing, you can use the PHREG procedure to fit these models.

The PHREG procedure fits the Cox proportional hazards model to survival data (refer to SAS/STAT documentation). The partial likelihood of Breslow has the same form as the likelihood in a conditional logit model.

Let z_l denote the vector of explanatory variables for individual l. Let $t_1 < t_2 < \ldots < t_k$ denote k distinct ordered event times. Let d_i denote the number of failures at t_i. Let s_i be the sum of the vectors z_l for those individuals that fail at t_i, and let $\mathcal{R}_i$ denote the set of indices for those who are at risk just before t_i.

The Breslow (partial) likelihood is

$$L_B(\boldsymbol{\theta}) = \prod_{i=1}^{k} \frac{\exp(\boldsymbol{\theta}' s_i)}{[\sum_{l \in \mathcal{R}_i} \exp(\boldsymbol{\theta}' z_l)]^{d_i}}$$

In a stratified analysis, the partial likelihood is the product of the partial likelihood for each individual stratum. For example, in a study of the time to first infection from a surgery, the variables of a patient consist of `Time` (time from surgery to the first infection), `Status` (an indicator of whether the observation time is censored, with value 2 identifying a censored time), `Z1` and `Z2` (explanatory variables thought to be related to the time to infection), and `Grp` (a variable identifying the stratum to which the observation belongs). The specification in PROC PHREG for fitting the Cox model using the Breslow likelihood is as follows:

```
proc phreg;
   model time*status(2) = z1 z2 / ties=breslow;
   strata grp;
   run;
```

To cast the likelihood of the conditional logit model in the form of the Breslow likelihood, consider m artificial observed times for each individual who chooses one of m alternatives. The kth alternative is chosen at time 1; the choices of all other alternatives (second choice, third choice, ...) are not observed and would have been chosen at some later time. So a choice variable is coded with an observed time value of 1 for the chosen alternative and a larger value, 2, for all unchosen (unobserved or censored alternatives). For each individual, there is exactly one event time (1) and $m - 1$ nonevent times, and the risk set just prior to this event time consists of all the m alternatives. For individual j with alternative-specific characteristics $\mathbf{Z}_{jl}$, the Breslow likelihood is then

$$L_B(\boldsymbol{\theta}) = \frac{\exp(\boldsymbol{\theta}' \mathbf{Z}_{jk})}{\sum_{l=1}^{m} \exp(\boldsymbol{\theta}' \mathbf{Z}_{jl})}$$

This is precisely the probability that individual j chooses alternative k in a conditional logit model. By stratifying on individuals, you get the likelihood of the conditional logit model. Note that the observed time values of 1 and 2 are chosen for convenience; however, the censored times have to be larger than the event time to form the correct risk set.

Before you invoke PROC PHREG to fit the conditional logit, you must arrange your data in such a way that there is a survival time for each individual-alternative. In the example of travel demand, let `Subject` identify the individuals, let `TravTime` represent the travel time for each mode of transportation, and let `Choice` have a value 1 if the alternative is chosen and 2 otherwise. The `Choice` variable is used as the artificial time variable as well as a censoring variable in PROC PHREG. The following SAS statements reshape the data set TRAVEL into data set CHOICE and display the first nine observations:

```
data choice(keep=subject mode travtime choice);
   array times[3] autotime plantime trantime;
   array allmodes[3] $ _temporary_ ('Auto' 'Plane' 'Transit');
   set travel;
   Subject = _n_;
   do i = 1 to 3;
      Mode = allmodes[i];
      TravTime = times[i];
      Choice = 2 - (chosen eq mode);
      output;
   end;
   run;

proc print data=choice(obs=9);
   run;
```

Obs	Subject	Mode	Trav Time	Choice
1	1	Auto	10.0	2
2	1	Plane	4.5	1
3	1	Transit	10.5	2
4	2	Auto	5.5	1
5	2	Plane	4.0	2
6	2	Transit	7.5	2
7	3	Auto	4.5	2
8	3	Plane	6.0	2
9	3	Transit	5.5	1

Notice that each observation in TRAVEL corresponds to a block of three observations in CHOICE, exactly one of which is chosen.

The following SAS statements invoke PROC PHREG to fit the conditional logit model. The Breslow likelihood is requested by specifying `ties=breslow`. `Choice` is the artificial time variable, and a value of 2 identifies censored times. `Subject` is used as a stratification variable.

```
proc phreg data=choice;
   model choice*choice(2) = travtime / ties=breslow;
   strata subject;
   title 'Conditional Logit Model Using PHREG';
   run;
```

Conditional Logit Model Using PHREG

The PHREG Procedure

Analysis of Maximum Likelihood Estimates

Variable	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq	Hazard Ratio
TravTime	1	-0.26549	0.10215	6.7551	0.0093	0.767

To study the relationship between the choice of transportation and the age of people making the choice, the analysis is based on the generalized logit model. You can use PROC CATMOD directly to fit the generalized logit model (refer to *SAS/STAT User's Guide, Vol. 1*). In the following invocation of PROC CATMOD, `Chosen` is the response variable and `Age` is the explanatory variable:

```
proc catmod data=travel;
   direct age;
   model chosen=age;
   title 'Multinomial Logit Model Using Catmod';
   run;
```

Response Profiles

Response	Chosen
1	Auto
2	Plane
3	Transit

Analysis of Maximum Likelihood Estimates

Parameter	Function Number	Estimate	Standard Error	Chi-Square	Pr > ChiSq
Intercept	1	3.0449	2.4268	1.57	0.2096
	2	2.7212	2.2929	1.41	0.2353
Age	1	-0.0710	0.0652	1.19	0.2762
	2	-0.0500	0.0596	0.70	0.4013

Note that there are two intercept coefficients and two slope coefficients for `Age`. The first `Intercept` and the first `Age` coefficients correspond to the effect on the probability of choosing auto over transit, and the second intercept and second age coefficients correspond to the effect of choosing plane over transit.

Let $\mathbf{X}_j$ be a $(p+1)$-vector representing the characteristics of individual j. The generalized logit model can be cast in the framework of a conditional model by defining the global parameter vector $\boldsymbol{\theta}$ and the alternative-specific regressor variables $\mathbf{Z}_{jk}$ as follows:

$$\boldsymbol{\theta} = \begin{bmatrix} \boldsymbol{\beta}_1 \\ \boldsymbol{\beta}_2 \\ \vdots \\ \boldsymbol{\beta}_{m-1} \end{bmatrix} \quad \mathbf{Z}_{j1} = \begin{bmatrix} \mathbf{X}_j \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix} \quad \mathbf{Z}_{j2} = \begin{bmatrix} \mathbf{0} \\ \mathbf{X}_j \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix} \quad \dots \quad \mathbf{Z}_{j,m-1} = \begin{bmatrix} \mathbf{0} \\ \vdots \\ \mathbf{0} \\ \mathbf{X}_j \end{bmatrix} \quad \mathbf{Z}_{jm} = \begin{bmatrix} \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix}$$

where the $\mathbf{0}$ is a $(p+1)$-vector of zeros. The probability that individual j chooses alternative k for the generalized logit model is put in the form that corresponds to a conditional logit model as follows:

$$\begin{aligned} \Pi_{jk} &= \frac{\exp(\boldsymbol{\beta}_k' \mathbf{X}_j)}{\sum_{l=1}^{m} \exp(\boldsymbol{\beta}_l' \mathbf{X}_j)} \\ &= \frac{\exp(\boldsymbol{\theta}' \mathbf{Z}_{jk})}{\sum_{l=1}^{m} \exp(\boldsymbol{\theta}' \mathbf{Z}_{jl})} \end{aligned}$$

Here, the vector $\mathbf{X}_j$ representing the characteristics of individual j includes the element 1 for the intercept parameter (provided that the intercept parameters are to be included in the model).

By casting the generalized logit model into a conditional logit model, you can then use PROC PHREG to analyze the generalized logit model. In the example of travel demand, the alternative-specific variables `Auto`, `Plane`, `AgeAuto`, and `AgePlane` are created from the individual characteristic variable `Age`. The following SAS statements reshape the data set TRAVEL into data set CHOICE2 and display the first nine observations:

```
data choice2;
   array times[3] autotime plantime trantime;
   array allmodes[3] $ _temporary_ ('Auto' 'Plane' 'Transit');
   set travel;
   Subject = _n_;
   do i = 1 to 3;
      Mode = allmodes[i];
      TravTime = times[i];
      Choice = 2 - (chosen eq mode);
      Auto = (i eq 1);
      Plane = (i eq 2);
      AgeAuto = auto * age;
      AgePlane = plane * age;
      output;
   end;
   keep subject mode travtime choice auto plane ageauto ageplane;
   run;
proc print data=choice2(obs=9);
   run;
```

Obs	Subject	Mode	Trav Time	Choice	Auto	Plane	Age Auto	Age Plane
1	1	Auto	10.0	2	1	0	32	0
2	1	Plane	4.5	1	0	1	0	32
3	1	Transit	10.5	2	0	0	0	0
4	2	Auto	5.5	1	1	0	13	0
5	2	Plane	4.0	2	0	1	0	13
6	2	Transit	7.5	2	0	0	0	0
7	3	Auto	4.5	2	1	0	41	0
8	3	Plane	6.0	2	0	1	0	41
9	3	Transit	5.5	1	0	0	0	0

The following SAS statements invoke PROC PHREG to fit the generalized logit model:

```
proc phreg data=choice2;
   model choice*choice(2) = auto plane ageauto ageplane /
         ties=breslow;
   strata subject;
   title 'Generalized Logit Model Using PHREG';
   run;
```

Generalized Logit Model Using PHREG

The PHREG Procedure

Analysis of Maximum Likelihood Estimates

Variable	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq	Hazard Ratio
Auto	1	3.04494	2.42682	1.5743	0.2096	21.009
Plane	1	2.72121	2.29289	1.4085	0.2353	15.199
AgeAuto	1	-0.07097	0.06517	1.1859	0.2762	0.931
AgePlane	1	-0.05000	0.05958	0.7045	0.4013	0.951

By transforming individual characteristics into alternative-specific variables, the mixed logit model can be analyzed as a conditional logit model.

Analyzing the travel demand data for the effects of both travel time and age of individual requires the same data set as the generalized logit model, only now the `TravTime` variable will be used as well. The following SAS statements use PROC PHREG to fit the mixed logit model:

```
proc phreg data=choice2;
   model choice*choice(2) = auto plane ageauto ageplane travtime /
         ties=breslow;
   strata subject;
   title 'Mixed Logit Model Using PHREG';
   run;
```

Mixed Logit Model Using PHREG

The PHREG Procedure

Analysis of Maximum Likelihood Estimates

Variable	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq	Hazard Ratio
Auto	1	2.50069	2.39585	1.0894	0.2966	12.191
Plane	1	-2.77912	3.52929	0.6201	0.4310	0.062
AgeAuto	1	-0.07826	0.06332	1.5274	0.2165	0.925
AgePlane	1	0.01695	0.07439	0.0519	0.8198	1.017
TravTime	1	-0.60845	0.27126	5.0315	0.0249	0.544

A special case of the mixed logit model is the conditional logit model with alternative-specific constants. Each alternative in the model can be represented by its own intercept, which captures the unmeasured desirability of the alternative.

```
proc phreg data=choice2;
   model choice*choice(2) = auto plane travtime / ties=breslow;
   strata subject;
   title 'Conditional Logit Model with Alternative Specific Constants';
   run;
```

Conditional Logit Model with Alternative Specific Constants

The PHREG Procedure

Analysis of Maximum Likelihood Estimates

Variable	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq	Hazard Ratio
Auto	1	-0.11966	0.70820	0.0285	0.8658	0.887
Plane	1	-1.63145	1.24251	1.7241	0.1892	0.196
TravTime	1	-0.48665	0.20725	5.5139	0.0189	0.615

With transit as the reference mode, the intercept for auto, which is negative, may reflect the inconvenience of having to drive over traveling by bus/train, and the intercept for plane may reflect the high expense of traveling by plane over bus/train.

Cross-Alternative Effects

Discrete choice models are often derived from the principle of maximum random utility. It is assumed that an unobserved utility V_k is associated with the kth alternative, and the response function Y is

determined by

$$Y = k \Leftrightarrow V_k = \max\{V_l, 1 \leq l \leq m\}$$

Both the generalized logit and the conditional logit models are based on the assumption that $V_1, \ldots, V_m$ are independently distributed and each follows an extreme maxima value distribution (Hoffman and Duncan, 1988). An important property of such models is Independence from Irrelevant Alternatives (IIA); that is, the ratio of the choice probabilities for any two alternatives for a particular observation is not influenced systematically by any other alternatives. IIA can be tested by fitting a model that contains all the cross-alternative effects and examining the significance of these effects. The cross-alternative effects pick up a variety of IIA violations and other sources of error in the model. (See pages 269, 275, 283, and 480 for other discussions of IIA.)

In the example of travel demand, there may be separate effects for the three travel modes and travel times. In addition, there may be cross-alternative effects for travel times. Not all the effects are estimable, only two of the three intercepts and three of the six cross-alternative effects can be estimated. The following SAS statements create the design variables for all the cross-alternative effects and display the first nine observations:

```
* Number of alternatives in each choice set;
%let m = 3;
data choice3;
   drop i j k autotime plantime trantime;

   * Values of the variable CHOSEN;
   array allmodes[&m] $
      _temporary_ ('Auto' 'Plane' 'Transit');
   * Travel times for the alternatives;
   array times[&m] autotime plantime trantime;
   * New variables that will contain the design:;
   array inters[&m]
      Auto      /*intercept for auto              */
      Plane     /*intercept for plane             */
      Transit;  /*intercept for transit           */
   array cross[%eval(&m * &m)]
      TimeAuto  /*time of auto alternative        */
      PlanAuto  /*cross effect of plane on auto   */
      TranAuto  /*cross effect of transit on auto */
      AutoPlan  /*cross effect of auto on plane   */
      TimePlan  /*time of plane alternative       */
      TranPlan  /*cross effect of transit on plane*/
      AutoTran  /*cross effect of auto on transit */
      PlanTran  /*cross effect of plane on transit*/
      TimeTran; /*time of transit alternative     */
   set travel;
   subject = _n_;
```

```
   * Create &m observations for each choice set;
   do i = 1 to &m;
      Mode = allmodes[i];  /* this alternative    */
      Travtime = times[i]; /* travel time         */
      Choice = 2 - (chosen eq mode);/* 1 - chosen */
      do j = 1 to &m;
         inters[j] = (i eq j);  /* mode indicator */
         do k = 1 to &m;
            * (j=k) - time, otherwise, cross effect;
            cross[&m*(j-1)+k]=times[k]*inters[j];
            end;
         end;
      output;
      end;
   run;
proc print data=choice3(obs=9) label noobs;
   var subject mode travtime choice auto plane transit
       timeauto timeplan timetran autoplan autotran planauto
       plantran tranauto tranplan;
   run;
```

subject	Mode	Travtime	Choice	Auto	Plane	Transit
1	Auto	10.0	2	1	0	0
1	Plane	4.5	1	0	1	0
1	Transit	10.5	2	0	0	1
2	Auto	5.5	1	1	0	0
2	Plane	4.0	2	0	1	0
2	Transit	7.5	2	0	0	1
3	Auto	4.5	2	1	0	0
3	Plane	6.0	2	0	1	0
3	Transit	5.5	1	0	0	1

Time Auto	Time Plan	Time Tran	Auto Plan	Auto Tran	Plan Auto	Plan Tran	Tran Auto	Tran Plan
10.0	0.0	0.0	0.0	0.0	4.5	0.0	10.5	0.0
0.0	4.5	0.0	10.0	0.0	0.0	0.0	0.0	10.5
0.0	0.0	10.5	0.0	10.0	0.0	4.5	0.0	0.0
5.5	0.0	0.0	0.0	0.0	4.0	0.0	7.5	0.0
0.0	4.0	0.0	5.5	0.0	0.0	0.0	0.0	7.5
0.0	0.0	7.5	0.0	5.5	0.0	4.0	0.0	0.0
4.5	0.0	0.0	0.0	0.0	6.0	0.0	5.5	0.0
0.0	6.0	0.0	4.5	0.0	0.0	0.0	0.0	5.5
0.0	0.0	5.5	0.0	4.5	0.0	6.0	0.0	0.0

PROC PHREG allows you to specify `test` statements for testing linear hypotheses of the parameters. The test is a Wald test, which is based on the asymptotic normality of the parameter estimators. The following SAS statements invoke PROC PHREG to fit the so called "Mother Logit" model that includes all the cross-alternative effects. The TEST statement, with label IIA, specifies the null hypothesis that cross-alternative effects `AutoPlan`, `PlanTran`, and `TranAuto` are 0. Since only three cross-alternative effects are estimable and these are the first cross-alternative effects specified in the model, they account for all the cross-alternative effects in the model.

```
proc phreg data=choice3;
   model choice*choice(2) = auto plane transit timeauto timeplan
         timetran autoplan plantran tranauto planauto tranplan
         autotran / ties=breslow;
   IIA: test autoplan,plantran,tranauto;
   strata subject;
   title 'Mother Logit Model';
   run;
```

Mother Logit Model

The PHREG Procedure

Convergence Status

Convergence criterion (GCONV=1E-8) satisfied.

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	46.142	24.781
AIC	46.142	40.781
SBC	46.142	49.137

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	21.3607	8	0.0062
Score	15.4059	8	0.0517
Wald	6.2404	8	0.6203

Analysis of Maximum Likelihood Estimates

Variable	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq	Hazard Ratio
Auto	1	-0.73812	3.05933	0.0582	0.8093	0.478
Plane	1	-3.62435	3.48049	1.0844	0.2977	0.027
Transit	0	0	.	.	.	.
TimeAuto	1	-2.23433	1.89921	1.3840	0.2394	0.107
TimePlan	1	-0.10112	0.68621	0.0217	0.8829	0.904
TimeTran	1	0.09785	0.70096	0.0195	0.8890	1.103
AutoPlan	1	0.44495	0.68616	0.4205	0.5167	1.560
PlanTran	1	-0.53234	0.63481	0.7032	0.4017	0.587
TranAuto	1	1.66295	1.51193	1.2097	0.2714	5.275
PlanAuto	0	0	.	.	.	.
TranPlan	0	0	.	.	.	.
AutoTran	0	0	.	.	.	.

Linear Hypotheses Testing Results

Label	Wald Chi-Square	DF	Pr > ChiSq
IIA	1.6526	3	0.6475

The χ^2 statistic for the Wald test is 1.6526 with 3 degrees of freedom, indicating that the cross-alternative effects are not statistically significant ($p = .6475$). A generally more preferable way of testing the significance of the cross-alternative effects is to compare the likelihood of the "Mother logit" model with the likelihood of the reduced model with the cross- alternative effects removed. The following SAS statements invoke PROC PHREG to fit the reduced model:

```
proc phreg data=choice3;
   model choice*choice(2) = auto plane transit timeauto
         timeplan timetran / ties=breslow;
   strata subject;
   title 'Reduced Model without Cross-Alternative Effects';
   run;
```

Reduced Model without Cross-Alternative Effects

The PHREG Procedure

Convergence Status

Convergence criterion (GCONV=1E-8) satisfied.

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	46.142	27.153
AIC	46.142	37.153
SBC	46.142	42.376

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	18.9886	5	0.0019
Score	14.4603	5	0.0129
Wald	7.3422	5	0.1964

Analysis of Maximum Likelihood Estimates

Variable	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq	Hazard Ratio
Auto	1	1.71578	1.80467	0.9039	0.3417	5.561
Plane	1	-3.60073	3.30555	1.1866	0.2760	0.027
Transit	0	0	.	.	.	.
TimeAuto	1	-0.79543	0.36327	4.7946	0.0285	0.451
TimePlan	1	0.12162	0.58954	0.0426	0.8366	1.129
TimeTran	1	-0.42184	0.25733	2.6873	0.1012	0.656

The chi-squared statistic for the likelihood ratio test of IIA is $(27.153 - 24.781) = 2.372$, which is not statistically significant ($p = .4989$) when compared to a χ^2 distribution with 3 degrees of freedom. This is consistent with the previous result of the Wald test. (See pages 269, 275, 283, and 476 for other discussions of IIA.)

Final Comments

For some discrete choice problems, the number of available alternatives is not the same for each individual. For example, in a study of consumer brand choices of laundry detergents as prices change, data are pooled from different locations, not all of which offer a brand that contains potash. The varying choice sets across individuals can easily be accommodated in PROC PHREG. For individual j who chooses from a set of m_j alternatives, consider m_j artificial times in which the chosen alternative has an event time 1 and the unchosen alternatives have a censored time of 2. The analysis is carried out in the same fashion as illustrated in the previous section.

Unlike the example of travel demand in which data for each individual are provided, choice data are often given in aggregate form, with choice frequencies indicating the repetition of each choice. One way of dealing with aggregate data is to expand the data to the individual level and carry out the analysis as if you have nonaggregate data. This approach is generally not recommended, because it

defeats the purpose of having a smaller aggregate data set. PROC PHREG provides a FREQ statement that allows you to specify a variable that identifies the frequency of occurrence of each observation. However, with the specification of a FREQ variable, the artificial event time is no longer the only event time in a given stratum, but has ties of the given frequency. With proper stratification, the Breslow likelihood is proportional to the likelihood of the conditional logit model. Thus PROC PHREG can be used to obtain parameter estimates and hypothesis testing results for the choice models.

The `ties=discrete` option should not be used instead of the `ties=breslow` option. This is especially detrimental with aggregate choice data because the likelihood that PROC PHREG is maximizing may no longer be the same as the likelihood of the conditional logit model. `ties=discrete` corresponds to the discrete logistic model for genuinely discrete time scale, which is also suitable for the analysis of case-control studies when there is more than one case in a matched set (Gail, Lubin, and Rubinstein, 1981). For nonaggregate choice data, all `ties=` options give the same results; however, the resources required for the computation are not the same, with `ties=breslow` being the most efficient.

Once you have a basic understanding of how PROC PHREG works, you can use it to fit a variety of models for the discrete choice data. The major involvement in such a task lies in reorganizing the data to create the observations necessary to form the correct risk sets and the appropriate design variables. There are many options in PROC PHREG that can also be useful in the analysis of discrete choice data. For example, the `offset=` option allows you to restrict the coefficient of an explanatory variable to the value of 1; the `selection=` option allows you to specify one of four methods for selecting variables into the model; the `outest=` option allows you to specify the name of the SAS data set that contains the parameter estimates, based on which you can easily compute the predicted probabilities of the alternatives.

This article deals with estimating parameters of discrete choice models. There is active research in the field of marketing research to use design of experiments to study consumer choice behavior. If you are interested in this area, refer to Carson et al. (1994), Kuhfeld et al. (1994), and Lazari et al. (1994).

Conjoint Analysis

Warren F. Kuhfeld

Abstract

Conjoint analysis is used to study consumers' product preferences and simulate consumer choice. This chapter describes conjoint analysis and provides examples using SAS. Topics include metric and nonmetric conjoint analysis, efficient experimental design, data collection and manipulation, holdouts, brand by price interactions, maximum utility and logit simulators, and change in market share.[†]

Introduction

Conjoint analysis is used to study the factors that influence consumers' purchasing decisions. Products possess attributes such as price, color, ingredients, guarantee, environmental impact, predicted reliability, and so on. Consumers typically do not have the option of buying the product that is best in every attribute, particularly when one of those attributes is price. Consumers are forced to make *trade-offs* as they decide which products to purchase. Consider the decision to purchase a car. Increased size generally means increased safety and comfort. The trade off is an increase in cost and environmental impact and a decrease in gas mileage and maneuverability. Conjoint analysis is used to study these trade-offs.

Conjoint analysis is a popular marketing research technique. It is used in designing new products, changing or repositioning existing products, evaluating the effects of price on purchase intent, and simulating market share. Refer to Green and Rao (1971) and Green and Wind (1975) for early introductions to conjoint analysis, Louviere (1988) for a more recent introduction, and Green and Srinivasan (1990) for a review article.

Conjoint Measurement

Conjoint analysis grew out of the area of *conjoint measurement* in mathematical psychology. Conjoint measurement is used to investigate the joint effect of a set of independent variables on an ordinal-scale-of-measurement dependent variable. The independent variables are typically nominal and sometimes interval-scaled variables. Conjoint measurement simultaneously finds a monotonic scoring of the dependent variable and numerical values for each level of each independent variable. The goal is to monotonically transform the ordinal values to equal the sum of their attribute level values. Hence, conjoint measurement is used to derive an interval variable from ordinal data. The conjoint measurement model is a mathematical model, not a statistical model, since it has no statistical error term.

[†]Copies of this chapter (TS-722H) and all of the macros are available on the web http://support.sas.com/techsup/tnote/tnote_stat.html#market.

Conjoint Analysis

Conjoint analysis is based on a main effects analysis-of-variance model. Subjects provide data about their preferences for hypothetical products defined by attribute combinations. Conjoint analysis decomposes the judgment data into components, based on qualitative attributes of the products. A numerical *part-worth utility* value is computed for each level of each attribute. Large part-worth utilities are assigned to the most preferred levels, and small part-worth utilities are assigned to the least preferred levels. The attributes with the largest part-worth utility range are considered the most important in predicting preference. Conjoint analysis is a statistical model with an error term and a loss function.

Metric conjoint analysis models the judgments directly. When all of the attributes are nominal, the metric conjoint analysis is a simple main-effects ANOVA with some specialized output. The attributes are the independent variables, the judgments comprise the dependent variable, and the part-worth utilities are the β's, the parameter estimates from the ANOVA model. The following is a metric conjoint analysis model for three factors:

$$y_{ijk} = \mu + \beta_{1i} + \beta_{2j} + \beta_{3k} + \epsilon_{ijk}$$

where

$$\sum \beta_{1i} = \sum \beta_{2j} = \sum \beta_{3k} = 0$$

This model could be used, for example, to investigate preferences for cars that differ on three attributes: mileage, expected reliability, and price. The y_{ijk} term is one subject's stated preference for a car with the *ith* level of mileage, the *jth* level of expected reliability, and the *kth* level of price. The grand mean is μ, and the error is ϵ_{ijk}. The predicted utility for the *ijk* product is:

$$\hat{y}_{ijk} = \hat{\mu} + \hat{\beta}_{1i} + \hat{\beta}_{2j} + \hat{\beta}_{3k}$$

Nonmetric conjoint analysis finds a monotonic transformation of the preference judgments. The model, which follows directly from conjoint measurement, iteratively fits the ANOVA model until the transformation stabilizes. The R^2 increases during every iteration until convergence, when the change in R^2 is essentially zero. The following is a nonmetric conjoint analysis model for three factors:

$$\Phi(y_{ijk}) = \mu + \beta_{1i} + \beta_{2j} + \beta_{3k} + \epsilon_{ijk}$$

where $\Phi(y_{ijk})$ designates a monotonic transformation of the variable `y`.

The R^2 for a nonmetric conjoint analysis model will always be greater than or equal to the R^2 from a metric analysis of the same data. The smaller R^2 in metric conjoint analysis is not necessarily a disadvantage, since results should be more stable and reproducible with the metric model. Metric conjoint analysis was derived from nonmetric conjoint analysis as a special case. Today, metric conjoint analysis is probably used more often than nonmetric conjoint analysis.

In the SAS System, conjoint analysis is performed with the SAS/STAT procedure TRANSREG (transformation regression). Metric conjoint analysis models are fit using ordinary least squares, and nonmetric conjoint analysis models are fit using an alternating least squares algorithm (Young, 1981; Gifi,

1990). Conjoint analysis is explained more fully in the examples. The "PROC TRANSREG Specifications" section of this chapter starting on page 585 documents the PROC TRANSREG statements and options that are most relevant to conjoint analysis. The "Samples of PROC TRANSREG Usage" section starting on page 594 shows some typical conjoint analysis specifications. This chapter shows some of the SAS programming that is used for conjoint analysis. Alternatively, there is a marketing research GUI that performs conjoint analysis available from the main display manager PMENU by selecting: `Solutions` → `Analysis` → `Market Research`.

Choice-Based Conjoint

The meaning of the word "conjoint" has broadened over the years from conjoint measurement to conjoint analysis (which at first always meant what we now call nonmetric conjoint analysis) and later to metric conjoint analysis. Metric and nonmetric conjoint analysis are based on a linear ANOVA model. In contrast, a different technique, discrete choice, is based on the nonlinear multinomial logit model. Discrete choice is sometimes referred to as "choice-based conjoint." This technique is not discussed in this chapter, however it is discussed in detail starting on page 141.

Experimental Design

Experimental design is a fundamental component of conjoint analysis. A conjoint study uses experimental design to create a list of products that vary on an assortment of attributes such as brand, price, size, and so on, and subjects rate or rank the products. There are many examples of making conjoint designs in this chapter. Before you read them, be sure to read the design chapters beginning on pages 47 and 99.

The Output Delivery System

The Output Delivery System (ODS) can be used to customize the output of SAS procedures including PROC TRANSREG, the procedure we use for conjoint analysis. PROC TRANSREG can produce a great deal of information for conjoint analysis, more than we often wish to see. We will use ODS primarily to exclude certain portions of the default conjoint output in which we are usually not interested. This will create a better, more parsimonious display for typical analyses. However, when we need it, we can revert back to getting the full array of information. See page 143 for other examples of customizing output using ODS. You can run the following step once to customize PROC TRANSREG conjoint analysis output.

```
proc template;
   edit Stat.Transreg.ParentUtilities;
      column Label Utility StdErr tValue Probt Importance Variable;
      header title;
      define title; text 'Part-Worth Utilities'; space=1; end;
      define Variable; print=off; end;
      end;
   run;
```

Running this step edits the templates for the main conjoint analysis results table and stores a copy in `sasuser`. These changes will remain in effect until you delete them. These changes move the variable label to the first column, turn off printing the variable names, and set the table header to "Part-Worth Utilities". These changes assume that each effect in the model has a variable label associated with it, so there is no need to print variable names. This will usually be the case. To return to the default output, run the following.

```
* Delete edited template, restore original template;
proc template;
   delete Stat.Transreg.ParentUtilities;
   run;
```

By default, PROC TRANSREG prints an ANOVA table for metric conjoint analysis and both univariate and multivariate ANOVA tables for nonmetric conjoint analysis. With nonmetric conjoint analysis, PROC TRANSREG sometimes prints liberal and conservative ANOVA tables. All of the possible ANOVA tables, along with some header notes, can be suppressed by specifying the following statement before running PROC TRANSREG.

```
ods exclude notes anova liberalanova conservanova
            mvanova liberalmvanova conservmvanova;
```

For metric conjoint analysis, this statement can be abbreviated as follows.

```
ods exclude notes mvanova anova;
```

The rest of this section gives more details about what the PROC TEMPLATE step does and why. The rest of this section can be helpful if you wish to further customize the output from TRANSREG or some other procedure. Impatient readers may skip ahead to the candy example on page 489.

We are most interested in the part-worth utilities table in conjoint analysis, which contains the part-worth utilities, their standard errors, and the importance of each attribute. We can first use PROC TEMPLATE to identify the template for the utilities table and then edit the template. First, let's have PROC TEMPLATE display the templates for PROC TRANSREG. The `source stat.transreg` statement specifies that we want to see PROC TEMPLATE source code for the STAT product and the TRANSREG procedure.

```
proc template;
   source stat.transreg;
   run;
```

If we search the results for "Utilities", we find the template for the table `Stat.Transreg.ParentUtilities`. Here is the template for the part-worth utilities table.

```
define table Stat.Transreg.ParentUtilities;
   notes "Parent Utilities Table for Proc Transreg";
   dynamic FootMessages TitleText;
   column Label Utility StdErr tValue Probt Importance Variable;
   header Title;
   footer Foot;
   define Title;
      text TitleText;
      space = 1;
      spill_margin;
      first_panel;
   end;
```

```
      define Label;
         parent = Stat.Transreg.Label;
         style = RowHeader;
      end;
      define Utility;
         header = "Utility";
         format_width = 7;
         parent = Stat.Transreg.Coefficient;
      end;
      define StdErr;
         parent = Stat.Transreg.StdErr;
      end;
      define tValue;
         parent = Stat.Transreg.tValue;
         print = OFF;
      end;
      define Probt;
         parent = Stat.Transreg.Probt;
         print = OFF;
      end;
      define Importance;
         header = %nrstr(";Importance;%(%% Utility;Range%)");
         translate _val_=._ into " ";
         format = 7.3;
      end;
      define Variable;
         parent = Stat.Transreg.Variable;
      end;
      define Foot;
         text FootMessages;
         just = l;
         maximize;
      end;
      control = control;
      required_space = 20;
   end;
```

Recall that we ran the following step to customize the output.

```
   proc template;
      edit Stat.Transreg.ParentUtilities;
         column Label Utility StdErr tValue Probt Importance Variable;
         header title;
         define title; text 'Part-Worth Utilities'; space=1; end;
         define Variable; print=off; end;
         end;
      run;
```

We specified the `edit Stat.Transreg.ParentUtilities` statement to name the table that we wish to change. The `column` statement was copied from the PROC TEMPLATE source listing, and it names all of the columns in the table. Some, like `tValue` and `Probt` do not print by default. We will change the `Variable` column to also not print. We redefine `Variable` with the `print=off` option specified. We also redefine the table header to read "Part-Worth Utilities". The names in the `column` and `header` statements must match the names in the original template.

Chocolate Candy Example

This example illustrates conjoint analysis with rating scale data and a single subject. The subject was asked to rate his preference for eight chocolate candies. The covering was either dark or milk chocolate, the center was either chewy or soft, and the candy did or did not contain nuts. The candies were rated on a 1 to 9 scale where 1 means low preference and 9 means high preference. Conjoint analysis is used to determine the importance of each attribute and the part-worth utility for each level of each attribute.

Metric Conjoint Analysis

After data collection, the attributes and the rating data are entered into a SAS data set.

```
title 'Preference for Chocolate Candies';

data choc;
   input Chocolate $ Center $ Nuts $& Rating;
   datalines;
Dark  Chewy  Nuts       7
Dark  Chewy  No Nuts    6
Dark  Soft   Nuts       6
Dark  Soft   No Nuts    4
Milk  Chewy  Nuts       9
Milk  Chewy  No Nuts    8
Milk  Soft   Nuts       9
Milk  Soft   No Nuts    7
;
```

Note that the "`&`" specification in the `input` statement is used to read character data with embedded blanks.

PROC TRANSREG is used to perform a metric conjoint analysis.

```
ods exclude notes mvanova anova;
proc transreg utilities separators=', ' short;
   title2 'Metric Conjoint Analysis';
   model identity(rating) = class(chocolate center nuts / zero=sum);
   run;
```

Printed output from the metric conjoint analysis is requested by specifying the `utilities` option in the `proc` statement. The value specified in the `separators=` option, in this case a comma followed by a blank, is used in constructing the labels for the part-worth utilities in the printed output. With these options, the labels will consist of the `class` variable name, a comma, a blank and the values of the `class` variables. We specify the `short` option to suppress the iteration history. PROC TRANSREG will still print a convergence summary table so we will know if there are any convergence problems. Since this is a metric conjoint analysis, there should be only one iteration and there should not be any problems. We specified `ods exclude notes mvanova anova` to exclude ANOVA information (which we usually want to ignore) and provide more parsimonious output. The analysis variables, the transformation of each variable, and transformation specific options are specified in the `model` statement.

The `model` statement provides for general transformation regression models, so it has a markedly different syntax from other SAS/STAT procedure `model` statements. Variable lists are specified in parentheses after a transformation name. The specification `identity(rating)` requests an `identity` transformation of the dependent variable `Rating`. A transformation name must be specified for all variable lists, even for the dependent variable in metric conjoint analysis, when no transformation is desired. The `identity` transformation of `Rating` will not change the original scoring. An equal sign follows the dependent variable specification, then the attribute variables are specified along with their transformation. The specification

```
class(chocolate center nuts / zero=sum)
```

designates the attributes as `class` variables with the restriction that the part-worth utilities sum to zero within each attribute. A slash must be specified to separate the variables from the transformation option `zero=sum`. The `class` specification creates a main-effects design matrix from the specified variables. This example produces only printed output; later examples will show how to store results in output SAS data sets.

The output is shown next. Recall that we used an `ods exclude` statement and we used PROC TEMPLATE on page 485 to customize the output from PROC TRANSREG.

Preference for Chocolate Candies
Metric Conjoint Analysis

The TRANSREG Procedure

Dependent Variable Identity(Rating)

Class Level Information

Class	Levels	Values
Chocolate	2	Dark Milk
Center	2	Chewy Soft
Nuts	2	No Nuts Nuts

Number of Observations Read	8
Number of Observations Used	8

Identity(Rating)
Algorithm converged.

Root MSE	0.50000	R-Square	0.9500
Dependent Mean	7.00000	Adj R-Sq	0.9125
Coeff Var	7.14286		

Part-Worth Utilities

Label	Utility	Standard Error	Importance (% Utility Range)
Intercept	7.0000	0.17678	
Chocolate, Dark	-1.2500	0.17678	50.000
Chocolate, Milk	1.2500	0.17678	
Center, Chewy	0.5000	0.17678	20.000
Center, Soft	-0.5000	0.17678	
Nuts, No Nuts	-0.7500	0.17678	30.000
Nuts, Nuts	0.7500	0.17678	

We see `Algorithm converged` in the output indicating no problems with the iterations. We also see $R^2 = 0.95$. The last table displays the part-worth utilities. The part-worth utilities show the most and least preferred levels of the attributes. Levels with positive utility are preferred over those with negative utility. Milk chocolate (part-worth utility = 1.25) was preferred over dark (-1.25), chewy center (0.5) over soft (-0.5), and nuts (0.75) over no nuts (-0.75).

Conjoint analysis provides an approximate decomposition of the original ratings. The predicted utility for a candy is the sum of the intercept and the part-worth utilities. The conjoint analysis model for the preference for chocolate type i, center j, and nut content k is

$$y_{ijk} = \mu + \beta_{1i} + \beta_{2j} + \beta_{3k} + \epsilon_{ijk}$$

for $i = 1, 2$; $j = 1, 2$; $k = 1, 2$; where

$$\beta_{11} + \beta_{12} = \beta_{21} + \beta_{22} = \beta_{31} + \beta_{32} = 0$$

The part-worth utilities for the attribute levels are the parameter estimates $\hat{\beta}_{11}$, $\hat{\beta}_{12}$, $\hat{\beta}_{21}$, $\hat{\beta}_{22}$, $\hat{\beta}_{31}$, and $\hat{\beta}_{32}$ from this main-effects ANOVA model. The estimate of the intercept is $\hat{\mu}$, and the error term is ϵ_{ijk}.

The predicted utility for the ijk combination is

$$\hat{y}_{ijk} = \hat{\mu} + \hat{\beta}_{1i} + \hat{\beta}_{2j} + \hat{\beta}_{3k}$$

For the most preferred milk/chewy/nuts combination, the predicted utility and actual preference values are

$$7.0 + 1.25 + 0.5 + 0.75 = 9.5 = \hat{y} \approx y = 9.0$$

For the least preferred dark/soft/no nuts combination, the predicted utility and actual preference values are

$$7.0 + -1.25 + -0.5 + -0.75 = 4.5 = \hat{y} \approx y = 4.0$$

The predicted utilities are regression predicted values; the squared correlation between the predicted utilities for each combination and the actual preference ratings is the R^2.

The *importance* value is computed from the part-worth utility range for each factor (attribute). Each range is divided by the sum of all ranges and multiplied by 100. The factors with the largest part-worth utility ranges are the most important in determining preference. Note that when the attributes have a varying number of levels, attributes with the most levels sometimes have inflated importances (Wittink, Krishnamurthi, and Reibstein; 1989).

The importance values show that type of chocolate, with an importance of 50%, was the most important attribute in determining preference.

$$\frac{100 \times (1.25 - -1.25)}{(1.25 - -1.25) + (0.50 - -0.50) + (0.75 - -0.75)} = 50\%$$

The second most important attribute was whether the candy contained nuts, with an importance of 30%.

$$\frac{100 \times (0.75 - -0.75)}{(1.25 - -1.25) + (0.50 - -0.50) + (0.75 - -0.75)} = 30\%$$

Type of center was least important at 20%.

$$\frac{100 \times (0.50 - -0.50)}{(1.25 - -1.25) + (0.50 - -0.50) + (0.75 - -0.75)} = 20\%$$

Nonmetric Conjoint Analysis

In the next part of this example, PROC TRANSREG is used to perform a nonmetric conjoint analysis of the candy data set. The difference between requesting a nonmetric and metric conjoint analysis is the dependent variable transformation; a `monotone` transformation of `Rating` variable is requested instead of an `identity` transformation. Also, we did not specify the `short` option this time so that we could see the iteration history table. The `output` statement is used to put the transformed rating into the `out=` output data set.

```
ods exclude notes anova liberalanova conservanova
            mvanova liberalmvanova conservmvanova;
proc transreg utilities separators=', ';
   title2 'Nonmetric Conjoint Analysis';
   model monotone(rating) = class(chocolate center nuts / zero=sum);
   output;
   run;
```

Nonmetric conjoint analysis iteratively derives the monotonic transformation of the ratings. Recall that we used an `ods exclude` statement and we used PROC TEMPLATE on page 485 to customize the output from PROC TRANSREG.

Preference for Chocolate Candies
Nonmetric Conjoint Analysis

The TRANSREG Procedure

Dependent Variable Monotone(Rating)

Class Level Information

Class	Levels	Values
Chocolate	2	Dark Milk
Center	2	Chewy Soft
Nuts	2	No Nuts Nuts

Number of Observations Read	8
Number of Observations Used	8

TRANSREG Univariate Algorithm Iteration History for Monotone(Rating)

Iteration Number	Average Change	Maximum Change	R-Square	Criterion Change	Note
1	0.08995	0.23179	0.95000		
2	0.01263	0.03113	0.96939	0.01939	
3	0.00345	0.00955	0.96981	0.00042	
4	0.00123	0.00423	0.96984	0.00003	
5	0.00050	0.00182	0.96985	0.00000	
6	0.00021	0.00078	0.96985	0.00000	
7	0.00009	0.00033	0.96985	0.00000	
8	0.00004	0.00014	0.96985	0.00000	
9	0.00002	0.00006	0.96985	0.00000	
10	0.00001	0.00003	0.96985	0.00000	Converged

Algorithm converged.

Root MSE	0.38829	R-Square	0.9698
Dependent Mean	7.00000	Adj R-Sq	0.9472
Coeff Var	5.54699		

Part-Worth Utilities

Label	Utility	Standard Error	Importance (% Utility Range)
Intercept	7.0000	0.13728	
Chocolate, Dark	-1.3143	0.13728	53.209
Chocolate, Milk	1.3143	0.13728	
Center, Chewy	0.4564	0.13728	18.479
Center, Soft	-0.4564	0.13728	
Nuts, No Nuts	-0.6993	0.13728	28.312
Nuts, Nuts	0.6993	0.13728	

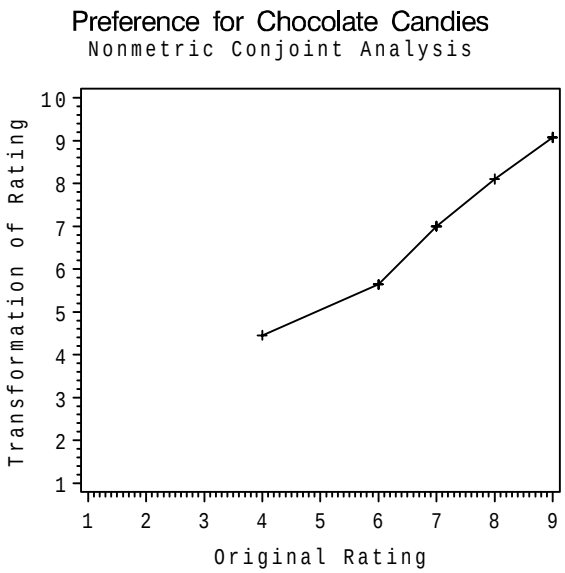
The standard errors are not adjusted for the fact that the dependent variable was transformed and so are generally liberal (too small).

The R^2 increases from 0.95 for the metric case to 0.96985 for the nonmetric case. The importances and part-worth utilities are slightly different from the metric analysis, but the overall pattern of results is the same.

The GPLOT procedure is used to plot the transformation of the ratings.

```
proc sort; by rating; run;

proc gplot;
   title  h=1.5 'Preference for Chocolate Candies';
   title2 h=1 'Nonmetric Conjoint Analysis';
   plot trating * rating = 1 / frame haxis=axis2 vaxis=axis1;
   symbol1 v=plus i=join;
   axis1 order=(1 to 10)
         label=(angle=90 'Transformation of Rating');
   axis2 order=(1 to  9) label=('Original Rating');
   run; quit;
```



In this case, the transformation is nearly linear. In practice, the R^2 may increase much more than it did in this example, and the transformation may be markedly nonlinear.

Frozen Diet Entrées Example (Basic)

This example uses PROC TRANSREG to perform a conjoint analysis to study preferences for frozen diet entrées. The entrées have four attributes: three with three levels and one with two levels. The attributes are shown in the table.

Factor	Levels		
Main Ingredient	Chicken	Beef	Turkey
Fat Claim Per Serving	8 Grams	5 Grams	2 Grams
Price	$2.59	$2.29	$1.99
Calories	350	250	

Choosing the Number of Stimuli

Ideally, for this design, we would like the number of *runs* in the experimental design to be divisible by 2 (because of the two-level factor), 3 (because of the three-level factors), $2 \times 3 = 6$ (to have equal numbers of products in each two-level and three-level factor combinations), and $3 \times 3 = 9$ (to have equal numbers of products in each pair of three-level factor combinations). If we fit a main-effects model, we need at least $1 + 3 \times (3 - 1) + (2 - 1) = 8$ runs. We can avoid doing this math ourselves and instead use the `%MktRuns` autocall macro to help us choose the number of products. See page 597 for macro documentation and information on installing and using SAS autocall macros. To use this macro, you specify the number of levels for each of the factors. For this example, specify three 3's and one 2.

```
title 'Frozen Diet Entrees';

%mktruns( 3 3 3 2 )
```

```
                    Frozen Diet Entrees

                      Design Summary

                    Number of
                    Levels       Frequency

                        2               1
                        3               3

                    Frozen Diet Entrees

              Saturated      = 8
              Full Factorial = 54
```

```
 Some Reasonable                      Cannot Be
    Design Sizes        Violations    Divided By

           18 *                0
           36 *                0
           12                  3     9
           24                  3     9
           30                  3     9
            9                  4     2 6
           27                  4     2 6
           15                  7     2 6 9
           21                  7     2 6 9
           33                  7     2 6 9

* - 100% Efficient Design can be made with the MktEx Macro.

     n    Design                              Reference

    18    2 **  1  3 **  7                   Orthogonal Array
    36    2 ** 16  3 **  4                   Orthogonal Array
    36    2 ** 11  3 ** 12                   Orthogonal Array
    36    2 ** 10  3 **  8  6 **  1          Orthogonal Array
    36    2 **  9  3 **  4  6 **  2          Orthogonal Array
    36    2 **  4  3 ** 13                   Orthogonal Array
    36    2 **  3  3 **  9  6 **  1          Orthogonal Array
    36    2 **  2  3 ** 12  6 **  1          Orthogonal Array
    36    2 **  2  3 **  5  6 **  2          Orthogonal Array
    36    2 **  1  3 **  8  6 **  2          Orthogonal Array
    36    2 **  1  3 **  3  6 **  3          Orthogonal Array
```

The output tells us that we need at least eight products, shown by the "Saturated = 8". The sizes 18 and 36 would be optimal. Twelve is a good size but three times it cannot be divided by $9 = 3 \times 3$. The "three times" comes from the $3(3-1)/2 = 3$ pairs of three-level factors. Similarly, the size 9 has four violations because it cannot be divided once by 2 and three times by $6 = 2 \times 3$ (once for each three-level factor and two-level factor pair). We could use a size smaller than 18 and not have equal frequencies everywhere, but 18 is a manageable number so we will use 18.

When an orthogonal and balanced design is available from the `%MktEx` macro, the `%MktRuns` macro tells us about it. For example, the macro tells us that our design, which can be designated $2^1 3^3$, is available in 18 runs, and can be constructed from a design with 1 two-level factor (`2 ** 1` or 2^1) and 7 three-level factors (`3 ** 7` or 3^7). Both the `%MktRuns` and `%MktEx` macros accept this '$n**m$' exponential syntax as input, which means m factors each at n levels. Hence, `2 3 ** 7` or `2 ** 1 3 ** 7` or `2 3 3 3 3 3 3 3` are all equivalent level-list specifications for the experimental design $2^1 3^7$, which has 1 two-level factor and 7 three-level factors.

Generating the Design

We can use the %MktEx autocall macro to find a design. When you invoke the %MktEx macro for a simple problem, you only need to specify the numbers of levels and number of runs. The macro does the rest. The %MktEx macro can create designs in a number of ways. For this problem, it simply looks up an orthogonal design. Here is the %MktEx macro call for this example:

```
%mktex(3 3 3 2, n=18)
```

The first argument to the %MktEx macro is a list of factor levels, and the second is the number of runs (n=18). These are all the options that are needed for a simple problem such as this one. However, throughout this book, random number seeds are explicitly specified with the seed= option so that you can reproduce these results.‡ Here is the code for creating our design with the random number seed and the actual factor names specified:

```
%mktex(3 3 3 2, n=18, seed=151)
%mktlab(vars=Ingredient Fat Price Calories)
```

The %MktEx macro always creates factors named x1, x2, and so on. The %MktLab autocall macro is used to change the names when you want to provide actual factor names. This example has four factors, Ingredient, Fat, and Price, each with three levels and Calories with two levels.

Here is the output:

Frozen Diet Entrees

Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
1	Start	100.0000	100.0000	Tab
1	End	100.0000		

Frozen Diet Entrees

The OPTEX Procedure

Class Level Information

Class	Levels	-Values-
x1	3	1 2 3
x2	3	1 2 3
x3	3	1 2 3
x4	2	1 2

‡By specifying a random number seed, results should be reproducible within a SAS release for a particular operating system. However, due to machine differences, some results may not be exactly reproducible on other machines. For most orthogonal and balanced designs, the results should be reproducible. When computerized searches are done, it is likely that you will not get the same design as the one in the book, although you would expect the efficiency differences to be slight.

Frozen Diet Entrees

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	100.0000	100.0000	100.0000	0.6667

We see that the macro had no trouble finding an optimal, 100% efficient experimental design through table look up. The value `Tab` in the `Notes` column of the algorithm search history tells us the macro was able to use table look up. See pages 597, 667, and the discrete choice examples starting on page 141 for more information on how the `%MktEx` macro works.

The `%MktEx` macro creates two output data sets with the experimental design, `Design` and `Randomized`. The `Design` data set is sorted and for a number of the tabled designs, often has a first row consisting entirely of ones. For these reasons, you should typically use the *randomized* design. In the randomized design, the profiles are presented in a random order and the levels have been randomly reassigned. Neither of these operations affects the design efficiency, balance, or orthogonality. When there are restrictions on the design (see for example page 551), the profiles are sorted into a random order, but the levels are *not* randomly reassigned. The randomized design is the default input to the `%MktLab` macro.

Evaluating and Preparing the Design

We will use the FORMAT procedure to create descriptive labels for the levels of the attributes. By default, the values of the factors are positive integers. For example for `ingredient`, we create a format `if` (for Ingredient Format) that assigns the descriptive value label "Chicken" for level 1, "Beef" for level 2, and "Turkey" for level 3. A permanent SAS data set is created with the formats assigned (although, as we will see in the next example, we could have done this previously in the `%MktLab` step). Finally, the design is printed.

```
proc format;
   value if  1='Chicken'  2='Beef'     3='Turkey';
   value ff  1='8 Grams'  2='5 Grams'  3='2 Grams';
   value pf  1='$2.59'    2='$2.29'    3='$1.99';
   value cf  1='350'      2='250';
   run;
data sasuser.dietdes;
   set final;
   format ingredient if. fat ff. price pf. calories cf.;
   run;
proc print; run;
```

Here is the design.

Frozen Diet Entrees

Obs	Ingredient	Fat	Price	Calories
1	Turkey	5 Grams	$1.99	350
2	Turkey	8 Grams	$2.29	350
3	Chicken	8 Grams	$1.99	350
4	Turkey	2 Grams	$2.59	250
5	Beef	8 Grams	$2.59	350
6	Beef	2 Grams	$1.99	350
7	Beef	5 Grams	$2.29	350
8	Beef	5 Grams	$2.29	250
9	Chicken	2 Grams	$2.29	350
10	Beef	8 Grams	$2.59	250
11	Turkey	8 Grams	$2.29	250
12	Chicken	5 Grams	$2.59	350
13	Chicken	5 Grams	$2.59	250
14	Chicken	2 Grams	$2.29	250
15	Turkey	5 Grams	$1.99	250
16	Turkey	2 Grams	$2.59	350
17	Beef	2 Grams	$1.99	250
18	Chicken	8 Grams	$1.99	250

Even when you know the design is 100% *D*-efficient, orthogonal, and balanced, it is good to run basic checks on your designs. You can use the `%MktEval` autocall macro to display information about the design.

```
%mkteval;
```

The macro first prints a matrix of canonical correlations between the factors. We hope to see an identity matrix (a matrix of ones on the diagonal and zeros everywhere else), which would mean that all of the factors are uncorrelated. Next, the macro prints all one-way frequencies for all attributes, all two-way frequencies, and all *n*-way frequencies (in this case four-way frequencies). We hope to see equal or at least nearly equal one-way and two-way frequencies, and we want to see that each combination occurs only once.

Frozen Diet Entrees
Canonical Correlations Between the Factors
There are 0 Canonical Correlations Greater Than 0.316

	Ingredient	Fat	Price	Calories
Ingredient	1	0	0	0
Fat	0	1	0	0
Price	0	0	1	0
Calories	0	0	0	1

```
                    Frozen Diet Entrees
                  Summary of Frequencies
There are 0 Canonical Correlations Greater Than 0.316

                          Frequencies

Ingredient                6 6 6
Fat                       6 6 6
Price                     6 6 6
Calories                  9 9
Ingredient Fat            2 2 2 2 2 2 2 2 2
Ingredient Price          2 2 2 2 2 2 2 2 2
Ingredient Calories       3 3 3 3 3 3
Fat Price                 2 2 2 2 2 2 2 2 2
Fat Calories              3 3 3 3 3 3
Price Calories            3 3 3 3 3 3
N-Way                     1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

A *canonical correlation* is the maximum correlation between linear combinations of the coded factors (see page 70). All zeros off the diagonal show that this design is orthogonal for main effects. If any off-diagonal canonical correlations had been greater than 0.316 ($r^2 > 0.1$), the macro would have listed them in a separate table. The last title line tells you that none of them were this large. For nonorthogonal designs and designs with interactions, the canonical-correlation matrix is not a substitute for looking at the variance matrix (with `examine=v`) in the `%MktEx` macro. The `%MktEx` macro just provides a quick and more-compact picture of the correlations between the factors. The variance matrix is sensitive to the actual model specified and the coding. The canonical-correlation matrix just tells you if there is some correlation between the main effects. In this case, there are no correlations.

The equal one-way frequencies show you that this design is balanced. The equal two-way frequencies show you that this design is orthogonal. Equal one-way and two-way frequencies together show you that this design is 100% D-efficient. The n-way frequencies, all equal to one, show you that there are no duplicate profiles. This is a perfect design for a main effects model. However, there are other 100% efficient designs for this problem with duplicate observations. In the last part of the output, the n-Way frequencies may contain some 2's for those designs. You can specify `options=nodups` in the `%MktEx` macro to ensure that there are no duplicate profiles.

The `%MktEval` macro produces a very compact summary of the design, hence some information, for example the levels to which the frequencies correspond, is not shown. You can use the `print=freqs` option in the `%MktEval` macro to get a less compact and more detailed display.

Printing the Stimuli and Data Collection

Next, we generate the stimuli. The `data _null_` step uses the `file` statement to set the print destination to the printed output destination. The design data set is read with the `set` statement. A `put` statement prints the attributes along with some constant text and the combination number. The `put` statement option `+3` skips 3 spaces, `@50` starts printing in column 50, `+(-1)` skips one space *backwards* getting rid of the blank that would by default appear after the stimulus number, and `/` skips to a new line. Text enclosed in quotes is literally copied to the output. For our attribute variables, the formatted

values are printed. The variable _n_ is the number of the current pass through the DATA step, which in this case is the stimulus number. The if statement causes six descriptions to be printed on a page.

```
title;
data _null_;
   file print;
   set sasuser.dietdes;
   put ///
       +3 ingredient 'Entree' @50 '(' _n_ +(-1) ')' /
       +3 'With ' fat 'of Fat and ' calories  'Calories' /
       +3 'Now for Only ' Price +(-1) '.'///;
   if mod(_n_, 6) = 0 then put _page_;
   run;
```

```
Turkey Entree                                (1)
With 5 Grams of Fat and 350 Calories
Now for Only $1.99.

Turkey Entree                                (2)
With 8 Grams of Fat and 350 Calories
Now for Only $2.29.

Chicken Entree                               (3)
With 8 Grams of Fat and 350 Calories
Now for Only $1.99.

Turkey Entree                                (4)
With 2 Grams of Fat and 250 Calories
Now for Only $2.59.

Beef Entree                                  (5)
With 8 Grams of Fat and 350 Calories
Now for Only $2.59.

Beef Entree                                  (6)
With 2 Grams of Fat and 350 Calories
Now for Only $1.99.

Beef Entree                                  (7)
With 5 Grams of Fat and 350 Calories
Now for Only $2.29.

Beef Entree                                  (8)
With 5 Grams of Fat and 250 Calories
Now for Only $2.29.

Chicken Entree                               (9)
With 2 Grams of Fat and 350 Calories
Now for Only $2.29.
```

```
Beef Entree                                (10)
With 8 Grams of Fat and 250 Calories
Now for Only $2.59.

Turkey Entree                              (11)
With 8 Grams of Fat and 250 Calories
Now for Only $2.29.

Chicken Entree                             (12)
With 5 Grams of Fat and 350 Calories
Now for Only $2.59.

Chicken Entree                             (13)
With 5 Grams of Fat and 250 Calories
Now for Only $2.59.

Chicken Entree                             (14)
With 2 Grams of Fat and 250 Calories
Now for Only $2.29.

Turkey Entree                              (15)
With 5 Grams of Fat and 250 Calories
Now for Only $1.99.

Turkey Entree                              (16)
With 2 Grams of Fat and 350 Calories
Now for Only $2.59.

Beef Entree                                (17)
With 2 Grams of Fat and 250 Calories
Now for Only $1.99.

Chicken Entree                             (18)
With 8 Grams of Fat and 250 Calories
Now for Only $1.99.
```

Next, we print the stimuli, produce the cards, and ask a subject to sort the cards from most preferred to least preferred. The combination numbers (most preferred to least preferred) are entered as data. For example, this subject's most preferred combination is 17, which is the "Beef Entree, With 2 Grams of Fat and 250 Calories, Now for Only $1.99", and her least preferred combination is 18, "Chicken Entree, With 8 Grams of Fat and 250 Calories, Now for Only $1.99".

Data Processing

The data are transposed, going from one observation and 18 variables to 18 observations and one variable named `Combo`. The next DATA step creates the variable `Rank`: 1 for the first and most preferred combination, ..., and 18 for the last and least preferred combination. The data are sorted by combination number and merged with the design.

```
title 'Frozen Diet Entrees';

data results;
   input combo1-combo18;
   datalines;
17 6 8 7 10 5 4 16 15 1 11 2 9 14 12 13 3 18
;
proc transpose out=results(rename=(col1=combo)); run;

data results; set results; Rank = _n_; drop _name_; run;

proc sort; by combo; run;

data results(drop=combo);
   merge sasuser.dietdes results;
   run;
proc print; run;
```

Frozen Diet Entrees

Obs	Ingredient	Fat	Price	Calories	Rank
1	Turkey	5 Grams	$1.99	350	10
2	Turkey	8 Grams	$2.29	350	12
3	Chicken	8 Grams	$1.99	350	17
4	Turkey	2 Grams	$2.59	250	7
5	Beef	8 Grams	$2.59	350	6
6	Beef	2 Grams	$1.99	350	2
7	Beef	5 Grams	$2.29	350	4
8	Beef	5 Grams	$2.29	250	3
9	Chicken	2 Grams	$2.29	350	13
10	Beef	8 Grams	$2.59	250	5
11	Turkey	8 Grams	$2.29	250	11
12	Chicken	5 Grams	$2.59	350	15
13	Chicken	5 Grams	$2.59	250	16
14	Chicken	2 Grams	$2.29	250	14
15	Turkey	5 Grams	$1.99	250	9
16	Turkey	2 Grams	$2.59	350	8
17	Beef	2 Grams	$1.99	250	1
18	Chicken	8 Grams	$1.99	250	18

Recall that the seventeenth combination was most preferred, and it has a rank of 1. The eighteenth combination was least preferred and it has a rank of 18.

Nonmetric Conjoint Analysis

PROC TRANSREG is used to perform the nonmetric conjoint analysis of the ranks.

```
ods exclude notes anova liberalanova conservanova
            mvanova liberalmvanova conservmvanova;
proc transreg utilities order=formatted separators=', ';
   model monotone(rank / reflect) =
         class(Ingredient Fat Price Calories / zero=sum);
   output out=utils p ireplace;
   run;
```

The `utilities` option prints the part-worth utilities and importance table. The `order=formatted` option sorts the levels of the attributes by the formatted values. By default, levels are sorted by their internal unformatted values (in this case the integers 1, 2, 3). The `model` statement names the variable `Rank` as the dependent variable and specifies a `monotone` transformation for the nonmetric conjoint analysis. The `reflect` transformation option is specified with rank data. With rank data, small values mean high preference and large values mean low preference. The `reflect` transformation option reflects the ranks around their mean (-(rank - mean rank) + mean rank) so that in the results, large part-worth utilities will mean high preference. With ranks ranging from 1 to 18, `reflect` transforms 1 to 18, 2 to 17, ..., r to $(19 - r)$, ..., and 18 to 1. (Note that the mean rank is the midpoint, in this case $(18+1)/2 = 9.5$, and $-(r-\bar{r})+\bar{r} = 2\bar{r}-r = 2(\max(r)+\min(r))/2-r = 19-r$.) The `class` specification names the attributes and scales the part-worth utilities to sum to zero within each attribute.

The `output` statement creates the `out=` data set, which contains the original variables, transformed variables, and indicator variables. The predicted utilities for all combinations are written to this data set by the `p` option (for predicted values). The `ireplace` option specifies that the transformed independent variables replace the original independent variables, since both are the same.

Here are the results of the conjoint analysis. Recall that we used an `ods exclude` statement and we used PROC TEMPLATE on page 485 to customize the output from PROC TRANSREG.

Frozen Diet Entrees

The TRANSREG Procedure

Dependent Variable Monotone(Rank)

Class Level Information

Class	Levels	Values
Ingredient	3	Beef Chicken Turkey
Fat	3	2 Grams 5 Grams 8 Grams
Price	3	$1.99 $2.29 $2.59
Calories	2	250 350

Number of Observations Read	18
Number of Observations Used	18

TRANSREG Univariate Algorithm Iteration History for Monotone(Rank)

Iteration Number	Average Change	Maximum Change	R-Square	Criterion Change	Note
1	0.07276	0.10014	0.99174		
2	0.00704	0.01074	0.99977	0.00802	
3	0.00468	0.00710	0.99990	0.00013	
4	0.00311	0.00470	0.99995	0.00006	
5	0.00207	0.00312	0.99998	0.00003	
6	0.00138	0.00208	0.99999	0.00001	
7	0.00092	0.00138	1.00000	0.00001	
8	0.00061	0.00092	1.00000	0.00000	
9	0.00041	0.00061	1.00000	0.00000	
10	0.00027	0.00041	1.00000	0.00000	
11	0.00018	0.00027	1.00000	0.00000	
12	0.00012	0.00018	1.00000	0.00000	
13	0.00008	0.00012	1.00000	0.00000	
14	0.00005	0.00008	1.00000	0.00000	
15	0.00004	0.00005	1.00000	0.00000	
16	0.00002	0.00004	1.00000	0.00000	
17	0.00002	0.00002	1.00000	0.00000	
18	0.00001	0.00002	1.00000	0.00000	
19	0.00001	0.00001	1.00000	0.00000	Converged

Algorithm converged.

Frozen Diet Entrees

The TRANSREG Procedure

The TRANSREG Procedure Hypothesis Tests for Monotone(Rank)

Root MSE	0.00007166	R-Square	1.0000
Dependent Mean	9.50000	Adj R-Sq	1.0000
Coeff Var	0.00075429		

Part-Worth Utilities

Label	Utility	Standard Error	Importance (% Utility Range)
Intercept	9.5000	0.00002	
Ingredient, Beef	6.0281	0.00002	74.999
Ingredient, Chicken	-6.0281	0.00002	
Ingredient, Turkey	-0.0000	0.00002	
Fat, 2 Grams	2.0094	0.00002	25.000
Fat, 5 Grams	0.0000	0.00002	
Fat, 8 Grams	-2.0094	0.00002	
Price, $1.99	0.0000	0.00002	0.000
Price, $2.29	0.0000	0.00002	
Price, $2.59	-0.0000	0.00002	
Calories, 250	0.0001	0.00002	0.001
Calories, 350	-0.0001	0.00002	

The standard errors are not adjusted for the fact that the dependent variable was transformed and so are generally liberal (too small).

We see in the conjoint output that main ingredient was the most important attribute at almost 75% and that beef was preferred over turkey, which was preferred over chicken. We also see that fat content was the second most important attribute at 25% and lower fat is preferred over higher fat. Price and calories only account for essentially none of the preference.

Next, the products in the `out=` data set are sorted by their predicted utility and the combinations are printed along with their rank, transformed and reflected rank, and predicted values (predicted utility). The variable `Rank` is the original rank variable; `TRank` contains the transformation of rank, in this case the reflection and monotonic transformation; and `PRank` contains the predicted utilities or predicted values. The first letter of the variable name comes from the first letter of "Transformation" and "Predicted".

```
proc sort; by descending prank; run;

proc print label;
   var ingredient fat price calories rank trank prank;
   label trank = 'Reflected Rank'
         prank = 'Utilities';
   run;
```

Frozen Diet Entrees

Obs	Ingredient	Fat	Price	Calories	Rank	Reflected Rank	Utilities
1	Beef	2 Grams	$1.99	250	1	17.5375	17.5375
2	Beef	2 Grams	$1.99	350	2	17.5373	17.5373
3	Beef	5 Grams	$2.29	250	3	15.5282	15.5281
4	Beef	5 Grams	$2.29	350	4	15.5279	15.5280
5	Beef	8 Grams	$2.59	250	5	13.5188	13.5188
6	Beef	8 Grams	$2.59	350	6	13.5186	13.5186
7	Turkey	2 Grams	$2.59	250	7	11.5095	11.5094
8	Turkey	2 Grams	$2.59	350	8	11.5092	11.5093
9	Turkey	5 Grams	$1.99	250	9	9.5001	9.5001
10	Turkey	5 Grams	$1.99	350	10	9.4999	9.4999
11	Turkey	8 Grams	$2.29	250	11	7.4908	7.4907
12	Turkey	8 Grams	$2.29	350	12	7.4905	7.4906
13	Chicken	2 Grams	$2.29	250	14	5.4813	5.4814
14	Chicken	2 Grams	$2.29	350	13	5.4813	5.4812
15	Chicken	5 Grams	$2.59	250	16	3.4719	3.4720
16	Chicken	5 Grams	$2.59	350	15	3.4719	3.4719
17	Chicken	8 Grams	$1.99	250	18	1.4626	1.4627
18	Chicken	8 Grams	$1.99	350	17	1.4626	1.4625

It is interesting to see that the sorted combinations support the information in the utilities table. The combinations are perfectly sorted on beef, turkey, and chicken. Furthermore, within ties in the main ingredient, the products are sorted by fat content.

Frozen Diet Entrées Example (Advanced)

This example is an advanced version of the previous example. It illustrates conjoint analysis with more than one subject. It has six parts.

- The `%MktEx` macro is used to generate an experimental design.
- Holdout observations are generated.
- The descriptions of the products are printed for data collection.
- The data are collected, entered, and processed.
- The metric conjoint analysis is performed.
- Results are summarized across subjects.

Creating a Design with the %MktEx Macro

The first thing you need to do in a conjoint study is decide on the product attributes and levels. Then you create the experimental design. We will use the same experimental design as we used in the previous example. The attributes and levels are shown in the table.

Factor	Levels		
Main Ingredient	Chicken	Beef	Turkey
Fat Claim Per Serving	8 Grams	5 Grams	2 Grams
Price	$2.59	$2.29	$1.99
Calories	350	250	

We will create our designs in the same way as we did in the previous example, starting on page 498. Only the random number seed has changed. Like before, we use the `%MktEval` macro to check the one-way and two-way frequencies and to ensure that each combination only appears once. See page 597 for macro documentation and information on installing and using SAS autocall macros.

```
title 'Frozen Diet Entrees';

proc format;
   value if  1='Chicken'  2='Beef'     3='Turkey';
   value ff  1='8 Grams'  2='5 Grams'  3='2 Grams';
   value pf  1='$2.59'    2='$2.29'    3='$1.99';
   value cf  1='350'      2='250';
   run;
%mktex(3 3 3 2, n=18, seed=205)
%mktlab(vars=Ingredient Fat Price Calories)
%mkteval;
```

Frozen Diet Entrees

Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
1	Start	100.0000	100.0000	Tab
1	End	100.0000		

Frozen Diet Entrees

The OPTEX Procedure

Class Level Information

Class	Levels	-Values-
x1	3	1 2 3
x2	3	1 2 3
x3	3	1 2 3
x4	2	1 2

Frozen Diet Entrees

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	100.0000	100.0000	100.0000	0.6667

Frozen Diet Entrees

Canonical Correlations Between the Factors

There are 0 Canonical Correlations Greater Than 0.316

	Ingredient	Fat	Price	Calories
Ingredient	1	0	0	0
Fat	0	1	0	0
Price	0	0	1	0
Calories	0	0	0	1

```
                    Frozen Diet Entrees
                  Summary of Frequencies
There are 0 Canonical Correlations Greater Than 0.316

                          Frequencies

Ingredient                6 6 6
Fat                       6 6 6
Price                     6 6 6
Calories                  9 9
Ingredient Fat            2 2 2 2 2 2 2 2 2
Ingredient Price          2 2 2 2 2 2 2 2 2
Ingredient Calories       3 3 3 3 3 3
Fat Price                 2 2 2 2 2 2 2 2 2
Fat Calories              3 3 3 3 3 3
Price Calories            3 3 3 3 3 3
N-Way                     1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

This design is 100% efficient, perfectly balanced and orthogonal, and each product occurs exactly once.

Designing Holdouts

The next steps add *holdout* observations to the design and display the results. Holdouts are ranked by the subjects but are analyzed with zero weight to exclude them from contributing to the utility computations. The correlation between the ranks for holdouts and their predicted utilities provide an indication of the validity of the results of the study.

The first `%MktEx` step recreates the formats and the design (just so you can see all of the code for a design with holdouts in one step). The next `%MktEx` step adds four holdouts to the randomized design created from the previous step. The specification `options=nodups` (no duplicates) ensures that the holdouts do not match products already in the design. The first `%MktEval` step evaluates just the original design, excluding the holdouts. The second `%MktEval` step evaluates the entire design. Both `%MktEval` steps ensure that the variable `w`, which flags the active and holdout observations, is excluded and not treated as a factor. The `%MktLab` step gives the factors informative names and assigns formats. Unlike the previous examples, this time we directly assign the formats in the `%MktLab` macro using the `statements=` option, specifying a complete `format` statement.

```
title 'Frozen Diet Entrees';

proc format;
   value if  1='Chicken'  2='Beef'     3='Turkey';
   value ff  1='8 Grams'  2='5 Grams'  3='2 Grams';
   value pf  1='$2.59'    2='$2.29'    3='$1.99';
   value cf  1='350'      2='250';
   run;
%mktex(3 3 3 2, n=18, seed=205)

%mktex(3 3 3 2, n=22, init=randomized, holdouts=4, options=nodups, seed=368)
```

```
proc print data=randomized; run;

%mkteval(data=randomized(where=(w=1)), factors=x:);
%mkteval(data=randomized(drop=w));

%mktlab(data=randomized, out=sasuser.dietdes,
        vars=Ingredient Fat Price Calories,
        statements=format Ingredient if. fat ff. price pf. calories cf.)

proc print; run;
```

Here is the last part of the output from the first `%MktEx` step, which shows that the macro found a 100% efficient design.

Frozen Diet Entrees

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	100.0000	100.0000	100.0000	0.6667

Next, is some of the output from the `%MktEx` step that finds the holdouts. Notice that the macro immediately enters the design refinement step.

Frozen Diet Entrees

Design Refinement History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
0	Initial	98.0764		Ini
1	Start	98.0764		Pre,Mut,Ann
1	22 1	98.2421	98.2421	Conforms
1	End	98.2421		
2	Start	98.2421		Pre,Mut,Ann
2	2 1	98.2421	98.2421	Conforms
2	End	98.2421		
3	Start	98.2421		Pre,Mut,Ann
3	2 1	98.2421	98.2421	Conforms
3	End	98.2421		

```
4      Start       98.2421                Pre,Mut,Ann
4      2   1       98.2421       98.2421  Conforms
4        End       98.2421

5      Start       98.2421                Pre,Mut,Ann
5      2   1       98.2421       98.2421  Conforms
5        End       98.2421

NOTE: Stopping since it appears that no improvement is possible.
```

Next, the raw design is printed. Observations with `w` equal to 1 comprise the original design. The observations with a missing `w` are the holdouts.

Frozen Diet Entrees

Obs	x1	x2	x3	x4	w
1	2	3	1	2	.
2	2	2	1	2	1
3	3	3	3	1	1
4	3	3	3	2	1
5	3	1	1	1	1
6	1	3	1	1	1
7	1	3	1	2	1
8	1	1	2	1	1
9	2	2	1	1	1
10	3	2	2	2	1
11	2	2	3	1	.
12	2	3	2	2	1
13	3	1	1	2	1
14	2	1	3	2	1
15	3	2	1	1	.
16	1	2	3	1	1
17	1	1	2	2	1
18	1	2	2	2	.
19	2	3	2	1	1
20	3	2	2	1	1
21	1	2	3	2	1
22	2	1	3	1	1

Here is the evaluation of the original design.

Frozen Diet Entrees
Canonical Correlations Between the Factors
There are 0 Canonical Correlations Greater Than 0.316

	x1	x2	x3	x4
x1	1	0	0	0
x2	0	1	0	0
x3	0	0	1	0
x4	0	0	0	1

Frozen Diet Entrees
Summary of Frequencies
There are 0 Canonical Correlations Greater Than 0.316

	Frequencies
x1	6 6 6
x2	6 6 6
x3	6 6 6
x4	9 9
x1 x2	2 2 2 2 2 2 2 2 2
x1 x3	2 2 2 2 2 2 2 2 2
x1 x4	3 3 3 3 3 3
x2 x3	2 2 2 2 2 2 2 2 2
x2 x4	3 3 3 3 3 3
x3 x4	3 3 3 3 3 3
N-Way	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Here is an evaluation of the design with the holdouts.

Frozen Diet Entrees
Canonical Correlations Between the Factors
There are 0 Canonical Correlations Greater Than 0.316

	x1	x2	x3	x4
x1	1	0.09	0.17	0.11
x2	0.09	1	0.09	0.11
x3	0.17	0.09	1	0.11
x4	0.11	0.11	0.11	1

Frozen Diet Entrees
Summary of Frequencies
There are 0 Canonical Correlations Greater Than 0.316
* - Indicates Unequal Frequencies

Frequencies

*	x1	7 8 7
*	x2	6 9 7
*	x3	8 7 7
	x4	11 11
*	x1 x2	2 3 2 2 3 3 2 3 2
*	x1 x3	2 3 2 3 2 3 3 2 2
*	x1 x4	3 4 4 4 4 3
*	x2 x3	2 2 2 3 3 3 3 2 2
*	x2 x4	3 3 5 4 3 4
*	x3 x4	4 4 3 4 4 3
	N-Way	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Here is the design, printed with descriptive factor names and formats.

Frozen Diet Entrees

Obs	Ingredient	Fat	Price	Calories	w
1	Beef	2 Grams	$2.59	250	.
2	Beef	5 Grams	$2.59	250	1
3	Turkey	2 Grams	$1.99	350	1
4	Turkey	2 Grams	$1.99	250	1
5	Turkey	8 Grams	$2.59	350	1
6	Chicken	2 Grams	$2.59	350	1
7	Chicken	2 Grams	$2.59	250	1
8	Chicken	8 Grams	$2.29	350	1
9	Beef	5 Grams	$2.59	350	1
10	Turkey	5 Grams	$2.29	250	1
11	Beef	5 Grams	$1.99	350	.
12	Beef	2 Grams	$2.29	250	1
13	Turkey	8 Grams	$2.59	250	1
14	Beef	8 Grams	$1.99	250	1
15	Turkey	5 Grams	$2.59	350	.

```
16     Chicken     5 Grams    $1.99      350       1
17     Chicken     8 Grams    $2.29      250       1
18     Chicken     5 Grams    $2.29      250       .
19     Beef        2 Grams    $2.29      350       1
20     Turkey      5 Grams    $2.29      350       1
21     Chicken     5 Grams    $1.99      250       1
22     Beef        8 Grams    $1.99      350       1
```

Print the Stimuli

Once the design is generated, the *stimuli* (descriptions of the combinations) must be generated for data collection. They are printed using the exact same step as we used on page 502.

```
title;
data _null_;
   file print;
   set sasuser.dietdes;
   put ///
       +3 ingredient 'Entree' @50 '(' _n_ +(-1) ')' /
       +3 'With ' fat 'of Fat and ' calories  'Calories' /
       +3 'Now for Only ' Price +(-1) '.'///;
   if mod(_n_, 6) = 0 then put _page_;
   run;
```

In the interest of space, only the first three are shown.

```
Beef Entree                                (1)
With 2 Grams of Fat and 250 Calories
Now for Only $2.59.

Beef Entree                                (2)
With 5 Grams of Fat and 250 Calories
Now for Only $2.59.

Turkey Entree                              (3)
With 2 Grams of Fat and 350 Calories
Now for Only $1.99.
```

Data Collection, Entry, and Preprocessing

The next step in the conjoint analysis study is data collection and entry. Each subject was asked to take the 22 cards and rank them from the most preferred combination to the least preferred combination. The combination numbers are entered as data. The data follow the `datalines` statement in the next DATA step. For the first subject, 4 was most preferred, 3 was second most preferred, ..., and 5 was the least preferred combination. The DATA step validates the data entry and converts the input to ranks.

```
title 'Frozen Diet Entrees';

%let m = 22; /* number of combinations */

* Read the input data and convert to ranks;
data ranks(drop=i k c1-c&m);
   input c1-c&m;
   array c[&m];
   array r[&m];
   do i = 1 to &m;
      k = c[i];
      if 1 le k le &m then do;
         if r[k] ne . then
            put 'ERROR: For subject ' _n_ +(-1) ', combination ' k
                'is given more than once.';
         r[k] = i; /* Convert to ranks. */
         end;
      else put 'ERROR: For subject ' _n_ +(-1) ', combination ' k
                'is invalid.';
      end;
   do i = 1 to &m;
      if r[i] = . then
         put 'ERROR: For subject ' _n_ +(-1) ', combination ' i
             'is not given.';
      end;
   name = 'Subj' || put(_n_, z2.);
   datalines;
4 3 7 21 12 10 6 19 1 16 18 11 20 14 17 15 2 22 9 8 13 5
4 12 3 1 19 7 10 6 11 21 16 2 18 20 15 9 14 22 13 17 5 8
4 3 7 12 19 21 1 6 10 18 16 11 20 15 2 14 9 17 22 8 13 5
4 12 1 10 21 14 18 3 7 2 17 13 19 11 22 20 16 15 6 9 5 8
4 21 14 11 16 3 12 22 19 18 10 17 8 20 7 1 6 2 9 13 15 5
4 21 16 12 3 14 11 22 18 19 7 10 1 17 8 6 2 20 9 13 15 5
12 4 19 1 3 7 6 21 18 11 16 2 10 20 9 15 14 17 22 8 13 5
4 21 3 16 14 11 12 22 18 10 19 20 17 8 7 6 1 2 13 15 9 5
4 21 3 16 11 14 22 12 18 10 20 19 17 8 7 6 1 13 15 2 9 5
4 3 14 11 21 12 16 22 19 10 18 20 17 1 7 8 2 13 9 6 15 5
15 22 17 21 6 11 13 19 4 12 3 18 9 7 1 10 8 20 14 16 5 2
12 4 3 7 21 19 1 18 11 6 16 2 14 10 17 22 20 9 15 8 13 5
;
```

The macro variable `&m` is set to 22, the number of combinations. This is done to make it easier to modify the code for future use with different sized studies. For each subject, the numbers of the 22 products are read into the variables `c1` through `c22`. The do loop, `do i = 1 to &m`, loops over each of the products. Consider the first product: `k` is set to `c[i]`, which is `c[1]`, which is 4 since the fourth product was ranked first by the first subject. The first data integrity check, `if 1 le k le &m then do` ensures that the number is in the valid range, 1 to 22. Otherwise an error is printed. Since the number is valid, `r[k]` is checked to see if it is missing. If it is not missing, another error is printed. The array `r` consists of 22 variables `r1` through `22`. These variables start out each pass through the DATA step as missing and end up as the ranks. If `r[k] eq .`, then the *kth* combination has not had

a rank assigned yet so everything is fine. If `r[k] ne .`, the same number appears twice in a subject's data so there is something wrong with the data entry. The statement `r[k] = i` assigns the ranks. For subject 1 and the first product, `k = c[i] = c[1] = 4` so the rank of the fourth product is set to 1 (`r[k] = r[4] = i = 1`). For subject 1 and the second product, `k = c[i] = c[2] = 3` so the rank of the third product is set to 2 (`r[k] = r[3] = i = 2`). For subject 1 and the last product, `k = c[i] = c[22] = 5` so the rank of the fifth product is set to 22 (`r[k] = r[5] = i = 22`). At the end of the `do i = 1 to &m` loop, each of the 22 variables in `r1-r22` should have been set to exactly one rank. If any of these variables are missing, then one or more product numbers did not appear in the data, so this is flagged as an error. The statement `name = 'Subj' || put(_n_, z2.)` creates a subject ID of the form `Subj01`, `Subj02`, ..., `Subj12`.

Say there was a mistake in data entry for the first subject−say product 17 had been entered as 7 instead of 17. We would get the following error messages.

```
ERROR: For subject 1, combination 7 is given more than once.
ERROR: For subject 1, combination 17 is not given.
```

If for the first subject, the 17 had been entered as 117 instead of 17, we would get the following error messages.

```
ERROR: For subject 1, combination 117 is invalid.
ERROR: For subject 1, combination 17 is not given.
```

The next step transposes the data set from one row per subject to one row per product. The `id name` statement on PROC TRANSPOSE names the rank variables `Subj01` through `Subj12`. Later, we will need to sort by these names. That is why we used leading zeros and names like `Subj01` instead of `Subj1`. Next, the input data set is merged with the design.

```
proc transpose data=ranks out=ranks2;
   id name;
   run;
data both;
   merge sasuser.dietdes ranks2;
   drop _name_;
   run;
proc print label;
   title2 'Data and Design Together';
   run;
```

Frozen Diet Entrees
Data and Design Together

Obs	Ingredient	Fat	Price	Calories	w	Subj01	Subj02	Subj03	Subj04
1	Beef	2 Grams	$2.59	250	.	9	4	7	3
2	Beef	5 Grams	$2.59	250	1	17	12	15	10
3	Turkey	2 Grams	$1.99	350	1	2	3	2	8
4	Turkey	2 Grams	$1.99	250	1	1	1	1	1
5	Turkey	8 Grams	$2.59	350	1	22	21	22	21
6	Chicken	2 Grams	$2.59	350	1	7	8	8	19

7	Chicken	2 Grams	$2.59	250	1	3	6	3	9
8	Chicken	8 Grams	$2.29	350	1	20	22	20	22
9	Beef	5 Grams	$2.59	350	1	19	16	17	20
10	Turkey	5 Grams	$2.29	250	1	6	7	9	4
11	Beef	5 Grams	$1.99	350	.	12	9	12	14
12	Beef	2 Grams	$2.29	250	1	5	2	4	2
13	Turkey	8 Grams	$2.59	250	1	21	19	21	12
14	Beef	8 Grams	$1.99	250	1	14	17	16	6
15	Turkey	5 Grams	$2.59	350	.	16	15	14	18
16	Chicken	5 Grams	$1.99	350	1	10	11	11	17
17	Chicken	8 Grams	$2.29	250	1	15	20	18	11
18	Chicken	5 Grams	$2.29	250	.	11	13	10	7
19	Beef	2 Grams	$2.29	350	1	8	5	5	13
20	Turkey	5 Grams	$2.29	350	1	13	14	13	16
21	Chicken	5 Grams	$1.99	250	1	4	10	6	5
22	Beef	8 Grams	$1.99	350	1	18	18	19	15

Obs	Subj05	Subj06	Subj07	Subj08	Subj09	Subj10	Subj11	Subj12
1	16	13	4	17	17	14	15	7
2	18	17	12	18	20	17	22	12
3	6	5	5	3	3	2	11	3
4	1	1	2	1	1	1	9	2
5	22	22	22	22	22	22	21	22
6	17	16	7	16	16	20	5	10
7	15	11	6	15	15	15	14	4
8	13	15	20	14	14	16	17	20
9	19	19	15	21	21	19	13	18
10	11	12	13	10	10	10	16	14
11	4	7	10	6	5	4	6	9
12	7	4	1	7	8	6	10	1
13	20	20	21	19	18	18	7	21
14	3	6	17	5	6	3	19	13
15	21	21	16	20	19	21	1	19
16	5	3	11	4	4	7	20	11
17	12	14	18	13	13	13	3	15
18	10	9	9	9	9	11	12	8
19	9	10	3	11	12	9	8	6
20	14	18	14	12	11	12	18	17
21	2	2	8	2	2	5	4	5
22	8	8	19	8	7	8	2	16

One more data set manipulation is sometimes necessary—the addition of *simulation* observations. Simulation observations are not rated by the subjects and do not contribute to the analysis. They are scored as passive observations. Simulations are *what-if* combinations. They are combinations that are entered to get a prediction of what their utility would have been if they had been rated. In this example, all combinations are added as simulations. The `%MktEx` macro is called to make a full-factorial design. The `n=` specification accepts expressions, so `n=3*3*3*2` and `n=54` are equivalent. The `data all` step reads in the design and data followed by the simulation observations. The flag variable `f`

indicates when the simulation observations are being processed. Simulation observations are given a weight of 0 to exclude them from the analysis and to distinguish them from the holdouts. Notice that the dependent variable has missing values for the simulations and nonmissing values for the holdouts and active observations.

```
proc format;
   value wf 1 = 'Active'
            . = 'Holdout'
            0 = 'Simulation';
   run;
%mktex(3 3 3 2, n=3*3*3*2)
%mktlab(data=design, vars=Ingredient Fat Price Calories)

data all;
   set both final(in=f);
   if f then w = 0;
   format w wf.;
   run;
proc print data=all(Obs=25 drop=subj04-subj12) label;
   title2 'Some of the Final Data Set';
   run;
```

Here the data for the first three subjects and the first 25 rows of the data set.

Frozen Diet Entrees
Some of the Final Data Set

Obs	Ingredient	Fat	Price	Calories	w	Subj01	Subj02	Subj03
1	Beef	2 Grams	$2.59	250	Holdout	9	4	7
2	Beef	5 Grams	$2.59	250	Active	17	12	15
3	Turkey	2 Grams	$1.99	350	Active	2	3	2
4	Turkey	2 Grams	$1.99	250	Active	1	1	1
5	Turkey	8 Grams	$2.59	350	Active	22	21	22
6	Chicken	2 Grams	$2.59	350	Active	7	8	8
7	Chicken	2 Grams	$2.59	250	Active	3	6	3
8	Chicken	8 Grams	$2.29	350	Active	20	22	20
9	Beef	5 Grams	$2.59	350	Active	19	16	17
10	Turkey	5 Grams	$2.29	250	Active	6	7	9
11	Beef	5 Grams	$1.99	350	Holdout	12	9	12
12	Beef	2 Grams	$2.29	250	Active	5	2	4
13	Turkey	8 Grams	$2.59	250	Active	21	19	21
14	Beef	8 Grams	$1.99	250	Active	14	17	16
15	Turkey	5 Grams	$2.59	350	Holdout	16	15	14
16	Chicken	5 Grams	$1.99	350	Active	10	11	11
17	Chicken	8 Grams	$2.29	250	Active	15	20	18
18	Chicken	5 Grams	$2.29	250	Holdout	11	13	10

19	Beef	2 Grams	$2.29	350	Active	8	5	5
20	Turkey	5 Grams	$2.29	350	Active	13	14	13
21	Chicken	5 Grams	$1.99	250	Active	4	10	6
22	Beef	8 Grams	$1.99	350	Active	18	18	19
23	Chicken	8 Grams	$2.59	350	Simulation	.	.	.
24	Chicken	8 Grams	$2.59	350	Simulation	.	.	.
25	Chicken	8 Grams	$2.59	350	Simulation	.	.	.

Metric Conjoint Analysis

In this part of this example, the conjoint analysis is performed with PROC TRANSREG.

```
ods exclude notes mvanova anova;
proc transreg data=all utilities short separators=', '
   method=morals outtest=utils;
   title2 'Conjoint Analysis';
   model identity(subj: / reflect) =
         class(Ingredient Fat Price Calories / zero=sum);
   weight w;
   output p ireplace out=results coefficients;
   run;
```

The `proc`, `model`, and `output` statements are typical for a conjoint analysis of rank-order data with more than one subject. (In this analysis, we perform a metric conjoint analysis. It is more typical to perform nonmetric conjoint analysis of rank-order data. However, it is not absolutely required.) The `proc` statement specifies `method=morals`, which fits the conjoint analysis model separately for each subject. The `proc` statement also requests an `outtest=` data set, which contains the ANOVA and part-worth utilities tables from the printed output. In the `model` statement, the dependent variable list `subj:` specifies all variables in the DATA= data set that begin with the prefix `subj` (in this case `subj01-subj12`). The `weight` variable designates the active (`weight` = 1), holdout (`weight` = .), and simulation (`weight` = 0) observations. Only the active observations are used to compute the part-worth utilities. However, predicted utilities are computed for all observations, including active, holdouts, and simulations, using those part-worths. The `output` statement creates an `out=` data set beginning with all results for the first subject, followed by all subject two results, and so on.

Here are the results. Recall that we used an `ods exclude` statement and we used PROC TEMPLATE on page 485 to customize the output from PROC TRANSREG. There is one set of output for each subject. Conjoint analysis fits individual-level models.

Frozen Diet Entrees
Conjoint Analysis

The TRANSREG Procedure

Class Level Information

Class	Levels	Values
Ingredient	3	Chicken Beef Turkey

Fat	3	8 Grams 5 Grams 2 Grams
Price	3	$2.59 $2.29 $1.99
Calories	2	350 250

Number of Observations Read	76
Number of Observations Used	18
Sum of Weights Read	18
Sum of Weights Used	18

Frozen Diet Entrees
Conjoint Analysis

The TRANSREG Procedure

Identity(Subj01)
Algorithm converged.

The TRANSREG Procedure Hypothesis Tests for Identity(Subj01)

Root MSE	1.81046	R-Square	0.9618
Dependent Mean	11.38889	Adj R-Sq	0.9351
Coeff Var	15.89675		

Part-Worth Utilities

Label	Utility	Standard Error	Importance (% Utility Range)
Intercept	11.3889	0.42673	
Ingredient, Chicken	1.5556	0.60349	13.095
Ingredient, Beef	-2.1111	0.60349	
Ingredient, Turkey	0.5556	0.60349	
Fat, 8 Grams	-6.9444	0.60349	50.000
Fat, 5 Grams	-0.1111	0.60349	
Fat, 2 Grams	7.0556	0.60349	
Price, $2.59	-3.4444	0.60349	23.810
Price, $2.29	0.2222	0.60349	
Price, $1.99	3.2222	0.60349	
Calories, 350	-1.8333	0.42673	13.095
Calories, 250	1.8333	0.42673	

Frozen Diet Entrees
Conjoint Analysis

The TRANSREG Procedure

Identity(Subj02)
Algorithm converged.

The TRANSREG Procedure Hypothesis Tests for Identity(Subj02)

Root MSE	1.30809	R-Square	0.9788
Dependent Mean	11.77778	Adj R-Sq	0.9640
Coeff Var	11.10646		

Part-Worth Utilities

Label	Utility	Standard Error	Importance (% Utility Range)
Intercept	11.7778	0.30832	
Ingredient, Chicken	-1.0556	0.43603	8.451
Ingredient, Beef	0.1111	0.43603	
Ingredient, Turkey	0.9444	0.43603	
Fat, 8 Grams	-7.7222	0.43603	64.789
Fat, 5 Grams	0.1111	0.43603	
Fat, 2 Grams	7.6111	0.43603	
Price, $2.59	-1.8889	0.43603	15.493
Price, $2.29	0.1111	0.43603	
Price, $1.99	1.7778	0.43603	
Calories, 350	-1.3333	0.30832	11.268
Calories, 250	1.3333	0.30832	

Frozen Diet Entrees
Conjoint Analysis

The TRANSREG Procedure

Identity(Subj03)
Algorithm converged.

The TRANSREG Procedure Hypothesis Tests for Identity(Subj03)

Root MSE	1.15470	R-Square	0.9844
Dependent Mean	11.66667	Adj R-Sq	0.9735
Coeff Var	9.89743		

Part-Worth Utilities

Label	Utility	Standard Error	Importance (% Utility Range)
Intercept	11.6667	0.27217	
Ingredient, Chicken	0.6667	0.38490	6.667
Ingredient, Beef	-1.0000	0.38490	
Ingredient, Turkey	0.3333	0.38490	
Fat, 8 Grams	-7.6667	0.38490	62.000
Fat, 5 Grams	-0.1667	0.38490	
Fat, 2 Grams	7.8333	0.38490	
Price, $2.59	-2.6667	0.38490	20.667
Price, $2.29	0.1667	0.38490	
Price, $1.99	2.5000	0.38490	
Calories, 350	-1.3333	0.27217	10.667
Calories, 250	1.3333	0.27217	

Frozen Diet Entrees
Conjoint Analysis

The TRANSREG Procedure

Identity(Subj04)
Algorithm converged.

The TRANSREG Procedure Hypothesis Tests for Identity(Subj04)

Root MSE	1.05935	R-Square	0.9849
Dependent Mean	11.72222	Adj R-Sq	0.9743
Coeff Var	9.03711		

Part-Worth Utilities

Label	Utility	Standard Error	Importance (% Utility Range)
Intercept	11.7222	0.24969	
Ingredient, Chicken	-2.1111	0.35312	13.490
Ingredient, Beef	0.7222	0.35312	
Ingredient, Turkey	1.3889	0.35312	
Fat, 8 Grams	-2.7778	0.35312	22.484
Fat, 5 Grams	-0.2778	0.35312	
Fat, 2 Grams	3.0556	0.35312	
Price, $2.59	-3.4444	0.35312	25.054
Price, $2.29	0.3889	0.35312	
Price, $1.99	3.0556	0.35312	
Calories, 350	-5.0556	0.24969	38.972
Calories, 250	5.0556	0.24969	

Frozen Diet Entrees
Conjoint Analysis

The TRANSREG Procedure

Identity(Subj05)
Algorithm converged.

The TRANSREG Procedure Hypothesis Tests for Identity(Subj05)

Root MSE	1.02198	R-Square	0.9854
Dependent Mean	11.22222	Adj R-Sq	0.9752
Coeff Var	9.10676		

Part-Worth Utilities

Label	Utility	Standard Error	Importance (% Utility Range)
Intercept	11.2222	0.24088	
Ingredient, Chicken	0.5556	0.34066	7.407
Ingredient, Beef	0.5556	0.34066	
Ingredient, Turkey	-1.1111	0.34066	
Fat, 8 Grams	-1.7778	0.34066	17.037
Fat, 5 Grams	-0.2778	0.34066	
Fat, 2 Grams	2.0556	0.34066	
Price, $2.59	-7.2778	0.34066	63.704
Price, $2.29	0.2222	0.34066	
Price, $1.99	7.0556	0.34066	
Calories, 350	-1.3333	0.24088	11.852
Calories, 250	1.3333	0.24088	

Frozen Diet Entrees
Conjoint Analysis

The TRANSREG Procedure

Identity(Subj06)
Algorithm converged.

The TRANSREG Procedure Hypothesis Tests for Identity(Subj06)

Root MSE	1.67000	R-Square	0.9636
Dependent Mean	11.27778	Adj R-Sq	0.9381
Coeff Var	14.80785		

Part-Worth Utilities

Label	Utility	Standard Error	Importance (% Utility Range)
Intercept	11.2778	0.39362	
Ingredient, Chicken	1.1111	0.55667	11.015
Ingredient, Beef	0.6111	0.55667	
Ingredient, Turkey	-1.7222	0.55667	
Fat, 8 Grams	-2.8889	0.55667	24.622
Fat, 5 Grams	-0.5556	0.55667	
Fat, 2 Grams	3.4444	0.55667	
Price, $2.59	-6.2222	0.55667	51.836
Price, $2.29	-0.8889	0.55667	
Price, $1.99	7.1111	0.55667	
Calories, 350	-1.6111	0.39362	12.527
Calories, 250	1.6111	0.39362	

Frozen Diet Entrees
Conjoint Analysis

The TRANSREG Procedure

Identity(Subj07)
Algorithm converged.

The TRANSREG Procedure Hypothesis Tests for Identity(Subj07)

Root MSE	1.06979	R-Square	0.9857
Dependent Mean	11.88889	Adj R-Sq	0.9756
Coeff Var	8.99821		

Part-Worth Utilities

Label	Utility	Standard Error	Importance (% Utility Range)
Intercept	11.8889	0.25215	
Ingredient, Chicken	0.2222	0.35660	7.353
Ingredient, Beef	0.7222	0.35660	
Ingredient, Turkey	-0.9444	0.35660	
Fat, 8 Grams	-7.6111	0.35660	68.382
Fat, 5 Grams	-0.2778	0.35660	
Fat, 2 Grams	7.8889	0.35660	
Price, $2.59	-1.9444	0.35660	15.441
Price, $2.29	0.3889	0.35660	
Price, $1.99	1.5556	0.35660	
Calories, 350	-1.0000	0.25215	8.824
Calories, 250	1.0000	0.25215	

Frozen Diet Entrees
Conjoint Analysis

The TRANSREG Procedure

Identity(Subj08)
Algorithm converged.

The TRANSREG Procedure Hypothesis Tests for Identity(Subj08)

Root MSE	0.79582	R-Square	0.9915
Dependent Mean	11.16667	Adj R-Sq	0.9855
Coeff Var	7.12677		

Part-Worth Utilities

Label	Utility	Standard Error	Importance (% Utility Range)
Intercept	11.1667	0.18758	
Ingredient, Chicken	0.5000	0.26527	4.412
Ingredient, Beef	-0.5000	0.26527	
Ingredient, Turkey	0.0000	0.26527	
Fat, 8 Grams	-2.3333	0.26527	20.588
Fat, 5 Grams	0.0000	0.26527	
Fat, 2 Grams	2.3333	0.26527	
Price, $2.59	-7.3333	0.26527	64.706
Price, $2.29	-0.0000	0.26527	
Price, $1.99	7.3333	0.26527	
Calories, 350	-1.1667	0.18758	10.294
Calories, 250	1.1667	0.18758	

Frozen Diet Entrees
Conjoint Analysis

The TRANSREG Procedure

Identity(Subj09)
Algorithm converged.

The TRANSREG Procedure Hypothesis Tests for Identity(Subj09)

Root MSE	1.05935	R-Square	0.9850
Dependent Mean	11.27778	Adj R-Sq	0.9745
Coeff Var	9.39325		

Part-Worth Utilities

Label	Utility	Standard Error	Importance (% Utility Range)
Intercept	11.2778	0.24969	
Ingredient, Chicken	0.6111	0.35312	7.389
Ingredient, Beef	-1.0556	0.35312	
Ingredient, Turkey	0.4444	0.35312	
Fat, 8 Grams	-2.0556	0.35312	18.473
Fat, 5 Grams	-0.0556	0.35312	
Fat, 2 Grams	2.1111	0.35312	
Price, $2.59	-7.3889	0.35312	65.764
Price, $2.29	-0.0556	0.35312	
Price, $1.99	7.4444	0.35312	
Calories, 350	-0.9444	0.24969	8.374
Calories, 250	0.9444	0.24969	

Frozen Diet Entrees
Conjoint Analysis

The TRANSREG Procedure

Identity(Subj10)
Algorithm converged.

The TRANSREG Procedure Hypothesis Tests for Identity(Subj10)

Root MSE	0.90062	R-Square	0.9889
Dependent Mean	11.27778	Adj R-Sq	0.9812
Coeff Var	7.98577		

Part-Worth Utilities

Label	Utility	Standard Error	Importance (% Utility Range)
Intercept	11.2778	0.21228	
Ingredient, Chicken	-1.3889	0.30021	9.722
Ingredient, Beef	0.9444	0.30021	
Ingredient, Turkey	0.4444	0.30021	
Fat, 8 Grams	-2.0556	0.30021	18.750
Fat, 5 Grams	-0.3889	0.30021	
Fat, 2 Grams	2.4444	0.30021	
Price, $2.59	-7.2222	0.30021	59.028
Price, $2.29	0.2778	0.30021	
Price, $1.99	6.9444	0.30021	
Calories, 350	-1.5000	0.21228	12.500
Calories, 250	1.5000	0.21228	

Frozen Diet Entrees
Conjoint Analysis

The TRANSREG Procedure

Identity(Subj11)
Algorithm converged.

The TRANSREG Procedure Hypothesis Tests for Identity(Subj11)

Root MSE	7.42369	R-Square	0.2393
Dependent Mean	12.16667	Adj R-Sq	-0.2932
Coeff Var	61.01660		

Part-Worth Utilities

Label	Utility	Standard Error	Importance (% Utility Range)
Intercept	12.1667	1.74978	
Ingredient, Chicken	1.6667	2.47456	23.950
Ingredient, Beef	-0.1667	2.47456	
Ingredient, Turkey	-1.5000	2.47456	
Fat, 8 Grams	0.6667	2.47456	45.378
Fat, 5 Grams	-3.3333	2.47456	
Fat, 2 Grams	2.6667	2.47456	
Price, $2.59	-1.5000	2.47456	21.429
Price, $2.29	0.1667	2.47456	
Price, $1.99	1.3333	2.47456	
Calories, 350	-0.6111	1.74978	9.244
Calories, 250	0.6111	1.74978	

Frozen Diet Entrees
Conjoint Analysis

The TRANSREG Procedure

Identity(Subj12)
Algorithm converged.

The TRANSREG Procedure Hypothesis Tests for Identity(Subj12)

Root MSE	1.49443	R-Square	0.9717
Dependent Mean	11.66667	Adj R-Sq	0.9519
Coeff Var	12.80944		

Part-Worth Utilities

Label	Utility	Standard Error	Importance (% Utility Range)
Intercept	11.6667	0.35224	
Ingredient, Chicken	0.8333	0.49814	8.974
Ingredient, Beef	0.6667	0.49814	
Ingredient, Turkey	-1.5000	0.49814	
Fat, 8 Grams	-6.1667	0.49814	51.923
Fat, 5 Grams	-1.1667	0.49814	
Fat, 2 Grams	7.3333	0.49814	
Price, $2.59	-2.8333	0.49814	23.718
Price, $2.29	-0.5000	0.49814	
Price, $1.99	3.3333	0.49814	
Calories, 350	-2.0000	0.35224	15.385
Calories, 250	2.0000	0.35224	

Next, we will print some of the output data set to see the predicted utilities for the first two subjects.

```
proc print data=results(drop=_depend_ t_depend_ intercept &_trgind) label;
   title2 'Predicted Utility';
   where w ne 0 and _depvar_ le 'Identity(Subj02)' and not (_type_ =: 'M');
   by _depvar_;
   label p_depend_ = 'Predicted Utility';
   run;
```

We print _TYPE_, _NAME_, and the weight variable, w; drop the original and transformed dependent variable, _depend_ and t_depend_; print the predicted values (predicted utilities), p_depend_; drop the intercept and coded independent variables; and print the original class variables. Note that the macro variable &_trgind is automatically created by PROC TRANSREG and its value is a list of the names of the coded variables. The where statement is used to exclude the simulation observations and just show results for the first two subjects. Here are the predicted utilities for each of the rated products for the first two subjects.

Frozen Diet Entrees
Predicted Utility

---------- Dependent Variable Transformation(Name)=Identity(Subj01) ----------

Obs	_TYPE_	_NAME_	w	Predicted Utility	Ingredient	Fat	Price	Calories
1		ROW1	Holdout	14.7222	Beef	2 Grams	$2.59	250
2	SCORE	ROW2	Active	7.5556	Beef	5 Grams	$2.59	250
3	SCORE	ROW3	Active	20.3889	Turkey	2 Grams	$1.99	350
4	SCORE	ROW4	Active	24.0556	Turkey	2 Grams	$1.99	250
5	SCORE	ROW5	Active	-0.2778	Turkey	8 Grams	$2.59	350
6	SCORE	ROW6	Active	14.7222	Chicken	2 Grams	$2.59	350
7	SCORE	ROW7	Active	18.3889	Chicken	2 Grams	$2.59	250
8	SCORE	ROW8	Active	4.3889	Chicken	8 Grams	$2.29	350
9	SCORE	ROW9	Active	3.8889	Beef	5 Grams	$2.59	350
10	SCORE	ROW10	Active	13.8889	Turkey	5 Grams	$2.29	250
11		ROW11	Holdout	10.5556	Beef	5 Grams	$1.99	350
12	SCORE	ROW12	Active	18.3889	Beef	2 Grams	$2.29	250
13	SCORE	ROW13	Active	3.3889	Turkey	8 Grams	$2.59	250
14	SCORE	ROW14	Active	7.3889	Beef	8 Grams	$1.99	250
15		ROW15	Holdout	6.5556	Turkey	5 Grams	$2.59	350
16	SCORE	ROW16	Active	14.2222	Chicken	5 Grams	$1.99	350
17	SCORE	ROW17	Active	8.0556	Chicken	8 Grams	$2.29	250
18		ROW18	Holdout	14.8889	Chicken	5 Grams	$2.29	250
19	SCORE	ROW19	Active	14.7222	Beef	2 Grams	$2.29	350
20	SCORE	ROW20	Active	10.2222	Turkey	5 Grams	$2.29	350
21	SCORE	ROW21	Active	17.8889	Chicken	5 Grams	$1.99	250
22	SCORE	ROW22	Active	3.7222	Beef	8 Grams	$1.99	350

---------- Dependent Variable Transformation(Name)=Identity(Subj02) ----------

Obs	_TYPE_	_NAME_	w	Predicted Utility	Ingredient	Fat	Price	Calories
79		ROW1	Holdout	18.9444	Beef	2 Grams	$2.59	250
80	SCORE	ROW2	Active	11.4444	Beef	5 Grams	$2.59	250
81	SCORE	ROW3	Active	20.7778	Turkey	2 Grams	$1.99	350
82	SCORE	ROW4	Active	23.4444	Turkey	2 Grams	$1.99	250
83	SCORE	ROW5	Active	1.7778	Turkey	8 Grams	$2.59	350
84	SCORE	ROW6	Active	15.1111	Chicken	2 Grams	$2.59	350
85	SCORE	ROW7	Active	17.7778	Chicken	2 Grams	$2.59	250
86	SCORE	ROW8	Active	1.7778	Chicken	8 Grams	$2.29	350
87	SCORE	ROW9	Active	8.7778	Beef	5 Grams	$2.59	350
88	SCORE	ROW10	Active	14.2778	Turkey	5 Grams	$2.29	250
89		ROW11	Holdout	12.4444	Beef	5 Grams	$1.99	350
90	SCORE	ROW12	Active	20.9444	Beef	2 Grams	$2.29	250
91	SCORE	ROW13	Active	4.4444	Turkey	8 Grams	$2.59	250
92	SCORE	ROW14	Active	7.2778	Beef	8 Grams	$1.99	250
93		ROW15	Holdout	9.6111	Turkey	5 Grams	$2.59	350
94	SCORE	ROW16	Active	11.2778	Chicken	5 Grams	$1.99	350
95	SCORE	ROW17	Active	4.4444	Chicken	8 Grams	$2.29	250
96		ROW18	Holdout	12.2778	Chicken	5 Grams	$2.29	250
97	SCORE	ROW19	Active	18.2778	Beef	2 Grams	$2.29	350
98	SCORE	ROW20	Active	11.6111	Turkey	5 Grams	$2.29	350
99	SCORE	ROW21	Active	13.9444	Chicken	5 Grams	$1.99	250
100	SCORE	ROW22	Active	4.6111	Beef	8 Grams	$1.99	350

Analyzing Holdouts

The next steps display the correlations between the predicted utility for holdout observations and their actual ratings. These correlations provide a measure of the validity of the results, since the holdout observations have zero weight and do not contribute to any of the calculations. The Pearson correlations are the ordinary correlation coefficients, and the Kendall Tau's are rank-based measures of correlation. These correlations should always be large. Subjects whose correlations are small may be unreliable.

PROC CORR is used to produce the correlations. Since the output is not very compact, ODS is used to suppress the normal printed output (`ods listing close`), output the Pearson correlations to an output data set P (`PearsonCorr=p`), and output the Kendall correlations to an output data set K (`KendallCorr=k`). The listing is reopened for normal output (`ods listing`), the two tables are merged renaming the variables to identify the correlation type, the subject number is pulled out of the subject variable names, and the results are printed.

```
ods output KendallCorr=k PearsonCorr=p;
ods listing close;
proc corr nosimple noprob kendall pearson
   data=results(where=(w=.));
   title2 'Holdout Validation Results';
   var p_depend_;
   with t_depend_;
   by notsorted _depvar_;
   run;
ods listing;

data both(keep=subject pearson kendall);
   length Subject 8;
   merge p(rename=(p_depend_=Pearson))
         k(rename=(p_depend_=Kendall));
   subject = input(substr(_depvar_, 14, 2), best2.);
   run;

proc print; run;
```

Here are the results.

Frozen Diet Entrees
Holdout Validation Results

Obs	Subject	Pearson	Kendall
1	1	0.93848	0.66667
2	2	0.94340	1.00000
3	3	0.99038	1.00000
4	4	0.97980	1.00000
5	5	0.98930	1.00000
6	6	0.98649	1.00000
7	7	0.99029	1.00000
8	8	0.99296	1.00000
9	9	0.99873	1.00000
10	10	0.99973	1.00000
11	11	-0.98184	-1.00000
12	12	0.92920	1.00000

Most of the correlations look great! However, the results from subject 11 look suspect. Subject 11's holdout correlations are negative. We can return to page 532 and look at the conjoint results. Subject 11 has an R^2 of 0.2393. In contrast, all of the other subjects have an R^2 over 0.95. Subject 11 almost certainly did not take the task seriously, so his or her results will be discarded.

```
data results2;
   set results;
   if not (index(_depvar_, '11'));
   run;
```

```
data utils2;
   set utils;
   if not (index(_depvar_, '11'));
   run;
```

Simulations

The next steps display simulation observations. The most preferred combinations are printed for each subject.

```
proc sort data=results2(where=(w=0)) out=sims(drop=&_trgind);
   by _depvar_ descending p_depend_;
   run;
data sims; /* Pull out first 10 for each subject. */
   set sims;
   by _depvar_;
   retain n 0;
   if first._depvar_ then n = 0;
   n = n + 1;
   if n le 10;
   drop w _depend_ t_depend_ n _name_ _type_ intercept;
   run;
proc print data=sims label;
   by _depvar_ ;
   title2 'Simulations Sorted by Decreasing Predicted Utility';
   title3 'Just the Ten Most Preferred Combinations are Printed';
   label p_depend_ = 'Predicted Utility';
   run;
```

Frozen Diet Entrees
Simulations Sorted by Decreasing Predicted Utility
Just the Ten Most Preferred Combinations are Printed

---------- Dependent Variable Transformation(Name)=Identity(Subj01) ----------

Obs	Predicted Utility	Ingredient	Fat	Price	Calories
1	22.0556	Chicken	2 Grams	$2.29	250
2	22.0556	Chicken	2 Grams	$2.29	250
3	22.0556	Chicken	2 Grams	$2.29	250
4	21.3889	Chicken	2 Grams	$1.99	350
5	21.3889	Chicken	2 Grams	$1.99	350
6	21.3889	Chicken	2 Grams	$1.99	350
7	20.3889	Turkey	2 Grams	$1.99	350
8	20.3889	Turkey	2 Grams	$1.99	350
9	20.3889	Turkey	2 Grams	$1.99	350
10	18.3889	Beef	2 Grams	$2.29	250

---------- Dependent Variable Transformation(Name)=Identity(Subj02) ----------

Obs	Predicted Utility	Ingredient	Fat	Price	Calories
11	20.9444	Beef	2 Grams	$2.29	250
12	20.9444	Beef	2 Grams	$2.29	250
13	20.9444	Beef	2 Grams	$2.29	250
14	20.7778	Turkey	2 Grams	$1.99	350
15	20.7778	Turkey	2 Grams	$1.99	350
16	20.7778	Turkey	2 Grams	$1.99	350
17	19.7778	Chicken	2 Grams	$2.29	250
18	19.7778	Chicken	2 Grams	$2.29	250
19	19.7778	Chicken	2 Grams	$2.29	250
20	19.7778	Turkey	2 Grams	$2.59	250

---------- Dependent Variable Transformation(Name)=Identity(Subj03) ----------

Obs	Predicted Utility	Ingredient	Fat	Price	Calories
21	21.6667	Chicken	2 Grams	$2.29	250
22	21.6667	Chicken	2 Grams	$2.29	250
23	21.6667	Chicken	2 Grams	$2.29	250
24	21.3333	Chicken	2 Grams	$1.99	350
25	21.3333	Chicken	2 Grams	$1.99	350
26	21.3333	Chicken	2 Grams	$1.99	350
27	21.0000	Turkey	2 Grams	$1.99	350
28	21.0000	Turkey	2 Grams	$1.99	350
29	21.0000	Turkey	2 Grams	$1.99	350
30	20.0000	Beef	2 Grams	$2.29	250

---------- Dependent Variable Transformation(Name)=Identity(Subj04) ----------

Obs	Predicted Utility	Ingredient	Fat	Price	Calories
31	20.9444	Beef	2 Grams	$2.29	250
32	20.9444	Beef	2 Grams	$2.29	250
33	20.9444	Beef	2 Grams	$2.29	250
34	20.2778	Beef	5 Grams	$1.99	250
35	20.2778	Beef	5 Grams	$1.99	250
36	20.2778	Beef	5 Grams	$1.99	250
37	18.4444	Turkey	8 Grams	$1.99	250
38	18.4444	Turkey	8 Grams	$1.99	250
39	18.4444	Turkey	8 Grams	$1.99	250
40	18.1111	Chicken	2 Grams	$2.29	250

---------- Dependent Variable Transformation(Name)=Identity(Subj05) ----------

Obs	Predicted Utility	Ingredient	Fat	Price	Calories
41	19.8889	Beef	5 Grams	$1.99	250
42	19.8889	Beef	5 Grams	$1.99	250
43	19.8889	Beef	5 Grams	$1.99	250
44	19.5556	Chicken	2 Grams	$1.99	350
45	19.5556	Chicken	2 Grams	$1.99	350
46	19.5556	Chicken	2 Grams	$1.99	350
47	18.3889	Beef	8 Grams	$1.99	250
48	18.3889	Beef	8 Grams	$1.99	250
49	18.3889	Beef	8 Grams	$1.99	250
50	17.8889	Turkey	2 Grams	$1.99	350

---------- Dependent Variable Transformation(Name)=Identity(Subj06) ----------

Obs	Predicted Utility	Ingredient	Fat	Price	Calories
51	21.3333	Chicken	2 Grams	$1.99	350
52	21.3333	Chicken	2 Grams	$1.99	350
53	21.3333	Chicken	2 Grams	$1.99	350
54	20.0556	Beef	5 Grams	$1.99	250
55	20.0556	Beef	5 Grams	$1.99	250
56	20.0556	Beef	5 Grams	$1.99	250
57	18.5000	Turkey	2 Grams	$1.99	350
58	18.5000	Turkey	2 Grams	$1.99	350
59	18.5000	Turkey	2 Grams	$1.99	350
60	17.7222	Beef	8 Grams	$1.99	250

---------- Dependent Variable Transformation(Name)=Identity(Subj07) ----------

Obs	Predicted Utility	Ingredient	Fat	Price	Calories
61	21.8889	Beef	2 Grams	$2.29	250
62	21.8889	Beef	2 Grams	$2.29	250
63	21.8889	Beef	2 Grams	$2.29	250
64	21.3889	Chicken	2 Grams	$2.29	250
65	21.3889	Chicken	2 Grams	$2.29	250
66	21.3889	Chicken	2 Grams	$2.29	250
67	20.5556	Chicken	2 Grams	$1.99	350
68	20.5556	Chicken	2 Grams	$1.99	350
69	20.5556	Chicken	2 Grams	$1.99	350
70	19.3889	Turkey	2 Grams	$1.99	350

---------- Dependent Variable Transformation(Name)=Identity(Subj08) ----------

Obs	Predicted Utility	Ingredient	Fat	Price	Calories
71	20.1667	Chicken	2 Grams	$1.99	350
72	20.1667	Chicken	2 Grams	$1.99	350
73	20.1667	Chicken	2 Grams	$1.99	350
74	19.6667	Turkey	2 Grams	$1.99	350
75	19.6667	Turkey	2 Grams	$1.99	350
76	19.6667	Turkey	2 Grams	$1.99	350
77	19.1667	Beef	5 Grams	$1.99	250
78	19.1667	Beef	5 Grams	$1.99	250
79	19.1667	Beef	5 Grams	$1.99	250
80	17.8333	Chicken	5 Grams	$1.99	350

---------- Dependent Variable Transformation(Name)=Identity(Subj09) ----------

Obs	Predicted Utility	Ingredient	Fat	Price	Calories
81	20.5000	Chicken	2 Grams	$1.99	350
82	20.5000	Chicken	2 Grams	$1.99	350
83	20.5000	Chicken	2 Grams	$1.99	350
84	20.3333	Turkey	2 Grams	$1.99	350
85	20.3333	Turkey	2 Grams	$1.99	350
86	20.3333	Turkey	2 Grams	$1.99	350
87	18.5556	Beef	5 Grams	$1.99	250
88	18.5556	Beef	5 Grams	$1.99	250
89	18.5556	Beef	5 Grams	$1.99	250
90	18.3333	Chicken	5 Grams	$1.99	350

---------- Dependent Variable Transformation(Name)=Identity(Subj10) ----------

Obs	Predicted Utility	Ingredient	Fat	Price	Calories
91	20.2778	Beef	5 Grams	$1.99	250
92	20.2778	Beef	5 Grams	$1.99	250
93	20.2778	Beef	5 Grams	$1.99	250
94	19.6111	Turkey	2 Grams	$1.99	350
95	19.6111	Turkey	2 Grams	$1.99	350
96	19.6111	Turkey	2 Grams	$1.99	350
97	18.6111	Beef	8 Grams	$1.99	250
98	18.6111	Beef	8 Grams	$1.99	250
99	18.6111	Beef	8 Grams	$1.99	250
100	18.1111	Turkey	8 Grams	$1.99	250

```
---------- Dependent Variable Transformation(Name)=Identity(Subj12) ----------
```

Obs	Predicted Utility	Ingredient	Fat	Price	Calories
101	21.3333	Chicken	2 Grams	$2.29	250
102	21.3333	Chicken	2 Grams	$2.29	250
103	21.3333	Chicken	2 Grams	$2.29	250
104	21.1667	Chicken	2 Grams	$1.99	350
105	21.1667	Chicken	2 Grams	$1.99	350
106	21.1667	Chicken	2 Grams	$1.99	350
107	21.1667	Beef	2 Grams	$2.29	250
108	21.1667	Beef	2 Grams	$2.29	250
109	21.1667	Beef	2 Grams	$2.29	250
110	18.8333	Turkey	2 Grams	$1.99	350

Summarizing Results Across Subjects

Conjoint analyses are performed on an individual basis, but usually the goal is to summarize the results across subjects. The `outtest=` data set contains all of the information in the printed output and can be manipulated to create additional reports including a list of the individual R^2s and the average of the importance values across subjects. Here is a listing of the variables in the `outtest=` data set.

```
proc contents data=utils2 position;
   ods select position;
   title2 'Variables in the OUTTEST= Data Set';
   run;
```

Frozen Diet Entrees
Variables in the OUTTEST= Data Set

The CONTENTS Procedure

Variables in Creation Order

#	Variable	Type	Len	Label
1	_DEPVAR_	Char	42	Dependent Variable Transformation(Name)
2	_TYPE_	Char	8	
3	Title	Char	80	Title
4	Variable	Char	42	Variable
5	Coefficient	Num	8	Coefficient
6	Statistic	Char	24	Statistic

7	Value	Num	8	Value
8	NumDF	Num	8	Num DF
9	DenDF	Num	8	Den DF
10	SSq	Num	8	Sum of Squares
11	MeanSquare	Num	8	Mean Square
12	F	Num	8	F Value
13	NumericP	Num	8	Numeric (Approximate) p Value
14	P	Char	9	Formatted p Value
15	LowerLimit	Num	8	95% Lower Confidence Limit
16	UpperLimit	Num	8	95% Upper Confidence Limit
17	StdError	Num	8	Standard Error
18	Importance	Num	8	Importance (% Utility Range)
19	Label	Char	256	Label

The individual R^2s are displayed by printing the `Value` variable for observations whose `Statistic` value is "R-Square".

```
proc print data=utils2 label;
   title2 'R-Squares';
   id _depvar_;
   var value;
   format value 4.2;
   where statistic = 'R-Square';
   label value = 'R-Square' _depvar_ = 'Subject';
   run;
```

Frozen Diet Entrees
R-Squares

Subject	R-Square
Identity(Subj01)	0.96
Identity(Subj02)	0.98
Identity(Subj03)	0.98
Identity(Subj04)	0.98
Identity(Subj05)	0.99
Identity(Subj06)	0.96
Identity(Subj07)	0.99
Identity(Subj08)	0.99
Identity(Subj09)	0.99
Identity(Subj10)	0.99
Identity(Subj12)	0.97

The next steps extract the importance values and create a table. The DATA step extracts the importance values and creates row and column labels. The PROC TRANSPOSE step creates a subjects by attributes matrix from a vector (of the number of subjects times the number of attribute values). PROC PRINT displays the importance values, and PROC MEANS displays the average importances.

```
data im;
   set utils2;
   if n(importance);   /* Exclude all missing, including specials.*/
   _depvar_ = scan(_depvar_, 2);   /* Discard transformation.     */
   label    = scan(label, 1, ','); /* Use up to comma for label.  */
   keep importance _depvar_ label;
   run;
proc transpose data=im out=im(drop=_name_ _label_);
   id label;
   by notsorted _depvar_;
   var importance;
   label _depvar_ = 'Subject';
   run;
proc print label;
   title2 'Importances';
   format _numeric_ 2.;
   id _depvar_;
   run;
proc means mean;
   title2 'Average Importances';
   run;
```

Frozen Diet Entrees
Importances

Subject	Ingredient	Fat	Price	Calories
Subj01	13	50	24	13
Subj02	8	65	15	11
Subj03	7	62	21	11
Subj04	13	22	25	39
Subj05	7	17	64	12
Subj06	11	25	52	13
Subj07	7	68	15	9
Subj08	4	21	65	10
Subj09	7	18	66	8
Subj10	10	19	59	13
Subj12	9	52	24	15

```
Frozen Diet Entrees
Average Importances

The MEANS Procedure

Variable              Mean
--------------------------
Ingredient       8.9069044
Fat             38.0953010
Price           39.0198700
Calories        13.9779245
--------------------------
```

On the average, price is the most important attribute followed very closely by fat content. These two attributes on the average account for 77% of preference. Calories and main ingredient account for the remaining 23%. Note that everyone does not have the same pattern of importance values. However, it is a little hard to compare subjects just by looking at the numbers.

We can make a nicer display of importances with stars flagging the most important attributes for each product as follows. These steps replace each importance variable with its formatted value followed by zero stars for 0 - 30, one star for 30 - 45, two stars for 45 - 60, three stars for 60 - 75, and so on. The value returned by the `ceil` function is the number of characters that are extracted from the string `' ******'`.

```
data im2;
   set im;
   label c1 = 'Ingredient' c2 = 'Fat' c3 = 'Price' c4 = 'Calories';
   c1 = put(ingredient, 2.) || substr('  ******', 1, ceil(ingredient / 15));
   c2 = put(fat       , 2.) || substr('  ******', 1, ceil(fat        / 15));
   c3 = put(price     , 2.) || substr('  ******', 1, ceil(price      / 15));
   c4 = put(calories  , 2.) || substr('  ******', 1, ceil(calories   / 15));
   run;
proc print label;
   title2 'Importances';
   var c1-c4;
   id _depvar_;
   run;
```

Frozen Diet Entrees
Importances

Subject	Ingredient	Fat		Price		Calories	
Subj01	13	50	**	24		13	
Subj02	8	65	***	15		11	
Subj03	7	62	***	21		11	
Subj04	13	22		25		39	*
Subj05	7	17		64	***	12	
Subj06	11	25		52	**	13	
Subj07	7	68	***	15		9	
Subj08	4	21		65	***	10	
Subj09	7	18		66	***	8	
Subj10	10	19		59	**	13	
Subj12	9	52	**	24		15	

Subject 4 is more concerned about calories. However, most individuals seem to fall into one of two groups, either primarily price conscious then fat conscious, or primarily fat conscious then price conscious.

Both the `out=` data set and the `outtest=` data set contain the part-worth utilities. In the `out=` data set, they are contained in the observations whose `_type_` value is `'M COEFFI'`. The part-worth utilities are the multiple regression coefficients. The names of the variables that contain the part-worth utilities are stored in the macro variable `&_trgind`, which is automatically created by PROC TRANSREG.

```
proc print data=results2 label;
   title2 'Part-Worth Utilities';
   where _type_ = 'M COEFFI';
   id _name_;
   var &_trgind;
   run;
```

Frozen Diet Entrees
Part-Worth Utilities

NAME	Ingredient, Chicken	Ingredient, Beef	Ingredient, Turkey	Fat, 8 Grams	Fat, 5 Grams
Subj01	1.55556	-2.11111	0.55556	-6.94444	-0.11111
Subj02	-1.05556	0.11111	0.94444	-7.72222	0.11111
Subj03	0.66667	-1.00000	0.33333	-7.66667	-0.16667
Subj04	-2.11111	0.72222	1.38889	-2.77778	-0.27778
Subj05	0.55556	0.55556	-1.11111	-1.77778	-0.27778
Subj06	1.11111	0.61111	-1.72222	-2.88889	-0.55556
Subj07	0.22222	0.72222	-0.94444	-7.61111	-0.27778
Subj08	0.50000	-0.50000	0.00000	-2.33333	0.00000
Subj09	0.61111	-1.05556	0.44444	-2.05556	-0.05556
Subj10	-1.38889	0.94444	0.44444	-2.05556	-0.38889
Subj12	0.83333	0.66667	-1.50000	-6.16667	-1.16667

NAME	Fat, 2 Grams	Price, $2.59	Price, $2.29	Price, $1.99	Calories, 350	Calories, 250
Subj01	7.05556	-3.44444	0.22222	3.22222	-1.83333	1.83333
Subj02	7.61111	-1.88889	0.11111	1.77778	-1.33333	1.33333
Subj03	7.83333	-2.66667	0.16667	2.50000	-1.33333	1.33333
Subj04	3.05556	-3.44444	0.38889	3.05556	-5.05556	5.05556
Subj05	2.05556	-7.27778	0.22222	7.05556	-1.33333	1.33333
Subj06	3.44444	-6.22222	-0.88889	7.11111	-1.61111	1.61111
Subj07	7.88889	-1.94444	0.38889	1.55556	-1.00000	1.00000
Subj08	2.33333	-7.33333	-0.00000	7.33333	-1.16667	1.16667
Subj09	2.11111	-7.38889	-0.05556	7.44444	-0.94444	0.94444
Subj10	2.44444	-7.22222	0.27778	6.94444	-1.50000	1.50000
Subj12	7.33333	-2.83333	-0.50000	3.33333	-2.00000	2.00000

These part-worth utilities can be clustered, for example using PROC FASTCLUS.

```
proc fastclus data=results2 maxclusters=3 out=clusts;
   where _type_ = 'M COEFFI';
   id _name_;
   var &_trgind;
   run;
proc sort; by cluster; run;
proc print label;
   title2 'Part-Worth Utilities, Clustered';
   by cluster;
   id _name_;
   var &_trgind;
   run;
```

Frozen Diet Entrees
Part-Worth Utilities, Clustered

---------- Cluster=1 ----------

NAME	Ingredient, Chicken	Ingredient, Beef	Ingredient, Turkey	Fat, 8 Grams	Fat, 5 Grams
Subj05	0.55556	0.55556	-1.11111	-1.77778	-0.27778
Subj06	1.11111	0.61111	-1.72222	-2.88889	-0.55556
Subj08	0.50000	-0.50000	0.00000	-2.33333	0.00000
Subj09	0.61111	-1.05556	0.44444	-2.05556	-0.05556
Subj10	-1.38889	0.94444	0.44444	-2.05556	-0.38889

NAME	Fat, 2 Grams	Price, $2.59	Price, $2.29	Price, $1.99	Calories, 350	Calories, 250
Subj05	2.05556	-7.27778	0.22222	7.05556	-1.33333	1.33333
Subj06	3.44444	-6.22222	-0.88889	7.11111	-1.61111	1.61111
Subj08	2.33333	-7.33333	-0.00000	7.33333	-1.16667	1.16667
Subj09	2.11111	-7.38889	-0.05556	7.44444	-0.94444	0.94444
Subj10	2.44444	-7.22222	0.27778	6.94444	-1.50000	1.50000

---------- Cluster=2 ----------

NAME	Ingredient, Chicken	Ingredient, Beef	Ingredient, Turkey	Fat, 8 Grams	Fat, 5 Grams
Subj01	1.55556	-2.11111	0.55556	-6.94444	-0.11111
Subj02	-1.05556	0.11111	0.94444	-7.72222	0.11111
Subj03	0.66667	-1.00000	0.33333	-7.66667	-0.16667
Subj07	0.22222	0.72222	-0.94444	-7.61111	-0.27778
Subj12	0.83333	0.66667	-1.50000	-6.16667	-1.16667

NAME	Fat, 2 Grams	Price, $2.59	Price, $2.29	Price, $1.99	Calories, 350	Calories, 250
Subj01	7.05556	-3.44444	0.22222	3.22222	-1.83333	1.83333
Subj02	7.61111	-1.88889	0.11111	1.77778	-1.33333	1.33333
Subj03	7.83333	-2.66667	0.16667	2.50000	-1.33333	1.33333
Subj07	7.88889	-1.94444	0.38889	1.55556	-1.00000	1.00000
Subj12	7.33333	-2.83333	-0.50000	3.33333	-2.00000	2.00000

---------- Cluster=3 ----------

NAME	Ingredient, Chicken	Ingredient, Beef	Ingredient, Turkey	Fat, 8 Grams	Fat, 5 Grams
Subj04	-2.11111	0.72222	1.38889	-2.77778	-0.27778

NAME	Fat, 2 Grams	Price, $2.59	Price, $2.29	Price, $1.99	Calories, 350	Calories, 250
Subj04	3.05556	-3.44444	0.38889	3.05556	-5.05556	5.05556

The clusters reflect what we saw looking at the importance information. Subject 4, who is the only subject that is primarily calorie conscious, is in a separate cluster from everyone else. Cluster 1 subjects 5, 6, 8, 9, and 10 are primarily price conscious. Cluster 2 subjects 1, 2, 3, 7, and 12 are primarily fat conscious.

Spaghetti Sauce

This example uses conjoint analysis in a study of spaghetti sauce preferences. The goal is to investigate the main effects for all of the attributes and the interaction of brand and price, and to simulate market share. Rating scale data are gathered from a group of subjects. The example has eight parts.

- An efficient experimental design is generated with the `%MktEx` macro.
- Descriptions of the spaghetti sauces are generated.
- Data are collected, entered, and processed.
- The metric conjoint analysis is performed with PROC TRANSREG.
- Market share is simulated with the maximum utility model.
- Market share is simulated with the Bradley-Terry-Luce and logit models.
- The simulators are compared.
- Change in market share is investigated.

Create an Efficient Experimental Design with the %MktEx Macro

In this example, subjects were asked to rate their interest in purchasing hypothetical spaghetti sauces. The table shows the attributes, the attribute levels, and the number of *df* associated with each effect.

Experimental Design		
Effects	**Levels**	*df*
Intercept		1
Brand	Pregu, Sundance, Tomato Garden	2
Meat Content	Vegetarian, Meat, Italian Sausage	2
Mushroom Content	Mushrooms, No Mention	1
Natural Ingredients	All Natural Ingredients, No Mention	1
Price	\$1.99, \$2.29, \$2.49, \$2.79, \$2.99	4
Brand × Price		8

The brand names "Pregu", "Sundance", and "Tomato Garden" are artificial. Usually, real brand names would be used−your client's or company's brand and the competitors' brands. The absence of a feature (for example, no mushrooms) is not mentioned in the product description, hence the "No Mention" in the table.

In this design there are 19 model *df*. A design with more than 19 runs must be generated if there are to be error *df*. A popular heuristic is to limit the design size to at most 30 runs. In this example, 30 runs allow us to have two observations in each of the 15 brand by price cells. Note however that when subjects are required to make that many judgments, there is the risk that the quality of the data will be poor. Caution should be used when generating designs with this many runs. We can use the `%MktRuns` macro to evaluate this and other design sizes. See page 597 for macro documentation

and information on installing and using SAS autocall macros. We specify the number of levels of each factor as the argument.

```
title 'Spaghetti Sauces';

%mktruns( 3 3 2 2 5 )
```

Spaghetti Sauces

Design Summary

Number of Levels	Frequency
2	2
3	2
5	1

Saturated = 11
Full Factorial = 180

Some Reasonable Design Sizes	Violations	Cannot Be Divided By
180 *	0	
60	1	9
90	1	4
120	1	9
30	2	4 9
150	2	4 9
210	2	4 9
36	5	5 10 15
72	5	5 10 15
108	5	5 10 15

* - 100% Efficient Design can be made with the MktEx Macro.

We see that 30 is a reasonable size, although it cannot be divided by $9 = 3 \times 3$ and $4 = 2 \times 2$, so perfect orthogonality will not be possible. We would need a much larger size like 60 or 180 to do better. Note that this output states "Saturated=11" referring to a main-effects model. In this example, we are also interested in the brand by price interaction. We can run the %MktRuns macro again, this time specifying the interaction.

```
%mktruns( 3 3 2 2 5, interact=1*5 )
```

Spaghetti Sauces

Design Summary

Number of Levels	Frequency
2	2
3	2
5	1

Spaghetti Sauces

Saturated = 19
Full Factorial = 180

Some Reasonable Design Sizes	Violations	Cannot Be Divided By
180	0	
90	1	4
60	2	9 45
120	2	9 45
30	3	4 9 45
150	3	4 9 45
210	3	4 9 45
36	8	5 10 15 30 45
72	8	5 10 15 30 45
108	8	5 10 15 30 45

Now the output states "Saturated=19", which includes the 8 *df* for the interaction. We see as before that 30 cannot be divided by $4 = 2 \times 2$. We also see that 30 cannot be divide by $45 = 3 \times 15$ so each level of meat content will not appear equally often in each brand/price cell. Since we would need a much larger size to do better, we will use 30 runs.

The next steps create and evaluate the design. First, formats for each of the factors are created using PROC FORMAT. The `%MktEx` macro is called to create the design. The factors `x1 = Brand` and `x2 = Meat` are designated as three-level factors, `x3 = Mushroom` and `x4 = Ingredients` as two-level factors, and `x5 = Price` as a five-level factor. The `interact=1*5` option specifies that the interaction between the first and fifth factors must be estimable (`x1` × `x5` which is brand by price), `n=30` specifies the number of runs, and `seed=289` specifies the random number seed. The `where` macro provides restrictions that eliminate unrealistic combinations. Specifically, products at the cheapest price, \$1.99, with meat, and products with Italian Sausage with All Natural Ingredients are eliminated from consideration.

We impose restrictions with the `%MktEx` macro by writing a macro, with IML statements, that quantifies the badness of each run of the design. The variable `bad` is set to zero when everything is fine, and values larger than zero when the row of the design does not conform to the restrictions. Ideally, when there are multiple restrictions, as there are here, the variable `bad` should be set to the number of violations, so the macro can know when it is moving in the right direction as it changes the design. Our first

restriction (contribution to the badness value) is `(x2 = 3 & x4 = 1)` and our second is `(x5 = 1 & (x2 = 2 | x2 = 3))`, where `&` means `and` and `|` means `or`.[§] The restrictions correspond to `(Meat = 'Italian Sausage' & Ingredients = 'All Natural')` and `(Price = 1.99 & (Meat = 'Meat' | Meat = 'Italian Sausage'))`. Each of these Boolean or logical expressions evaluates to 1 when the expression is true and 0 when it is false. The sum of the two restrictions is: 0 - no problem, 1 - one restriction violation, or 2 - two restriction violations.

The `%MktLab` macro assigns actual descriptive factor names instead of the default `x1-x5` and formats for the levels. The default input to the `%MktLab` macro is the data set `Randomized`, which is the randomized design created by the `%MktEx` macro.

The default output from the `%MktLab` macro is a data set called `Final`. We instead use the `out=` option to store the results in a permanent SAS data set. The `%MktEx` macro is used to display the frequencies for each level, the two-way frequencies, and the number of times each product occurs in the design (five-way frequencies).

```
title 'Spaghetti Sauces';

proc format;
   value br 1='Pregu'       2='Sundance'  3='Tomato Garden';
   value me 1='Vegetarian'  2='Meat'      3='Italian Sausage';
   value mu 1='Mushrooms'   2='No Mention';
   value in 1='All Natural' 2='No Mention';
   value pr 1='1.99' 2='2.29' 3='2.49' 4='2.79' 5='2.99';
   run;

%macro where;
   bad = (x2 = 3 & x4 = 1) + (x5 = 1 & (x2 = 2 | x2 = 3));
   %mend;

%mktex(3 3 2 2 5, interact=1*5, n=30, seed=289, restrictions=where)
%mktlab(vars=Brand Meat Mushroom Ingredients Price,
        statements=format brand br. meat me. mushroom mu.
                   ingredients in. price pr.,
        out=sasuser.spag);
%mkteval;

proc print data=sasuser.spag; run;
```

[§]In the restrictions macro, you must use the logical symbols `| & ∧ ¬ > < >= <= = ∧= ¬=` and *not* the logical words `OR AND NOT GT LT GE LE EQ NE`.

Here is some of the output from the `%MktEx` macro.

Spaghetti Sauces

Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
1	Start	92.6280		Can
1	2 1	92.6280	92.6280	Conforms
1	End	92.6280		
2	Start	78.9640		Tab,Unb
2	28 1	91.5726		Conforms
2	End	91.6084		
3	Start	78.9640		Tab,Unb
3	1 1	91.5434		Conforms
3	End	91.6084		
4	Start	77.5906		Tab,Ran
4	28 1	91.9486		Conforms
4	5 4	92.6280	92.6280	
4	End	92.6280		
.				
.				
.				
21	Start	74.7430		Ran,Mut,Ann
21	24 1	89.9706		Conforms
21	End	91.6084		

Spaghetti Sauces

Design Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
0	Initial	92.6280	92.6280	Ini
1	Start	92.6280		Can
1	2 1	92.6280	92.6280	Conforms
1	End	92.6280		

Spaghetti Sauces

Design Refinement History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
0	Initial	92.6280	92.6280	Ini
1	Start	90.4842		Pre,Mut,Ann
1	2 1	91.2145		Conforms
1	End	91.6084		
.				
.				
.				
6	Start	91.1998		Pre,Mut,Ann
6	2 1	91.6084		Conforms
6	End	91.6084		

NOTE: Stopping since it appears that no improvement is possible.

Spaghetti Sauces

The OPTEX Procedure

Class Level Information

Class	Levels	-Values--
x1	3	1 2 3
x2	3	1 2 3
x3	2	1 2
x4	2	1 2
x5	5	1 2 3 4 5

Spaghetti Sauces

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	92.6280	82.6056	97.6092	0.7958

The *D*-Efficiency looks reasonable at 92.63. For this problem, the full-factorial design is small (180 runs), and the macro found the same *D*-efficiency several times. This suggests that we have probably

indeed found the optimal design for this situation. Here is the output from the `%MktEval` macro.

```
                            Spaghetti Sauces
                 Canonical Correlations Between the Factors
            There are 2 Canonical Correlations Greater Than 0.316

                 Brand     Meat    Mushroom    Ingredients    Price

   Brand         1         0.21      0            0.17         0
   Meat          0.21      1         0.08         0.42         0.52
   Mushroom      0         0.08      1            0            0
   Ingredients   0.17      0.42      0            1            0.17
   Price         0         0.52      0            0.17         1

                            Spaghetti Sauces
            Canonical Correlations > 0.316 Between the Factors
            There are 2 Canonical Correlations Greater Than 0.316

                                          r    r Square

               Meat    Price           0.52      0.27
               Meat    Ingredients     0.42      0.17

                            Spaghetti Sauces
                         Summary of Frequencies
            There are 2 Canonical Correlations Greater Than 0.316
                     * - Indicates Unequal Frequencies

                                 Frequencies

       Brand                  10 10 10
  *    Meat                   15 9 6
       Mushroom               15 15
  *    Ingredients            12 18
       Price                  6 6 6 6 6
  *    Brand Meat             4 3 3 5 4 1 6 2 2
       Brand Mushroom         5 5 5 5 5 5
  *    Brand Ingredients      3 7 5 5 4 6
       Brand Price            2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
  *    Meat Mushroom          7 8 5 4 3 3
  *    Meat Ingredients       7 8 5 4 0 6
  *    Meat Price             6 3 2 2 2 0 2 2 3 2 0 1 2 1 2
  *    Mushroom Ingredients   6 9 6 9
       Mushroom Price         3 3 3 3 3 3 3 3 3 3
  *    Ingredients Price      3 3 2 2 2 3 3 4 4 4
       N-Way                  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
                              1 1 1 1 1 1 1 1 1 1 1
```

The meat and price factors are correlated, as are the meat and ingredients factors. This is not surprising since we excluded cells for these factor combinations and hence forced some correlations. The rest of the correlations are small.

The frequencies look good. The n-way frequencies at the end of this listing show that each product occurs only once. Each brand, price, and brand/price combination occurs equally often, as does each mushroom level. There are more vegetarian sauces (the first formatted level) than either of the meat sauces because of the restrictions that meat cannot occur at the lowest price and Italian sausage cannot be paired with all-natural ingredients. The design is shown next.

Spaghetti Sauces

Obs	Brand	Meat	Mushroom	Ingredients	Price
1	Pregu	Meat	No Mention	No Mention	2.79
2	Tomato Garden	Vegetarian	No Mention	No Mention	2.79
3	Pregu	Meat	Mushrooms	All Natural	2.29
4	Tomato Garden	Vegetarian	Mushrooms	All Natural	2.49
5	Sundance	Vegetarian	Mushrooms	No Mention	1.99
6	Pregu	Italian Sausage	No Mention	No Mention	2.49
7	Tomato Garden	Vegetarian	No Mention	No Mention	2.99
8	Tomato Garden	Italian Sausage	Mushrooms	No Mention	2.29
9	Pregu	Vegetarian	Mushrooms	No Mention	2.49
10	Pregu	Vegetarian	No Mention	No Mention	2.29
11	Sundance	Vegetarian	Mushrooms	No Mention	2.79
12	Tomato Garden	Vegetarian	Mushrooms	No Mention	1.99
13	Sundance	Meat	No Mention	No Mention	2.29
14	Sundance	Meat	Mushrooms	No Mention	2.99
15	Pregu	Italian Sausage	Mushrooms	No Mention	2.79
16	Tomato Garden	Italian Sausage	Mushrooms	No Mention	2.99
17	Sundance	Vegetarian	Mushrooms	All Natural	2.29
18	Pregu	Meat	Mushrooms	All Natural	2.99
19	Tomato Garden	Meat	No Mention	No Mention	2.49
20	Sundance	Meat	Mushrooms	All Natural	2.49
21	Pregu	Vegetarian	No Mention	All Natural	1.99
22	Sundance	Meat	No Mention	All Natural	2.79
23	Tomato Garden	Vegetarian	No Mention	All Natural	1.99
24	Sundance	Italian Sausage	No Mention	No Mention	2.49
25	Sundance	Vegetarian	No Mention	All Natural	1.99
26	Sundance	Vegetarian	No Mention	All Natural	2.99
27	Pregu	Italian Sausage	No Mention	No Mention	2.99
28	Tomato Garden	Vegetarian	No Mention	All Natural	2.29
29	Pregu	Vegetarian	Mushrooms	No Mention	1.99
30	Tomato Garden	Meat	Mushrooms	All Natural	2.79

Generating the Questionnaire

Next, preparations are made for data collection. A DATA step is used to print descriptions of each product combination. Here is an example:

```
Try Pregu brand vegetarian spaghetti sauce, now with
mushrooms.  A 26 ounce jar serves four adults for only
$1.99.
```

Remember that "No Mention" is not mentioned. The following step prints the questionnaires including a cover sheet.

```
options ls=80 ps=74 nonumber nodate;
title;

data _null_;
   set sasuser.spag;
   length lines $ 500 aline $ 60;
   file print linesleft=ll;

   * Format meat level, preserve 'Italian' capitalization;
   aline = lowcase(put(meat, me.));
   if aline =: 'ita' then substr(aline, 1, 1) = 'I';

   * Format meat differently for 'vegetarian';
   if meat > 1
      then lines = 'Try ' || trim(put(brand, br.)) ||
                   ' brand spaghetti sauce with ' || aline;
      else lines = 'Try ' || trim(put(brand, br.)) ||
                   ' brand ' || trim(aline) || ' spaghetti sauce ';
   * Add mushrooms, natural ingredients to text line;
   n = (put(ingredients, in.) =: 'All');
   m = (put(mushroom,    mu.) =: 'Mus');

   if n or m then do;
      lines = trim(lines) || ', now with';

      if m then do;
         lines = trim(lines) || ' ' || lowcase(put(mushroom, mu.));
         if n then lines = trim(lines) || ' and';
         end;
      if n then lines = trim(lines) || ' ' ||
                        lowcase(put(ingredients, in.)) || ' ingredients';
      end;
   * Add price;
   lines = trim(lines) ||
          '.  A 26 ounce jar serves four adults for only $' ||
          put(price, pr.) || '.';
```

```
* Print cover page, with subject number, instructions, and rating scale;
if _n_ = 1 then do;
   put ///// +41 'Subject: ________' ////
       +5 'Please rate your willingness to purchase the following' /
       +5 'products on a nine point scale.' ///
       +9 '1   Definitely Would Not Purchase This Product' ///
       +9 '2'  ///
       +9 '3   Probably Would Not Purchase This Product' ///
       +9 '4'  ///
       +9 '5   May or May Not Purchase This Product' ///
       +9 '6'  ///
       +9 '7   Probably Would Purchase This Product' ///
       +9 '8'  ///
       +9 '9   Definitely Would Purchase This Product' /////
       +5 'Please rate every product and be sure to rate' /
       +5 'each product only once.' //////
       +5 'Thank you for your participation!';
   put _page_;
   end;
if ll < 8 then put _page_;
* Break up description, print on several lines;

start = 1;
do l = 1 to 10 until(aline = ' ');

   * Find a good place to split, blank or punctuation;
   stop = start + 60;
   do i = stop to start by -1 while(substr(lines, i, 1) ne ' '); end;
   do j = i to max(start, i - 8) by -1;
      if substr(lines, j, 1) in ('.' ',') then do; i = j; j = 0; end;
      end;

   stop = i; len = stop + 1 - start;
   aline = substr(lines, start, len);
   start = stop + 1;
   if l = 1 then put +5 _n_ 2. ') ' aline;
   else          put +9 aline;
   end;
* Print rating scale;
put +9 'Definitely                                    Definitely ' /
    +9 'Would Not   1   2   3   4   5   6   7   8   9   Would      ' /
    +9 'Purchase                                      Purchase   ' //;
run;

options ls=80 ps=60 nonumber nodate;
```

In the interest of space, not all questions are printed.

Subject: _____

Please rate your willingness to purchase the following products on a nine point scale.

1 Definitely Would Not Purchase This Product

2

3 Probably Would Not Purchase This Product

4

5 May or May Not Purchase This Product

6

7 Probably Would Purchase This Product

8

9 Definitely Would Purchase This Product

Please rate every product and be sure to rate each product only once.

Thank you for your participation!

1) Try Pregu brand spaghetti sauce with meat. A 26 ounce jar serves four adults for only $2.79.

Definitely Would Not Purchase	1	2	3	4	5	6	7	8	9	Definitely Would Purchase

2) Try Tomato Garden brand vegetarian spaghetti sauce. A 26 ounce jar serves four adults for only $2.79.

Definitely Would Not Purchase	1	2	3	4	5	6	7	8	9	Definitely Would Purchase

3) Try Pregu brand spaghetti sauce with meat, now with mushrooms and all natural ingredients. A 26 ounce jar serves four adults for only $2.29.

Definitely Would Not Purchase	1	2	3	4	5	6	7	8	9	Definitely Would Purchase

4) Try Tomato Garden brand vegetarian spaghetti sauce, now with mushrooms and all natural ingredients. A 26 ounce jar serves four adults for only $2.49.

Definitely Would Not Purchase	1	2	3	4	5	6	7	8	9	Definitely Would Purchase

5) Try Sundance brand vegetarian spaghetti sauce, now with mushrooms. A 26 ounce jar serves four adults for only $1.99.

Definitely Would Not Purchase	1	2	3	4	5	6	7	8	9	Definitely Would Purchase

6) Try Pregu brand spaghetti sauce with Italian sausage. A 26 ounce jar serves four adults for only $2.49.

Definitely Would Not Purchase	1	2	3	4	5	6	7	8	9	Definitely Would Purchase

7) Try Tomato Garden brand vegetarian spaghetti sauce. A 26 ounce jar serves four adults for only $2.99.

Definitely Would Not Purchase	1	2	3	4	5	6	7	8	9	Definitely Would Purchase

```
        .
        .
        .

      30) Try Tomato Garden brand spaghetti sauce with meat, now with
          mushrooms and all natural ingredients.  A 26 ounce jar
          serves four adults for only $2.79.

          Definitely                                      Definitely
          Would Not   1   2   3   4   5   6   7   8   9   Would
          Purchase                                        Purchase
```

Data Processing

The data are entered next. Some cases have ordinary '.' missing values. This code was used at data entry for no response. When there were multiple responses or the response was not clear, the special underscore missing value was used. The statement `missing _` specifies that underscore missing values are to be expected in the data. The `input` statement reads the subject number and the 30 ratings. A name like `Subj001`, `Subj002`, ..., `Subj030` is created from the subject number. If there are any missing data, all data for that subject are excluded by the `if nmiss(of rate:) = 0` statement.

```
title 'Spaghetti Sauces';

data rawdata;
   missing _;
   input subj @5 (rate1-rate30) (1.);
   name = compress('Sub' || put(subj, z3.));
   if nmiss(of rate:) = 0;
   datalines;
  1 319591129691132168146121171191
  2 749173216928911175549891841791
  3 449491116819413186158171961791
   .
   .
   .
 14 1139812951994_9466149198915699
   .
   .
   .
 19 2214922399981121.1116161941991
   .
   .
   .
;
```

Next, the data are transposed from one row per subject and 30 columns to one column per subject and 30 rows, one for each product rated. Then the data are merged with the experimental design.

```
proc transpose data=rawdata(drop=subj) out=temp(drop=_name_);
    id name;
    run;

data inputdata; merge sasuser.spag temp; run;
```

Metric Conjoint Analysis

Next, we use PROC TRANSREG to perform the conjoint analysis.

```
ods exclude notes mvanova anova;
proc transreg data=inputdata utilities short separators='' ', '
   lprefix=0 outtest=utils method=morals;
   title2 'Conjoint Analysis';
   model identity(sub:) =
         class(brand | price meat mushroom ingredients / zero=sum);
   output p ireplace out=results1 coefficients;
   run;
```

The `utilities` option requests conjoint analysis output, and the `short` option suppresses the iteration histories. The `lprefix=0` option specifies that zero variable name characters are to be used to construct the labels for the part-worths; the labels will simply consist of formatted values. The `outtest=` option creates an output SAS data set, `Utils`, that contains all of the statistical results. The `method=morals`, algorithm fits the conjoint analysis model separately for each subject. We specify `ods exclude notes mvanova anova` to exclude ANOVA information (which we usually want to ignore) and provide more parsimonious output.

The `model` statement names the ratings for each subject as dependent variables and the factors as independent variables. Since this is a metric conjoint analysis, `identity` is specified for the ratings. The `identity` transformation is the no-transformation option, which is used for variables that need to enter the model with no further manipulations. The factors are specified as `class` variables, and the `zero=sum` option is specified to constrain the parameter estimates to sum to zero within each effect. The `brand | price` specification asks for a simple `brand` effect, a simple `price` effect, and the `brand * price` interaction.

The `p` option in the `output` statement requests predicted values, the `ireplace` option suppresses the output of transformed independent variables, and the `coefficients` option requests that the part-worth utilities be output. These options control the contents of the `out=results` data set, which contains the ratings, predicted utilities for each product, indicator variables, and the part-worth utilities.

In the interest of space, only the results for the first subject are printed here. Recall that we used an `ods exclude` statement and we used PROC TEMPLATE on page 485 to customize the output from PROC TRANSREG.

Conjoint Analysis

The TRANSREG Procedure

Class Level Information

Class	Levels	Values
Brand	3	Pregu Sundance Tomato Garden
Price	5	1.99 2.29 2.49 2.79 2.99
Meat	3	Vegetarian Meat Italian Sausage
Mushroom	2	Mushrooms No Mention
Ingredients	2	All Natural No Mention

Number of Observations Read	30
Number of Observations Used	30

Conjoint Analysis

The TRANSREG Procedure

Identity(Sub001)
Algorithm converged.

The TRANSREG Procedure Hypothesis Tests for Identity(Sub001)

Root MSE	2.09608	R-Square	0.8344
Dependent Mean	3.73333	Adj R-Sq	0.5635
Coeff Var	56.14499		

Part-Worth Utilities

Label	Utility	Standard Error	Importance (% Utility Range)
Intercept	3.0675	0.45364	
Pregu	2.0903	0.55937	28.924
Sundance	0.2973	0.55886	
Tomato Garden	-2.3876	0.55205	

1.99	-0.6836	0.91331	7.134
2.29	0.3815	0.77035	
2.49	0.4209	0.78975	
2.79	-0.5397	0.79677	
2.99	0.4209	0.78975	
Pregu, 1.99	0.7430	1.09161	15.639
Pregu, 2.29	0.9491	1.13055	
Pregu, 2.49	-0.7433	1.14528	
Pregu, 2.79	-1.0115	1.13157	
Pregu, 2.99	0.0626	1.13769	
Sundance, 1.99	0.0361	1.09135	
Sundance, 2.29	-1.2578	1.09310	
Sundance, 2.49	-0.1443	1.16287	
Sundance, 2.79	1.1633	1.09574	
Sundance, 2.99	0.2027	1.12077	
Tomato Garden, 1.99	-0.7791	1.08788	
Tomato Garden, 2.29	0.3087	1.16798	
Tomato Garden, 2.49	0.8876	1.16026	
Tomato Garden, 2.79	-0.1518	1.10376	
Tomato Garden, 2.99	-0.2654	1.13455	
Vegetarian	2.2828	0.68783	27.813
Meat	-0.2596	0.70138	
Italian Sausage	-2.0231	0.86266	
Mushrooms	1.5514	0.38441	20.042
No Mention	-1.5514	0.38441	
All Natural	-0.0347	0.45814	0.448
No Mention	0.0347	0.45814	

The next steps process the `outtest=` data set, saving the R^2, adjusted R^2, and *df*. Subjects whose adjusted R^2 is less than 0.3 (R^2 approximately 0.73) are flagged for exclusion. We want the final analysis to be based on subjects who seemed to be taking the task seriously. The next steps flag the subjects whose fit seems bad and create a macro variable `&droplist` that contains a list of variables to be dropped from the final analysis.

```
data model;
   set utils;
   if statistic in ('R-Square', 'Adj R-Sq', 'Model');
   Subj = scan(_depvar_, 2);
   if statistic = 'Model' then do;
      value = numdf;
      statistic = 'Num DF';
      output;
      value = dendf;
      statistic = 'Den DF';
      output;
      value = dendf + numdf + 1;
      statistic = 'N';
      end;
   output;
   keep statistic value subj;
   run;
proc transpose data=model out=summ;
   by subj;
   idlabel statistic;
   id statistic;
   run;
data summ2(drop=list);
   length list $ 1000;
   retain list;
   set summ end=eof;
   if adj_r_sq < 0.3 then do;
      Small = '*';
      list = trim(list) || ' ' || subj;
      end;
   if eof then call symput('droplist', trim(list));
   run;
%put &droplist;
proc print label data=summ2(drop=_name_ _label_); run;
```

The `outtest=` data set contains for each subject the ANOVA, R^2, and part-worth utility tables. The numerator *df* is found in the variable `NumDF`, the denominator *df* is found in the variable `DenDF`, and the R^2 and adjusted R^2 are found in the variable `Value`. The first DATA step processes the `outtest=` data set, stores all of the statistics of interest in the variable `Value`, and discards the extra observations and variables. The PROC TRANSPOSE step creates a data set with one observation per subject. Here is the `&droplist` macro variable.

```
Sub011 Sub021 Sub031 Sub051 Sub071 Sub081 Sub092 Sub093 Sub094 Sub096
```

Here is some of the R^2 and *df* summary. We see the *df* are right, and most of the R^2's look good.

Conjoint Analysis

Obs	Subj	Num DF	Den DF	N	R-Square	Adj R-Sq	Small
1	Sub001	18	11	30	0.83441	0.56345	
2	Sub002	18	11	30	0.91844	0.78497	
3	Sub003	18	11	30	0.92908	0.81302	
.							
.							
.							
10	Sub010	18	11	30	0.97643	0.93786	
.							
.							
.							
84	Sub091	18	11	30	0.85048	0.60581	
85	Sub092	18	11	30	0.64600	0.06673	*
86	Sub093	18	11	30	0.45024	-0.44936	*
87	Sub094	18	11	30	0.62250	0.00477	*
88	Sub095	18	11	30	0.85996	0.63081	
89	Sub096	18	11	30	0.73321	0.29664	*
90	Sub097	18	11	30	0.94155	0.84589	
91	Sub099	18	11	30	0.88920	0.70789	
92	Sub100	18	11	30	0.90330	0.74507	

We can run the conjoint again, this time using the `drop=&droplist` data set option to drop the subjects with poor fit. In the interest of space, the `noprint` option was specified on this step. The printed output will be the same as in the previous step, except for the fact that a few subject's tables are deleted.

```
proc transreg data=inputdata(drop=&droplist) utilities short noprint
   separators='' ', ' lprefix=0 outtest=utils method=morals;
   title2 'Conjoint Analysis';
   model identity(sub:) =
         class(brand | price meat mushroom ingredients / zero=sum);
   output p ireplace out=results2 coefficients;
   run;
```

Simulating Market Share

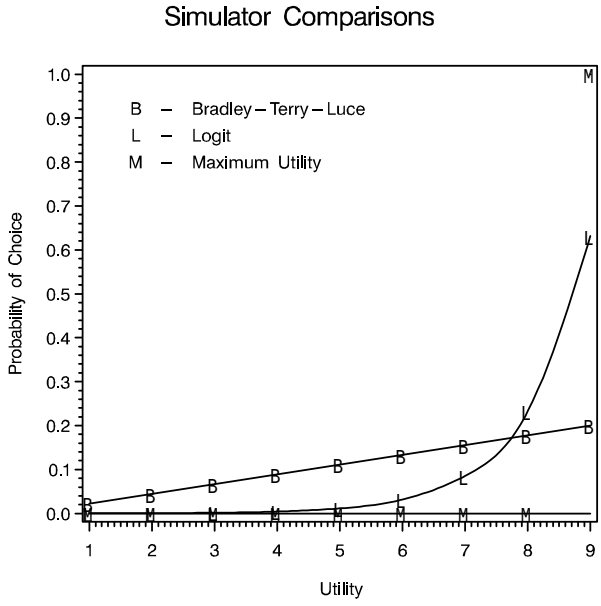
In many conjoint analysis studies, the conjoint analysis is not the primary goal. The conjoint analysis is used to generate part-worth utilities, which are then used as input to consumer choice and market share simulators. The end result for a product is its expected "preference share," which when properly weighted can be used to predict the proportion of times that the product will be purchased. The effects on market share of introducing new products can also be simulated.

One of the most popular ways to simulate market share is with the maximum utility model, which assumes each subject will buy with probability one the product for which he or she has the highest utility. The probabilities for each product are averaged across subjects to get predicted market share.

Other simulation methods include the Bradley-Terry-Luce (BTL) model and the logit model. Unlike the maximum utility model, the BTL and the logit models do not assign all of the probability of choice to the most preferred alternative. Probability is a continuous function of predicted utility. In the maximum utility model, probability of choice is a binary step function of utility. In the BTL model, probability of choice is a linear function of predicted utility. In the logit model, probability of choice is an increasing nonlinear logit function of predicted utility. The BTL model computes the probabilities by dividing each utility by the sum of the predicted utilities within each subject. The logit model divides the exponentiated predicted utilities by the sum of exponentiated utilities, again within subject.

$$\begin{aligned}
\text{Maximum Utility:} \quad p_{ijk} &= 1.0 \quad \text{if } y_{ijk} = \text{MAX}(y_{ijk}), \\
&\quad\ 0.0 \quad \text{otherwise} \\
\text{BTL:} \quad p_{ijk} &= y_{ijk} / \sum\sum\sum y_{ijk} \\
\text{Logit:} \quad p_{ijk} &= \exp(y_{ijk}) / \sum\sum\sum \exp(y_{ijk})
\end{aligned}$$

The following plot shows the different assumptions made by the three choice simulators. This plot shows expected market share for a subject with utilities ranging from one to nine.



The maximum utility line is flat at zero until it reaches the maximum utility, where it jumps to 1.0. The BTL line increases from 0.02 to 0.20 as utility ranges from 1 to 9. The logit function increases exponentially, with small utilities mapping to near-zero probabilities and the largest utility mapping to a proportion of 0.63.

The maximum utility, BTL, and logit models are based on different assumptions and produce different results. The maximum utility model has the advantage of being scale-free. Any strictly monotonic transformation of each subject's predicted utilities will produce the same market share. However, this model is unstable because it assigns a zero probability of choice to all alternatives that do not have the maximum predicted utility, including those that have predicted utilities near the maximum. The disadvantage of the BTL and logit models is that results are not invariant under linear transformations of the predicted utilities. These methods are considered inappropriate by some researchers for this reason. With negative predicted utilities, the BTL method produces negative probabilities, which are

invalid. The BTL results change when a constant is added to the predicted utilities but do not change when a constant is multiplied by the predicted utilities. Conversely, the logit results change when a constant is multiplied by the predicted utilities but do not change when a constant is added to the predicted utilities. The BTL method is not often used in practice, the logit model is sometimes used, and the maximum utility model is most often used. Refer to Finkbeiner (1988) for a discussion of conjoint analysis choice simulators. Do not confuse a logit model choice simulator and the multinomial logit model; they are quite different.

The three simulation methods will produce different results. This is because all three methods make different assumptions about how consumers translate utility into choice. To see why the models differ, imagine a product that is everyone's second choice. Further imagine that there is wide-spread disagreement on first choice. Every other product is someone's first choice, and all other products are preferred about equally often. In the maximum utility model, this second choice product will have zero probability of choice because no one would choose it first. In the other models, it should be the most preferred, because for every individual it will have a high, near-maximum probability of choice. Of course, preference patterns are not usually as weird as the one just described. If consumers are perfectly rational and always choose the alternative with the highest utility, then the maximum utility model is correct. However, you need to be aware that your results will depend on the choice of simulator model and in BTL and logit, the scaling of the utilities. One reason why the discrete choice model is popular in marketing research is discrete choice models choices directly, whereas conjoint simulates choices indirectly.

Here is the code that made the plot. You can try this program with different minima and maxima to see the effects of linear transformations of the predicted utilities.

```
%let min = 1;
%let max = 9;
%let by  = 1;
%let list = &min to &max by &by;
data a;
   sumb = 0;
   suml = 0;
   do u = &list;
      logit = exp(u);
      btl  = u;
      sumb = sumb + btl;
      suml = suml + logit;
      end;
   do u = &list;
      logit = exp(u);
      btl = u;
      max = abs(u - (&max)) < (0.5 * (&by));
      btl = btl / sumb;
      logit = logit / suml;
      output;
      end;
   run;
```

```
goptions ftext=swiss colors=(black) hsize=4.5in vsize=4.5in;
proc gplot;
   title h=1.5 'Simulator Comparisons';
   plot max * u = 1 btl * u = 2 logit * u = 3 /
      vaxis=axis2 haxis=axis1 overlay frame;
   symbol1 v=M i=step;
   symbol2 v=B i=join;
   symbol3 v=L i=spline;
   axis1 order=(&list) label=('Utility');
   axis2 order=(0 to 1 by 0.1)
         label=(angle=90 "Probability of Choice");
   note move=(2.5cm, 9.2cm)
        font=swissu 'B - ' font=swiss 'Bradley-Terry-Luce';
   note move=(2.5cm, 8.7cm)
        font=swissu 'L - ' font=swiss 'Logit';
   note move=(2.5cm, 8.2cm)
        font=swissu 'M - ' font=swiss 'Maximum Utility';
   run; quit;
```

Simulating Market Share, Maximum Utility Model

This section shows how to use the predicted utilities from a conjoint analysis to simulate choice and predict market share. The end result for a hypothetical product is its expected market share, which is a prediction of the proportion of times that the product will be purchased. Note however, that a term like "expected market share," while widely used, is a misnomer. Without purchase volume data, it is unlikely that these numbers would mirror true market share. Nevertheless, conjoint analysis is a useful and popular marketing research technique.

A SAS macro is used to simulate market share. It takes a `method=morals` output data set from PROC TRANSREG and creates a data set with expected market share for each combination. First, market share is computed with the maximum utility model. The macro finds the most preferred combination(s) for each subject, which are those combinations with the largest predicted utility, and assigns the probability that each combination will be purchased. Typically, for each subject, one product will have a probability of purchase of 1.0, and all other produces will have zero probability of purchase. However, when two predicted utilities are tied for the maximum, that subject will have two probabilities of 0.5 and the rest will be zero. The probabilities are averaged across subjects for each product to get market share. Subjects can be differentially weighted.

```
                        /*---------------------------------------*/
                        /* Simulate Market Share                 */
                        /*---------------------------------------*/
%macro sim(data=_last_, /* SAS data set with utilities.         */
           idvars=,     /* Additional variables to display with  */
                        /* market share results.                 */
           weights=,    /* By default, each subject contributes  */
                        /* equally to the market share           */
                        /* computations.  To differentially      */
                        /* weight the subjects, specify a vector */
                        /* of weights, one per subject.          */
                        /* Separate the weights by blanks.       */
           out=shares,  /* Output data set name.                 */
           method=max   /* max   - maximum utility model.        */
                        /* btl   - Bradley-Terry-Luce model.     */
                        /* logit - logit model.                  */
                        /* WARNING: The Bradley-Terry-Luce model */
                        /* and the logit model results are not   */
                        /* invariant under linear                */
                        /* transformations of the utilities.     */
          );            /*---------------------------------------*/
options nonotes;

%if &method = btl or &method = logit %then
   %put WARNING: The Bradley-Terry-Luce model and the logit model
results are not invariant under linear transformations of the
utilities.;
%else %if &method ne max %then %do;
   %put WARNING: Invalid method &method..  Assuming method=max.;
   %let method = max;
   %end;
* Eliminate coefficient observations, if any;
data temp1;
   set &data(where=(_type_ = 'SCORE' or _type_ = ' '));
   run;

* Determine number of runs and subjects.;
proc sql;
   create table temp2 as select nruns,
      count(nruns) as nsubs, count(distinct nruns) as chk
      from (select count( _depvar_) as nruns
      from temp1 where _type_ in ('SCORE', ' ') group by _depvar_);
   quit;
```

```
data _null_;
   set temp2;
   call symput('nruns', compress(put(nruns, 5.0)));
   call symput('nsubs', compress(put(nsubs, 5.0)));
   if chk > 1 then do;
      put 'ERROR: Corrupt input data set.';
      call symput('okay', 'no');
      end;
   else call symput('okay', 'yes');
   run;
%if &okay ne yes %then %do;
   proc print;
     title2 'Number of runs should be constant across subjects';
     run;
   %goto endit;
   %end;

%else %put NOTE: &nruns runs and &nsubs subjects.;
%let w = %scan(&weights, %eval(&nsubs + 1), %str( ));
%if %length(&w) > 0 %then %do;
   %put ERROR: Too many weights.;
   %goto endit;
   %end;

* Form nruns by nsubs data set of utilities;
data temp2;
   keep _u1 - _u&nsubs &idvars;
   array u[&nsubs] _u1 - _u&nsubs;
   do j = 1 to &nruns;

      * Read ID variables;
      set temp1(keep=&idvars) point = j;

      * Read utilities;
      k = j;
      do i = 1 to &nsubs;
         set temp1(keep=p_depend_) point = k;
         u[i] = p_depend_;
         %if &method = logit %then u[i] = exp(u[i]);;
         k = k + &nruns;
         end;

      output;
      end;

   stop;
   run;
```

```
* Set up for maximum utility model;
%if &method = max %then %do;

   * Compute maximum utility for each subject;
   proc means data=temp2 noprint;
      var _u1-_u&nsubs;
      output out=temp1 max=_sum1 - _sum&nsubs;
      run;
   * Flag maximum utility;
   data temp2(keep=_u1 - _u&nsubs &idvars);
      if _n_ = 1 then set temp1(drop=_type_ _freq_);
      array u[&nsubs] _u1 - _u&nsubs;
      array m[&nsubs] _sum1 - _sum&nsubs;
      set temp2;
      do i = 1 to &nsubs;
         u[i] = ((u[i] - m[i]) > -1e-8); /* < 1e-8 is considered 0 */
         end;
      run;

   %end;
* Compute sum for each subject;
proc means data=temp2 noprint;
   var _u1-_u&nsubs;
   output out=temp1 sum=_sum1 - _sum&nsubs;
   run;
* Compute expected market share;
data &out(keep=share &idvars);
   if _n_ = 1 then set temp1(drop=_type_ _freq_);
   array u[&nsubs] _u1 - _u&nsubs;
   array m[&nsubs] _sum1 - _sum&nsubs;
   set temp2;
   * Compute final probabilities;

   do i = 1 to &nsubs;
      u[i] = u[i] / m[i];
      end;
   * Compute expected market share;

   %if %length(&weights) = 0 %then %do;
      Share = mean(of _u1 - _u&nsubs);
      %end;
```

```
      %else %do;
         Share = 0;
         wsum  = 0;
         %do i = 1 %to &nsubs;
            %let w = %scan(&weights, &i, %str( ));
            %if %length(&w) = 0 %then %let w = .;
            if &w < 0 then do;
               if _n_ > 1 then stop;
               put "ERROR: Invalid weight &w..";
               call symput('okay', 'no');
               end;
            share = share + &w * _u&i;
            wsum  = wsum  + &w;
            %end;
         share = share / wsum;
         %end;
      run;
   options notes;

   %if &okay ne yes %then %goto endit;
   proc sort;
      by descending share &idvars;
      run;
   proc print label noobs;
      title2 'Expected Market Share';
      title3 %if       &method = max %then "Maximum Utility Model";
             %else %if &method = btl %then "Bradley-Terry-Luce Model";
             %else                         "Logit Model";;
      run;

   %endit:

   %mend;
   title 'Spaghetti Sauces';

   %sim(data=results2, out=maxutils, method=max,
        idvars=price brand meat mushroom ingredients);
```

Spaghetti Sauces
Expected Market Share
Maximum Utility Model

Brand	Price	Meat	Mushroom	Ingredients	Share
Sundance	1.99	Vegetarian	Mushrooms	No Mention	0.18293
Pregu	1.99	Vegetarian	No Mention	All Natural	0.14228
Tomato Garden	2.29	Italian Sausage	Mushrooms	No Mention	0.12195
Pregu	2.29	Vegetarian	No Mention	No Mention	0.10976
Pregu	1.99	Vegetarian	Mushrooms	No Mention	0.10366
Tomato Garden	1.99	Vegetarian	Mushrooms	No Mention	0.09146
Tomato Garden	1.99	Vegetarian	No Mention	All Natural	0.07520
Sundance	2.29	Vegetarian	Mushrooms	All Natural	0.07317
Sundance	1.99	Vegetarian	No Mention	All Natural	0.05081
Pregu	2.29	Meat	Mushrooms	All Natural	0.02439
Sundance	2.29	Meat	No Mention	No Mention	0.01220
Sundance	2.49	Italian Sausage	No Mention	No Mention	0.01220
Tomato Garden	2.29	Vegetarian	No Mention	All Natural	0.00000
Pregu	2.49	Vegetarian	Mushrooms	No Mention	0.00000
Pregu	2.49	Italian Sausage	No Mention	No Mention	0.00000
Sundance	2.49	Meat	Mushrooms	All Natural	0.00000
Tomato Garden	2.49	Vegetarian	Mushrooms	All Natural	0.00000
Tomato Garden	2.49	Meat	No Mention	No Mention	0.00000
Pregu	2.79	Meat	No Mention	No Mention	0.00000
Pregu	2.79	Italian Sausage	Mushrooms	No Mention	0.00000
Sundance	2.79	Vegetarian	Mushrooms	No Mention	0.00000
Sundance	2.79	Meat	No Mention	All Natural	0.00000
Tomato Garden	2.79	Vegetarian	No Mention	No Mention	0.00000
Tomato Garden	2.79	Meat	Mushrooms	All Natural	0.00000
Pregu	2.99	Meat	Mushrooms	All Natural	0.00000
Pregu	2.99	Italian Sausage	No Mention	No Mention	0.00000
Sundance	2.99	Vegetarian	No Mention	All Natural	0.00000
Sundance	2.99	Meat	Mushrooms	No Mention	0.00000
Tomato Garden	2.99	Vegetarian	No Mention	No Mention	0.00000
Tomato Garden	2.99	Italian Sausage	Mushrooms	No Mention	0.00000

The largest market share (18.29%) is for Sundance brand vegetarian sauce with mushrooms costing $1.99. The next largest share (14.23%) is Pregu brand vegetarian sauce with all natural ingredients costing $1.99. Five of the seven most preferred sauces all cost $1.99−the minimum. It is not clear from this simulation if any brand is the leader.

Simulating Market Share, Bradley-Terry-Luce and Logit Models

The Bradley-Terry-Luce model and the logit model are also available in the %SIM macro.

```
title 'Spaghetti Sauces';

%sim(data=results2, out=btl, method=btl,
     idvars=price brand meat mushroom ingredients);

%sim(data=results2, out=logit, method=logit,
     idvars=price brand meat mushroom ingredients);
```

Spaghetti Sauces
Expected Market Share
Bradley-Terry-Luce Model

Brand	Price	Meat	Mushroom	Ingredients	Share
Pregu	1.99	Vegetarian	Mushrooms	No Mention	0.053479
Sundance	1.99	Vegetarian	Mushrooms	No Mention	0.052990
Tomato Garden	1.99	Vegetarian	Mushrooms	No Mention	0.051751
Pregu	1.99	Vegetarian	No Mention	All Natural	0.050683
Sundance	1.99	Vegetarian	No Mention	All Natural	0.050193
Tomato Garden	1.99	Vegetarian	No Mention	All Natural	0.048955
Sundance	2.29	Vegetarian	Mushrooms	All Natural	0.048236
Pregu	2.29	Vegetarian	No Mention	No Mention	0.043972
Tomato Garden	2.29	Vegetarian	No Mention	All Natural	0.042035
Pregu	2.49	Vegetarian	Mushrooms	No Mention	0.041532
Pregu	2.29	Meat	Mushrooms	All Natural	0.041063
Sundance	2.29	Meat	No Mention	No Mention	0.036321
Tomato Garden	2.29	Italian Sausage	Mushrooms	No Mention	0.032995
Sundance	2.79	Vegetarian	Mushrooms	No Mention	0.032067
Sundance	2.49	Meat	Mushrooms	All Natural	0.031310
Tomato Garden	2.49	Vegetarian	Mushrooms	All Natural	0.031057
Sundance	2.99	Vegetarian	No Mention	All Natural	0.026879
Pregu	2.49	Italian Sausage	No Mention	No Mention	0.026046
Pregu	2.99	Meat	Mushrooms	All Natural	0.025318
Pregu	2.79	Meat	No Mention	No Mention	0.025038
Tomato Garden	2.79	Vegetarian	No Mention	No Mention	0.024325
Pregu	2.79	Italian Sausage	Mushrooms	No Mention	0.024263
Sundance	2.49	Italian Sausage	No Mention	No Mention	0.022383
Sundance	2.99	Meat	Mushrooms	No Mention	0.022264
Tomato Garden	2.99	Vegetarian	No Mention	No Mention	0.022113
Sundance	2.79	Meat	No Mention	All Natural	0.021858
Tomato Garden	2.79	Meat	Mushrooms	All Natural	0.021415
Tomato Garden	2.49	Meat	No Mention	No Mention	0.019142
Pregu	2.99	Italian Sausage	No Mention	No Mention	0.016391
Tomato Garden	2.99	Italian Sausage	Mushrooms	No Mention	0.013926

Spaghetti Sauces
Expected Market Share
Logit Model

Brand	Price	Meat	Mushroom	Ingredients	Share
Sundance	1.99	Vegetarian	Mushrooms	No Mention	0.10463
Pregu	1.99	Vegetarian	No Mention	All Natural	0.09621
Tomato Garden	1.99	Vegetarian	Mushrooms	No Mention	0.09001
Pregu	1.99	Vegetarian	Mushrooms	No Mention	0.08358
Pregu	2.29	Vegetarian	No Mention	No Mention	0.07755
Sundance	2.29	Vegetarian	Mushrooms	All Natural	0.07102
Tomato Garden	1.99	Vegetarian	No Mention	All Natural	0.06872
Tomato Garden	2.29	Italian Sausage	Mushrooms	No Mention	0.06735
Sundance	1.99	Vegetarian	No Mention	All Natural	0.06419
Pregu	2.29	Meat	Mushrooms	All Natural	0.04137
Pregu	2.49	Vegetarian	Mushrooms	No Mention	0.03578
Sundance	2.29	Meat	No Mention	No Mention	0.03273
Sundance	2.49	Italian Sausage	No Mention	No Mention	0.02081
Tomato Garden	2.99	Italian Sausage	Mushrooms	No Mention	0.02055
Sundance	2.79	Vegetarian	Mushrooms	No Mention	0.02022
Tomato Garden	2.29	Vegetarian	No Mention	All Natural	0.01996
Pregu	2.79	Italian Sausage	Mushrooms	No Mention	0.01233
Pregu	2.49	Italian Sausage	No Mention	No Mention	0.01199
Sundance	2.49	Meat	Mushrooms	All Natural	0.01010
Sundance	2.99	Meat	Mushrooms	No Mention	0.00964
Pregu	2.79	Meat	No Mention	No Mention	0.00763
Pregu	2.99	Italian Sausage	No Mention	No Mention	0.00637
Pregu	2.99	Meat	Mushrooms	All Natural	0.00547
Tomato Garden	2.49	Vegetarian	Mushrooms	All Natural	0.00538
Tomato Garden	2.79	Meat	Mushrooms	All Natural	0.00516
Sundance	2.99	Vegetarian	No Mention	All Natural	0.00399
Sundance	2.79	Meat	No Mention	All Natural	0.00266
Tomato Garden	2.79	Vegetarian	No Mention	No Mention	0.00209
Tomato Garden	2.99	Vegetarian	No Mention	No Mention	0.00162
Tomato Garden	2.49	Meat	No Mention	No Mention	0.00088

The three methods produce different results.

Change in Market Share

The following steps simulate what would happen to the market if new products were introduced. Simulation observations are added to the data set and given zero weight. The conjoint analyses are rerun to compute the predicted utilities for the active observations and the simulations. The maximum utility model is used.

Recall that the design has numeric variables with values like 1, 2, and 3. Formats are used to print the descriptions of the levels of the attributes. The first thing we want to do is read in products

to simulate. We could read in values like 1, 2, and 3 or we could read in more descriptive values and convert them to numerics using informats. We chose the latter approach. First we use PROC FORMAT to create the informats. Previously, we created formats with PROC FORMAT by specifying a `value` statement followed by pairs of the form *numeric-value=descriptive-character-string*. We create an informat with PROC FORMAT by specifying an `invalue` statement followed by pairs of the form *descriptive-character-string=numeric-value*.

```
title 'Spaghetti Sauces';

proc format;
   invalue inbrand 'Preg'=1 'Sun' =2 'Tom' =3;
   invalue inmeat  'Veg' =1 'Meat'=2 'Ital'=3;
   invalue inmush  'Mush'=1 'No'  =2;
   invalue iningre 'Nat' =1 'No'  =2;
   invalue inprice '1.99'=1 '2.29'=2 '2.49'=3 '2.79'=4 '2.99'=5;
   run;
```

Next, we read the observations we want to consider for a sample market using the informats we just created. An `input` statement specification of the form "*variable* : *informat*" reads values starting with the first nonblank character.

```
data simulat;
   input brand       : inbrand.
         meat        : inmeat.
         mushroom    : inmush.
         ingredients : iningre.
         price       : inprice.;
   datalines;
Preg  Veg   Mush  Nat  1.99
Sun   Veg   Mush  Nat  1.99
Tom   Veg   Mush  Nat  1.99
Preg  Meat  Mush  Nat  2.49
Sun   Meat  Mush  Nat  2.49
Tom   Meat  Mush  Nat  2.49
Preg  Ital  Mush  Nat  2.79
Sun   Ital  Mush  Nat  2.79
Tom   Ital  Mush  Nat  2.79
;
```

Next, the original input data set is combined with the simulation observations. The subjects with poor fit are dropped and a `weight` variable is created to flag the simulation observations. The `weight` variable is not strictly necessary since all of the simulation observations will have missing values on the ratings so will be excluded from the analysis that way. Still, it is good practice to explicitly use weights to exclude observations.

```
data inputdata2(drop=&droplist);
   set inputdata(in=w) simulat;
   Weight = w;
   run;
```

```
proc print;
   title2 'Simulation Observations Have a Weight of Zero';
   id weight;
   var brand -- price;
   run;
```

Spaghetti Sauces
Simulation Observations Have a Weight of Zero

Weight	Brand	Meat	Mushroom	Ingredients	Price
1	Pregu	Meat	No Mention	No Mention	2.79
1	Tomato Garden	Vegetarian	No Mention	No Mention	2.79
1	Pregu	Meat	Mushrooms	All Natural	2.29
1	Tomato Garden	Vegetarian	Mushrooms	All Natural	2.49
1	Sundance	Vegetarian	Mushrooms	No Mention	1.99
1	Pregu	Italian Sausage	No Mention	No Mention	2.49
1	Tomato Garden	Vegetarian	No Mention	No Mention	2.99
1	Tomato Garden	Italian Sausage	Mushrooms	No Mention	2.29
1	Pregu	Vegetarian	Mushrooms	No Mention	2.49
1	Pregu	Vegetarian	No Mention	No Mention	2.29
1	Sundance	Vegetarian	Mushrooms	No Mention	2.79
1	Tomato Garden	Vegetarian	Mushrooms	No Mention	1.99
1	Sundance	Meat	No Mention	No Mention	2.29
1	Sundance	Meat	Mushrooms	No Mention	2.99
1	Pregu	Italian Sausage	Mushrooms	No Mention	2.79
1	Tomato Garden	Italian Sausage	Mushrooms	No Mention	2.99
1	Sundance	Vegetarian	Mushrooms	All Natural	2.29
1	Pregu	Meat	Mushrooms	All Natural	2.99
1	Tomato Garden	Meat	No Mention	No Mention	2.49
1	Sundance	Meat	Mushrooms	All Natural	2.49
1	Pregu	Vegetarian	No Mention	All Natural	1.99
1	Sundance	Meat	No Mention	All Natural	2.79
1	Tomato Garden	Vegetarian	No Mention	All Natural	1.99
1	Sundance	Italian Sausage	No Mention	No Mention	2.49
1	Sundance	Vegetarian	No Mention	All Natural	1.99
1	Sundance	Vegetarian	No Mention	All Natural	2.99
1	Pregu	Italian Sausage	No Mention	No Mention	2.99
1	Tomato Garden	Vegetarian	No Mention	All Natural	2.29
1	Pregu	Vegetarian	Mushrooms	No Mention	1.99
1	Tomato Garden	Meat	Mushrooms	All Natural	2.79
0	Pregu	Vegetarian	Mushrooms	All Natural	1.99
0	Sundance	Vegetarian	Mushrooms	All Natural	1.99
0	Tomato Garden	Vegetarian	Mushrooms	All Natural	1.99
0	Pregu	Meat	Mushrooms	All Natural	2.49
0	Sundance	Meat	Mushrooms	All Natural	2.49

```
0      Tomato Garden    Meat              Mushrooms     All Natural    2.49
0      Pregu            Italian Sausage   Mushrooms     All Natural    2.79
0      Sundance         Italian Sausage   Mushrooms     All Natural    2.79
0      Tomato Garden    Italian Sausage   Mushrooms     All Natural    2.79
```

The next steps run the conjoint analyses suppressing the printed output using the `noprint` option. The statement `weight weight` is specified since we want the simulation observations (which have zero weight) excluded from contributing to the analysis. However, the procedure will still compute an expected utility for every observation including observations with zero, missing, and negative weights. The `outtest=` data set is created like before so we can check to make sure the *df* and R^2 look reasonable.

```
ods exclude notes mvanova anova;
proc transreg data=inputdata2 utilities short noprint
   separators=', ' lprefix=0 method=morals outtest=utils;
   title2 'Conjoint Analysis';
   model identity(sub:) =
         class(brand | price meat mushroom ingredients / zero=sum);
   output p ireplace out=results3 coefficients;
   weight weight;
   run;
data model;
   set utils;
   if statistic in ('R-Square', 'Adj R-Sq', 'Model');
   Subj = scan(_depvar_, 2);
   if statistic = 'Model' then do;
      value = numdf;
      statistic = 'Num DF';
      output;
      value = dendf;
      statistic = 'Den DF';
      output;
      value = dendf + numdf + 1;
      statistic = 'N';
      end;
   output;
   keep statistic value subj;
   run;
proc transpose data=model out=summ;
   by subj;
   idlabel statistic;
   id statistic;
   run;
proc print label data=summ(drop=_name_ _label_); run;
```

The SAS log tells us that the nine simulation observations were deleted both because of zero weight and because of missing values in the dependent variables.

```
NOTE: 9 observations were deleted from the analysis but not from the
      output data set due to missing values.
NOTE: 9 observations were deleted from the analysis but not from the
      output data set due to nonpositive weights.
NOTE: A total of 9 observations were deleted.
```

The *df* and R^2 results, some of which are shown next, look fine.

Spaghetti Sauces
Conjoint Analysis

Obs	Subj	Num DF	Den DF	N	R-Square	Adj R-Sq
1	Sub001	18	11	30	0.83441	0.56345
2	Sub002	18	11	30	0.91844	0.78497
3	Sub003	18	11	30	0.92908	0.81302
.						
.						
.						
81	Sub099	18	11	30	0.88920	0.70789
82	Sub100	18	11	30	0.90330	0.74507

The simulation observations are pulled out of the `out=` data set, and the %SIM macro is run to simulate market share.

```
data results4;
   set results3;
   where weight = 0;
   run;
%sim(data=results4, out=shares2, method=max,
     idvars=price brand meat mushroom ingredients);
```

Spaghetti Sauces
Expected Market Share
Maximum Utility Model

Brand	Price	Meat	Mushroom	Ingredients	Share
Pregu	1.99	Vegetarian	Mushrooms	All Natural	0.35976
Sundance	1.99	Vegetarian	Mushrooms	All Natural	0.29878
Tomato Garden	1.99	Vegetarian	Mushrooms	All Natural	0.19512
Tomato Garden	2.79	Italian Sausage	Mushrooms	All Natural	0.08537
Sundance	2.79	Italian Sausage	Mushrooms	All Natural	0.02439
Pregu	2.49	Meat	Mushrooms	All Natural	0.01220
Sundance	2.49	Meat	Mushrooms	All Natural	0.01220
Pregu	2.79	Italian Sausage	Mushrooms	All Natural	0.01220
Tomato Garden	2.49	Meat	Mushrooms	All Natural	0.00000

For this set of products, the inexpensive vegetarian sauces have the greatest market share with Pregu brand preferred over Sundance and Tomato Garden. Now we'll consider adding six more products to the market, the six meat sauces we just saw, but at a lower price.

```
data simulat2;
   input brand       : inbrand.
         meat        : inmeat.
         mushroom    : inmush.
         ingredients : iningre.
         price       : inprice.;
   datalines;
Preg Meat Mush Nat 2.29
Sun  Meat Mush Nat 2.29
Tom  Meat Mush Nat 2.29
Preg Ital Mush Nat 2.49
Sun  Ital Mush Nat 2.49
Tom  Ital Mush Nat 2.49
;
data inputdata3(drop=&droplist);
   set inputdata(in=w) simulat simulat2;
   weight = w;
   run;
ods exclude notes mvanova anova;
proc transreg data=inputdata3 utilities short noprint
   separators=', ' lprefix=0 method=morals outtest=utils;
   title2 'Conjoint Analysis';
   model identity(sub:) =
         class(brand | price meat mushroom ingredients / zero=sum);
   output p ireplace out=results5 coefficients;
   weight weight;
   run;
```

Now we see that 15 simulation observations were excluded.

```
NOTE: 15 observations were deleted from the analysis but not from the
      output data set due to missing values.
NOTE: 15 observations were deleted from the analysis but not from the
      output data set due to nonpositive weights.
NOTE: A total of 15 observations were deleted.
```

These steps extract the *df* and R^2.

```
data model;
   set utils;
   if statistic in ('R-Square', 'Adj R-Sq', 'Model');
   Subj = scan(_depvar_, 2);
   if statistic = 'Model' then do;
      value = numdf;
      statistic = 'Num DF';
      output;
      value = dendf;
      statistic = 'Den DF';
      output;
      value = dendf + numdf + 1;
      statistic = 'N';
      end;
   output;
   keep statistic value subj;
   run;
proc transpose data=model out=summ;
   by subj;
   idlabel statistic;
   id statistic;
   run;
proc print label data=summ(drop=_name_ _label_); run;
```

The *df* and R^2 still look fine.

Spaghetti Sauces
Conjoint Analysis

Obs	Subj	Num DF	Den DF	N	R-Square	Adj R-Sq
1	Sub001	18	11	30	0.83441	0.56345
2	Sub002	18	11	30	0.91844	0.78497
3	Sub003	18	11	30	0.92908	0.81302
.						
.						
.						
81	Sub099	18	11	30	0.88920	0.70789
82	Sub100	18	11	30	0.90330	0.74507

Now we'll run the simulation with all 15 simulation observations.

```
data results6;
   set results5;
   where weight = 0;
   run;

%sim(data=results6, out=shares3, method=max,
     idvars=price brand meat mushroom ingredients);
```

Spaghetti Sauces
Expected Market Share
Maximum Utility Model

Brand	Price	Meat	Mushroom	Ingredients	Share
Sundance	1.99	Vegetarian	Mushrooms	All Natural	0.25813
Pregu	1.99	Vegetarian	Mushrooms	All Natural	0.20935
Pregu	2.29	Meat	Mushrooms	All Natural	0.19512
Tomato Garden	1.99	Vegetarian	Mushrooms	All Natural	0.15447
Sundance	2.49	Italian Sausage	Mushrooms	All Natural	0.08537
Sundance	2.29	Meat	Mushrooms	All Natural	0.03659
Tomato Garden	2.49	Italian Sausage	Mushrooms	All Natural	0.01829
Tomato Garden	2.29	Meat	Mushrooms	All Natural	0.01220
Pregu	2.49	Italian Sausage	Mushrooms	All Natural	0.01220
Tomato Garden	2.79	Italian Sausage	Mushrooms	All Natural	0.01220
Sundance	2.79	Italian Sausage	Mushrooms	All Natural	0.00610
Pregu	2.49	Meat	Mushrooms	All Natural	0.00000
Sundance	2.49	Meat	Mushrooms	All Natural	0.00000
Tomato Garden	2.49	Meat	Mushrooms	All Natural	0.00000
Pregu	2.79	Italian Sausage	Mushrooms	All Natural	0.00000

These steps merge the data set containing the old market shares with the data set containing the new market shares to show the effect of adding the new products.

```
title 'Spaghetti Sauces';

proc sort data=shares2;
   by price brand meat mushroom ingredients;
   run;
proc sort data=shares3;
   by price brand meat mushroom ingredients;
   run;
data both;
   merge shares2(rename=(share=OldShare)) shares3;
   by price brand meat mushroom ingredients;
   if oldshare = . then Change = 0;
   else change = oldshare;
   change = share - change;
   run;
```

```
proc sort;
   by descending share price brand meat mushroom ingredients;
   run;
options missing=' ';
proc print noobs;
   title2 'Expected Market Share and Change';
   var price brand meat mushroom ingredients
       oldshare share change;
   format oldshare -- change 6.3;
   run;
options missing=.;
```

Spaghetti Sauces
Expected Market Share and Change

Price	Brand	Meat	Mushroom	Ingredients	Old Share	Share	Change
1.99	Sundance	Vegetarian	Mushrooms	All Natural	0.299	0.258	-0.041
1.99	Pregu	Vegetarian	Mushrooms	All Natural	0.360	0.209	-0.150
2.29	Pregu	Meat	Mushrooms	All Natural		0.195	0.195
1.99	Tomato Garden	Vegetarian	Mushrooms	All Natural	0.195	0.154	-0.041
2.49	Sundance	Italian Sausage	Mushrooms	All Natural		0.085	0.085
2.29	Sundance	Meat	Mushrooms	All Natural		0.037	0.037
2.49	Tomato Garden	Italian Sausage	Mushrooms	All Natural		0.018	0.018
2.29	Tomato Garden	Meat	Mushrooms	All Natural		0.012	0.012
2.49	Pregu	Italian Sausage	Mushrooms	All Natural		0.012	0.012
2.79	Tomato Garden	Italian Sausage	Mushrooms	All Natural	0.085	0.012	-0.073
2.79	Sundance	Italian Sausage	Mushrooms	All Natural	0.024	0.006	-0.018
2.49	Pregu	Meat	Mushrooms	All Natural	0.012	0.000	-0.012
2.49	Sundance	Meat	Mushrooms	All Natural	0.012	0.000	-0.012
2.49	Tomato Garden	Meat	Mushrooms	All Natural	0.000	0.000	0.000
2.79	Pregu	Italian Sausage	Mushrooms	All Natural	0.012	0.000	-0.012

We see that the vegetarian sauces are most preferred, but we predict they would lose share if the new meat sauces were entered in the market. In particular, the Sundance and Pregu meat sauces would gain significant market share under this model.

PROC TRANSREG Specifications

PROC TRANSREG (transformation regression) is used to perform conjoint analysis and many other types of analyses, including simple regression, multiple regression, redundancy analysis, canonical correlation, analysis of variance, and external unfolding, all with nonlinear transformations of the variables. This section documents the statements and options available in PROC TRANSREG that are commonly used in conjoint analyses. Refer to "The TRANSREG Procedure" in the *SAS/STAT User's Guide* for more information on PROC TRANSREG. This section documents only a small subset of the capabilities of PROC TRANSREG.

The following statements are used in the TRANSREG procedure for conjoint analysis:

```
PROC TRANSREG <DATA=SAS-data-set> <OUTTEST=SAS-data-set>
              <a-options> <o-options>;
   MODEL transform(dependents   </ t-options>) =
         transform(independents </ t-options>)
        <transform(independents </ t-options>) ...> </ a-options>;
   OUTPUT <OUT=SAS-data-set> <o-options>;
   WEIGHT variable;
   ID variables;
   BY variables;
```

Specify the `proc` and `model` statements to use PROC TRANSREG. The `output` statement is required to produce an `out=` output data set, which contains the transformations, indicator variables, and predicted utility for each product. The `outtest=` data set, which contains the ANOVA, regression, and part-worth utility tables, is requested in the `proc` statement. All options can be abbreviated to their first three letters.

PROC TRANSREG *Statement*

```
PROC TRANSREG <DATA=SAS-data-set> <OUTTEST=SAS-data-set>
              <a-options> <o-options>;
```

The `data=` and `outtest=` options can appear only in the PROC TRANSREG statement. The algorithm options *(a-options)* appear in the `proc` or `model` statement. The output options *(o-options)* can appear in the `proc` or `output` statement.

`DATA=`*SAS-data-set*
specifies the input SAS data. If the `data=` option is not specified, PROC TRANSREG uses the most recently created SAS data set.

`OUTTEST=`*SAS-data-set*
specifies an output data set that will contain the ANOVA table, R^2, and the conjoint analysis part-worth utilities, and the attribute importances.

Algorithm Options

```
PROC TRANSREG <DATA=SAS-data-set> <OUTTEST=SAS-data-set>
              <a-options> <o-options>;
   MODEL transform(dependents   </ t-options>) =
         transform(independents </ t-options>)
        <transform(independents </ t-options>) ...> </ a-options>;
```

Algorithm options can appear in the `proc` or `model` statement as *a-options*.

`CONVERGE=`*n*
specifies the minimum average absolute change in standardized variable scores that is required to continue iterating. By default, `converge=0.00001`.

`DUMMY`
requests a canonical initialization. When `spline` transformations are requested, specify `dummy` to solve for the optimal transformations without iteration. Iteration is only necessary when there are monotonicity constraints.

`LPREFIX=`*n*
specifies the number of first characters of a `class` variable's label (or name if no label is specified) to use in constructing labels for part-worth utilities. For example, the default label for `Brand=Duff` is "Brand Duff". If you specify `lprefix=0` then the label is simply "Duff".

`MAXITER=`*n*
specifies the maximum number of iterations. By default, `maxiter=30`.

`NOPRINT`
suppresses the display of all output.

`ORDER=FORMATTED`
`ORDER=INTERNAL`
specifies the order in which the CLASS variable levels are reported. The default, `order=internal`, sorts by unformatted value. Specify `order=formatted` when you want the levels sorted by formatted value. Sort order is machine dependent. Note that in Version 6 and Version 7 of the SAS System, the default sort order was `order=formatted`. The default was changed to `order=internal` in Version 8 to be consistent with Base SAS procedures.

`METHOD=MORALS`
`METHOD=UNIVARIATE`
specifies the iterative algorithm. Both `method=morals` and `method=univariate` fit univariate multiple regression models with the possibility of nonlinear transformations of the variables. They differ in the way they structure the output data set when there is more than one dependent variable. When it can be used, `method=univariate` is more efficient than `method=morals`.

You can use `method=univariate` when no transformations of the independent variables are requested, for example when the independent variables are all designated `class`, `identity`, or `pspline`. In this case, the final set of independent variables will be the same for all subjects. If transformations such as

`monotone`, `identity`, `spline` or `mspline` are specified for the independent variables, the transformed independent variables may be different for each dependent variable and so must be output separately for each dependent variable. In conjoint analysis, there will typically be one dependent variable for each subject. This is illustrated in the examples.

With `method=univariate` and more than one dependent variable, PROC TRANSREG creates a data set with the same number of score observations as the original but with more variables. The untransformed dependent variable names are unchanged. The default transformed dependent variable names consist of the prefix "T" and the original variable names. The default predicted value names consist of the prefix "P" and the original variable names. The full set of independent variables appears once.

When more than one dependent variable is specified, `method=morals` creates a *rolled-out* data set with the dependent variable in `_depend_`, its transformation in `t_depend_`, and its predicted values in `p_depend_`. The full set of independents is repeated for each (original) dependent variable.

The procedure chooses a default method based on what is specified in the `model` statement. When transformations of the independent variables are requested, the default method is `morals`. Otherwise the default method is `univariate`.

`SEPARATORS=`*string-1 <string-2 >*
specifies separators for creating labels for the part-worth utilities. By default, `separators=' ' ' * '` ("blank" and "blank asterisk blank"). The first value is used to separate variable names and values in interactions. The second value is used to separate interaction components. For example, the default label for `Brand=Duff` is "Brand Duff". If you specify `separators=', '` then the label is "Brand, Duff". Furthermore, the default label for the interaction of `Brand=Duff` and `Price=3.99` is "Brand Duff * Price 3.99". You could specify `lprefix=0` and `separators='' ' @ '` to instead create labels like "Duff @ 3.99". You use the `lprefix=0` option when you want to construct labels using zero characters of the variable name, that is when you want to construct labels from just the formatted level. The option `separators='' ' @ '` specifies in the second string a separator of the form "blank at blank". In this case, the first string is ignored because with `lprefix=0` there is no name to separate from the level.

`SHORT`
suppresses the iteration histories. For most standard metric conjoint analyses, no iterations are necessary, so specifying `short` eliminates unnecessary output. PROC TRANSREG will print a message if it ever fails to converge, so it is usually safe to specify the `short` option.

`UTILITIES`
prints the part-worth utilities and importances table and an ANOVA table. Note that you can use an `ods exclude` statement to exclude ANOVA tables and unnecessary notes from the conjoint output (see page 486).

Output Options

```
PROC TRANSREG <DATA=SAS-data-set> <OUTTEST=SAS-data-set>
              <a-options> <o-options>;
   OUTPUT <OUT=SAS-data-set> <o-options>;
```

The `out=` option can only appear in the `output` statement. The other output options can appear in the `proc` or `output` statement as *o-options*.

COEFFICIENTS
outputs the part-worth utilities to the out= data set.

P
includes the predicted values in the out= output data set, which are the predicted utilities for each product. By default, the predicted values variable name is the original dependent variable name prefixed with a "P".

IREPLACE
replaces the original independent variables with the transformed independent variables in the output data set. The names of the transformed variables in the output data set correspond to the names of the original independent variables in the input data set.

OUT=*SAS-data-set*
names the output data set. When an output statement is specified without the out= option, PROC TRANSREG creates a data set and uses the DATA*n* convention. To create a permanent SAS data set, specify a two-level name. The data set will contain the original input variables, the coded indicator variables, the transformation of the dependent variable, and the optionally predicted utilities for each product.

RESIDUALS
outputs to the out= data set the differences between the observed and predicted utilities. By default, the residual variable name is the original dependent variable name prefixed with an "R".

Transformations and Expansions

```
MODEL transform(dependents   </ t-options>) =
      transform(independents </ t-options>)
     <transform(independents </ t-options>) ...> </ a-options>;
```

The operators "*", "|", and "@" from the GLM procedure are available for interactions with class variables.

```
class(a * b ...
      c | d ...
      e | f ... @ n)
```

For example, this statement fits 100 individual main-effects models:

```
model identity(rating1-rating100) = class(x1-x5 / zero=sum);
```

This fits models with main effects and all two-way interactions:

```
model identity(rating1-rating100) = class(x1|x2|x3|x4|x5@2 / zero=sum);
```

This fits models with main effects and some two-way interactions:

```
model identity(rating1-rating100) = class(x1-x5 x1*x2 x3*x4 / zero=sum);
```

You can also fit separate price functions within each brand by specifying:

```
model identity(rating1-rating100) =
      class(brand / zero=none) | spline(price);
```

The list `x1-x5` is equivalent to `x1 x2 x3 x4 x5`. The vertical bar specifies all main effects and interactions, and the at sign limits the interactions. For example, `@2` limits the model to main effects and two-way interactions. The list `x1|x2|x3|x4|x5@2` is equivalent to `x1 x2 x1 * x2 x3 x1 * x3 x2 * x3 x4 x1 * x4 x2 * x4 x3 * x4 x5 x1 * x5 x2 * x5 x3 * x5 x4 * x5`. The specification `x1 * x2` indicates the two-way interaction between `x1` and `x2`, and `x1 * x2 * x3` indicates the three-way interaction between `x1`, `x2`, and `x3`.

Each of the following can be specified in the `model` statement as a *transform*. The `pspline` and `class` expansions create more than one output variable for each input variable. The rest are transformations that create one output variable for each input variable.

`CLASS`
designates variables for analysis as nominal-scale-of-measurement variables. For conjoint analysis, the `zero=sum` *t-option* is typically specified: `class(`*variables* `/ zero=sum)`. Variables designated as `class` variables are expanded to a set of indicator variables. Usually the number output variables for each `class` variable is the number of different values in the input variables. Dependent variables should not be designated as `class` variables.

`IDENTITY`
variables are not changed by the iterations. The `identity(`*variables*`)` specification designates interval-scale-of-measurement variables when no transformation is permitted. When small data values mean high preference, you will need to use the `reflect` transformation option.

`MONOTONE`
monotonically transforms variables; ties are preserved. When `monotone(`*variables*`)` is used with dependent variables, a nonmetric conjoint analysis is performed. When small data values mean high preference, you will need to use the `reflect` transformation option. The `monotone` specification can also be used with independent variables to impose monotonicity on the part-worth utilities. When it is known that monotonicity should exist in an attribute variable, using `monotone` instead of `class` for that attribute may improve prediction. An option exists in PROC TRANSREG for optimally untying tied values, but this option should not be used because it almost always produces a degenerate result.

`MSPLINE`
monotonically and smoothly transforms variables. By default, `mspline(`*variables)* fits a monotonic quadratic spline with no knots. Knots are specified as *t-options*, for example `mspline(`*variables* `/ nknots=3)` or `mspline(`*variables* `/ knots=5 to 15 by 5)`. Like `monotone`, `mspline`, finds a monotonic transformation. Unlike `monotone`, `mspline` places a bound on the *df* (number of knots + degree) used by the transformation. With `mspline`, it is possible to allow for nonlinearity in the responses and still have error *df*. This is not always possible with `monotone`. When small data values mean high preference, you will need to use the `reflect` transformation option. You can also use `mspline` with attribute variables to impose monotonicity on the part-worth utilities.

PSPLINE
expands each variable to a piece-wise polynomial spline basis. By default, `pspline(`*variables*`)` uses a cubic spline with no knots. Knots are specified as *t-options*. Specify `pspline(`*variable* `/ degree=2)` for an attribute variable to fit a quadratic model. For each `pspline` variable, $d + k$ output variables are created, where d is the degree of the polynomial and k is the number of knots. You should not specify `pspline` with the dependent variables.

RANK
performs a rank transformation, with ranks averaged within ties. Rating-scale data can be transformed to ranks by specifying `rank(`*variables*`)`. When small data values mean high preference, you will need to use the `reflect` transformation option. Typically, `rank` is only used for dependent variables. For example, if a rating-scale variable has sorted values 1, 1, 1, 2, 3, 3, 4, 5, 5, 5, then the rank transformation is 2, 2, 2, 4, 5.5, 5.5, 7, 9, 9, 9. A conjoint analysis of the original rating-scale variable will not usually be the same as a conjoint analysis of a rank transformation of the ratings. With ordinal-scale-of-measurement data, it is often good to analyze rank transformations instead of the original data. An alternative is to specify `monotone`, which performs a nonmetric conjoint analysis. For real data, `monotone` will always find a better fit than `rank`, but `rank` may lead to better prediction.

SPLINE
smoothly transforms variables. By default, `spline(`*variables*`)` fits a cubic spline with no knots. Knots are specified as *t-options*. Like `pspline`, `spline` models nonlinearities in the attributes.

Transformation Options

```
MODEL transform(dependents   </ t-options>) =
      transform(independents </ t-options>)
     <transform(independents </ t-options>) ...> </ a-options>;
```

The following are specified in the `model` statement as *t-options*'s.

DEGREE=*n*
specifies the degree of the spline. The defaults are `degree=3` (cubic spline) for `spline` and `pspline`, and `degree=2` (quadratic spline) for `mspline`. For example, to request a quadratic spline, specify `spline(`*variables* `/ degree=2)`.

EVENLY
is used with the `nknots=` option to evenly space the knots for splines. For example, if `spline(x / nknots=2 evenly)` is specified and `x` has a minimum of 4 and a maximum of 10, then the two interior knots are 6 and 8. Without `evenly`, the `nknots=` option places knots at percentiles, so the knots are not evenly spaced.

KNOTS=*numberlist*
specifies the interior knots or break points for splines. By default, there are no knots. For example, to request knots at 1, 2, 3, 4, 5, specify `spline(`*variable* `/ knots=1 to 5)`.

`NKNOTS=`*k*
creates *k* knots for splines: the first at the $100/(k+1)$ percentile, the second at the $200/(k+1)$ percentile, and so on. Unless `evenly` is specified, knots are placed at data values; there is no interpolation. For example, with `spline(`*variable* `/ NKNOTS=3)`, knots are placed at the twenty-fifth percentile, the median, and the seventy-fifth percentile. By default, `nknots=0`.

`REFLECT`
reflects the transformation around its mean, $Y = -(Y - \bar{Y}) + \bar{Y}$, after the iterations are completed and before the final standardization and results calculations. This option is particularly useful with the dependent variable. When the dependent variable consists of ranks with the most preferred combination assigned 1.0, `identity(`*variable* `/ reflect)` will reflect the transformation so that positive utilities mean high preference.

`ZERO=SUM`
constrains the part-worth utilities to sum to zero within each attribute. The specification `class(`*variables* `/ zero=sum)` creates a less than full rank model, but the coefficients are uniquely determined due to the sum-to-zero constraint.

BY *Statement*

```
BY variables;
```

A `by` statement can be used with PROC TRANSREG to obtain separate analyses on observations in groups defined by the `by` variables. When a `by` statement appears, the procedure expects the input data set to be sorted in order of the `by` variables.

If the input data set is not sorted in ascending order, use one of the following alternatives:

- Use the SORT procedure with a similar `by` statement to sort the data.
- Use the `by` statement options `notsorted` or `descending` in the `by` statement for the TRANSREG procedure. As a cautionary note, the `notsorted` option does not mean that the data are unsorted. It means that the data are arranged in groups (according to values of the `by` variables), and these groups are not necessarily in alphabetical or increasing numeric order.
- Use the DATASETS procedure (in base SAS software) to create an index on the `by` variables.

For more information on the `by` statement, refer to the discussion in *SAS Language: Reference*. For more information on the DATASETS procedure, refer to the discussion in *SAS Procedures Guide*.

ID *Statement*

```
ID variables;
```

The `id` statement includes additional character or numeric variables from the input data set in the `out=` data set.

WEIGHT *Statement*

```
WEIGHT variable;
```

A `weight` statement can be used in conjoint analysis to distinguish ordinary active observations, holdouts, and simulation observations. When a `weight` statement is used, a weighted residual sum of squares is minimized. The observation is used in the analysis only if the value of the `weight` statement variable is greater than zero. For observations with positive weight, the `weight` statement has no effect on *df* or number of observations, but the weights affect most other calculations.

Assign each active observation a weight of 1. Assign each holdout observation a weight that excludes it from the analysis, such as missing. Assign each simulation observation a different weight that excludes it from the analysis, such as zero. Holdouts are rated by the subjects and so have nonmissing values in the dependent variables. Simulation observations are not rated and so have missing values in the dependent variable. It is useful to create a format for the `weight` variable that distinguishes the three types of observations in the input and output data sets.

```
proc format;
   value wf 1  = 'Active'
            .  = 'Holdout'
            0  = 'Simulation';
   run;
```

PROC TRANSREG does not distinguish between weights that are zero, missing, or negative. All nonpositive weights exclude the observations from the analysis. The holdout and simulation observations are given different nonpositive values and a format to make them easy to distinguish in subsequent analyses and listings. The part-worth utilities for each attribute are computed using only those observations with positive weight. The predicted utility is computed for all products, even those with nonpositive weights.

Monotone, Spline, and Monotone Spline Comparisons

When you choose the transformation of the ratings or rankings, you choose among

`identity` - model the data directly

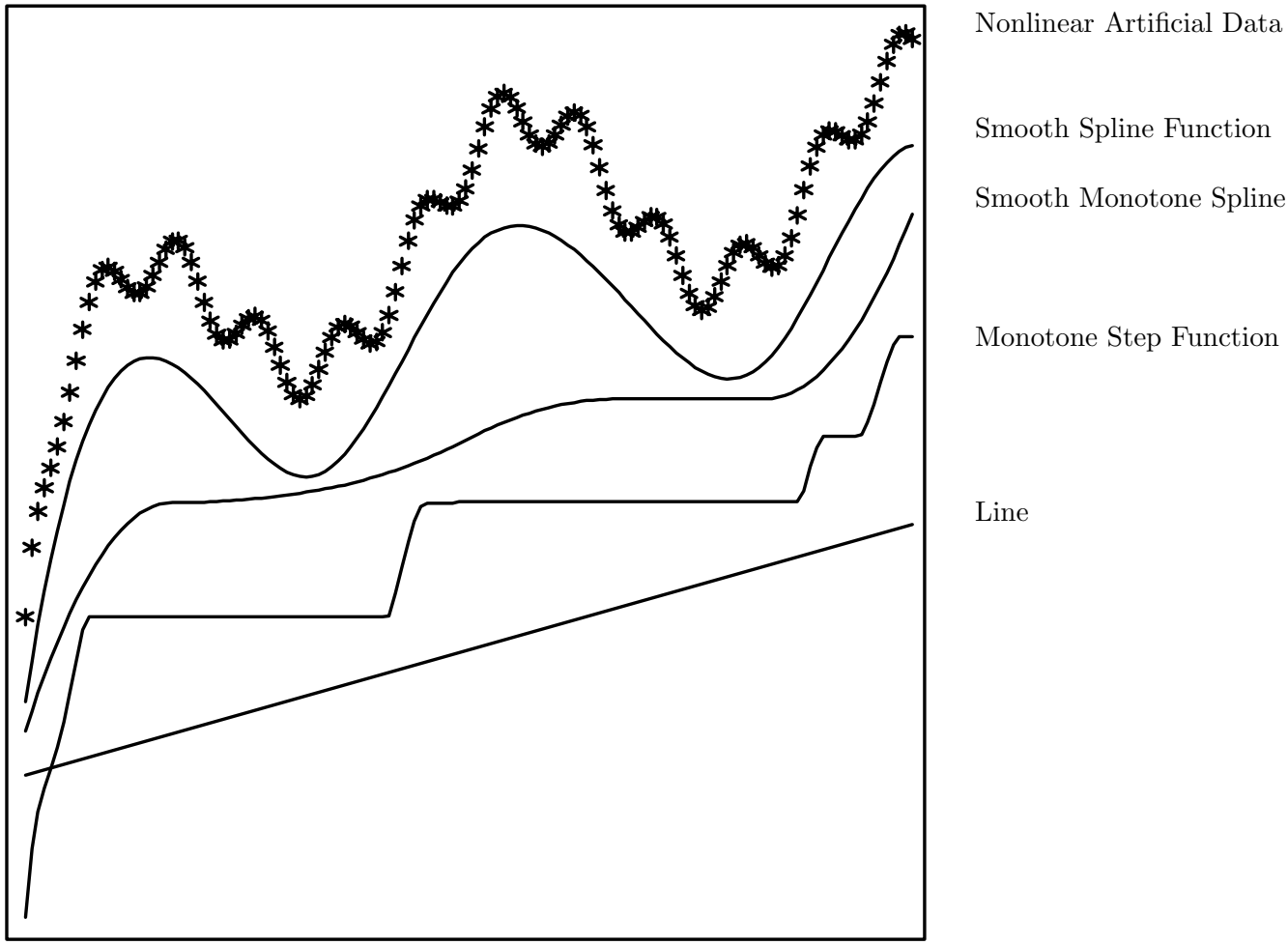
`monotone` - model an increasing step function of the data

`mspline` - model a nonlinear but smooth and increasing function of the data

`spline` - model a smooth function of the data

The following plot shows examples of the different types of functions you can fit in PROC TRANSREG. At the top of the plot are some artificial nonlinear data. Below that is a spline function, created by `spline`. It is smooth and nonlinear. It follows the overall shape of the data, but smoothes out the smaller bumps. Below that is a monotone spline function, created by `mspline`. Like the spline function, it is smooth and nonlinear. Unlike the spline function, it is monotonic. The function never decreases; it always rises or stays flat. The monotone spline function follows the overall upward trend in the data, and it shows the changes in upward trend, but it smoothes out all the dips and bumps in the function. Below the monotone spline function is a monotone step function, created by `monotone`. It is not smooth, but it is monotonic. Like the monotone spline, the monotone step function follows the

Functions Available in PROC TRANSREG



overall upward trend in the data, and it smoothes out all the dips and bumps in the function. However, the function is not smooth, and it typically requires many more parameters be fit than with monotone splines. Below the monotone step function is a line, created by `identity`. It is smooth and linear. It follows the overall upward trend in the data, but it smoothes over all the dips, bumps, and changes in upward trend.

Typical conjoint analyses are metric (using `identity`) or nonmetric (using `monotone`). While not often used in practice, monotone splines have a lot to recommend them. They allow for nonlinearities in the transformation of preference, but unlike `monotone`, they are smooth and do not use up all of your error *df*. One would typically never use `spline` on the ratings or rankings in a conjoint analysis, but if for some reason, you had a lot of price points,[¶] you could fit a spline function of the price attribute. This would allow for nonlinearities in preferences for different prices while constraining the part-worth utility function to be smooth.

[¶]For design efficiency reasons, you typically should not.

Samples of PROC TRANSREG Usage

Conjoint analysis can be performed in many ways with PROC TRANSREG. This section provides sample specifications for some typical and some more esoteric conjoint analyses. The dependent variables typically contain ratings or rankings of products by a number of subjects. The independent variables, `x1-x5`, are the attributes. For metric conjoint analysis, the dependent variable is designated `identity`. For nonmetric conjoint analysis, `monotone` is used. Attributes are usually designated as `class` variables with the restriction that the part-worth utilities within each attribute sum to zero.

The `utilities` option requests an overall ANOVA table, a table of part-worth utilities, their standard errors, and the importance of each attribute. The `p` (predicted values) option outputs to a data set the predicted utility for each product. The `ireplace` option suppresses the separate output of transformed independent variables since the independent variable transformations are the same as the raw independent variables. The `weight` variable is used to distinguish active observations from holdouts and simulation observations. The `reflect` transformation option reflects the transformation of the ranking so that large transformed values, positive utility, and positive evaluation will all correspond.

Today, metric conjoint analysis is used more often than nonmetric conjoint analysis, and rating-scale data are collected more often than rankings.

Metric Conjoint Analysis with Rating-Scale Data

This is a metric conjoint analysis with rating-scale data.

```
ods exclude notes mvanova anova;
proc transreg data=a utilities short method=morals;
   model identity(rating1-rating100) = class(x1-x5 / zero=sum);
   output p ireplace;
   weight w;
   run;
```

Nonmetric Conjoint Analysis

This is a nonmetric conjoint analysis specification, which has *many* parameters for the transformations.

```
ods exclude notes anova liberalanova conservanova
            mvanova liberalmvanova conservmvanova;
proc transreg data=a utilities short maxiter=500 method=morals;
   model monotone(ranking1-ranking100 / reflect) = class(x1-x5 / zero=sum);
   output p ireplace;
   weight w;
   run;
```

Monotone Splines

This is a conjoint analysis that is more restrictive than a nonmetric analysis but less restrictive than a metric conjoint analysis. By default, the monotone spline transformation has two parameters (degree two with no knots).

```
ods exclude notes anova liberalanova conservanova
            mvanova liberalmvanova conservmvanova;
proc transreg data=a utilities short maxiter=500 method=morals;
   model mspline(ranking1-ranking100 / reflect) =
         class(x1-x5 / zero=sum);
   output p ireplace;
   weight w;
   run;
```

If less smoothness is desired, specify knots. For example:

```
ods exclude notes anova liberalanova conservanova
            mvanova liberalmvanova conservmvanova;
proc transreg data=a utilities short maxiter=500 method=morals;
   model mspline(ranking1-ranking100 / reflect nknots=3) =
         class(x1-x5 / zero=sum);
   output p ireplace;
   weight w;
   run;
```

Each knot requires estimation of an additional parameter.

Constraints on the Utilities

Here is a metric conjoint analysis specification with linearity constraints imposed on `x4` and monotonicity constraints imposed on `x5`.

```
ods exclude notes anova liberalanova conservanova
            mvanova liberalmvanova conservmvanova
            liberalutilities liberalfitstatistics;
proc transreg data=a utilities short maxiter=500 method=morals;
   model identity(rating1-rating100) = class(x1-x3 / zero=sum)
         identity(x4) monotone(x5);
   output p ireplace;
   weight w;
   run;
```

With the monotonic constraints on the part-worth utilities, PROC TRANSREG prints some extra information, liberal and conservative part-worth utility and fit statistics tables. These tables report the same part-worth utilities, but are based on different methods of counting the number of parameters estimated. The liberal test tables can be suppressed by adding `liberalutilities liberalfitstatistics` to the `ods exclude` statement.

Here is another example, specifying monotonic step-function constraints on `x1-x5` and a smooth, monotonic transformation of price:

```
ods exclude notes anova liberalanova conservanova
            mvanova liberalmvanova conservmvanova
            liberalutilities liberalfitstatistics;
proc transreg data=a utilities short maxiter=500 method=morals;
   model identity(rating1-rating100) = monotone(x1-x5) mspline(price);
   output p ireplace;
   weight w;
   run;
```

A Discontinuous Price Function

The utility of price may not be a continuous function of price. It has been frequently found that utility is discontinuous at *round* numbers such as $1.00, $2.00, $100, $1000, and so on. If `price` has many values in the data set, say over the range $1.05 to $3.95, then a monotone function of price with discontinuities at $2.00 and $3.00 can be requested as follows.

```
ods exclude notes anova liberalanova conservanova
            mvanova liberalmvanova conservmvanova
            liberalutilities liberalfitstatistics;
proc transreg data=a utilities short maxiter=500 method=morals;
   model identity(rating1-rating100) =
         class(x1-x5 / zero=sum)
         mspline(price / knots=2 2 2 3 3 3);
   output p ireplace;
   weight w;
   run;
```

The monotone spline is degree two. The *order* of the spline is one greater than the degree; in this case the order is three. When the same knot value is specified *order* times, the transformation is discontinuous at the knot. Refer to page 785, for some applications of splines to conjoint analysis.

Experimental Design and Choice Modeling Macros

Warren F. Kuhfeld

Abstract

SAS provides a set of macros for designing experiments and analyzing choice data. Syntax and usage of these macros is discussed in this chapter. An additional macro for scatter plots of labeled points is documented here as well.*

Introduction

The following SAS autocall macros are available.

Macro	Page	Required Products	Purpose
`%ChoiceEff`	600	STAT, IML	efficient choice design
`%MktAllo`	632		processes allocation data
`%MktBal`	635	STAT, QC	balanced main-effects designs
`%MktBlock`	638	STAT, IML	block a linear or choice design
`%MktDes`	648	STAT, QC	efficient factorial design via candidate set search
`%MktDups`	655		identify duplicate choice sets or runs
`%MktEval`	663	STAT, IML	evaluate an experimental design
`%MktEx`	667	STAT, IML, QC	efficient factorial design
`%MktKey`	710		aid creation of the `key=` data set
`%MktLab`	712	STAT	relabel, rename and assign levels to design factors
`%MktMerge`	723		merges a choice design with choice data
`%MktOrth`	725	IML	lists 116,590 entry orthogonal array catalog
`%MktPPro`	731	STAT, IML	partial profiles through BIB designs
`%MktRoll`	735		rolls a linear design into a choice design
`%MktRuns`	740		selecting experimental design number of runs
`%PhChoice`	748		customizes the printed output from a choice model
`%PlotIt`	753	STAT, GRAPH	graphical scatter plots of labeled points

*Copies of this chapter (TS-722I) and all of the macros are available on the web http://support.sas.com/techsup/tnote/tnote_stat.html#market.

These macros are used for generating efficient factorial designs, efficient choice designs, processing and evaluating designs, processing data from choice experiments, and for certain graphical displays. The BASE product is required to run all macros along with any additional indicated products. To run the full suite of macros you need to have BASE, SAS/STAT, SAS/QC and SAS/IML installed, and you need SAS/GRAPH for the plotting macro. Once you have the right products installed, and you have installed the macros, you can call them and use them just as you would use a SAS procedure. However, the syntax for macros is a little different from SAS procedures. Here is an example of making an experimental design with a two-level and 7 three-level factors with 18 runs or profiles.

```
%mktex( 2 3 ** 7, n=18 )
```

Changes and Enhancements

With this release, the orthogonal array catalog has almost quadrupled in size, and the `%PlotIt` macro has a new default style.

Installation

If your site has installed the autocall libraries supplied by SAS and uses the standard configuration of SAS supplied software, you need to ensure that the SAS system option `mautosource` is in effect to begin using the autocall macros. Note however, that there are differences between the macros used in this documentation and those that were shipped with your Version of SAS. Be sure to get the latest macros from the Web http://support.sas.com/techsup/tnote/tnote_stat.html#market. These macros will not work with Version 6.12, however they should work with Version 8.2 and later versions of SAS. You should install *all* of these macros, not just one or some. Some of the macros call other macros and will not work if the other macros are not there or if only older versions of the other macros are there. For example, the `%MktEx` macro calls: the `%MktRuns` macro to parse the factors list, the `%MktDes` macro for candidate set generation and search, and the `%MktOrth` macro to get the orthogonal array catalog.

The macros do not have to be included (for example, with a `%include` statement). They can be called directly once they are properly installed. For more information about autocall libraries, refer to *SAS Macro Language: Reference*. On a PC for example, the autocall library may be installed in the `stat\sasmacro` directory off of your SAS root directory. The name of your SAS root directory could be anything, but it is probably something like SAS or SAS\V8, or SAS\V9, and so on. One way to find the right directory is to use `Start` → `Find` to find one of the existing autocall macros such as `mktdes.sas` or `plotit.sas`. Here are some examples of SAS/STAT autocall macro directories on a PC.

```
C:\Program Files\SAS Institute\SAS\V8\stat\sasmacro
C:\Program Files\SAS\SAS 9.1\stat\sasmacro
```

Unix should have a similar directory structure to the PC. The autocall library in Unix may be installed in the `stat/sasmacro` directory off of your SAS root directory. On MVS, each macro will be a different member of a PDS. For details on installing autocall macros, consult your host documentation.

Usually, an autocall library is a directory containing individual files, each of which contains one macro definition. An autocall library can also be a SAS catalog. To use a directory as a SAS autocall library, store the source code for each macro in a separate file in the directory. The name of the file must be the

same as the macro name, typically followed by `.sas`. For example, the macro `%MktEx` must typically be stored in a file named `mktex.sas`. On most hosts, the reserved `fileref sasautos` is assigned at invocation time to the autocall library supplied by SAS or another one designated by your site.

The libraries that SAS searches for autocall macros are controlled by the SASAUTOS option. The default value for this option is set in the configuration file. One way to have SAS find your autocall macros is to update this option in the SAS configuration file. For example, searching for SAS*.CFG may turn up SASV9.CFG or SASV8.CFG that on a PC will contain lines like this. The directories listed in this option will depend on what SAS products you have licensed.

```
/* Setup the SAS System autocall library definition */
-SET SASAUTOS  (
                "!sasroot\core\sasmacro"
                "!sasext0\cpe\sasmacro"
                "!sasext0\risk\sasmacro"
                "!sasext0\dmine\sasmacro"
                "!sasext0\access\sasmacro"
                "!sasext0\assist\sasmacro"
                "!sasext0\eis\sasmacro"
                "!sasext0\ets\sasmacro"
                "!sasext0\gis\sasmacro"
                "!sasext0\graph\sasmacro"
                "!sasext0\iml\sasmacro"
                "!sasext0\intrnet\sasmacro"
                "!sasext0\or\sasmacro"
                "!sasext0\qc\sasmacro"
                "!sasext0\share\sasmacro"
                "!sasext0\stat\sasmacro"
                "!sasext0\webhound\sasmacro"
                "C:\Program Files\SAS Institute\SAS\V8\test\sasmacro"
               )
```

If you replace the existing macros in `stat\sasmacro`, you should never have to change this file. However, if you install the new macros anywhere else, you need to ensure that that location appears in your configuration file **before** `stat\sasmacro` or you will not get all of the most recent macros. For details, refer to your host documentation and SAS macro language documentation.

%ChoicEff Macro

The %ChoicEff autocall macro is used to find efficient experimental designs for choice experiments. See pages 321, 363, 399, and 406 for examples. You supply sets of candidate alternatives. The macro searches the candidates for an efficient experimental design−a design in which the variances of the parameter estimates are minimized, given an assumed parameter vector $\boldsymbol{\beta}$.

There are two ways you can use the macro:

- You can create a candidate set of alternatives, and the macro will create a design consisting of choice sets built from the alternatives you supplied. You must designate for each candidate alternative the design alternative(s) for which it is a candidate. For a branded study with m brands, you must create m lists of candidate alternatives, one for each brand.

- You can create a candidate set of choice sets, and the macro will build a design from the choice sets that you supplied. Typically, you would only use this approach when there are restrictions across alternatives (certain alternatives may not appear with certain other alternatives) and with partial-profile designs.

The %ChoicEff macro uses a modified Fedorov candidate-set-search algorithm, just like PROC OPTEX and the %MktEx macro. Typically, you use as a candidate set a full-factorial, fractional-factorial, or a tabled design created with the %MktEx macro. First, the %ChoicEff macro either constructs a random initial design from the candidates or it uses an initial design that you specified. The macro considers swapping out every design alternative/set and replacing it with each candidate alternative/set. Swaps that increase efficiency are performed. The process of evaluating and swapping continues until efficiency stabilizes. This process is repeated with different initial designs, and the best design is output for use. The key differences between the %ChoicEff macro and the %MktEx macro are as follows. The %ChoicEff macro requires you to specify the true (or assumed true) parameters and it optimizes the variance matrix for a multinomial logit model, whereas the %MktEx macro optimizes the variance matrix for a linear model, which does not depend on the parameters.

Here is an example. This example creates a design for a generic model with 3 three-level factors. First, the %MktEx macro is used to create a set of candidate alternatives, where x1-x3 are the factors. Note that the n= specification allows expressions. Our candidate set must also contain flag variables, one for each alternative, that flag which candidates can be used for which alternative(s). Since this is a generic model, each candidate can appear in any alternative, so we need to add flags that are constant: f1=1 f2=1 f3=1. The %MktEx macro does not allow you to create constant factors. Instead, we can use the %MktLab macro to add the flag variables, essentially by specifying that we have three intercepts. The option int=f1-f3 creates three variables with values all one. A 1 in variable f1 indicates that the candidate can be used for the first alternative, a 1 in f2 indicates that the candidate can be used for the second alternative, and so on. In this case, all candidates can be used for all alternatives, otherwise the flag variables would contain zeros for candidates that cannot be used for certain alternatives. The default output data set from the %MktLab macro is called FINAL. This code makes the candidate set.

```
%mktex(3 ** 3, n=3**3, seed=238)
%mktlab(int=f1-f3)

proc print; run;
```

Here is the candidate set.

Obs	f1	f2	f3	x1	x2	x3
1	1	1	1	3	3	3
2	1	1	1	3	2	1
3	1	1	1	1	3	1
4	1	1	1	2	2	3
5	1	1	1	1	2	2
6	1	1	1	3	1	2
7	1	1	1	3	1	2
8	1	1	1	2	3	2
9	1	1	1	2	2	3
10	1	1	1	3	3	3
11	1	1	1	3	1	2
12	1	1	1	2	2	3
13	1	1	1	2	3	2
14	1	1	1	2	1	1
15	1	1	1	2	3	2
16	1	1	1	1	2	2
17	1	1	1	2	1	1
18	1	1	1	1	3	1
19	1	1	1	2	1	1
20	1	1	1	1	2	2
21	1	1	1	3	3	3
22	1	1	1	1	3	1
23	1	1	1	1	1	3
24	1	1	1	3	2	1
25	1	1	1	3	2	1
26	1	1	1	1	1	3
27	1	1	1	1	1	3

Next, the `%ChoicEff` macro is run to find an efficient design for the unbranded, purely generic model assuming $\boldsymbol{\beta} = \mathbf{0}$. Here is the code.

```
%choiceff(data=final, model=class(x1-x3), nsets=9,
          flags=f1-f3, beta=zero, seed=145, maxiter=100)

proc print; var x1-x3; id set; by set; run;
```

The option `data=final` names the input data set, `model=class(x1-x3)` specifies the PROC TRANSREG `model` statement for coding the design, `nsets=9` specifies nine choice sets, `flags=f1-f3` specifies the three alternative flag variables, `beta=zero` specifies all zero parameters, and `seed=145` specifies the random number seed, and `maxiter=100` specifies the number of designs to create. Here is part of the output.

n	Name	Beta	Label
1	x11	0	x1 1
2	x12	0	x1 2
3	x21	0	x2 1
4	x22	0	x2 2
5	x31	0	x3 1
6	x32	0	x3 2

Design	Iteration	D-Efficiency	D-Error
1	0	0.945985	1.057100
	1	1.634565	0.611784
	2	1.702866	0.587245
	3	1.702866	0.587245

.
.
.

Design	Iteration	D-Efficiency	D-Error
84	0	0.870220	1.149135
	1	1.608264	0.621789
	2	1.732051	0.577350
	3	1.732051	0.577350

.
.
.

Final Results

Design	84
Choice Sets	9
Alternatives	3
D-Efficiency	1.732051
D-Error	0.577350

n	Variable Name	Label	Variance	DF	Standard Error
1	x11	x1 1	0.66667	1	0.81650
2	x12	x1 2	0.66667	1	0.81650
3	x21	x2 1	0.66667	1	0.81650
4	x22	x2 2	0.66667	1	0.81650
5	x31	x3 1	0.66667	1	0.81650
6	x32	x3 2	0.66667	1	0.81650
				==	
				6	

Set	x1	x2	x3
1	2	1	2
	1	2	3
	3	3	1
2	3	2	2
	1	1	1
	2	3	3
3	2	3	1
	3	2	2
	1	1	3
4	1	2	2
	2	3	3
	3	1	1
5	1	1	3
	2	2	1
	3	3	2
6	1	3	1
	2	1	2
	3	2	3
7	3	1	1
	2	2	3
	1	3	2
8	2	2	1
	3	1	3
	1	3	2
9	2	1	2
	3	3	3
	1	2	1

The output from the `%ChoicEff` macro consists of a list of the parameter names, values and labels, followed by two iteration histories (each based on a different random initial design), then a brief report on the most efficient design found, and finally a table with the parameter names, variances, *df*, and standard errors. The design is printed using PROC PRINT.

Here is another example. These next steps directly create an optimal design for this generic model and evaluate its efficiency using the `%ChoicEff` macro and the initial design options. The DATA step creates a cyclic design. In a cyclic design, the factor levels increase cyclically from one alternative to the next. The levels for a factor for the three alternatives will always be one of the following: (1, 2, 3) or (2, 3, 1) or (3, 1, 2).

```
* Cyclic (Optimal) Design;
data x(keep=f1-f3 x1-x3);
   retain f1-f3 1;
   d1 = ceil(_n_ / 3); d2 = mod(_n_ - 1, 3) + 1; input d3 @@;
   do i = -1 to 1;
      x1 = mod(d1 + i, 3) + 1;
      x2 = mod(d2 + i, 3) + 1;
      x3 = mod(d3 + i, 3) + 1;
      output;
      end;
   datalines;
1 2 3 3 1 2 2 3 1
;

   proc print data=x; var x: f:; run;
```

Here is part of the cyclic design. Notice the cyclical pattern. Each level in the second or third alternative is one greater than the level in the previous alternative, where 3+1 is defined to be 1. The flag variables `f1-f3` contain all ones showing that each candidate can be used in any alternative.

Obs	x1	x2	x3	f1	f2	f3
1	1	1	1	1	1	1
2	2	2	2	1	1	1
3	3	3	3	1	1	1
4	1	2	2	1	1	1
5	2	3	3	1	1	1
6	3	1	1	1	1	1
.						
.						
.						
25	3	3	1	1	1	1
26	1	1	2	1	1	1
27	2	2	3	1	1	1

This is the code that evaluates the design.

```
%choiceff(data=x, model=class(x1-x3), nsets=9, flags=f1-f3,
          beta=zero, init=x, initvars=x1-x3, intiter=0)
```

The option `init=x` specifies the initial design, `initvars=x1-x3` specifies the factors in the initial design, and `intiter=0` specifies the number of internal iterations. Specify `intiter=0` when you just want to evaluate the efficiency of a given design. Here is the output from the `%ChoicEff` macro.

n	Name	Beta	Label
1	x11	0	x1 1
2	x12	0	x1 2
3	x21	0	x2 1
4	x22	0	x2 2
5	x31	0	x3 1
6	x32	0	x3 2

Design	Iteration	D-Efficiency	D-Error
1	0	1.732051	0.577350

Final Results

Design	1
Choice Sets	9
Alternatives	3
D-Efficiency	1.732051
D-Error	0.577350

n	Variable Name	Label	Variance	DF	Standard Error
1	x11	x1 1	0.66667	1	0.81650
2	x12	x1 2	0.66667	1	0.81650
3	x21	x2 1	0.66667	1	0.81650
4	x22	x2 2	0.66667	1	0.81650
5	x31	x3 1	0.66667	1	0.81650
6	x32	x3 2	0.66667	1	0.81650
				==	
				6	

These next steps use the `%MktEx` and `%MktRoll` macros to create a candidate set of choice sets and the `%ChoiceEff` macro to search for an efficient design using the candidate-set-swapping algorithm.

```
%mktex(3 ** 9, n=2187)

data key;
   input (x1-x3) ($);
   datalines;
x1 x2 x3
x4 x5 x6
x7 x8 x9
;
%mktroll(design=design, key=key, out=rolled)

%choiceff(data=rolled, model=class(x1-x3), nsets=9, nalts=3,
          beta=zero, seed=17)
```

The first steps create a candidate set of choice sets. The %MktEx macro creates a design with nine factors, three for each of the three alternatives. The KEY data set specifies that the first alternative is made from the linear design factors x1-x3, the second alternative is made from x4-x6, and the third alternative is made from x7-x9. The %MktRoll macro turns a linear design into a choice design using the rules specified in the KEY data set.

In the %ChoicEff macro, the nalts=3 option specifies that there are three alternatives. There must always be a constant number of alternatives in each choice set, even if all of the alternatives will not be used. When a nonconstant number of alternatives is desired, you must use a weight variable to flag those alternatives that the subject will not see. When you swap choice sets, you need to specify nalts=. The output from these steps is not appreciably different from what we saw previously, so it is not shown.

This next example has brand effects and uses the alternative-swapping algorithm.

```
%mktex(3 ** 4, n = 3**4)
%mktlab(data=design, vars=x1-x3 Brand)

data full(drop=i);
   set final;
   array f[3];
   do i = 1 to 3; f[i] = (brand eq i); end;
   run;

proc print data=full(obs=9); run;
```

The %MktEx macro makes the linear candidate design. The %MktLab macro changes the name of the variable x4 to Brand while retaining the original names for x1-x3 and original values for all factors of 1, 2, and 3. The DATA step creates the flags. The flag f1 flags brand 1 candidates as available for the first alternative, f2 flags brand 2 candidates as available for the second alternative, and so on. The Boolean expression (brand eq i) evaluates to 1 if true and 0 if false. Here is the first part of the candidate set.

Obs	x1	x2	x3	Brand	f1	f2	f3
1	1	1	1	1	1	0	0
2	1	1	1	2	0	1	0
3	1	1	1	3	0	0	1
4	1	1	2	1	1	0	0
5	1	1	2	2	0	1	0
6	1	1	2	3	0	0	1
7	1	1	3	1	1	0	0
8	1	1	3	2	0	1	0
9	1	1	3	3	0	0	1

Here is the %ChoicEff macro call for making the design.

```
%choiceff(data=full, seed=151,
          model=class(brand brand*x1 brand*x2 brand*x3 / zero=' '),
          nsets=15, flags=f1-f3, beta=zero, converge=1e-12);
```

The `model=` specification states that `Brand` and `x1-x3` are classification or categorical variables and brand effects and brand by attribute interactions (alternative-specific effects) (see page 222) are desired. The `zero=' '` specification is like `zero=none` except `zero=none` applies to all factors in the specification whereas `zero=' '` applies to just the first. This `zero=' '` specification specifies that there is no reference level for the first factor (`Brand`), and last level will by default be the reference category for the other factors (`x1-x3`). Hence, binary variables are created for all three brands, but only two binary variables are created for the 3 three-level factors. We need to do this because we need the alternative-specific effects for all brands, including Brand 3. Notice that the candidate set consists of branded alternatives with flags such that only brand n is considered for the nth alternative of each choice set. In the interest of space, not all of the output is shown. Here is some of the output.

```
Design   Iteration  D-Efficiency       D-Error
-----------------------------------------------
   1         0                 0             .
             1                 0             .
               0.300256 (Ridged)
             2                 0             .
               0.302184 (Ridged)
             3                 0             .
               0.303659 (Ridged)
             4                 0             .
               0.305192 (Ridged)
             5                 0             .
               0.305192 (Ridged)

Design   Iteration  D-Efficiency       D-Error
-----------------------------------------------
   2         0                 0             .
             1                 0             .
               0.295570 (Ridged)
             2                 0             .
               0.303106 (Ridged)
             .
             .
             .
             7                 0             .
               0.304929 (Ridged)

                 Final Results

          Design                 1
          Choice Sets           15
          Alternatives           3
          D-Efficiency           0
          D-Error                .
```

n	Variable Name	Label	Variance	DF	Standard Error
1	Brand1	Brand 1	5.70845	1	2.38924
2	Brand2	Brand 2	3.89246	1	1.97293
3	Brand3	Brand 3	.	0	.
4	Brand1x11	Brand 1 * x1 1	1.99395	1	1.41207
5	Brand1x12	Brand 1 * x1 2	2.09289	1	1.44668
6	Brand2x11	Brand 2 * x1 1	1.80635	1	1.34400
7	Brand2x12	Brand 2 * x1 2	2.09299	1	1.44672
8	Brand3x11	Brand 3 * x1 1	2.12281	1	1.45699
9	Brand3x12	Brand 3 * x1 2	2.16706	1	1.47209
.					
.					
.					
21	Brand3x32	Brand 3 * x3 2	2.18010	1	1.47652
				==	
				20	

The following is printed to the log.

```
Redundant Variables:

Brand3
```

Notice that at each step, the efficiency is zero, but a nonzero ridged value is printed. This model contains a structural-zero coefficient in `Brand3`. While we need alternative-specific effects for Brand 3 (like `Brand3x11` and `Brand3x12`), we do not need the Brand 3 effect (`Brand3`) This can be seen from both the `Redundant Variables` list and from looking at the variance and *df* table. The inclusion of the `Brand3` term in the model makes the efficiency of the design zero. However, the `%ChoicEff` macro can still optimize the goodness of the design by optimizing a ridged efficiency criterion – a small constant is added to each diagonal entry of the information matrix to make it nonsingular. That is what is shown in the iteration history. Unlike the `%MktEx` macro, the `%ChoicEff` macro does not have an explicit `ridge=` option. It automatically ridges, but only when needed. The option `converge=1e-12` was specified because for this example, iteration stops prematurely with the default convergence criterion. This next step switches to a full-rank coding, dropping the redundant variable `Brand3`, and using the output from the last step as the initial design.

```
%choiceff(data=full, init=best(keep=index), drop=brand3, seed=522,
          model=class(brand brand*x1 brand*x2 brand*x3 / zero=' '),
          nsets=15, flags=f1-f3, beta=zero, converge=1e-12);
```

The option `drop=brand3` is used to drop the parameter with the zero coefficient. We could have moved the brand specification into its own `class` specification (separate from the alternative-specific effects) and not specified `zero=' '` with it (see for example page 609). However, sometimes it is easier to specify a model with more terms than you really need, and then list the terms to drop, so that is what we illustrate here.

In this usage of `init=` with alternative swapping, the only part of the initial design that is required is the `Index` variable. It contains indices into the candidate set of the alternatives that are used to make the initial design. This usage is for the situation where the initial design was output from the macro. (In contrast, in the example usage on page 603, the option `initvars=x1-x3` was specified because the

initial design was not created by the `%ChoicEff` macro.) Here is some of the output. Notice that now there are no zero parameters so *D*-efficiency can be directly computed.

Design	Iteration	D-Efficiency	D-Error
1	0	0.683825	1.462362
	1	0.683825	1.462362

Final Results

Design	1
Choice Sets	15
Alternatives	3
D-Efficiency	0.683825
D-Error	1.462362

n	Variable Name	Label	Variance	DF	Standard Error
1	Brand1	Brand 1	5.70845	1	2.38924
2	Brand2	Brand 2	3.89246	1	1.97293
3	Brand1x11	Brand 1 * x1 1	1.99395	1	1.41207
4	Brand1x12	Brand 1 * x1 2	2.09289	1	1.44668
5	Brand2x11	Brand 2 * x1 1	1.80635	1	1.34400
6	Brand2x12	Brand 2 * x1 2	2.09299	1	1.44672
7	Brand3x11	Brand 3 * x1 1	2.12281	1	1.45699
8	Brand3x12	Brand 3 * x1 2	2.16706	1	1.47209
9	Brand1x21	Brand 1 * x2 1	2.38373	1	1.54393
10	Brand1x22	Brand 1 * x2 2	2.17427	1	1.47454
11	Brand2x21	Brand 2 * x2 1	2.28422	1	1.51136
12	Brand2x22	Brand 2 * x2 2	2.25113	1	1.50038
13	Brand3x21	Brand 3 * x2 1	2.21430	1	1.48805
14	Brand3x22	Brand 3 * x2 2	2.09383	1	1.44701
15	Brand1x31	Brand 1 * x3 1	2.39416	1	1.54731
16	Brand1x32	Brand 1 * x3 2	2.10564	1	1.45108
17	Brand2x31	Brand 2 * x3 1	2.55697	1	1.59905
18	Brand2x32	Brand 2 * x3 2	2.25251	1	1.50084
19	Brand3x31	Brand 3 * x3 1	2.29769	1	1.51581
20	Brand3x32	Brand 3 * x3 2	2.18010	1	1.47652
				==	
				20	

These next steps handle the same problem, only this time, we use the set-swapping algorithm, and we will specify a parameter vector that is not zero. At first, we will omit the `beta=` option, just to see the coding. We specified the `effects` option in the PROC TRANSREG `class` specification to get -1, 0, 1 coding.

```
%mktex(3 ** 9, n=2187, seed=121)

data key;
   input (Brand x1-x3) ($);
   datalines;
1 x1 x2 x3
2 x4 x5 x6
3 x7 x8 x9
;
%mktroll(design=design, key=key, alt=brand, out=rolled)

%choiceff(data=rolled, nsets=15, nalts=3,
          model=class(brand)
                class(brand*x1 brand*x2 brand*x3 / effects zero=' '))
```

The output tells us the parameter names and the order in which we need to specify parameters.

n	Name	Beta	Label
1	Brand1	.	Brand 1
2	Brand2	.	Brand 2
3	Brand1x11	.	Brand 1 * x1 1
4	Brand1x12	.	Brand 1 * x1 2
5	Brand2x11	.	Brand 2 * x1 1
6	Brand2x12	.	Brand 2 * x1 2
7	Brand3x11	.	Brand 3 * x1 1
8	Brand3x12	.	Brand 3 * x1 2
9	Brand1x21	.	Brand 1 * x2 1
10	Brand1x22	.	Brand 1 * x2 2
11	Brand2x21	.	Brand 2 * x2 1
12	Brand2x22	.	Brand 2 * x2 2
13	Brand3x21	.	Brand 3 * x2 1
14	Brand3x22	.	Brand 3 * x2 2
15	Brand1x31	.	Brand 1 * x3 1
16	Brand1x32	.	Brand 1 * x3 2
17	Brand2x31	.	Brand 2 * x3 1
18	Brand2x32	.	Brand 2 * x3 2
19	Brand3x31	.	Brand 3 * x3 1
20	Brand3x32	.	Brand 3 * x3 2

Now that we are sure we know the order of the parameters, we can specify the assumed betas on the `beta=` option. These numbers are based on prior research or our expectations of approximately what we expect the parameter estimates will be. We also specified `n=100` on this run, which is a sample size we are considering.

```
%choiceff(data=rolled, nsets=15, nalts=3, n=100, seed=462,
          beta=1 2 -0.5 0.5 -0.75 0.75 -1 1
               -0.5 0.5 -0.75 0.75 -1 1 -0.5 0.5 -0.75 0.75 -1 1,
          model=class(brand)
                class(brand*x1 brand*x2 brand*x3 / effects zero=' '))
```

Here is some of the output. Notice that parameters and test statistics are incorporated into the output. The `n=` value is incorporated into the variance matrix and hence the efficiency statistics, variances and tests.

n	Variable Name	Label	Variance	Assumed Beta	DF	Standard Error	Wald	Prob > Squared Wald
1	Brand1	Brand 1	0.011889	1.00	1	0.10903	9.1714	0.0001
2	Brand2	Brand 2	0.020697	2.00	1	0.14386	13.9021	0.0001
3	Brand1x11	Brand 1 * x1 1	0.008617	-0.50	1	0.09283	-5.3865	0.0001
4	Brand1x12	Brand 1 * x1 2	0.008527	0.50	1	0.09234	5.4147	0.0001
5	Brand2x11	Brand 2 * x1 1	0.009283	-0.75	1	0.09635	-7.7842	0.0001
6	Brand2x12	Brand 2 * x1 2	0.012453	0.75	1	0.11159	6.7208	0.0001
7	Brand3x11	Brand 3 * x1 1	0.021764	-1.00	1	0.14753	-6.7784	0.0001
8	Brand3x12	Brand 3 * x1 2	0.015657	1.00	1	0.12513	7.9917	0.0001
9	Brand1x21	Brand 1 * x2 1	0.012520	-0.50	1	0.11189	-4.4685	0.0001
10	Brand1x22	Brand 1 * x2 2	0.010685	0.50	1	0.10337	4.8370	0.0001
11	Brand2x21	Brand 2 * x2 1	0.010545	-0.75	1	0.10269	-7.3035	0.0001
12	Brand2x22	Brand 2 * x2 2	0.012654	0.75	1	0.11249	6.6672	0.0001
13	Brand3x21	Brand 3 * x2 1	0.018279	-1.00	1	0.13520	-7.3964	0.0001
14	Brand3x22	Brand 3 * x2 2	0.012117	1.00	1	0.11008	9.0845	0.0001
15	Brand1x31	Brand 1 * x3 1	0.009697	-0.50	1	0.09848	-5.0774	0.0001
16	Brand1x32	Brand 1 * x3 2	0.010787	0.50	1	0.10386	4.8141	0.0001
17	Brand2x31	Brand 2 * x3 1	0.009203	-0.75	1	0.09593	-7.8181	0.0001
18	Brand2x32	Brand 2 * x3 2	0.013923	0.75	1	0.11800	6.3562	0.0001
19	Brand3x31	Brand 3 * x3 1	0.016546	-1.00	1	0.12863	-7.7742	0.0001
20	Brand3x32	Brand 3 * x3 2	0.014235	1.00	1	0.11931	8.3815	0.0001
					==			
					20			

These next steps create a design for a cross-effects model with five brands at three prices and a constant alternative. (See the examples beginning on pages 261 and 283 for more information on cross effects.) Note the choice-set-swapping algorithm can handle cross effects but not the alternative-swapping algorithm.

```
%mktex(3 ** 5, n=3**5)
```

```
data key;
   input (Brand Price) ($);
   datalines;
1 x1
2 x2
3 x3
4 x4
5 x5
. .
;
%mktroll(design=design, key=key, alt=brand, out=rolled, keep=x1-x5)

proc print; by set; id set; where set in (1, 48, 101, 243); run;
```

The `keep=` option on the `%MktRoll` macro is used to keep the price variables that are needed to make the cross effects. Here are a few of the candidate choice sets.

Set	Brand	Price	x1	x2	x3	x4	x5
1	1	1	1	1	1	1	1
	2	1	1	1	1	1	1
	3	1	1	1	1	1	1
	4	1	1	1	1	1	1
	5	1	1	1	1	1	1
		.	1	1	1	1	1
48	1	1	1	2	3	1	3
	2	2	1	2	3	1	3
	3	3	1	2	3	1	3
	4	1	1	2	3	1	3
	5	3	1	2	3	1	3
		.	1	2	3	1	3
101	1	2	2	1	3	1	2
	2	1	2	1	3	1	2
	3	3	2	1	3	1	2
	4	1	2	1	3	1	2
	5	2	2	1	3	1	2
		.	2	1	3	1	2
243	1	3	3	3	3	3	3
	2	3	3	3	3	3	3
	3	3	3	3	3	3	3
	4	3	3	3	3	3	3
	5	3	3	3	3	3	3
		.	3	3	3	3	3

Notice that `x1` contains the price for Brand 1, `x2` contains the price for Brand 2, and so on, and the price of brand i in a choice set is the same, no matter which alternative it is stored with.

Here is the `%ChoicEff` macro call for creating the choice design with cross effects.

```
%choiceff(data=rolled, seed=17,
          model=class(brand brand*price / zero=none)
                identity(x1-x5) * class(brand / zero=none),
          nsets=20, nalts=6, beta=zero);
```

Cross effects are created by interacting the price factors with brand. See pages 268 and 321 for more information about cross effects.

Here is the redundant variable list from the log.

```
Redundant Variables:

Brand1Price3 Brand2Price3 Brand3Price3 Brand4Price3 Brand5Price3 x1Brand1
x2Brand2 x3Brand3 x4Brand4 x5Brand5
```

Next, we will run the macro again, this time requesting a full-rank model. The list of dropped names was created by copying from the redundant variable list. Also, `zero=none` was changed to `zero=' '` so no level would be zeroed for `Brand` but the last level of `Price` would be zeroed.

```
%choiceff(data=rolled, seed=17,
          model=class(brand brand*price / zero=' ')
                identity(x1-x5) * class(brand / zero=none),
          drop=x1Brand1 x2Brand2 x3Brand3 x4Brand4 x5Brand5,
          nsets=20, nalts=6, beta=zero);
```

Here is the last part of the output. Notice that we have five brand parameters, two price parameters for each of the five brands, and four cross effect parameters for each of the five brands.

n	Variable Name	Label	Variance	DF	Standard Error
1	Brand1	Brand 1	13.8149	1	3.71683
2	Brand2	Brand 2	13.5263	1	3.67782
3	Brand3	Brand 3	13.2895	1	3.64547
4	Brand4	Brand 4	13.5224	1	3.67728
5	Brand5	Brand 5	16.3216	1	4.04000
6	Brand1Price1	Brand 1 * Price 1	2.8825	1	1.69779
7	Brand1Price2	Brand 1 * Price 2	3.5118	1	1.87399
8	Brand2Price1	Brand 2 * Price 1	2.8710	1	1.69441
9	Brand2Price2	Brand 2 * Price 2	3.5999	1	1.89733
10	Brand3Price1	Brand 3 * Price 1	2.8713	1	1.69448
11	Brand3Price2	Brand 3 * Price 2	3.5972	1	1.89662
12	Brand4Price1	Brand 4 * Price 1	2.8710	1	1.69441
13	Brand4Price2	Brand 4 * Price 2	3.5560	1	1.88574
14	Brand5Price1	Brand 5 * Price 1	2.8443	1	1.68649
15	Brand5Price2	Brand 5 * Price 2	3.8397	1	1.95953
16	x1Brand2	x1 * Brand 2	0.7204	1	0.84878
17	x1Brand3	x1 * Brand 3	0.7209	1	0.84908
18	x1Brand4	x1 * Brand 4	0.7204	1	0.84878
19	x1Brand5	x1 * Brand 5	0.7204	1	0.84877

```
20    x2Brand1        x2 * Brand 1          0.7178     1     0.84722
21    x2Brand3        x2 * Brand 3          0.7178     1     0.84724
22    x2Brand4        x2 * Brand 4          0.7178     1     0.84720
23    x2Brand5        x2 * Brand 5          0.7248     1     0.85133
24    x3Brand1        x3 * Brand 1          0.7178     1     0.84722
25    x3Brand2        x3 * Brand 2          0.7178     1     0.84721
26    x3Brand4        x3 * Brand 4          0.7178     1     0.84720
27    x3Brand5        x3 * Brand 5          0.7248     1     0.85133
28    x4Brand1        x4 * Brand 1          0.7178     1     0.84722
29    x4Brand2        x4 * Brand 2          0.7178     1     0.84721
30    x4Brand3        x4 * Brand 3          0.7178     1     0.84724
31    x4Brand5        x4 * Brand 5          0.7293     1     0.85402
32    x5Brand1        x5 * Brand 1          0.7111     1     0.84325
33    x5Brand2        x5 * Brand 2          0.7180     1     0.84737
34    x5Brand3        x5 * Brand 3          0.7248     1     0.85135
35    x5Brand4        x5 * Brand 4          0.7179     1     0.84731
                                                      ==
                                                      35
```

In this final `%ChoicEff` macro example, the goal is to create a design for a pricing study with ten brands plus a constant alternative. Each brand has a single attribute, price. However, the prices are potentially different for each brand and they do not even have the same numbers of levels. A model is desired with brand and alternative-specific price effects. Here are the design specifications.

Brand	Levels	Prices
Brand 1	8	0.89 0.94 0.99 1.04 1.09 1.14 1.19 1.24
Brand 2	8	0.94 0.99 1.04 1.09 1.14 1.19 1.24 1.29
Brand 3	6	0.99 1.04 1.09 1.14 1.19 1.24
Brand 4	6	0.89 0.94 0.99 1.04 1.09 1.14
Brand 5	6	1.04 1.09 1.14 1.19 1.24 1.29
Brand 6	4	0.89 0.99 1.09 1.19
Brand 7	4	0.99 1.09 1.19 1.29
Brand 8	4	0.94 0.99 1.14 1.19
Brand 9	4	1.09 1.14 1.19 1.24
Brand 10	4	1.14 1.19 1.24 1.29

There are two challenging aspects of this problem: creating the candidate set and coping with the price asymmetries. The candidate set must contain 8 rows for the eight Brand 1 prices, 8 rows for the eight Brand 2 prices, 6 rows for the six Brand 3 prices, ..., and 4 rows for the four Brand 10 prices. It also must contain a constant alternative. Furthermore, if we are to use the alternative-swapping algorithm, the candidate set must contain 11 flag variables, each of which will designate the appropriate group of alternatives. We could run the `%MktEx` macro ten times to make a candidate set for each of the brands, but since we have only one factor per brand, it would be much easier to generate the candidate set with a DATA step. Before we discuss the code, here is the candidate set that we need.

Brand	Price	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11
1	1	1	0	0	0	0	0	0	0	0	0	0
1	2	1	0	0	0	0	0	0	0	0	0	0
1	3	1	0	0	0	0	0	0	0	0	0	0
1	4	1	0	0	0	0	0	0	0	0	0	0
1	5	1	0	0	0	0	0	0	0	0	0	0
1	6	1	0	0	0	0	0	0	0	0	0	0
1	7	1	0	0	0	0	0	0	0	0	0	0
1	8	1	0	0	0	0	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0	0	0
2	2	0	1	0	0	0	0	0	0	0	0	0
2	3	0	1	0	0	0	0	0	0	0	0	0
2	4	0	1	0	0	0	0	0	0	0	0	0
2	5	0	1	0	0	0	0	0	0	0	0	0
2	6	0	1	0	0	0	0	0	0	0	0	0
2	7	0	1	0	0	0	0	0	0	0	0	0
2	8	0	1	0	0	0	0	0	0	0	0	0
3	1	0	0	1	0	0	0	0	0	0	0	0
3	2	0	0	1	0	0	0	0	0	0	0	0
3	3	0	0	1	0	0	0	0	0	0	0	0
3	4	0	0	1	0	0	0	0	0	0	0	0
3	5	0	0	1	0	0	0	0	0	0	0	0
3	6	0	0	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	0	0	0	0	0
4	2	0	0	0	1	0	0	0	0	0	0	0
4	3	0	0	0	1	0	0	0	0	0	0	0
4	4	0	0	0	1	0	0	0	0	0	0	0
4	5	0	0	0	1	0	0	0	0	0	0	0
4	6	0	0	0	1	0	0	0	0	0	0	0
5	1	0	0	0	0	1	0	0	0	0	0	0
5	2	0	0	0	0	1	0	0	0	0	0	0
5	3	0	0	0	0	1	0	0	0	0	0	0
5	4	0	0	0	0	1	0	0	0	0	0	0
5	5	0	0	0	0	1	0	0	0	0	0	0
5	6	0	0	0	0	1	0	0	0	0	0	0
6	1	0	0	0	0	0	1	0	0	0	0	0
6	2	0	0	0	0	0	1	0	0	0	0	0
6	3	0	0	0	0	0	1	0	0	0	0	0
6	4	0	0	0	0	0	1	0	0	0	0	0
7	1	0	0	0	0	0	0	1	0	0	0	0
7	2	0	0	0	0	0	0	1	0	0	0	0
7	3	0	0	0	0	0	0	1	0	0	0	0
7	4	0	0	0	0	0	0	1	0	0	0	0
8	1	0	0	0	0	0	0	0	1	0	0	0
8	2	0	0	0	0	0	0	0	1	0	0	0
8	3	0	0	0	0	0	0	0	1	0	0	0
8	4	0	0	0	0	0	0	0	1	0	0	0

9	1	0	0	0	0	0	0	0	0	1	0	0
9	2	0	0	0	0	0	0	0	0	1	0	0
9	3	0	0	0	0	0	0	0	0	1	0	0
9	4	0	0	0	0	0	0	0	0	1	0	0
10	1	0	0	0	0	0	0	0	0	0	1	0
10	2	0	0	0	0	0	0	0	0	0	1	0
10	3	0	0	0	0	0	0	0	0	0	1	0
10	4	0	0	0	0	0	0	0	0	0	1	0
.	.	0	0	0	0	0	0	0	0	0	0	1

It begins with eight alternatives for the eight prices for the first brand (`Brand` = 1 `f1` = 1, `f2-f11` = 0). It is followed by eight alternatives for the eight prices for the second brand (`Brand` = 2 `f2` = 1, `f1` = 0, `f3` through `f11` = 0). At the end is the constant alternative. For now, we do not need to worry about the actual price levels, since price will be treated as a qualitative factor. Here is the code that generated the candidate design.

```
data cand;
   array n[10] _temporary_ (8 8 6 6 6 4 4 4 4 4);
   retain f1-f11 0;
   array f[11];
   do Brand = 1 to 10;
      f[brand] = 1;
      do Price = 1 to n[brand]; output; end;
      f[brand] = 0;
      end;
   brand = .; price = .; f11 = 1; output;
   run;

proc print; id brand price; run;
```

It has the statement `do Brand = 1 to 10` that creates the ten brands, plus an `output` statement at the end to generate the constant alternative. Inside the `do Brand` loop is another `do` loop that creates the `n[brand]` prices. The `n` array is a temporary array, which means it will not create any variables to go into the output data set. It just gives us a convenient way to access the number of levels for each of the ten brands.

This call to the `%ChoicEff` macro creates the design naming `Brand` and `Price` as classification variables. Dummy variables will be created for all nonmissing levels of brand.

```
%choiceff(data=cand, seed=462,
          model=class(brand / zero=none) class(brand*price / zero=' '),
          nsets=24, flags=f1-f11, beta=zero)
```

Here is some of the output.

Design	Iteration	D-Efficiency	D-Error
1	0	0	.
	1	0	.
		0.001445 (Ridged)	
	2	0	.
		0.001445 (Ridged)	

Design	Iteration	D-Efficiency	D-Error
2	0	0	.
	1	0	.
		0.001445 (Ridged)	
	2	0	.
		0.001445 (Ridged)	

Final Results

Design	1
Choice Sets	24
Alternatives	11
D-Efficiency	0
D-Error	.

n	Variable Name	Label	Variance	DF	Standard Error
1	Brand1	Brand 1	4.51944	1	2.12590
2	Brand2	Brand 2	4.50242	1	2.12189
3	Brand3	Brand 3	3.50008	1	1.87085
4	Brand4	Brand 4	3.49251	1	1.86882
5	Brand5	Brand 5	3.48040	1	1.86558
6	Brand6	Brand 6	2.46342	1	1.56953
7	Brand7	Brand 7	2.46617	1	1.57041
8	Brand8	Brand 8	2.47129	1	1.57203
9	Brand9	Brand 9	2.47975	1	1.57472
10	Brand10	Brand 10	2.46659	1	1.57054
11	Brand1Price1	Brand 1 * Price 1	8.18241	1	2.86049
12	Brand1Price2	Brand 1 * Price 2	8.22704	1	2.86828
13	Brand1Price3	Brand 1 * Price 3	8.20656	1	2.86471
14	Brand1Price4	Brand 1 * Price 4	8.28067	1	2.87762
15	Brand1Price5	Brand 1 * Price 5	8.20668	1	2.86473
16	Brand1Price6	Brand 1 * Price 6	8.21663	1	2.86647
17	Brand1Price7	Brand 1 * Price 7	8.25795	1	2.87366

18	Brand2Price1	Brand	2 * Price 1	8.16766	1	2.85791
19	Brand2Price2	Brand	2 * Price 2	8.25283	1	2.87277
20	Brand2Price3	Brand	2 * Price 3	8.18178	1	2.86038
21	Brand2Price4	Brand	2 * Price 4	8.22588	1	2.86808
22	Brand2Price5	Brand	2 * Price 5	8.24887	1	2.87208
23	Brand2Price6	Brand	2 * Price 6	8.19937	1	2.86345
24	Brand2Price7	Brand	2 * Price 7	8.21050	1	2.86540
25	Brand3Price1	Brand	3 * Price 1	6.18856	1	2.48768
26	Brand3Price2	Brand	3 * Price 2	6.16883	1	2.48371
27	Brand3Price3	Brand	3 * Price 3	6.22013	1	2.49402
28	Brand3Price4	Brand	3 * Price 4	6.17914	1	2.48579
29	Brand3Price5	Brand	3 * Price 5	6.17185	1	2.48432
30	Brand3Price6	Brand	3 * Price 6	.	0	.
31	Brand3Price7	Brand	3 * Price 7	.	0	.
32	Brand4Price1	Brand	4 * Price 1	6.16116	1	2.48217
33	Brand4Price2	Brand	4 * Price 2	6.18716	1	2.48740
34	Brand4Price3	Brand	4 * Price 3	6.16633	1	2.48321
35	Brand4Price4	Brand	4 * Price 4	6.25094	1	2.50019
36	Brand4Price5	Brand	4 * Price 5	6.13517	1	2.47693
37	Brand4Price6	Brand	4 * Price 6	.	0	.
38	Brand4Price7	Brand	4 * Price 7	.	0	.
39	Brand5Price1	Brand	5 * Price 1	6.15820	1	2.48157
40	Brand5Price2	Brand	5 * Price 2	6.17572	1	2.48510
41	Brand5Price3	Brand	5 * Price 3	6.14151	1	2.47821
42	Brand5Price4	Brand	5 * Price 4	6.17153	1	2.48426
43	Brand5Price5	Brand	5 * Price 5	6.14552	1	2.47902
44	Brand5Price6	Brand	5 * Price 6	.	0	.
45	Brand5Price7	Brand	5 * Price 7	.	0	.
46	Brand6Price1	Brand	6 * Price 1	4.14170	1	2.03512
47	Brand6Price2	Brand	6 * Price 2	4.15481	1	2.03834
48	Brand6Price3	Brand	6 * Price 3	4.09000	1	2.02238
49	Brand6Price4	Brand	6 * Price 4	.	0	.
50	Brand6Price5	Brand	6 * Price 5	.	0	.
51	Brand6Price6	Brand	6 * Price 6	.	0	.
52	Brand6Price7	Brand	6 * Price 7	.	0	.
53	Brand7Price1	Brand	7 * Price 1	4.12837	1	2.03184
54	Brand7Price2	Brand	7 * Price 2	4.09248	1	2.02299
55	Brand7Price3	Brand	7 * Price 3	4.16503	1	2.04084
56	Brand7Price4	Brand	7 * Price 4	.	0	.
57	Brand7Price5	Brand	7 * Price 5	.	0	.
58	Brand7Price6	Brand	7 * Price 6	.	0	.
59	Brand7Price7	Brand	7 * Price 7	.	0	.
60	Brand8Price1	Brand	8 * Price 1	4.16889	1	2.04179
61	Brand8Price2	Brand	8 * Price 2	4.16045	1	2.03972
62	Brand8Price3	Brand	8 * Price 3	4.09355	1	2.02325
63	Brand8Price4	Brand	8 * Price 4	.	0	.
64	Brand8Price5	Brand	8 * Price 5	.	0	.
65	Brand8Price6	Brand	8 * Price 6	.	0	.
66	Brand8Price7	Brand	8 * Price 7	.	0	.

```
67    Brand9Price1     Brand  9 * Price 1     4.11932     1     2.02961
68    Brand9Price2     Brand  9 * Price 2     4.15745     1     2.03898
69    Brand9Price3     Brand  9 * Price 3     4.17574     1     2.04346
70    Brand9Price4     Brand  9 * Price 4      .          0      .
71    Brand9Price5     Brand  9 * Price 5      .          0      .
72    Brand9Price6     Brand  9 * Price 6      .          0      .
73    Brand9Price7     Brand  9 * Price 7      .          0      .
74    Brand10Price1    Brand 10 * Price 1     4.12770     1     2.03167
75    Brand10Price2    Brand 10 * Price 2     4.12731     1     2.03158
76    Brand10Price3    Brand 10 * Price 3     4.11729     1     2.02911
77    Brand10Price4    Brand 10 * Price 4      .          0      .
78    Brand10Price5    Brand 10 * Price 5      .          0      .
79    Brand10Price6    Brand 10 * Price 6      .          0      .
80    Brand10Price7    Brand 10 * Price 7      .          0      .
                                                          ==
                                                          54
```

There are unneeded parameters in our model, and for the moment, that is fine. We see 10 parameters for `Brand`. The constant alternative (not shown) is the reference alternative. We see 7 parameters for Brand 1's price (8 prices - 1 = 7), 7 parameters for Brand 2's price, 5 parameters for Brand 3's price (6 prices - 1 = 5), ..., and 3 parameters for Brand 10's price (4 prices - 1 = 3). This all looks correct.

The log file contains the lines:

```
Redundant Variables:

Brand3Price6 Brand3Price7 Brand4Price6 Brand4Price7 Brand5Price6 Brand5Price7
Brand6Price4 Brand6Price5 Brand6Price6 Brand6Price7 Brand7Price4 Brand7Price5
Brand7Price6 Brand7Price7 Brand8Price4 Brand8Price5 Brand8Price6 Brand8Price7
Brand9Price4 Brand9Price5 Brand9Price6 Brand9Price7 Brand10Price4 Brand10Price5
Brand10Price6 Brand10Price7
```

Here they are again, manually reformatted to one brand per line:

```
Redundant Variables:

Brand3Price6 Brand3Price7
Brand4Price6 Brand4Price7
Brand5Price6 Brand5Price7
Brand6Price4 Brand6Price5 Brand6Price6 Brand6Price7
Brand7Price4 Brand7Price5 Brand7Price6 Brand7Price7
Brand8Price4 Brand8Price5 Brand8Price6 Brand8Price7
Brand9Price4 Brand9Price5 Brand9Price6 Brand9Price7
Brand10Price4 Brand10Price5 Brand10Price6 Brand10Price7
```

For Brands 1 and 2 we have 7 parameters and the last level for price 8 is the reference level and does not appear in the model. The specification `class(brand*price / zero=' ')` sets the reference level for brand to blank so it will use all 10 brands and uses the ordinary default last level for the reference level for price. This `zero=` specification names a list of reference levels, blank for the first variable and nothing specified, and hence the default, for the second.

For Brands 3 through 5, level 8, which does not appear, is the reference level as it is for all the brands. In addition, since these brands have only six levels, two more terms are not estimable, the terms for price levels 6 and 7. Hence the factors `Brand3Price6, Brand3Price7, Brand4Price6, Brand4Price7, Brand5Price6,` and `Brand5Price7` are not needed. Similarly for Brands 6 through 10, we can drop the terms for the fourth through seventh price levels. We can run the macro again, this time deleting all these terms.

```
%choiceff(data=cand, seed=462,
          model=class(brand / zero=none) class(brand*price / zero=' '),
          nsets=24, flags=f1-f11, beta=zero,
          drop=
          brand3price6 brand3price7
          brand4price6 brand4price7
          brand5price6 brand5price7
          brand6price4 brand6price5 brand6price6 brand6price7
          brand7price4 brand7price5 brand7price6 brand7price7
          brand8price4 brand8price5 brand8price6 brand8price7
          brand9price4 brand9price5 brand9price6 brand9price7
          brand10price4 brand10price5 brand10price6 brand10price7
          );
```

Here is some of the output.

Design	Iteration	D-Efficiency	D-Error
1	0	0.313841	3.186325
	1	0.340743	2.934765
	2	0.340743	2.934765

Design	Iteration	D-Efficiency	D-Error
2	0	0	.
	1	0.341011	2.932458
	2	0.341011	2.932458

Final Results

Design	2
Choice Sets	24
Alternatives	11
D-Efficiency	0.341011
D-Error	2.932458

n	Variable Name	Label	Variance	DF	Standard Error
1	Brand1	Brand 1	4.50417	1	2.12230
2	Brand2	Brand 2	4.52567	1	2.12736
3	Brand3	Brand 3	3.47776	1	1.86487
4	Brand4	Brand 4	3.48724	1	1.86742
5	Brand5	Brand 5	3.49982	1	1.87078
6	Brand6	Brand 6	2.47337	1	1.57270
7	Brand7	Brand 7	2.45738	1	1.56760
8	Brand8	Brand 8	2.45142	1	1.56570
9	Brand9	Brand 9	2.45282	1	1.56615
10	Brand10	Brand 10	2.44575	1	1.56389
11	Brand1Price1	Brand 1 * Price 1	8.19264	1	2.86228
12	Brand1Price2	Brand 1 * Price 2	8.19269	1	2.86229
13	Brand1Price3	Brand 1 * Price 3	8.18940	1	2.86171
14	Brand1Price4	Brand 1 * Price 4	8.23067	1	2.86891
15	Brand1Price5	Brand 1 * Price 5	8.21587	1	2.86633
16	Brand1Price6	Brand 1 * Price 6	8.19365	1	2.86246
17	Brand1Price7	Brand 1 * Price 7	8.23031	1	2.86885
18	Brand2Price1	Brand 2 * Price 1	8.16830	1	2.85802
19	Brand2Price2	Brand 2 * Price 2	8.23185	1	2.86912
20	Brand2Price3	Brand 2 * Price 3	8.21687	1	2.86651
21	Brand2Price4	Brand 2 * Price 4	8.23295	1	2.86931
22	Brand2Price5	Brand 2 * Price 5	8.27059	1	2.87586
23	Brand2Price6	Brand 2 * Price 6	8.22612	1	2.86812
24	Brand2Price7	Brand 2 * Price 7	8.28203	1	2.87785
25	Brand3Price1	Brand 3 * Price 1	6.15558	1	2.48104
26	Brand3Price2	Brand 3 * Price 2	6.17640	1	2.48524
27	Brand3Price3	Brand 3 * Price 3	6.13255	1	2.47640
28	Brand3Price4	Brand 3 * Price 4	6.14840	1	2.47960
29	Brand3Price5	Brand 3 * Price 5	6.11249	1	2.47234
30	Brand4Price1	Brand 4 * Price 1	6.17231	1	2.48441
31	Brand4Price2	Brand 4 * Price 2	6.22760	1	2.49552
32	Brand4Price3	Brand 4 * Price 3	6.12111	1	2.47409
33	Brand4Price4	Brand 4 * Price 4	6.19792	1	2.48956
34	Brand4Price5	Brand 4 * Price 5	6.12131	1	2.47413
35	Brand5Price1	Brand 5 * Price 1	6.21514	1	2.49302
36	Brand5Price2	Brand 5 * Price 2	6.15748	1	2.48143
37	Brand5Price3	Brand 5 * Price 3	6.17697	1	2.48535
38	Brand5Price4	Brand 5 * Price 4	6.16121	1	2.48218
39	Brand5Price5	Brand 5 * Price 5	6.20067	1	2.49011
40	Brand6Price1	Brand 6 * Price 1	4.16170	1	2.04002
41	Brand6Price2	Brand 6 * Price 2	4.11324	1	2.02811
42	Brand6Price3	Brand 6 * Price 3	4.13298	1	2.03297
43	Brand7Price1	Brand 7 * Price 1	4.10703	1	2.02658
44	Brand7Price2	Brand 7 * Price 2	4.11083	1	2.02752
45	Brand7Price3	Brand 7 * Price 3	4.10632	1	2.02641

46	Brand8Price1	Brand	8 * Price 1	4.12107	1	2.03004
47	Brand8Price2	Brand	8 * Price 2	4.10075	1	2.02503
48	Brand8Price3	Brand	8 * Price 3	4.08366	1	2.02081
49	Brand9Price1	Brand	9 * Price 1	4.11157	1	2.02770
50	Brand9Price2	Brand	9 * Price 2	4.10049	1	2.02497
51	Brand9Price3	Brand	9 * Price 3	4.10522	1	2.02614
52	Brand10Price1	Brand	10 * Price 1	4.07896	1	2.01964
53	Brand10Price2	Brand	10 * Price 2	4.09065	1	2.02253
54	Brand10Price3	Brand	10 * Price 3	4.11148	1	2.02768
					==	
					54	

We can see that we now have all the terms for the final model.

```
proc print data=best(obs=22); id set; by set; var brand price; run;
```

Here are the first two choice sets.

Set	Brand	Price
1	1	2
	2	8
	3	6
	4	6
	5	4
	6	4
	7	2
	8	4
	9	4
	10	3
	.	.
2	1	4
	2	8
	3	4
	4	3
	5	2
	6	4
	7	4
	8	3
	9	4
	10	3
	.	.

Because of the asymmetry, assigning the actual prices is not as simple as using a format. You could write a lot of code of the form `if brand = 1 and price = 1 then price = 0.89; else ...`, however that would be difficult. Instead, we will start by transposing our choice design.

```
proc transpose data=best out=lin(keep=x1-x10) prefix=x;
   var price; by set;
   run;

proc print; run;
```

The transposed data set has one row per choice set and each of the 10 prices for the ten nonconstant alternatives in each choice set. This is the linear version of our choice design. The factors `x1` and `x2` have 8 prices, 1 to 8, the factors `x3` through `x5` have 6 prices, 1 to 6. and the factors `x6` through `x10` have 4 prices, 1 to 4.

Obs	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
1	2	8	6	6	4	4	2	4	4	3
2	4	8	4	3	2	4	4	3	4	3
3	8	4	3	5	5	4	1	4	4	4
4	2	1	4	3	4	1	1	3	3	2
5	6	8	6	6	5	2	3	3	3	2
6	6	1	5	5	2	4	3	4	4	3
7	8	6	5	6	4	3	3	3	2	4
8	4	5	2	3	3	3	3	1	3	4
9	5	3	5	1	6	2	4	1	3	4
10	3	4	2	2	4	2	4	2	1	1
11	3	5	5	1	3	3	2	4	2	2
12	2	3	1	5	1	4	2	2	1	3
13	6	2	3	3	6	2	3	1	2	3
14	5	1	1	4	2	3	1	2	3	2
15	8	6	4	4	5	2	2	2	1	2
16	5	2	6	5	6	3	4	3	2	1
17	1	7	1	4	5	1	1	3	1	1
18	3	3	2	6	6	4	1	1	3	3
19	4	5	1	2	1	1	3	4	2	1
20	1	6	3	1	1	3	4	2	4	1
21	7	7	3	2	1	1	2	2	1	4
22	7	2	2	4	3	1	1	1	4	1
23	7	4	4	2	3	1	2	1	2	2
24	1	7	6	1	2	2	4	4	1	4

Now we can use formats or other means such as the `%MktLab` macro to assign prices within the brand/price factors and then restore the choice design format. Here is the `key=` data set for the `%MktLab` macro. It contains all of the different prices. The `%MktLab` macro assigns the prices and the `%MktRoll` macro is used to convert our linear design back into a choice design.[†] The `%MktLab` macro uses a KEY data set to specify the prices (see page 712). The `%MktRoll` macro uses a different KEY data set to specify which linear design factor applies to which brand (see page 735).

[†]See page 60 for an illustration of linear versus choice designs.

```
data key;
   input x1-x10;
   datalines;
0.89 0.94 0.99 0.89 1.04 0.89 0.99 0.94 1.09 1.14
0.94 0.99 1.04 0.94 1.09 0.99 1.09 0.99 1.14 1.19
0.99 1.04 1.09 0.99 1.14 1.09 1.19 1.14 1.19 1.24
1.04 1.09 1.14 1.04 1.19 1.19 1.29 1.19 1.24 1.29
1.09 1.14 1.19 1.09 1.24 .    .    .    .    .
1.14 1.19 1.24 1.14 1.29 .    .    .    .    .
1.19 1.24 .    .    .    .    .    .    .    .
1.24 1.29 .    .    .    .    .    .    .    .
;
proc print; run;

%mktlab(data=lin, key=key, out=final)

%mktkey(x1-x10)

data key;
   if _n_ le 10 then Brand = put(_n_, 2.);
   input Price $ @@;
   datalines;
x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 .
;
proc print; run;

%mktroll(design=final,
         alt=brand, key=key, out=sasuser.prices)

proc print data=sasuser.prices(obs=22); id set; by set; var brand price; run;
```

Here is the %MktLab KEY data set.

Obs	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
1	0.89	0.94	0.99	0.89	1.04	0.89	0.99	0.94	1.09	1.14
2	0.94	0.99	1.04	0.94	1.09	0.99	1.09	0.99	1.14	1.19
3	0.99	1.04	1.09	0.99	1.14	1.09	1.19	1.14	1.19	1.24
4	1.04	1.09	1.14	1.04	1.19	1.19	1.29	1.19	1.24	1.29
5	1.09	1.14	1.19	1.09	1.24	.	.	.	.	.
6	1.14	1.19	1.24	1.14	1.29	.	.	.	.	.
7	1.19	1.24	.	.	.	.	.	.	.	.
8	1.24	1.29	.	.	.	.	.	.	.	.

Here is the %MktRoll KEY data set.

Obs	Brand	Price
1	1	x1
2	2	x2
3	3	x3
4	4	x4
5	5	x5
6	6	x6
7	7	x7
8	8	x8
9	9	x9
10	10	x10
11		

Here are the first two choice sets with the actual prices assigned.

Set	Brand	Price
1	1	0.94
	2	1.29
	3	1.24
	4	1.14
	5	1.19
	6	1.19
	7	1.09
	8	1.19
	9	1.24
	10	1.24
		.
2	1	1.04
	2	1.29
	3	1.14
	4	0.99
	5	1.09
	6	1.19
	7	1.29
	8	1.14
	9	1.24
	10	1.24
		.

%ChoicEff Macro Options

The following options can be used with the `%ChoicEff` macro.

Option	Description
`bestcov=`*SAS-data-set*	covariance matrix for the best design
`bestout=`*SAS-data-set*	best design
`beta=`*list*	true parameters
`converge=`*n*	convergence criterion
`cov=`*SAS-data-set*	all of the covariance matrices
`data=`*SAS-data-set*	input choice candidate set
`drop=`*variable-list*	variables to drop from the model
`fixed=`*variable-list*	variable that flags fixed alternatives
`flags=`*variable-list*	variables that flag the alternative(s)
`init=`*SAS-data-set*	input initial design data set
`initvars=`*variable-list*	initial variables
`intiter=`*n*	maximum number of internal iterations
`iter=`*n*	maximum iterations (designs to create)
`maxiter=`*n*	maximum iterations (designs to create)
`model=`*model-specification*	`model` statement list of effects
`morevars=`*variable-list*	more variables to add to the model
`n=`*n*	number of observations
`nalts=`*n*	number of alternatives
`nsets=`*n*	number of choice sets desired
`options=`*options-list*	binary options
`out=`*SAS-data-set*	all designs data set
`seed=`*n*	random number seed
`submat=`*number-list*	submatrix for efficiency calculations
`types=`*integer-list*	number of sets of each type
`typevar=`*variable*	choice set types variable
`weight=`*weight-variable*	optional weight variable

Required Options

You must specify both the `model=` and `nsets=` options and either the `flags=` or `nalts=` options. You can omit `beta=` if you just want a listing of effects, however you must specify `beta=` to create a design. The rest of the options are optional.

model= *model-specification*

specifies a PROC TRANSREG `model` statement list of effects. There are many potential forms for the model specification and a number of options. See the SAS/STAT PROC TRANSREG documentation.

Generic effects example:

```
model=class(x1-x3),
```

Brand and alternative-specific effects example:

```
model=class(b)
      class(b*x1 b*x2 b*x3 / effects zero=' '),
```

Brand, alternative-specific, and cross effects:

```
model=class(b b*p / zero=' ')
      identity(x1-x5) * class(b / zero=none),
```

See pages 600 through 625 for other examples of `model` syntax. Furthermore, all of the PROC TRANSREG and `%ChoicEff` macro examples from pages 173 through 409 show examples of model syntax for choice models.

nsets= *n*

specifies the number of choice sets desired.

Other Required Options

You must specify exactly one of these next two options. When the candidate set consists of individual alternatives to be swapped, specify the alternative flags with `flags=`. When the candidate set consists of entire sets of alternatives to be swapped, specify the number of alternatives in each set with `nalts=`.

flags= *variable-list*

specifies variables that flag the alternative(s) for which each candidate may be used. There must be one flag variable per alternative. If every candidate can be used in all alternatives, then the flags are constant. For example, with three alternatives, create these constant flags: `f1=1 f2=1 f3=1`. Otherwise, with three alternatives, specify `flags=f1-f3` and create a candidate set where: alternative 1 candidates are indicated by `f1=1 f2=0 f3=0`, alternative 2 candidates are indicated by `f1=0 f2=1 f3=0`, and alternative 3 candidates are indicated by `f1=0 f2=0 f3=1`.

nalts= *n*

specifies the number of alternatives in each choice set for the set-swapping algorithm.

Other Options

The rest of the parameters are optional. You may specify zero or more of them.

bestcov= *SAS-data-set*

specifies a name for the data set containing the covariance matrix for the best design. By default, this data set is called BESTCOV.

bestout= *SAS-data-set*

specifies a name for the data set containing the best design. By default, this data set is called BEST. Often, you will want to specify a two-level name to create a permanent SAS data set so the design will be available later for analysis.

beta= *list*
specifies the true parameters. By default, when `beta=` is not specified, the macro just reports on coding. You can specify `beta=zero` to assume all zeros. Otherwise specify a number list: `beta=1 -1 2 -2 1 -1`.

converge= *n*
specifies the *D*-efficiency convergence criterion. By default, `converge=0.005`.

cov= *SAS-data-set*
specifies a name for the data set containing all of the covariance matrices for all of the designs. By default, this data set is called COV.

data= *SAS-data-set*
specifies the input choice candidate set. By default, the macro uses the last data set created.

drop= *variable-list*
specifies a list of variables to drop from the model. If you specified a less-than-full-rank `model=` specification, you can use `drop=` to produce a full rank coding. When there are redundant variables, the macro prints a list that you can use in the `drop=` option on a subsequent run.

fixed= *variable-list*
specifies the variable that flags the fixed alternatives. When `fixed=variable` is specified, the `init=` data set must contain the named variable, which indicates which alternatives are fixed (cannot be swapped out) and which ones may be changed. Example: `fixed=fixed, init=init, initvars=x1-x3`. Values of the `fixed=` variable include:

1 - means this alternative can never be swapped out.

0 - means this alternative is used in the initial design, but it may be swapped out.

. - means this alternative should be randomly initialized, and it may be swapped out.

The `fixed=` option may be specified only when both `init=` and `initvars=` are specified.

init= *SAS-data-set*
specifies an input initial design data set. Null means a random start. One usage is to specify the `bestout=` data set for an initial start. When `flags=` is specified, `init=` must contain the index variable. Example: `init=best(keep=index)`. When `nalts=` is specified, `init=` must contain the choice set variable. Example: `init=best(keep=set)`.

Alternatively, the `init=` data set can contain an arbitrary design, potentially created outside this macro. In that case, you must also specify `initvars=factors`, where factors are the factors in the design, for example `initvars=x1-x3`. When alternatives are swapped, this data set must also contain the `flags=` variables. When `init=` is specified with `initvars=`, the data set may also contain a variable specified on the `fixed=` option, which indicates which alternatives are fixed, and which ones can be swapped in and out.

intiter= *n*
specifies the maximum number of internal iterations. Specify `intiter=0` to just evaluate efficiency of an existing design. By default, `intiter=10`.

initvars= *variable-list*
specifies the factor variables in the `init=` data set that must match up with the variables in the `data=` data set. See `init=`. All of these variables must be of the same type.

maxiter= *n*
iter= *n*
specifies the maximum iterations (designs to create). By default, `maxiter=10`.

morevars= *variable-list*
specifies more variables to add to the model. This option gives you the ability to specify a list of variables to copy along as is, through the TRANSREG coding, then add them to the model.

n= *n*
specifies the number of observations to use in the variance matrix formula. By default, `n=1`.

options= *options-list*
specifies binary options. By default, none of these options are specified. Specify one or more of the following values after `options=`.

`coded`
prints the coded candidate set.

`detail`
prints the details of the swaps.

`nocode`
skips the PROC TRANSREG coding stage, assuming that WORK.TMP_CAND was created by a previous step. This is most useful with set swapping when the candidate set can be big. It is important with `options=nocode` to note that the effect of `morevars=` and `drop=` in previous runs has already been taken care of, so do not specify them (unless for instance you want to drop still more variables).

`nodups`
prevents the same choice set from coming out more than once. This option does not affect the initialization, so the random initial design may have duplicates. This option forces duplicates out during the iterations, so do not set `intiter=` to a small value. It may take several iterations to eliminate all duplicates. It is possible that efficiency will decrease as duplicates are forced out. With set swapping, this macro checks the candidate choice set numbers to avoid duplicates. With alternative swapping, this macro checks the candidate alternative index to avoid duplicates. The macro does not look at the actual factors. This makes the checks faster, but if the candidate set contains duplicate choice sets or alternatives, the macro may not succeed in eliminating all duplicates. Run the `%MktDups` macro (which looks at the actual factors) on the design to check and make sure all duplicates are eliminated. If you are using set swapping to make a generic design make sure you run the `%MktDups` macro on the candidate set to eliminate duplicate choice sets in advance.

`notests`
suppresses printing the diagonal of the covariance matrix, and hypothesis tests for this n and $\boldsymbol{\beta}$. When $\boldsymbol{\beta}$ is not zero, the results include a Wald test statistic (β divided by the standard error), which is normally distributed, and the probability of a larger squared Wald statistic.

`orthcan`
orthogonalizes the candidate set.

out= *SAS-data-set*
specifies a name for the output SAS data set with all of the final designs. The default is `out=results`.

seed= *n*
specifies the random number seed. By default, `seed=0`, and clock time is used as the random number seed. By specifying a random number seed, results should be reproducible within a SAS release for a particular operating system and for a particular version of the macro. However, due to machine and macro differences, some results may not be exactly reproducible everywhere, although you would expect the efficiency differences to be slight.

submat= *number-list*
specifies a submatrix for which efficiency calculations are desired. Specify an index vector. For example, with 3 three-level factors, `a`, `b`, and `c`, and the model `class(a b c a*b)`, specify `submat=1:6`, to see the efficiency of just the 6×6 matrix of main effects. Specify `submat=3:6`, to see the efficiency of just the 4×4 matrix of `b` and `c` main effects.

types= *integer-list*
specifies the number of sets of each type to put into the design. This option is used when you have multiple types of choice sets and you want the design to consist of only certain numbers of each type. This option can be specified with the set-swapping algorithm. The argument is an integer list. When you specify `types=`, you must also specify `typevar=`. Say you are creating a design with 30 choice sets, and you want the first 10 sets to consist of sets whose `typevar=` variable in the candidate set is type 1, and you want the rest to be type 2. You would specify `types=10 20`.

typevar= *variable*
specifies a variable in the candidate data set that contains choice set types. The types must be integers starting with 1. This option can only be specified with the set-swapping algorithm. When you specify `typevar=`, you must also specify `types=`.

weight= *weight-variable*
specifies an optional weight variable. Typical usage is with an availability design. Give unavailable alternatives a weight of zero and available alternatives a weight of one. The number of alternatives must always be constant, so varying numbers of alternatives are handled by giving unavailable or unseen alternatives a weight of zero.

%ChoicEff Macro Notes

This macro specifies `options nonotes` throughout most of its execution. If you want to see all of the notes, submit the statement `%let mktopts = notes;` before running the macro. To see the macro version, submit the statement `%let mktopts = version;` before running the macro.

%MktAllo Macro

The %MktAllo autocall macro is used for manipulating data for an allocation choice experiment. See page 345 for an example. The %MktAllo macro takes as input a data set with one row for each alternative of each choice set. For example, in a study with 10 brands plus a constant alternative and 27 choice sets, there are $27 \times 11 = 297$ observations in the input data set. Here is an example of an input data set. It contains a choice set variable, product attributes (Brand and Price) and a frequency variable (Count) that contains the total number of times that each alternative was chosen.

Obs	Set	Brand	Price	Count
1	1			0
2	1	Brand 1	$50	103
3	1	Brand 2	$75	58
4	1	Brand 3	$50	318
5	1	Brand 4	$100	99
6	1	Brand 5	$100	54
7	1	Brand 6	$100	83
8	1	Brand 7	$75	71
9	1	Brand 8	$75	58
10	1	Brand 9	$75	100
11	1	Brand 10	$50	56
.				
.				
.				
296	27	Brand 9	$100	94
297	27	Brand 10	$50	65

The end result is a data set with twice as many observations that contains the number of times each alternative was chosen and the number of times it was not chosen. This data set also contains a variable c with values 1 for first choice and 2 for second or subsequent choice.

Obs	Set	Brand	Price	Count	c
1	1			0	1
2	1			1000	2
3	1	Brand 1	$50	103	1
4	1	Brand 1	$50	897	2
5	1	Brand 2	$75	58	1
6	1	Brand 2	$75	942	2
7	1	Brand 3	$50	318	1
8	1	Brand 3	$50	682	2
.					
.					
.					

```
593    27     Brand 10     $50       65    1
594    27     Brand 10     $50      935    2
```

Here is an example of usage:

```
%mktallo(data=allocs2, out=allocs3, nalts=11,
         vars=set brand price, freq=Count)
```

The option `data=` names the input data set, `out=` names the output data set, `nalts=` specifies the number of alternatives, `vars=` names the variables in the data set that will be used in the analysis excluding the `freq=` variable, and `freq=` names the frequency variable.

%MktAllo Macro Options

The following options can be used with the `%MktAllo` macro.

Option	Description
`data=`*SAS-data-set*	input SAS data set
`freq=`*variable*	frequency variable
`nalts=`*n*	number of alternatives
`out=`*SAS-data-set*	output SAS data set
`vars=`*variable-list*	input variables

You must specify the `nalts=`, `freq=`, and `vars=` options.

data= *SAS-data-set*
specifies the input SAS data set. By default, the macro uses the last data set created.

freq= *variable*
specifies the frequency variable, which contains the number of times this alternative was chosen. This option must be specified.

nalts= *n*
specifies the number of alternatives (including if appropriate the constant alternative). This option must be specified.

out= *SAS-data-set*
specifies the output SAS data set. The default is `out=allocs`.

vars= *variable-list*
specifies the variables in the data set that will be used in the analysis but not the `freq=` variable. This option must be specified.

%MktAllo Macro Notes

This macro specifies `options nonotes` throughout most of its execution. If you want to see all of the notes, submit the statement `%let mktopts = notes;` before running the macro. To see the macro version, submit the statement `%let mktopts = version;` before running the macro.

%MktBal Macro

The %MktBal macro creates factorial designs using an algorithm that ensures that the design is perfectly balanced, or when the number of levels of a factor does not divide the number of runs, as close to perfectly balanced as possible. Do not use the %MktBal macro until you have tried the %MktEx macro and determined that it does not make a design that is balanced enough for your needs. The %MktEx macro can directly create thousands of orthogonal and balanced designs that the %MktBal algorithm will never be able to find. Even when the %MktEx macro cannot create an orthogonal and balanced design, it will usually find a nearly balanced design. Designs created with the %MktBal macro, while perfectly balanced, may be less efficient than designs found with the %MktEx macro, and for large problems, the %MktBal macro can be slow. It is likely that the current algorithm used by the %MktBal macro will be changed in the future to use some now unknown algorithm that is both faster and better.

The %MktBal macro is *not* a full-featured experimental design generator. For example, you cannot specify interactions that you want to estimate or specify restrictions such as which levels may or may not appear together. You must use the %MktEx macro for that. The %MktBal macro builds a design by creating a balanced first factor, optimally blocking it to create the second factor, then optimally blocking the first two factors to create the third, and so on. Once it creates all factors, it refines each factor. Each factor is in turn removed from the design, and the rest of the design is reblocked, replacing the initial factor if the new design is more *D*-efficient.

Here is a simple example of creating a design with 2 two-level factors and 3 three-level factors in 18 runs. The %MktEval macro evaluates the results. This design is in fact optimal.

```
%mktbal(2 2 3 3 3, n=18, seed=151)
%mkteval;
```

In all cases, the factors are named x1, x2, x3, and so on.

This next example, at 120 runs and with factor levels greater than 5, is starting to get big, and by default, it will run slowly. You can use the maxstarts=, maxtries=, and maxiter= options to make the macro run more quickly. For example, the second example shown next runs much faster than the first.

```
%mktbal(2 3 4 5 6 7 8 9 10, n=120, options=progress, seed=17)

%mktbal(2 3 4 5 6 7 8 9 10, n=120, options=progress, seed=17,
        maxstarts=1, maxiter=1, maxtries=1)
```

%MktBal Macro Options

The following options can be used with the `%MktBal` macro.

Option	Description
`iter=`*n*	maximum iterations (designs to create)
`list`	list of the numbers of levels
`maxiter=`*n*	maximum iterations (designs to create)
`maxstarts=`*n*	maximum number of random starts
`maxtries=`*n*	times to try refining each factor
`n=`*n*	number of runs in the design
`options=`*options-list*	binary options
`out=`*SAS-data set*	output experimental design
`seed=`*n*	random number seed

list

specifies a list of the numbers of levels of all the factors. For example, for 3 two-level factors specify either `2 2 2` or `2 ** 3`. Lists of numbers, like `2 2 3 3 4 4` or a *levels**number of factors* syntax like: `2**2 3**2 4**2` can be used, or both can be combined: `2 2 3**4 5 6`. The specification `3**4` means 4 three-level factors. You must specify a list. Note that the factor list is a positional parameter. This means it must come first, and unlike all other parameters, it is not specified after a name and an equal sign.

n= *n*

specifies the number of runs in the design. You must specify `n=`. You can use the `%MktRuns` macro to get suggestions for values of `n=`.

out= *SAS-data set*

specifies the output experimental design. The default is `out=design`.

These next options control some of the details of the `%MktBal` macro.

maxiter= *n*
iter= *n*

specifies the maximum iterations (designs to create). By default, `maxiter=5`.

maxstarts= *n*

specifies the maximum number of random starts for each factor. With larger values, the macro tends to find slightly better designs at a cost of slower run times. The default is `maxstarts=10`.

maxtries= *n*

specifies the maximum number of times to try refining each factor after the initialization stage. The default is `maxtries=10`.

options= *options-list*

specifies binary options. By default, none of these options are specified. Specify one or more of the following values after `options=`.

`noprint`
specifies that the final *D*-efficiency should not be printed.

`progress`
reports on the macro's progress. For large numbers of factors, a large number or runs, or when the number of levels is large, this macro is slow. The `options=progress` specification gives you information about which step is being executed.

seed= *n*

specifies the random number seed. By default, `seed=0`, and clock time is used to make the random number seed. By specifying a random number seed, results should be reproducible within a SAS release for a particular operating system and for a particular version of the macro. However, due to machine and macro differences, some results may not be exactly reproducible everywhere, although you would expect the efficiency differences to be slight.

%MktBal Macro Notes

This macro specifies `options nonotes` throughout most of its execution. If you want to see all of the notes, submit the statement `%let mktopts = notes;` before running the macro. To see the macro version, submit the statement `%let mktopts = version;` before running the macro.

%MktBlock Macro

The %MktBlock autocall macro is used to block a choice design or an ordinary linear experimental design. See pages 244 and 308 for examples. When a choice design is too large to show all choice sets to each subject, the design is blocked and a block of choice sets is shown to each subject. For example, if there are 36 choice sets, instead of showing each subject 36 sets, you could instead create 2 blocks and show 2 groups of subjects 18 sets each. You could also create 3 blocks of 12 choice sets or 4 blocks of 9 choice sets. You can also request just one block if you want to see the correlations and frequencies among all of the attributes of all of the alternatives of a choice design.

The design can be in one of two formats. Typically, a choice design has one row for each alternative of each choice set and one column for each of the attributes. Typically, this kind of design is produced by either the %ChoicEff or %MktRoll macro. Alternatively, a "linear" design is an intermediate step in preparing some choice designs.[‡] The linear design has one row for each choice set and one column for each attribute of each alternative. Typically, the linear design is produced by the %MktEx macro. The output from the %MktBlock macro is a data set containing the design, with the blocking variable added and hence not in the original order, with runs or choice sets nested within blocks.

The macro tries to create a blocking factor that is uncorrelated with every attribute of every alternative. In other words, the macro is trying to optimally add one additional factor, a blocking factor, to the linear design. It is trying to make a factor that is orthogonal to all of the attributes of all of the alternatives. For linear designs, you can usually ask for a blocking factor directly as just another factor in the design, and then use the %MktLab macro to provide a name like Block, or you can use the %MktBlock macro.

Here is an example of creating the blocking variable directly.

```
%mktex(3 ** 7, n=27, seed=350)

%mktlab(vars=x1-x6 Block)
```

Here is an example of creating a design and then blocking it.

```
%mktex(3 ** 6, n=27, seed=350)

%mktblock(data=randomized, nblocks=3, seed=377, maxiter=50)
```

The output shows that the blocking factor is uncorrelated with all of the factors in the design. This output comes from the %MktEval macro, which is called by the %MktBlock macro.

[‡]See page 60 for an illustration of linear versus choice designs.

Canonical Correlations Between the Factors
There are 0 Canonical Correlations Greater Than 0.316

	Block	x1	x2	x3	x4	x5	x6
Block	1	0	0	0	0	0	0
x1	0	1	0	0	0	0	0
x2	0	0	1	0	0	0	0
x3	0	0	0	1	0	0	0
x4	0	0	0	0	1	0	0
x5	0	0	0	0	0	1	0
x6	0	0	0	0	0	0	1

Summary of Frequencies
There are 0 Canonical Correlations Greater Than 0.316

	Frequencies
Block	9 9 9
x1	9 9 9
x2	9 9 9
x3	9 9 9
x4	9 9 9
x5	9 9 9
x6	9 9 9
Block x1	3 3 3 3 3 3 3 3 3
Block x2	3 3 3 3 3 3 3 3 3
Block x3	3 3 3 3 3 3 3 3 3
Block x4	3 3 3 3 3 3 3 3 3
Block x5	3 3 3 3 3 3 3 3 3
Block x6	3 3 3 3 3 3 3 3 3
x1 x2	3 3 3 3 3 3 3 3 3
x1 x3	3 3 3 3 3 3 3 3 3
x1 x4	3 3 3 3 3 3 3 3 3
x1 x5	3 3 3 3 3 3 3 3 3
x1 x6	3 3 3 3 3 3 3 3 3
x2 x3	3 3 3 3 3 3 3 3 3
x2 x4	3 3 3 3 3 3 3 3 3
x2 x5	3 3 3 3 3 3 3 3 3
x2 x6	3 3 3 3 3 3 3 3 3
x3 x4	3 3 3 3 3 3 3 3 3
x3 x5	3 3 3 3 3 3 3 3 3
x3 x6	3 3 3 3 3 3 3 3 3
x4 x5	3 3 3 3 3 3 3 3 3
x4 x6	3 3 3 3 3 3 3 3 3
x5 x6	3 3 3 3 3 3 3 3 3
N-Way	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
	1 1 1 1 1 1 1 1

Canonical Correlations Between the Factors by Block

Block		x1	x2	x3	x4	x5	x6
1	x1	1	0.58	0	0.58	0.58	0
	x2	0.58	1	0	0	0.58	0.58
	x3	0	0	1	0	0	0
	x4	0.58	0	0	1	0.58	0.58
	x5	0.58	0.58	0	0.58	1	0.58
	x6	0	0.58	0	0.58	0.58	1
2	x1	1	0.58	0	0.58	0.58	0
	x2	0.58	1	0	0	0.58	0.58
	x3	0	0	1	0	0	0
	x4	0.58	0	0	1	0.58	0.58
	x5	0.58	0.58	0	0.58	1	0.58
	x6	0	0.58	0	0.58	0.58	1
3	x1	1	0.58	0	0.58	0.58	0
	x2	0.58	1	0	0	0.58	0.58
	x3	0	0	1	0	0	0
	x4	0.58	0	0	1	0.58	0.58
	x5	0.58	0.58	0	0.58	1	0.58
	x6	0	0.58	0	0.58	0.58	1

Notice that even with a perfect blocking variable like we have in this example, canonical correlations within each block will not be all zero.

Here is the blocked linear design (3 blocks of nine choice sets). Note that in the linear version of the design, there is one row for each choice set and all of the attributes of all of the alternatives are in the same row.

Block	Run	x1	x2	x3	x4	x5	x6
1	1	1	1	3	1	3	1
	2	1	1	1	3	1	3
	3	1	2	2	1	3	2
	4	3	3	2	3	2	1
	5	2	1	2	2	1	3
	6	3	2	1	2	3	2
	7	2	3	1	1	2	1
	8	2	3	3	2	1	2
	9	3	2	3	3	2	3

Block	Run	x1	x2	x3	x4	x5	x6
2	1	2	1	3	1	2	2
	2	3	1	2	2	3	1
	3	2	2	2	1	2	3
	4	2	1	1	3	3	1
	5	3	3	3	2	3	3
	6	3	3	1	1	1	2
	7	1	3	2	3	1	2
	8	1	2	3	3	1	1
	9	1	2	1	2	2	3

Block	Run	x1	x2	x3	x4	x5	x6
3	1	3	1	3	1	1	3
	2	2	3	2	3	3	3
	3	3	2	2	1	1	1
	4	1	1	2	2	2	2
	5	2	2	1	2	1	1
	6	2	2	3	3	3	2
	7	3	1	1	3	2	2
	8	1	3	1	1	3	3
	9	1	3	3	2	2	1

Next, we will create and block a choice design with two blocks of nine sets instead of blocking the linear version of a choice design.

```
%mktex(3 ** 6, n=3**6)

* Create an efficient choice design;
data key;
   input (x1-x3) ($);
   datalines;
x1 x2 x3
x4 x5 x6
;

%mktroll(design=design, key=key, out=out)

%choiceff(data=out, model=class(x1-x3), nsets=18, nalts=2,
          beta=zero, options=nodups, seed=151)

* Block the choice design.  Ask for 2 blocks;
%mktblock(data=best, nalts=2, nblocks=2, factors=x1-x3, seed=472)
```

(Note that if this had been a branded example, and if you were running SAS version 8.2 or an earlier release, specify `id=brand`; do not add your brand variable to the factor list. For SAS 9.0 and later SAS releases, it is fine to add your brand variable to the factor list.)

Both the design and the blocking are not as good this time. The variable names in the output are composed of `Alt`, the alternative number, and the factor name. Since there are two alternatives each composed of three factors plus one blocking variable ($2 \times 3 + 1 = 7$), a 7×7 correlation matrix is reported. Here is some of the output.

Canonical Correlations Between the Factors
There are 11 Canonical Correlations Greater Than 0.316

	Block	Alt1_x1	Alt1_x2	Alt1_x3	Alt2_x1	Alt2_x2	Alt2_x3
Block	1	0.13	0	0	0.15	0.13	0
Alt1_x1	0.13	1	0.36	0.33	0.63	0.29	0.26
Alt1_x2	0	0.36	1	0.47	0.34	0.59	0.47
Alt1_x3	0	0.33	0.47	1	0.37	0.30	0.60
Alt2_x1	0.15	0.63	0.34	0.37	1	0.23	0.36
Alt2_x2	0.13	0.29	0.59	0.30	0.23	1	0.35
Alt2_x3	0	0.26	0.47	0.60	0.36	0.35	1

Summary of Frequencies
There are 11 Canonical Correlations Greater Than 0.316
* - Indicates Unequal Frequencies

		Frequencies
	Block	9 9
*	Alt1_x1	7 7 4
*	Alt1_x2	8 2 8
*	Alt1_x3	8 4 6
*	Alt2_x1	5 5 8
*	Alt2_x2	5 9 4
*	Alt2_x3	4 8 6
*	Block Alt1_x1	4 3 2 3 4 2
*	Block Alt1_x2	4 1 4 4 1 4
*	Block Alt1_x3	4 2 3 4 2 3
*	Block Alt2_x1	2 3 4 3 2 4
*	Block Alt2_x2	3 4 2 2 5 2
*	Block Alt2_x3	2 4 3 2 4 3
*	Alt1_x1 Alt1_x2	3 1 3 4 1 2 1 0 3
*	Alt1_x1 Alt1_x3	4 2 1 3 1 3 1 1 2
*	Alt1_x1 Alt2_x1	0 4 3 2 0 5 3 1 0
*	Alt1_x1 Alt2_x2	3 3 1 1 4 2 1 2 1
*	Alt1_x1 Alt2_x3	1 4 2 2 2 3 1 2 1
*	Alt1_x2 Alt1_x3	4 1 3 2 0 0 2 3 3
*	Alt1_x2 Alt2_x1	1 3 4 1 0 1 3 2 3

```
*    Alt1_x2 Alt2_x2    0 5 3 1 0 1 4 4 0
*    Alt1_x2 Alt2_x3    2 2 4 0 2 0 2 4 2
*    Alt1_x3 Alt2_x1    3 3 2 1 1 2 1 1 4
*    Alt1_x3 Alt2_x2    2 5 1 2 1 1 1 3 2
*    Alt1_x3 Alt2_x3    0 5 3 1 0 3 3 3 0
*    Alt2_x1 Alt2_x2    1 3 1 1 3 1 3 3 2
*    Alt2_x1 Alt2_x3    0 3 2 2 2 1 2 3 3
*    Alt2_x2 Alt2_x3    0 3 2 3 3 3 1 2 1
     N-Way              1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Note that in this example, the input is a choice design (as opposed to the linear version of a choice design) so the results are in choice design format. There is one row for each alternative of each choice set.

Block	Set	Alt	x1	x2	x3
1	1	1	3	1	3
		2	2	3	1
1	2	1	2	1	1
		2	1	2	3
.					
.					
.					
Block	Set	Alt	x1	x2	x3
2	1	1	2	1	1
		2	3	2	3
2	2	1	2	2	1
		2	1	3	2
.					
.					
.					

%MktBlock Macro Options

The following options can be used with the `%MktBlock` macro.

Option	Description
`alt=`*variable*	alternative number variable
`block=`*variable*	block number variable
`data=`*SAS-data-set*	either the choice or linear design
`factors=`*variable-list*	factors in the design
`id=`*variable-list*	variables to copy to output data set
`initblock=`*variable*	initial blocking variable
`iter=`*n*	times to try to block the design
`list=`*n*	list larger canonical correlations
`maxiter=`*n*	times to try to block the design
`nalts=`*n*	number of alternatives in choice set
`nblocks=`*n*	number of blocks to create
`next=`*n*	where to look for the next exchange
`options=`*options-list*	binary options
`out=`*SAS-data-set*	output data set with block numbers
`outr=`*SAS-data-set*	randomized output data set
`print=`*print-options*	printing options
`ridge=`*n*	ridging factor
`seed=`*n*	random number seed
`set=`*variable*	choice set number variable
`vars=`*variable-list*	factors in the design

alt= *variable*
specifies the alternative number variable. If this variable is in the input data set, it is excluded from the factor list. The default is `alt=Alt`.

block= *variable*
specifies the block number variable. If this variable is in the input data set, it is excluded from the factor list. The default is `block=Block`.

data= *SAS-data-set*
specifies either the choice or linear design. The choice design has one row for each alternative of each choice set and one column for each of the attributes. Typically, this design is produced by either the `%ChoicEff` or `%MktRoll` macro. For choice designs, you must also specify the `nalts=` option. By default, the macro uses the last data set created. The linear design has one row for each choice set and one column for each attribute of each alternative. Typically, this design is produced by the `%MktEx` macro. This is the design that is input into the `%MktRoll` macro.

factors= *variable-list*
vars= *variable-list*
specifies the factors in the design. By default, all numeric variables are used, except variables with names matching those specified in the `block=`, `set=`, `alt=`, and `id=` options. (By default, the variables `Block`, `Set`, `Run`, and `Alt` are excluded from the factor list.) If you are using version 8.2 or an earlier SAS release with a branded choice design (assuming the brand factor is called `Brand`), specify `id=Brand`. Do not add the brand factor to the factor list unless you are using SAS 9.0 or a later SAS release.

id= *variable-list*
specifies the `data=` data set variables to copy to the output data set. If you are using version 8.2 or an earlier SAS release with a branded choice design (assuming the brand factor is called `Brand`), specify `id=Brand`. Do not add the brand factor to the factor list unless you are using SAS 9.0 or a later SAS release.

initblock= *variable*
specifies the name of the variable in the data set that is to be used as the initial blocking variable for the first iteration.

list= r
lists canonical correlations larger than `list=`r. The default is $r = 0.316 \approx \sqrt{r^2 = 0.1}$.

maxiter= n
iter= n
specifies the number of times to try to block the design starting with a different random blocking. By default, the macro tries five random starts, and iteratively refines each until D-efficiency quits improving, then in the end selects the blocking with the best D-efficiency.

nalts= n
specifies the number of alternatives in each choice set. If you are inputting a choice design, you must specify `nalts=`, otherwise the macro assumes you are inputting a linear design.

nblocks= n
specifies the number of blocks to create. The option `nblocks=1` just reports information about the design. The `nblocks=` option must be specified.

next= n
specifies how far into the design to go to look for the next exchange. The specification `next=1` specifies that the macro should try exchanging the level for each run with the level for the next run and all other runs. The specification `next=2` considers exchanges with half of the other runs, which makes the algorithm run more quickly. The macro considers exchanging the level for run i with run $i + 1$ then uses the `next=` value to find the next potential exchanges. Other values, including nonintegers can be specified as well. For example `next=1.5` considers exchanging observation 1 with observations 2, 4, 5, 7, 8, 10, 11, and so on. With smaller values, the macro tends to find a slightly better blocking variable at a cost of much slower run time.

options= *options-list*
specifies binary options. By default, no binary options are specified. Specify the following value after `options=`.

`nosort`
do not sort the design into blocks. This is useful anytime you want the order of the observations in the output data set to match the order of the observations in the input data set. You will typically not want to specify `options=nosort` when you are using the `%MktBlock` macro to block a design. However, `options=nosort` is handy when you are using the `%MktBlock` macro to add just another factor to the design.

out= *SAS-data-set*
specifies the output data set with the block numbers. The default is `out=blocked`. Often, you will want to specify a two-level name to create a permanent SAS data set so the design will be available later for analysis.

outr= *SAS-data-set*
specifies the randomized output data set if you would like the design randomly sorted within blocks. Often, you will want to specify a two-level name to create a permanent SAS data set so the design will be available later for analysis.

print= *print-options*
specifies both the `%MktBlock` and the `%MktEval` macro printing options, which control the printing of the results. The default is `print=normal`. Specify one or more values from the following list.

`all`	all printed output
`corr`	canonical correlations
`block`	canonical correlations within blocks
`design`	blocked design
`freqs`	long frequencies list
`list`	list of big canonical correlations
`nonzero`	like `ordered` but sets `list=1e-6`
`noprint`	no printed output
`normal`	`corr list summ block design note`
`note`	blocking note
`ordered`	like `list` but ordered by variable names
`short`	`corr summ note`
`summ`	frequency summaries

ridge= n
specifies the value to add to the diagonal of $\mathbf{X}'\mathbf{X}$ to make it nonsingular. Usually, you will not need to change this value. If you do, you probably will not notice any effect. Specify `ridge=0` to use a generalized inverse instead of ridging. The default is `ridge=0.01`.

seed= n
specifies the random number seed. By default, `seed=0`, and clock time is used to make the random number seed. By specifying a random number seed, results should be reproducible within a SAS release

for a particular operating system and for a particular version of the macro. However, due to machine and macro differences, some results may not be exactly reproducible everywhere, although you would expect the efficiency differences to be slight.

set= *variable*
specifies the choice set number variable. When `nalts=` is specified, the default is `Set`, otherwise the default is `Run`. If this variable is in the input data set, it is excluded from the factor list.

%MktBlock Macro Notes

This macro specifies `options nonotes` throughout most of its execution. If you want to see all of the notes, submit the statement `%let mktopts = notes;` before running the macro. To see the macro version, submit the statement `%let mktopts = version;` before running the macro.

%MktDes Macro

The %MktDes autocall macro creates efficient experimental designs. Usually, we will not need to call the %MktDes macro directly. Instead, we will usually use the %MktEx autocall macro, which calls the %MktDes macro as one of its many tools. At the heart of the %MktDes macro are PROC PLAN, PROC FACTEX, and PROC OPTEX. We use a macro instead of calling these procedures directly because the macro has a simpler syntax. You specify the names of the factors and the number of levels for each factor. You also specify the number of runs you want in your final design. Here for example is how you can create a design in 18 runs with 2 two-level factors (x1 and x2) and 3 three-level factors (x3, x4, and x5).

```
%mktdes(factors=x1-x2=2 x3-x5=3, n=18)
```

You can optionally specify interactions that you want to be estimable. The macro creates a candidate design in which every effect you want to be estimable is estimable, but the candidate design is bigger than you want. By default, the candidate set is stored in a SAS data set called CAND1. The macro then uses PROC OPTEX to search the candidate design for an efficient final design. By default, the final experimental design is stored in a SAS data set called DESIGN.

When the full-factorial design is small (by default less than 2189 runs, although sizes up to 5000 or 6000 runs are reasonably small), the experimental design problem is straightforward. First, the macro uses PROC PLAN to create a full-factorial candidate set. Next, PROC OPTEX searches the full-factorial candidate set. For very small problems (a few hundred candidates) PROC OPTEX will often find the optimal design, and for larger problems, it may not find *the* optimal design, but given sufficient iteration (for example, specify iter=100 or more) it will find very good designs. Run time will typically be a few seconds or a few minutes, but it could be longer. Here is a typical example of using the %MktDes macro to find an optimal nonorthogonal design when the full-factorial design is small (108 runs):

```
*---2 two-level factors and 3 three-level factors in 18 runs---;
%mktdes(factors=x1-x2=2 x3-x5=3, n=18, maxiter=500)
```

When the full-factorial design is larger, the macro uses PROC FACTEX to create a fractional-factorial candidate set. In those cases, the methods found in the %MktEx macro usually make better designs than those found with the %MktDes macro.

%MktDes Macro Options

The following options can be used with the `%MktDes` macro.

Option	Description
`big=`*n*	size of big candidate set
`cand=`*SAS-data-set*	candidate design
`classopts=`*options*	`class` statement options
`coding=`*name*	`coding=` option
`examine=`I | V	matrices that you want to examine
`facopts=`*options*	PROC FACTEX statement options
`factors=`*factor-list*	factors and levels for each factor
`generate=`*options*	`generate` statement options
`interact=`*interaction-list*	interaction terms
`iter=`*n*	number of designs
`keep=`*n*	number of designs to keep
`maxiter=`*n*	number of designs
`method=`*name*	search method
`n=`*n* | SATURATED	number of runs
`nlev=`*n*	number of levels for pseudo-factors
`options=`*options-list*	binary options
`otherfac=`*variable-list*	other factors
`otherint=`*terms*	multi-step interaction terms
`out=`*SAS-data-set*	output experimental design
`procopts=`*options*	PROC OPTEX statement options
`run=`*procedure-list*	list of procedures that may be run
`seed=`*n*	random number seed
`size=`*n* | MIN	candidate-set size
`step=`*n*	step number
`where=`*where-clause*	`where` clause

big= *n*

specifies the size at which the candidate set is considered to be big. By default, `big=2188`. If the size of the full-factorial design is less than or equal to this size, and if PROC PLAN is in the `run=` list, the macro uses PROC PLAN instead of PROC FACTEX to create the candidate set. The default of 2188 is $\max(2^{11}, 3^7) + 1$. Specifying values as large as `big=6000` or even slightly more is often reasonable. However, run time is slower as the size of the candidate set increases. The `%MktEx` macro coordinate-exchange algorithm will usually work better than a candidate-set search when the full-factorial design has more than several thousand runs.

cand= *SAS-data-set*

specifies the output data set with the candidate design (from PROC FACTEX or PROC PLAN). The default name is `Cand` followed by the step number, for example: `Cand1` for step 1, `Cand2` for step 2, and so on. You should only use this option when you are reading an external candidate set. When you specify `step=` values greater than 1, the macro assumes the default candidate set names, CAND1, CAND2, and so on, were used in previous steps. Specify just a data set name, no data set options.

classopts= *options*
specifies PROC OPTEX `class` statement options. The default, is `classopts=param=orthref`. You probably never want to change this option.

coding= *name*
specifies the PROC OPTEX `coding=` option. This option is usually not needed.

examine= I | V
specifies the matrices that you want to examine. The option `examine=I` prints the information matrix, $\mathbf{X}'\mathbf{X}$; `examine=V` prints the variance matrix, $(\mathbf{X}'\mathbf{X})^{-1}$; and `examine=I V` prints both. By default, these matrices are not printed.

facopts= *options*
specifies PROC FACTEX statement options.

factors= *factor-list*
specifies the factors and the number of levels for each factor. The `factors=` option must be specified. All other options are not required. Here is a simple example of creating a design with 10 two-level factors.

```
%mktdes(factors=x1-x10=2)
```

First, a factor list, which is a valid SAS variable list, is specified. The factor list must be followed by an equal sign and an integer, which gives the number of levels. Multiple lists can be specified. For example, to create 5 two-level factors, 5 three-level factors, and 5 five-level factors, specify:

```
%mktdes(factors=x1-x5=2 x6-x10=3 x11-x15=5)
```

By default, this macro creates each factor in a fractional-factorial candidate set from a minimum number of pseudo-factors. Pseudo-factors are not output; they are used to create the factors of interest and then discarded. For example, with `nlev=2`, a three-level factor `x1` is created from 2 two-level pseudo-factors (`_1` and `_2`) and their interaction by coding down:

```
(_1=1, _2=1) -> x1=1
(_1=1, _2=2) -> x1=2
(_1=2, _2=1) -> x1=3
(_1=2, _2=2) -> x1=1
```

This creates imbalance−the 1 level appears twice as often as 2 and 3. Somewhat better balance can be obtained by instead using three pseudo-factors. The number of pseudo-factors may be specified in parentheses after the number of levels. Example:

```
%mktdes(factors=x1-x5=2 x6-x10=3(3))
```

The levels 1 to 8 are coded down to 1 2 3 1 2 3 1 3, which is better balanced. The cost is candidate-set size may increase and efficiency may actually decrease. Some researchers are willing to sacrifice a little bit of efficiency in order to achieve better balance.

generate= *options*
specifies the PROC OPTEX `generate` statement options. By default, additional options are not added to the `generate` statement.

interact= *interaction-list*
specifies interactions that must be estimable. By default, no interactions are guaranteed to be estimable. Examples:

```
interact=x1*x2
interact=x1*x2 x3*x4*x5
interact=x1|x2|x3|x4|x5@2
```

The interaction syntax is like PROC GLM's and many of the other modeling procedures. It uses "`*`" for simple interactions (`x1*x2` is the interaction between `x1` and `x2`), "`|`" for main effects and interactions (`x1|x2|x3` is the same as `x1 x2 x1*x2 x3 x1*x3 x2*x3 x1*x2*x3`) and "`@`" to eliminate higher-order interactions (`x1|x2|x3@2` eliminates `x1*x2*x3` and is the same as `x1 x2 x1*x2 x3 x1*x3 x2*x3`). The specification "`@2`" allows only main effects and two-way interactions. Only "`@`" values of `2` or `3` are allowed.

iter= n
maxiter= n
specifies the PROC OPTEX `iter=` option which creates n designs. By default, `iter=10`.

keep= n
specifies the PROC OPTEX `keep=` option which keeps the n best designs. By default, `keep=5`.

nlev= n
specifies the number of levels from which factors are constructed through pseudo-factors and coding down. The value must be a prime or a power of a prime: 2, 3, 4, 5, 7, 8, 9, 11 .... This option is used with PROC FACTEX:

```
factors factors / nlev=&nlev;
```

By default, the macro uses the minimum prime or power of a prime from the `factors=` list or 2 if no suitable value is found.

method= *name*
specifies the PROC OPTEX `method=` search method option. The default is `method=m_Fedorov` (modified Fedorov).

n= n | **SATURATED**
specifies the PROC OPTEX `n=` option, which is the number of runs in the final design. The default is the PROC OPTEX default and depends on the problem. Typically, you will not want to use the default. Instead, you should pick a value using the information produced by the `%MktRuns` macro as guidance (see page 740). The `n=saturated` option creates a design with the minimum number of runs.

options= *options-list*
specifies binary options. By default, none of these options are specified. Specify one or more of the following values after `options=`.

`check`
checks the efficiency of a given design, specified in `cand=`.

`nocode`
suppresses printing the PROC PLAN, PROC FACTEX, and PROC OPTEX code.

`allcode`
shows all code, even code that will not be run.

otherfac= *variable-list*
specifies other terms to mention in the `factors` statement of PROC FACTEX. These terms are not guaranteed to be estimable. By default, there are no other factors.

otherint= *terms*
specifies interaction terms that will only be specified with PROC OPTEX for multi-step macro invocations. By default, no interactions are guaranteed to be estimable. Normally, interactions that are specified via the `interact=` option affect both the PROC FACTEX and the PROC OPTEX `model` statements. In multi-step problems, part of an interaction may not be in a particular PROC FACTEX step. In that case, the interaction term must only appear in the PROC OPTEX step. For example, if `x1` is created in one step and `x4` is created in another, and if the `x1*x4` interaction must be estimable, specify `otherint=x1*x4` on the final step, the one that runs PROC OPTEX.

```
%mktdes(step=1, factors=x1-x3=2, n=30, run=factex)

%mktdes(step=2, factors=x4-x6=3, n=30, run=factex)

%mktdes(step=3, factors=x7-x9=5, n=30, run=factex optex,
        otherint=x1*x4)
```

out= *SAS-data-set*
specifies the output experimental design (from PROC OPTEX). By default, `out=design`. Often, you will want to specify a two-level name to create a permanent SAS data set so the design will be available later for analysis.

procopts= *options*
specifies PROC OPTEX statement options. By default, no options are added to the PROC OPTEX statement.

run= *procedure-list*
specifies the list of procedures that the macro may run. Normally, the macro runs either PROC FACTEX or PROC PLAN and then PROC OPTEX. By default, `run=plan factex optex`. You can skip steps by omitting procedure names from this list. When both PLAN and FACTEX are in the list, the macro chooses between them based on the size of the full-factorial design and the value of `big=`.

When PLAN is not in the list, the macro generates code for PROC FACTEX.

seed= *n*

specifies the random number seed. By default, `seed=0`, and clock time is used to make the random number seed. By specifying a random number seed, results should be reproducible within a SAS release for a particular operating system and for a particular version of the macro. However, due to machine and macro differences, some results may not be exactly reproducible everywhere, although you would expect the efficiency differences to be slight.

size= *n* | MIN

specifies the candidate-set size. Start with the default `size=min` and see how big that design is. If you want, subsequently you can specify larger values that are `nlev=`*n* multiples of the minimum size. This option is used with PROC FACTEX:

```
size design=&size;
```

When `nlev=`*n*, increase the `size=` value by a factor of *n* each time. For example, when `nlev=2`, increase the `size=` value by a factor of two each time. If `size=min` implies `size=128`, then 256, 512, 1024, and 2048 are reasonable sizes to try. Integer expressions like `size=128*4` are allowed.

step= *n*

specifies the step number. By default, there is only one step. However, sometimes, a better design can be found using a multi-step approach. Do not specify the `cand=` option on any step of a multi-step run. Consider the problem of making a design with 3 two-level factors, 3 three-level factors, and 3 five-level factors. The simplest approach is to do something like this−create a design from two-level factors using pseudo-factors and coding down.

```
%mktdes(factors=x1-x3=2 x4-x6=3 x7-x9=5, n=30)
```

However, for small problems like this, the following three-step approach will usually be better.

```
%mktdes(step=1, factors=x1-x3=2, n=30, run=factex)
%mktdes(step=2, factors=x4-x6=3, n=30, run=factex)
%mktdes(step=3, factors=x7-x9=5, n=30, run=factex optex)
```

Note however, that the following `%MktEx` macro call will usually be better still.

```
%mktex(2 2 2 3 3 3 5 5 5, n=30)
```

The first `%MktDes` macro step uses PROC FACTEX to create a fractional-factorial design for the two-level factors. The second step uses PROC FACTEX to create a fractional-factorial design for the three-level factors and cross it with the two-level factors. The third step uses PROC FACTEX to create a fractional-factorial design for the five-level factors and cross it with the design for the two and three-level factors and then run PROC OPTEX.

Each step globally stores two macro variables (`&class1` and `&inter1` for the first step, `&class2` and `&inter2` for the second step, ...) that are used to construct the PROC OPTEX `class` and `model` statements. When `step` $>$ 1, variables from the previous steps are used in the `class` and `model` statements. In this example, the following PROC OPTEX code is created by step 3:

```
proc optex data=Cand3;
   class
      x1-x3
      x4-x6
      x7-x9
      / param=orthref;
   model
      x1-x3
      x4-x6
      x7-x9
      ;
   generate n=30 iter=10 keep=5 method=m_fedorov;
   output out=Design;
   run; quit;
```

This step uses the previously stored macro variables `&class1=x1-x3` and `&class2=x4-x6`.

where= *where-clause*
specifies a SAS `where` clause for the candidate design, which is used to restrict the candidates. By default, the candidate design is not restricted.

%MktDes Macro Notes

This macro specifies `options nonotes` throughout much of its execution. If you want to see all of the notes, submit the statement `%let mktopts = notes;` before running the macro. To see the macro version, submit the statement `%let mktopts = version;` before running the macro.

%MktDups Macro

The `%MktDups` autocall macro detects duplicate choice sets and duplicate alternatives within generic choice sets. See page 406 for an example. To illustrate, consider a simple experiment with these two choice sets. These choice sets are completely different and are not duplicates.

```
a b c       a b c
1 2 1       1 1 1
2 1 2       2 2 2
1 1 2       2 2 1
2 1 1       1 2 2
```

Now consider these two choice sets:

```
a b c       a b c
1 2 1       2 1 2
2 1 2       1 1 2
1 1 2       2 1 1
2 1 1       1 2 1
```

They are the same for a generic study because all of the same alternatives are there, they are just in a different order. However, for a branded study they are different. For a branded study, there would be a different brand for each alternative, so the choice sets would be the same only if all the same alternatives appeared in the same order. For both a branded and generic study, these choice sets are duplicates:

```
a b c       a b c
1 2 1       1 2 1
2 1 2       2 1 2
1 1 2       1 1 2
2 1 1       2 1 1
```

Now consider these choice sets for a generic study.

```
a b c       a b c
1 2 1       1 2 1
2 1 1       1 2 1
1 1 2       1 1 2
2 1 1       2 1 1
```

First, each of these choice sets has duplicate alternatives (2 1 1 in the first and 1 2 1 in the second). Second, these two choice sets are flagged as duplicates, even though they are not exactly the same. They are flagged as duplicates because every alternative in choice set one is also in choice set two, and every alternative in choice set two is also in choice set one. In generic studies, two choice sets are considered duplicates unless one has one or more alternatives that are not in the other choice set.

Here is an example. A design is created with the %ChoicEff macro choice-set-swapping algorithm for a branded study, then the %MktDups macro is run to check for and eliminate duplicate choice sets.

```
%mktex(3 ** 9, n=27, seed=424)

data key;
   input (Brand x1-x3) ($);
   datalines;
Acme   x1 x2 x3
Ajax   x4 x5 x6
Widgit x7 x8 x9
;

%mktroll(design=randomized, key=key, alt=brand, out=cand)

%choiceff(data=cand, model=class(brand x1-x3), seed=420,
          nsets=18, nalts=3, beta=zero)

proc freq; tables set; run;

%mktdups(branded, data=best, factors=brand x1-x3, nalts=3, out=out)

proc freq; tables set; run;
```

The first PROC FREQ output shows us that several candidate choice sets occur more than once in the design.

The FREQ Procedure

Set	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	6	11.11	6	11.11
3	3	5.56	9	16.67
4	6	11.11	15	27.78
13	6	11.11	21	38.89
15	6	11.11	27	50.00
16	3	5.56	30	55.56
21	3	5.56	33	61.11
22	3	5.56	36	66.67
23	6	11.11	42	77.78
25	9	16.67	51	94.44
27	3	5.56	54	100.00

The %MktDups macro prints the following information to the log:

```
Design:          Branded
Factors:         brand x1-x3
                 Brand
                 x1 x2 x3
Duplicate Sets:  7
```

The output from the %MktDups macro contains the following table:

Choice Set	Duplicate Choice Sets To Delete
1	5
2	16
3	9
	17
7	13
8	15
11	14

The first line of the first table tells us that this is a branded design as opposed to generic. The second line tells us the factors as specified on the factors= option. These are followed by the actual variable names for the factors. The last line reports the number of duplicates. The second table tells us that choice set 1 is the same as choice set 5. Similarly, 2 and 16 are the same as are 3, 9, and 17, and so on. The out= data set will contain the design with the duplicate choice set eliminated.

Now consider an example with purely generic alternatives.

```
%mktex(2 ** 5, n=2**5, seed=109)
%mktlab(int=f1-f4)

%choiceff(data=final, model=class(x1-x5), seed=93,
          nsets=42, flags=f1-f4, beta=zero)
%mktdups(generic, data=best, factors=x1-x5, nalts=4, out=out)
```

The macro produces the following tables:

```
Design:          Generic
Factors:         x1-x5
                 x1 x2 x3 x4 x5
Sets w Dup Alts: 1
Duplicate Sets:  1
```

Choice Set	Duplicate Choice Sets To Delete	
2	25	
39		Alternatives

For each choice set listed in the choice set column, either the other choice sets it duplicates are listed or the word `Alternatives` is printed if the problem is with duplicate alternatives.

Here are just the choice sets with duplication problems:

```
proc print data=best;
   var x1-x5;
   id set; by set;
   where set in (2, 25, 39);
   run;
```

Set	x1	x2	x3	x4	x5
2	1	2	1	1	1
	2	2	1	1	1
	1	1	2	2	2
	2	1	2	2	2
25	1	1	2	2	2
	2	1	2	2	2
	2	2	1	1	1
	1	2	1	1	1
39	1	1	2	1	1
	1	1	2	1	1
	2	2	1	2	2
	2	2	1	2	2

You can see that the macro detects duplicates even though the alternatives do not always appear in the same order in the different choice sets.

Now consider another example.

```
%mktex(2 ** 6, n=2**6)

data key;
   input (x1-x2) ($) @@;
   datalines;
x1 x2 x3 x4 x5 x6
;
%mktroll(design=design, key=key, out=cand)
```

```
%mktdups(generic, data=cand, factors=x1-x2, nalts=3, out=out)

proc print; by set; id set; run;
```

Here is some of the output. The output lists, for each set of duplicates, the choice set that will be kept (in the first column) and all the matching choice sets that will be deleted (in the second column).

```
Design:          Generic
Factors:         x1-x2
                 x1 x2
Sets w Dup Alts: 40
Duplicate Sets:  50
```

Choice Set	Duplicate Choice Sets To Delete	
1		Alternatives
2		Alternatives
	5	
	6	
	17	
	18	
	21	
.		
.		
.		

Here are the unique choice sets.

Set	_Alt_	x1	x2
7	1	1	1
	2	1	2
	3	2	1
8	1	1	1
	2	1	2
	3	2	2
12	1	1	1
	2	2	1
	3	2	2
28	1	1	2
	2	2	1
	3	2	2

This next example creates a conjoint design[§] and tests it for duplicates.

```
%mktex( 3 ** 3 2 ** 2, n=19, seed=121)

%mktdups(linear, factors=x1-x5)
```

```
Design:          Linear
Factors:         x1-x5
                 x1 x2 x3 x4 x5
Duplicate Runs:  2
```

Run	Duplicate Runs To Delete
3	4
10	11

%MktDups Macro Options

The following options can be used with the `%MktDups` macro.

Option	Description
`data=`*SAS-data-set*	input choice design
`factors=`*variable-list*	factors in the design
`nalts=`*n*	number of alternatives
`options`	binary options
`out=`*SAS-data-set*	output data set
`outlist=`*SAS-data-set*	output data set with duplicates
`vars=`*variable-list*	factors in the design

Positional Parameter

The options list is a positional parameter. This means it must come first, and unlike all other parameters, it is not specified after a name and an equal sign.

options

For the first option, specify one or more of the following. You may specify `noprint` and one of the following: `generic`, `branded`, or `linear`.

`branded`
specifies that since one of the factors is brand, the macro only needs to compare corresponding alternatives in each choice set.

[§]Normally, we would use 18 runs and not a prime number like 19 that is not divisible by any of the numbers of levels, 2 and 3. We picked a silly number like 19 to ensure duplicates for this example.

`generic`
specifies a generic design and is the default. This means that there are no brands, so options are interchangeable, so the macro needs to compare each alternative with every other alternative in every choice set.

`linear`
specifies a linear not a choice design. Specify `linear` for a full-profile conjoint design, for an ANOVA design, or for the linear version of a branded choice design.

`noprint`
specifies no printed output. This option will be used when you are only interested in the output data set or macro variable.

Example:

```
%mktdups(branded noprint, nalts=3)
```

Required Options

This next option is mandatory with choice designs.

nalts= *n*
specifies the number of alternatives. This option must be specified with generic or branded designs. It is ignored with linear designs. For generic or branded designs, the `data=` data set must contain `nalts=` observations for the first choice set, `nalts=` observations for the second choice set, and so on.

Other Options

Here are the other options.

data= *SAS-data-set*
specifies the input choice design. By default, the macro uses the last data set created.

out= *SAS-data-set*
specifies an output data set that contains the design with duplicate choice sets excluded. By default, no data set is created, and the macro just reports on duplicates. Often, you will want to specify a two-level name to create a permanent SAS data set so the design will be available later for analysis.

outlist= *SAS-data-set*
specifies the output data set with the list of duplicates. By default, `outlist=outdups`.

vars= *variable-list*
factors= *variable-list*
specifies the factors in the design. By default, all numeric variables are used.

%MktDups Macro Notes

This macro specifies `options nonotes` throughout most of its execution. If you want to see all of the notes, submit the statement `%let mktopts = notes;` before running the macro. To see the macro version, submit the statement `%let mktopts = version;` before running the macro.

%MktEval Macro

The %MktEval autocall macro helps you evaluate an experimental design. There are numerous examples of its usage from pages 161 through 348. The %MktEval macro reports on balance and orthogonality. Typically, you will call it immediately after running the %MktEx macro. The output from this macro contains two default tables. The first table shows the canonical correlations between pairs of coded factors. A canonical correlation is the maximum correlation between linear combinations of the coded factors. See page 70 for more information about canonical correlations. All zeros off the diagonal show that the design is orthogonal for main effects. Off-diagonal canonical correlations greater than 0.316 ($r^2 > 0.1$) are listed in a separate table.

For nonorthogonal designs and designs with interactions, the canonical-correlation matrix is not a substitute for looking at the variance matrix with the %MktEx macro. It just provides a quick and more-compact picture of the correlations between the factors. The variance matrix is sensitive to the actual model specified and the coding. The canonical-correlation matrix just tells you if there is some correlation between the main effects. When is a canonical correlation too big? You will have to decide that for yourself. In part, the answer depends on the factors and how the design will be used. A high correlation between the client's and the main competitor's price factor is a serious problem meaning you will need to use a different design. In contrast, a moderate correlation in a choice design between one brand's minor attribute and another brand's minor attribute may be perfectly fine.

The macro also prints one-way, two-way and n-way frequencies. Equal one-way frequencies occur when the design is balanced. Equal two-way frequencies occur when the design is orthogonal. Equal n-way frequencies, all equal to one, occur when there are no duplicate runs or choice sets.

Here is a typical usage:

```
%mktex(2 2 3 ** 6, n=18, unbalanced=0, seed=289)
%mkteval;
```

Canonical Correlations Between the Factors
There is 1 Canonical Correlation Greater Than 0.316

	x1	x2	x3	x4	x5	x6	x7	x8
x1	1	0.33	0	0	0	0	0	0
x2	0.33	1	0	0	0	0	0	0
x3	0	0	1	0	0	0	0	0
x4	0	0	0	1	0	0	0	0
x5	0	0	0	0	1	0	0	0
x6	0	0	0	0	0	1	0	0
x7	0	0	0	0	0	0	1	0
x8	0	0	0	0	0	0	0	1

Canonical Correlations > 0.316 Between the Factors
There is 1 Canonical Correlation Greater Than 0.316

		r	r Square
x1	x2	0.33	0.11

```
                Summary of Frequencies
There is 1 Canonical Correlation Greater Than 0.316
          * - Indicates Unequal Frequencies

                Frequencies

       x1       9 9
       x2       9 9
       x3       6 6 6
       x4       6 6 6
       x5       6 6 6
       x6       6 6 6
       x7       6 6 6
       x8       6 6 6
 *     x1 x2    3 6 6 3
       x1 x3    3 3 3 3 3 3
       x1 x4    3 3 3 3 3 3
       x1 x5    3 3 3 3 3 3
       x1 x6    3 3 3 3 3 3
       x1 x7    3 3 3 3 3 3
       x1 x8    3 3 3 3 3 3
       x2 x3    3 3 3 3 3 3
       x2 x4    3 3 3 3 3 3
       x2 x5    3 3 3 3 3 3
       x2 x6    3 3 3 3 3 3
       x2 x7    3 3 3 3 3 3
       x2 x8    3 3 3 3 3 3
       x3 x4    2 2 2 2 2 2 2 2 2
       x3 x5    2 2 2 2 2 2 2 2 2
       x3 x6    2 2 2 2 2 2 2 2 2
       x3 x7    2 2 2 2 2 2 2 2 2
       x3 x8    2 2 2 2 2 2 2 2 2
       x4 x5    2 2 2 2 2 2 2 2 2
       x4 x6    2 2 2 2 2 2 2 2 2
       x4 x7    2 2 2 2 2 2 2 2 2
       x4 x8    2 2 2 2 2 2 2 2 2
       x5 x6    2 2 2 2 2 2 2 2 2
       x5 x7    2 2 2 2 2 2 2 2 2
       x5 x8    2 2 2 2 2 2 2 2 2
       x6 x7    2 2 2 2 2 2 2 2 2
       x6 x8    2 2 2 2 2 2 2 2 2
       x7 x8    2 2 2 2 2 2 2 2 2
       N-Way    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

All factors in this design are perfectly balanced, and almost all are orthogonal, but `x1` and `x2` are correlated with each other.

%MktEval Macro Options

The following options can be used with the `%MktEval` macro.

Option	Description
`blocks=`*variable*	blocking variable
`data=`*SAS-data-set*	input data set with design
`factors=`*variable-list*	factors in the design
`format=`*format*	format for canonical correlations
`freqs=`*frequency-list*	frequencies to print
`list=`*n*	minimum canonical correlation to list
`outcb=`*SAS-data-set*	within-block canonical correlations
`outcorr=`*SAS-data-set*	canonical correlation matrix
`outfreq=`*SAS-data-set*	frequencies
`outfsum=`*SAS-data-set*	frequency summaries
`outlist=`*SAS-data-set*	list of largest canonical correlations
`print=`*print-options*	controls the printing of the results
`vars=`*variable-list*	list of the factors

blocks= *variable*
specifies a blocking variable. This option prints separate canonical correlations within each block. By default, there is one block.

data= *SAS-data-set*
specifies the input SAS data set with the experimental design. By default, the macro uses the last data set created.

factors= *variable-list*
vars= *variable-list*
specifies a list of the factors in the experimental design. The default is all of the numeric variables in the data set.

freqs= *frequency-list*
specifies the frequencies to print. By default, `freqs=1 2 n`, and 1-way, 2-way, and *n*-way frequencies are printed. Do not specify the exact number of ways instead of `n`. For ways other than `n`, the macro checks for and prints zero cell frequencies. For *n*-ways, the macro does not output or print zero frequencies. Only the full-factorial design will have nonzero cells, so specifying something like `freqs=1 2 20` will make the macro take a *long* time, and it will try to create *huge* data sets and will probably run out of memory or disk space before it is done. However, `freqs=1 2 n` runs very reasonably.

format= *format*
specifies the format for printing canonical correlations. The default format is `4.2`.

list= *n*
specifies the minimum canonical correlation to list. The default is 0.316, the square root of $r^2 = 0.1$.

outcorr= *SAS-data-set*
specifies the output SAS data set for the canonical correlation matrix. The default data set name is CORR.

outcb= *SAS-data-set*
specifies the output SAS data set for the within-block canonical correlation matrices. The default data set name is CB.

outlist= *SAS-data-set*
specifies the output data set for the list of largest canonical correlations. The default data set name is LIST.

outfreq= *SAS-data-set*
specifies the output data set for the frequencies. The default data set name is FREQ.

outfsum= *SAS-data-set*
specifies the output data set for the frequency summaries. The default data set name is FSUM.

print= *print-options*
controls the printing of the results. The default is `print=short`. Specify one or more values from the following list.

`all`	all printed output
`corr`	prints the canonical correlations matrix
`block`	prints the canonical correlations within block
`freqs`	prints the frequencies, specified by the `freqs=` option
`list`	prints the list of canonical correlations greater than the `list=` value
`nonzero`	like `ordered` but sets `list=1e-6`
`ordered`	like `list` but ordered by variable names
`short`	is the default and is equivalent to: `corr list summ block`
`summ`	prints the frequency summaries
`noprint`	no printed output

By default, the frequency list, which contains the factor names, levels, and frequencies is not printed, but the more compact frequency summary list, which contains the factors and frequencies but not the levels is printed.

%MktEval Macro Notes

This macro specifies `options nonotes` throughout most of its execution. If you want to see all of the notes, submit the statement `%let mktopts = notes;` before running the macro. To see the macro version, submit the statement `%let mktopts = version;` before running the macro.

%MktEx Macro

The %MktEx autocall macro is designed for researchers who need good, efficient factorial designs. There are numerous examples of its usage from pages 158 through 465. The %MktEx macro is designed to be very simple to use and to run in seconds for trivial problems, minutes for small problems, and in less than an hour for larger and difficult problems. This macro is a full-featured factorial-experimental designer that can handle simple problems like main-effects designs and more complicated problems including designs with interactions and restrictions on which levels can appear together. The macro is designed to easily create the kinds of designs that marketing researchers need for conjoint and choice experiments. For most factorial-design problems, you can simply run the macro once, specifying only the number of runs and the numbers of levels of all the factors. You will no longer have to try different algorithms and different approaches to see which one works best. The macro does all of that for you. We state on page 100 "The best approach to design creation is to use the computer as a tool along with traditional design skills, not as a substitute for thinking about the problem." With the %MktEx macro, we try to automate some of the thought processes of the expert designer.

Here is an example of using the %MktEx macro to create a design with 5 two-level factors, 4 three-level factors, 3 five-level factors, 2 six-level factors, all in 60 runs (row, experimental conditions, conjoint profiles, or choice sets).

```
%mktex( 2 ** 5  3 ** 4  5 5 5  6 6,  n=60 )
```

The notation `m ** n` means m^n or n m-level factors. For example `2 ** 5` means $2 \times 2 \times 2 \times 2 \times 2$ or 5 two-level factors.

The %MktEx macro creates efficient factorial designs using several approaches. The macro will try to directly create an orthogonal design (strength-two orthogonal array), it will search a set of candidate runs (rows of the design), and it will use a coordinate-exchange algorithm using both random initial designs and also a partially orthogonal design initialization. The macro stops if at any time it finds a perfect, 100% efficient, orthogonal and balanced design. This first phase is the algorithm search phase. In it, the macro determines which approach is working best for this problem. At the end of this phase, the macro chooses the method that has produced the best design and performs another set of iterations using exclusively the chosen approach. Finally, the macro performs a third set of iterations where it takes the best design it found so far and tries to improve it.

In all phases, the macro attempts to optimize *D*-efficiency (sometimes known as *D*-optimality), which is a standard measure of the goodness of the experimental design. As *D*-efficiency increases, the standard errors of the parameter estimates in the linear model decrease. A perfect design is orthogonal and balanced and has 100% *D*-efficiency. A design is orthogonal when all of the parameter estimates are uncorrelated. A design is balanced when all of the levels within each of the factors occur equally often. A design is orthogonal and balanced when the variance matrix, which is proportional to $(\mathbf{X}'\mathbf{X})^{-1}$ is diagonal, where $\mathbf{X}$ is a suitable orthogonal coding (see page 64) of the design matrix. See pages 47 and 99, for more information on efficient experimental designs.

For most problems, you only need to specify the levels of all the factors and the number of runs. For more complicated problems, you may need to also specify the interactions that you want to be estimable or restrictions on which levels may not appear together. Other than that, you should not need any other options for most problems. This macro is not like other design tools that you have to tell what to do. With this macro, you just tell it what you want, and it figures out a good way to do it. For some problems, the sophisticated user, with a lot of work, may be able to adjust the options to come up with a better design. However, this macro should always produce a very good design with

minimal effort for even the most unsophisticated users.

The %MktEx macro has the world's largest catalog of strength-two (main effects) orthogonal arrays. The orthogonal arrays are constructed using methods and arrays from a variety of sources, including: Addelman (1962a,b); Bose (1947); Dawson (1985); De Cock and Stufken (2000); de Launey (1986, 1987a,b); Dey (1985); Hadamard (1893); Hedayat, Sloane, and Stufken (1999); Kharaghania and Tayfeh-Rezaiea (2004); Kuhfeld (2004); Paley (1933); Rao (1947); Sloane (2004); Suen (1989a,b, 2003a,b,c); Suen and Kuhfeld (2004); Taguchi (1987); Wang and Wu (1989, 1991); Wang (1996a,b); Williamson(1944); Xu (2002); Zhang, Lu and Pang(1999); Zhang, Pang and Wang (2001); Zhang, Weiguo, Meixia and Zheng (2004); and the SAS FACTEX procedure. Most of the newest designs come from new difference schemes, and in particular, new generalized Hadamard matrices based on work by de Launey, Dawson, and Zhang and colleagues.

For n's up through 256 that are a multiple of 4, and many n's beyond that, the %MktEx macro can construct orthogonal designs with up to $n-1$ two-level factors. The two-level designs are constructed from Hadamard matrices (Hadamard, 1893; Paley, 1933; Williamson, 1944; Hedayat, Sloane, and Stufken, 1999). The next table shows the available sizes up through n=1000:

Hadamard Matrix Sizes Up to n=1000															
4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64
68	72	76	80	84	88	92	96	100	104	108	112	116	120	124	128
132	136	140	144	148	152	156	160	164	168	172	176	180	184	188	192
196	200	204	208	212	216	220	224	228	232	236	240	244	248	252	256
264	272	276	280	284	288	296	300	304	308	312	316	320	328	332	336
344	348	352	360	364	368	376	380	384	388	392	396	400	408	416	428
420	424	428	432	440	444	448	456	460	464	468	472	480	484	488	492
496	500	504	512	516	524	528	540	544	548	552	556	560	564	568	572
576	588	592	600	608	616	620	624	628	632	636	640	644	656	660	664
672	676	684	688	692	696	700	704	708	720	728	736	740	748	752	760
768	776	780	784	788	792	796	800	804	812	816	820	828	832	840	844
848	860	864	868	880	884	888	896	900	908	912	916	920	924	928	936
944	948	960	968	972	976	984	992	1000							

Larger sizes are available as well. The %MktEx macro can construct these designs when n is a multiple of 4 and one or more of the following hold:

- $n \leq 256$
- $n-1$ is prime
- $n/2-1$ is prime and $\text{mod}(n/2, 4) = 2$
- n is a power of 2 (2, 4, 8, 16, ...) times the size of a smaller Hadamard matrix that is available.

%MktEx can also make the 428-run Hadamard matrix that was recently discovered by Kharaghania and Tayfeh-Rezaiea (2004).

When n is a multiple of 8, the macro can create orthogonal designs with a small number (say m) four-level factors in place of $3 \times m$ of the two-level factors (for example, $2^{70}\ 4^3$ in 80 runs).

You can see more sizes of Hadamard matrices that the macro knows how to make by running the next program. Note however that the fact that a number appears in this program's listing, does not guarantee that your computer will have enough memory and other resources to create it.

```
data x;
   length Method $ 30;
   do n = 4 to 10000 by 4;
      HadSize = n; method = ' ';

      do while(mod(hadsize, 8) eq 0); hadsize = hadsize / 2; end;
      link paley;
      if method eq ' ' and hadsize le 256 then method = 'Williamson';
      if method eq ' ' and hadsize gt 256 then do;
         do while(hadsize lt n and method eq ' ');
            hadsize = hadsize * 2;
            link paley;
            end;
         end;
      if method eq ' ' and hadsize = 428 then
         method = 'Kharaghania and Tayfeh-Rezaiea';
      if method ne ' ' then do; Change = n - lag(n); output; end;
      end;
   return;
   paley:;
      ispm1 = 1; ispm2 = mod(hadsize / 2, 4) eq 2;
      h = hadsize - 1;
      do i = 3 to sqrt(hadsize) by 2 while(ispm1);
         ispm1 = mod(h, i);
         end;
      h = hadsize / 2 - 1;
      do i = 3 to sqrt(hadsize / 2) by 2 while(ispm2);
         ispm2 = mod(h, i);
         end;
      if      ispm1 then method = 'Paley 1';
      else if ispm2 then method = 'Paley 2';
      return;
   run;

options ps=11000;
proc print label noobs;
   label hadsize = 'Reduced Hadamard Matrix Size';
   var n hadsize method change;
   run;
```

Here is a summary of the parent and design sizes up through 513 runs that are available with the `%MktEx` macro. Designs with 128, 144, 256, 432, 448, and 512 runs are listed separately from the rest of their category since there are so many of them.

Number of Runs	Parents		All Designs	
4- 50	86	12.03%	181	0.16%
51-100	176	24.62%	596	0.51%
101-127 129-150	154	21.54%	281	0.24%
128	21	2.94%	740	0.63%
144	16	2.24%	1,241	1.06%
151-200	43	6.01%	1,071	0.92%
201-250	41	5.73%	1,101	0.94%
251-255 257-300	32	4.48%	3,311	2.84%
256	2	0.28%	6,101	5.23%
301-350	35	4.90%	2,160	1.85%
351-400	39	5.45%	4,649	3.99%
401-431 433-447 449-450	21	2.94%	422	0.36%
432	7	0.98%	10,246	8.79%
448	4	0.56%	8,598	7.37%
451-500	32	4.48%	1,787	1.53%
501-511 513	4	0.56%	113	0.10%
512	2	0.28%	73,992	63.46%
	715		116,590	

Shown next are the 715 parent designs. (Listing all of the 116,590 designs in this format, at 200 designs per page, would require 583 pages.) Many more orthogonal arrays not explicitly in this catalog can be created as well such as full-factorial designs with more than 144 runs, Hadamard designs with more than 512 runs, and fractional-factorial designs in 256, 512, or more runs.

4 2^3
6 $2^1 3^1$
8 $2^4 4^1$
9 3^4
10 $2^1 5^1$
12 2^{11}
$2^4 3^1$
$2^2 6^1$
$3^1 4^1$
14 $2^1 7^1$
15 $3^1 5^1$
16 $2^8 8^1$
4^5
18 $2^1 9^1$
$3^6 6^1$
20 2^{19}
$2^8 5^1$
$2^2 10^1$
$4^1 5^1$
21 $3^1 7^1$
22 $2^1 11^1$
24 $2^{20} 4^1$
$2^{13} 3^1 4^1$
$2^{12} 12^1$
$2^{11} 4^1 6^1$
$3^1 8^1$
25 5^6
26 $2^1 13^1$
27 $3^9 9^1$
28 2^{27}
$2^{12} 7^1$
$2^2 14^1$
$4^1 7^1$
30 $2^1 15^1$
$3^1 10^1$
$5^1 6^1$
32 $2^{16} 16^1$
$4^8 8^1$
33 $3^1 11^1$
34 $2^1 17^1$
35 $5^1 7^1$
36 2^{35}
$2^{27} 3^1$
$2^{20} 3^2$
$2^{18} 3^1 6^1$
$2^{13} 6^2$
$2^{13} 9^1$
$2^{10} 3^8 6^1$
$2^{10} 3^1 6^2$
$2^9 3^4 6^2$
$2^8 6^3$
$2^4 3^1 6^3$
$2^3 3^9 6^1$
$2^3 3^2 6^3$
$2^2 3^5 6^2$
$2^2 18^1$
$2^1 3^3 6^3$
$3^{12} 12^1$
$3^7 6^3$
$4^1 9^1$
38 $2^1 19^1$
39 $3^1 13^1$
40 $2^{36} 4^1$
$2^{25} 4^1 5^1$
$2^{20} 20^1$
$2^{19} 4^1 10^1$
$5^1 8^1$
42 $2^1 21^1$
$3^1 14^1$
$6^1 7^1$
44 2^{43}
$2^{12} 11^1$
$2^2 22^1$
$4^1 11^1$
45 $3^9 15^1$
$5^1 9^1$
46 $2^1 23^1$
48 $2^{40} 8^1$
$2^{33} 3^1 8^1$
$2^{31} 6^1 8^1$
$2^{24} 24^1$
$3^1 16^1$
$4^{12} 12^1$
49 7^8
50 $2^1 25^1$
$5^{10} 10^1$
51 $3^1 17^1$
52 2^{51}
$2^{12} 13^1$
$2^2 26^1$
$4^1 13^1$
54 $2^1 27^1$
$3^{20} 6^1 9^1$
$3^{18} 18^1$
55 $5^1 11^1$
56 $2^{52} 4^1$
$2^{37} 4^1 7^1$
$2^{28} 28^1$
$2^{27} 4^1 14^1$
$7^1 8^1$
57 $3^1 19^1$
58 $2^1 29^1$
60 2^{59}
$2^{30} 3^1$
$2^{22} 5^1$
$2^{18} 6^1$
$2^{18} 10^1$
$2^{15} 3^1 5^1$
$2^{13} 3^1 10^1$
$2^{13} 5^1 6^1$
$2^{13} 15^1$
$2^{11} 6^1 10^1$
$2^2 30^1$
$3^1 20^1$
$4^1 15^1$
$5^1 12^1$
62 $2^1 31^1$
63 $3^{12} 21^1$
$7^1 9^1$
64 $2^{32} 32^1$
$2^5 4^{17} 8^1$
$2^5 4^{10} 8^4$
$4^{16} 16^1$
$4^{14} 8^3$
$4^7 8^6$
8^9
65 $5^1 13^1$
66 $2^1 33^1$
$3^1 22^1$
$6^1 11^1$
68 2^{67}
$2^{13} 17^1$
$2^2 34^1$
$4^1 17^1$
69 $3^1 23^1$
70 $2^1 35^1$
$5^1 14^1$
$7^1 10^1$
72 $2^{68} 4^1$
$2^{60} 3^1 4^1$
$2^{53} 3^2 4^1$
$2^{51} 3^1 4^1 6^1$
$2^{46} 4^1 6^2$
$2^{46} 4^1 9^1$
$2^{44} 3^{12} 4^1$
$2^{43} 3^8 4^1 6^1$
$2^{43} 3^1 4^1 6^2$
$2^{42} 3^4 4^1 6^2$
$2^{41} 4^1 6^3$
$2^{37} 3^{13} 4^1$
$2^{37} 3^1 4^1 6^3$
$2^{36} 3^9 4^1 6^1$
$2^{36} 3^2 4^1 6^3$
$2^{36} 36^1$
$2^{35} 3^{12} 4^1 6^1$
$2^{35} 3^5 4^1 6^2$
$2^{35} 4^1 18^1$
$2^{34} 3^8 4^1 6^2$
$2^{34} 3^3 4^1 6^3$
$2^{31} 6^4$
$2^{30} 3^1 6^4$
$2^{28} 3^2 6^4$
$2^{27} 3^{11} 6^1 12^1$
$2^{27} 3^6 6^4$
$2^{19} 3^{20} 4^1 6^1$
$2^{18} 3^{16} 4^1 6^2$
$2^{17} 3^{12} 4^1 6^3$
$2^{16} 3^8 4^1 6^4$
$2^{12} 3^{21} 4^1 6^1$
$2^{11} 3^{20} 6^1 12^1$
$2^{11} 3^{17} 4^1 6^2$
$2^{10} 3^{20} 4^1 6^2$
$2^{10} 3^{16} 6^2 12^1$
$2^{10} 3^{13} 4^1 6^3$
$2^9 3^{16} 4^1 6^3$
$2^9 3^{12} 6^3 12^1$
$2^9 3^9 4^1 6^4$
$2^8 3^{12} 4^1 6^4$
$2^7 3^7 6^5 12^1$
$2^6 3^3 6^6 12^1$

$3^{24}24^{1}$
$8^{1}9^{1}$
74 $2^{1}37^{1}$
75 $3^{1}25^{1}$
$5^{8}15^{1}$
76 2^{75}
$2^{13}19^{1}$
$2^{2}38^{1}$
$4^{1}19^{1}$
77 $7^{1}11^{1}$
78 $2^{1}39^{1}$
$3^{1}26^{1}$
$6^{1}13^{1}$
80 $2^{61}5^{1}8^{1}$
$2^{55}8^{1}10^{1}$
$2^{51}4^{3}20^{1}$
$2^{40}40^{1}$
$4^{10}20^{1}$
$5^{1}16^{1}$
81 $3^{27}27^{1}$
9^{10}
82 $2^{1}41^{1}$
84 2^{83}
$2^{28}3^{1}$
$2^{27}7^{1}$
$2^{26}6^{1}$
$2^{20}3^{1}7^{1}$
$2^{18}3^{1}14^{1}$
$2^{18}6^{1}7^{1}$
$2^{13}6^{1}14^{1}$
$2^{13}21^{1}$
$2^{2}42^{1}$
$3^{1}28^{1}$
$4^{1}21^{1}$
$7^{1}12^{1}$
85 $5^{1}17^{1}$
86 $2^{1}43^{1}$
87 $3^{1}29^{1}$
88 $2^{84}4^{1}$
$2^{53}4^{1}11^{1}$
$2^{44}44^{1}$
$2^{43}4^{1}22^{1}$
$8^{1}11^{1}$
90 $2^{1}45^{1}$
$3^{30}30^{1}$
$3^{8}6^{1}15^{1}$
$5^{1}18^{1}$
$9^{1}10^{1}$
91 $7^{1}13^{1}$
92 2^{91}
$2^{13}23^{1}$
$2^{2}46^{1}$
$4^{1}23^{1}$
93 $3^{1}31^{1}$
94 $2^{1}47^{1}$
95 $5^{1}19^{1}$
96 $2^{80}16^{1}$
$2^{73}3^{1}16^{1}$
$2^{71}6^{1}16^{1}$
$2^{48}48^{1}$
$2^{44}4^{11}8^{1}12^{1}$
$2^{43}4^{15}8^{1}$
$2^{43}4^{12}6^{1}8^{1}$
$2^{39}3^{1}4^{14}8^{1}$
$2^{12}4^{20}24^{1}$
$3^{1}32^{1}$
98 $2^{1}49^{1}$
$7^{14}14^{1}$
99 $3^{13}33^{1}$
$9^{1}11^{1}$
100 2^{99}
$2^{51}5^{3}$
$2^{40}5^{4}$
$2^{34}5^{3}10^{1}$
$2^{29}5^{5}$
$2^{18}5^{9}10^{1}$
$2^{17}5^{3}10^{2}$
$2^{13}25^{1}$
$2^{7}5^{10}10^{1}$
$2^{2}50^{1}$
$4^{1}25^{1}$
$5^{20}20^{1}$
$5^{8}10^{3}$
10^{4}
102 $2^{1}51^{1}$
$3^{1}34^{1}$
$6^{1}17^{1}$
104 $2^{100}4^{1}$
$2^{61}4^{1}13^{1}$
$2^{52}52^{1}$
$2^{51}4^{1}26^{1}$
$8^{1}13^{1}$
105 $3^{1}35^{1}$
$5^{1}21^{1}$
$7^{1}15^{1}$
106 $2^{1}53^{1}$
108 2^{107}
$2^{34}6^{1}$
$2^{27}3^{33}9^{1}$
$2^{26}3^{1}6^{1}$
$2^{20}3^{34}9^{1}$
$2^{19}3^{2}6^{1}$
$2^{18}3^{33}6^{1}9^{1}$
$2^{18}3^{31}18^{1}$
$2^{17}3^{1}6^{2}$
$2^{13}3^{30}6^{1}18^{1}$
$2^{13}27^{1}$
$2^{12}6^{3}$
$2^{10}3^{40}6^{1}9^{1}$
$2^{10}3^{33}6^{2}9^{1}$
$2^{10}3^{31}6^{1}18^{1}$
$2^{9}3^{36}6^{2}9^{1}$
$2^{9}3^{34}6^{1}18^{1}$
$2^{9}3^{1}6^{3}$
$2^{8}3^{30}6^{2}18^{1}$
$2^{7}6^{4}$
$2^{4}3^{33}6^{3}9^{1}$
$2^{4}3^{31}6^{2}18^{1}$
$2^{3}3^{41}6^{1}9^{1}$
$2^{3}3^{39}18^{1}$
$2^{3}3^{34}6^{3}9^{1}$
$2^{3}3^{32}6^{2}18^{1}$
$2^{3}3^{1}6^{4}$
$2^{2}3^{42}18^{1}$
$2^{2}3^{37}6^{2}9^{1}$
$2^{2}3^{35}6^{1}18^{1}$
$2^{2}3^{2}6^{4}$
$2^{2}54^{1}$
$2^{1}3^{35}6^{3}9^{1}$
$2^{1}3^{33}6^{2}18^{1}$
$3^{44}9^{1}12^{1}$
$3^{39}6^{3}9^{1}$
$3^{37}6^{2}18^{1}$
$3^{36}36^{1}$
$3^{3}6^{4}$
$4^{1}27^{1}$
110 $2^{1}55^{1}$
$5^{1}22^{1}$
$10^{1}11^{1}$
111 $3^{1}37^{1}$
112 $2^{104}8^{1}$
$2^{89}7^{1}8^{1}$
$2^{79}8^{1}14^{1}$
$2^{75}4^{3}28^{1}$
$2^{56}56^{1}$
$4^{12}28^{1}$
$7^{1}16^{1}$
114 $2^{1}57^{1}$
$3^{1}38^{1}$
$6^{1}19^{1}$
115 $5^{1}23^{1}$
116 2^{115}
$2^{13}29^{1}$
$2^{2}58^{1}$
$4^{1}29^{1}$
117 $3^{13}39^{1}$
$9^{1}13^{1}$
118 $2^{1}59^{1}$
119 $7^{1}17^{1}$
120 $2^{116}4^{1}$
$2^{87}3^{1}4^{1}$
$2^{79}4^{1}5^{1}$
$2^{75}4^{1}6^{1}$
$2^{75}4^{1}10^{1}$
$2^{72}4^{1}15^{1}$
$2^{70}3^{1}4^{1}10^{1}$
$2^{70}4^{1}5^{1}6^{1}$
$2^{68}4^{1}6^{1}10^{1}$
$2^{60}60^{1}$
$2^{59}4^{1}30^{1}$
$2^{30}6^{1}20^{1}$
$2^{27}10^{1}12^{1}$
$3^{1}40^{1}$
$5^{1}24^{1}$
$8^{1}15^{1}$
121 11^{12}
122 $2^{1}61^{1}$
123 $3^{1}41^{1}$
124 2^{123}
$2^{13}31^{1}$

$2^{2}62^{1}$
$4^{1}31^{1}$
125 $5^{25}25^{1}$
126 $2^{1}63^{1}$
$3^{24}14^{1}$
$3^{23}6^{1}7^{1}$
$3^{21}42^{1}$
$3^{20}6^{1}21^{1}$
$7^{1}18^{1}$
$9^{1}14^{1}$
128 $2^{64}64^{1}$
$2^{6}4^{33}8^{1}16^{1}$
$2^{6}4^{26}8^{4}16^{1}$
$2^{6}4^{19}8^{7}16^{1}$
$2^{6}4^{12}8^{10}16^{1}$
$2^{6}4^{5}8^{13}16^{1}$
$2^{5}4^{31}8^{2}16^{1}$
$2^{5}4^{24}8^{5}16^{1}$
$2^{5}4^{17}8^{8}16^{1}$
$2^{5}4^{10}8^{11}16^{1}$
$2^{5}4^{8}8^{14}$
$2^{4}4^{36}16^{1}$
$2^{4}4^{29}8^{3}16^{1}$
$2^{4}4^{22}8^{6}16^{1}$
$2^{4}4^{15}8^{9}16^{1}$
$2^{4}4^{8}8^{12}16^{1}$
$2^{3}4^{25}8^{7}$
$2^{3}4^{18}8^{10}$
$2^{3}4^{11}8^{13}$
$4^{32}32^{1}$
$8^{16}16^{1}$
129 $3^{1}43^{1}$
130 $2^{1}65^{1}$
$5^{1}26^{1}$
$10^{1}13^{1}$
132 2^{131}
$2^{42}6^{1}$
$2^{27}11^{1}$
$2^{20}3^{1}11^{1}$
$2^{18}3^{1}22^{1}$
$2^{18}6^{1}11^{1}$
$2^{13}6^{1}22^{1}$
$2^{13}33^{1}$
$2^{2}66^{1}$
$3^{1}44^{1}$
$4^{1}33^{1}$
$11^{1}12^{1}$
133 $7^{1}19^{1}$
134 $2^{1}67^{1}$
135 $3^{27}45^{1}$
$3^{20}9^{1}15^{1}$
$5^{1}27^{1}$
136 $2^{132}4^{1}$
$2^{78}4^{1}17^{1}$
$2^{68}68^{1}$
$2^{67}4^{1}34^{1}$
$8^{1}17^{1}$
138 $2^{1}69^{1}$
$3^{1}46^{1}$
$6^{1}23^{1}$
140 2^{139}
$2^{38}7^{1}$
$2^{34}14^{1}$
$2^{28}5^{1}$
$2^{27}5^{1}7^{1}$
$2^{26}10^{1}$
$2^{25}5^{1}14^{1}$
$2^{21}7^{1}10^{1}$
$2^{17}10^{1}14^{1}$
$2^{13}35^{1}$
$2^{2}70^{1}$
$4^{1}35^{1}$
$5^{1}28^{1}$
$7^{1}20^{1}$
141 $3^{1}47^{1}$
142 $2^{1}71^{1}$
143 $11^{1}13^{1}$
144 $2^{136}8^{1}$
$2^{114}8^{1}9^{1}$
$2^{113}3^{1}24^{1}$
$2^{111}6^{1}24^{1}$
$2^{103}8^{1}18^{1}$
$2^{76}3^{12}6^{4}8^{1}$
$2^{76}3^{7}4^{1}6^{5}12^{1}$
$2^{75}3^{3}4^{1}6^{6}12^{1}$
$2^{74}3^{4}6^{6}8^{1}$
$2^{72}72^{1}$
$2^{44}3^{11}12^{2}$
$2^{6}3^{3}6^{6}24^{1}$
$3^{48}48^{1}$
$4^{36}36^{1}$
$4^{11}12^{2}$
12^{7}
147 $7^{7}21^{1}$
148 2^{147}
150 $5^{10}30^{1}$
152 $2^{148}4^{1}$
$2^{76}76^{1}$
153 $3^{25}17^{1}$
156 2^{155}
160 $2^{144}16^{1}$
$2^{138}4^{7}$
$2^{133}5^{1}16^{1}$
$2^{127}10^{1}16^{1}$
$2^{80}80^{1}$
$4^{16}40^{1}$
162 $3^{65}6^{1}27^{1}$
$3^{54}54^{1}$
164 2^{163}
168 $2^{164}4^{1}$
$2^{84}84^{1}$
169 13^{14}
171 $3^{28}19^{1}$
172 2^{171}
175 $5^{10}35^{1}$
176 $2^{168}8^{1}$
$2^{166}4^{3}$
$2^{88}88^{1}$
$4^{12}44^{1}$
180 2^{179}
$3^{30}60^{1}$
$6^{2}30^{1}$
184 $2^{180}4^{1}$
$2^{92}92^{1}$
188 2^{187}
189 $3^{36}63^{1}$
192 $2^{160}32^{1}$
$2^{96}96^{1}$
$4^{48}48^{1}$
$8^{8}24^{1}$
196 2^{195}
$7^{14}28^{1}$
14^{3}
198 $3^{30}66^{1}$
200 $2^{196}4^{1}$
$2^{100}100^{1}$
$5^{20}40^{1}$
$10^{5}20^{1}$
204 2^{203}
207 $3^{25}23^{1}$
208 $2^{200}8^{1}$
$2^{198}4^{3}$
$2^{104}104^{1}$
$4^{16}52^{1}$
212 2^{211}
216 $2^{212}4^{1}$
$2^{108}108^{1}$
$2^{11}3^{77}12^{1}18^{1}$
$3^{72}72^{1}$
$3^{66}6^{5}12^{1}18^{1}$
$6^{7}36^{1}$
220 2^{219}
224 $2^{208}16^{1}$
$2^{193}7^{1}16^{1}$
$2^{183}14^{1}16^{1}$
$2^{112}112^{1}$
$4^{56}56^{1}$
225 $3^{27}75^{1}$
$5^{20}45^{1}$
15^{5}
228 2^{227}
232 $2^{228}4^{1}$
$2^{116}116^{1}$
234 $3^{30}78^{1}$
236 2^{235}
240 $2^{232}8^{1}$
$2^{230}4^{3}$
$2^{205}5^{1}24^{1}$
$2^{199}10^{1}24^{1}$
$2^{120}120^{1}$
$4^{20}60^{1}$
242 $11^{22}22^{1}$
243 $3^{81}81^{1}$
$9^{27}27^{1}$
244 2^{243}
245 $7^{7}35^{1}$
248 $2^{244}4^{1}$
$2^{124}124^{1}$
250 $5^{50}50^{1}$
252 2^{251}

	$3^{42}84^{1}$		$4^{40}80^{1}$	388	2^{387}	459	$3^{72}9^{1}17^{1}$
	$6^{2}42^{1}$		$8^{8}40^{1}$	392	$2^{388}4^{1}$	460	2^{459}
256	$8^{32}32^{1}$	324	$3^{143}12^{1}27^{1}$		$2^{196}7^{14}28^{1}$	464	$2^{456}8^{1}$
	16^{17}		$3^{108}108^{1}$		$7^{14}56^{1}$		$2^{454}4^{3}$
261	$3^{27}87^{1}$		$6^{2}54^{1}$		$14^{5}28^{1}$		$2^{348}116^{1}$
264	$2^{260}4^{1}$	325	$5^{20}65^{1}$	396	2^{395}		$4^{36}116^{1}$
	$2^{132}132^{1}$	328	$2^{324}4^{1}$		$3^{132}132^{1}$	468	2^{467}
270	$3^{90}90^{1}$	332	2^{331}		$6^{2}66^{1}$		$3^{49}52^{1}$
272	$2^{264}8^{1}$	333	$3^{36}111^{1}$	400	$2^{392}8^{1}$		$6^{2}78^{1}$
	$2^{262}4^{3}$	336	$2^{328}8^{1}$		$2^{390}4^{3}$	472	$2^{468}4^{1}$
	$2^{136}136^{1}$		$2^{326}4^{3}$		$2^{300}100^{1}$	475	$5^{20}95^{1}$
	$4^{32}68^{1}$		$2^{297}7^{1}24^{1}$		$4^{36}100^{1}$	477	$3^{37}53^{1}$
275	$5^{10}55^{1}$		$2^{287}14^{1}24^{1}$		$5^{80}80^{1}$	480	$2^{464}16^{1}$
276	2^{275}		$2^{252}84^{1}$		$10^{6}40^{1}$		$2^{458}4^{7}$
279	$3^{30}93^{1}$		$4^{36}84^{1}$	405	$3^{81}135^{1}$		$2^{360}120^{1}$
280	$2^{276}4^{1}$	338	$13^{26}26^{1}$	408	$2^{404}4^{1}$		$4^{56}120^{1}$
	$2^{140}140^{1}$	342	$3^{30}114^{1}$	414	$3^{48}138^{1}$	484	2^{483}
284	2^{283}	343	$7^{49}49^{1}$	416	$2^{400}16^{1}$		$11^{22}44^{1}$
288	$2^{272}16^{1}$	344	$2^{340}4^{1}$		$2^{394}4^{7}$		22^{3}
	$2^{250}9^{1}16^{1}$	348	2^{347}		$2^{312}104^{1}$	486	$3^{216}54^{1}$
	$2^{239}16^{1}18^{1}$	350	$5^{20}70^{1}$		$4^{48}104^{1}$	488	$2^{484}4^{1}$
	$2^{144}144^{1}$	351	$3^{39}117^{1}$	420	2^{419}	490	$7^{14}70^{1}$
	$3^{96}96^{1}$	352	$2^{336}16^{1}$	423	$3^{30}141^{1}$	492	2^{491}
	$4^{36}72^{1}$		$2^{330}4^{7}$	424	$2^{420}4^{1}$	495	$3^{42}5^{1}33^{1}$
	$6^{8}48^{1}$		$2^{264}88^{1}$	425	$5^{20}85^{1}$	496	$2^{488}8^{1}$
	$12^{6}24^{1}$		$4^{32}88^{1}$	432	$2^{424}8^{1}$		$2^{486}4^{3}$
289	17^{18}	360	$2^{356}4^{1}$		$2^{386}9^{1}24^{1}$		$2^{372}124^{1}$
294	$7^{14}42^{1}$		$3^{48}120^{1}$		$2^{375}18^{1}24^{1}$		$4^{18}124^{1}$
296	$2^{292}4^{1}$		$6^{6}60^{1}$		$3^{144}144^{1}$	500	2^{499}
297	$3^{39}99^{1}$	361	19^{20}		$4^{108}108^{1}$		$5^{100}100^{1}$
300	2^{299}	363	$11^{11}33^{1}$		$6^{12}72^{1}$		$10^{2}50^{1}$
	$5^{20}60^{1}$	364	2^{363}		$12^{6}36^{1}$	504	$2^{500}4^{1}$
	$10^{2}30^{1}$	368	$2^{360}8^{1}$	440	$2^{436}4^{1}$		$2^{84}3^{84}84^{1}$
304	$2^{296}8^{1}$		$2^{358}4^{3}$	441	$3^{42}7^{7}21^{1}$		$6^{7}84^{1}$
	$2^{294}4^{3}$		$2^{276}92^{1}$		$7^{14}63^{1}$	512	$8^{64}64^{1}$
	$2^{228}76^{1}$		$4^{36}92^{1}$		21^{5}		$16^{32}32^{1}$
	$4^{16}76^{1}$	369	$3^{30}123^{1}$	444	2^{443}	513	$3^{81}9^{1}19^{1}$
306	$3^{48}102^{1}$	375	$5^{40}75^{1}$	448	$2^{416}32^{1}$		
308	2^{307}	376	$2^{372}4^{1}$		$2^{336}112^{1}$		
312	$2^{308}4^{1}$	378	$3^{72}126^{1}$		$4^{56}112^{1}$		
315	$3^{29}105^{1}$	380	2^{379}		$8^{56}56^{1}$		
316	2^{315}	384	$2^{320}64^{1}$	450	$3^{54}5^{10}30^{1}$		
320	$2^{288}32^{1}$		$4^{96}96^{1}$		$5^{90}90^{1}$		
	$2^{274}4^{15}$		$8^{16}48^{1}$		$15^{4}30^{1}$		
	$2^{240}80^{1}$	387	$3^{48}129^{1}$	456	$2^{452}4^{1}$		

Here is a simple example of using the `%MktEx` macro to request the L_{36} design, $2^{11}3^{12}$, which has 11 two-level factors and 12 three-level factors.

```
%mktex( n=36 )
```

No iterations are needed, and the macro immediately creates the L_{36}, which is 100% efficient. This example runs in a few seconds. The factors are always named `x1`, `x2`, ... and the levels are always consecutive integers starting with 1. You can use the `%MktLab` macro to assign different names and levels (see page 712).

By default, the macro creates two output data sets with the design.

- `out=Design` - the experimental design, sorted by the factor levels.
- `outr=Randomized` - the randomized experimental design.

The two designs are equivalent and have the same *D*-efficiency. The `out=Design` data set is sorted and hence is usually easier to look at, however the `outr=Randomized` design is the better one to use. The randomized design has the rows sorted into a random order, and all of the factor levels are randomly reassigned. For example with two-level factors, approximately half of the original (1, 2) mappings are reassigned (2, 1). Similarly, with three level factors, the mapping (1, 2, 3) are changed to one of the following: (1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2), or (3, 2, 1). The reassignment of levels is usually not critical for the iteratively derived designs, but it can be very important the orthogonal designs, many of which have all ones in the first row.

The candidate-set search has two parts. First, either PROC PLAN is run to create a full-factorial design for small problems, or PROC FACTEX is run to create a fractional-factorial design for large problems. Either way, this design is a candidate set that in the second part is searched by PROC OPTEX using the modified Fedorov algorithm. A design is built from a selection of the rows of the candidate set (Fedorov, 1972; Cook and Nachtsheim, 1980). The modified Fedorov algorithm considers each run in the design and each candidate run. Candidate runs are swapped in and design runs are swapped out if the swap improves *D*-efficiency.

Next, the `%MktEx` macro uses the coordinate-exchange algorithm, based on Meyer and Nachtsheim (1995). The coordinate-exchange algorithm considers each level of each factor, and considers the effect on *D*-efficiency of changing a level ($1 \rightarrow 2$, or $1 \rightarrow 3$, or $2 \rightarrow 1$, or $2 \rightarrow 3$, or $3 \rightarrow 1$, or $3 \rightarrow 2$, and so on). Exchanges that increase efficiency are performed. Typically, the macro first tries to initialize the design with an orthogonal (or tabled) design (`Tab`) and a random design (`Ran`) both. Levels that are not orthogonally initialized may be exchanged for other levels if the exchange increases efficiency.

The initialization may be more complicated. Say you asked for the design $4^1 5^1 3^5$ in 18 runs. The macro would use the orthogonal design $3^6 6^1$ in 18 runs to initialize the three-level factors orthogonally, and the five-level factor with the six-level factor coded down to five levels (and hence unbalanced). The four-level factor would be randomly initialized. The macro would also try the same initialization but with a random rather than unbalanced initialization of the five-level factor, as a minor variation on the first initialization. In the next initialization variation, the macro would use a fully random initialization. If the number of runs requested were smaller than the number or runs in the initial orthogonal design, the macro would initialize the design with just the first n rows of the orthogonal design. Similarly, if the number of runs requested were larger than the number or runs in the initial orthogonal design, the macro would initialize part of the design with the orthogonal design and the remaining rows and columns randomly. The coordinate-exchange algorithm considers each level of each factor that is not orthogonally initialized, and it exchanges a level if the exchange improves *D*-efficiency.

When the number or runs in the orthogonal design does not match the number of runs desired, none of the design is initialized orthogonally.

The coordinate-exchange algorithm is not restricted by having a candidate set and hence can *potentially* consider every possible design. That is, no design is precluded from consideration due to the limitations of a candidate set. In practice, however, both the candidate-set-based and coordinate-exchange algorithms consider only a *tiny* fraction of the possible designs. When the number of runs in the full-factorial design is very small (say 100 or 200 runs), the modified Fedorov algorithm and coordinate exchange algorithms usually work equally well. When the number of runs in the full-factorial design is small (up to several thousand), the modified Fedorov algorithm is usually superior to coordinate exchange, particularly in finding designs with interactions. When the full-factorial design is larger, coordinate exchange is usually the superior approach. However, heuristics like these are often wrong, which is why the macro tries both methods to see which one is really best for each problem.

Next, the `%MktEx` macro determines which algorithm (candidate set search, coordinate exchange with partial orthogonal initialization, or coordinate exchange with random initialization) is working best and tries more iterations using that approach. It starts by printing the initial (`Ini`) best efficiency.

Next, the `%MktEx` macro tries to improve the best design it found previously. Using the previous best design as an initialization (`Pre`), and random mutations of the initialization (`Mut`) and simulated annealing (`Ann`), the macro uses the coordinate-exchange algorithm to try to find a better design. This step is important because the best design that the macro found may be an intermediate design and may not be the final design at the end of an iteration. Sometimes the iterations deliberately make the designs less efficient, and sometimes, the macro never finds a design as efficient or more efficient again. Hence it is worthwhile to see if the best design found so far can be improved. At the end, PROC OPTEX is called to print the levels of each factor and the final *D*-efficiency.

Random mutations involve adding random noise to the initial design before iterations start (levels are randomly changed). This may eliminate the perfect balance that will often be in the initial design. By default, random mutations are used with designs with fully random initializations and in the design refinement step; orthogonal initial designs are not mutated.

Coordinate exchange can be combined with the simulated annealing optimization technique (Kirkpatrick, Gellat, and Vecchi 1983). Annealing refers to the cooling of a liquid in a heat bath. The structure of the solid depends on the rate of cooling. Coordinate exchange without simulated annealing seeks to maximize *D*-efficiency at every step. Coordinate exchange with simulated annealing allows *D*-efficiency to occasionally decrease with a probability that decreases with each iteration. This is analogous to slower cooling, and it helps overcome local optima.

For design 1, for the first level of the first factor, by default, the macro may execute an exchange (say change a 2 to a 1) that makes the design worse with probability 0.05. As more and more exchanges occur, this probability decreases so at the end of the processing of design 1, exchanges that decrease efficiency are hardly ever done. For design 2, this same process is repeated, again starting by default with an annealing probability of 0.05. This often helps the algorithm overcome local efficiency maxima. To envision this, imagine that you are standing on a molehill next to a mountain. The only way you can start going up the mountain is to first step down off the molehill. Once you are on the mountain, you may occasionally hit a dead end, where all you can do is step down and look for a better place to continue going up. Simulated annealing, by occasionally stepping down the efficiency function, often allows the macro to go farther up it than it would otherwise. The simulated annealing is why you will sometimes see designs getting worse in the iteration history. The macro keeps track of the best design, not the final design in each step. By default, annealing is used with designs with fully random initializations and in the design refinement step. Simulated annealing is not used with orthogonally initialized designs.

%MktEx Macro Notes

The `%MktEx` macro prints notes to the SAS log to show you what it is doing while it is running. Most of the notes that would normally come out of the macro's procedure and DATA steps are suppressed by default by an `options nonotes` statement. This macro specifies `options nonotes` throughout most of its execution. If you want to see all of the notes, submit the statement `%let mktopts = notes;` before running the macro. To see the macro version, submit the statement `%let mktopts = version;` before running the macro. This section describes the notes that are normally not suppressed.

The macro will usually start by printing one of the following notes (filling in a value after `n=`).

```
NOTE: Generating the Hadamard design, n=.
NOTE: Generating the full-factorial design, n=.
NOTE: Generating the fractional-factorial design, n=.
NOTE: Generating the orthogonal array design, n=.
```

These messages tell you which type of orthogonal design the macro is constructing. The design may be the final design, or it may provide an initialization for the coordinate exchange algorithm. In some cases, it may not have the same number of runs, `n`, as the final design. Usually this step is fast, but constructing some fractional-factorial designs may be time consuming.

If the macro is going to use PROC OPTEX to search a candidate set, it will print this note.

```
NOTE: Generating the candidate set.
```

This step will usually be fast. Next, when a candidate set is searched, the macro will print this next note, substituting in values for the ellipses.

```
NOTE: Performing ... searches of ... candidates.
```

This step may be take a while depending on the size of the candidate set and the size of the design. When there are a lot of restrictions and a fractional-factorial candidate set is being used, the candidate set may be so restricted that it does not contain enough information to make the design. In that case, you will get this message.

```
NOTE: The candidate-set initialization failed,
      but the MKTEX macro is continuing.
```

Even though part of the macro's algorithm failed, it is *not* a problem. The macro just goes on to the coordinate-exchange algorithm, which will almost certainly work better than searching any severely-restricted candidate set.

For large designs, you usually will want to skip the PROC OPTEX iterations. The macro may print this note.

```
NOTE: With a design this large, you may get faster results with OPTITER=0.
```

Sometimes you will get this note.

```
NOTE: Stopping since it appears that no improvement is possible.
```

When the macro keeps finding the same maximum D-efficiency over and over again in different designs, it may stop early. This may mean that the macro has found the optimal design, or it may mean that the macro keeps finding a very attractive local optimum. Either way, it is unlikely that the macro will do any better. You can control this using the `stopearly=` option.

The macro has options that control the amount of time it spends trying different techniques. When

time expires, the macro may switch to other techniques before it completes the usual maximum number of iterations. When this happens, the macro tells you.

```
NOTE: Switching to a random initialization after ... minutes and
      ... designs.
NOTE: Quitting the algorithm search after ... minutes and ... designs.
NOTE: Quitting the design search after ... minutes and ... designs.
NOTE: Quitting the refinement step after ... minutes and ... designs.
```

When there are restrictions, or when you specify that you do not want duplicate runs, you may also specify `options=accept`. This means that you are willing to accept designs that violate the restrictions. With `options=accept`, the macro will tell you if the restrictions are not met.

```
NOTE: The restrictions were not met.
NOTE: The design has duplicate runs.
```

`%MktEx` optimizes a ridged efficiency criterion, that is, a small number is added to the diagonal of $(\mathbf{X}'\mathbf{X})^{-1}$. Usually, the ridged criterion is virtually the same as the unridged criterion. When `%MktEx` detects that this is not true, it prints these notes.

```
NOTE: The final   ridged D-efficiency criterion is ....
NOTE: The final unridged D-efficiency criterion is ....
```

The macro ends with one of the following two messages.

```
NOTE: The MKTEX macro used ... seconds.
NOTE: The MKTEX macro used ... minutes.
```

%MktEx Macro Iteration History

This section provides information on interpreting the iteration history table produced by the `%MktEx` macro. Here is part of a table.

Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
1	Start	82.2172	82.2172	Can
1	End	82.2172		
2	Start	78.5039		Tab,Ran
2	5 14	83.2098	83.2098	
2	6 14	83.3917	83.3917	
2	6 15	83.5655	83.5655	
2	7 14	83.7278	83.7278	
2	7 15	84.0318	84.0318	
2	7 15	84.3370	84.3370	
2	8 14	85.1449	85.1449	
.				
.				
.				
2	End	98.0624		

```
.
.
.
12      Start       51.8915               Ran,Mut,Ann
12        End       93.0214
.
.
.
```

Design Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
0	Initial	98.8933	98.8933	Ini
1	Start	80.4296		Tab,Ran
1	End	98.8567		

```
.
.
.
```

Design Refinement History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
0	Initial	98.9438	98.9438	Ini
1	Start	94.7490		Pre,Mut,Ann
1	End	92.1336		

```
.
.
.
```

The first column, `Design`, is a design number. Each design corresponds to a complete iteration using a different initialization. Initial designs are numbered zero. The second column is `Row,Col`, which shows the design row and column that is changing in the coordinate-exchange algorithm. This column also contains `Start` for displaying the initial efficiency, `End` for displaying the final efficiency, and `Initial` for displaying the efficiency of a previously created initial design (perhaps created externally or perhaps created in a previous step). The `Current D-Efficiency` column contains the *D*-efficiency for the design including starting, intermediate and final values. The next column is `Best D-Efficiency`. Values are put in this column for initial designs and when a design is found that is as good as or better than the previous best design. The last column, `Notes`, contains assorted algorithm and explanatory details. Values are added to the table at the beginning of an iteration, at the end of an iteration, when a better design is found, and when a design first conforms to restrictions. Details of the candidate search iterations are not shown. Only the *D*-efficiency for the best design found through candidate search is shown.

Here are the notes.

`Can`	- the results of a candidate-set search
`Tab`	- tabled (orthogonal array, full, or fractional factorial) initialization (full or in part)
`Ran`	- random initialization (full or in part)
`Unb`	- unbalanced initialization (usually in part)
`Ini`	- initial design
`Mut`	- random mutations of the initial design were performed
`Ann`	- simulated annealing was used in this iteration
`Pre`	- using previous best design as a starting point
`Conforms`	- design conforms to restrictions
`Violations`	- number of restriction violations

Often, more than one note appears. For example, the triples `Ran,Mut,Ann` and `Pre,Mut,Ann` frequently appear together.

The iteration history consists of three tables.

`Algorithm Search History`	- searches for a design and the best algorithm for this problem
`Design Search History`	- uses the best algorithm to search further
`Design Refinement History`	- tries to refine the best design

%MktEx Macro Options

The following options can be used with the `%MktEx` macro.

	Option	Description
	`anneal=`$n1 < n2 < n3 >>$	starting probability for annealing
	`annealfun=`*function*	annealing probability function
	`anniter=`$n1 < n2 < n3 >>$	first annealing iteration
*	`balance=`n	maximum allowed level-frequency range
	`big=`$n <$ *choose* $>$	size of big full-factorial design
	`canditer=`$n1 < n2 >$	iterations for OPTEX designs
	`cat=`*SAS-data-set*	input design catalog
	`detfuzz=`n	determinants change increment
	`examine=`I | V	matrices that you want to examine
	`exchange=`n	number of factors to exchange
	`fixed=`*variable*	indicates runs that are fixed
	`holdouts=`n	adds holdout observations
	`imlopts=`*options*	IML PROC statement options
*	`init=`*SAS-data-set*	initial (input) experimental design
	`interact=`*interaction-list*	interaction terms
	`iter=`$n1 < n2 < n3 >>$	maximum number of iterations
	`levels=`*value*	method for assigning final factor levels
	`list`	list of the numbers of factor levels
	`maxdesigns=`n	maximum number of designs to make
	`maxiter=`$n1 < n2 < n3 >>$	maximum number of iterations
	`maxstages=`n	maximum number of algorithm stages
	`maxtime=`$n1 < n2 < n3 >>$	approximate maximum run time
*	`mintry=`n	minimum number of rows to process
	`mutate=`$n1 < n2 < n3 >>$	mutation probability
	`mutiter=`$n1 < n2 < n3 >>$	first iteration to consider mutating
	`n=`n	number of runs in the design
*	`options=`*options-list*	binary options
	`optiter=`$n1 < n2 >$	OPTEX iterations
*	`order=`*value*	coordinate exchange column order
	`out=`*SAS-data-set*	output experimental design
	`outall=`*SAS-data-set*	output data set with all designs found
	`outr=`*SAS-data-set*	randomized output experimental design
	`partial=`n	partial-profile design
*	`repeat=`$n1\ n2\ n3$	number of times to iterate on a row
	`reslist=`*list*	constant matrix list
	`resmac=`*macro-name*	constant matrix creation macro
	`restrictions=`*macro-name*	restrictions macro
	`ridge=`n	ridging factor
	`seed=`n	random number seed
	`stopearly=`n	that the macro may stop early
	`tabiter=`$n1 < n2 >$	tabled design iterations
	`tabsize=`n	orthogonal array size
	`target=`n	target efficiency criterion
	`unbalanced=`$n1 < n2 >$	unbalanced factors iterations

$*$ - a new option or an option with new features in this release.

Required Options

These options are almost always required.

list

specifies a list of the numbers of levels of all the factors. For example, for 3 two-level factors specify either `2 2 2` or `2 ** 3`. Lists of numbers, like `2 2 3 3 4 4` or a *levels**number of factors* syntax like: `2**2 3**2 4**2` can be used, or both can be combined: `2 2 3**4 5 6`. The specification `3**4` means 4 three-level factors. Note that the factor list is a positional parameter. This means that if it is specified, it must come first, and unlike all other parameters, it is not specified after a name and an equal sign. Usually, you have to specify a list. However, in some cases, you can just specify `n=` and omit the list and a default list is implied. For example, `n=18` implies a list of `2 3 ** 7`. When the list is omitted, and if there are no interactions, restrictions, or duplicate exclusions, then by default there are no OPTEX iterations (`optiter=0`).

n= *n*

specifies the number of runs in the design. You must specify `n=`. Here is an example of using the `%MktRuns` macro to get suggestions for values of `n=`:

```
%mktruns( 4 2 ** 5 3 ** 5 )
```

In this case, this macro suggests several sizes including orthogonal designs with `n=72` and `n=144` runs and some smaller nonorthogonal designs including `n=36, 24, 48, 60`.

Basic Options

This next group of options contains some of the more commonly used options.

balance= *n*

specifies the maximum allowable level-frequency range. This option allows you to tell the macro that it should make an extra effort to ensure that the design is nearly balanced. Specify a positive integer, usually 1 or 2, that specifies the degree of imbalance that is acceptable. The `balance=n` option specifies that for each factor, a difference between the frequencies of the most and least frequently occuring levels should be no larger than n. You may specify `balance=0`, however, this usually is not a good idea because the macro needs the flexibility to have imbalance as it refines the design. Often, the design actually found will be better balanced than your `balance=n` specification would require. For this reason, it is good to start by specifying a value larger than the minimum acceptable value. The larger the value, the more freedom the algorithm has to optimize both balance and efficiency.

The `balance=` option works by adding restrictions to the design. The badness of each column (how far each column is from conforming to the balance restrictions) is evaluated and the results stored in a scalar `__bbad`. When you specify other restrictions, this is added to the `bad` value created by your restrictions macro. You can use your restrictions macro to change or differentially weight `__bbad` before the final addition of the components of design badness takes place (see page 688).

The `%MktEx` macro usually does a good job of producing nearly balanced designs, but if balance is critically important, and your designs are not balanced enough, you can sometimes achieve better balance by specifying `balance=`, but usually at the price of worse efficiency, sometimes much worse. By default, no additional restrictions are added. Another approach is to instead use the `%MktBal`

macro, which for main effects plans with no restrictions, produces designs that are guaranteed to have optimal balance.

The `balance=` option has changed in this release. It now uses the new `mintry=` option, and it usually makes better designs than it used to. When you specify `balance=`, you should also specify `mintry=` (perhaps something like `mintry=5 * n`, or `mintry=10 * n`). When `balance=` and `mintry=`*mt* are both specified, then the balance restrictions are ignored for the first `mt - 3 * n / 2` passes through the design. During this period, the badness function for the balance restrictions is set to 1 so that `%MktEx` knows that the design does not conform. After that, all restrictions are considered. The `balance=` option works best when its restrictions are imposed on a reasonably efficient design not an inefficient initial design.

examine= I | V

specifies the matrices that you want to examine. The option `examine=I` prints the information matrix, $\mathbf{X}'\mathbf{X}$; `examine=V` prints the variance matrix, $(\mathbf{X}'\mathbf{X})^{-1}$; and `examine=I V` prints both. By default, these matrices are not printed.

interact= *interaction-list*

specifies interactions that must be estimable. By default, no interactions are guaranteed to be estimable. Examples:

```
interact=x1*x2
interact=x1*x2 x3*x4*x5
interact=x1|x2|x3|x4|x5@2
```

The interaction syntax is like PROC GLM's and many of the other modeling procedures. It uses "`*`" for simple interactions (`x1*x2` is the interaction between `x1` and `x2`), "`|`" for main effects and interactions (`x1|x2|x3` is the same as `x1 x2 x1*x2 x3 x1*x3 x2*x3 x1*x2*x3`) and "`@`" to eliminate higher-order interactions (`x1|x2|x3@2` eliminates `x1*x2*x3` and is the same as `x1 x2 x1*x2 x3 x1*x3 x2*x3`). The specification "`@2`" allows only main effects and two-way interactions. Only "`@`" values of `2` or `3` are allowed. For the factor names, you must specify either the actual variable names (for example, `x1 x2 ...`) or you can just specify the number without the "`x`" (for example, `x1*x2` is equivalent to `1*2`). You can also specify `interact=@2` for all main effects and two-way interactions. For example, these two specifications are equivalent:

```
%mktex(2 ** 5, interact=@2, n=32)
%mktex(2 ** 5, interact=1|2|3|4|5@2, n=32)
```

mintry= *n*

specifies the minimum number of rows to process before giving up for each design. For example, to ensure that the macro passes through each row of the design at least five times, you can specify `mintry=5 * n`. You can specify a number or a DATA step expression involving n (rows) and m (columns). By default, the macro will always consider at least n rows. This option can be useful with certain restrictions, particularly with `balance=`. When `balance=` and `mintry=mt` are both specified, then the balance restrictions are ignored for the first `mt - 3 * n / 2` passes through the design. During this period, the badness function for the balance restrictions is set to 1 so that `%MktEx` knows that the design does not conform. After that, all restrictions are considered. The `balance=` option works best when its restrictions are imposed on a reasonably efficient design not an inefficient initial design.

The %MktEx macro sometimes prints:

```
WARNING: It may be impossible to meet all restrictions.
```

In previous releases, it did this when %MktEx passed through the entire design (n rows) without succeeding in minimizing the badness in any of the rows. Now, the message is printed after mintry=n rows are passed without any success. Sometimes, it is premature to expect any success during the first pass. When you know this, you can specify this option to prevent that warning from coming out.

options= *options-list*

specifies binary options. By default, none of these options are specified. Specify one or more of the following values after options=.

accept
allows the macro to output designs that violate restrictions imposed by restrictions=, balance=, or partial=, or have duplicates with options=nodups. Normally, the macro will not output such designs. With options=accept, a design becomes eligible for output when the macro can no longer improve on the restrictions or eliminate duplicates. Without options=accept, a design is only eligible when all restrictions are met and all duplicates are eliminated.

check
checks the efficiency of a given design, specified in init=, and disables the out=, outr=, and outall= options. If init= is not specified, options=check is ignored.

file
render to a generated file name, e.g.: OA(36,2^11,3^12).

int
add an intercept to the design, variable, x0.

justinit
specifies that the macro should stop processing as soon as it is done making the initial design, even if that design would not normally be the final design. Usually, this design will be an orthogonal array or some function of an orthogonal array (e.g. some three-level factors could be recoded into two-level factors), but there are no guarantees. Use this option when you want to output the initial tabled design, for example, if you want to see an orthogonal but unbalanced design that %MktEx sometimes uses as an initial design. The options=justinit specification implies optiter=0 and outr=, and options=justinit nofinal both stops processing and prevents the final design from being evaluated. Particularly when you specify options=nofinal, you must ensure that this design has a suitable efficiency.

`largedesign`
allows the macro to stop after `maxtime=` minutes have elapsed in the coordinate exchange algorithm. Typically, you would use this with `maxstages=1` and other options that make the algorithm run faster. By default, the macro checks time after it finishes with a design. With this option, the macro checks the time at the end of each row, after it has completed the first full pass through the design, and after any restrictions have been met, so the macro may stop before *D*-efficiency has converged. For really large problems and problems with restrictions, this option may make the macro run much faster but at a price of lower *D*-efficiency. For example, for large problems with restrictions, you might just want to try one run through the coordinate exchange algorithm with no candidate set search, orthogonal arrays, or mutations.

`lineage`
prints the lineage or "family tree" of the orthogonal array. For example, the lineage of the design $2^1 3^{25}$ in 54 runs is `54 ** 1 :  54 ** 1 > 3 ** 20 6 ** 1 9 ** 1 :  9 ** 1 > 3 ** 4 :  6 ** 1 > 2 ** 1 3 ** 1`. This states that the design starts as a single 54-level factor, then 54^1 is replaced by $3^{20}6^1 9^1$, 9^1 is replaced by 3^4, and finally 6^1 is replaced by $2^1 3^1$ to make the final design.

`nodups`
eliminates duplicate runs.

`nofinal`
skips calling PROC OPTEX to print the efficiency of the final experimental design.

`nohistory`
does not print the iteration history.

`nosort`
does not sort the design. One use of this option is with Hadamard matrices. Hadamard matrices are generated with a banded structure that is lost when the design is sorted. If you want to see the original Hadamard matrix, and not just a design constructed from the Hadamard matrix, specify `options=nosort`.

`render`
print the design compactly.

`refine`
specifies that with an `init=` design data set with at least one nonpositive entry, each successive design iteration tries to refine the best design from before. By default, the part of the design that is not fixed is randomly reinitialized each time. The default strategy is usually superior.

`resrep`
reports on the progress of the restrictions. You may want to specify this option with large problems with lots of restrictions or if you try to create a design and find that `%MktEx` is unable to make a design that conforms to the restrictions. By default, the iteration history is not printed for the stage where `%MktEx` is trying to make the design conform to the restrictions. Specify `options=resrep` when you want to see the progress in making the design conform.

`+-`
with render, print -1 as '-' and 1 as '+' in two-level factors. This option is typically used with `levels=i` for printing Hadamard matrices.

`3`
modifies `options=+-` to apply to three-level factors as well: -1 as '-', 0 as '0', and 1 as '+'.

`512`
adds some larger designs in 512 runs with mixes of 16, 8, 4, and 2-level factors to the catalog, which gives added flexibility in 512 runs at a cost of potentially *much* slower run time. This option replaces the default $4^{160}32^1$ parent with $16^{32}32^1$ and adds over 60,000 new designs to the catalog. Many of these designs are automatically available with FACTEX, so do not use this option unless you have first tried and failed to find the design without it.

partial= *n*

specifies a partial-profile design (Chrzan and Elrod, 1995). The default is an ordinary linear design. Specify for example `partial=4` if you only want 4 attributes to vary in each row of the design (except the first run, in which none vary). This option works by adding restrictions to the design (see `restrictions=`) and specifying `order=random` and `exchange=2`. The badness of each row (how far each row is from conforming to the partial-profile restrictions) is evaluated and the results stored in a scalar `__pbad`. When you specify other restrictions, this is added to the `bad` value created by your restrictions macro. You can use your restrictions macro to change or differentially weight `__pbad` before the final addition of the components of design badness takes place (see page 688). Because of the default `exchange=2` with partial-profile designs, the construction is slow, so you may want to specify `maxdesigns=1` or other options to make `%MktEx` run faster. For large problems, you may get faster but less good results by specifying `order=seqran`. Specifying `options=accept` or `balance=` with `partial=` is *not* a good idea. Here is the first part of a partial-profile design with twelve factors, each of which has three levels that vary and one level that means the attribute is not shown.

```
%mktex(4 ** 12, n=48, partial=4, seed=205, maxdesigns=1)
%mktlab(values=. 1 2 3, nfill=99)
options missing=' ';
proc print data=final(obs=10); run;
options missing='.';
```

Obs	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12
1												
2	1			2	1		2					
3				3	2		3					1
4			1					2	1			1
5		2	2		1							2
6				1			2				1	2
7			3		2				1			3
8				1	1		3		3			
9			2			2	2		2			
10		3				3	3				1	

reslist= *list*

specifies a list of constant matrices. Begin all names with an underscore to ensure that they do not conflict with any of names that `%MktEx` uses. If you specify more than one name, then names must be separated by commas. Example: `reslist=%str(_a, _b)`.

resmac= *macro-name*

specifies the name of a macro that creates the matrices named on the `reslist=` option. Begin all names including all intermediate matrix names with an underscore to ensure that they do not conflict with any of the names that `%MktEx` uses.

The `reslist=` and `resmac=` options can be used jointly for certain complicated restrictions to set up some constant matrices that the restrictions macro can use. Since the restrictions macro is called a lot, anything you can do only once helps speed up the algorithm.

Another way you can use these options is when you want to access a `%MktEx` matrix in your restrictions macro that you normally could not access. This would require knowledge of the internal workings of the `%MktEx` macro, so it is not a capability that you would usually need. Note that the `%MktEx` matrix `try` is now automatically available.

restrictions= *macro-name*

specifies the name of a macro that places restrictions on the design. By default, there are no restrictions. If you have restrictions on the design, what combinations can appear with what other combinations, then you must create a macro that creates a variable called `bad` that contains a numerical summary of how bad the row of the design is. When everything is fine, set `bad` to zero. Otherwise set `bad` to a larger value that is a function of the number of restriction violations. The `bad` variable must not be binary (0 - ok, 1 - bad) unless there is only one simple restriction. You must set `bad` so that the macro knows if the changes it is considering are moving the design in the right direction. See page 700 for examples of restrictions. The macro must consist of PROC IML statements and possibly some macro statements.

When you have restrictions, you should usually specify `options=resrep` so that you can get a report on the restriction violations in the iteration history. This can be a great help in debugging your restrictions macro. Also, be sure to check the log when you specify `restrictions=`. The macro cannot always ensure that your statements are syntax-error free and stop if they are not. There are many options that can impose restrictions, including `restrictions=`, `options=nodups`, `balance=`, `partial=`, and `init=`. If you specify more than one of these options, be sure that the combination makes sense, and be sure that it is possible to simultaneously satisfy all of the restrictions.

Your macro can look at several things in quantifying badness, and it must store its results in `bad`.

`i` - is a scalar that contains the number of the row currently being changed or evaluated. If you are writing restrictions that use the variable `i`, you almost certainly should specify `options=nosort`.

`try` - is a scalar similar to `i`, which contains the number of the row currently being changed. However, `try`, starts at zero and is incremented for each row, but it is only set back to zero when a new design starts, not when `%MktEx` reaches the last row. Use `i` as a matrix index and `try` to evaluate how far `%MktEx` is into the process of constructing the design. In previous releases, `try` was not automatically available.

`x` - is a row vector of factor levels for row `i` that always containing integer values beginning with 1 and continuing on to the number of levels for each factor. These values are always one-based, even if `levels=` is specified.

`x1` is the same as `x[1]`, `x2` is the same as `x[2]`, and so on.

`j1` - is a scalar that contains the number of the column currently being changed. In the steps where the badness macro is called once per row, `j1` = 1.

`j2` - is a scalar that contains the number of the other column currently being changed (along with `j1`) with `exchange=2` and larger `exchange=` values. This scalar will not exist with `exchange=1`. In the steps where the badness macro is called once per row, `j1` = `j21` = 1.

`j3` - is a scalar that contains the number of the third column currently being changed (along with `j1` and `j2`) with `exchange=3` and larger `exchange=` values. This scalar will not exist with `exchange=1` and `exchange=2`. If and only if the `exchange=`value is greater than 3, there will be a `j4` and so on. In the steps where the badness macro is called once per row, `j1` = `j2` = `j3` = 1.

`xmat` - is the entire `x` matrix. Note that the *ith* row of `xmat` may not be `x` since `x` will contain information on the exchanges being considered, whereas `xmat` contains the current design.

`bad` - results: 0 - fine, or the number of violations of restrictions. This value can be large or small and integers or real numbers. However, the values should always be nonnegative. When there are multiple sources of design badness, it is sometimes good to scale the different sources on different scales so that they do not trade off against each other. For example, for one source, you may multiply the number of violations by 1000, by 100 for another source, by 10 for another source, by 1 for another source, and even sometimes by 0.1 or 0.01 for another source. The final badness is the sum of `bad`, `__pbad` (when it exists), and `__bbad` (when it exists). The scalars `__pbad` and `__bbad` are explained next.

`__pbad` - is the badness from the `partial=` option. When `partial=` is not specified, this scalar does not exist. Your macro can weight this value, typically by multiplying it times a constant, to differentially weight the contributors to badness, e.g.: `__pbad = __pbad * 10`.

`__bbad` - is the badness from the `balance=` option. When `balance=` is not specified, this scalar does not exist. Your macro can weight this value, typically by multiplying it times a constant, to differentially weight the contributors to badness, e.g.: `__bbad = __bbad * 100`.

Do not use these names (other than `bad`) for intermediate values!

Other than that, you can create intermediate variables without worrying about conflicts with the names in the macro. The levels of the factors for one row of the experimental design are stored in a vector `x`, and the first level is always 1, the second always 2, and so on. All restrictions must be defined in terms of `x[j]` (or alternatively, `x1`, `x2`, ..., and perhaps the other matrices). For example, if there are 5 three-level factors and if it is bad if the level of a factor equals the level for the following factor, create a macro `restrict` as follows and specify `restrictions=restrict`.

```
%macro restrict;
   bad = (x1 = x2) +
         (x2 = x3) +
         (x3 = x4) +
         (x4 = x5);
   %mend;
```

Note that you specify just the macro name and no percents on the `restrictions=` option. Also note that IML does not have the full set of Boolean operators that the DATA step and other parts of SAS have. For example, these are *not* available: `OR AND NOT GT LT GE LE EQ NE`. Here are the operators you can use along with their meaning.

Specify	For	Do Not Specify
=	equals	EQ
∧ = or ¬ =	not equals	NE
<	less than	LT
<=	less than or equal to	LE
>	greater than	GT
>=	greater than or equal to	GE
&	and	AND
|	or	OR
∧ or ¬	not	NOT

Restrictions can substantially slow down the algorithm.

With restrictions, the `Current D-Efficiency` column of the iteration history table may contain values larger than the `Best D-Efficiency` column. This is because the design corresponding to the current *D*-efficiency may have restriction violations. Values are only reported in the best *D*-efficiency column after all of the restriction violations have been removed. You can specify `options=accept` with `restrictions=` when it is okay if the restrictions are not met.

See page 700 for more information on restrictions. See pages 286 and 403 for examples of restrictions. There are many examples of restrictions in the partial-profile examples starting on page 397.

seed= *n*

specifies the random number seed. By default, `seed=0`, and clock time is used to make the random number seed. By specifying a random number seed, results should be reproducible within a SAS release for a particular operating system and for a particular version of the macro. However, due to machine and macro differences, some results may not be exactly reproducible everywhere. For most orthogonal and balanced designs, the results should be reproducible. When computerized searches are done, it is likely that you will not get the same design across different computers, operating systems and different SAS and macro releases, although you would expect the efficiency differences to be slight.

Data Set Options

These next options specify the names of the input and output data sets.

cat= *SAS-data-set*

specifies the input design catalog. By default, the `%MktEx` macro automatically runs the `%MktOrth` macro to get this catalog. However, many designs can be made in multiple ways, so you can instead run `%MktOrth` yourself, and select the exact design you want and specify the resulting data set on the `cat=` option. Be sure to specify `options=dups lineage` when you run the `%MktOrth` macro. For example, the design 2^{71} in 72 runs can be made from either $2^{36}36^1$ or $2^{68}4^1$: This example selects the $2^{36}36^1$ method:

```
%mktorth(range=n=72, options=dups lineage)

proc print data=mktdeslev; var lineage; run;

data lev;
   set mktdeslev(where=(x2 = 71 and index(lineage, '2 ** 36 36 ** 1')));
   run;

%mktex(2 ** 71, n=72, cat=lev, out=b)
```

Running these next steps shows that you are in fact getting a design that is different from the default.

```
%mktex(2 ** 71, n=72, out=a)

proc compare data=a compare=b noprint note;
   run;
```

init= *SAS-data-set*
specifies the initial (input) experimental design. If all values in the initial design are positive, then a first step evaluates the design, the next step tries to improve it, and subsequent steps try to improve the best design found. However, if any values in the initial design are nonpositive (or missing) then a different approach is used. The initial design can have three types of values:

- positive integers are fixed and constant and will not change throughout the course of the iterations.
- zero and missing values are replaced by random values at the start of each new design search and can change throughout the course of the iterations.
- negative values are replaced by their absolute value at the start of each new design attempt and can change throughout the course of the iterations.

When absolute orthogonality and balance are required in a few factors, you can fix them in advance. This example illustrates how.

```
* Get first four factors;
%mktex(8 6 2 2, n=48)

* Flag the first four as fixed and set up to solve for the next six;
data init;
   set design;
   retain x5-x10 .;
   run;

* Get the last factors holding the first 4 fixed;
%mktex(8 6 2 2 4 ** 6, n=48, init=init, maxiter=100)

%mkteval;
```

Alternatively you can use HOLDOUTS= or FIXED= to fix just certain rows.

out= *SAS-data-set*
specifies the output experimental design. The default is `out=Design`. By default, this design is sorted unless you specify `options=nosort`. This is the output data set to look at in evaluating the design. See the `outr=` option for a randomized version of the same design, which is generally more suitable for actual use. Specify a null value for `out=` if you do not want this data set created. Often, you will want to specify a two-level name to create a permanent SAS data set so the design will be available later for analysis.

outall= *SAS-data-set*
specifies the output data set containing all designs found. By default, this data set is not created.

outr= *SAS-data-set*
specifies the randomized output experimental design. The default is `outr=Randomized`. Levels are randomly reassigned within factors, and the runs are sorted into a random order. Neither of these operations affects efficiency. When `restrictions=` or `partial=` is specified, only the random sort is performed. Specify a null value for `outr=` if you do not want a randomized design created. Often, you will want to specify a two-level name to create a permanent SAS data set so the design will be available later for analysis.

Iteration Options

These next options control some of the details of the iterations. Some of these options can take three arguments, one for each set of iterations. The macro can perform three sets of iterations. The `Algorithm Search` set of iterations looks for efficient designs using three different approaches. It then determines which approach appears to be working best and uses that approach exclusively in the second set of `Design Search` iterations. The third set or `Design Refinement` iterations tries to refine the best design found so far by using level exchanges combined with random mutations and simulated annealing.

The first set of iterations can have up to three parts. The first part uses either PROC PLAN or PROC FACTEX followed by PROC OPTEX, called through the `%MktDes` macro, to create and search a candidate set for an optimal initial design. The second part may use an orthogonal array or fractional-factorial design as an initial design. The next part consists of level exchanges starting with random initial designs.

In the first part, if the full-factorial design is manageable (arbitrarily defined as < 5185 runs), it is used as a candidate set, otherwise a fractional-factorial candidate set is used. The macro tries `optiter=` iterations to make an optimal design using the `%MktDes` macro and PROC OPTEX.

In the second part, the macro will try to generate and improve a standard orthogonal array or fractional-factorial design. Sometimes, this can lead immediately to an optimal design, for example with $2^{11}3^{12}$ and $n = 36$. In other cases, when only part of the desired design matches some standard design, only part of the design is initialized with the standard design and multiple iterations are run using the standard design as a partial initialization with the rest of the design randomly initialized.

In the third part, the macro uses the coordinate-exchange algorithm with random initial designs.

anneal= $n1 < n2 < n3 >>$
specifies the starting probability for simulated annealing in the coordinate-exchange algorithm. The default is `anneal=.05 .05 .01`. Specify a zero or null value for no annealing. You can specify more than one value if you would like to use a different value for the algorithm search, design search, and design refinement iterations. Specifying a value (greater than zero and less than one, for example 0.1) allows the design to get worse with decreasing probability as the number of iterations increases. This often helps the algorithm overcome local efficiency maxima. Allowing efficiency to decrease can help get past the bumps on the efficiency function.

Examples: `anneal=` or `anneal=0` specifies no annealing, `anneal=0.1` specifies an annealing probability of 0.1 during all three sets of iterations, `mutate=0 0.1 0.05` specifies no annealing during the initial iterations, an annealing probability of 0.1 during the search iterations, and an annealing probability of 0.05 during the refinement iterations.

anniter= $n1 < n2 < n3 >>$
specifies the first iteration to consider using annealing on the design. The default is `anniter=. . .`, which means that the macro chooses values to use. The default is the first iteration that uses a fully random initial design in each of the three sets of iterations. Hence by default, there is no random annealing in any part of the initial design when part of the initial design comes from an orthogonal design.

canditer= $n1 < n2 >$
specifies the number of coordinate-exchange iterations that will be used to try to improve a candidate-set based, OPTEX-generated initial design. The default is `canditer=1 1`. Note that `optiter=` controls the number of OPTEX iterations. Unless you are using annealing or mutation in the `canditer=` iterations (by default you are not) or unless you are using `options=nodups`, do not change theses values. The default value of `canditer=1 1`, along with the default `mutiter=` and `anniter=` values of missing, mean that the results of the OPTEX iterations are presented once in the algorithm iteration history, and if appropriate, once in the design search iteration history. Furthermore, by default, OPTEX generated designs are not improved with level exchanges except in the design refinement phase.

maxdesigns= n
specifies that the macro should stop after `maxdesigns=` designs have been created. This option may be useful for big, slow problems with restrictions. You could specify for example `maxdesigns=3` and `maxtime=0` and the macro would perform one candidate-set-based iteration, one orthogonal design initialization iteration, and one random initialization iteration and then stop. By default, this option is ignored and stopping is based on the other iteration options. For large designs with restrictions, a typical specification is `optiter=0, tabiter=0, maxdesigns=1, options=largedesign`.

maxiter= $n1 < n2 < n3 >>$
iter= $n1 < n2 < n3 >>$
specifies the maximum number of iterations or designs to generate. The default is `maxiter=21 25 10`. With larger values, the macro tends to find better designs at a cost of slower run times. You can specify more than one value if you would like to use a different value for the algorithm search, design search, and design refinement iterations. The second value is only used if the second set of iterations consists of coordinate-exchange iterations. Otherwise, the number of iterations for the second set is specified with the `tabiter=`, or `canditer=` and `optiter=` options. If you want more iterations, be sure

to set the `maxtime=` option as well, because iteration stops when the maximum number of iterations is reached or the maximum amount of time, whichever comes first. Examples: `maxiter=10` specifies 10 iterations for the initial, search, and refinement iterations, and `maxiter=10 10 5` specifies 10 initial iterations, followed by 10 search iterations, followed by 5 refinement iterations.

maxstages= *n*

specifies that the macro should stop after `maxstages=` algorithm stages have been completed. This option may be useful for big, slow problems with restrictions. You could specify `maxstages=1` and the macro will stop after the algorithm search stage, or `maxstages=2` and the macro will stop after the design search stage. The default is `maxstages=3`, which means the macro will stop after the design refinement stage.

maxtime= *n1 < n2 < n3 >>*

specifies the approximate maximum amount of time in minutes to run each phase. The default is `maxtime=10 20 5`. When an iteration completes (a design is completed), if more than the specified amount of time has elapsed, the macro quits iterating in that phase. Usually, run time will be no more than 10% or 20% larger than the specified values. However, for large problems, with restrictions, and with `exchange=` values other than 1, run time may be quite a bit larger than the specified value, since the macro only checks time after a design finishes. You can specify more than one value if you would like to use a different value for the algorithm search, design search, and design refinement iterations. By default, the macro spends up to 10 minutes on the algorithm search iterations, 20 minutes on the design search iterations, and 5 minutes in the refinement stage. Most problems run in much less time than this. Note that the second value is ignored for OPTEX iterations since OPTEX does not have any timing options. This option also affects, in the algorithm search iterations, when the macro switches between using an orthogonal initial design to using a random initial design. If the macro is not done using orthogonal initializations, and one half of the first time value has passed, it switches. Examples: `maxtime=60` specifies up to one hour for each phase. `maxtime=20 30 10` specifies 20 minutes for the first phase and 30 minutes for the second, and 10 for the third. The option `maxtime=0` provides a way to get a quick run, with no more than one iteration in each phase. However, even with `maxtime=0`, run time can be several minutes or more for large problems. See the `maxdesigns=` and `maxstages=` options for other ways to drastically cut run time for large problems.

If you specify really large time values (anything more than hours), you probably need to also specify `optiter=` since the default values depend on `maxtime=`.

mutate= *n1 < n2 < n3 >>*

specifies the probability at which each value in an initial design may mutate or be assigned a different random value before the coordinate-exchange iterations begin. The default is `mutate=.05 .05 .01`. Specify a zero or null value for no mutation. You can specify more than one value if you would like to use a different value for the algorithm search, design search, and design refinement iterations. Examples: `mutate=` or `mutate=0` specifies no random mutations. The `mutate=0.1` option specifies a mutation probability of 0.1 during all three sets of iterations. The `mutate=0 0.1 0.05` option specifies no mutations during the first iterations, a mutation probability of 0.1 during the search iterations, and a mutation probability of 0.05 during the refinement iterations.

mutiter= $n1 < n2 < n3 >>$
specifies the first iteration to consider mutating the design. The default is `mutiter=. . .`, which means that the macro chooses values to use. The default is the first iteration that uses a fully random initial design in each of the three sets of iterations. Hence by default, there are no random mutations of any part of the initial design when part of the initial design comes from an orthogonal design.

optiter= $n1 < n2 >$
specifies the number of iterations to use in the OPTEX candidate-set based searches in the algorithm and design search iterations. The default is `optiter=. .`, which means that the macro chooses values to use. When the first value is "." (missing), the macro will choose a value usually no smaller than 20 for larger problems and usually no larger than 200 for smaller problems. However, `maxtime=` values other than the defaults can make the macro choose values outside this range. When the second value is missing, the macro will choose a value based on how long the first OPTEX run took and the value of `maxtime=`, but no larger than 5000. When a missing value is specified for the first `optiter=` value, the default, the macro may choose to not perform any OPTEX iterations to save time if it thinks it can find a perfect design without them.

repeat= $n1\ n2\ n3$
specifies the maximum number of times to repeatedly work on a row to eliminate restriction violations. The default value of `repeat=25 . .`, specifies that a row should be worked on up to 25 times to eliminate violations. The second value is the place in the design refinement where this processing starts. This is based on a zero-based total number of rows processed so far. This is like a zero-based row index, but it never resets within a design. The third value is the place where this extra repeated processing stops. Let m be the `mintry=`m value, which by default is n, the number of rows. By default, when the second value is missing, the process starts after m rows have been processed (the second complete pass through the design). By default, the process stops after `m + 10 * n` rows have been processed where m is the second (specified or derived) `repeat=` value.

tabiter= $n1 < n2 >$
specifies the number of times to try to improve an orthogonal or fractional-factorial initial design. The default is `tabiter=10 200`, which means 10 iterations in the algorithm search and 200 iterations in the design search.

unbalanced= $n1 < n2 >$
specifies the proportion of the `tabiter=` iterations to consider using unbalanced factors in the initial design. The default is `unbalanced=.2 .1`. One way that unbalanced factors occur is through coding down. Coding down for example creates a three-level factor from a four-level factor: (1 2 3 4) ⇒ (1 2 3 3) or a two-level factor from a three-level factor: (1 2 3) ⇒ (1 2 2). For any particular problem, this strategy is probably either going to work really well or not well at all, without much variability in the results, so it is not tried very often by default. This option will try to create two-level through five-level factors from three-level through six-level factors. It will not attempt for example to code down a twenty-level factor into a nineteen-level factor (although the macro is often capable of in effect doing just that through level exchanges).

Miscellaneous Options

This section contains some miscellaneous options that some users may occasionally find useful.

big= $n <$ choose $>$

specifies the full-factorial-design size that is considered to be big. The default is `big=5185 choose`. The default value was chosen because 5185 is approximately 5000 and greater than $2^6 3^4 = 5184$, $2^{12} = 4096$, and $2 \times 3^7 = 4374$. When the full-factorial design is smaller than the `big=` value, the `%MktEx` macro searches a full-factorial candidate set. Otherwise, it searches a fractional-factorial candidate set. When `choose` is specified as well (the default), the macro is allowed to choose to use a fractional-factorial even if the full-factorial design is not too big if it appears that the final design can be created from the fractional-factorial design. This may be useful for example when you are requesting a fractional-factorial design with interactions. Using FACTEX to create the fractional-factorial design may be a better strategy than searching a full-factorial design with PROC OPTEX.

exchange= n

specifies the number of factors to consider at a time when exchanging levels. You can specify `exchange=2` to do pair-wise exchanges. Pair-wise exchanges are *much* slower, but may produce better designs. For this reason, you may want to specify `maxtime=0` or `maxdesigns=1` or other iteration options to make fewer designs and make the macro run faster. The `exchange=` option interacts with the `order=` option. The `order=seqran` option is faster with `exchange=2` than `order=sequential` or `order=random`. The default is `exchange=2` when `partial=` is specified. With `order=matrix`, the `exchange=` value is the number of matrix columns. Otherwise, the default is `exchange=1`.

With partial-profile designs and certain other highly restricted designs, it is important to do pair-wise exchanges. Consider for example, the following design row with `partial=4`

```
1 1 2 3 1 1 1 2 1 1 1 3
```

The `%MktEx` macro cannot consider changing a 1 to a 2 or 3 unless it can also consider changing one of the current 2's or 3's to 1 to maintain the partial-profile restriction of exactly four values not equal to 1. Specifying the `exchange=2` option gives `%MktEx` that flexibility.

fixed= *variable*

specifies an `init=` data set variable that indicates which runs are fixed (cannot be changed) and which ones may be changed. By default, no runs are fixed.

1 - (or any nonmissing) means this run may never change.
0 - means this run is used in the initial design, but it may be swapped out.
. - means this run should be randomly initialized, and it may be swapped out.

This option can be used to add holdout runs to a conjoint design, but see `holdouts=` for an easier way. To more generally fix parts of the design, see the `init=` option.

holdouts= *n*

adds holdout observations to the `init=` data set. This option augments an initial design. Specifying `holdouts=`*n* optimally adds *n* runs to the `init=` design. The option `holdouts=`*n* works by adding a `fixed=` variable and extra runs to the `init=` data set. Do not specify both `fixed=` and `holdouts=`. The number of rows in the `init=` design, plus the value specified in `holdouts=` must equal the `n=` value.

levels= *value*

specifies the method of assigning the final factor levels. This recoding occurs after the design is created, so all restrictions must be expressed in terms of one-based factors, regardless of what is specified in the `levels=` option.

Values:

`1` - default, one based, the levels are 1, 2, ...

`0` - zero based, the levels are 0, 1, ...

`c` - centered, possibly resulting in nonintegers 1 2 → -0.5 0.5, 1 2 3 → -1 0 1.

`i` - centered and scaled to integers. 1 2 → -1 1, 1 2 3 → -1 0 1.

You can also specify separate values for two- and three-level factors by preceding a value by "2" or "3". For example, `levels=2 i 3 0 c` means two-level factors are coded -1, 1 and three-level factors are coded 0, 1, 2. The remaining factors are centered. Note that the centering is based on centering the level values not on centering the (potentially unbalanced) factor. So for example the centered levels for a two-level factor in five runs (1 2 1 2 1) are (-0.5 0.5 -0.5 0.5 -0.5) not (-0.4 0.6 -0.4 0.6 -0.4). If you want the latter form of centering, use `proc standard m=0`. See the `%MktLab` macro for more general level setting.

You can also specify three other values:

`first` - means the first row of the design should consist entirely of the first level.

`last` - means the first row of the design should consist entirely of the last level, which is useful for Hadamard matrices.

`int` - adds an intercept column to the design.

order= `col=`*n* | `matrix=SAS-data-set`| `random` | `random=`*n* | `ranseq` | `sequential`

specifies the order in which the columns are worked on in the coordinate exchange algorithm. Valid values include:

`col=`*n* - process *n* random columns in each row
`matrix=`*SAS-data-set* - read the order from a data set
`random` - random order
`random=`*n* - random order with partial-profile exchanges
`ranseq` - sequential from a random first column
`seqran` - alias for `ranseq`
`sequential` - 1, 2, 3, ...
null - (the default) `random` when there are partial-profile restrictions, `ranseq` when there are other restrictions, and `sequential` otherwise.

For order=col=n, specify an integer for n, for example order=col=2. This option should only be used for huge problems where you do not care if you hit every column. Typically, this option will be used in conjunction with options=largedesign, maxdesigns=1, optiter=0, tabiter=0. You would use it when you have a large problem and you do not have enough time for one complete pass through the design. You just want to iterate for approximately the maxtime= amount of time then stop. You should not use order=col= with restrictions.

The options order=random=n is like order=random, but with an adaptation that is particularly useful for partial-profile choice designs. Use this option with exchange=2. Say you are making a partial-profile design with ten attributes and three alternatives. Then attribute 1 is made from x1, x11, and x21; attribute 2 is made from x2, x12, and x22; and so on. Specifying order=random=10 means that the columns, as shown by column index j1, are traversed in a random order. A second loop (with variable j2) traverses all of the factors in the current attribute. So for example when j1 is 13, then j2 = 3, 13, 23. This allows pair-wise exchanges within choice attributes.

The order= option interacts with the exchange= option. With a random order and exchange=2, the variable j1 loops over the columns of the design in a random order and for each j1, j2 loops over the columns greater than j1 in a random order. With a sequential order and exchange=2, the variable j1 loops over the columns in 1, 2, 3 order and for each j1, j2 loops over the columns greater than j1 in a j1+1, j1+2, j1+3 order. The order=ranseq option is a bit different. With exchange=2, the variable j1 loops over the columns in an order r, r+1, r+2, ..., m, 1, 2, ..., r-1 (for random r), and for each j1 there is a single random j2. Hence, order=ranseq is the fastest option since it does not consider all pairs, just one pair. The order=ranseq option provides the only situation where you might try exchange=3.

The order=matrix=SAS-data-set option allows you to specify exactly what columns are worked on in what order and in what groups. The SAS data set provides one row for every column grouping. Say you wanted to use this option to work on columns in pairs. (Note that you could just use exchange=2 to do this.) Then the data set would have two columns. The first variable would contain the number of a design column, and the second variable would contain the number of a second column that is to be exchanged with the first. The names of the variables are arbitrary. Here is an example data set for five factors.

```
%let m = 5;
data ex;
   do i = 1 to &m;
      do j = i + 1 to &m;
         output;
         end;
      end;
   run;

proc print noobs; run;
```

i	j
1	2
1	3
1	4
1	5
2	3
2	4
2	5
3	4
3	5
4	5

The specified `exchange=` value is ignored, and the actual `exchange=` value is set to two because the data set has two columns. The values must be integers between 1 and m, where m is the number of factors. The values may also be missing except in the first column. Missing values are replaced by a random column (potentially a different random column each time).

In a model with interactions, you can use this option to ensure that the terms that enter into interactions together get processed together. Here is an example.

```
data mat;
   input x1-x3;
   datalines;
1 1 1
2 3 .
2 4 .
3 4 .
2 3 4
5 5 .
6 7 .
8 . .
;

%mktex( 4 4 2 2 3 3 2 3, n=36, order=matrix=mat,
        interact=x2*x3 x2*x4 x3*x4 x6*x7, seed=472 )
```

The data set MAT contains eight rows, so there will be eight column groupings processed. The data set contains three columns, so up to three-way exchanges will be considered. The first row mentions column 1 three times. Any repeats of a column number are ignored, so the first group of columns simply consists of column 1. The second column consists of 2, 3, and ., so the second group consists of columns 2, 3, and some random column. The random column could be any of the columns including 2 and 3, so this will sometimes be a two-way and sometimes be a three-way exchange. This group was specified since `x2*x3` is one of the interaction terms. Similarly, other groups consist of the other two-way interaction terms and a random factor: 2 and 4, 3 and 4, and 6 and 7. In addition, to help with the 3 two-way interactions involving `x2`, `x3`, and `x4`, there is one three-way term. Each time, this will consider $4 \times 2 \times 2$ exchanges, the product of the three numbers of levels. In principle, there is no limit on the number of columns, but in practice, this number could easily get too big to be useful with more than a few exchanges at a time. The row `5 5 .` requests an exchange between column 5 and a

random factor. The row `8 . .` requests an exchange between column 8 and two random factors.

stopearly= *n*

specifies that the macro may stop early when it keeps finding the same maximum *D*-efficiency over and over again in different designs. The default is `stopearly=5`. By default, during the design search iterations and refinement iterations, the macro will stop early if 5 times, the macro finds a *D*-efficiency essentially equal to the maximum but not greater than the maximum. This may mean that the macro has found the optimal design, or it may mean that the macro keeps finding a very attractive local optimum. Either way, it is unlikely it will do any better. When the macro stops for this reason, the macro will print

```
NOTE: Stopping since it appears that no improvement is possible.
```

Specify either 0 or a very large value to turn off the stop-early checking.

tabsize= *n*

allows you some control on which tabled (OA, FACTEX or Hadamard) design is used for the partial initialization when an exact match to a tabled design is not found. Specify the number of runs in the tabled design. By default, the macro chooses an orthogonal design that bests matches the specified design. See the `cat=` option for more detailed control.

target= *n*

specifies the target efficiency criterion. The default is `target=100`. The macro stops when it finds an efficiency value greater than or equal to this number. If you know what the maximum efficiency criterion is, or you know how big is big enough, you can sometimes make the macro run faster by allowing it to stop when it reaches the specified efficiency. You can also use this option if you just want to see the initial design that `%MktEx` is using: `target=1, optiter=0`. By specifying `target=1`, the macro will stop after the initialization as long as the initial efficiency is ≥ 1.

Esoteric Options

This last set of options contains all of the other miscellaneous options. Most of the time, most users should not specify options from this list.

annealfun= *function*

specifies the function that controls how the simulated annealing probability changes with each pass through the design. The default is `annealfun=anneal * 0.85`. Note that the IML operator `#` performs ordinary (scalar) multiplication. Most users will never need this option.

detfuzz= *n*

specifies the value used to determine if determinants are changing. The default is `detfuzz=1e-8`. If `newdeter > olddeter * (1 + detfuzz)` then the new determinant is larger. Otherwise if `newdeter > olddeter * (1 - detfuzz)` then the new determinant is the same. Otherwise the new determinant is smaller. Most users will never need this option.

imlopts= *options*
specifies IML PROC statement options. For example, for very large problems, you can use this option to specify the IML `symsize=` or `worksize=` options: `imlopts=symsize=`n `worksize=`m, substituting numeric values for n and m. The defaults for these options are host dependent. Most users will never need this option.

ridge= n
specifies the value to add to the diagonal of $\mathbf{X}'\mathbf{X}$ to make it nonsingular. The default is `ridge=1e-7`. Usually, for normal problems, you will not need to change this value. If you want the macro to create designs with more parameters than runs, you must specify some other value, usually something like 0.01. By default, the macro will quit when there are more parameters than runs. Specifying a `ridge=` value other than the default (even if you just change the "e" in 1e-7 to "E") allows the macro to create a design with more parameters than runs. This option is sometimes needed for advanced design problems.

Advanced Restrictions

It is extremely important with restrictions to appropriately quantify the badness of the run. The `%MktEx` macro has to know when it considers an exchange if it is considering

- eliminating restriction violations making the design better,
- causing more restriction violations making the design worse,
- a change that neither increases nor decreases the number of violations.

Your restrictions macro must tell `%MktEx` when it is making progress in the right direction. If it does not, `%MktEx` will probably not find an acceptable design.

Complicated Restrictions

Consider designing a choice experiment with two alternatives each composed of 25 attributes, the first 22 of which will have restrictions on them. Attribute one in the choice design will be made from `x1` and `x23`, attribute two in the choice design will be made from `x2` and `x24`, ..., and attribute 22 in the choice design will be made from `x22` and `x44`. The remaining attributes will be made from `x45 - x50`. The restrictions are as follows: each choice attribute must contain two 1's between 5 and 9 times, each choice attribute must contain exactly one 1 between 5 and 9 times, and each choice attribute must contain two 2's between 5 and 9 times. Here is an example of how *NOT* to accomplish this.

```
%macro sumres;
   allone = 0; oneone = 0; alltwo = 0;
   do k = 1 to 22;
      if      (x[k] = 1 & x[k+22] = 1) then allone = allone + 1;
      else if (x[k] = 1 | x[k+22] = 1) then oneone = oneone + 1;
      else if (x[k] = 2 & x[k+22] = 2) then alltwo = alltwo + 1;
      end;
```

```
   * Bad example.  Need to quantify badness.;
   bad = (^((5 <= allone & allone <= 9) &
            (5 <= oneone & oneone <= 9) &
            (5 <= alltwo & alltwo <= 9)));
   %mend;
%mktex(3 ** 50, n=135, optiter=0, tabiter=0, maxdesigns=1,
      restrictions=sumres, seed=289, options=resrep)
```

The problem with the preceding code is there are complicated restrictions but badness is binary. If all the counts are in the right range, badness is 0, otherwise it is 1. You need write a macro that lets `%MktEx` know when it is going in the right direction or it will probably never find a suitable design. One thing that is correct about the preceding code is the compound Boolean range expressions like `(5 <= allone & allone <= 9)`. Abbreviated expressions like `(5 <= allone <= 9)` that work correctly in the DATA step work incorrectly and without warning in IML. Another thing that is correct is the way the `sumres` macro creates new variables, `k`, `allone`, `oneone`, and `alltwo`. Care was taken to avoid using names like `i` and `x` that conflict with the matrices that you are allowed to examine in quantifying badness. The full list of names that you must avoid are `i`, `try`, `x`, `x1`, `x2`, ..., through `x`n for n factors, `j1`, `j2`, `j3`, and `xmat`. Here is a slightly better but still bad example of the macro.

```
%macro sumres;
   allone = 0; oneone = 0; alltwo = 0;
   do k = 1 to 22;
      if      (x[k] = 1 & x[k+22] = 1) then allone = allone + 1;
      else if (x[k] = 1 | x[k+22] = 1) then oneone = oneone + 1;
      else if (x[k] = 2 & x[k+22] = 2) then alltwo = alltwo + 1;
      end;
   * Better, badness is quantified, and almost correctly too!;
   bad = (^((5 <= allone & allone <= 9) &
            (5 <= oneone & oneone <= 9) &
            (5 <= alltwo & alltwo <= 9))) #
         (abs(allone - 7) + abs(oneone - 7) + abs(alltwo - 7));
   %mend;
%mktex(3 ** 50, n=135, optiter=0, tabiter=0, maxdesigns=1,
      restrictions=sumres, seed=289, options=resrep)
```

This restrictions macro seems at first glance to do everything right−it quantifies badness. We need to examine this macro more closely. It counts in `allone`, `oneone`, and `alltwo` the number of times choice attributes are all one, have exactly one 1, or are all two. Everything is fine when the all one count is in the range 5 to 9 `(5 <= allone & allone <= 9)`, and the exactly one 1 count is in the range 5 to 9 `(5 <= oneone & oneone <= 9)`, and the all two count is in the range 5 to 9 `(5 <= alltwo & alltwo <= 9)`. It is bad when this is not true `(^((5 <= allone & allone <= 9) & (5 <= oneone & oneone <= 9) & (5 <= alltwo & alltwo <= 9)))`, the Boolean not operator "^" performs the logical negation. This Boolean expression is 1 for bad and 0 for OK. It is multiplied times a quantitative sum of how far these counts are outside the right range `(abs(allone - 7) + abs(oneone - 7) + abs(alltwo - 7))`. When the run meets all restrictions, this sum of absolute differences will be multiplied by zero. Otherwise badness gets larger as the counts get farther away from the middle of the 5 to 9 interval.

In the `%MktEx` macro, we specified `options=resrep` which produces a report in the iteration history on the process of meeting the restrictions. When you run `%MktEx` and it is having trouble making a design that conforms to restrictions, this report can be extremely helpful. Next, we will examine some of the output from running the preceding macros.

Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
1	Start	59.7632		Ran,Mut,Ann
1	1	60.0363		0 Violations
1	2	60.3715		0 Violations
1	3	60.9507		0 Violations
1	4	61.2319		5 Violations
1	5	61.6829		0 Violations
1	6	62.1529		0 Violations
1	7	62.4004		0 Violations
1	8	62.9747		3 Violations
.				
.				
.				
1	132	70.4482		6 Violations
1	133	70.3394		4 Violations
1	134	70.4054		0 Violations
1	135	70.4598		0 Violations

So far we have seen the results from the first pass through the design. With `options=resrep` the macro prints one line per row with the number of violations when it is done with the row. Notice that the macro is succeeding in eliminating violations in some but not all rows. This is the first thing you should look for. If it is not succeeding in any rows, you may have written a set of restrictions that is impossible to satisfy. Let's look next at some of the output from the second pass through the design.

1	1	70.5586		0 Violations
1	2	70.7439		0 Violations
1	3	70.7383		0 Violations
1	4	70.7429		5 Violations
1	4	70.6392		4 Violations
1	4	70.7081		4 Violations
1	4	70.7717		4 Violations
1	4	70.7717		4 Violations
1	4	70.7717		4 Violations
1	4	70.7717		4 Violations
1	4	70.7717		4 Violations
1	4	70.7202		4 Violations
1	4	70.7717		4 Violations
1	4	70.7717		4 Violations
1	4	70.7717		4 Violations

```
1      4          70.7264              4 Violations
1      4          70.7717              4 Violations
1      4          70.7717              4 Violations
1      4          70.7717              4 Violations
1      4          70.7717              4 Violations
1      4          70.7717              4 Violations
1      4          70.7274              4 Violations
1      4          70.7717              4 Violations
1      4          70.7515              4 Violations
1      4          70.7636              4 Violations
1      4          70.7717              4 Violations
1      4          70.7591              4 Violations
1      4          70.7717              4 Violations
1      5          70.7913              0 Violations
1      6          70.9467              0 Violations
1      7          71.0102              0 Violations
1      8          71.0660              0 Violations
```

In the second pass, in situations where the macro had some reasonable success in the first pass, `%MktEx` tries extra hard to impose restrictions. We see it trying over and over again without success to impose the restrictions in the fourth row. All it manages to do is lower the number of violations from 5 to 4. We also see it has no trouble removing all violations in the eighth row that were still there after the first pass. The macro produces volumes of output like this. For several iterations, it will devote extra attention to rows with some violations but in this case without complete success. When you see this pattern, some success but also some stubborn rows that the macro cannot fix, there may be something wrong with your restrictions macro. Are you *really* telling `%MktEx` when it is doing a better job? These preceding steps illustrate some of the things that can go wrong with restrictions macros. It is important to carefully evaluate the results—look at the design, look at the iteration history, specify `options=resrep`, and so on to ensure your restrictions are doing what you want. The problem in this case is in the quantification of badness, which is shown again next.

```
bad = (^((5 <= allone & allone <= 9) &
         (5 <= oneone & oneone <= 9) &
         (5 <= alltwo & alltwo <= 9))) #
       (abs(allone - 7) + abs(oneone - 7) + abs(alltwo - 7));
```

For one thing, notice that we have three nonindependent contributors to the badness function, the three counts. As a level gets changed, it could increase one count and decrease another. There is a larger problem too. Say that `allone` and `oneone` are in the right range but `alltwo` is not. Then the function fragments `abs(allone - 7)` and `abs(oneone - 7)` incorrectly contribute to the badness function. The fix is to clearly differentiate the three sources of badness *and* weight the pieces so that one part never trades off against the other. Here is an example.

```
%macro sumres;
   allone = 0; oneone = 0; alltwo = 0;
   do k = 1 to 22;
      if      (x[k] = 1 & x[k+22] = 1) then allone = allone + 1;
      else if (x[k] = 1 | x[k+22] = 1) then oneone = oneone + 1;
      else if (x[k] = 2 & x[k+22] = 2) then alltwo = alltwo + 1;
      end;
```

```
    bad = 100 # (^(5 <= allone & allone <= 9)) # abs(allone - 7) +
           10 # (^(5 <= oneone & oneone <= 9)) # abs(oneone - 7) +
                (^(5 <= alltwo & alltwo <= 9)) # abs(alltwo - 7);
    %mend;

%mktex(3 ** 50, n=135, optiter=0, tabiter=0, maxdesigns=1,
       restrictions=sumres, seed=289, options=resrep)
```

Now a component of the badness only contributes to the function when it is really part of the problem. We gave the first part weight 100 and the second part weight 10. Now the macro will never change `oneone` or `alltwo` if that causes a problem for `allone`, and it will never change `alltwo` if that causes a problem for `oneone`. Previously the macro was getting stuck in some rows because it could never figure out how to fix one component of badness without making another component worse. *For some problems, figuring out how to differentially weight the components of badness so that they never trade off against each other is the key to writing a successful restrictions macro.* Often, it does not matter which component gets the most weight, what is important is that each component gets a *different* weight so that `%MktEx` does not get caught cycling back and forth making A better and B worse then making B better and A worse. Here is some of the output from the first pass through the design.

The SAS System

Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
1	Start	59.7632		Ran,Mut,Ann
1	1	60.1415		0 Violations
1	2	60.5303		0 Violations
1	3	61.0148		0 Violations
1	4	61.4507		0 Violations
1	5	61.7717		0 Violations
1	6	62.2353		0 Violations
1	7	62.5967		0 Violations
1	8	63.1628		3 Violations
.				
.				
.				
1	126	72.3566		4 Violations
1	127	72.2597		0 Violations
1	128	72.3067		0 Violations
1	129	72.3092		0 Violations
1	130	72.0980		0 Violations
1	131	71.8163		0 Violations
1	132	71.3795		0 Violations
1	133	71.4446		0 Violations
1	134	71.2805		0 Violations
1	135	71.3253		0 Violations

We can see that in the first pass, the macro is imposing all restrictions for most but not all of the rows. Here is some of the output from the second pass.

```
1      1          71.3968              0 Violations
1      2          71.5017              0 Violations
1      3          71.7295              0 Violations
1      4          71.7839              0 Violations
1      5          71.8671              0 Violations
1      6          71.9544              0 Violations
1      7          72.0444              0 Violations
1      8          72.0472              0 Violations
.
.
.
1    126          77.1597              0 Violations
1    127          77.1604              0 Violations
1    128          77.1323              0 Violations
1    129          77.1584              0 Violations
1    130          77.0708              0 Violations
1    131          77.1013              0 Violations
1    132          77.1721              0 Violations
1    133          77.1651              0 Violations
1    134          77.1651              0 Violations
1    135          77.2061              0 Violations
```

In the second pass, `%MktEx` has imposed all the restrictions in rows 8 and 126, the rows that still had violations after the first pass (and all of the other not shown rows too). The third pass ends like this.

```
1    126          78.7813              0 Violations
1    127   1      78.7813      78.7813 Conforms
1    127  18      78.7899      78.7899
1    127  19      78.7923      78.7923
1    127  32      78.7933      78.7933
1    127  40      78.7971      78.7971
1    127  44      78.8042      78.8042
1    127  47      78.8250      78.8250
1    127  50      78.8259      78.8259
1    127   1      78.8296      78.8296
1    127   5      78.8296      78.8296
1    127   8      78.8449      78.8449
1    127  10      78.8456      78.8456
1    128  48      78.8585      78.8585
1    128  49      78.8591      78.8591
1    128   7      78.8591      78.8591
```

The `%MktEx` macro completes a full pass through row 126, the place of the last violation, without

finding any new violations so the macro states in row 127 that the design conforms to the restrictions and the iteration history proceeds in the normal fashion from then on (not shown). Here is the final efficiency.

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	85.0645	72.2858	95.6858	0.8650

This next code creates the choice design. Notice the slightly unusual arrangement of the KEY data set due to the fact that the first 22 attributes get made from the first 44 factors of the linear design.

```
%mktkey(x1-x50)

data key;
   input (x1-x25) ($);
   datalines;
x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13
x14 x15 x16 x17 x18 x19 x20 x21 x22                  x45 x46 x47
x23 x24 x25 x26 x27 x28 x29 x30 x31 x32 x33 x34
x35 x36 x37 x38 x39 x40 x41 x42 x43 x44              x48 x49 x50
;
%mktroll(design=design, key=key, out=chdes)

proc print; by set; id set; where set le 2 or set ge 134; run;
```

Here are a few of the choice sets.

```
   _
   A
  S l                   x x x x x x x x x x x x x x x x x
  e t x x x x x x x x x 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2
  t _ 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5

  1 1 1 1 1 1 1 2 1 3 1 1 2 3 2 3 2 3 2 2 2 1 2 2 1 1 1
    2 2 2 1 1 3 2 1 3 3 1 3 2 2 3 2 1 2 3 3 1 1 2 3 2 3

  2 1 1 1 1 1 1 3 1 1 2 2 3 2 2 2 1 2 3 1 1 3 1 2 1 1 2
    2 3 3 1 1 1 1 3 3 2 2 2 1 2 2 2 3 1 1 3 3 1 2 1 2 2

134 1 3 3 2 3 3 2 1 1 1 1 1 1 2 2 1 3 2 2 1 3 3 1 2 1 2
    2 1 1 2 2 1 2 1 1 1 1 1 1 3 2 2 3 1 1 2 1 1 3 3 1 3 3

135 1 3 3 3 1 3 1 1 1 1 2 2 3 1 2 3 3 1 3 2 1 2 1 2 3 1
    2 2 1 1 1 1 1 1 1 3 1 2 2 1 3 2 1 3 3 1 2 1 2 1 2 2 2
```

Where the Restrictions Macro Gets Called

There is one more aspect to restrictions that must be understood for the most sophisticated usages of restrictions. The macro that imposes the restrictions is defined and called in four distinct places in the `%MktEx` macro. First, the restrictions macro is called in a separate, preliminary IML step, just to catch some syntax errors that you might have made. Next, it is called in between calling PROC PLAN or PROC FACTEX and calling PROC OPTEX. Here, the restrictions macro is used to impose restrictions on the candidate set. Next, it is used in the obvious way during design creation and the coordinate-exchange algorithm. Finally, when `options=accept` is specified, which means that restriction violations are acceptable, the macro is called after all of the iterations have completed to report on restriction violations in the final design. For some advanced restrictions, we may not want exactly the same code running in all four places. When the restrictions are purely written in terms of restrictions on `x`, which is the *ith* row of the design matrix, there is no problem. The same macro will work fine for all uses. However, when `xmat` (the full `x` matrix) or `i` or `j1` (the row or column number) are used, the same code typically cannot be used for all applications, although sometimes it does not matter. Next are some notes on each of the four phases.

Syntax Check. In this phase, the macro is defined and called just to check for syntax errors. When there are errors, having this separate step, allows the macro to end more gracefully and provide better information about the nature of the error than it would otherwise. Your restrictions macro can recognize when it is in this phase because the macro variable `&main` is set to 0 and the macro variable `&pass` is set to null. The pass variable is null before the iterations begin, 1 for the algorithm search phase, 2 for the design search phase, 3 for the design refinement stage, and 4 after the iterations end. You can conditionally execute code in this step or not using the following macro statements.

```
%if      &main eq 0 and &pass eq  %then %do;  /* execute in syntax check     */
%if not (&main eq 0 and &pass eq) %then %do;  /* not execute in syntax check */
```

You will usually not need to worry about this step. It just calls the macro once and ignores the results to check for syntax errors. For this step, `xmat` is a matrix of ones, and `x` is a vector of ones (since the design does not exist yet) and `j1 = j2 = j3 = i = 1`. If you have complicated restrictions involving the row or column exchange indices (`i`, `j1`, `j2`, `j3`) you may need to worry about this step. You may need to either not execute your restrictions in this step or *conditionally* execute some assignment statements (just for this step) that set up `j1`, `j2`, and `j3` more appropriately. Sometimes you can set things up appropriately by using the `resmac=` option. Be aware however, that this step checks (`i`, `try`, `j1`, `j2`, `j3`, `x`, and `xmat` after your macro is called to ensure that you are not changing them because this is usually a sign of an error. If you get the following warning

```
WARNING: Restrictions macro is changing i, try, j1, j2, j3, x, or xmat.
         This might be a serious problem.  Check your macro.
```

make sure you are not incorrectly changing one of the matrices that you should not be changing. If this step detects a syntax error, it will try to tell you where it is and what the problem is. If you have syntax errors in your restrictions macro and you cannot figure out what they are, sometimes the best thing to do is directly submit the statements in your restrictions macro to so you can see the syntax errors. First submit the following statements.

```
%let n = 27; /* substitute number of runs    */
%let m = 10; /* substitute number of factors */
proc iml;
   xmat = j(&n, &m, 1);
   i = 1; j1 = 1; j2 = 1; j3 = 1; bad = 0; x = xmat[i,];
```

Candidate Check. In this phase, the macro is used to impose restrictions on the candidate set created by PROC PLAN or PROC FACTEX before it is searched by PROC OPTEX. The macro is called once for each row with the column index, `j1` set to 1. For some problems, such as most partial-profile problems, the restrictions are so severe that virtually none of the candidates will conform. Also, restrictions that are based on row number and column number do not make sense in the context of a candidate design. Your restrictions macro can recognize when it is in this phase because the macro variable `&main` is set to 0 and the macro variable `&pass` is set to 1 or 2. You can conditionally execute code in this step or not using the following macro statements.

```
%if      &main eq 0 and &pass ge 1 and &pass le 2
         %then %do;                              /* execute on candidates     */
%if not (&main eq 0 and &pass ge 1 and &pass le 2)
         %then %do;                              /* not execute on candidates */
```

For simple restrictions, not involving the column exchange indices (`j1`, `j2`, `j3`), you probably do not need to worry about this step. If you use `j1`, `j2`, or `j3`, you will need to either not execute your restrictions in this step or conditionally execute some assignment statements that set up `j1`, `j2`, and `j3` appropriately. Ordinarily for this step, `xmat` contains the candidate design, `x` contains the *ith* row, `j1 = 0; j2 = 0; j3 = 0; try = 1; i` is set to the candidate row number.

Main Coordinate-Exchange Algorithm. In this phase, the macro is used to impose restrictions on the design as it is being built in the coordinate-exchange algorithm. Your restrictions macro can recognize when it is in this phase because the macro variable `&main` is set to 1 and the macro variable `&pass` is set to 1, 2, or 3. You can conditionally execute code in this step or not using the following macro statements.

```
%if      &main eq 1 and &pass ge 1 and &pass le 3
         %then %do;                    /* execute on coordinate exchange     */
%if not (&main eq 1 and &pass ge 1 and &pass le 3)
         %then %do;                    /* not execute on coordinate exchange */
```

For this step, `xmat` contains the candidate design, `x` contains the *ith* row, `j1`, `j2`, and `j3` typically contain the column indices, `i` is the row number, and `try` is the zero-based cumulative row number. With `exchange=1`, `j1` exists, with `exchange=2`, `j1` and `j2` exist, and so on. Sometimes in this phase, the restrictions macro is called once per row with the `j*` indices all set to 1. If you use the `j*` indices in your restrictions, you may need to allow for this. For example, if you are checking the current `j1` column for balance, and you used an `init=` data set with column one fixed and unbalanced, you will not want to perform the check when `j1 = 1`. Note that for some designs that are partially initialized with an orthogonal array and for some uses of `init=`, not all columns or cells in the design are evaluated.

Restrictions Violations Check. In this phase, the macro is used to check the design when there are restrictions and `options=accept`. The macro is called once for each row of the design. Your restrictions macro can recognize when it is in this phase because the macro variable `&main` is set to 1 and the macro variable `&pass` is greater than 3. You can conditionally execute code in this step or not using the following macro statements.

```
%if      &main eq 1 and &pass gt 3  %then %do; /* execute on final check     */
%if not (&main eq 1 and &pass gt 3) %then %do; /* not execute on final check */
```

For this step, `xmat` contains the candidate design, `x` contains the *ith* row, `j1 = 1; j2 = 1; j3 = 1; try = 1;` and `i` is the row number.

Here is an example of a partial-profile macro that does what the `partial=4` option does.

```
%macro partprof;
   nvary = sum(x ^= 1);
   %if &main %then %do;
      if i = 1 then bad = nvary;
      else          bad = abs(nvary - 4);
      %end;
   %else %do;
      bad = ^ (nvary = 0 | nvary = 4);
      %end;
   %mend;
```

In the main algorithm, when imposing restrictions on the design, we restrict the first run to be constant and all other runs to have four attributes varying. For the candidate-set restrictions, when MAIN is zero, any observation with zero or four varying factors is acceptable. For the candidate-set restrictions, there is no reason to count the number of violations. A candidate run is either acceptable or not. We do not worry about the syntax error or final check steps; both versions will work fine in either.

%MktKey Macro

The %MktKey macro creates expanded lists of variable names.

```
%mktkey(x1-x15)
```

The %MktKey macro produced the following line.

```
x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15
```

You can cut and paste this list to make it easier to construct the key= data set for the %MktRoll macro.

```
data key;
   input (x1-x5) ($);
   datalines;
 x1  x2  x3  x4  x5
 x6  x7  x8  x9 x10
x11 x12 x13 x14 x15
  .   .   .   .   .
;
```

The %MktKey macro has an alternative syntax as well. You can specify the number of rows followed by a number of columns. The output is a data set called KEY. Here is an example.

```
%mktkey(5 10)
```

Here is the output data set KEY with 5 rows and 10 columns and $5 \times 1 = 50$ variable names, x1-x50.

x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
x11	x12	x13	x14	x15	x16	x17	x18	x19	x20
x21	x22	x23	x24	x25	x26	x27	x28	x29	x30
x31	x32	x33	x34	x35	x36	x37	x38	x39	x40
x41	x42	x43	x44	x45	x46	x47	x48	x49	x50

Alternatively, you can specify the number of rows and number of columns followed by a t or T and get the transpose of this data set. The output data set is again called KEY. Here is an example.

```
%mktkey(5 10 t)
```

Here is the output data set KEY with 5 rows and 10 columns and $5 \times 1 = 50$ variable names, x1-x50, but this time the names progress down the columns instead of across the rows.

x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
x1	x6	x11	x16	x21	x26	x31	x36	x41	x46
x2	x7	x12	x17	x22	x27	x32	x37	x42	x47
x3	x8	x13	x18	x23	x28	x33	x38	x43	x48
x4	x9	x14	x19	x24	x29	x34	x39	x44	x49
x5	x10	x15	x20	x25	x30	x35	x40	x45	x50

%MktKey Macro Options

The only argument to the `%MktKey` macro is a variable list.

list

specifies a variable list. Note that the variable list is a positional parameter and it is not specified after a name and an equal sign.

Alternatively, the list contains the number of rows followed by the number of columns, optionally followed by a `t` or `T` (for transpose). Without the `t` the names go `x1`, `x2`, `x3`, ..., across each row. With the `t` the names go `x1`, `x2`, `x3`, ..., down each column.

%MktLab Macro

The macro %MktLab is used to process an experimental design, usually created by the %MktEx macro, and assign the final variable names and levels. There are numerous examples of its usage from pages 178 through 401. For example, say you used the %MktEx macro to create a design with 11 two-level factors (with default levels of 1 and 2).

```
%mktex(n=12, options=nosort)

proc print noobs; run;
```

x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11
1	1	1	1	1	1	1	1	1	1	1
1	2	2	2	1	1	2	1	1	2	2
2	2	1	1	2	1	2	2	1	2	1
1	1	1	2	1	2	2	2	2	2	1
1	2	2	2	2	1	1	2	2	1	1
1	2	1	1	2	2	2	1	2	1	2
2	2	1	2	1	2	1	2	1	1	2
2	1	1	2	2	1	1	1	2	2	2
2	1	2	2	2	2	2	1	1	1	1
1	1	2	1	2	2	1	2	1	2	2
2	2	2	1	1	2	1	1	2	2	1
2	1	2	1	1	1	2	2	2	1	2

Either the %MktEx macro or the %MktLab macro can be used to assign levels of -1 and 1 and add an intercept. Here is how you can do it directly with the %MktEx macro, using levels=i int. The value of i specifies centered integer levels, and int adds the intercept.

```
%mktex(n=12, options=nosort, levels=i int)
```

However, if you want to change the factor names, and for more complicated relabeling of the levels, you need to use the %MktLab macro.

```
%mktex(n=12, options=nosort)
%mktlab(data=design, values=1 -1, int=Had0, prefix=Had)
```

The %MktLab macro assigns levels of -1 and 1, adds an intercept named Had0, and changes the variable name prefixes from x to Had. This creates a Hadamard matrix (although, of course, the Hadamard matrix can have any set of variable names).

```
%mktlab(data=design, values=1 -1, int=Had0, prefix=Had)

proc print noobs; run;
```

Here is the resulting Hadamard matrix:

Had0	Had1	Had2	Had3	Had4	Had5	Had6	Had7	Had8	Had9	Had10	Had11
1	1	1	1	1	1	1	1	1	1	1	1
1	1	-1	-1	-1	1	1	-1	1	1	-1	-1
1	-1	-1	1	1	-1	1	-1	-1	1	-1	1
1	1	1	1	-1	1	-1	-1	-1	-1	-1	1
1	1	-1	-1	-1	-1	1	1	-1	-1	1	1
1	1	-1	1	1	-1	-1	-1	1	-1	1	-1
1	-1	-1	1	-1	1	-1	1	-1	1	1	-1
1	-1	1	1	-1	-1	1	1	1	-1	-1	-1
1	-1	1	-1	-1	-1	-1	-1	1	1	1	1
1	1	1	-1	1	-1	-1	1	-1	1	-1	-1
1	-1	-1	-1	1	1	-1	1	1	-1	-1	1
1	-1	1	-1	1	1	1	-1	-1	-1	1	-1

Here is an alternative way of doing the same thing using a `key=` data set.

```
data key;
  array Had[11];
  input Had1 @@;
  do i = 2 to 11; Had[i] = Had1; end;
  drop i;
  datalines;
1 -1
;
proc print data=key; run;
```

Here is the `key=` data set.

Obs	Had1	Had2	Had3	Had4	Had5	Had6	Had7	Had8	Had9	Had10	Had11
1	1	1	1	1	1	1	1	1	1	1	1
2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

```
%mktlab(data=design, key=key, int=Had0)
```

The Hadamard matrix from this step (not shown) is exactly the same as the one just shown previously.

The `key=` data set contains all of the variables that you want in the design and all of their levels. This information will be applied to the design, by default the one stored in a data set called RANDOMIZED, which is the default `outr=` data set name from the `%MktEx` macro. The results are stored in a new data set, FINAL, with the desired factor names and levels.

Consider the consumer food product example from page 111. Here is one possible design.

```
data randomized;
  input x1-x8 @@;
  datalines;
4 2 1 1 1 2 2 2 2 1 1 2 1 3 1 3 3 4 2 2 1 3 2 3 4 3 2 1 3 2 2 3 4 1 2 1
1 1 1 1 2 4 1 2 1 2 1 1 1 2 1 2 3 3 2 1 2 2 2 2 2 2 2 3 1 4 2 1 1 2 2 2
3 2 2 1 3 1 2 1 1 4 1 2 2 3 1 2 1 3 2 2 1 3 1 1 3 2 1 2 2 1 2 3 3 4 1 1
3 1 1 3 4 1 2 2 2 1 2 1 2 3 2 1 2 3 2 2 2 1 2 1 3 3 1 3 4 2 2 2 1 3 1 2
2 4 2 2 3 1 1 2 3 1 2 2 3 2 1 2 3 3 1 1 2 3 1 1 4 4 2 1 2 2 1 3 1 1 1 1
3 2 1 2 4 3 1 2 3 3 2 2 1 2 2 1 2 1 1 3 1 3 1 1 1 1 2 3
;
```

Designs created by the %MktEx macro always have factor names x1, x2, ..., and so on, and the levels are consecutive integers beginning with 1 (1, 2 for two-level factors; 1, 2, 3 for three-level factors; and so on). The %MktLab macro provides you with a convenient way to change the names and levels to more meaningful values. The data set KEY contains the variable names and levels that you ultimately want.

```
data key;
  missing N;
  input Client ClientLineExtension ClientMicro $ ShelfTalker $
        Regional Private PrivateMicro $ NationalLabel;
  format _numeric_ dollar5.2;
  datalines;
1.29 1.39 micro Yes 1.99 1.49 micro 1.99
1.69 1.89 stove No  2.49 2.29 stove 2.39
2.09 2.39 .     .   N    N   .     N
N    N    .     .   .    .   .     .
;
%mktlab(key=key)

proc sort; by shelftalker; run;

proc print; by shelftalker; run;
```

The variable Client with 4 levels will be made from x1, ClientLineExtension with 4 levels will be made from x2, ClientMicro with 2 levels will be made from x3. The N (for not available) is treated as a special missing value. The KEY data set has four rows because the maximum number of levels is four. Factors with fewer than four levels are filled in with ordinary missing values. The %MktLab macro takes the default data=randomized data set from %MktEx and uses the rules in the key=key data set, to create the information in the out=final data set, which is shown next, sorted by the shelf talker variable.

Here is some of the design:

```
------------------------------ ShelfTalker=No ------------------------------

                 Client
                  Line       Client                            Private   National
Obs   Client    Extension    Micro     Regional    Private      Micro     Label

  1   $1.69       $1.39      micro      $1.99         N         micro       N
  2   $2.09         N        stove      $1.99         N         stove       N
  3   $1.69         N        micro      $1.99       $2.29       micro     $1.99
  .
  .
  .

------------------------------ ShelfTalker=Yes -----------------------------

                 Client
                  Line       Client                            Private   National
Obs   Client    Extension    Micro     Regional    Private      Micro     Label

 14     N         $1.89      micro      $1.99       $2.29       stove     $2.39
 15     N         $2.39      stove        N         $2.29       stove       N
 16     N         $1.39      stove      $1.99       $1.49       micro     $1.99
  .
  .
  .
```

This macro creates the `out=` data set by repeatedly reading and rereading the `key=` data set, one datum at a time, using the information in the `data=` data set to determine which levels to read from the `key=` data set. In this example, for the first observation, `x1`=4 so the fourth value of the first `key=` variable is read, then `x2`=2 so the second value of the second `key=` variable is read, then `x3`=1 so the first value of the third `key=` variable is read, ..., then `x8`=2 so the second value of the eighth `key=` variable is read, then the first observation is output. This continues for all observations. This is why the `data=` data set must have been created with the default `levels=` specification, and must have integer values beginning with 1.

This example creates the L_{36}, renames the two-level factors `two1-two11` and assigns them values -1, 1, and renames the three-level factors `thr1-thr12` and assigns them values -1, 0, 1.

```
%mktex(n=36, seed=420)

data key;
  array x[23] two1-two11 thr1-thr12;
  input two1 thr1;
  do i =  2 to 11; x[i] = two1; end;
  do i = 13 to 23; x[i] = thr1; end;
  drop i;
  datalines;
-1 -1
 1  0
 .  1
;
%mktlab(key=key)

proc print data=key noobs; var two:; run;
proc print data=key noobs; var thr:; run;

proc print data=final(obs=5) noobs; var two:; run;
proc print data=final(obs=5) noobs; var thr:; run;
```

Here is the KEY data set.

two1	two2	two3	two4	two5	two6	two7	two8	two9	two10	two11
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	1	1	1	1	1	1	1	1	1	1
.	.	.	.	.	.	.	.	.	.	.

thr1	thr2	thr3	thr4	thr5	thr6	thr7	thr8	thr9	thr10	thr11	thr12
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1

Here are the first five rows of the design.

two1	two2	two3	two4	two5	two6	two7	two8	two9	two10	two11
-1	-1	1	1	1	1	1	1	1	1	-1
1	1	1	1	-1	1	1	-1	-1	1	1
-1	-1	-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	1	1	1	-1	-1	-1	1
-1	1	-1	-1	-1	-1	1	1	1	1	1

thr1	thr2	thr3	thr4	thr5	thr6	thr7	thr8	thr9	thr10	thr11	thr12
0	0	0	1	1	1	-1	1	1	1	-1	-1
1	-1	-1	1	0	0	-1	1	1	0	1	0
0	-1	0	0	-1	-1	-1	0	-1	0	-1	0
0	1	0	1	1	0	1	-1	-1	-1	1	0
1	-1	1	1	1	1	0	-1	-1	0	-1	1

This next step creates a design and blocks it. This example shows that it is okay if not all of the variables in the input design are used. The variables `Block`, `Run`, and `x4` are just copied from the input to the output.

```
%mktex(n=18, seed=396)

%mktblock(nblocks=2, factors=x1-x4, seed=292)

data key;
   input Brand $ Price Size;
   format price dollar5.2;
   datalines;
Acme 1.49   6
Apex 1.79   8
.    1.99  12
;
%mktlab(data=blocked, key=key)
```

Here are the results:

Block	Run	Brand	Price	Size	x4
1	1	Acme	$1.49	12	2
	2	Acme	$1.79	6	3
	3	Acme	$1.79	8	2
	4	Acme	$1.99	8	1
	5	Acme	$1.99	12	1
	6	Apex	$1.49	8	3
	7	Apex	$1.49	12	3
	8	Apex	$1.79	6	1
	9	Apex	$1.99	6	2
2	1	Acme	$1.49	6	1
	2	Acme	$1.49	6	2
	3	Acme	$1.79	8	3
	4	Acme	$1.99	12	3
	5	Apex	$1.49	8	1
	6	Apex	$1.79	12	1
	7	Apex	$1.79	12	2
	8	Apex	$1.99	6	3
	9	Apex	$1.99	8	2

This next example illustrates using the `labels=` option. This option is more typically used with `values=` input, rather than when you construct the `key=` data set yourself, but it can be used either way. This example is from a vacation choice example.

```
%mktex(3 ** 15, n=36, seed=17, maxtime=0)

%mktblock(data=randomized, nblocks=2, factors=x1-x15, seed=448)

%macro lab;
   label X1  = 'Hawaii, Accommodations'
         X2  = 'Alaska, Accommodations'
         X3  = 'Mexico, Accommodations'
         X4  = 'California, Accommodations'
         X5  = 'Maine, Accommodations'
         X6  = 'Hawaii, Scenery'
         X7  = 'Alaska, Scenery'
         X8  = 'Mexico, Scenery'
         X9  = 'California, Scenery'
         X10 = 'Maine, Scenery'
         X11 = 'Hawaii, Price'
         X12 = 'Alaska, Price'
         X13 = 'Mexico, Price'
         X14 = 'California, Price'
         X15 = 'Maine, Price';
   format x11-x15 dollar5.;
%mend;

data key;
   length x1-x5 $ 16 x6-x10 $ 8 x11-x15 8;
   input x1 & $ x6 $ x11;
   x2  = x1;    x3 = x1;    x4 = x1;    x5 = x1;
   x7  = x6;    x8 = x6;    x9 = x6;   x10 = x6;
   x12 = x11;  x13 = x11;  x14 = x11;  x15 = x11;
   datalines;
Cabin            Mountains   999
Bed & Breakfast  Lake       1249
Hotel            Beach      1499
;

%mktlab(data=blocked, key=key, labels=lab)

proc contents p; ods select position; run;
```

Here is the variable name, label, and format information.

The CONTENTS Procedure

Variables in Creation Order

#	Variable	Type	Len	Format	Label
1	x1	Char	16		Hawaii, Accommodations
2	x2	Char	16		Alaska, Accommodations
3	x3	Char	16		Mexico, Accommodations
4	x4	Char	16		California, Accommodations
5	x5	Char	16		Maine, Accommodations
6	x6	Char	8		Hawaii, Scenery
7	x7	Char	8		Alaska, Scenery
8	x8	Char	8		Mexico, Scenery
9	x9	Char	8		California, Scenery
10	x10	Char	8		Maine, Scenery
11	x11	Num	8	DOLLAR5.	Hawaii, Price
12	x12	Num	8	DOLLAR5.	Alaska, Price
13	x13	Num	8	DOLLAR5.	Mexico, Price
14	x14	Num	8	DOLLAR5.	California, Price
15	x15	Num	8	DOLLAR5.	Maine, Price
16	Block	Num	8		
17	Run	Num	8		

%MktLab Macro Options

The following options can be used with the `%MktLab` macro.

Option	Description
`cfill=`*character-string*	character fill value
`data=`*SAS-data-set*	input design data set
`dolist=`*do-list*	new values using a do-list syntax
`int=`*variable-list*	name of an intercept variable
`key=`*SAS-data-set*	key data set
`labels=`*macro-name*	macro that provides labels and formats
`nfill=`*number*	numeric fill value
`out=`*SAS-data-set*	output data set with recoded design
`prefix=`*variable-prefix*	prefix for naming variables
`statements=`*SAS-code*	add extra statements
`values=`*value-list*	the new values for all of the variables
`vars=`*variable-list*	list of variable names

cfill= *character-string*
specifies the fill value in the `key=` data set for character variables. See the `nfill=` option for more information on fill values. The default is `cfill=' '`.

data= *SAS-data-set*
specifies the input data set with the experimental design, usually created by the `%MktEx` macro. The default is `data=Randomized`. The factor levels in the `data=` data set must be consecutive integers beginning with 1.

dolist= *do-list*
specifies the new values, using a do-list syntax (`n TO m <BY p>`), for example: `dolist=1 to 10` or `dolist=0 to 9`. With asymmetric designs (not all factors have the same levels), specify the levels for the largest number of levels. For example, with two-level and three-level factors and `dolist= 0 to 2`, the two-level factors will be assigned levels 0 and 1, and the three-level factors will be assigned levels 0, 1, and 2. Do not specify both `values=` and `dolist=`. By default, when `key=`, `values=`, and `dolist=` are all not specified, the default value list comes from `dolist=1 to 100`.

int= *variable-list*
specifies the name of an intercept variable (column of ones), if you want an intercept added to the `out=` data set. You can also specify a variable list instead of a variable name if you would like to make a list of variables with values all one. This can be useful, for example, for creating flag variables for generic choice models when the design is going to be used as a candidate set for the `%ChoicEff` macro.

key= *SAS-data-set*
specifies the input data set with the key to recoding the design. When `values=` or `dolist=` is specified, this data set is made for you. By default, when `key=`, `values=`, and `dolist=` are all not specified, the default value list comes from `dolist=1 to 100`.

labels= *macro-name*
specifies the name of a macro that provides labels, formats, or other additional information to the `key=` data set. For a simple format specification, it is easier to use `statements=`. For more involved specifications, use `labels=`. Note that you specify just the macro name, no percents on the `labels=` option. Example:

```
%mktex(3 ** 4, n=18, seed=205)

%macro labs;
   label x1 = 'Sploosh' x2 = 'Plumbob'
         x3 = 'Platter' x4 = 'Moosey';
   format x1-x4 dollar5.2;
   %mend;

%mktlab(values=1.49 1.99 2.49, labels=labs)

proc print label; run;
```

Obs	Sploosh	Plumbob	Platter	Moosey
1	$2.49	$2.49	$2.49	$1.49
2	$2.49	$2.49	$1.99	$1.99
3	$1.49	$1.49	$1.49	$1.49
4	$1.99	$1.99	$2.49	$2.49
.				
.				
.				

nfill= *number*
specifies the fill value in the `key=` data set for numeric variables. For example when the maximum number of levels is three, the last value in the `key=` data set for numeric two-level factors should have a value of `nfill=`, which by default is ordinary missing. If the macro tries to access one of these values, it prints a warning. If you would like ordinary missing (.) to be a legitimate level, specify a different `nfill=` value and use it for the extra places in the `key=` data set.

out= *SAS-data-set*
specifies the output data set with the final, recoded design. The default is `out=final`. Often, you will want to specify a two-level name to create a permanent SAS data set so the design will be available later for analysis.

prefix= *variable-prefix*
specifies a prefix for naming variables when `values=` is specified. For example `prefix=Var` creates variables `Var1`, `Var2`, and so on. By default, the variables are `x1`, `x2`, ... This option is ignored when `vars=` is specified.

statements= *SAS-code*
is an alternative to `labels=` that you can use to add extra statements to the `key=` data set. For a simple format specification, it is easier to use `statements=`. For more involved specifications, use `labels=`. Example:

```
%mktex(3 ** 4, n=18, seed=205)

%mktlab(values=1.49 1.99 2.49,
        vars=Sploosh Plumbob Platter Moosey,
        statements=format Sploosh Plumbob Platter Moosey dollar5.2)

proc print; run;
```

Obs	Sploosh	Plumbob	Platter	Moosey
1	$2.49	$2.49	$2.49	$1.49
2	$2.49	$2.49	$1.99	$1.99
3	$1.49	$1.49	$1.49	$1.49
4	$1.99	$1.99	$2.49	$2.49
.				
.				
.				

values= *value-list*
specifies the new values for all of the variables. If all variables will have the same value, it is easier to specify `values=` or `dolist=` than `key=`. When you specify `values=`, the `key=` data set is created for you. Specify a list of levels separated by blanks. If your levels contain blanks, separate them with two blanks. With asymmetric designs (not all factors have the same levels) specify the levels for the largest number of levels. For example, with two-level and three-level factors and `values=a b c`, the two-level factors will be assigned levels `'a'` and `'b'`, and the three-level factors will be assigned levels `'a'`, `'b'`, and `'c'`. Do not specify both `values=` and `dolist=`. By default, when `key=`, `values=`, and `dolist=` are all not specified, the default value list comes from `dolist=1 to 100`.

vars= *variable-list*
specifies a list of variable names when `values=` or `dolist=` is specified. If `vars=` is not specified with `values=`, then `prefix=` is used.

%MktLab Macro Notes

This macro specifies `options nonotes` throughout most of its execution. If you want to see all of the notes, submit the statement `%let mktopts = notes;` before running the macro. To see the macro version, submit the statement `%let mktopts = version;` before running the macro.

%MktMerge Macro

The %MktMerge autocall macro merges a data set containing an experimental design for a choice model with the data for the choice model. There are numerous examples of its usage from pages 171 through 339. Here is a typical usage of the macro.

```
%mktmerge(design=rolled, data=results, out=res2,
          nsets=18, nalts=5, setvars=choose1-choose18)
```

The design= data set comes from the %MktRoll macro. The data= data set contains the data, and the setvars= variables in the data= data set contain the numbers of the chosen alternatives for each of the 18 choice sets. The nsets= option specifies the number of choice sets, and the nalts= option specifies the number of alternatives. The out= option names the output SAS data set that contains the experimental design and a variable c that contains 1 for the chosen alternatives (first choice) and 2 for unchosen alternatives (second or subsequent choice).

When the data= data set contains a blocking variable, name it on the blocks= option. When there is blocking, it is assumed that the design= data set contains blocks of *nalts* × *nsets* observations. The blocks= variable must contain values 1, 2, ..., *n* for *n* blocks. Here is an example of using the %MktMerge macro with blocking.

```
%mktmerge(design=rolled, data=results, out=res2, blocks=form,
          nsets=18, nalts=5, setvars=choose1-choose18)
```

%MktMerge Macro Options

The following options can be used with the %MktMerge macro.

Option	Description
blocks=1 | *variable*	blocking variable
data=*SAS-data-set*	input SAS data set
design=*SAS-data-set*	input SAS choice design data set
nalts=*n*	number of alternatives
nsets=*n*	number of choice sets
out=*SAS-data-set*	output SAS data set
setvars=*variable-list*	variables with the data
statements=SAS-statements	additional statements

You must specify the design=, nalts=, nsets=, and setvars= options.

blocks= 1 | *variable*
specifies either a 1 (the default) if there is no blocking or the name of a variable in the data= data set that contains the block number. When there is blocking, it is assumed that the design= data set contains blocks of *nalts* × *nsets* observations, one set per block. The blocks= variable must contain values 1, 2, ..., *n* for *n* blocks.

data= *SAS-data-set*
specifies an input SAS data set with data for the choice model. By default, the `data=` data set is the last data set created.

design= *SAS-data-set*
specifies an input SAS data set with the choice design. This data set could have been created for example with the `%MktRoll` macro. This option must be specified.

nalts= *n*
specifies the number of alternatives. This option must be specified.

nsets= *n*
specifies the number of choice sets. This option must be specified.

out= *SAS-data-set*
specifies the output SAS data set. If `out=` is not specified, the DATAn convention is used. This data set contains the experimental design and a variable `c` that contains 1 for the chosen alternatives (first choice) and 2 for unchosen alternatives (second or subsequent choice).

setvars= *variable-list*
specifies a list of variables, one per choice set, in the `data=` data set that contains the numbers of the chosen alternatives. It is assumed that the values of these variables range from 1 to *nalts*. This option must be specified.

statements= SAS-statements
specifies additional statements like `format` and `label` statements. Example:

```
%mktmerge(design=rolled, data=results, out=res2, blocks=form,
          nsets=&n, nalts=&m, setvars=choose1-choose&n,
          statements=%str(price = input(put(price, price.), 5.);
                          format scene scene. lodge lodge.;))
```

%MktMerge Macro Notes

This macro specifies `options nonotes` throughout most of its execution. If you want to see all of the notes, submit the statement `%let mktopts = notes;` before running the macro. To see the macro version, submit the statement `%let mktopts = version;` before running the macro.

%MktOrth Macro

The %MktOrth macro lists some of the 100% orthogonal main-effects plans that the %MktEx macro can generate. See page 187 for an example. The %MktOrth macro can help you see what orthogonal designs are available and decide which ones to use. Here is a typical usage. This line requests all the designs in the catalog with 100 or fewer runs and two-level through six-level factors (with no higher-level factors.)

```
%mktorth(maxn=100, maxlev=6)
```

The macro creates data sets and no printed output except log notes.

```
NOTE: The data set WORK.MKTDESLEV has 345 observations and 9 variables.
NOTE: The data set WORK.MKTDESCAT has 345 observations and 3 variables.
```

This next step generates the entire catalog of 116,590 designs including an additonal 60,000 designs in 512 runs that are not generated by default.

```
%mktorth(maxlev=144, options=512)
```

This next step generates the catalog of 54,010 designs inlucing designs with up to 144-level factors. This step may take on the order of several minutes to run. Unless you really want to see all of the designs, you can make the %MktOrth macro run much faster by specifying smaller values for range= or maxn= (which control the number of runs) and maxlev= (which controls the maximum number of factor levels and the number of variables in the MKTDESLEV data set) than the defaults (range=n le 513, maxn=513, maxlev=50). The maximum number of runs is 513, and the maximum number of levels you can specifiy is 144.

```
%mktorth(maxlev=144)
```

Here are the first few and some of the last few designs in the catalog.

```
proc print data=mktdeslev(where=(n le 12 or n eq 512 and x2 le 4));
   var design reference;
   id n; by n;
   run;
```

n	Design	Reference
4	2 ** 3	Hadamard
6	2 ** 1 3 ** 1	Full-Factorial
8	2 ** 7	Hadamard
	2 ** 4 4 ** 1	Fractional-Factorial
9	3 ** 4	Fractional-Factorial
10	2 ** 1 5 ** 1	Full-Factorial
12	2 ** 11	Hadamard
	2 ** 4 3 ** 1	Orthogonal Array
	2 ** 2 6 ** 1	Orthogonal Array
	3 ** 1 4 ** 1	Full-Factorial

```
512    2 **  4          4 **169                          Fractional-Factorial
       2 **  4          4 ** 22   8 ** 63                Fractional-Factorial
       2 **  4          4 ** 17   8 ** 63  16 **  1     Fractional-Factorial
       2 **  4          4 ** 15   8 ** 66                Fractional-Factorial
       2 **  4          4 **  8   8 ** 69                Fractional-Factorial
       2 **  4          4 **  1   8 ** 72                Fractional-Factorial
       2 **  4          4 **  1   8 ** 63  64 **  1     Fractional-Factorial
       2 **  3          4 **167   8 **  1                Fractional-Factorial
       2 **  3          4 **159  32 **  1                Fractional-Factorial
       2 **  3          4 ** 20   8 ** 64                Fractional-Factorial
       2 **  3          4 ** 15   8 ** 64  16 **  1     Fractional-Factorial
       2 **  3          4 ** 13   8 ** 67                Fractional-Factorial
       2 **  3          4 **  6   8 ** 70                Fractional-Factorial
                        8 ** 73                          Fractional-Factorial
                        8 ** 64  64 **  1                Fractional-Factorial
                        4 **168   8 **  1                Fractional-Factorial
                        4 **160  32 **  1                Fractional-Factorial
                        4 ** 21   8 ** 64                Fractional-Factorial
                        4 ** 16   8 ** 64  16 **  1     Fractional-Factorial
                        4 ** 14   8 ** 67                Fractional-Factorial
                        4 **  7   8 ** 70                Fractional-Factorial
```

Here are the first few designs and variables in the MKTDESLEV data set.

```
proc print data=mktdeslev(where=(n le 12));
   var design reference x1-x6;
   id n; by n;
   run;
```

n	Design	Reference	x1	x2	x3	x4	x5	x6
4	2 ** 3	Hadamard	0	3	0	0	0	0
6	2 ** 1 3 ** 1	Full-Factorial	0	1	1	0	0	0
8	2 ** 7	Hadamard	0	7	0	0	0	0
	2 ** 4 4 ** 1	Fractional-Factorial	0	4	0	1	0	0
9	3 ** 4	Fractional-Factorial	0	0	4	0	0	0
10	2 ** 1 5 ** 1	Full-Factorial	0	1	0	0	1	0
12	2 ** 11	Hadamard	0	11	0	0	0	0
	2 ** 4 3 ** 1	Orthogonal Array	0	4	1	0	0	0
	2 ** 2 6 ** 1	Orthogonal Array	0	2	0	0	0	1
	3 ** 1 4 ** 1	Full-Factorial	0	0	1	1	0	0

If you just want to display a list of designs, possibly selecting on n, the number of runs, you can use the MKTDESCAT data set. However, if you would like to do more advanced processing, based on the numbers of levels of some of the factors, you can use the `outlev=mktdeslev` data set to select potential designs. You can look at the level information in MKTDESLEV and see the number of two-level factors in `x2`, the number of three-level factors in `x3`, ..., the number of fifty-level factors is in `x50`, ..., and the number of 144-level factors in `x144`. The number of one-level factors, `x1`, is always zero, but `x1` is available so you can make arrays (for example, `array x[50]`) and have `x[2]` refer to `x2`, the number of two-level factors, and so on.

Say you are interested in the design $2^5 3^5 4^1$. Here are some of the ways in which it is available.

```
%mktorth(maxn=100)

proc print data=mktdeslev noobs;
   where x2 ge 5 and x3 ge 5 and x4 ge 1;
   var n design reference;
   run;
```

n	Design	Reference
72	2 ** 44 3 ** 12 4 ** 1	Orthogonal Array
72	2 ** 43 3 ** 8 4 ** 1 6 ** 1	Orthogonal Array
72	2 ** 37 3 ** 13 4 ** 1	Orthogonal Array
72	2 ** 36 3 ** 9 4 ** 1 6 ** 1	Orthogonal Array
72	2 ** 35 3 ** 12 4 ** 1 6 ** 1	Orthogonal Array
.		
.		
.		

Here is one way that you can see all the designs in a certain range of sizes.

```
%mktorth(range=12 le n le 20)

proc print; id n; by n; run;
```

n	Design	Reference
12	2 ** 11	Hadamard
	2 ** 4 3 ** 1	Orthogonal Array
	2 ** 2 6 ** 1	Orthogonal Array
	3 ** 1 4 ** 1	Full-Factorial
14	2 ** 1 7 ** 1	Full-Factorial
15	3 ** 1 5 ** 1	Full-Factorial
16	2 ** 15	Hadamard
	2 ** 12 4 ** 1	Fractional-Factorial
	2 ** 9 4 ** 2	Fractional-Factorial

```
          2 **  8            8 **  1              Fractional-Factorial
          2 **  6            4 **  3              Fractional-Factorial
          2 **  3            4 **  4              Fractional-Factorial
                             4 **  5              Fractional-Factorial

     18   2 **  1  3 **  7                        Orthogonal Array
          2 **  1            9 **  1              Full-Factorial
                   3 **  6   6 **  1              Orthogonal Array

     20   2 ** 19                                 Hadamard
          2 **  8            5 **  1              Orthogonal Array
          2 **  2           10 **  1              Orthogonal Array
                             4 **  1   5 **  1    Full-Factorial
```

The `%MktOrth` macro can output the lineage of each design, which is the set of steps that the `%MktEx` macro uses to create it. Here is an example.

```
%mktorth(range=n=36, options=lineage)

proc print noobs;
   where index(design, '2 ** 11') and index(design, '3 ** 12');
   run;
```

```
 n          Design             Reference

36    2 ** 11  3 ** 12    Orthogonal Array

                          Lineage

36 ** 1 : 36 ** 1 > 3 ** 12 12 ** 1 : 12 ** 1 > 2 ** 11
```

The design $2^{11}3^{12}$ in 36 runs starts out as a single 36-level factor, 36^1. Then 36^1 is replaced by $3^{12}12^1$. Finally, 12^1 is replaced by 2^{11} resulting in $2^{11}3^{12}$.

%MktOrth Macro Options

The following options can be used with the `%MktOrth` macro.

Option	Description
`maxn=`*n*	maximum number of runs of interest
`maxlev=`*n*	maximum number of levels
`options=`*options-list*	binary options
`outall=`*SAS-data-set*	output data set with all designs
`outcat=`*SAS-data-set*	design catalog data set
`outlev=`*SAS-data-set*	output data set with the list of levls
`range=`*range-specification*	number of runs of interest

maxlev= *n*

specifies the maximum number of levels to consider. Specify a value n, such that $2 \leq n \leq 144$. The default is `maxlev=50`. This option controls the number of `x` variables in the `outlev=` data set. It also excludes from consideration designs with factors of more than `maxlev=` levels so it affects the number of rows in the output data sets. Note that specifying `maxlev=`n does not preclude designs with more than n-level factors from being used as parents for other designs, it just precludes the larger designs from being output. For example, with `maxlev=3`, the design $3^{12}12^1$ in 36 runs is used to make $2^{11}3^{12}$ before the $3^{12}12^1$ design is discarded. Specifying smaller values will make the macro run faster. With the maximum, `maxlev=144`, run time to generate the entire catalog can be on the order of several minutes.

maxn= *n*

specifies the maximum number of runs of interest. Specifying small numbers (e.g. $n \leq 200$) will make the macro run faster.

options= *options-list*

specifies binary options. By default, none of these options are specified. Specify one or more of the following values after `options=`.

`lineage`
construct the design lineage, which is the set of instructions on how the design is made.

`mktex`
specifies that the macro is being called from the `%MktEx` macro and just the `outlev=` data set is needed. The macro takes short cuts to make it run faster doing only what the `%MktEx` macro needs.

`mktruns`
specifies that the macro is being called from the `%MktRuns` macro and just the `outlev=` data set is needed. The macro takes short cuts to make it run faster doing only what `%MktRuns` needs.

`parent`
specifies that only parent designs should be output.

`dups`
specifies that the `%MktRuns` macro should not filter out duplicate and inferior designs from the catalog. This can be useful when you are creating a data set for the `cat=` option in the `%MktEx` macro.

`512`
adds some larger designs in 512 runs with mixes of 16, 8, 4, and 2-level factors to the catalog, which gives added flexibility in 512 runs at a cost of much slower run time. This option replaces the default $4^{160}32^1$ parent with $16^{32}32^1$.

outall= *SAS-data-set*

specifies the output data set with all designs. This data set is not created by default. This data set is like the `outlev=` data set, except larger. The `outall=` data set includes *all* of the `%MktEx` design catalog, including all of the smaller designs that can be trivially made from larger designs by dropping

factors. For example, when the `outlev=` data set has `x2`=2 `x3`=2, then the `outall=` data set has that design and also `x1=2 x3=1`, `x1=1 x3=2`, and `x1=1 x2=1`. When you specify `outall=` you must also specify a reasonably small `range=` or `maxn=` value. Otherwise, the `outall=` specification will take a *long* time and create a *huge* data set, which will very likely be too large to store on your computer.

outcat= *SAS-data-set*
specifies the output data set with the catalog of designs that the `%MktEx` macro can create. The default is `outcat=MktDesCat`.

outlev= *SAS-data-set*
specifies the output data set with the list of designs and 50 (by default) more variables, `x1-x50`, which includes: `x2` - the number of two-level factors, `x3` - the number of three-level factors, and so on. The default is `outlev=MktDesLev`. The number of `x` variables is determined by the `maxlev=` option.

range= *range-specification*
specifies the number of runs of interest. Specify a range involving `n`, where `n` is the number of runs. Your range specification must be a logical expression involving `n`. Examples:

```
range=n=36
range=18 le n le 36
range=n eq 18 or n eq 36
```

%MktOrth Macro Notes

This macro specifies `options nonotes` throughout most of its execution. If you want to see all of the notes, submit the statement `%let mktopts = notes;` before running the macro. To see the macro version, submit the statement `%let mktopts = version;` before running the macro.

%MktPPro Macro

The `%MktPPro` macro is used to make a partial-profile design from an incomplete blocks design and an orthogonal array. This macro reads an orthogonal array and an incomplete blocks design and makes an optimal partial-profile choice design. This example uses a known balanced incomplete blocks design (BIBD). An IBD is balanced when each attribute pair appears equally often. In the context of partial profiles, an IBD is used to determine which attributes to vary in each choice set. This example makes an *optimal* partial-profile choice design with 16 binary attributes, four of which vary, and it does so in 80 choice sets.

```
%mktex(2 4 2 2 2, n=8)

proc print; run;

data design; set design; drop x2; run;

proc iml;
   b = { 1  1  1  1  1  2  2  2  2  3  3  3  3  4  4  4  4  5  6  7 ,
         2  5  8 11 14  5  6  7 10  5  6  7  9  5  6  7  8  9 10  8 ,
         3  6  9 12 15  8 12  9 13 13  8 10 11 10  9 11 12 12 11 13 ,
         4  7 10 13 16 11 14 15 16 15 16 12 14 14 13 16 15 16 15 14 }‘;
   create b from b; append from b;
   quit;

%mktppro(ibd=b, print=f p)

%choiceff(data=chdes, model=class(x1-x16), nsets=80, nalts=2,
          beta=zero, init=chdes, initvars=x1-x16)
```

In this example code, a BIBD is entered through IML and it is transposed, so the BIBD has 20 rows and 4 columns, and the first row is (1 2 3 4). This says that in the first block of $m = 4$ choice sets, attributes 1, 2, 3, and 4 will vary; in the second block of 4 choice sets, attributes 1, 5, 6, and 7 will vary; and so on. Alternative $k = 1, ..., p$, in each block of m choice sets will be made from a block of m runs in the orthogonal array. For example, alternative 1 of the first m choice sets will be made from the first m runs in the orthogonal array, and alternative 2 will be made from the next m runs in the orthogonal array, and so on. The on-diagonal and above-diagonal elements of this matrix show how often each attribute appears with every other apptribute in our design. Each attribute must appear an equal number of times in an IBD, and this is shown on the diagonal. When every element above the diagonal is 1, as it is in this case, the IBD is *balanced* or a BIBD. If the diagonal were constant (4, 3, 2, 1) and the above-diagonal entries were ones and zeros, then we would have an IBD.

Attribute Frequencies

5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	5	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	5	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	5	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	5	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	5	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	5	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	5	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	5	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	5	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	5	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	5	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	5	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	5	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5

The number of choice sets is m times the number of rows in the IBD (in this case, $4 \times 20 = 80$). The number of attributes is the maximum value in the IBD (in this case 16). The number of attributes that vary at any one time is m. All factors have p levels, and all choice sets have p alternatives.

Two designs are input to the `%MktPPro` macro. The second is an orthogonal array. The orthogonal array must be a p^m subset of the design $p^m m^1$ in $p \times m$ runs. Examples: 2^4 in 8 runs, selected from $2^4 4^1$ in 8 runs; 3^3 in 9 runs, selected from $3^3 3^1$ in 9 runs; 4^4 in 16 runs, selected from $4^4 4^1$ in 16 runs The number of levels, p, must be a prime or a power of a prime: 2, 3, 4, 5, 7, 8, 9, 11, ... You can use the `%MktOrth` macro to see if the orthogonal array of interest exists.

The rows of the orthogonal array design must be sorted into the right order. The easiest way to do this is to first request one of the m p-level factors, then request the m-level factor, then request the remaining $(m-1)$ p-level factors. Then *after* the design is created, discard `x2`, the m-level factor, as shown in the example code.

All but the most interested readers may skip this paragraph. Our goal is to create an orthogonal array with p blocks of m rows. Each block is a difference scheme, and blocks 2 through p are obtained from the preceding block by adding 1 (in the appropriate Galois field). For example, when p is 2, add 1 modulo 2; and when p is 3, add 1 modulo 3. You will not get optimal results if you stick in any other kind of orthogonal array.

The approach of asking for m^1 as the second factor and then dropping it always works. Sometimes it is possible to not ask for m^1 and still get the right results, but you must ensure that you made the right design. Note that you cannot drop the second factor in the `%MktEx` step by specifying `out=design(drop=x2)` because `%MktEx` will drop the variable and then sort, and your rows will be in the wrong order.

Here is an example of using `%MktEx` with restrictions to find an IBD, using `%MktEx` to find the orthogonal array, using `%MktPPro` to create the partial-profile choice design, and using `%ChoicEff` to evaluate the partial-profile design. The restrictions that are used to get the IBD are discussed in detail starting on page 445.

```
%let lev  = 4;   /* levels                                        */
%let attr = 16;  /* total attributes                              */
%let vary = 4;   /* number of attributes that vary                */
%let rep  = 2;   /* number of times each attribute must appear */
%let seed = 17;  /* random number seed                            */

%let b    = %sysevalf((&attr * &rep) / &vary);
%let sets = %eval(&b * &vary);

data _null_;
   if &b ne floor(&b) then do;
      put "ERROR: b = attr * rep / vary = &b must be an integer.";
      put 'ERROR: Try changing rep or attr.';
      end;
   put "NOTE: attr=&attr, rep=&rep, vary=&vary, sets=&sets, b=&b..";
   run;

%macro con;
   _d = &rep # i(&m);
   _f = ((1:&m) @ j(&m, 1, 1) > j(1, &m, 1) @ (1:&m)`) + _d;
   _p = (_f = &rep) + 10 # (_f = 1);
   %mend;

%macro res;
   bad = 1000 # abs(sum(x = 2) - &vary);
   f = j(&m, &m, 0);
   do ii = 1 to &n;
      if ii = i then l = loc(x = 2);
      else l = loc(xmat[ii,] = 2);
      if ncol(l) then f[l, l] = f[l, l] + 1;
      end;
   bad = bad + sum((abs(f - _f) >< abs(f - _d)) # _p);
   %mend;

%mktex(2 ** &attr, n=&b, tabiter=0, optiter=0,
       restrictions=res, resmac=con, reslist=%str(_f, _p, _d),
       order=random, options=resrep nofinal, exchange=2,
       seed=&seed, maxdesigns=1, out=ibd, ridge=0.01)

%mktex(&lev &vary &lev ** %eval(&vary - 1), n=&lev * &vary)

data design; set design; drop x2; run;

%mktppro(x=ibd)

%choiceff(data=chdes, model=class(x1-x&attr), nsets=&sets, nalts=&lev,
          beta=zero, init=chdes, initvars=x1-x&attr)
```

%MktPPro Macro Options

The following options can be used with the `%MktPPro` macro.

Option	Description
`ibd=`*SAS-data-set*	incomplete blocks design
`design=`*SAS-data-set*	orthogonal array
`out=`*SAS-data-set*	output partial-profile design
`print=`*print-options*	printing options
`x=`*SAS-data-set*	binary BIBD from `%MktEx`

You must specify either `ibd=` or `x=` but not both.

ibd= *SAS-data-set*
specifies an incomplete blocks design.

design= *SAS-data-set*
specifies the orthogonal array design from the `%MktEx` macro.

out= *SAS-data-set*
specifies the output choice design.

print= *print-options*
specifies both of the printing options, which control the printing of the results. The default is `print=f`. Specify one or more values from the following list.

x= *SAS-data-set*
specifies the x-matrix version of the BIBD, which is a binary coding of the BIBD (or more typically a subset of the BIBD) and is typically created by the `%MktEx` macro.

`i` incomplete blocks design
`f` crosstabulation of attribute frequencies
`p` partial-profile design

%MktPPro Macro Notes

This macro specifies `options nonotes` throughout most of its execution. If you want to see all of the notes, submit the statement `%let mktopts = notes;` before running the macro. To see the macro version, submit the statement `%let mktopts = version;` before running the macro.

%MktRoll Macro

The %MktRoll autocall macro is used for manipulating the experimental design for choice experiments. There are numerous examples of its usage from pages 165 through 406. The %MktRoll macro takes as input a SAS data set containing an experimental design with one row per choice set, the *linear design*, for example a design created by the %MktEx macro. This data set is specified in the design= option. This data set has one variable for each attribute of each alternative in the choice experiment. The output from this macro is an out= SAS data set is the *choice design* containing the experimental design with one row per alternative per choice set. There is one column for each different attribute. For example, in a simple branded study, design= could contain the variables x1-x5 which contain the prices of each of five alternative brands. The output data set would have one factor, Price, that contains the price of each of the five alternatives. In addition, it would have the number (or optionally the name) of each alternative.¶

The rules for determining the mapping between factors in the design= data set and the out= data set are contained in the key= data set. For example, assume that the design= data set contains the variables x1-x5 which contain the prices of each of five alternative brands: Brand A, B, C, D, and E. Here is how you would create the key= data set. The choice design has two factors, Brand and Price. Brand A price is made from x1, Brand B price is made from x2, ..., and Brand E price is made from x5.

A convenient way to get all the names in a variable list like x1-x5 is with the %MktKey macro. This askes for five names in a single column.

```
%mktkey(5 1)
```

The %MktKey macro produced the following data set.

x1
x1
x2
x3
x4
x5

Here is the KEY data set.

```
data key;
   input (Brand Price) ($);
   datalines;
A x1
B x2
C x3
D x4
E x5
;
```

¶See page 60 for an illustration of linear versus choice designs.

This data set has two variables. `Brand` contains the brand names, and `Price` contains the names of the factors that are used to make the price effects for each of the alternatives. The `out=` data set will contain the variables with the same names as the variables in the `key=` data set.

Here is how you can create the linear design with one row per choice set:

```
%mktex(3 ** 5, n=12)
```

Here is how you can create the choice design with one row per alternative per choice set:

```
%mktroll(design=randomized, key=key, out=sasuser.design, alt=brand)
```

For example, if the data set RANDOMIZED contains the row:

Obs	x1	x2	x3	x4	x5
9	3	1	1	2	1

then the data set SASUSER.DESIGN contains the rows:

Obs	Set	Brand	Price
41	9	A	3
42	9	B	1
43	9	C	1
44	9	D	2
45	9	E	1

The price for Brand A is made from `x1=3`, ..., and the price for Brand E is made from `x5=1`.

Now assume that there are three alternatives, each a different brand, and each composed of four factors: `Price`, `Size`, `Color`, and `Shape`. In addition, there is a constant alternative. First, the `%MktEx` macro is used to create a design with 12 factors, one for each attribute of each alternative.

```
%mktex(2 ** 12, n=16, seed=109)
```

Next, the `key=` data set is created. It shows that there are three brands, A, B, and C, and also None.

```
data key;
   input (Brand Price Size Color Shape) ($); datalines;
          A     x1     x2   x3    x4
          B     x5     x6   x7    x8
          C     x9    x10  x11   x12
          None   .      .    .     .
;
```

Brand A is created from `Brand` = "A", `Price` = `x1`, `Size` = `x2`, `Color` = `x3`, `Shape` = `x4`.

Brand B is created from `Brand` = "B", `Price` = `x5`, `Size` = `x6`, `Color` = `x7`, `Shape` = `x8`.

Brand C is created from `Brand` = "C", `Price` = `x9`, `Size` = `x10`, `Color` = `x11`, `Shape` = `x12`.

The constant alternative is created from `Brand` = "None" and none of the attributes. The "`.`" notation is used to indicate missing values in input data sets. The actual values in the KEY data set will be blank (character missing).

Here is how you create the design with one row per alternative per choice set:

```
%mktroll(key=key, design=randomized, out=sasuser.design, alt=brand)
```

For example, if the data set RANDOMIZED contains the row:

Obs	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12
8	2	2	2	2	2	1	1	2	2	2	1	2

then the data set SASUSER.DESIGN contains the rows:

29	8	A	2	2	2	2
30	8	B	2	1	1	2
31	8	C	2	2	1	2
32	8	None	.	.	.	.

Now assume like before that there are three branded alternatives, each composed of four factors: `Price`, `Size`, `Color`, and `Shape`. In addition, there is a constant alternative. Also, there is an alternative-specific factor, `Pattern`, that only applies to Brand A and Brand C. First, the `%MktEx` macro is used to create a design with 14 factors, one for each attribute of each alternative.

```
%mktex(2 ** 14, n=16, seed=114)
```

Next, the `key=` data set is created. It shows that there are three brands, A, B, and C, plus None.

```
data key;
   input (Brand Price Size Color Shape Pattern) ($);
   datalines;
A     x1     x2   x3    x4   x13
B     x5     x6   x7    x8     .
C     x9    x10  x11   x12   x14
None   .      .    .     .     .
;
```

Brand A is created from `Brand` = "A", `Price` = `x1`, `Size` = `x2`, `Color` = `x3`, `Shape` = `x4`, `Pattern` = `x13`.

Brand B is created from `Brand` = "B", `Price` = `x5`, `Size` = `x6`, `Color` = `x7`, `Shape` = `x8`.

Brand C is created from `Brand` = "C", `Price` = `x9`, `Size` = `x10`, `Color` = `x11`, `Shape` = `x12`, `Pattern` = `x14`.

The constant alternative is `Brand` = "None" and none of the attributes.

Here is how you can create the design with one row per alternative per choice set:

```
%mktroll(key=key, design=randomized, out=sasuser.design, alt=brand)
```

For example, if the data set RANDOMIZED contains the row:

Obs	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14
8	2	1	1	2	1	2	1	2	1	1	1	2	1	2

then the data set SASUSER.DESIGN contains the rows:

Obs	Set	Brand	Price	Size	Color	Shape	Pattern
29	8	A	2	1	1	2	1
30	8	B	1	2	1	2	.
31	8	C	1	1	1	2	2
32	8	None	.	.	.	.	.

Now assume we are going to fit a model with price cross effects so we need `x1`, `x5`, and `x9` (the three price effects) available in the `out=` data set. See pages 261 and 283 for other examples of cross effects.

```
%mktroll(key=key, design=randomized, out=sasuser.design, alt=brand,
         keep=x1 x5 x9)
```

Now the data set also contains the three original price variables.

Obs	Set	Brand	Price	Size	Color	Shape	Pattern	x1	x5	x9
29	8	A	2	1	1	2	1	2	1	1
30	8	B	1	2	1	2	.	2	1	1
31	8	C	1	1	1	2	2	2	1	1
32	8	None	.	.	.	.	.	2	1	1

Every value in the `key=` data set must appear as a variable in the `design=` data set. The macro prints a warning if it encounters a variable name in the `design=` data set that does not appear as a value in the `key=` data set.

%MktRoll Macro Options

The following options can be used with the `%MktRoll` macro.

Option	Description
`alt=`*variable*	variable with name of each alternative
`design=`*SAS-data-set*	input SAS data set
`keep=`*variable-list*	factors to keep
`key=`*SAS-data-set*	key data set
`options=`*options-list*	binary options
`out=`*SAS-data-set*	output SAS data set
`set=`*variable*	choice set number variable

You must specify the `design=` and `key=` options.

alt= *variable*
specifies the variable in the `key=` data set that contains the name of each alternative. Often this will be something like `alt=Brand`. When `alt=` is not specified, the macro creates a variable `_Alt_` that contains the alternative number.

design= *SAS-data-set*
specifies an input SAS data set with one row per choice set. The `design=` option must be specified.

keep= *variable-list*
specifies factors from the `design=` data set that should also be kept in the `out=` data set. This option is useful to keep terms that will be used to create cross effects.

key= *SAS-data-set*
specifies an input SAS data set containing the rules for mapping the `design=` data set to the `out=` data set. The `key=` option must be specified.

options= *options-list*
specifies binary options. By default, none of these options are specified. Specify one or more of the following values after `options=`.

`nowarn`
do not print a warning when the `design=` data set contains variables not mentioned in the `key=` data set. Sometimes this is perfectly fine.

out= *SAS-data-set*
specifies the output SAS data set. If `out=` is not specified, the DATAn convention is used.

set= *variable*
specifies the variable in the `out=` data set that will contain the choice set number. By default, this variable is named `Set`.

%MktRoll Macro Notes

This macro specifies `options nonotes` throughout most of its execution. If you want to see all of the notes, submit the statement `%let mktopts = notes;` before running the macro. To see the macro version, submit the statement `%let mktopts = version;` before running the macro.

%MktRuns Macro

runs

The %MktRuns autocall macro suggests reasonable sizes for experimental designs. There are numerous examples of its usage from pages 156 through 364. The %MktRuns macro tries to find sizes in which perfect balance and orthogonality can occur, or at least sizes in which violations of orthogonality and balance are minimized. Typically, the macro takes one argument, a list of the number of levels of each factor.

For example, with 3 two-level and 4 three-level factors, specify either of the following.

```
%mktruns( 2 2 2 3 3 3 3 )

%mktruns( 2 ** 3 3 ** 4 )
```

The output from the macro in this example is:

Design Summary

Number of Levels	Frequency
2	3
3	4

Saturated = 12
Full Factorial = 648

Some Reasonable Design Sizes	Violations	Cannot Be Divided By
36 *	0	
72 *	0	
18	3	4
54	3	4
12	6	9
24	6	9
48	6	9
60	6	9
30	9	4 9
42	9	4 9

* - 100% Efficient Design can be made with the MktEx Macro.

n	Design	Reference
36	2 ** 13 3 ** 4	Orthogonal Array
36	2 ** 11 3 ** 12	Orthogonal Array
36	2 ** 10 3 ** 8 6 ** 1	Orthogonal Array

```
36    2 **  9  3 **  4   6 **  2            Orthogonal Array
36    2 **  4  3 ** 13                       Orthogonal Array
36    2 **  3  3 **  9   6 **  1            Orthogonal Array
72    2 ** 49  3 **  4                       Orthogonal Array
72    2 ** 47  3 ** 12                       Orthogonal Array
72    2 ** 46  3 **  8   6 **  1            Orthogonal Array
72    2 ** 46  3 **  4   4 **  1            Orthogonal Array
72    2 ** 45  3 **  4   6 **  2            Orthogonal Array
 .
 .
 .
```

The macro reports that the saturated design has 12 runs and that 36 and 72 are optimal design sizes. The macro picks 36, because it is the smallest integer >= 12 that can be divided by 2, 3, 2×2, 2×3, and 3×3. The macro also reports 18 as a reasonable size. There are three violations with 18, because 18 cannot be divided by each of the three pairs of 2×2, so perfect orthogonality in the two-level factors will not be possible with 18 runs. Larger sizes are reported as well. The macro prints orthogonal designs that are available from the `%MktEx` macro that match your specification.

To see every size the macro considered, simply run PROC PRINT after the macro finishes. The output from this step is not shown.

```
proc print label data=nums split='-';
   id n;
   run;
```

For 2 two-level factors, 2 three-level factors, 2 four-level factors, and 2 five-level factors specify:

```
%mktruns( 2 2 3 3 4 4 5 5 )
```

Here are the results:

```
                    Design Summary

                  Number of
                  Levels       Frequency

                       2               2
                       3               2
                       4               2
                       5               2

Saturated      = 21
Full Factorial = 14,400
```

Some Reasonable Design Sizes	Violations	Cannot Be Divided By
120	3	9 16 25
180	6	8 16 25
60	7	8 9 16 25
144	15	5 10 15 20 25
48	16	5 9 10 15 20 25
72	16	5 10 15 16 20 25
80	16	3 6 9 12 15 25
96	16	5 9 10 15 20 25
160	16	3 6 9 12 15 25
192	16	5 9 10 15 20 25

Among the smaller design sizes, 60 or 48 look like good possibilities.

The macro has an optional keyword parameter: `max=`. It specifies the maximum number of sizes to try. Usually you will not need to specify the `max=` option. The smallest design that is considered is the saturated design. This next specification tries 5000 sizes (21 to 5020) and reports that a perfect design can be found with 3600 runs The `%MktEx` macro does not explicitly know how to make this design, however, it can usually find it or come extremely close with the coordinate exchange algorithm.

```
%mktruns(2 2 3 3 4 4 5 5, max=5000)
```

Design Summary

Number of Levels	Frequency
2	2
3	2
4	2
5	2

Saturated = 21
Full Factorial = 14,400

Some Reasonable Design Sizes	Violations	Cannot Be Divided By
3600	0	
720	1	25
1200	1	9
1440	1	25
1800	1	16

```
2160              1    25
2400              1     9
2880              1    25
4320              1    25
4800              1     9
```

Now consider again the problem with 3 two-level and 4 three-level factors, but this time we want to be estimable the interaction of two of the two-level factors.

```
%mktruns( 2 2 2 3 3 3 3, interact=1*2, options=source )
```

Since `options=source` was specified, the first part of the output lists the sources for orthogonality violations that the macro will consider. We see that n must be divided by: 2 since `x1-x3` are two-level factors, 3 since `x4-x7` are three-level factors, 4 since `x1*x2`, `x1*x3`, and `x2*x3` interactions are specified, 6 since we have two-level and three-level factors, 8 since we have the two-way interaction of 2 two-level factors and the main-effect of an additional two-level factor, 9 since we have multiple three-level factors, and 12 since we have the two-way interaction of 2 two-level factors and the main-effect of additional three-level factors.

N Must Be Divided By	Source	Variables
2	2	1
		2
		3
3	3	4
		5
		6
		7
4	2*2	1 2
		1 3
		2 3
6	2*3	1 4
		1 5
		1 6
		1 7
		2 4
		2 5
		2 6
		2 7
		3 4
		3 5
		3 6
		3 7
8	2*2*2	1 2 3

```
 9        3*3          4 5
                       4 6
                       4 7
                       5 6
                       5 7
                       6 7

12        2*2*3        1 2 4
                       1 2 5
                       1 2 6
                       1 2 7


 Number of
 Levels       Frequency

      2           1
      3           4

Saturated      = 13
Full Factorial = 648

 Some Reasonable                    Cannot Be
    Design Sizes     Violations     Divided By

             72              0
            144              0
             36              1     8
            108              1     8
             24              6     9
             48              6     9
             96              6     9
            120              6     9
             60              7     8  9
             84              7     8  9
```

Now we need 72 runs for perfect balance and orthogonality although the `%MktEx` design catalog does not contain designs with interactions.

%MktRuns Macro Options

The following options can be used with the `%MktRuns` macro.

Option	Description
`interact=`*interaction-list*	interaction terms
`list`	numbers of levels of all the factors
`max=`n $< m >$	maximum number of design sizes to try
`n=`n	design size to evaluate
`maxlev=`n	maximum number of levels
`options=`*options-list*	binary options
`out=`*SAS-data-set*	data set with the suggested sizes

The `%MktRuns` macro has one positional parameter, `list`, and several keyword parameters.

list

specifies a list of the numbers of levels of all the factors. For example, for 3 two-level factors specify either `2 2 2` or `2 ** 3`. Lists of numbers, like `2 2 3 3 4 4` or a *levels**number of factors* syntax like: `2**2 3**2 4**2` can be used, or both can be combined: `2 2 3**4 5 6`. The specification `3**4` means 4 three-level factors. You must specify a list. Note that the factor list is a positional parameter. This means it must come first, and unlike all other parameters, it is not specified after a name and an equal sign.

interact= *interaction-list*

specifies interactions that must be estimable. By default, no interactions are guaranteed to be estimable. Examples:

```
interact=x1*x2
interact=x1*x2 x3*x4*x5
interact=x1|x2|x3|x4|x5@2
```

The interaction syntax is like PROC GLM's and many of the other modeling procedures. It uses "`*`" for simple interactions (`x1*x2` is the interaction between `x1` and `x2`), "`|`" for main effects and interactions (`x1|x2|x3` is the same as `x1 x2 x1*x2 x3 x1*x3 x2*x3 x1*x2*x3`) and "`@`" to eliminate higher-order interactions (`x1|x2|x3@2` eliminates `x1*x2*x3` and is the same as `x1 x2 x1*x2 x3 x1*x3 x2*x3`). The specification "`@2`" allows only main effects and two-way interactions. Only "`@`" values of `2` or `3` are allowed.

max= n $< m >$

specifies the maximum number of design sizes to try. By default, `max=200 2`. The macro tries up to n sizes starting with the saturated design. The macro stops trying larger sizes when it finds a design size with zero violations that is m times as big as a previously found size with zero violations. The macro reports the best 10 sizes. For example, if the saturated design has 10 runs, and there are zero violations in 16 runs, then by default, the largest size that the macro will consider is $32 = 2 \times 16$ runs.

maxlev= *n*
specifies the maximum number of levels to consider when generating the orthogonal array list. The default is `maxlev=50`, and the actual maximum is the max of the specified `maxlev=` value and the maximum number of levels in the factor list. Specify a value $2 \leq n \leq 144$.

n= *n*
specifies the design size to evaluate. By default, this option is not specified, and the `max=` option specification provides a range of design sizes to evaluate.

options= *options-list*
specifies binary options. By default, none of these options are specified. Specify one the following values after `options=`.

`justparse`
is used by other `Mkt` macros to have this macro just parse the list argument and return it as a simple list of integers.

`multiple`
specifies that a term that is required for orthogonality may be counted multiple times when counting orthogonality violations. For example, combinations of levels for the pair of variable 1 and variable 2 must have equal frequencies for orthogonality in the main effects, and if two-way interactions are required as well, then the (1, 2) pair must have equal frequencies again. By default, each combination of variables is only counted once. The difference between the single and multiple sources is the single source method just counts the places in the design where equal frequencies must occur. The multiple source method weights this count by the number of times each of these terms is important for achieving orthogonality. The results for the two methods are often highly correlated, but they can be different.

`multiple2`
specifies both `option=multiple` and more detailed output including the reason each term appears is added to the source table when `options=source multiple2` is specified.

`source`
prints the source of all of the numbers in the table of reasonable design sizes.

`512`
adds some larger designs in 512 runs with mixes of 16, 8, 4, and 2-level factors to the catalog, which gives added flexibility in 512 runs at a cost of much slower run time. This option replaces the default $4^{160}32^{1}$ parent with $16^{32}32^{1}$.

out= *SAS-data-set*
specifies the name of a SAS data set with the suggested sizes. The default is `out=nums`.

%MktRuns Macro Notes

This macro specifies `options nonotes` throughout most of its execution. If you want to see all of the notes, submit the statement `%let mktopts = notes;` before running the macro. To see the macro version, submit the statement `%let mktopts = version;` before running the macro.

%PhChoice Macro

The %PhChoice autocall macro is used to customize the discrete choice output from PROC PHREG. Typically, you run the following macro once to customize the PROC PHREG output.

```
%phchoice(on)
```

The macro uses PROC TEMPLATE and ODS (Output Delivery System) to customize the output from PROC PHREG. Running this code edits the templates and stores copies in SASUSER. These changes will remain in effect until you delete them. Note that these changes assume that each effect in the choice model has a variable label associated with it so there is no need to print variable names. If you are coding with PROC TRANSREG, this will usually be the case. To return to the default output from PROC PHREG, run the following macro.

```
%phchoice(off)
```

If you ever have errors running this macro, like invalid page errors, see "Macro Errors" on page 784. The rest of this section discusses the details of what the %PhChoice macro does and why. Unless you are interested in further customization of the output, you should skip to "%PhChoice Macro Options" on page 752.

We are most interested in the Analysis of Maximum Likelihood Estimates table, which contains the parameter estimates. We can first use PROC TEMPLATE to identify the template for the parameter estimates table and then edit the template. First, let's have PROC TEMPLATE display the templates for PROC PHREG. The source stat.phreg statement specifies that we want to see PROC TEMPLATE source code for the STAT product and the PHREG procedure.

```
proc template;
   source stat.phreg;
   run;
```

If we search the results for the Analysis of Maximum Likelihood Estimates table we find the following code, which defines the Stat.Phreg.ParameterEstimates table.

```
define table Stat.Phreg.ParameterEstimates;
   notes "Parameter Estimates Table";
   dynamic Confidence NRows;
   column Variable DF Estimate StdErr StdErrRatio ChiSq ProbChiSq HazardRatio
      HRLowerCL HRUpperCL Label;
   header h1 h2;
   define h1;
      text "Analysis of Maximum Likelihood Estimates";
      space = 1;
      spill_margin;
   end;
   define h2;
      text Confidence BEST8. %nrstr("%% Hazard Ratio Confidence Limits");
      space = 0;
      end = HRUpperCL;
      start = HRLowerCL;
      spill_margin = OFF;
   end;
```

```
define Variable;
   header = "Variable";
   style = RowHeader;
   id;
end;
define DF;
   parent = Common.ParameterEstimates.DF;
end;
define Estimate;
   header = ";Parameter;Estimate;";
   format = D10.;
   parent = Common.ParameterEstimates.Estimate;
end;
define StdErr;
   header = ";Standard;Error;";
   format = D10.;
   parent = Common.ParameterEstimates.StdErr;
end;
define StdErrRatio;
   header = ";StdErr;Ratio;";
   format = 6.3;
end;
define ChiSq;
   parent = Stat.Phreg.ChiSq;
end;
define ProbChiSq;
   parent = Stat.Phreg.ProbChiSq;
end;
define HazardRatio;
   header = ";Hazard;Ratio;";
   glue = 2;
   format = 8.3;
end;
define HRLowerCL;
   glue = 2;
   format = 8.3;
   print_headers = OFF;
end;
define HRUpperCL;
   format = 8.3;
   print_headers = OFF;
end;
define Label;
   header = "Variable Label";
end;
```

```
      col_space_max = 4;
      col_space_min = 1;
      required_space = NRows;
   end;
```

It contains header, format, spacing and other information for each column in the table. Most of this need not concern us now. The template contains this `column` statement, which lists the columns of the table.

```
      column Variable DF Estimate StdErr StdErrRatio ChiSq ProbChiSq HazardRatio
         HRLowerCL HRUpperCL Label;
```

Since we will usually have a label that adequately names each parameter, we do not need the variable column. We also do not need the hazard information. If we move the label to the front of the list and drop the variable column and the hazard columns, we get this.

```
   column Label DF Estimate StdErr ChiSq ProbChiSq;
```

We use the `edit` statement to edit the template. We can also modify some headers. We specify the new `column` statement and the new headers. We can also modify the Summary table, which is `Stat.Phreg.CensoredSummary`, to use the vocabulary of choice models instead of survival analysis models. The code is grabbed from the PROC TEMPLATE step with the `source` statement. The overall header "Summary of the Number of Event and Censored Values" is changed to "Summary of Subjects, Sets, and Chosen and Unchosen Alternatives", "Total" is changed to "Number of Alternatives", "Event" is changed to "Chosen Alternatives", "Censored" is changed to "Not Chosen", and "Percent Censored" is dropped. Finally `Style=RowHeader` was specified on the label column. This sets the color, font, and general style for HTML output. The `RowHeader` style is typically used on first columns that provide names or labels for the rows. Here is the code that the `%phchoice(on)` macro runs.

```
   proc template;
      edit stat.phreg.ParameterEstimates;
         column Label DF Estimate StdErr ChiSq ProbChiSq;
         header h1;
         define h1;
            text "Multinomial Logit Parameter Estimates";
            space = 1;
            spill_margin;
            end;
         define Label;
            header = " " style = RowHeader;
            end;
         end;
         edit Stat.Phreg.CensoredSummary;
            column Stratum Pattern Freq GenericStrVar Total
                   Event Censored;
            header h1;
            define h1;
               text "Summary of Subjects, Sets, "
                    "and Chosen and Unchosen Alternatives";
               space = 1;
               spill_margin;
               first_panel;
            end;
```

```
      define Freq;
        header=";Number of;Choices" format=6.0;
      end;
   define Total;
      header = ";Number of;Alternatives";
      format_ndec = ndec;
      format_width = 8;
   end;
   define Event;
      header = ";Chosen;Alternatives";
      format_ndec = ndec;
      format_width = 8;
   end;
   define Censored;
      header = "Not Chosen";
      format_ndec = ndec;
      format_width = 8;
   end;
   end;

run;
```

Here is the code that `%phchoice(off)` runs.

```
* Delete edited templates, restore original templates;
proc template;
   delete Stat.Phreg.ParameterEstimates;
   delete Stat.Phreg.CensoredSummary;
   run;
```

Our editing of the multinomial logit parameter estimates table assumes that each independent variable has a label. If you are coding with PROC TRANSREG, this will be true of all variables created by `class` expansions. You may have to provide labels for `identity` and other variables. Alternatively, if you want variable names to appear in the table, you can do that as follows. This may be useful when you are not coding with PROC TRANSREG.

```
%phchoice(on, Variable DF Estimate StdErr ChiSq ProbChiSq Label)
```

The optional second argument provides a list of the column names to print. The available columns are: `Variable DF Estimate StdErr StdErrRatio ChiSq ProbChiSq HazardRatio HRLowerCL HRUpperCL Label`. (`HRLowerCL` and `HRUpperCL` are confidence limits on the hazard ratio.) For very detailed customizations, you may have to run PROC TEMPLATE directly.

%PhChoice Macro Options

The `%PhChoice` macro has two positional parameters, `onoff` and `column`. Positional parameters must come first, and unlike all other parameters, are not specified after a name and an equal sign.

onoff

`ON` specifies choice model customization.

`OFF` turns off the choice model customization and returns to the default PROC PHREG templates.

`EXPB` turns on choice model customization and adds the hazard ratio to the output.

Upper/lower case does not matter.

column

specifies an optional column list for more extensive customizations.

%PlotIt Macro

The %PlotIt macro is used to make graphical scatter plots of labeled points. It is particularly designed to display raw data and results from analyses such as regression, correspondence analysis, MDPREF, PREFMAP, and MDS. However, it can make many other types of graphical displays as well. It can plot points, labeled points, vectors, circles and density. See pages 21–34 and 803–828 for example plots and more on these methods.

By default, the %PlotIt macro creates a graphical scatter plot on your screen. The macro will by default use the last data set created, so you must specify data= if you run %PlotIt a second time. The macro creates an output Annotate data set that cannot be used as input to the macro. If no graphics device has been previously specified (either directly or indirectly), you will be prompted for a device as follows:

```
No device name has been given--please enter device name:
```

Enter your graphics device. This name will be remembered for the duration of your SAS session or until you change the device. You can modify the gopprint= and gopplot= options to set default devices so that you will not be prompted. Note that all graphics options specified on a goptions statement (except device=) are ignored by default. Use the macro options gopprint=, gopplot=, gopts2=, and gopts= to set goptions.

To display a plot on your screen using the default goptions, specify:

```
%plotit(data=coor, datatype=corresp)
```

To create a color postscript file named myplot.ps, suitable for printing on a cljps device, specify:

```
%plotit(data=coor, datatype=corresp,
        method=print, post=myplot.ps, gopts=device=cljps)
```

Alternatively, change the default for gopprint= below to name your typical device, for example from

```
gopprint=gsfmode=replace gaccess=gsasfile,
```

to

```
gopprint=gsfmode=replace gaccess=gsasfile device=cljps,
```

Then to create a postscript file, specify:

```
%plotit(data=coor, datatype=corresp, method=print, post=myplot.ps)
```

Then the file may be previewed and printed. Another alternative is to send the plot directly to the printer. In this example, MyPrinter is a printer that prints on ordinary 8.5 × 11 paper.

```
%plotit(data=coor, datatype=corresp, gopts=device=MyPrinter)
```

To just see the printer plot, specify method=plot. Use gout= to write the plot to a catalogue.

With this release, a number of the default colors and options has changed. The new default displays a border box with a gray background outside the plot, a white background inside the plot, and black axes. Also, the color magenta is moved up ahead of cyan in the colors list since magenta displays better than cyan on a white background. If you prefer the old style, change the default of the style= option to style=a. The new default style is style=b, see page 777. If you do not like the default background color, specify gopplot=cback=*some-color* (substituting your favorite color for *some-color*) or gopplot= to use the default background. Similarly, you can change the color defaults with color=, and colors=,

as you see fit.

Sample Usage

This example performs a simple correspondence analysis. For many plots, you only the need to specify the `data=` and `datatype=` options.

```
*------Simple Correspondence Analysis------;
proc corresp all data=cars outc=coor;
   tables marital, origin;
   title 'Simple Correspondence Analysis';
   run;

%plotit(data=coor, datatype=corresp)
```

This next example performs multiple correspondence analysis.

```
*------Multiple Correspondence Analysis------;
proc corresp mca observed data=cars outc=coor;
   tables origin size type income home marital sex;
   title 'Multiple Correspondence Analysis';
   run;

%plotit(data=coor, datatype=mca)
```

This next example performs multidimensional preference analysis. The vector lengths are increased by a factor of 2.5 to make a better graphical display.

```
*------MDPREF------;
proc prinqual data=carpref out=results n=2
              replace standard scores correlations;
   id model mpg reliable ride;
   transform ide(judge1-judge25);
   title 'Multidimensional Preference (MDPREF) Analysis';
   run;

%plotit(data=results, datatype=mdpref 2.5)
```

This next example performs a preference mapping, vector model. Again, the vector lengths are increased by a factor of 2.5 to make a better graphical display.

```
*------PREFMAP, Vector Model------;
proc transreg data=results(where=(_type_ = 'SCORE'));
   model ide(mpg reliable ride)=identity(prin1 prin2);
   output tstandard=center coefficients replace out=tresult1;
   id model;
   title 'Preference Mapping (PREFMAP) Analysis - Vector';
   run;

%plotit(data=tresult1,datatype=vector 2.5)
```

This next example performs a preference mapping, ideal point model. The `antiidea=1` option is specified to handle anti-ideal points when large data values are positive or ideal.

```
*------PREFMAP, Ideal Point------;
proc transreg data=results(where=(_type_ = 'SCORE'));
   model identity(mpg reliable ride)=point(prin1 prin2);
   output tstandard=center coordinates replace out=tresult1;
   id model;
   title 'Preference Mapping (PREFMAP) Analysis - Ideal';
   run;

%plotit(data=tresult1,datatype=ideal,antiidea=1)
```

This next example performs multidimensional preference analysis. The `mdpref2` specification means MDPREF and label the vectors *too*. The vector lengths are increased by a factor of 3 to make a better graphical display. The `symlen=2` option specifies two-character symbols. The specification `vechead=`, (a null value) means no vector heads since there are labels. The `adjust1=` option is used to add full SAS DATA step statements to the preprocessed data set. This example processes `_type_ = 'CORR'` observations (those that contain vector the coordinates) the original variable names (`sub1`, `sub2`, `sub3`, ..., from the `activity` variable) and creates symbol values (1, 2, 3, ...) of size 0.7. The result is a plot with each vector labeled with a subject number.

```
*------MDPREF, labeled vector end points------;
proc prinqual cor data=recreate out=rec score std rep;
   transform identity(sub:);
   id activity active relaxing spectato;
   title 'MDPREF of Recreational Activities';
   run;

%plotit(data=rec,datatype=mdpref2 3,
        symlen=2,vechead=,adjust1=%str(
        if _type_ = 'CORR' then do;
           __symbol = substr(activity,4);
           __ssize  = 0.7;
           activity = ' ';
           end;))
```

This next example creates a contour plot, displaying density with color. The `paint=z white blue magenta red` option specifies that color interpolation is based on the variable `z`, going from white (zero density) through blue, magenta, and to red (maximum density). This color list is designed for the default style with a background of white. The option `extend=close` is used with contour plots so that the plot boundaries appear exactly at the edge of the contour data. By default, `%PlotIt` usually adds a bit of extra white space between the data and the plot boundaries which provides extra room for labels, which are not used in this example.

```
*------Bivariate Normal Density Function------;
proc iml;
   title 'Bivariate Normal Density Function';
   s = inv({1 0.25 , 0.25 1});
   m = -2.5; n = 2.5; inc = 0.05; k = 0;
   x = j((1 + (n - m) / inc) ** 2, 3, 0);
   c = sqrt(det(s)) / (2 * 3.1415);
```

```
   do i = m to n by inc;
      do j = m to n by inc;
         v = i || j; z = c * exp(-0.5 * v * s * v‘);
         k = k + 1; x[k,] = v || z;
         end;
      end;
   create x from x[colname={’x’ ’y’ ’z’}]; append from x;
   quit;
%plotit(datatype=contour, data=x, extend=close,
         paint=z white blue magenta red)
```

The goal of this next example is to create a plot of the Fisher iris data set with each observation identified by its species. Species name is centered at each point's location, and each species name is plotted in a different color. This scatter plot is overlaid on the densities used by PROC DISCRIM to classify the observations. There are three densities, one for each species. Density is portrayed by a color contour plot with white (the assumed background color) indicating a density of essentially zero. Yellow, orange, and red indicate successively increasing density.

The `data=` option names the input SAS data set. The `plotvars=` option names the y-axis and x-axis variables. The `labelvar=_blank_` option specifies that all labels are blank. This example does not use any of PROC PLOT's label collision avoidance code. It simply uses PROC PLOT to figure out how big to make the plot, and then the macro puts everything inside the plot independently of PROC PLOT, so the printer plot is blank. The `symlen=4` option specifies that the maximum length of a symbol value is 4 characters. This is because we want the first four characters of the species names as symbols. The `exttypes=symbol contour` option explicitly specifies that PROC PLOT will know nothing about the symbols or the contours. They are external types that will be added to the graphical plot by the macro after PROC PLOT has finished. The `ls=100` option specifies a constant line size. Since no label avoidance is done, there can be no collisions, and the macro will not iteratively determine the plot size. The default line size of 65 is too small for this example, whereas `ls=100` makes a better display. The `paint=` option specifies that based on values the variable `density`, colors should be interpolated ranging from white (minimum `density`) to yellow to orange to red (maximum `density`). The `rgbtypes=contour` option specifies that the `paint=` option should apply to contour type observations.

The grid (created with the loops: `do sepallen = 30 to 90 by 0.6;` and `do petallen = 0 to 80 by 0.6;`) is not square, so for optimal results the macro must be told the number of horizontal and vertical positions. The PLOTDATA DATA step creates these values and stores them in macro variables `&hnobs` and `&vnobs`, so the specification `hnobs=&hnobs, vnobs=&vnobs`, specifies the grid size. Of course these values could have been specified directly instead of through symbolic variables. The `excolors=CXFFFFFF` option is included for efficiency. The input data consist of a large grid for the contour plot. Most of the densities are essentially zero, so many of the colors will be `CXFFFFFF`, which is white, computed by `paint=`, which is the same color as the background. (See the `paint=` option, page 778 for information on CXrrggbb color specifications.) Excluding them from processing makes the macro run faster and creates smaller datasets.

This example shows how to manually do the kinds of things that the `datatype=` option does for you with standard types of data sets. The macro expects the data set to contain observations of one or more types. Each type is designated by a different value in a variable, usually named `_type_`. In this example, there are four types of observations, designated by the `_type_` variable's four values, 1, 2, 3, 4, which are specified in the `types=` option. The `symtype=` option specifies the symbol types for these four observation types. The first three types of observations are `symbol` and the last type, `_type_`

= 4, designates the contour observations. The first three symbols are the species names (symbols= values) printed in symfont=swiss font. The last symbol is null because contours do not use symbols. The first three symbols, since they are words as opposed to a single character, are given a small size (symsize=0.7). A value of 1 is specified for the symbol size for contour type observations. The macro determines the optimal size for each color rectangle of the contour plot. Constant colors are only specified for the noncontour observations since a variable color is computed for contour observations.

```
*------Discriminant Analysis------;
data plotdata;  * Create a grid over which DISCRIM outputs densities.;
   do SepalLength = 30 to 90 by 0.6;
      h + 1; * Number of horizontal cells;
      do PetalLength = 0 to 80 by 0.6;
         n + 1; * Total number of cells;
         output;
         end;
      end;
   call symput('hnobs', compress(put(h    , best12.))); * H grid size;
   call symput('vnobs', compress(put(n / h, best12.))); * V grid size;
   drop n h;
   run;
proc discrim data=iris testdata=plotdata testoutd=plotd
             method=normal pool=no short noclassify;
   class species;
   var PetalLength SepalLength;
   title 'Discriminant Analysis of Fisher (1936) Iris Data';
   title2 'Using Normal Density Estimates with POOL=NO';
   run;
data all;
   * Set the density observations first so the scatter plot points
     will be on top of the contour plot.  Otherwise the contour plot
     points will hide the scatter plot points.;
   set plotd iris(in=iris);
   if iris then do;
      _type_ = species; * unformatted species number 1, 2, 3;
      output;
      end;
   else do;
      _type_ = 4; * density observations;
      density = max(setosa,versicolor,virginica);
      output;
      end;
   run;
```

```
%plotit(data=all,plotvars=PetalLength SepalLength,labelvar=_blank_,
        symlen=4,exttypes=symbol contour,ls=100,
        paint=density white yellow orange red,rgbtypes=contour,
        hnobs=&hnobs,vnobs=&vnobs,excolors=CXFFFFFF,
        types  =1      2          3          4,
        symtype=symbol symbol     symbol     contour,
        symbols=Set    Vers       Virg       '',
        symsize=0.7    0.7        0.7        1,
        symfont=swiss  swiss      swiss      solid,
        colors =blue   magenta    green
        )
```

How %PlotIt Works

You create a data set either with a DATA step or with a procedure. Then you run the macro to create a graphical scatter plot. This macro is not a SAS/GRAPH procedure and does not behave like a typical SAS/GRAPH procedure. The `%PlotIt` macro performs the following steps.

1. The `%PlotIt` macro reads an input data set and preprocesses it. The preprocessed data set contains information such as the axis variables, the point-symbol and point-label variables, and symbol and label types, sizes, fonts, and colors. The nature of the preprocessing depends on the type of data analysis that generated the input data set. For example, if the option `datatype=mdpref` was specified with an input data set created by PROC PRINQUAL for a multidimensional preference analysis, then the `%PlotIt` macro creates blue points for `_type_ = 'SCORE'` observations and red vectors for `_type_ = 'CORR'` observations.

2. A DATA step, using the DATA Step Graphics Interface, determines how big to make the graphical plot.

3. PROC PLOT determines where to position the point labels. By default, if some of the point label characters are hidden, the `%PlotIt` macro recreates the printer plot with a larger line and page size, and hence creates more cells and more room for the labels. Note that when there are no point labels, the printer plot may be empty. All of the information that is in the graphical scatter plot may be stored in the `extraobs=` data set. All results from PROC PLOT are written to data sets with ODS. The macro will clear existing `ods select` and `ods exclude` statements.

4. The printer plot is read and information from it, the preprocessed data set, and the extra observations data set are combined to create an Annotate data set. The label position information is read from the PROC PLOT output, and all of the symbol, size, font, and color information is extracted from the preprocessed (or extra observations) data set. The Annotate data set contains all of the instructions for drawing the axes, ticks, tick marks, titles, point symbols, point labels, axis labels, and so on. Circles can be drawn around certain locations, and vectors can be drawn from the origin to other locations.

5. The Annotate data set is displayed with the GANNO procedure. The `%PlotIt` macro does not use PROC GPLOT.

Debugging

When you have problems, try `debug=vars` to see what the macro thinks you specified. It is also helpful to specify: `debug=mprint notes`. You can also print the final Annotate data set and the preprocessing data set:

```
options ls=180;
proc print data=anno uniform;
   format text $20. comment $40.;
   run;

proc print data=preproc uniform;
   run;
```

Advanced Processing

You can post-process the Annotate DATA step to change colors, fonts, undesirable placements, and so on. Sometimes, this can be done with the `adjust4=` option. Alternatively, when you specify `method=none`, you create an Annotate data set without displaying it. The data set name is by default WORK.ANNO. You can then manipulate it further with a DATA step or PROC FSEDIT to change colors, fonts, or sizes for some labels; move some labels; and so on. If the final result is a new data set called ANNO2, display it by running:

```
proc ganno annotate=anno2;
   run;
```

Notes

With `method=print`, the macro creates a file. See the `filepref=` and `post=` options and make sure that the file name does not conflict with existing names.

This macro creates variable names that begin with two underscores and assumes that these names will not conflict with any input data set variable names.

It is not feasible with a macro to provide the full range of error checking that is provided with a procedure. Extensive error checking is provided, but not all errors will be diagnosed.

Not all options will work with all other options. Some combinations of options may produce macro errors or Annotate errors.

This macro may not be fully portable. When you switch operating systems or graphics devices, some changes may be necessary to get the macro to run successfully again.

Graphics device differences may also be a problem. We do not know of any portability problems, but the macro has not been tested on all supported devices.

This macro tries to create a plot with equated axes, where a centimeter on one axis represents the same data range as a centimeter on the other axis. The only way to accomplish this is by explicitly and jointly controlling the `hsize=`, `vsize=`, `hpos=`, and `vpos= goptions`. By default, the macro tries to ensure that all of the values work for the specific device. See `makefit=`, `xmax=`, and `ymax=`. By default the macro uses GASK to determine `xmax` and `ymax`. If you change any of these options, your axes may not be equated. Axes are equated when `vsize` × `hpos` / `hsize` × `hpos` = `vtoh`.

When you are plotting variables that have very different scales, you may need to specify appropriate tick increments for both axes to get a reasonable plot. Here is an example: `plotopts=haxis=by 20 vaxis=by 5000`. Alternatively, just specifying the smaller increment is often sufficient: `plotopts=haxis=by 20`. Alternatively, specify `vtoh=`, (null value) to get a plot like PROC GPLOT's, with the window filled.

By default, the macro iteratively creates and recreates the plot, increasing the line size and the flexibility in the `placement=` list until there are no penalties.

The SAS system option `ovp` (overprint) is not supported by this macro.

%PlotIt Macro Options

The following options can be used with the `%PlotIt` macro.

Option	Description
`adjust1=`*SAS-statements*	adjust the preprocessing data set
`adjust2=`*SAS-statements*	includes statements with PROC PLOT
`adjust3=`*SAS-statements*	extra statements for the final DATA step
`adjust4=`*SAS-statements*	extra statements for the final DATA step
`adjust5=`*SAS-statements*	extra statements for the final DATA step
`antiidea=`*n*	eliminates PREFMAP anti-ideal points
`blue=`*expression*	blue part of RGB colors
`bright=`*n*	generates random label colors
`britypes=`*type*	types to which `bright=` applies
`cframe=`*color*	color of background within the frame
`cirsegs=`*n*	circle smoothness parameter
`color=`*color*	default color
`colors=`*colors-list*	default label and symbol color list
`cursegs=`*n*	number of segments in a curve
`curvecol=`*color*	color of curve
`data=`*SAS-data-set*	input data set
`datatype=`*data-type*	data analysis that generated data set
`debug=`*values*	debugging output
`excolors=`*color-list*	excludes from the Annotate data set
`extend=`*axis-extensions*	extend the x and y axes
`extraobs=`*SAS-data-set*	extra observations data set
`exttypes=`*type*	types for `extraobs=` data set
`filepref=`*prefix*	file name prefix
`font=`*font*	default font
`framecol=`*color*	color of frame
`gdesc=`*description*	catalog description
`gname=`*name*	catalog entry
`gopplot=`*goptions*	`goptions` for plotting to screen
`gopprint=`*goptions*	`goptions` for printing
`gopts2=`*goptions*	`goptions` that are always used
`gopts=`*goptions*	additional `goptions`
`gout=`*catalog*	`proc anno gout=` catalog
`green=`*expression*	green part of RGB colors
`hminor=`*n* | *do-list*	horizontal axis minor tick marks
`hnobs=`*n*	horizontal observations for contour plots
`hpos=`*n*	horizontal positions in graphics area
`href=`*do-list*	horizontal reference lines
`hsize=`*n*	horizontal graphics area size
`inc=`*n*	`haxis=by inc`, `vaxis=by inc`
`interpol=`*method*	axis interpolation method
`labcol=`*label-colors*	colors for the point labels
`label=`*label-statement*	`label` statement

`labelcol=`*color*	color of variable labels
`labelvar=`*label-variable*	point label variable
`labfont=`*label-fonts*	fonts for the point labels
`labsize=`*label-sizes*	sizes for the point labels
`ls=`*n*	how line sizes are generated
`lsinc=`*n*	increment to line size
`lsizes=`*number-list*	line sizes (thicknesses)
`makefit=`*n*	proportion of graphics window to use
`maxiter=`*n*	maximum number of iterations
`maxokpen=`*n*	maximum acceptable penalty sum
`method=`*value*	where to send the plot
`monochro=`*color*	overrides all other colors
`nknots=`*n*	number of knots option
`offset=`*n*	move symbols for coincident points
`options=`*options-list*	binary options
`out=`*SAS-data-set*	output Annotate data set
`outward=none` | *'c'*	PLOT statement `outward=`
`paint=`*interpolation*	color interpolation
`place=`*placement*	generates a `placement=` option
`plotopts=`*options*	PLOT statement options
`plotvars=`*variable-list*	*y*-axis and *x*-axis variables
`post=`*filename*	graphics stream file name
`preproc=`*SAS-data-set*	preprocessed `data=` data set
`procopts=`*options*	PROC PLOT statement options
`ps=`*n*	page size
`radii=`*do-list*	radii of circles
`red=`*expression*	red part of RGB colors
`regdat=`*SAS-data-set*	intermediate regression results data set
`regopts=`*options*	regression curve fitting options
`regprint=`*regression-option*	regression options
`rgbround=`*RGB-rounding*	`paint=` rounding factors
`rgbtypes=`*type*	types to which RGB options apply
`style=`*style*	controls several colors and options
`symbols=`*symbol-list*	plotting symbols
`symcol=`*symbol-colors*	colors of the symbols
`symfont=`*symbol fonts*	symbol fonts
`symlen=`*n*	length of the symbols
`symsize=`*symbol-sizes*	sizes of symbols
`symtype=`*symbol-types*	types of symbols
`symvar=`*symbol-variable*	plotting symbol variable
`tempdat1=`*SAS-data-set*	intermediate results data set
`tempdat2=`*SAS-data-set*	intermediate results data set
`tempdat3=`*SAS-data-set*	intermediate results data set
`tempdat4=`*SAS-data-set*	intermediate results data set
`tempdat5=`*SAS-data-set*	intermediate results data set
`tempdat6=`*SAS-data-set*	intermediate results data set
`tickaxes=`*axis-string*	axes to draw tick marks
`tickcol=`*color*	color of ticks
`tickfor=`*format*	tick format used by `interpol=tick`

`ticklen=`*n*	length of tick mark in horizontal cells
`titlecol=`*color*	color of title
`tsize=`*n*	default text size
`types=`*observation-types*	observations types
`typevar=`*variable*	observation types variable
`unit=in` | `cm`	`hsize=` and `vsize=` unit
`vechead=`*vector-head-size*	how to draw vector heads
`vminor=`*n* | *do-list*	vertical axis minor tick marks
`vnobs=`*n*	vertical observations for contour plots
`vpos=`*n*	vertical positions in graphics area
`vref=`*do-list*	vertical reference lines
`vsize=`*n*	vertical graphics area size
`vtoh=`*n*	PROC PLOT `vtoh=` option
`xmax=`*n*	maximum horizontal graphics area size
`ymax=`*n*	maximum vertical graphics area size

Note that for many analyses, the only options you need to specify are `data=`, `datatype=`, and sometimes `method=`. To specify variables to plot, specify `plotvars=`, `labelvar=`, and `symvar=`.

Overriding Options

This macro looks for a special global macro variable named `plotitop`. If it exists, its values are used to override the macro options. Say you have a series of calls to the plotting macro and you want to route them all to a postscript file, you can specify this once:

```
%let plotitop = gopts=gsfmode=append gaccess=gsasfile device=qmscolor;
```

and then run the macro repeatedly without change. The value of the `plotitop` macro variable must consist of a name, followed by an equal sign, followed by a value. Optionally, it may continue with a comma, followed by a `name=`*value*, and so on, just like the way options are specified with the macro. Option values must not contain commas. Here is another example:

```
%let plotitop = color=black, gopts=cback=cyan;
```

Destination and GOPTIONS

The options in this section specify the plot destination and SAS `goptions`. Note that with the `%PlotIt` macro, you do not specify a `goptions` statement. If you do, it will be overridden. All `goptions` (except `device=`) are specified with macro options. If you would prefer to specify your own `goptions` statement and have the macro use it, just specify or change the default for these four options to null: `gopplot=, gopprint=, gopts2=, gopts=`. If you use a locally installed copy of the macro, you can modify the `gopprint=` and `gopplot=` options defaults to include the devices that you typically use. Otherwise, the macro checks the `goptions` to get a device.

gopplot= *goptions*
specifies the `goptions` for directly plotting on the screen. There are no default goptions for `gopplot=`.

gopprint= *goptions*
specifies the `goptions` for printing (creating a graphics stream file). The default is

`gopprint=gsfmode=replace gaccess=gsasfile`.

Here is an example of how you might modify the defaults for `gopprint=` and `gopplot=` option defaults to set default devices.

```
gopprint=gsfmode=replace gaccess=gsasfile device=qmscolor,
gopplot=cback=black device=win,
```

gopts= *goptions*

provides a way to specify additional `goptions` that are always used. There are no default goptions for `gopts=`. For example, to rotate to a landscape orientation with a black background color, specify `gopts=rotate cback=black`.

gopts2= *goptions*

specifies the `goptions` that are always used, no matter which `method=` is specified. The default is `gopts2=reset=goptions erase`.

method= gplot | plot | print | none

specifies where to send the plot. The default is `method=gplot`.

`gplot` - displays a graphical scatter plot on your screen using the `goptions` from `gopplot=`. The `gopplot=` option should contain the `goptions` that only apply to plots displayed on the screen.

`plot` - creates a printer plot only.

`print` - routes the plot to a graphics stream file, such as a postscript file, using the `goptions` from `gopprint=`. The `gopprint=` option should contain the `goptions` that only apply to hard-copy plots. Specify the file name with `post=`.

`none` - just creates the Annotate data set and sets up `goptions` using `gopplot=`.

Data Set and Catalog Options

These options specify the input SAS data set, output Annotate data set, and options for writing plots to files and catalogs.

data= *SAS-data-set*

specifies the input data set. The default input data set is the last data set created. You should always specify the `data=` option since the macro creates data sets that are not suitable for use as input.

filepref= *file-name-prefix*

specifies the file name prefix. The default is `filepref=sasplot`.

gdesc= *description*

specifies the name of a catalog description. This option can optionally be used with `proc anno gout=` to provide the `description=`.

gname= *name*
specifies the name of a catalog entry. This option can optionally be used with `proc anno gout=` to provide the `name=`.

gout= *catalog*
specifies the `proc anno gout=` catalog. With `gout=gc.slides`, first specify: `libname gc '.'`; Then to replay, run: `proc greplay igout=gc.slides; run;` Note that replayed plots will not in general have the correct aspect ratio.

out= *SAS-data-set*
specifies the output Annotate data set. This data set contains all of the instructions for drawing the graph. The default is `out=anno`.

post= *filename*
specifies the graphics stream file name. The default name is constructed from the `filepref=` value and '`ps`' in a host-specific way.

Typical Options

These are some of the most frequently used options.

datatype= *data-type*
specifies the type of data analysis that generated the data set. This option is used to set defaults for other options and to do some preprocessing of the data.

When the data type is `corresp`, `mds`, `mca`, `row`, `column`, `mdpref`, `mdpref2`, `vector`, or `ideal`, the `label=typical` option is implied when `label=` is not otherwise specified. The default point label variable is the last character variable in the data set.

Some data types (`mdpref`, `vector`, `ideal`, `corresp`, `row`, `mca`, `column`, `mds`) expect certain observation types and set the `types=` list accordingly. For example, `mdpref` expects `_type_ = 'SCORE'` and `_type_ = 'CORR'` observations. The remaining data types do not expect any specific value of the `typevar=` variable. So if you do not get the right data types associated with the right observation types, specify `types=`, and specify the `types=` values in an order that corresponds to the order of the symbol types in the `Types Legend` table. Unlike `symtype=`, the order in which you specify `datatype=` values is irrelevant.

A null value (`datatype=`, the default) specifies no special processing, and the default plotting variables are the first two numeric variables in the data set. Specifying `corresp`, `mds`, `mca`, `row`, or `column` will set the default `plotvars` to `dim2` and `dim1`. Otherwise, when a nonnull value is specified, the default `plotvars` are `prin2` and `prin1`.

Here are the various data types.

`datatype=column`
specifies a `proc corresp profile=column` analysis. Row points are plotted as vectors.

`datatype=contour`
draws solid color contour plots. When the number of row points is not the same as the number of column points in the grid, use `hnobs=` and `vnobs=` to specify the number of points. This method creates an `hnobs=` by `vnobs=` grid of colored rectangles. Each of the rectangles should touch all adjacent rectangles. This method works well with a regular grid of points. The `method=square` option is a good alternative when the data do not fall in a regular grid.

`datatype=corresp`
specifies an ordinary correspondence analysis.

`datatype=curve`
fits a curve through the scatter plot.

`datatype=curve2`
fits a curve through the scatter plot and tries to make the labels avoid overprinting the curve.

`datatype=function`
draws functions. Typically, no labels or symbols are drawn. This option has a similar effect to the PROC GPLOT `symbol` statement options `i=join v=none`.

`datatype=ideal`
specifies a PREFMAP ideal point model. See the `antiidea=`, `radii=`, and `cirsegs=` options.

`datatype=mca`
specifies a multiple correspondence analysis.

`datatype=mdpref`
specifies multidimensional preference analysis with vectors with blank labels. Note that `datatype=mdpref` can also be used for ordinary principal component analysis.

`datatype=mdpref2`
specifies MDPREF with vector labels (MDPREF and labels *too*).

`datatype=mds`
specifies multidimensional scaling.

`datatype=mds ideal`
specifies PREFMAP ideal point after the MDS.

`datatype=mds vector`
specifies PREFMAP after MDS.

`datatype=row`
specifies a `proc corresp profile=row` analysis. Column points are plotted as vectors.

`datatype=square`
plots each point as a solid square. The `datatype=square` option is useful as a form of contour plotting when the data do not form a regular grid. The `datatype=square` option, unlike `datatype=contour`, does not try to scale the size of the square so that each square will touch another square.

`datatype=symbol`
specifies an ordinary scatter plot.

`datatype=vector`
specifies a PREFMAP vector model.

`datatype=vector ideal`
specifies both PREFMAP vectors and ideal points.

For some `datatype=` values, a number may be specified after the name. This is primarily useful for biplot data sets produced by PROC PRINQUAL and PREFMAP data sets produced by PROC TRANSREG. This number specifies that the lengths of vectors should be changed by this amount. The number must be specified last. Examples: `datatype=mdpref 2`, `datatype=mds vector 1.5`.

The primary purpose of the `datatype=` option is to provide an easy mechanism for specifying defaults for the options in the next section (`typevar=` through `outward=`).

labelvar= *label-variable* | _blank_

specifies the variable that contains the point labels. The default is the last character variable in the data set. If `labelvar=_blank_` is specified, the macro will create a blank label variable.

options= *options-list*

specifies binary options. Specify zero, one, or more in any order. For example: `options=nocenter nolegend`.

`border`
draws a border box around the outside of the graphics area, like the border `goption`. This is the default with `style=b` unless `options=noborder` is specified.

`close`
if a border is being drawn, perform the same adjustments on the border that are performed on the axes. This option is most useful with contour plots.

`diag`
draws a diagonal reference line.

`expand`
specifies Annotate data set post processing, typically for use with `extend=close` and contour plots. This option makes the plot bigger to fill up more of the window.

`noborder`
specifies that `%PlotIt` should not add a border to the plot with `style=b`.

`noback`
specifies that `%PlotIt` should not set the frame color (the background color within the plot boundary) to white with `style=b`.

`nocenter`
do not center. By default, when `nocenter` is not specified, `vsize=` and `hsize=` are set to their maximum values, and the `vpos=` and `hpos=` values are increased accordingly. The x and y coordinates are increased to position the plot in the center of the full graphics area.

`noclip`
do not clip. By default, when `noclip` is not specified, labels that extend past the edges of the plot are clipped. This option will not absolutely prevent labels from extending beyond the plot, particularly when sizes are greater than 1.

`nocode`
suppresses the printing of the PROC PLOT and `goptions` statements.

`nodelete`
do not delete intermediate data sets.

`nohistory`
suppresses the printing of the iteration history table.

`nolegend`
suppresses the printing of the legends.

`noprint`
equivalent to `nolegend`, `nocode`, and `nohistory`.

`square`
uses the same ticks for both axes and tries to make the plot square by tinkering with the `extend=` option. Otherwise, ticks may be different.

`textline`
put text in the data set, followed by lines, so lines overwrite text. Otherwise text overwrites lines.

plotvars= *two-variable-names*

specifies the *y*-axis variable then the *x*-axis variable. To plot `dim2` and `dim3`, specify `plotvars=dim2 dim3`. The `datatype=` option controls the default variable list.

symlen= *n*

specifies the length of the symbols. By default, symbols are single characters, but the macro can center longer strings at the symbol location.

symvar= *symbol-variable* | `_symbol_`

specifies the variable that contains the plotting symbol for input to PROC PLOT. When `_symbol_` is specified, which is the default, the symbol variable is created, typically from the `symbols=` list, which may be constructed inside the macro. (Note that the variable `__symbol` is created to contain the symbol for the graphical scatter plot. The variables `_symbol_` and `__symbol` may or may not contain the same values.) Variables can be specified, and the first `symlen=` characters are used for the symbol. When a null value (`symvar=`) or a constant value is specified, the symbol from the printer plot will be used (which is always length one, no matter what is specified for `symlen=`). To get PROC PLOT pointer symbols, specify `symvar='00'x`, (hex null constant). To center labels with no symbols, specify: `symvar=, place=0`.

Observation-Type List Options

Data sets for plotting can have different types of observations that are plotted differently. These options allow you to specify the types of observations, the variable that contains the observation types, and the different ways the different types should be plotted. For many types of analyses, these can all be handled easily with the `datatype=` option, which sets analysis-specific defaults for the list options. When you can, you should use `datatype=` instead of the list options. If you do use the list options, specify a variable, in `typevar=`, whose values distinguish the observation types. Specify the list of valid types in `types=`. Then specify colors, fonts, sizes, and so on for the various observation types. Alternatively, you can use these options with `datatype=`. Specify lists for just those label or symbol characteristics you want to change, for example colors, fonts or sizes.

The lists do not all have to have the same number of elements. The number of elements in `types=` determines how many of the other list elements are used. When an observation type does not match one of the `type=` values, the results are the same as if the first type were matched. If one of the other lists is shorter than the `types=` list, the shorter list is extended by adding copies of the last element to the end. Individual list values may be quoted, but quotes are not required. *Embedded blanks are not permitted. If you embed blanks, you will not get the right results.* Values of the `typevar=` variable are compressed before they are used, so for example, an _`type`_ value of `'M COEFFI'` must be specified as `'MCOEFFI'`.

britypes= *type*

specifies the types to which `bright=` applies. The default is `britypes=symbol`.

colors= *colors-list*

specifies the default color list for the `symcol=` and `labcol=` options. The default is `colors=blue red green magenta cyan orange gold lilac olive purple brown gray rose violet salmon yellow`. With the original color style, `style=a`, the order of magenta and cyan are switched in the list. With the default style of style=`b` magenta comes before cyan.

exttypes= *type*

specifies the types to always put in the `extraobs=` data set when they have blank labels. The default is `exttypes=vector`.

labcol= *label-colors*

specifies the colors for the point labels. The default list is constructed from the `colors=` option. Examples:

```
labcol='red'
labcol='red' 'white' 'blue'
```

labfont= *label-fonts*

specifies the fonts for the point labels. Examples:

```
labfont='swiss'
labfont='swiss' 'swissi'
```

labsize= *label-sizes*
specifies the sizes for the point labels. Examples:

```
labsize=1
labsize=1 1.5
labsize=1 0
```

rgbtypes= *type*
specifies the types to which `paint=`, `red=`, `green=`, and `blue=` apply. The default is `rgbtypes=symbol`.

symbols= *symbol-list*
specifies the plotting symbols. Symbols may be more than a single character. You must specify `symlen=n` for longer lengths. Blank symbols must be specified as `''` with no embedded blanks. Examples:

```
symbols='*'
symbols='**'
symbols='*' '+' '*' ''
symbols='NC' 'OH' 'NJ' 'NY'
```

symcol= *symbol-colors*
specifies the colors of the symbols. The default list is constructed from the `colors=` option. Examples:

```
symcol='red'
symcol='red' 'white' 'blue'
```

symtype= *symbol-types*
specifies the types of symbols. Valid values are `symbol`, `vector`, `circle`, `contour`, and `square`. Examples:

```
symtype='symbol'
symtype='symbol' 'vector'
symtype='symbol' 'circle'
```

symfont= *symbol fonts*
specifies the symbol fonts. The font is ignored for vectors with no symbols. Examples:

```
symfont='swiss'
symfont='swiss' 'swissi'
```

symsize= *symbol-sizes*
specifies the sizes of symbols. Examples:

```
symsize=1
symsize=1 1.5
```

types= *observation-types*
specifies the observations types. Observation types are usually values of a variable like _type_. Embedded blanks are not permitted. Examples:

```
types='SCORE'
types='OBS' 'SUPOBS' 'VAR' 'SUPVAR'
types='SCORE'
types='SCORE' 'MCOEFFI'
```

The order in which values are specified for the other options depends on the order of the types. The default types for various datatype= values are given next:

```
corresp:  'VAR' 'OBS' 'SUPVAR' 'SUPOBS'
row:      'VAR' 'OBS' 'SUPVAR' 'SUPOBS'
mca:      'VAR' 'OBS' 'SUPVAR' 'SUPOBS'
column:   'VAR' 'OBS' 'SUPVAR' 'SUPOBS'
mdpref:   'SCORE' 'CORR'
vector:   'SCORE' 'MCOEFFI'
ideal:    'SCORE' 'MPOINT'
mds:      'SCORE' 'CONFIG'
```

For combinations of options, these lists are combined in order, but without repeating 'SCORE', for example with datatype=mdpref vector ideal, the default types= list is: 'SCORE' 'CORR' 'MCOEFFI' 'MPOINT'.

typevar= *variable*
specifies a variable that is looked at for the observation types. By default, this will be _type_ if it is in the input data set.

Internal Data Set Options

The macro creates one or more of these data sets internally to store intermediate results.

extraobs= *SAS-data-set*
specifies a data set used to contain the extra observations that do not go through PROC PLOT. The default is extraobs=extraobs.

preproc= *SAS-data-set*
specifies a data set used to contain the preprocessed data= data set. The default is preproc=preproc.

regdat= *SAS-data-set*
specifies a data set used to contain intermediate regression results for curve fitting. The default is regdat=regdat.

tempdat1= *SAS-data-set*
specifies a data set used to hold intermediate results. The default is `tempdat1=tempdat1`.

tempdat2= *SAS-data-set*
specifies a data set used to hold intermediate results. The default is `tempdat2=tempdat2`.

tempdat3= *SAS-data-set*
specifies a data set used to hold intermediate results. The default is `tempdat3=tempdat3`.

tempdat4= *SAS-data-set*
specifies a data set used to hold intermediate results. The default is `tempdat4=tempdat4`.

tempdat5= *SAS-data-set*
specifies a data set used to hold intermediate results. The default is `tempdat5=tempdat5`.

tempdat6= *SAS-data-set*
specifies a data set used to hold intermediate results. The default is `tempdat6=tempdat6`.

Miscellaneous Options

Here are some options that are sometimes needed for certain situations to control the details of the plots.

antiidea= *n*
eliminates PREFMAP anti-ideal points. The TRANSREG ideal-point model assumes that small attribute ratings mean that the object is similar to the attribute and large ratings imply dissimilarity to the attribute. For example, if the objects are food and the attribute is "sweetness," then the analysis assumes that 1 means sweet and 9 is much less sweet. The resulting coordinates are usually ideal points, representing an ideal amount of the attribute, but sometimes they are anti-ideal points and need to be converted to ideal points. This option is used to specify the nature of the data (small ratings mean similar or dissimilar) and to request automatic conversion of anti-ideal points.

null value - (`antiidea=`, the default) - do nothing.

1 - reverses in observations whose `_TYPE_` contains 'POINT' when `_issq_` > 0. Specify `antiidea=1` with `datatype=ideal` for the unusual case when large data values are positive or ideal.

-1 - reverses in observations whose `_type_` contains 'POINT' when `_issq_` < 0. Specify `antiidea=-1` with `datatype=ideal` for the typical case when small data values are positive or ideal.

extend= *axis-extensions*
is used to extend the x and y axes beyond their default lengths. Specify four values, for the left, right, top, and bottom axes. If the word `close` is specified somewhere in the string, then macro moves the axes in close to the extreme data values, and the computed values are added to the specified values (if any). Sample specifications: `extend=2 2`, or `extend=3 3 -0.5 0.5`. Specifying a positive value n

extends the axis n positions in the indicated direction. Specifying a negative value shrinks the axis. The defaults are in the range -2 to 2, and are chosen in an attempt to add a little extra horizontal space and make equal the extra space next to each of the four extreme ticks. When there is enough space, the horizontal axis is slightly extended by default to decrease the chance of a label slightly extending outside the plot. PROC PLOT usually puts one or two more lines on the top of the plot than in the bottom. The macro tries to eliminate this discrepancy. This option does not add any tick marks; it just extends or shrinks the ends of the axis lines. So typically, only small values should be specified. Be careful with this option and a positive `makefit=` value.

font= *font*
specifies the default font. The default is `font=swiss`.

hminor= *n* | *do-list*
specifies the number of horizontal axis minor tick marks between major tick marks. A typical value is 9. The number cannot be specified when `haxis=` is specified with `plotopts=`. Alternatively, specify a DATA step `do` list. Note that with log scaling, specify $\log_{10}$'s of the data values. For example, specify `hminor=0.25 to 5 by 0.25`, with data ranging up to 10**5.

href= *do-list*
specifies horizontal-axis reference lines (which are drawn vertically). Specify a DATA step `do` list. By default, there are no reference lines.

inc= *n*
specifies `haxis=by inc` and `vaxis=by inc` values. The specified increments apply to both axes. To individually control the increments, you must specify the PLOT statement `haxis=` and `vaxis=` options on the `plotopts=` option. When you are plotting variables that have very different scales, you may need to independently specify appropriate tick increments for both axes to get a reasonable plot. Here is an example: `plotopts=haxis=by 20 vaxis=by 5000`.

interpol= `ls` | `tick` | `no` | `hlog` | `vlog` | `yes`
specifies the axis interpolation method.

`ls` - uses the least-squares method only. This method computes the mapping between data and positions using ordinary least-squares linear regression. Usually, you should not specify `interpol=ls` because slight inaccuracies may result, producing aesthetically unappealing plots.

`hlog` - specifies that the x-axis is on a log scale.

`no` - does not interpolate.

`tick` - uses the tick mark method. This method computes the slope and intercept using tick marks and their values. Tick marks are read using the `tickfor=` format.

`vlog` - specifies that the y-axis is on a log scale.

`yes` - the default, interpolates symbol locations, starting with least squares but replacing them with tick-based estimates when they are available.

This option makes the symbols, vectors, and circles map to the location they would in a true graphical scatter plot, not the cell locations from PROC PLOT. This option has no effect on labels, the frame, reference lines, titles, or ticks. With `interpol=no`, plots tend to look nicer whereas `interpol=yes` plots are slightly more accurate. Note that the strategy used to interpolate can be defeated in certain cases. If the horizontal axis tick values print vertically, specify `interpol=ls`. The `hlog` and `vlog` values are specified in addition to the method. For example, `interpol=yes vlog hlog`.

label= *label-statement*
specifies a `label` statement. Note that specifying the keyword `label` to begin the statement is optional. You can specify `label=typical` to request a label statement constructed with `'Dimension'` and the numeric suffix of the variable name, for example, `label dim1 = 'Dimension 1' dim2 = 'Dimension 2';` when `plotvars=dim2 dim1`. The `label=typical` option can only be used with variable names that consist of a prefix and a numeric suffix.

ls= *n* | *iterative-specification*
specifies how line sizes are generated. The default is `ls=compute search`. When the second word is `search`, the macro searches for an optimal line size. See the `place=` option for more information on searches. When the first word is `compute`, the line size is computed from the iteration number so that the line sizes are: 65 80 100 125 150 175 200. Otherwise the first word is the first linesize and with each iteration the linesize is incremented by the `lsinc=` amount. Example: `ls=65 search`.

lsinc= *n*
specifies the increment to line size in iterations when line size is not computed. The default is `lsinc=15`.

lsizes= *number-list*
specifies the line sizes (thicknesses) for frame, ticks, vectors, circles, curves, respectively. The default is `lsizes=1 1 1 1 1`.

maxiter= *n*
specifies the maximum number of iterations. The default is `maxiter=15`.

maxokpen= *n*
specifies the maximum acceptable penalty sum. The default is `maxokpen=0`. Penalties accrue when label characters collide, labels get moved too far from their symbols, or words get split.

offset= *n*
move symbols for coincident points `offset=` spaces up/down and left/right. This helps to better display coincident symbols. Specify a null value (`offset=,`) to turn off offsetting. The default is `offset=0.25`.

place= *placement-specification*
generates a `placement=` option for the plot request. The default is `place=2 search`. Specify a non-negative integer. Values greater than 13 are set to 13. As the value gets larger, the procedure is given more freedom to move the labels farther from the symbols. The generated placement list will be printed in the log. You can still specify `placement=` directly on the `plotopts=` option. This option just gives

you a shorthand notation. For example:

```
place=0 - placement=((s=center))
place=1 - placement=((h=2 -2 : s=right left)
                     (v=1 * h=0 -1 to -2 by alt))
place=2 - placement=((h=2 -2 : s=right left)
                     (v=1 -1 * h=0 -1 to -5 by alt))
place=3 - placement=((h=2 -2 : s=right left)
                     (v=1 to 2 by alt * h=0 -1 to -10 by alt))
place=4 - placement=((h=2 -2 : s=right left)
                     (v=1 to 2 by alt * h=0 -1 to -10 by alt)
                     (s=center right left * v=0 1 to 2 by alt *
                      h=0 -1 to -6 by alt * l= 1 to 2))
```

and so on.

The `place=` option, along with the `ls=` option can be used to search for an optimal placement list and an optimal line size. By default, the macro will create and recreate the plot until it avoids all collisions. The search is turned off when a `placement=` option is detected in the plot request or plot options.

If search is not specified with `place=` or `ls=`, the specified value is fixed. If search is specified with the other option, only that option's value is incremented in the search.

plotopts= *options*

specifies PLOT statement options. The `box` option will be specified, even if you do not specify it. Reference lines should not be specified using the PROC PLOT `href=` and `vref=` options. Instead, they should be specified directly using the `href=` and `vref=` macro options. By default, no PLOT statement options are specified except `box`.

procopts= *options*

specifies PROC PLOT statement options. The default is `procopts=nolegend`.

tickaxes= *axis-string*

specifies the axes to draw tick marks. The default, `tickaxes=`LRTBFlb, means major ticks on left (L), right (R), top (T), and bottom (B), and the full frame (F) is to be drawn, and potentially minor tick marks on the left (l) and bottom (b). Minor ticks on the right (r) and top (t) can also be requested. To just have major tick marks on the left and bottom axes, and no full frame, specify `tickaxes=LB`. Order and spacing do not matter. `hminor=` and `vminor=` must also be specified to get minor ticks.

tickfor= *format*

specifies the tick format used by `interpol=tick`. You should change this if the tick values in the PROC PLOT output cannot be read with the default `tickfor=32.` format. For example, specify `tickfor=date7.` with dates.

ticklen= *n*
specifies the length of tick marks in horizontal cells. A negative value can be specified to indicate that only half ticks should be used, which means the ticks run to but not across the axes. The default is `ticklen=1.5`.

tsize= *n*
specifies the default text size. The default is `tsize=1`.

vminor= *n* | *do-list*
specifies the number of vertical axis minor tick marks between major tick marks. A typical value is 9. The number cannot be specified when `vaxis=` is specified with `plotopts=`. Alternatively, specify a DATA step `do` list. Note that with log scaling, specify $\log_{10}$'s of the data values. For example, specify `vminor=0.25 to 5 by 0.25`, with data ranging up to 10**5.

vref= *do-list*
specifies vertical reference lines (which are drawn horizontally). Specify a DATA step `do` list. By default, there are no reference lines.

Color Options

The symbol and point label colors are set by the `labcol=` and `symcol=` options. Here are the other color options.

bright= *n*
generates random label colors for `britypes=` values. In congested plots, it may be easier to see which labels and symbols go together if each label/symbol pair has a different random color. Colors are computed so that the mean RGB (red, green, blue) components equal the specified `bright=` value. The valid range is $5 \leq bright \leq 250$. 128 is a good value. Small values will produce essentially black labels and large values will produce essentially white labels, and so should be avoided. The default is a null value, `bright=`, and there are no random label colors. If you get a color table full error message, you need to specify larger values for the `rgbround=` option.

cframe= *color*
specifies the color of the background within the frame. This is analogous to the `cframe=` SAS/GRAPH option. With style=`b`, the default is `cframe=white` unless `options=noback` is specified.

color= *color*
specifies the default color that is used when no other color is set. The default color is black. The default specification is `color=cyanorblack`, which means that the default color is black with the default `style=b` and cyan with the original style, style=`a`. Normally, a color name must be specified. The default value of "cyanorblack" is a special value that allows the macro to choose a color based on another option. If you specify any other value, the macro will not change it.

curvecol= *color*
specifies the color of curves in a regression plot. The default comes from `color=`.

excolors= *color-list*
excludes observations from the Annotate data set with colors in this list. For example, with a white background, to exclude all observations that have a color set to white as well as those with a computed white color, for example from `bright=` or `paint=`, specify `excolors=white CXFFFFFF`. This is done for efficiency, to make the Annotate data set smaller. (See the `paint=` option, page 778 for information on CXrrggbb color specifications.)

framecol= *color*
specifies the color of the frame. The default comes from `color=`.

labelcol= *color*
specifies the color of the variable labels. The default comes from `color=`.

monochro= *color*
overrides all other specified colors. This option is useful when you have specified colors and you want to temporarily override them to send the plot to a monochrome device. By default, when `monochro=` is null, this option has no effect. Typical usage: `monochro=black`.

style= `A | B`
specifies the output style. The new style, arbitrarily, style "b" is the default. To get the old style, specify `style=a` or `style=` without an argument. The new style has a gray background outside the plot, a white background inside the plot, and axis and tick colors are black. The new style does the following: When `cback=` is not specified in `gopplot=`, `cback=graye0` is added to the `gopplot=` specification. When `color=cyanorblack` is specified (the default), `color=cyan` is used with `style=a` and `color=black` is used with `style=b`. The `border` option is added to the `options=` list unless `options=noborder` is specified. In the `colors=` list, cyan comes before magenta with `style=a`, and magenta comes before cyan with `style=b`.

tickcol= *color*
specifies the color of ticks. The default comes from `color=`.

titlecol= *color*
specifies the color of the title. The default comes from `color=`.

Color Interpolation and Painting

These next options are used to create label and symbol colors using some function of the input data set variables. For example, you can plot the first two principal components on the x and y axes and show the third principal component in the same plot by using it to control the label colors. The `paint=` option gives you a simple and fairly general way to interpolate colors. The `red=`, `green=`, and `blue=` options are used together for many other types of interpolations, but these options are much harder to use. These options apply to `rgbtypes=` observations. If `red=`, `green=`, and `blue=` are not flexible enough, for example if you need full statements, specify `red=128` (so later code will know you are computing colors) then insert the full statements you need to generate the colors using `adjust1=`.

paint= *color-interpolation-specification*
is used to interpolate between colors based on the values of a variable. The simplest specification is `paint=variable`. More generally, specify:

```
paint=variable optional-color-list optional-data-value-list
```

The following color names are recognized: red, green, blue, yellow, magenta, cyan, black, white, orange, brown, gray, olive, pink, purple, violet. For other colors, specify the RGB color name. Colors can be represented as `CX`*rrggbb* where *rr* is the red value, *gg* is the green, and *bb* is blue, all three specified in hex. The base ten numbers 0 to 255 map to 00 to FF in hex. For example, white is `CXFFFFFF` (all colors at their maximum), black is `CX000000` (all colors at their minimum), red is `CXFF0000` (maximum red, minimum green and blue), blue is `CX0000FF` (maximum blue, minimum red and green), and magenta is `CXFF00FF` (maximum red and blue, minimum green). When a variable named `z` is specified with no other arguments, the default is `paint=z blue magenta red`. The option `paint=z red green 1 10` interpolates between red and green, based on the values of the variable `z`, where values of 1 or less map to red, values of 10 or more map to green, and values in between map to colors in between. The specification `paint=z red yellow green 1 5 10`, interpolates between red at `z=1`, yellow at `Z=5`, and green at `Z=10`. If the data value list is omitted, it is computed from the data.

red= *expression*
green= *expression*
blue= *expression*
specify for arithmetic expressions that produce integers in the range 0 to 255. Colors will be created as follows:

```
__color = 'CX' ||
          put(%if &red   ne %then round(&red,  __roured); %else 128; ,hex2.) ||
          put(%if &green ne %then round(&green,__rougre); %else 128;, hex2.) ||
          put(%if &blue  ne %then round(&blue, __roublu); %else 128; ,hex2.);}
```

The `__rou` variables are extracted from the second through fourth values of the `rgbround=` option. Example: `red = min(100 + (z - 10) * 3, 255)`, `blue=50`, `green=50`. Then all labels are various shades of red, depending on the value of `z`. Be aware that light colors (small red-green-blue values) do not show up well on white backgrounds and dark colors do not show up well on dark backgrounds. Typically, you will not want to use the full range of possible red-green-blue values. Computed values greater than 255 will be set to 255.

rgbround= *RGB-rounding-specification*

specifies rounding factors used for the `paint=` variable and RGB values. The default is `rgbround=-240 1 1 1`. The first value is used to round the `paint=` variable. Specify a positive value to have the variable rounded to multiples of that value. Specify a negative value $-n$ to have a maximum of n colors. For the other three values, specify positive values. The last three are rounding factors used to round the values for the red, green, and blue component of the color (see `red=`). If more than 256 colors are generated, you will get the error that a color was not added because the color table is full. By default, when a value is missing, there is no rounding. Rounding the `paint=` variable is useful with contour plots.

Contour Options

Use these options with contour plots. For example if the grid for a contour plot was generated as follows.

```
do x = -4 to 4 by 0.1;
   do y = -2 to 2 by 0.1;
      ... statements ...
      end;
   end;
```

then specify `hnobs=81`, `vnobs=41`. By default, the square root of the number of contour type observations is used for both `hnobs=` and `vnobs=` (which assumes a square grid).

hnobs= *n*

specifies the number of horizontal observations in the grid for contour plots.

vnobs= *n*

specifies the number of vertical observations in the grid for contour plots.

Advanced Plot Control Options

You can use the these next options to add full SAS DATA step statements to strategic places in the macro, such as the PROC PLOT step, the end of the preprocessing, and last full data steps. These options do minor adjustments before the final plot is produced. These options allow very powerful customization of your results to an extent not typically found in procedures. However, they may require a fair amount of work and some trial and error to understand and get right.

adjust1= *SAS-statements*

The following variables are created in the preprocessing data set:

`__lsize` - label size
`__lfont` - label font
`__lcolor` - label color
`__ssize` - symbol size
`__sfont` - symbol font
`__scolor` - symbol color
`__stype` - symbol type
`__symbol` - symbol value

__otype - observation type

Use adjust1= to adjust these variables in the preprocessing data set. You must specify complete statements with semicolons. Examples:

```
adjust1=%str(__lsize = 1.2; __lcolor = green;)}

adjust1=%str(if z > 20 then do;
__scolor = 'green'; __lcolor = 'green'; end;)}
```

adjust2= *SAS-statements*
includes statements with PROC PLOT such as format statements. Just specify the full statement.

adjust3= *SAS-statements*
adjust4= *SAS-statements*
specify options to adjust the final Annotate data set. For example, in Swiss fonts, asterisks are not vertically centered when printed, so adjust3= converts to use the SYMBOL function, so by default, adjust3=%str(if text = '*' and function = 'LABEL' then do; style = ' '; text = 'star'; function = 'SYMBOL'; end;). The default for adjust4= is null, so you can use it to add new statements. If you add new variables to the data set, you must also include a keep statement. Here is an example of using adjust4= to vertically print the y-axis label, like it would be in PROC PLOT.

```
adjust4=%str(if angle = 90 then do; angle = 270; rotate = 90; keep rotate; end;)
```

This example changes the size of title lines.

```
adjust4=%str(if index(comment, 'title') then size = 2;)
```

adjust5= *SAS-statements*
adds extra statements to the final DATA step that is used only for datatype=function. For example, to periodically mark the function with pluses, specify:

```
adjust5=%str( if mod(_n_,30) = 0 then do;
                size=0.25; function = 'LABEL'; text = '+'; output; end;)
```

Other Options

Here are the remaining options for the %PlotIt macro.

cirsegs= *n*
specifies a circle smoothness parameter used in determining the number of line segments in each circle. Smaller values create smoother circles. The cirsegs= value is approximately related to the length of the line segments that compose the circle. The default is cirsegs=.1.

cursegs= *n*
specifies the number of segments in a regression function curve. The default is cursegs=200.

debug= vars | dprint | notes | time | mprint
specifies values that control debugging output.

`dprint` - print intermediate data sets.

`mprint` - run with `options mprint`.

`notes` - do not specify `options nonotes` during most of the macro.

`time` - prints total macro run time, ignored with `options nostimer;`

`vars` - print macro options and macro variables for debugging.

You should provide a list of names for more than one type of debugging. Example: `debug=vars dprint notes time mprint`. The default is `debug=time`.

hpos= *n*
specifies the number of horizontal positions in the graphics area.

hsize= *n*
specifies the horizontal graphics area size in `unit=` units. The default is the maximum size for the device. By default, when `options=nocenter` is not specified, `hsize=` affects the size of the plot but not the `hsize= goption`. When `options=nocenter` is specified, `hsize=` affects both the plot size and the `hsize= goption`. If you specify just the `hsize=` but not `vsize=`, the vertical size will be scaled accordingly.

makefit= *n*
specifies the proportion of the graphics window to use. When the `makefit=` value is negative, the absolute value will be used, and the final value may be changed if the macro thinks that part of the plot may extend over the edge. When a positive value is specified, it will not be changed by the macro. When nonnull, the macro uses GASK to determine the minimum and maximum graphics window sizes and makes sure the plot can fit in them. The macro uses `gopprint=` or `gopplot=` to determine the device. The default is `makefit=-0.95`.

nknots= *n*
specifies the PROC TRANSREG number of knots option for regression functions.

outward= none | *'c'*
specifies a string for the PLOT statement `outward=` option. Normally, this option's value is constructed from the symbol that holds the place for vectors. Specify `outward=none` if you want to not have `outward=` specified for vectors. The `outward=` option is used to greatly increase the likelihood that labels from vectors will be printed outward−away from the origin.

ps= *n*
specifies the page size.

radii= *do-list*
specifies the radii of circles (in a DATA step do list). The unit corresponds to the horizontal axis variable. The `radii=` option can also specify a variable in the input data set when radii vary for each point. By default, no circles are drawn.

regopts= *options*
specifies the PROC TRANSREG options for curve fitting. Example: `regopts=nknots=10 evenly`.

regfun= *regression-function*
specifies the function for curve fitting. Possible values include:

`linear` - line

`spline` - nonlinear spline function, perhaps with knots

`mspline` - nonlinear but monotone spline function, perhaps with knots

`monotone` - nonlinear, monotone step function

See PROC TRANSREG documentation for more information

regprint= *regression-options*
specifies the PROC TRANSREG PROC statement options, typically printing options such as:

`noprint` - no regression printed output

`short` - suppress iteration histories

`ss2` - regression results

To see the regression table, specify: `regprint=ss2 short`. The default is `regprint=noprint`.

unit= `in` | `cm`
specifies the `hsize=` and `vsize=` unit in inches or centimeters (in or cm). The default is `unit=in`.

vechead= *vector-head-size* specifies how to draw vector heads. For example, the default specification `vechead=0.2 0.05`, specifies a head consisting of two hypotenuses from triangles with sides 0.2 units long along the vector and 0.05 units on the side perpendicular to the vector.

vpos= *n*
specifies the number of vertical positions in the graphics area.

vsize= *n*
specifies the vertical graphics area size in `unit=` units. The default is the maximum size for the device. By default when `options=nocenter` is not specified, `vsize=` affects the size of the plot but not the `vsize= goption`. When `options=nocenter` is specified, `vsize=` affects both the plot size and the `vsize= goption`. If you specify just the `vsize=` but not `hsize=`, the horizontal size will be scaled accordingly.

vtoh= *n*

specifies the PROC PLOT `vtoh=` option. The `vtoh=` option specifies the ratio of the vertical height of a typical character to the horizontal width. The default is `vtoh=2`. Do not specify values much different than 2, especially by default when you are using proportional fonts. There is no one-to-one correspondence between characters and cells and character widths vary, but characters tend to be approximately twice as high as they are wide. When you specify `vtoh=` values larger than 2, near-by labels may overlap, even when they do not collide in the printer plot. The macro uses this option to equate the axes so that a centimeter on one axis represents the same data range as a centimeter on the other axis. A null value can be specified, `vtoh=`, when you want the macro to just fill the window, like a typical GPLOT.

Smaller values give you more lines and smaller labels. The specification `vtoh=1.75` is a good alternative to `vtoh=2` when you need more lines to avoid collisions. The specification `vtoh=1.75` means 7 columns for each 4 rows between ticks (7 / 4 = 1.75). The `vtoh=2` specification means the plot will have 8 columns for each 4 rows between ticks. Note that PROC PLOT sometimes takes this value as a hint, not as a rigid specification so the actual value may be slightly different, particularly when a value other than 2.0 is specified. This is generally not a problem; the macro adjusts accordingly.

xmax= *n*

specifies the maximum horizontal size of the graphics area.

ymax= *n*

specifies the maximum vertical size of the graphics area.

Macro Errors

Usually, if you make a mistake in specifying macro options, the macro will print an informative message and quit. These macros go to great lengths to check their input and issue informative errors. However, *complete* error checking like we have with procedures is impossible in macros, and sometimes you will get a cascade of less than helpful error messages.* In that case, you will have to check the input and hunt for errors. One of the more common errors is a missing comma between options. Sometimes for harder errors, specifying `options mprint;` will help you locate the problem. You may get a listing with a *lot* of code, almost all of which you can ignore. Search for the error and look at the code that comes before the error for ideas about what went wrong. Once you think you know which option is involved, be sure to also check the option before and after in your macro invocation, because that might be where the problem really is.

The `%PhChoice` macro uses PROC TEMPLATE and ODS to create customized output tables. Typically, the instructions for this customization, created by PROC TEMPLATE, are stored in a file under the `sasuser` directory with a host dependent name. On some hosts, this name is `templat.sas7bitm`. On other hosts, the name is some variation of the name `templat`. Sometimes this file can be corrupted. When this happens, these macros will not run correctly, and you will see error messages including errors about invalid pages. The solution is to find the corrupt file under `sasuser` and delete it (using your ordinary operating system file deletion method). After that, this macros should run fine again. If you have run any other PROC TEMPLATE customizations, you will need to rerun them after deleting the file. For more information, see "Template Store" or "Item Store" in the SAS ODS documentation.

Sometimes, you will run the `%MktEx` macro, and everything will seem to run fine in the entire job, but at the end of your SAS log, you will see the message:

```
ERROR: Errors printed on page ....
```

Typically, this is caused by one or more PROC FACTEX steps failing to find the requested design. When this happens, the macro recovers and continues searching. The macro does not always know in advance if PROC FACTEX will succeed. The only way for it to find out is for it to try. The macro suppresses the PROC FACTEX error messages along with most other notes and warnings that would ordinarily come out. However, SAS still knows that a procedure tried to print an error message, and prints an error at the end of the log. This error can be ignored.

*If this happens, please write Warren.Kuhfeld@sas.com, and I will see if I can make the macros better handle that problem in the next release. Send all the code necessary to reproduce what you have done.

Linear Models and Conjoint Analysis with Nonlinear Spline Transformations

Warren F. Kuhfeld

Mark Garratt

Abstract

Many common data analysis models are based on the general linear univariate model, including linear regression, analysis of variance, and conjoint analysis. This chapter discusses the general linear model in a framework that allows nonlinear transformations of the variables. We show how to evaluate the effect of each transformation. Applications to marketing research are presented.*

Why Use Nonlinear Transformations?

In marketing research, as in other areas of data analysis, relationships among variables are not always linear. Consider the problem of modeling product purchasing as a function of product price. Purchasing may decrease as price increases. For consumers who consider price to be an indication of quality, purchasing may increase as price increases but then start decreasing as the price gets too high. The number of purchases may be a discontinuous function of price with jumps at "round numbers" such as even dollar amounts. In any case, it is likely that purchasing behavior is not a linear function of price. Marketing researchers who model purchasing as a linear function of price may miss valuable nonlinear information in their data. A transformation regression model can be used to investigate the nonlinearities. The data analyst is not required to specify the form of the nonlinear function; the data suggest the function.

The primary purpose of this chapter is to suggest the use of linear regression models with nonlinear transformations of the variables—*transformation regression* models. It is common in marketing research to model nonlinearities by fitting a quadratic polynomial model. Polynomial models often have collinearity problems, but that can be overcome with orthogonal polynomials. The problem that polynomials cannot overcome is the fact that polynomial curves are rigid; they do not do a good job of locally fitting the data. Piecewise polynomials or *splines* are generalizations of polynomials that provide more flexibility than ordinary polynomials.

*This chapter is a revision of a paper that was presented to the American Marketing Association, Advanced Research Techniques Forum, June 14–17, 1992, Lake Tahoe, Nevada. The authors are: Warren F. Kuhfeld, Manager, Multivariate Models R&D, SAS Institute Inc., Cary NC 27513-2414. Mark Garratt was with Conway | Milliken & Associates, when this paper was presented and is now with Miller Brewing Company. Copies of this chapter (TS-722J) and all of the macros are available on the web http://support.sas.com/techsup/tnote/tnote_stat.html#market.

Background and History

The foundation for our work can be found mostly in the psychometric literature. Some relevant references include: Kruskal & Shepard (1974); Young, de Leeuw, & Takane (1976); de Leeuw, Young, & Takane (1976); Perreault & Young (1980); Winsberg & Ramsay (1980); Young (1981); Gifi (1981, 1990); Coolen, van Rijckevorsel, & de Leeuw (1982); van Rijckevorsel (1982); van der Burg & de Leeuw (1983); de Leeuw (1986), and many others. The transformation regression problem has also received attention in the statistical literature (Breiman & Friedman 1985, Hastie & Tibshirani, 1986) under the names *ACE* and *generalized additive models*.

Our work is characterized by the following statements:

- Transformation regression is an inferential statistical technique, not a purely descriptive technique.
- We prefer smooth nonlinear spline transformations over step-function transformations.
- Transformations are found that minimize a squared-error loss function.

Many of the models discussed in this chapter can be directly fit with some data manipulations and any multiple regression or canonical correlation software; some models require specialized software. Algorithms are given by Kuhfeld (1990), de Boor (1978), and in SAS/STAT documentation.

Next, we present notation and review some fundamentals of the general linear univariate model.

The General Linear Univariate Model

A general linear univariate model has the scalar form

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_m x_m + \epsilon$$

and the matrix form

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

The dependent variable $\mathbf{y}$ is an $(n \times 1)$ vector of observations; $\mathbf{y}$ has expected value $E(\mathbf{y}) = \mathbf{X}\boldsymbol{\beta}$ and expected variance $V(\mathbf{y}) = \sigma^2\mathbf{I}_n$. The vector $\boldsymbol{\epsilon} = \mathbf{y} - \mathbf{X}\boldsymbol{\beta}$ contains the unobservable deviations from the expected values. The assumptions on $\mathbf{y}$ imply $E(\epsilon) = \mathbf{0}$ and $V(\epsilon) = \sigma^2\mathbf{I}_n$. The columns of $\mathbf{X}$ are the independent variables; $\mathbf{X}$ is an $(n \times m)$ matrix of constants that are assumed to be known without appreciable error. The elements of the column vector $\boldsymbol{\beta}$ are the parameters. The objectives of a linear models analysis are to estimate the parameter vector $\boldsymbol{\beta}$, estimate interesting linear combinations of the elements of $\boldsymbol{\beta}$, and test hypotheses about the parameters $\boldsymbol{\beta}$ or linear combinations of $\boldsymbol{\beta}$.

We discuss fitting linear models with nonlinear spline transformations of the variables. Note that we do *not* discuss models that are nonlinear in the parameters such as

$$y = e^{x\beta} + \epsilon$$

$$y = \beta_0 x^{\beta_1} + \epsilon$$

$$y = \frac{\beta_1 x_1 + \beta_2 x_1^2}{\beta_3 x_2 + \beta_4 x_2^2} + \epsilon$$

Our nonlinear transformations are found within the framework of the general linear model.

There are numerous special cases of the general linear model that are of interest. When all of the columns of **y** and **X** are interval variables, the model is a multiple regression model. When all of the columns of **X** are indicator variables created from nominal variables, the model is a main-effects analysis of variance model, or a metric conjoint analysis model. The model

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \epsilon$$

is of special interest. It is a *linear* model because it is linear in the parameters, and it models y as a *nonlinear* function of x. It is a *cubic polynomial* regression model, which is a special case of a spline.

Polynomial Splines

Splines are curves that are typically required to be continuous and smooth. Splines are usually defined as piecewise polynomials of degree d whose function values and first $d-1$ derivatives agree at the points where they join. The abscissa values of the join points are called knots. The term spline is also used for polynomials (splines with no knots), and piecewise polynomials with more than one discontinuous derivative. Splines with more knots or more discontinuities fit the data better and use more degrees of freedom (*df*). Fewer knots and fewer discontinuities provide smoother splines that user fewer *df*. A spline of degree three is a cubic spline, degree two splines are quadratic splines, degree one splines are piecewise linear, and degree zero splines are step functions. Higher degrees are rarely used.

A simple special case of a spline is the line,

$$\beta_0 + \beta_1 x$$

from the simple regression model

$$y = \beta_0 + \beta_1 x + \epsilon$$

A line is continuous and completely smooth. However, there is little to be gained by thinking of a line as a spline. A special case of greater interest was mentioned previously. The polynomial

$$\beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$$

from the linear model

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \epsilon$$

is a cubic spline with no knots. This equation models y as a *nonlinear* function of x, but does so with a *linear* regression model; y is a linear function of x, x^2, and x^3. Table 1 shows the **X** matrix, $(\mathbf{1} \quad \mathbf{x} \quad \mathbf{x}^2 \quad \mathbf{x}^3)$, for a cubic polynomial, where $x = -5, -4, ..., 5$. Figure 1 plots the polynomial terms (except the intercept). See Smith (1979) for an excellent introduction to splines.

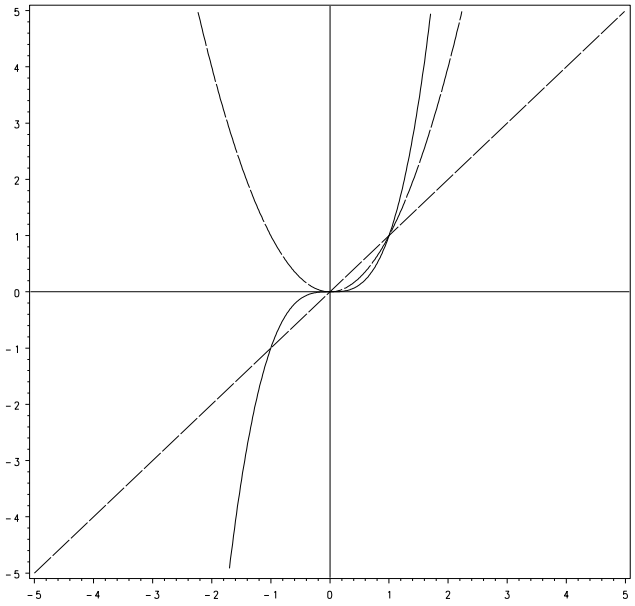


Figure 1. Linear, Quadratic, and Cubic Curves

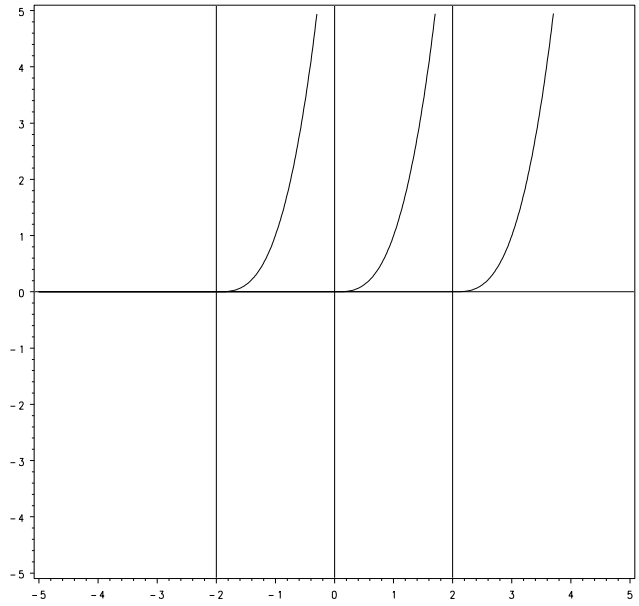


Figure 2. Curves For Knots at −2, 0, 2

Splines with Knots

Here is an example of a polynomial spline model with three knots at t_1, t_2, and t_3.

$$\begin{aligned} y &= \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \\ &\quad \beta_4 (x > t_1)(x - t_1)^3 + \\ &\quad \beta_5 (x > t_2)(x - t_2)^3 + \\ &\quad \beta_6 (x > t_3)(x - t_3)^3 + \epsilon \end{aligned}$$

The Boolean expression $(x > t_1)$ is 1 if $x > t_1$, and 0 otherwise. The term

$$\beta_4 (x > t_1)(x - t_1)^3$$

is zero when $x \leq t_1$ and becomes nonzero, letting the curve change, as x becomes greater than knot t_1. This spline uses more *df* and is less smooth than the polynomial model

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \epsilon$$

Assume knots at -2, 0, and 2; the spline model is:

$$\begin{aligned} y &= \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \\ &\quad \beta_4 (x > -2)(x - -2)^3 + \\ &\quad \beta_5 (x > \ \ 0)(x - \ \ 0)^3 + \\ &\quad \beta_6 (x > \ \ 2)(x - \ \ 2)^3 + \epsilon \end{aligned}$$

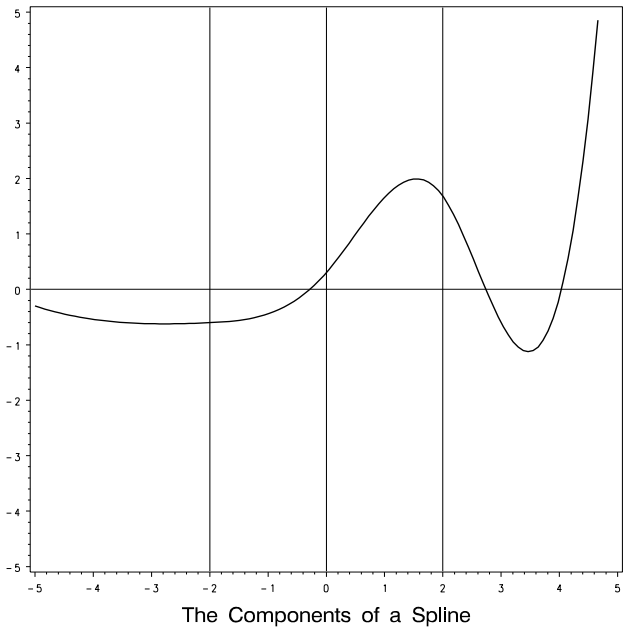


Figure 3. A Spline Curve With Knots at −2, 0, 2

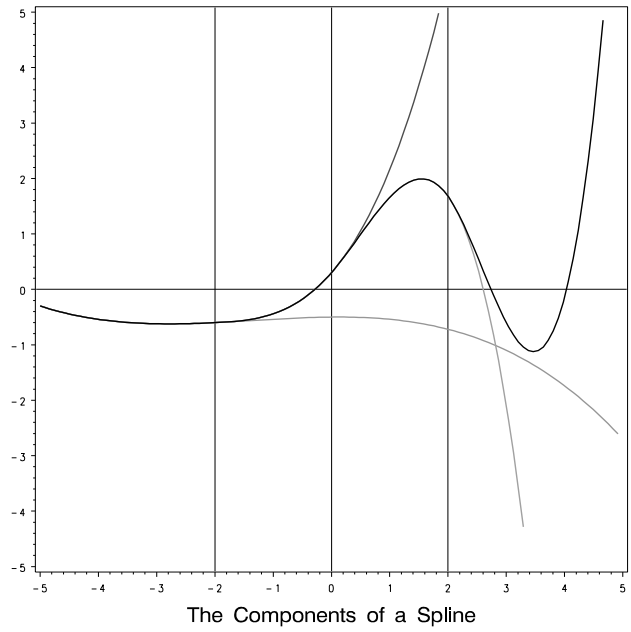


Figure 4. The Components of the Spline

Table 2 shows an **X** matrix for this model, Figure 1 plots the polynomial terms, and Figure 2 plots the knot terms.

The β_0, $\beta_1 x$, $\beta_2 x^2$, and $\beta_3 x^3$ terms contribute to the overall shape of the curve. The

$$\beta_4(x > -2)(x - -2)^3$$

term has no effect on the curve before $x = -2$, and allows the curve to change at $x = -2$. The $\beta_4(x > -2)(x - -2)^3$ term is exactly zero at $x = -2$ and increases as x becomes greater than -2. The

Table 1
Cubic Polynomial
Spline Basis

1	-5	25	-125
1	-4	16	-64
1	-3	9	-27
1	-2	4	-8
1	-1	1	-1
1	0	0	0
1	1	1	1
1	2	4	8
1	3	9	27
1	4	16	64
1	5	25	125

Table 2
Cubic Polynomial
With Knots at −2, 0, 2

1	-5	25	-125	0	0	0
1	-4	16	-64	0	0	0
1	-3	9	-27	0	0	0
1	-2	4	-8	0	0	0
1	-1	1	-1	1	0	0
1	0	0	0	8	0	0
1	1	1	1	27	1	0
1	2	4	8	64	8	0
1	3	9	27	125	27	1
1	4	16	64	216	64	8
1	5	25	125	343	125	27

Table 3
Basis for a Discontinuous (at 0) Spline

1	-5	25	-125	0	0	0	0
1	-4	16	-64	0	0	0	0
1	-3	9	-27	0	0	0	0
1	-2	4	-8	0	0	0	0
1	-1	1	-1	0	0	0	0
1	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
1	2	4	8	1	2	4	8
1	3	9	27	1	3	9	27
1	4	16	64	1	4	16	64
1	5	25	125	1	5	25	125

$\beta_4(x > -2)(x - -2)^3$ term contributes to the shape of the curve even beyond the next knot at $x = 0$, but at $x = 0$,

$$\beta_5(x > 0)(x - 0)^3$$

allows the curve to change again. Finally, the last term

$$\beta_6(x > 2)(x - 2)^3$$

allows one more change. For example, consider the curve in Figure 3. It is the spline

$$\begin{aligned} y &= -0.5 + 0.01x + -0.04x^2 + -0.01x^3 + \\ &\quad 0.1(x > -2)(x - -2)^3 + \\ &\quad -0.5(x > 0)(x - 0)^3 + \\ &\quad 1.5(x > 2)(x - 2)^3 \end{aligned}$$

It is constructed from the curves in Figure 4. At $x = -2.0$ there is a branch;

$$y = -0.5 + 0.01x + -0.04x^2 + -0.01x^3$$

continues over and down while the curve of interest,

$$\begin{aligned} y &= -0.5 + 0.01x + -0.04x^2 + -0.01x^3 + \\ &\quad 0.1(x > -2)(x - -2)^3 \end{aligned}$$

starts heading upwards. At $x = 0$, the addition of

$$-0.5(x > 0)(x - 0)^3$$

slows the ascent until the curve starts decreasing again. Finally, the addition of

$$1.5(x > 2)(x - 2)^3$$

produces the final change. Notice that the curves do not immediately diverge at the knots. The function and its first two derivatives are continuous, so the function is smooth everywhere.

Derivatives of a Polynomial Spline

The next equations show a cubic spline model with a knot at t_1 and its first three derivatives with respect to x.

$$\begin{aligned} y &= \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \\ &\quad \beta_4(x > t_1)(x - t_1)^3 + \epsilon \end{aligned}$$

$$\begin{aligned} \frac{dy}{dx} &= \beta_1 + 2\beta_2 x + 3\beta_3 x^2 + \\ &\quad 3\beta_4(x > t_1)(x - t_1)^2 \end{aligned}$$

$$\begin{aligned} \frac{d^2y}{dx^2} &= 2\beta_2 + 6\beta_3 x + \\ &\quad 6\beta_4(x > t_1)(x - t_1) \end{aligned}$$

$$\frac{d^3y}{dx^3} = 6\beta_3 + 6\beta_4(x > t_1)$$

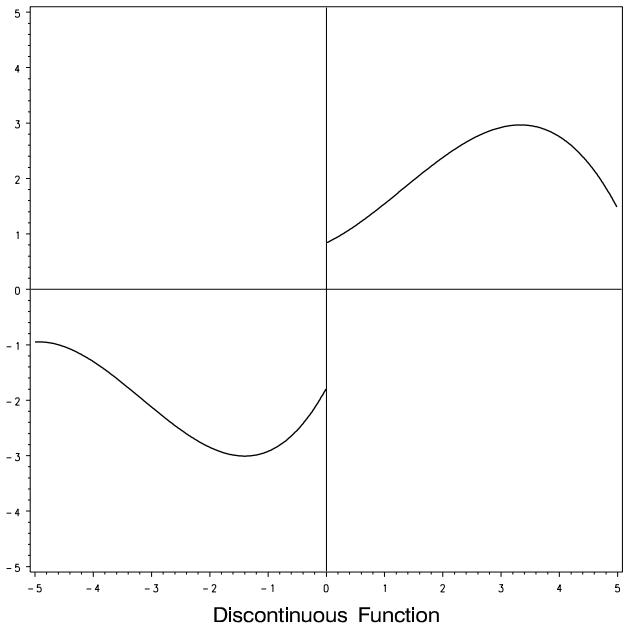


Figure 5. A Discontinuous Spline Function

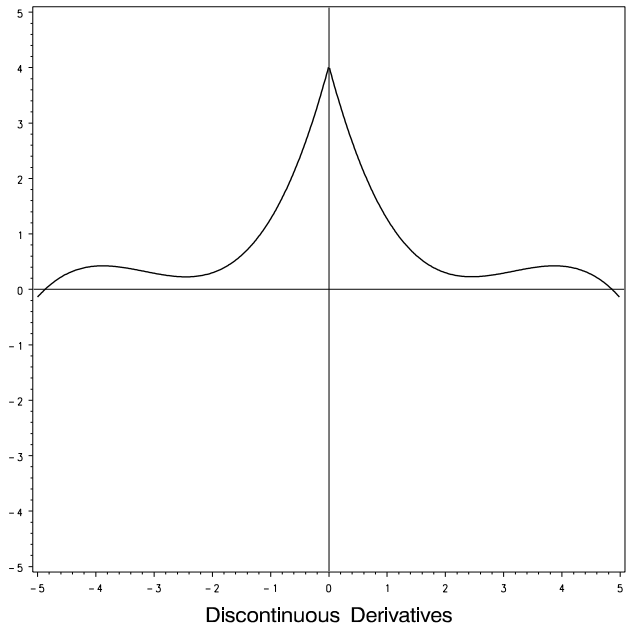


Figure 6. A Spline With a Discontinuous Slope

The first two derivatives are continuous functions of x at the knots. This is what gives the spline function its smoothness at the knots. In the vicinity of the knots, the curve is continuous, the slope of the curve is a continuous function, and the rate of change of the slope function is a continuous function. The third derivative is discontinuous at the knots. It is the horizontal line $6\beta_3$ when $x \leq t_1$ and jumps to the horizontal line $6\beta_3 + 6\beta_4$ when $x > t_1$. In other words, the cubic part of the curve changes at the knots, but the linear and quadratic parts do not change.

Discontinuous Spline Functions

Here is an example of a spline model that is discontinuous at $x = t_1$.

$$\begin{aligned} y &= \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \\ &\quad \beta_4 (x > t_1) + \\ &\quad \beta_5 (x > t_1)(x - t_1) + \\ &\quad \beta_6 (x > t_1)(x - t_1)^2 + \\ &\quad \beta_7 (x > t_1)(x - t_1)^3 + \epsilon \end{aligned}$$

Figure 5 shows an example, and Table 3 shows a design matrix for this model with $t_1 = 0$. The fifth column is a binary (zero/one) vector that allows the discontinuity. It is a change in the intercept parameter. Note that the sixth through eighth columns are necessary if the spine is to consist of two independent polynomials. Without them, there is a jump at $t_1 = 0$, but both curves are based on the

same polynomial. For $x \leq t_1$, the spline is

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \epsilon$$

and for $x > t_1$, the spline is

$$\begin{aligned} y &= \beta_0 + \beta_4 + \\ &\quad \beta_1 x + \beta_5 (x - t_1) + \\ &\quad \beta_2 x^2 + \beta_6 (x - t_1)^2 + \\ &\quad \beta_3 x^3 + \beta_7 (x - t_1)^3 + \epsilon \end{aligned}$$

The discontinuities are as follows:

$$\beta_7 (x > t_1)(x - t_1)^3$$

specifies a discontinuity in the third derivative of the spline function at t_1,

$$\beta_6 (x > t_1)(x - t_1)^2$$

specifies a discontinuity in the second derivative at t_1,

$$\beta_5 (x > t_1)(x - t_1)$$

specifies a discontinuity in the derivative at t_1, and

$$\beta_4 (x > t_1)$$

specifies a discontinuity in the function at t_1. The function consists of two separate polynomial curves, one for $-\infty < x \leq t_1$ and the other for $t_1 < x < \infty$. This kind of spline can be used to model a discontinuity in price.

Here is an example of a spline model that is continuous at $x = t_1$ but does not have $d - 1$ continuous derivatives.

$$\begin{aligned} y &= \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \\ &\quad \beta_4 (x > t_1)(x - t_1) + \\ &\quad \beta_5 (x > t_1)(x - t_1)^2 + \\ &\quad \beta_6 (x > t_1)(x - t_1)^3 + \epsilon \end{aligned}$$

$$\begin{aligned} \frac{dy}{dx} &= \beta_1 + 2\beta_2 x + 3\beta_3 x^2 + \\ &\quad \beta_4 (x > t_1) + \\ &\quad 2\beta_5 (x > t_1)(x - t_1) + \\ &\quad 3\beta_6 (x > t_1)(x - t_1)^2 \end{aligned}$$

Since the first derivative is not continuous at $t_1 = x$, the slope of the spline is not continuous at $t_1 = x$. Figure 6 contains an example with $t_1 = 0$. Notice that the slope of the curve is indeterminate at zero.

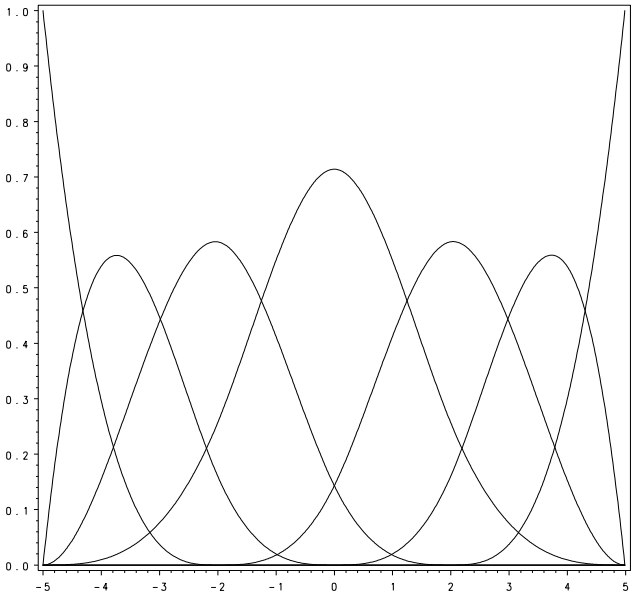


Table 4
Cubic B-Spline With Knots at −2, 0, 2

1.00	0.00	0.00	0.00	0.00	0.00	0.00
0.30	0.54	0.15	0.01	0.00	0.00	0.00
0.04	0.45	0.44	0.08	0.00	0.00	0.00
0.00	0.16	0.58	0.26	0.00	0.00	0.00
0.00	0.02	0.41	0.55	0.02	0.00	0.00
0.00	0.00	0.14	0.71	0.14	0.00	0.00
0.00	0.00	0.02	0.55	0.41	0.02	0.00
0.00	0.00	0.00	0.26	0.58	0.16	0.00
0.00	0.00	0.00	0.08	0.44	0.45	0.04
0.00	0.00	0.00	0.01	0.15	0.54	0.30
0.00	0.00	0.00	0.00	0.00	0.00	1.00

Figure 7. B-Splines With Knots at −2, 0, 2

Monotone Splines and B-Splines

An increasing *monotone spline* never decreases; its slope is always positive or zero. Decreasing monotone splines, with slopes that are always negative or zero, are also possible. Monotone splines cannot be conveniently created from polynomial splines. A different basis, the *B-spline* basis, is used instead. Polynomial splines provide a convenient way to describe splines, but B-splines provide a better way to fit spline models.

The columns of the B-spline basis are easily constructed with a recursive algorithm (de Boor, 1978, pages 134−135). A basis for a vector space is a linearly independent set of vectors; every vector in the space has a unique representation as a linear combination of a given basis. Table 4 shows the B-spline $\mathbf{X}$ matrix for a model with knots at −2, 0, and 2. Figure 7 shows the B-spline curves. The columns of the matrix in Table 4 can all be constructed by taking linear combinations of the columns of the polynomial spline in Table 2. Both matrices form a basis for the same vector space.

The numbers in the B-spline basis are all between zero and one, and the marginal sums across columns are all ones. The matrix has a diagonally banded structure, such that the band moves one position to the right at each knot. The matrix has many more zeros than the matrix of polynomials and much smaller numbers. The columns of the matrix are not orthogonal like a matrix of orthogonal polynomials, but collinearity is not a problem with the B-spline basis like it is with a polynomial spline. The B-spline basis is very stable numerically.

To illustrate, 1000 evenly spaced observations were generated over the range -5 to 5. Then a B-spline basis and polynomial spline basis were constructed with knots at −2, 0, and 2. The eigenvalues for the centered $\mathbf{X}'\mathbf{X}$ matrices, excluding the last structural zero eigenvalue, are given in Table 5. In the

Table 5

Polynomial and B-Spline Eigenvalues

B-Spline Basis		
Eigenvalue	Proportion	Cumulative
0.107872	0.358718	0.35872
0.096710	0.321599	0.68032
0.046290	0.153933	0.83425
0.030391	0.101062	0.93531
0.012894	0.042878	0.97819
0.006559	0.021810	1.00000

Polynomial Spline Basis		
Eigenvalue	Proportion	Cumulative
10749.8	0.941206	0.94121
631.8	0.055317	0.99652
37.7	0.003300	0.99982
1.7	0.000148	0.99997
0.3	0.000029	1.00000
0.0	0.000000	1.00000

polynomial splines, the first two components already account for more than 99% of the variation of the points. In the B-splines, the cumulative proportion does not pass 99% until the last term. The eigenvalues show that the B-spline basis is better conditioned than the piecewise polynomial basis. B-splines are used instead of orthogonal polynomials or Box-Cox transformations because B-splines allow knots and more general curves. B-splines also allow monotonicity constraints.

A transformation of x is monotonically increasing if the coefficients that are used to combine the columns of the B-spline basis are monotonically increasing. Models with splines can be fit directly using ordinary least squares (OLS). OLS does not work for monotone splines because OLS has no method of ensuring monotonicity in the coefficients. When there are monotonicity constraints, an alternating least square (ALS) algorithm is used. Both OLS and ALS attempt to minimize a squared error loss function. See Kuhfeld (1990) for a description of the iterative algorithm that fits monotone splines. See Ramsay (1988) for some applications and a different approach to ensuring monotonicity.

Transformation Regression

If the dependent variable is not transformed and if there are no monotonicity constraints on the independent variable transformations, the transformation regression model is the same as the OLS regression model. When only the independent variables are transformed, the transformation regression model is nothing more than a different rendition of an OLS regression. All of the spline models presented up to this point can be reformulated as

$$y = \beta_0 + \Phi(x) + \epsilon$$

The nonlinear transformation of x is $\Phi(x)$; it is solved for by fitting a spline model such as

$$\begin{aligned} y \;=\; & \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \\ & \beta_4 (x > t_1)(x - t_1)^3 + \\ & \beta_5 (x > t_2)(x - t_2)^3 + \\ & \beta_6 (x > t_3)(x - t_3)^3 + \epsilon \end{aligned}$$

where

$$\begin{aligned}\Phi(x) &= \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \\ &\quad \beta_4 (x > t_1)(x - t_1)^3 + \\ &\quad \beta_5 (x > t_2)(x - t_2)^3 + \\ &\quad \beta_6 (x > t_3)(x - t_3)^3\end{aligned}$$

Consider a model with two polynomials:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2 + \beta_3 x_1^3 + \beta_4 x_2 + \beta_5 x_2^2 + \beta_6 x_2^3 + \epsilon$$

It is the same as a transformation regression model

$$y = \beta_0 + \Phi_1(x_1) + \Phi_2(x_2) + \epsilon$$

where $\Phi_\bullet(\bullet)$ designates cubic spline transformations with no knots. The actual transformations in this case are

$$\widehat{\Phi}_1(x_1) = \widehat{\beta}_1 x_1 + \widehat{\beta}_2 x_1^2 + \widehat{\beta}_3 x_1^3$$

and

$$\widehat{\Phi}_2(x_2) = \widehat{\beta}_4 x_2 + \widehat{\beta}_5 x_2^2 + \widehat{\beta}_6 x_2^3$$

There are six model *df*. The test for the effect of the transformation $\Phi_1(x_1)$ is the test of the linear hypothesis $\beta_1 = \beta_2 = \beta_3 = 0$, and the $\Phi_2(x_2)$ transformation test is a test that $\beta_4 = \beta_5 = \beta_6 = 0$. Both tests are *F*-tests with three numerator *df*. When there are monotone transformations, constrained least-squares estimates of the parameters are obtained.

Degrees of Freedom

In an ordinary general linear model, there is one parameter for each independent variable. In the transformation regression model, many of the variables are used internally in the bases for the transformations. Each linearly independent basis column has one parameter and one model *df*. If a variable is not transformed, it has one parameter. Nominal classification variables with c categories have $c - 1$ parameters. For degree d splines with k knots and $d - 1$ continuous derivatives, there are $d + k$ parameters.

When there are monotonicity constraints, counting the number of scoring parameters is less precise. One way of handling a monotone spline transformation is to treat it as if it were simply a spline transformation with $d + k$ parameters. However, there are typically fewer than $d + k$ *unique* parameter estimates since some of those $d + k$ scoring parameter estimates may be tied to impose the order constraints. Imposing ties is equivalent to fitting a model with fewer parameters. So, there are two available scoring parameter counts: $d + k$ and a potentially smaller number that is determined during the analysis. Using $d + k$ as the model *df* is *conservative* since the scoring parameter estimates are restricted. Using the smaller count is too *liberal* since the data and the model together are being used to determine the number of parameters. Our solution is to report tests using both liberal and conservative *df* to provide lower and upper bounds on the "true" *p*-values.

Dependent Variable Transformations

When a dependent variable is transformed, the problem becomes multivariate:

$$\Phi_0(y) = \beta_0 + \Phi_1(x_1) + \Phi_2(x_2) + \epsilon$$

Hypothesis tests are performed in the context of a multivariate linear model, with the number of dependent variables equal to the number of scoring parameters for the dependent variable transformation. Multivariate normality is assumed even though it is known that the assumption is *always* violated. This is one reason that all hypothesis tests in the presence of a dependent variable transformation should be considered approximate at best.

For the transformation regression model, we redefine three of the usual multivariate test statistics: Pillai's Trace, Wilks' Lambda, and the Hotelling-Lawley Trace. These statistics are normally computed using all of the squared canonical correlations, which are the eigenvalues of the matrix $\mathbf{H}(\mathbf{H}+\mathbf{E})^{-1}$. Here, there is only one linear combination (the transformation) and hence only one squared canonical correlation of interest, which is equal to the R^2. We use R^2 for the first eigenvalue; all other eigenvalues are set to zero since only one linear combination is used. Degrees of freedom are computed assuming that all linear combinations contribute to the Lambda and Trace statistics, so the F-tests for those statistics are conservative. In practice, the adjusted Pillai's Trace is very conservative—perhaps too conservative to be useful. Wilks' Lambda is less conservative, and the Hotelling-Lawley Trace seems to be the least conservative.

It may seem that the Roy's Greatest Root statistic, which always uses only the largest squared canonical correlation, is the only statistic of interest. Unfortunately, Roy's Greatest Root is very liberal and only provides a lower bound on the p-value. The p-values for the liberal and conservative statistics are used together to provide approximate lower and upper bounds on p.

Scales of Measurement

Early work in scaling, such as Young, de Leeuw, & Takane (1976), and Perreault & Young (1980) was concerned with fitting models with mixed nominal, ordinal, and interval scale of measurement variables. Nominal variables were optimally scored using Fisher's (1938) optimal scoring algorithm. Ordinal variables were optimally scored using the Kruskal and Shepard (1974) monotone regression algorithm. Interval and ratio scale of measurement variables were left alone nonlinearly transformed with a polynomial transformation.

In the transformation regression setting, the Fisher optimal scoring approach is equivalent to using an indicator variable representation, as long as the correct *df* are used. The optimal scores are category means. Introducing optimal scaling for nominal variables does not lead to any increased capability in the regression model.

For ordinal variables, we believe the Kruskal and Shepard monotone regression algorithm should be reserved for the situation when a variable has only a few categories, say five or fewer. When there are more levels, a monotone spline is preferred because it uses fewer model *df* and because it is less likely to capitalize on chance.

Interval and ratio scale of measurement variables can be left alone or nonlinearly transformed with splines or monotone splines. When the true model has a nonlinear function, say

$$y = \beta_0 + \beta_1 log(x) + \epsilon$$

or

$$y = \beta_0 + \beta_1/x + \epsilon$$

the transformation regression model

$$y = \beta_0 + \Phi(x) + \epsilon$$

can be used to hunt for parametric transformations. Plots of $\widehat{\Phi}(x)$ may suggest log or reciprocal transformations.

Conjoint Analysis

Green & Srinivasan (1990) discuss some of the problems that can be handled with a transformation regression model, particularly the problem of degrees of freedom. Consider a conjoint analysis design where a factor with $c > 3$ levels has an inherent ordering. By finding a quadratic monotone spline transformation with no knots, that variable will use only two *df* instead of the larger $c - 1$. The model *df* in a spline transformation model are determined by the data analyst, not by the number of categories in the variables. Furthermore, a "*quasi-metric*" conjoint analysis can be performed by finding a monotone spline transformation of the dependent variable. This model has fewer restrictions than a metric analysis, but will still typically have error *df*, unlike the nonmetric analysis.

Curve Fitting Applications

With a simple regression model, you can fit a line through a $y \times x$ scatter plot. With a transformation regression model, you can fit a curve through the scatter plot. The y-axis coordinates of the curve are

$$\widehat{y} = \widehat{\beta}_0 + \widehat{\Phi}(x)$$

from the model

$$y = \beta_0 + \Phi(x) + \epsilon$$

With more than one group of observations and a multiple regression model, you can fit multiple lines, lines with the same slope but different intercepts, and lines with common intercepts but different slopes. With the transformation regression model, you can fit multiple curves through a scatter plot. The curves can be monotone or not, constrained to be parallel, or constrained to have the same intercept. Consider the problem of modeling the number of product purchases as a function of price. Separate curves can be simultaneously fit for two groups who may behave differently, for example those who are making a planned purchase and those who are buying impulsively. Later in this chapter, there is an example of plotting brand by price interactions.

Figure 8 contains an artificial example of two separate spline functions; the shapes of the two curves are independent of each other, and $R^2 = 0.87$. In Figure 9, the splines are constrained to be parallel, and $R^2 = 0.72$. The parallel curve model is more restrictive and fits the data less well than the unconstrained model. In Figure 8, each curve follows its swarm of data. In Figure 9, the curves find paths through the data that are best on the average considering both swarms together. In the vicinity of $x = -2$, the top curve is high and the bottom curve is low. In the vicinity of $x = 1$, the top curve is low and the bottom curve is high.

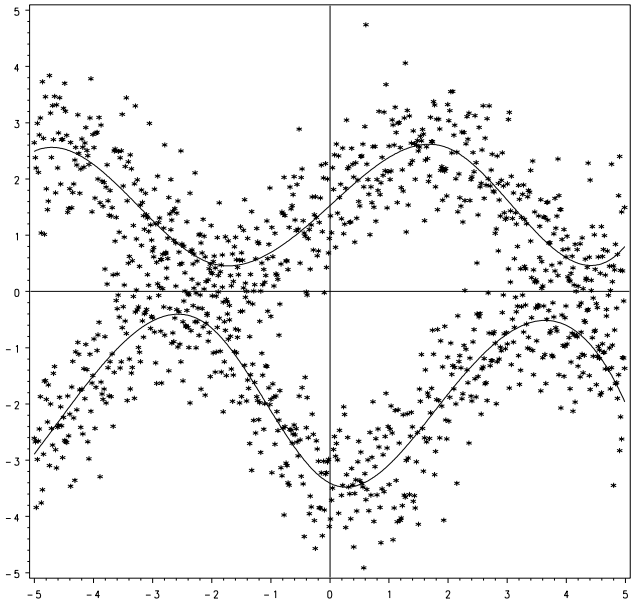


Figure 8. Separate Spline Functions, Two Groups

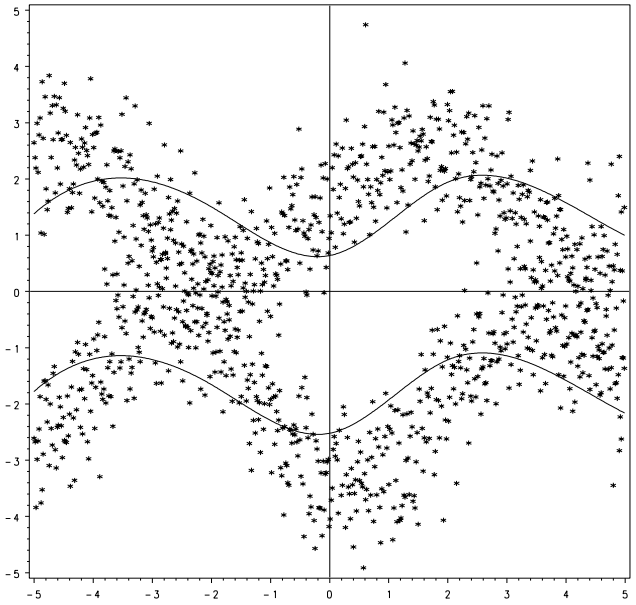


Figure 9. Parallel Spline Functions, Two Groups

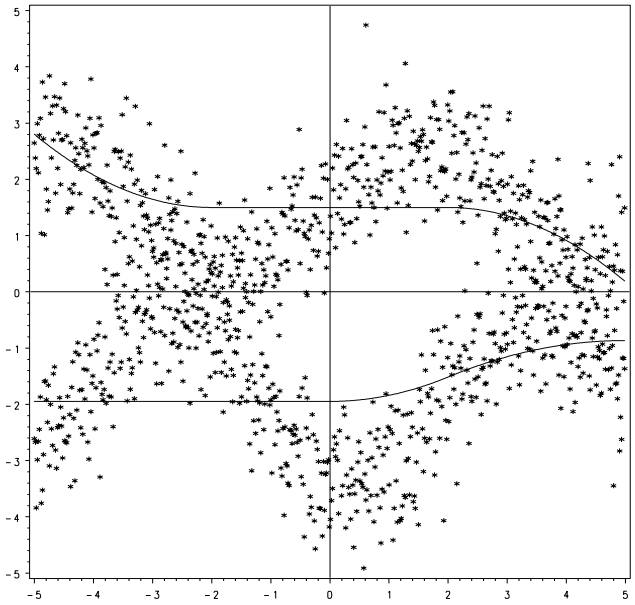


Figure 10. Monotone Spline Functions, Two Groups

Figure 10 contains the same data and two monotonic spline functions; the shapes of the two curves are independent of each other, and $R^2 = 0.71$. The top curve is monotonically decreasing, whereas the bottom curve is monotonically increasing. The curves in Figure 10 flatten where there is nonmonotonicity in Figure 8.

Parallel curves are very easy to model. If there are two groups and the variable g is a binary variable indicating group membership, fit the model

$$y = \beta_0 + \beta_1 g + \Phi_1(x) + \epsilon$$

where $\Phi_1(x)$ is a linear, spline, or monotone spline transformation. Plot $\widehat{y}$ as a function of x to see the two curves. Separate curves are almost as easy; the model is

$$y = \beta_0 + \beta_1 g + \Phi_1(x \times (1 - g)) + \Phi_2(x \times g) + \epsilon$$

When $x \times (1 - g)$ is zero, $x \times g$ is x, and vice versa.

Spline Functions of Price

This section illustrates splines with an artificial data set. Imagine that subjects were asked to rate their interest in purchasing various types of spaghetti sauces on a one to nine scale, where nine indicated definitely will buy and one indicated definitely will *not* buy. Prices were chosen from typical retail trade prices, such as \$1.49, \$1.99, \$2.49, and \$2.99; and one penny more than a typical price, \$1.00, \$1.50, \$2.00, and \$2.50. Between each "round" number price, such as \$1.00, and each typical price, such as \$1.49, three additional prices were chosen, such as \$1.15, \$1.25, and \$1.35. The goal is to allow a model with a separate spline for each of the four ranges: \$1.00 — \$1.49, \$1.50 — \$1.99, \$2.00 — \$2.49, and \$2.50 — \$2.99. For each range, a spline with zero or one knot can be fit.

One rating for each price was constructed and various models were fit to the data. Figures 11 through 18 contain results. For each figure, the number of model *df* are printed. One additional *df* for the intercept is also used. The SAS/STAT procedure TRANSREG was used to fit all of the models in this chapter.

Figure 11 shows the linear fit, Figure 12 uses a quadratic polynomial, and Figure 13 uses a cubic polynomial. The curve in Figure 13 has a slight nonmonotonicity in the tail, and since it is a polynomial, it is rigid and cannot locally fit the data values.

Figure 14 shows a monotone spline. It closely follows the data and never increases. A range for the model *df* is specified; the larger value is a conservative count and the smaller value is a liberal count.

The curves in Figures 12 through 14 are all continuous and smooth. These curves do a good job of following the data, but inspection of the data suggests that a different model may be more appropriate. There is a large drop in purchase interest when price increases from \$1.49 to \$1.50, a smaller drop between \$1.99 and \$2.00, and a still smaller drop between \$2.49 and \$2.50.

In Figure 15, a separate quadratic polynomial is fit for each of the four price ranges: \$1.00 — \$1.49, \$1.50 — \$1.99, \$2.00 — \$2.49, and \$2.50 — \$2.99. The functions are connected. The function over the range \$1.00 — \$1.49 is nearly flat; there is high purchase interest for all of these prices. Over the range \$1.50 — \$1.99, purchase interest drops more rapidly with a slight leveling in the low end; the slope decreases as the function increases. Over the range \$2.00 — \$2.49, purchase interest drops less

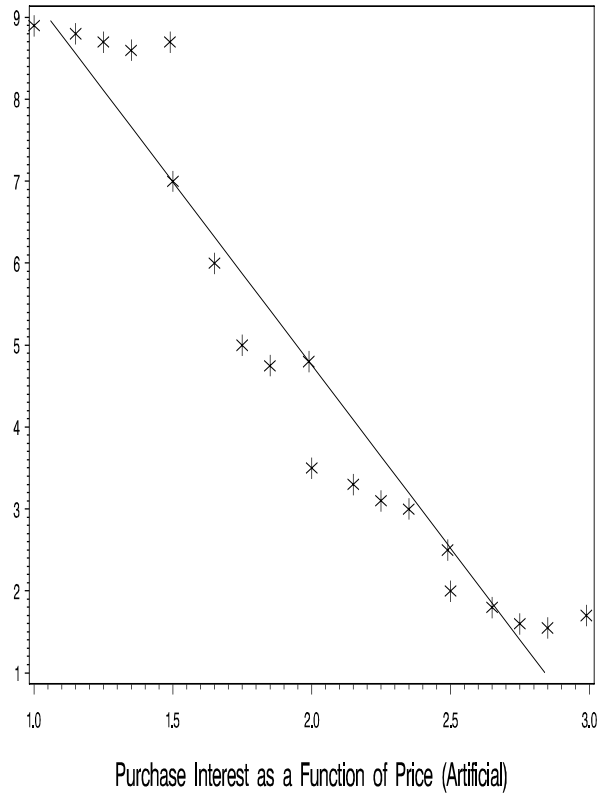


Figure 11. Linear Function, 1 df

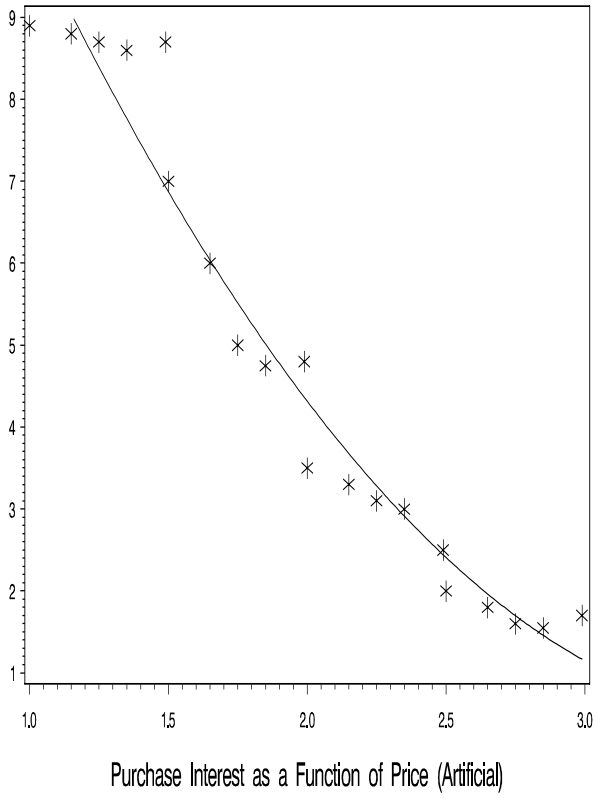


Figure 12. Quadratic Function, 2 df

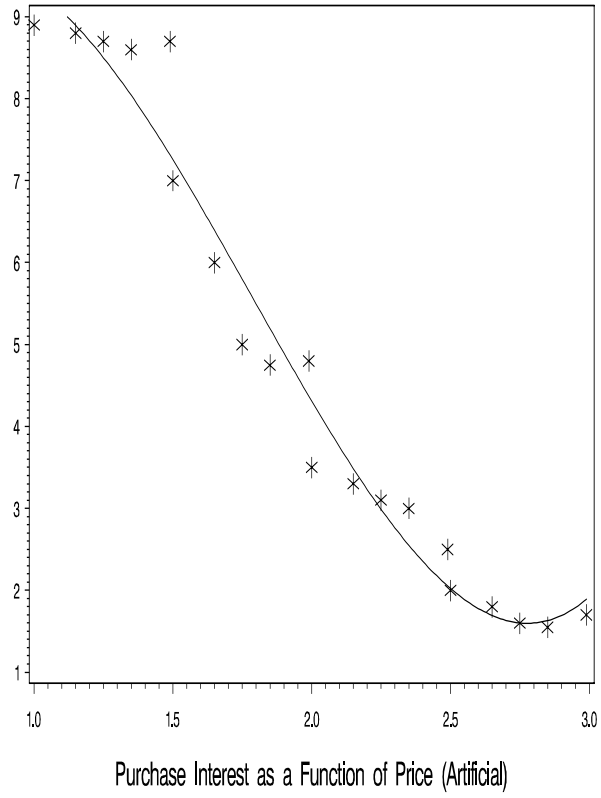


Figure 13. Cubic Function, 3 df

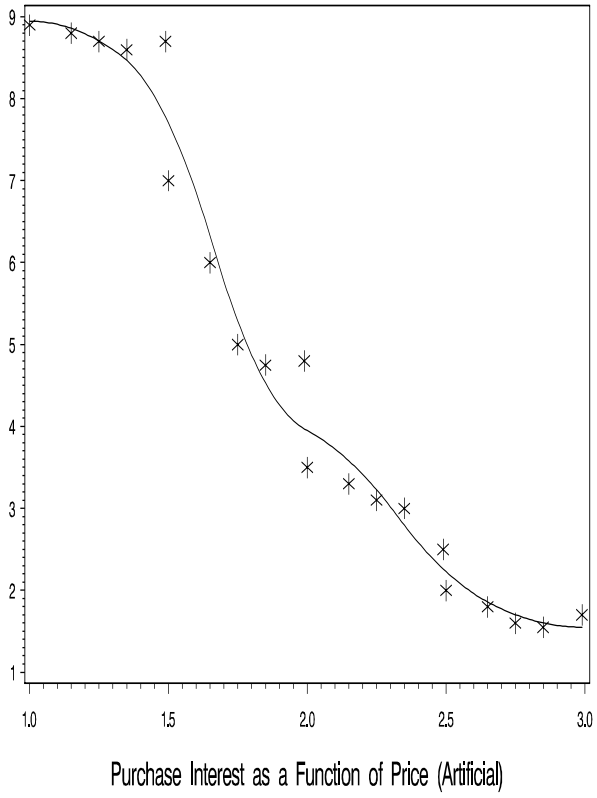


Figure 14. Monotone Function, 5–7 df

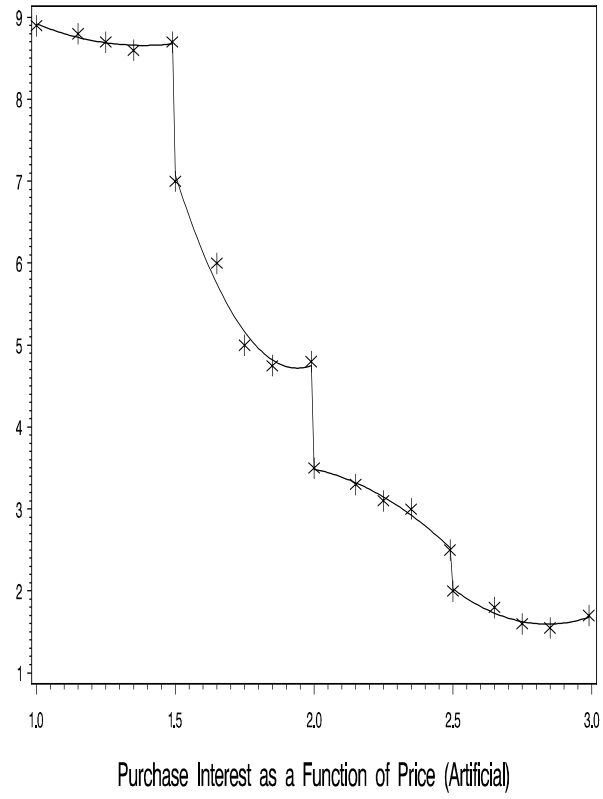


Figure 15. Discontinuous Polynomial, 11 df

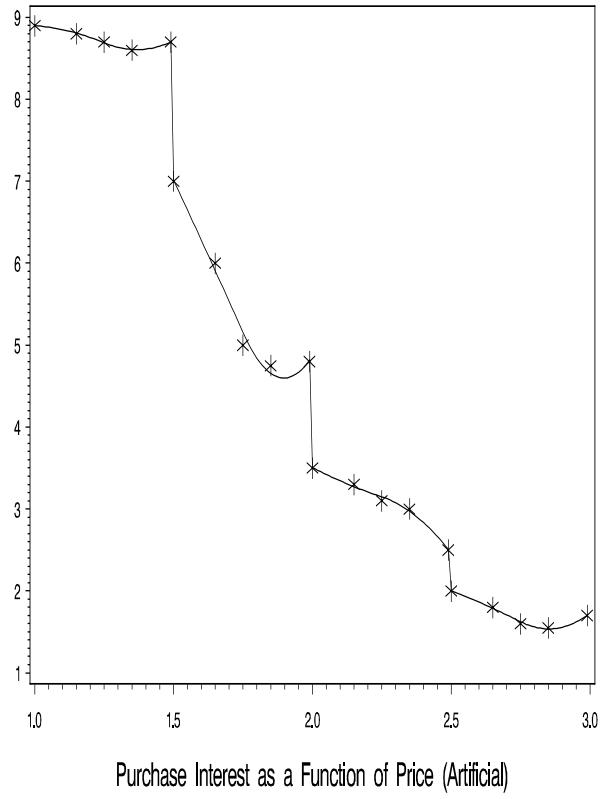


Figure 16. Discontinuous Spline Function, 15 df

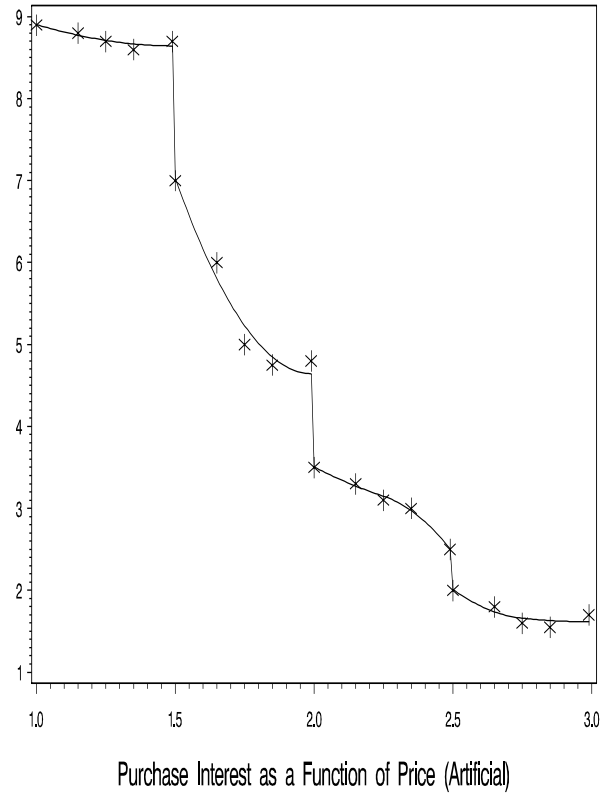


Figure 17. Discontinuous Monotone Spline, 12–15 df

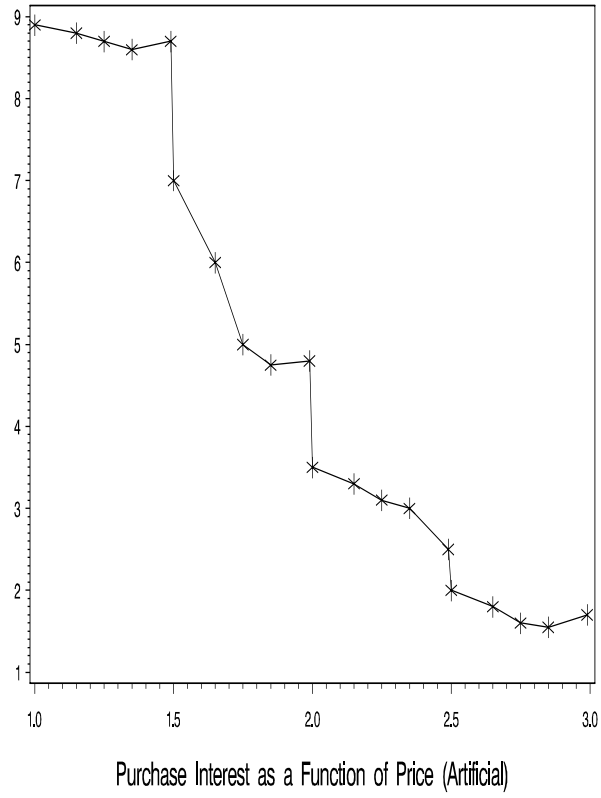


Figure 18. Cell Means, 19 df

rapidly; the slope increases as the function increases. Over the range \$2.50 — \$2.99, the function is nearly flat. At \$1.99, \$2.49, and \$2.99 there is a slight increase in purchase interest.

In Figure 16, there is a knot in the middle of each range. This gives the spline more freedom to follow the data. Figure 17 uses the same model as Figure 16, but monotonicity is imposed. When monotonicity is imposed the curves touch fewer of the data values, passing in between the nonmonotonic points. In Figure 18, the means for each price are plotted and connected. This analysis uses the most model *df* and is the least smooth of the plots.

Benefits of Splines

In marketing research and conjoint analysis, the use of spline models can have several benefits. Whenever a factor has three or more levels and an inherent ordering of the levels, that factor can be modeled as a quadratic monotone spline. The *df* used by the variable is controlled at data analysis time; it is not simply the number of categories minus one. When the alternative is a model in which a factor is designated as nominal, splines can be used to fit a more restrictive model with fewer model *df*. Since the spline model has fewer *df*, it should yield more reproducible results.

The opposite alternative is also important. Consider a variable with many values, like price in some examples. Instead of using a restrictive single *df* linear model, splines can be used to fit a more general model with more *df*. The more general model may show information in the data that is not apparent from the ordinary linear model. This can be a benefit in conjoint analyses that focus on price, in the analysis of scanner data, and in survey research. Splines give you the ability to examine the nonlinearities that may be very important in predicting consumer behavior.

Fitting quadratic and cubic polynomial models is common in marketing research. Splines extend that capability by adding the possibility of knots and hence more general curves. Spline curves can also be restricted to be monotone. Monotone splines are less restrictive than a line and more restrictive than splines that can have both positive and negative slopes. You are no longer restricted to fitting just a line, polynomial, or a step function. Splines give you possibilities in between.

Conclusions

Splines allow you to fit curves to your data. Splines may not revolutionize the way you analyze data, but they will provide you with some new tools for your data analysis toolbox. These new tools allow you to try new methods for solving old problems and tackle new problems that could not be adequately solved with your old tools. We hope you will find these tools useful, and we hope that they will help you to better understand your marketing data.

Graphical Scatter Plots of Labeled Points

Warren F. Kuhfeld

Abstract

The %PlotIt (PLOT ITeratively) macro creates graphical scatter plots of labeled points. It is designed to make it easy to display raw data, regressions, and the results of many other data analyses. You can draw curves, vectors, and circles, and you can control the colors, sizes, fonts, and general appearance of the plots. The %PlotIt macro is a part of the SAS autocall library.*

Introduction

SAS has provided software for producing scatter plots for many years (for example, the PLOT and GPLOT procedures). For many types of data analyses, it is useful to have each point in the plot labeled with the value of a variable. However, before the creation of the %PlotIt macro, there was not a satisfactory way to do this. PROC GPLOT produces graphical scatter plots. Combined with the Annotate facility, it allows long point labels, but it does not provide any way to optimally position them. The PLOT procedure can optimally position long point labels in the scatter plot, however PROC PLOT cannot create a graphical scatter plot. The PROC PLOT label-placement algorithm was developed by Kuhfeld (1991), and the PROC PLOT options are documented in Base SAS documentation.

The macro, %PlotIt (PLOT ITeratively), creates graphical scatter plots of labeled points. It can fit curves, draw vectors, and draw circles. It has many options, but only a small number are needed for many types of plots. The %PlotIt macro uses DATA steps and multiple procedures, including PLOT and GANNO. The %PlotIt macro is over 5500 lines long, so it is not printed here. It is fully documented starting on page 753 and in the header comments. This article illustrates through examples some of the main features of the %PlotIt macro.

*This chapter originally appeared in the SAS Journal **Observations**, Fourth Quarter, 1994, pages 23–37. This version of the chapter has been updated for SAS Version 8.2. Copies of this chapter (TS-722K) and all of the macros are available on the web http://support.sas.com/techsup/tnote/tnote_stat.html#market.

An Overview of the %PlotIt Macro

The `%PlotIt` macro performs the following steps.

1. It reads an input data set and preprocesses it. The preprocessed data set contains information such as the axis variables, the point-symbol and point-label variables, and symbol and label types, sizes, fonts, and colors. The nature of the preprocessing depends on the type of data analysis that generated the input data set. For example, if the option DATATYPE=MDPREF was specified with an input data set created by PROC PRINQUAL for a multidimensional preference analysis, then the `%PlotIt` macro creates blue points for `_TYPE_ = 'SCORE'` observations and red vectors for `_TYPE_ = 'CORR'` observations.

2. A DATA step, using the DATA Step Graphics Interface, determines how big to make the graphical plot.

3. PROC PLOT determines where to position the point labels. The results are sent to output SAS data sets using ODS. By default, if some of the point label characters are hidden, the `%PlotIt` macro recreates the printer plot with a larger line and page size, and hence creates more cells and more room for the labels.

4. The PROC PLOT output data sets are read, and information from them and the preprocessed data set are combined to create an Annotate data set. The label position information is read from the PROC PLOT output, and all of the symbol, size, font, and color information is extracted from the preprocessed data set. The Annotate data set contains all of the instructions for drawing the axes, ticks, tick marks, titles, point symbols, point labels, axis labels, and so on.

5. The Annotate data set is displayed with the GANNO procedure. The `%PlotIt` macro does not use PROC GPLOT.

With the `%PlotIt` macro, you can:

- display plots and create graphics stream files and `gout=` entries
- easily display the results of correspondence analysis, multidimensional preference analysis, preference mapping, multidimensional scaling, regression analysis, and density estimation
- use single or multi-character symbols and control their color, font, and size
- use multi-character point labels and control their color, font, and size
- use fixed, variable, and random colors, and use colors to display changes in a third dimension
- automatically determine a good line size, page size, and list of point label placements
- automatically equate the axes for all devices
- control the colors, sizes, fonts, and general appearance of all aspects of the plot
- pre- and post-process the data
- specify many `goptions`

Since `%PlotIt` is a macro, you can modify it, change the defaults, add new options, and so on. The `%PlotIt` macro is heavily commented to make it easier for you to modify it to suit your needs. There is substantial error checking and options to print intermediate results for debugging when you do not get the results you expect. Furthermore, you have complete access to all of the data sets it creates, including the preprocessed version of the input and the Annotate data set. You can modify the results by editing the Annotate and preprocessed data sets.

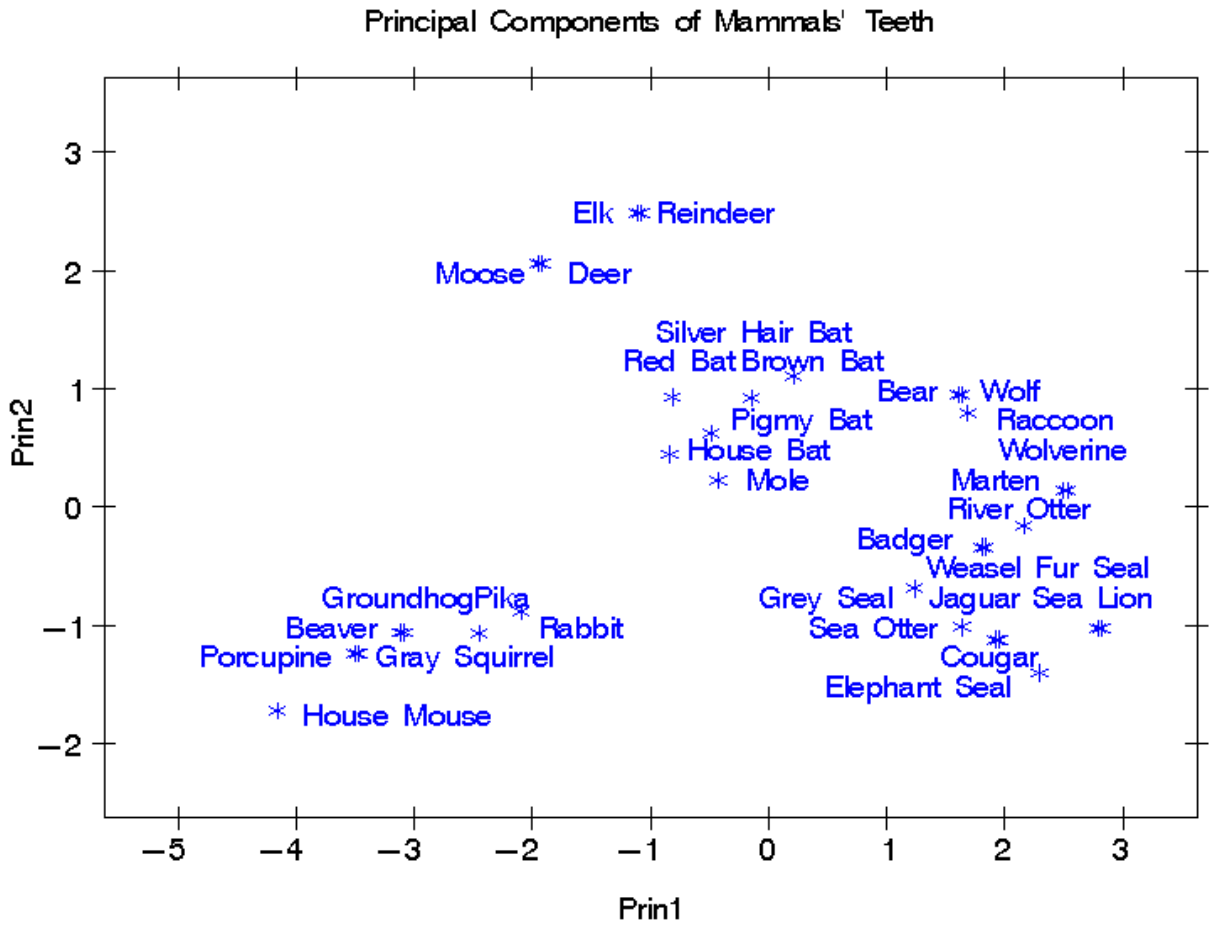


Figure 1

Examples

This section contains examples of some of the capabilities of the `%PlotIt` macro. Rather than interpreting the plots or discussing the details of the statistical analyses, this section concentrates on showing what the `%PlotIt` macro can do. Most of the examples are based on SAS/STAT example data sets. Data for all of the examples can be found in the SAS/STAT sample program `plotitex.sas`.

Example 1: Principal Components of Mammal's Teeth

Principal component analysis computes a low-dimensional approximation to a set of data. Principal components are frequently displayed graphically. This example is based on the Mammal's Teeth data set. To perform a principal component analysis, specify:

```
proc princomp data=teeth out=scores(keep=prin1 prin2 mammal);
   title "Principal Components of Mammals' Teeth";
   run;

%plotit()
```

The plot is shown in Figure 1. No options were specified in the `%PlotIt` macro, so by default a plot

is constructed from the first two numeric variables and the last character variable in the last data set created. The %PlotIt macro printed the following information to the log:

```
                Iterative Scatter Plot of Labeled Points Macro

          Iteration    Place    Line Size    Page Size    Penalty
          ------------------------------------------------------
              1          2         65           45           34
              2          3         80           50            0

The following code will create the printer plot on which the graphical
plot is based:

options nonumber ls=80 ps=50;
proc plot nolegend formchar='|----|+|---' data=preproc vtoh=2;
   plot Prin2 * Prin1 $ mammal = _symbol_ /
        haxis=by 1 vaxis=by 1 box list=1
        placement=((h=2 -2 : s=right left) (v=1 to 2 by alt * h=0 -1 to -10
        by alt));
   label Prin2 = '#' Prin1 = '#';
   run; quit;

The plot was created with the following goptions:

goptions reset=goptions erase hpos=99 vpos=34 hsize=11.71in vsize=7.98in
         device=WIN;

The OUT=anno Annotate data set has 148 observations.
The PLOTIT macro used 1.7seconds to create OUT=anno.
```

The iteration table shows that the %PlotIt macro tried twice to create the plot, with line sizes of 65 and 80. It stopped when all point label characters were plotted with zero penalty.[†] The %PlotIt macro displays PROC PLOT code for the printer plot, on which the graphical plot is based. It also displays the goptions statement that was used with PROC GANNO.[‡]

There are several notable features of the plot in Figure 1.

1. Symbols for several pairs of points, such as Elk and Reindeer, are coincident. By default, the %PlotIt macro slightly offsets coincident symbols so that it is clear that more than one point maps to the same location.
2. Point labels map into discrete rows, just as they would in a printer plot produced by PROC PLOT. However, unlike printer plots, the %PlotIt macro uses proportional fonts.
3. Symbols are not restricted to fixed cells. Their mapping is essentially continuous, more like PROC GPLOT's than PROC PLOT's.
4. A fixed distance represents the same data range along both axes, which means the axes are equated so that distances and angles will have meaning. In contrast, procedures such as PLOT and GPLOT fill the available space by default, so the axes are not equated.

[†]Penalties accrue when point labels are nonoptimally placed, such as when two label characters map to the same location. PROC PLOT tries to minimize the penalties for all point labels. See PROC PLOT documentation for more information.

[‡]This code could be different depending on your device. By default, the macro uses your default device.

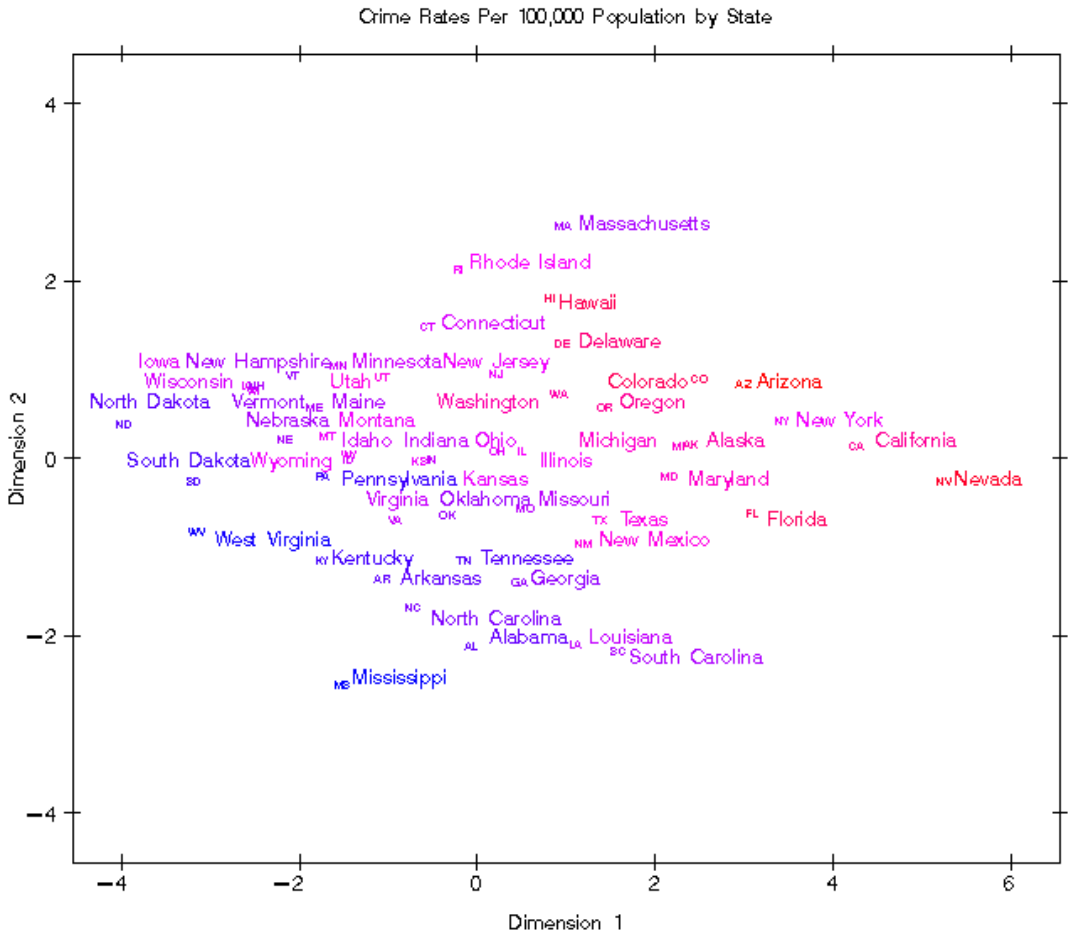


Figure 2

Example 2: Principal Components of Crime Rates

A typical plot has for each point a single-character symbol and a multi-character label; however, this is not required. This example is based on the Crime data set. The point labels are state names, and the symbol for each label is a two-character postal code.

```
proc princomp data=crime out=crime2;
   title 'Crime Rates Per 100,000 Population by State';
   run;

%plotit(data=crime2,plotvars=prin2 prin1,
        symvar=postcode,symlen=2,symsize=0.6,paint=larceny,
        labelvar=state,label=typical)
```

This plot request specifies:

- the input data set: `crime2`
- the y-axis and x-axis variables: `prin2` and `prin1`
- the symbol variable: `postcode`
- the number of symbol characters: 2
- the size of the symbol font in the plot: 0.6

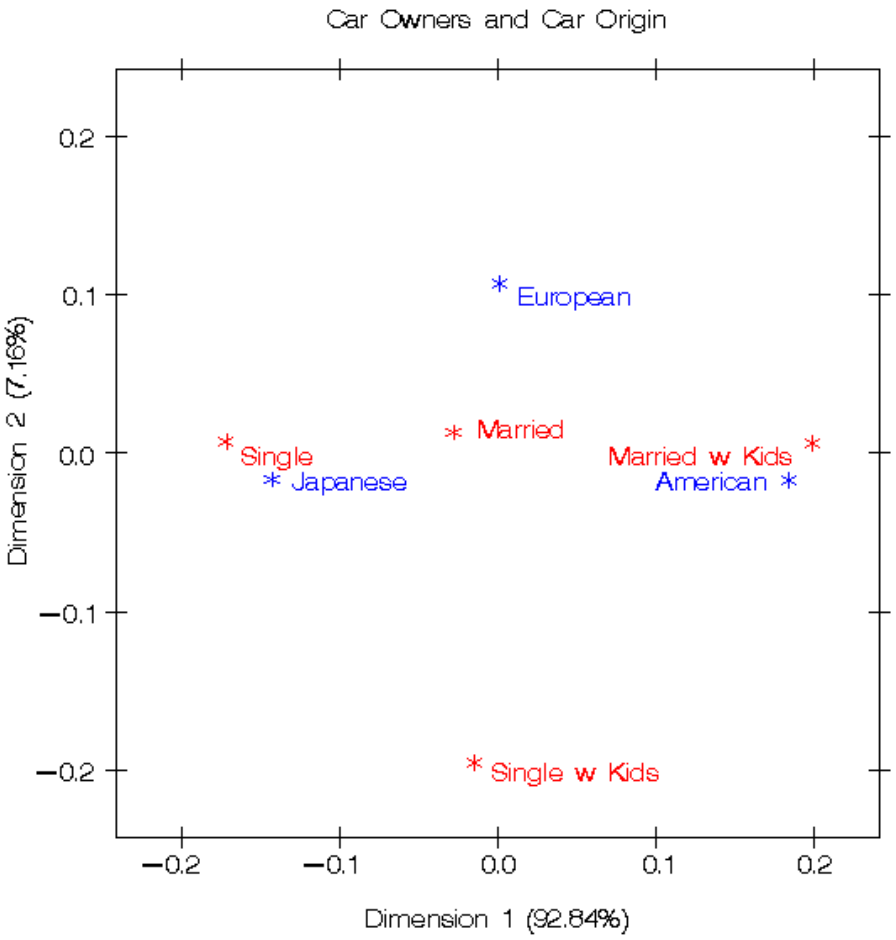


Figure 3

- the colors are based on the variable: `larceny`
- the point label variable: `state`
- the typical method of generating variable labels for the plot axes

A symbol size of 0.6 instead of the normal 1.0 is specified to make the symbol small, because two characters are mapping to a location where there is usually just one. The option `paint=larceny` creates interpolated label and symbol colors, by default between blue, magenta, and red, so that states that have a low larceny rate are blue and high-rate states are red. `Label=typical` for variables `prin2` and `prin1` generates the following label statement:

```
label prin2 = 'Dimension 2' prin1 = 'Dimension 1';
```

This plot request is much more complicated than most. Often, you need to specify only the type of analysis that generated the data set. The plot is shown in Figure 2.

Examples 3 & 4: Correspondence Analysis of the Car Ownership Survey
Correspondence analysis graphically displays crosstabulations. These examples use the Car Survey data. To perform a correspondence analysis, specify:

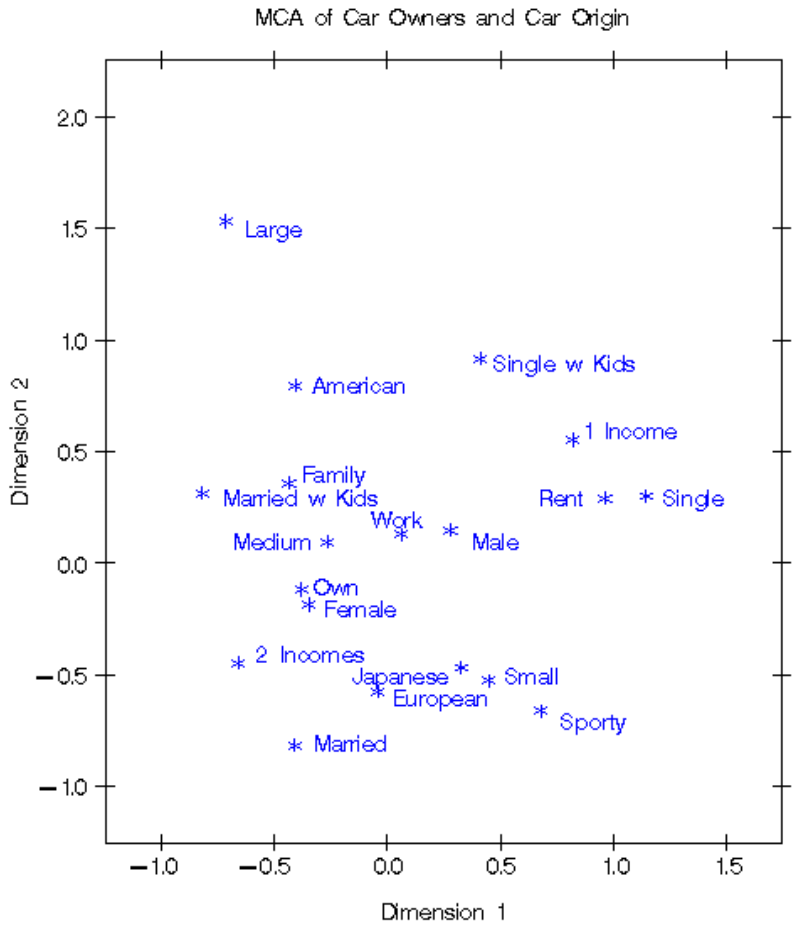


Figure 4

```
proc corresp data=cars outc=coors;
   title 'Car Owners and Car Origin';
   tables marital, origin;
   run;

%plotit(data=coors,datatype=corresp)
```

The plot is shown in Figure 3. With `datatype=corresp`, the `%PlotIt` macro automatically incorporates the proportion of inertia[§] into the axis labels and plots the row points in red and the column points in blue.

For a multiple correspondence analysis, specify:

```
proc corresp mca observed data=cars outc=coors;
   title 'MCA of Car Owners and Car Origin';
   tables origin size type income home marital sex;
   run;

%plotit(data=coors,datatype=mca)
```

The plot is shown in Figure 4.

[§]Inertia in correspondence analysis is analogous to variance in principal component analysis.

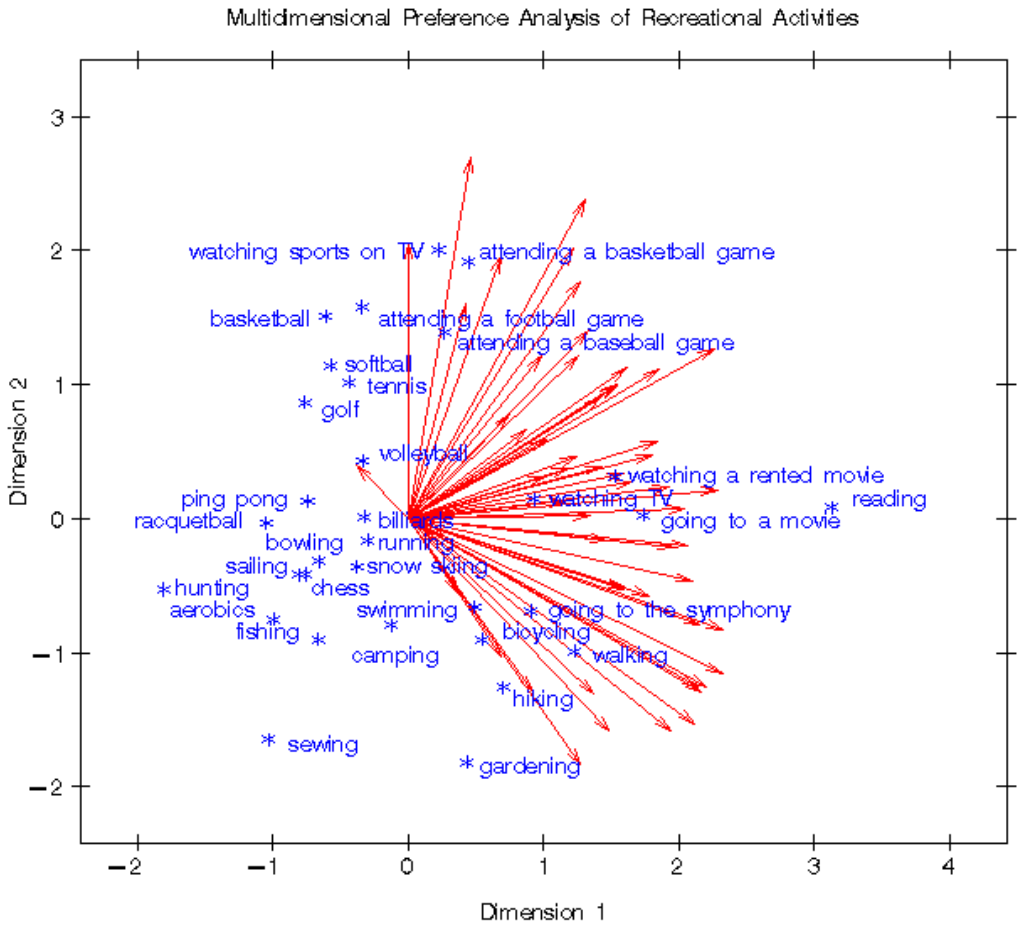


Figure 5

Examples 5 & 6: Multidimensional Preference Analysis of Recreational Activities
Multidimensional preference analysis is a variant on principal component analysis that simultaneously displays people and their preferences for objects. Each person is a variable in the input data set, and each object is a row. Each person is represented in the plot as a vector that points in approximately the direction of his or her most preferred objects. These examples use the Preferences for Recreational Activities data set. For multidimensional preference analysis, specify:

```
proc prinqual cor data=recreate out=rec score std rep;
   title1 'Multidimensional Preference Analysis of Recreational Activities';
   transform identity(sub1-sub56);
   id activity;
   run;

%plotit(data=rec,datatype=mdpref 3)
```

The plot is shown in Figure 5. With `datatype=mdpref`, the `%PlotIt` macro automatically displays the people as vectors and the activities as points (based on the `_TYPE_` variable). The `3` after the MDPREF is a scaling factor for the vectors. The lengths of all the vectors are increased by a factor of three to create a better graphical display. You can also label the vectors by specifying:

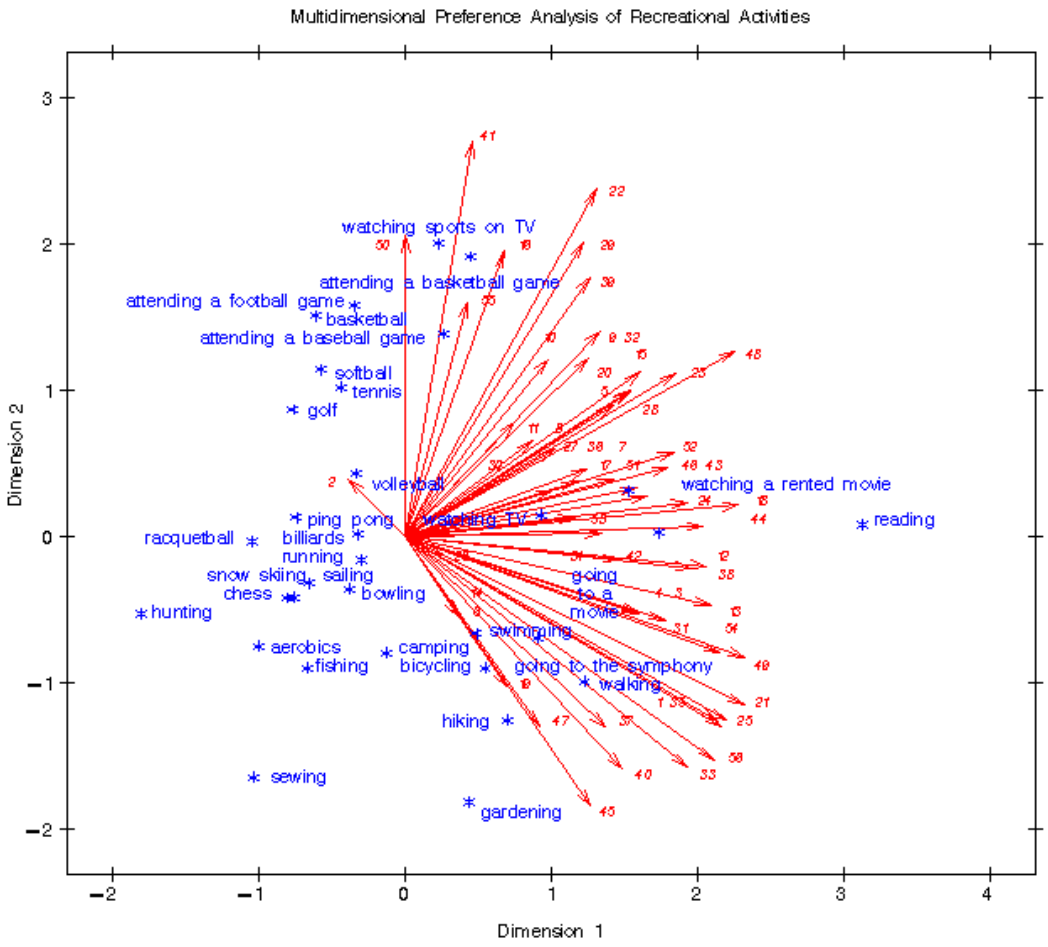


Figure 6

```
%plotit(data=rec,datatype=mdpref2 3)
```

MDPREF2 specifies a MDPREF analysis with labeled vectors (the `2` means labels *too*). This plot is not shown because in this input data set, each subject is identified by a variable name of the form `sub1`, `sub2`, ..., and the graphical display looks cluttered with all those `sub`'s. The default point label variable is the ID statement variable `activity`, because it is the last character variable in the data set. PROC PRINQUAL fills in this variable for the `_TYPE_ = 'CORR'` observations (the people that plot as vectors) with the variable names: `sub1-sub56`. You can preprocess the input data set directly in the `%PlotIt` macro to remove the `sub`'s as follows:

```
%plotit(data=rec,datatype=mdpref2 3,
        adjust1=%str(if _type_ = 'CORR' then
                       activity = substr(activity,4);))
```

The plot is shown in Figure 6. The `adjust1` option adds DATA step statements to the end of the preprocessing step. By default, the `%PlotIt` macro tries to position the vector labels outward, not between the vector head and the origin.

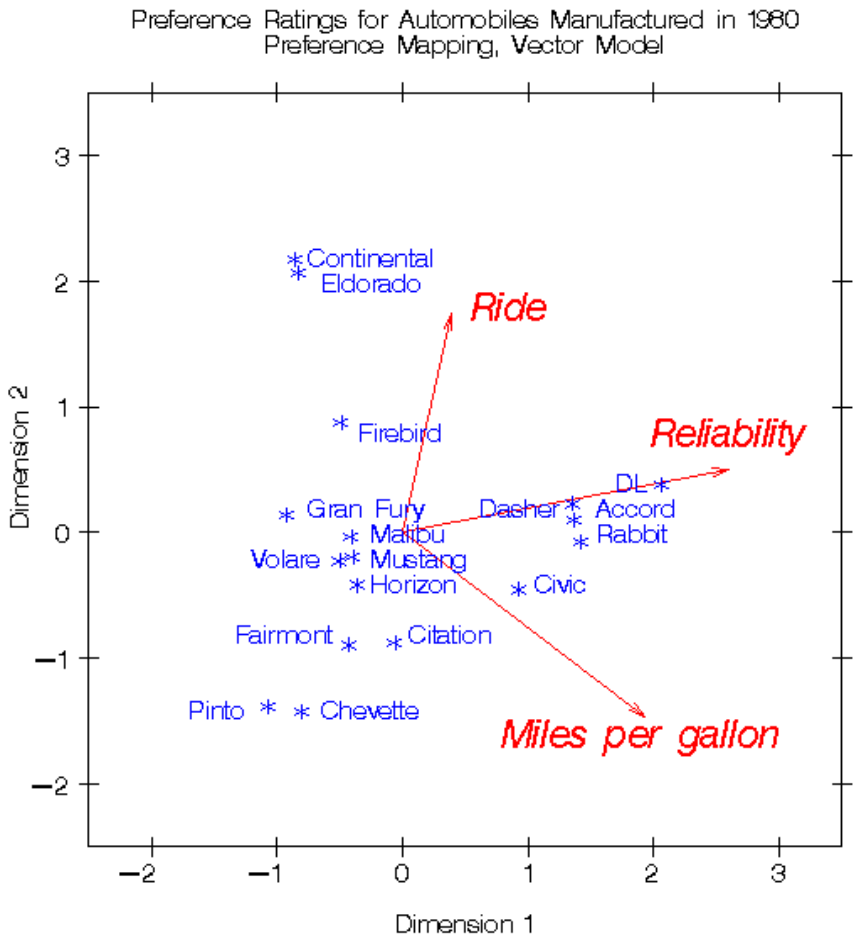


Figure 7

Examples 7 & 8: Preference Mapping of Cars

Preference mapping simultaneously displays objects and attributes of those objects. These examples use the Car Preference data set to illustrate preference mapping. The following code fits a preference mapping vector model:

```
*---Compute Coordinates for a 2-Dimensional Scatter plot of Cars---;
proc prinqual data=carpref out=presults(drop=judge1-judge25) n=2
              replace standard scores;
   title 'Preference Ratings for Automobiles Manufactured in 1980';
   id model mpg reliable ride;
   transform ide(judge1-judge25);
   run;
*---Compute Endpoints for Vectors---;
proc transreg data=presults;
   title2 'Preference Mapping, Vector Model';
   model ide(mpg reliable ride)=identity(prin1 prin2);
   output tstandard=center coefficients replace out=vector;
   id model;
   run;
```

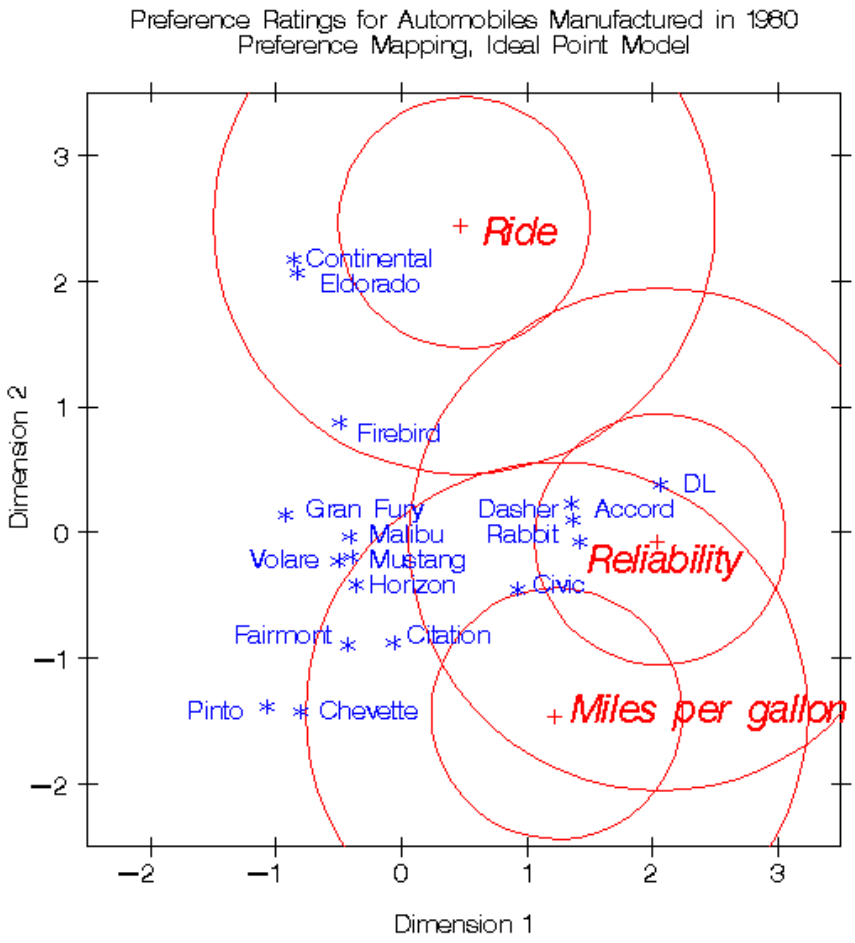


Figure 8

```
%plotit(data=vector,datatype=vector 2.5)
```

The plot is shown in Figure 7. Each attribute is represented as a vector that points in approximately the direction of the objects with larger values of the attribute. The `datatype=vector 2.5` option requests the vector model, with the vectors stretched by a factor of 2.5. Alternatively, you can represent attributes as points by specifying:

```
*---Compute Ideal Point Coordinates---;
proc transreg data=presults;
   title2 'Preference Mapping, Ideal Point Model';
   model identity(mpg reliable ride)=point(prin1 prin2);
   output tstandard=center coordinates replace out=ideal;
   id model;
   run;
%plotit(data=ideal,datatype=ideal,antiidea=1)
```

The plot is shown in Figure 8. The option `datatype=ideal` requests a preference mapping, with each attribute represented as an ideal point. Circles are drawn to show distances between the cars and the ideal points, which are locations of hypothetical cars that have the ideal amount of the attribute.

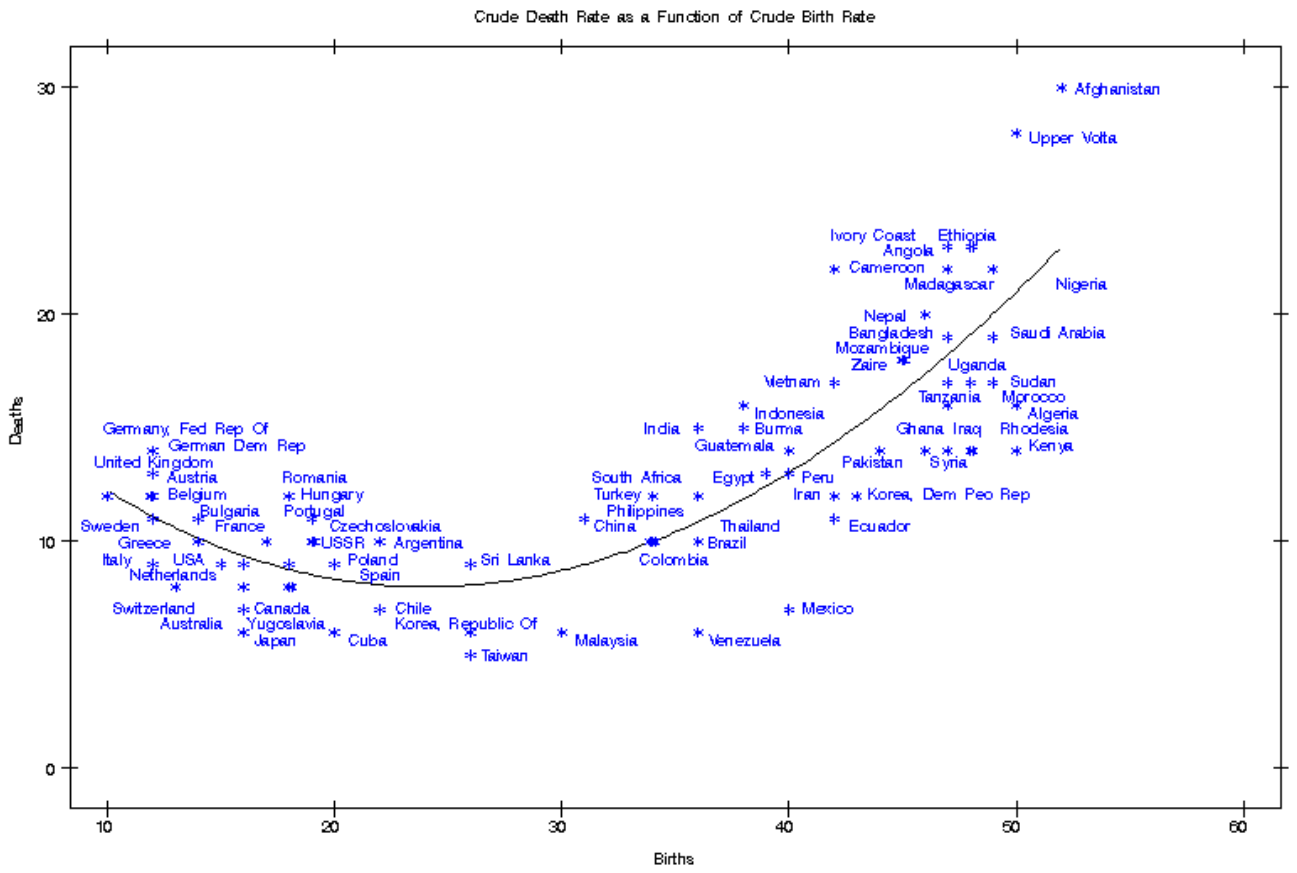


Figure 9

The `antiidea=1` option specifies how anti-ideal points are recognized.¶ By default, the labels for the attributes are larger than the other point labels and hence sometimes extend slightly beyond the plot. This happened with "Miles per gallon" in Figure 8. You can move the label up one character unit and to the left 12 character units by adding the following option:

```
adjust4=%str(if text =: 'Miles' then do; y = y + 1; x = x - 12; end;)
```

The `adjust4` option adds DATA step statements to the end of the final Annotate DATA step. The `%PlotIt` macro has no sense of esthetics; sometimes a little human intervention is needed for the final production plots.

Examples 9 & 10: Curve Fitting of Birth and Death Rates

It is often useful to display a set of points along with a regression line or nonlinear function. The `%PlotIt` macro can fit and display lines and curves (and optionally print the regression and ANOVA table). These examples use the Vital Statistics data set. The following requests a cubic-polynomial regression function:

¶Anti-ideal points have their signs wrong, so the macro must reverse them before plotting. When small ratings are good, specify `antiidea=-1`, and when small ratings are not good, specify `antiidea=1`.

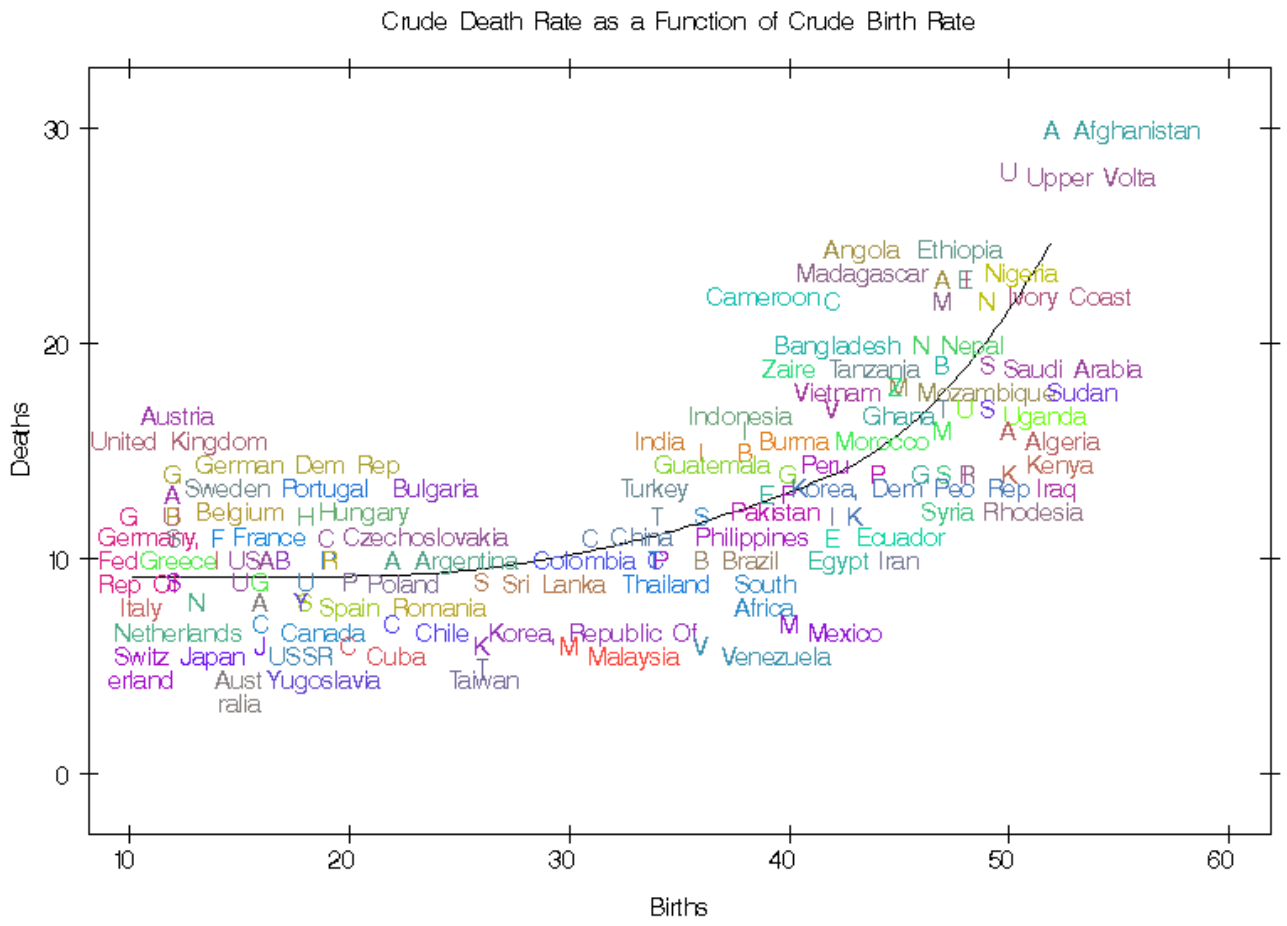


Figure 10

```
title 'Crude Death Rate as a Function of Crude Birth Rate';

%plotit(data=vital,vtoh=1.75,datatype=curve2)
```

The plot is shown in Figure 9. The option `vtoh=1.75` specifies the PROC PLOT aspect ratio (vertical to horizontal). The default is 2.0. Smaller values create plots with more cells for label characters, which is helpful when the point cloud is relatively dense. The option `datatype=curve2` instructs the `%PlotIt` macro to fit a curve and have the point labels avoid the curve (the `2` means label avoidance *too*).

You can control the type of curve. The `%PlotIt` macro uses PROC TRANSREG to fit the curve, and you can specify PROC TRANSREG options. For example, to request a monotone spline regression function with two knots, specify:

```
%plotit(data=vital,datatype=curve,bright=128,maxiter=4,
        symvar=country,regfun=mspline,nknots=2)
```

The plot is shown in Figure 10. There are several differences between Figures 9 and 10, in addition to the difference in the regression function. The option `datatype=curve` was specified, not `datatype=curve2`, so there is more overlap between the point labels and the curve. For each point in the plot, the plotting

symbol is the first letter of the country and the point label is the country. Each label/symbol pair is a different random color with brightness (average RGB or red-green-blue value) of 128. These options make it much easier to find the symbol that corresponds to each label. Also, the default `vtoh=2` was used to decrease the number of cells and make the labels larger. With these data, the penalty sum at iteration four is eight. Specifying `maxiter=4` prevents the algorithm from reaching iteration 5, which prevents the line size from increasing from 125 to 150. This also makes the labels larger. The price is that some label characters collide (for example, "Germany" and "S") and the plot looks more cluttered because there are fewer cells with white space.

Availability

If your site has installed the autocall libraries supplied by SAS and uses the standard configuration of SAS supplied software, you need only to ensure that the SAS system option `mautosource` is in effect to begin using autocall macros. That is, the macros do *not* have to be included (for example with a `%include` statement). They can be called directly. For more information about autocall libraries, refer to *SAS Macro Language: Reference*, pages 597–599, and your host documentation.

Base SAS and SAS/GRAPH software are required to run the `%PlotIt` macro. The `datatype=curve` and `datatype=curve2` options use PROC TRANSREG, which is in SAS/STAT. All of the other `datatype=` values assume an input data set in a form created by a SAS/STAT procedure.

Conclusions

The `%PlotIt` macro provides a convenient way to display the results from many types of data analyses. Usually, only a small number of options are needed; the macro does the rest. The `%PlotIt` macro does not replace procedures like GPLOT and PLOT. Instead, it makes it easy to generate many types of plots that are extremely difficult to produce with standard procedures.

Graphical Methods for Marketing Research

Warren F. Kuhfeld

Abstract

Correspondence analysis, multiple correspondence analysis, preference mapping, and multidimensional preference analysis are descriptive statistical methods that generate graphical displays from data matrices. These methods are used by marketing researchers to investigate relationships among products and individual differences in preferences for those products. The end result is a two- or three-dimensional scatter plot that shows the most salient information in the data matrix. This chapter describes these methods, shows examples of the graphical displays, and discusses marketing research applications.*

Introduction

Correspondence analysis (CA), multiple correspondence analysis (MCA), preference mapping (PREFMAP), and multidimensional preference analysis (MDPREF) are descriptive statistical methods that generate graphical displays from data matrices. These methods are sometimes referred to as perceptual mapping methods. They simultaneously locate two or more sets of points in a single plot, and all emphasize presenting the geometry of the data. CA simultaneously displays in a scatter plot the row and column labels from a two-way contingency table or crosstabulation constructed from two categorical variables. MCA simultaneously displays in a scatter plot the category labels from more than two categorical variables. MDPREF displays both the row labels (products) and column labels (people) from a data matrix of continuous variables. PREFMAP shows rating scale data projected into a plot of row labels—for example, from an MDPREF analysis. These methods are used by marketing researchers to investigate relationships among products and individual differences in preferences for those products.

This chapter will only discuss these techniques as methods of generating two-dimensional scatter plots. However, three-dimensional and higher-dimensional results can also be generated and displayed with modern interactive graphics software and with scatter plot matrices.

*This chapter is a revision of a paper that was published in the 1992 National Computer Graphics Association Conference Proceedings. Copies of this chapter (TS-722L) and all of the macros are available on the web http://support.sas.com/techsup/tnote/tnote_stat.html#market.

Methods

This section presents the algebra and example plots for MDPREF, PREFMAP, CA, and MCA. These methods are all similar in spirit to the biplot, which is discussed first to provide a foundation for the other methods.

The Biplot. A *biplot* (Gabriel 1981) simultaneously displays the rows and columns of a data matrix in a low-dimensional (typically two-dimensional) plot. The "bi" in "biplot" refers to the *joint* display of rows and columns, not to the dimensionality of the plot. Consider an $(n \times m)$ data matrix $\mathbf{Y}$, an $(n \times q)$ matrix $\mathbf{A}$ with row vectors $\mathbf{a}'_1$, $\mathbf{a}'_2$, ..., $\mathbf{a}'_n$, and an $(m \times q)$ matrix $\mathbf{B}$ with row vectors $\mathbf{b}'_1$, $\mathbf{b}'_2$, ..., $\mathbf{b}'_m$. The n rows of $\mathbf{A}$ correspond to the rows of $\mathbf{Y}$, and the m columns of $\mathbf{B}'$ correspond to the columns of $\mathbf{Y}$. The rank of $\mathbf{Y}$ is $q \leq \text{MIN}(n, m)$. $\mathbf{A}$ and $\mathbf{B}$ are chosen such that $y_{ij} = \mathbf{a}'_i\mathbf{b}_j$. If $q = 2$ and the rows of $\mathbf{A}$ and $\mathbf{B}$ are plotted in a two-dimensional scatter plot, the scalar product of the coordinates $\mathbf{a}'_i$ and $\mathbf{b}'_j$ *exactly* equals the data value y_{ij}. This kind of scatter plot is a biplot; it geometrically shows the algebraic relationship $\mathbf{AB}' = \mathbf{Y}$. Typically, the row coordinates are plotted as points, and the column coordinates are plotted as vectors.

When $q > 2$ and two dimensions are plotted, then $\mathbf{a}'_i\mathbf{b}_j$ is *approximately* equal to y_{ij}, and the display is an *approximate biplot.*[*] The approximate biplot geometrically shows the algebraic relationship $\mathbf{AB}' \approx \mathbf{Y}$. The best values for $\mathbf{A}$ and $\mathbf{B}$, in terms of minimum squared error in approximating $\mathbf{Y}$, are found using a singular value decomposition (SVD),[†] $\mathbf{Y} = \mathbf{AB}' = \mathbf{UDV}'$, where $\mathbf{D}$ is a $(q \times q)$ diagonal matrix and $\mathbf{U}'\mathbf{U} = \mathbf{V}'\mathbf{V} = \mathbf{I}_q$, a $(q \times q)$ identity matrix. Solutions for $\mathbf{A}$ and $\mathbf{B}$ include $\mathbf{A} = \mathbf{U}$ and $\mathbf{B} = \mathbf{VD}$, or $\mathbf{A} = \mathbf{UD}$ and $\mathbf{B} = \mathbf{V}$, or more generally $\mathbf{A} = \mathbf{UD}^r$ and $\mathbf{B} = \mathbf{VD}^{(1-r)}$, for $0 \leq r \leq 1$. See Gabriel (1981) for more information on the biplot.

Multidimensional Preference Analysis. Multidimensional Preference Analysis (Carroll 1972) or MDPREF is a biplot analysis for preference data. Data are collected by asking respondents to rate their preference for a set of objects. Typically in marketing research, the objects are products−the client's products and the competitors'. Questions that can be addressed with MDPREF analyses include: Who are my customers? Who else should be my customers? Who are my competitors' customers? Where is my product positioned relative to my competitors' products? What new products should I create? What audience should I target for my new products?

For example, consumers can be asked to rate their preference for a group of automobiles on a 0 to 9 scale, where 0 means no preference and 9 means high preference. $\mathbf{Y}$ is an $(n \times m)$ matrix that contains ratings of the n products by the m consumers. The data are stored as the transpose of the typical data matrix, since the columns are the people. The goal is to produce a plot with the cars represented as points and the consumers represented as vectors. Each person's vector points in *approximately* the direction of the cars that the person most preferred and away from the cars that are least preferred.

Figure 1 contains an example in which 25 consumers rated their preference for 17 new (at the time) 1980 automobiles. This plot is based on a principal component model. It differs from a proper biplot of $\mathbf{Y}$ due to scaling factors. In principal components, the columns in data matrix $\mathbf{Y}$ are standardized to mean zero and variance one. The SVD is $\mathbf{Y} = \mathbf{UDV}'$, and the principal component model is $\mathbf{Y} = ((n-1)^{1/2}\mathbf{U})\,((n-1)^{-1/2}\mathbf{D})\,(\mathbf{V}')$. The standardized principal component scores matrix, $\mathbf{A} = (n-1)^{1/2}\mathbf{U}$, and the component structure matrix, $(n-1)^{-1/2}\mathbf{DV}'$, are plotted. The advantage of creating a biplot based on $(n-1)^{1/2}\mathbf{U}$ and $(n-1)^{-1/2}\mathbf{DV}'$ instead of $\mathbf{U}$ and $\mathbf{DV}'$ is that the coordinates

[*]In practice, the term biplot is sometimes used without qualification to refer to an approximate biplot.

[†]SVD is sometimes referred to in the psychometric literature as an Eckart-Young (1936) decomposition.

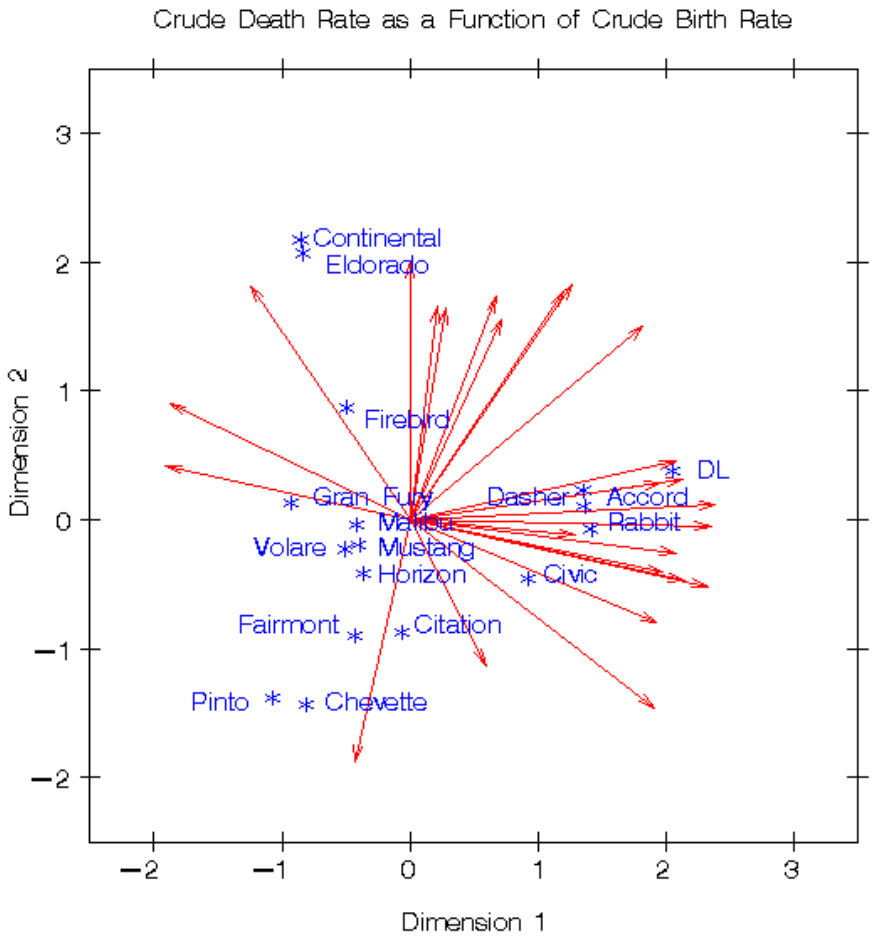


Figure 1. Multidimensional Preference Analysis

do not get smaller as sample size increases. The fit, or proportion of the variance in the data accounted for by the first two dimensions, is the sum of squares of the first two elements of $(n-1)^{-1/2}\mathbf{D}$ divided by the sum of squares of all of the elements of $(n-1)^{-1/2}\mathbf{D}$.

The dimensions of the MDPREF biplot are the first two principal components. The first principal component represents the information that is most salient to the preference judgments. At one end of the plot of the first principal component are the most preferred automobiles; the least preferred automobiles are at the other end of the plot. The second principal component represents the direction that is most salient to the preference judgments that is orthogonal to the first principal component. The automobile point coordinates are the scores of the automobile on the first two principal components. The judge vectors point in *approximately* the direction of judges most preferred cars, with preference increasing as the vector moves from the origin.

Let $\mathbf{a}_i'$ be row i of $\mathbf{A} = (n-1)^{1/2}\mathbf{U}$, $\mathbf{b}_j'$ be row j of $\mathbf{B} = (n-1)^{-1/2}\mathbf{VD}$, $\|\mathbf{a}_i\|$ be the length of $\mathbf{a}_i$, $\|\mathbf{b}_j\|$ be the length of $\mathbf{b}_j$, and θ be the angle between the vectors $\mathbf{a}_i$ and $\mathbf{b}_j$. The predicted degree of (scaled) preference that an individual judge has for an automobile is $\mathbf{a}_i'\mathbf{b}_j = \|\mathbf{a}_i\|\,\|\mathbf{b}_j\| cos\theta$. Each car point can be orthogonally projected onto each judge's vector. The projection of the ith car on the jth judge vector is $\mathbf{b}_j((\mathbf{a}_i'\mathbf{b}_j)/(\mathbf{b}_j'\mathbf{b}_j))$, and the length of this projection is $\|\mathbf{a}_i\| cos\theta$. The automobile that

projects farthest along a judge vector has the highest predicted preference. The length of this projection, $\|\mathbf{a}_i\| cos\theta$, differs from the predicted preference, $\|\mathbf{a}_i\| \, \|\mathbf{b}_j\| cos\theta$, only by $\|\mathbf{b}_j\|$, which is constant within each judge. Since the goal is to look at projections of points onto the vectors, the absolute length of a judge's vector is unimportant. The relative lengths of the vectors indicate fit, with longer vectors indicating better fit. The coordinates for the endpoints of the vectors were multiplied by 2.5 to extend the vectors and create a better graphical display. The direction of the preference scale is important. The vectors point in the direction of increasing values of the data values. If the data had been ranks, with 1 the most preferred and n the least preferred, then the vectors would point in the direction of the least preferred automobiles.

The people in the top left portion of the plot most prefer the large American cars. Other people, with vectors pointing up and nearly vertical, also show this pattern of preference. There is a large cluster of people who prefer the Japanese and European cars. A few people, most notably the person whose vector passes through the "e" in "Chevette", prefer the small and inexpensive American Cars. There are no vectors pointing through the bottom left portion of the of the plot, which suggests that the smaller American cars are generally not preferred by anyone within this group.

The first dimension, which is a measure of overall evaluation, discriminates between the American cars on the left and the Japanese and European cars on the right. The second dimension seems to reflect the sizes of the automobiles. Some cars have a similar pattern of preference, most notably Continental and Eldorado, which share a symbol in the plot. Marketers of Continental or Eldorado may want to try to distinguish their car from the competition. Dasher, Accord, and Rabbit were rated similarly, as were Malibu, Mustang, Volare, and Horizon.

This 1980 example is quite prophetic even though it is based on a small nonrandom sample. Very few vectors point toward the smaller American cars, and Mustang is the only one of them that is still being made. Many vectors are pointing toward the European and Japanese cars, and they are still doing quite well in the market place. Many vectors are pointing in the one to two o'clock range where there are no cars in the plot. One can speculate that these people would prefer Japanese and European luxury cars such as Accura, Lexus, Infinity, BMW, and Mercedes.

Preference Mapping. Preference mapping[‡] (Carroll 1972) or PREFMAP plots resemble biplots, but are based on a different model. The goal in PREFMAP is to take a set of coordinates for a set of objects, such as the MDPREF car coordinates in example in Figure 1, and project in external information that can aid in interpreting the configuration of points. Questions that can be addressed with PREFMAP analyses include: Where is my product positioned relative to my competitors' products? Why is my product positioned there? How can I reposition my existing products? What new products should I create?

The Preference Mapping Vector Model. Figure 2 contains an example in which three attribute variables (ride, reliability, and miles per gallon) are displayed in the plot of the first two principal components of the car preference data. Each of the automobiles was rated on these three dimensions on a 1 to 5 scale, where 1 is poor and 5 is good. Figure 2 is based on the simplest version of PREFMAP−the *vector model*. The vector model assumes that some is good and more is *always* better. This model is appropriate for miles per gallon and reliability−the more miles a motorist can travel without refueling or breaking down, the better. The end points for the attribute vectors are obtained by projecting the attribute variables into the car space. If the attribute ratings are stored in matrix $\mathbf{R}$, then the coordinates for the end points are in the matrix $\boldsymbol{\beta}$ from the multivariate linear regression model $\mathbf{R} =$

[‡]Preference mapping is sometimes referred to as external unfolding.

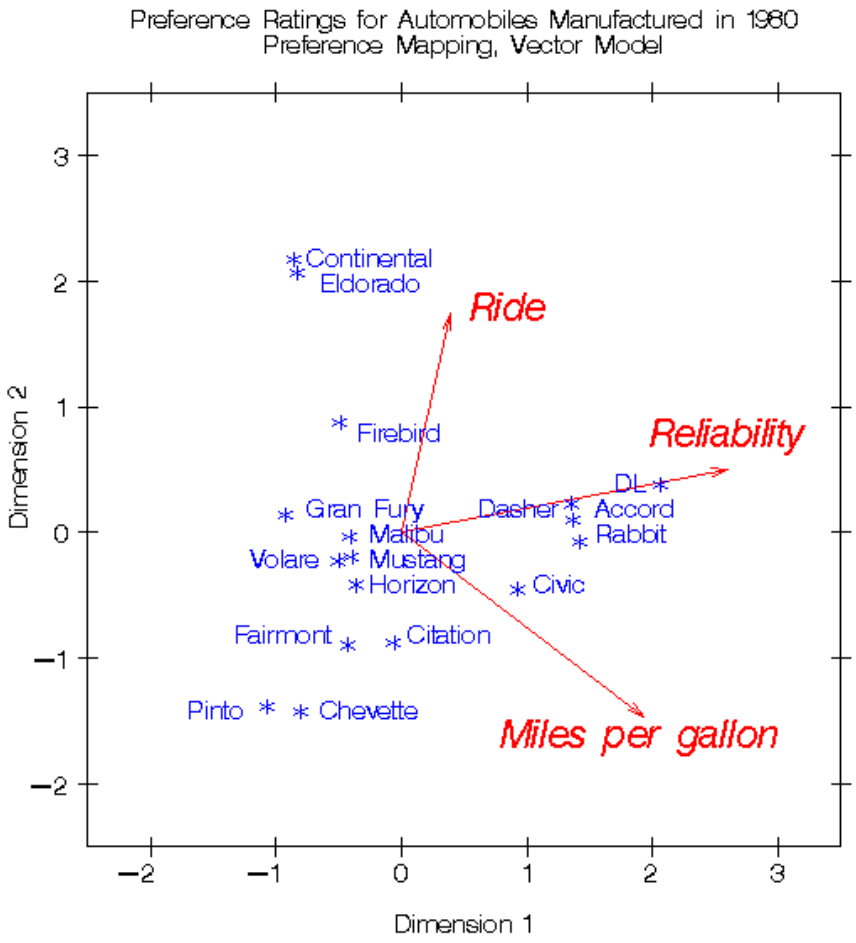


Figure 2. Preference Mapping, Vector Model

$\mathbf{A}\boldsymbol{\beta} + \boldsymbol{\epsilon}$. $\mathbf{A}$ is the matrix of standardized principal component scores, or $\mathbf{A}$ could be the coordinates from a multidimensional scaling analysis. The relative lengths of the vectors indicate fit, which is given by the R^2. As with MDPREF, the lengths of all vectors can be scaled by the same constant to make a better graphical display.

PREFMAP analyses can help in the interpretation of principal component, multidimensional scaling, and MDPREF analyses by projecting in external information that helps explain the configuration. Orthogonal projections of the product points on an attribute vector give an *approximate* ordering of the products on the attribute. The ride vector points almost straight up showing that the larger cars, such as the Eldorado and Continental, have the best ride. In Figure 1, it was shown that most people preferred the DL, Japanese cars, and larger American cars. Figure 2 shows that the DL and Japanese cars were rated as the most reliable and have the best fuel economy. The small American cars are not rated highly on any of the three dimensions, although some are on the positive end of miles per gallon.

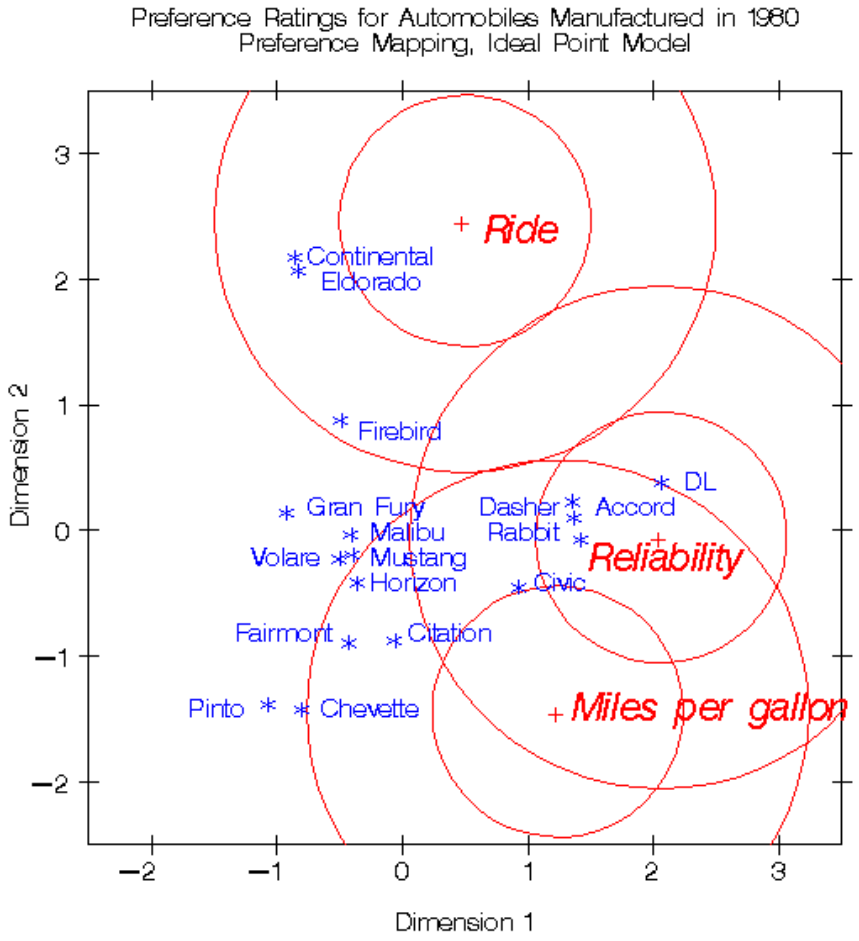


Figure 3. Preference Mapping, Ideal Point Model

The Preference Mapping Ideal Point Model. The *ideal point* model differs from the vector model in that the ideal point model does not assume that more is better, *ad infinitum*. Consider the sugar content of cake. There is an ideal amount of sugar that cake should contain—not enough sugar is not good, and too much sugar is also not good. In the current example, the ideal number of miles per gallon and the ideal reliability are unachievable. It makes sense to consider a vector model, because the ideal point is infinitely far away. This argument is less compelling for ride; the point for a car with smooth, quiet ride may not be infinitely far away. Figure 3 shows the results of fitting an ideal point model for the three attributes. In the vector model, results are interpreted by orthogonally projecting the car points on the attribute vectors. In the ideal point model, Euclidean distances between car points and ideal points are compared. Eldorado and Continental have the best predicted ride, because they are closest to the ride ideal point. The concentric circles drawn around the ideal points help to show distances between the cars and the ideal points. The numbers of circles and their radii are arbitrary. The overall interpretations of Figures 2 and 3 are the same. All three ideal points are at the edge of the car points, which suggests the simpler vector model is sufficient for these data.

The ideal point model is fit with a multiple regression model and some pre- and post-processing. First the $\mathbf{A}$ matrix is augmented by a variable that is the sum of squares of columns of $\mathbf{A}$ creating $\mathbf{A}^*$. Then solve for $\boldsymbol{\beta}$ from $\mathbf{R} = \mathbf{A}^*\boldsymbol{\beta} + \boldsymbol{\epsilon}$. For a two-dimensional scatter plot, the ideal point coordinates are

given by dividing each coefficient for the two axes by the coefficient for the sum-of-squares variable, then multiplying the resulting values by −0.5. The coordinates are $-0.5\,\boldsymbol{\beta}\,\mathrm{diag}(\boldsymbol{\beta}_3)^{-1}$, where $\mathrm{diag}(\boldsymbol{\beta}_3)$ is a diagonal matrix constructed from the third row of $\boldsymbol{\beta}$. This is a constrained response-surface model. The fit is given by the R^2. See Carroll (1972) for the justification for the formula.

The results in Figure 3 were modified from the raw results to eliminate *anti-ideal points*. The ideal point model is a distance model. The rating data are interpreted as distances between attribute ideal points and the products. In this example, each of the automobiles was rated on these three dimensions, on a 1 to 5 scale, where 1 is poor and 5 is good. The data are the reverse of what they should be−a ride rating of 1 should mean this car is similar to a car with a good ride, and a rating of 5 should mean this car is different from a car with a good ride. So the raw coordinates must be multiplied by −1 to get ideal points. Even if the scoring had been reversed, anti-ideal points can occur. If the coefficient for the sum-of-squares variable is negative, the point is an anti-ideal point. In this example, there is the possibility of *anti-anti-ideal points*. When the coefficient for the sum-of-squares variable is negative, the two multiplications by −1 cancel, and the coordinates are ideal points. When the coefficient for the sum-of-squares variable is positive, the coordinates are multiplied by −1 to get an ideal point.

Other PREFMAP Models. The ideal point model presented here is based on an ordinary Euclidean distance model. All points falling on a circle centered around an ideal point are an equal distance from the ideal point. Two more PREFMAP models are sometimes used. The more general models allow for differential weighting of the axes and rotations, so ellipses, not circles, show equal weighted distances. All three ideal point models are response surface models. See Carroll (1972) for more information.

Correspondence Analysis. Correspondence analysis (CA) is a weighted SVD of a contingency table. It is used to find a low-dimensional graphical representation of the association between rows and columns of a table. Each row and column is represented by a point in a Euclidean space determined from cell frequencies. Like MDPREF, CA is based on a singular value decomposition, but ordinary SVD of a contingency table does not portray a desirable geometry.

Questions that can be addressed with CA and MCA include: Who are my customers? Who else should be my customers? Who are my competitors' customers? Where is my product positioned relative to my competitors' products? Why is my product positioned there? How can I reposition my existing products? What new products should I create? What audience should I target for my new products?

CA is a popular data analysis method in France and Japan. In France, CA was developed under the strong influence of Jean-Paul Benzécri; in Japan, under Chikio Hayashi. CA is described in Lebart, Morineau, and Warwick (1984); Greenacre (1984); Nishisato (1980); Tenenhaus and Young (1985); Gifi (1990); Greenacre and Hastie (1987); and many other sources. Hoffman and Franke (1986) provide a good introductory treatment using examples from marketing research.

Simple CA. This section is primarily based on the theory of CA found in Greenacre (1984). Let $\mathbf{N}$ be an $(n_r \times n_c)$ contingency table of rank $q \leq MIN(n_r, n_c)$. Let $\mathbf{1}$ be a vector of ones of the appropriate order, $\mathbf{I}$ be an identity matrix, and diag() be a matrix-valued function that creates a diagonal matrix from a vector. Let $f = \mathbf{1'N1}$, $\mathbf{P} = (1/f)\mathbf{N}$, $\mathbf{r} = \mathbf{P1}$, $\mathbf{c} = \mathbf{P'1}$, $\mathbf{D_r} = \mathrm{diag}(\mathbf{r})$, and $\mathbf{D_c} = \mathrm{diag}(\mathbf{c})$. The scalar f is the sum of all elements in $\mathbf{N}$. $\mathbf{P}$ is a matrix of relative frequencies. The vector $\mathbf{r}$ contains row marginal proportions or row *masses*. The vector $\mathbf{c}$ contains column marginal proportions or column masses. $\mathbf{D_r}$ and $\mathbf{D_c}$ are diagonal matrices of marginals. The coordinates of the CA are based on the generalized singular value decomposition of $\mathbf{P}$, $\mathbf{P} = \mathbf{AD_uB'}$, where $\mathbf{A'D_r}^{-1}\mathbf{A} = \mathbf{B'D_c}^{-1}\mathbf{B} = \mathbf{I}$. $\mathbf{A}$

is an $(n_r \times q)$ matrix of left generalized singular vectors, $\mathbf{D_u}$ is a $(q \times q)$ diagonal matrix of singular values, and $\mathbf{B}$ is an $(n_c \times q)$ matrix of right generalized singular vectors. The first (trivial) column of $\mathbf{A}$ and $\mathbf{B}$ and the first singular value in $\mathbf{D_u}$ are discarded before any results are displayed. This step centers the table and is analogous to centering the data in ordinary principal component analysis. In practice, this centering is done by subtracting ordinary chi-square expected values from $\mathbf{P}$ before the SVD. The columns of $\mathbf{A}$ and $\mathbf{B}$ define the principal axes of the column and row point clouds, respectively. The fit, or proportion of the *inertia* (analogous to variance) in the data accounted for by the first two dimensions, is the sum of squares of the first two singular values, divided by the sum of squares of all of the singular values. Three sets of coordinates are typically available from CA, one based on rows, one based on columns, and the usual set is based on both rows and columns.

The *row profile* (conditional probability) matrix is defined as $\mathbf{R} = \mathbf{D_r}^{-1}\mathbf{P} = \mathbf{D_r}^{-1}\mathbf{A}\mathbf{D_u}\mathbf{B}'$. Each (i, j) element of $\mathbf{R}$ contains the observed probability of being in column j given membership in row i. The values in each row of $\mathbf{R}$ sum to one. The row coordinates, $\mathbf{D_r}^{-1}\mathbf{A}\mathbf{D_u}$, and column coordinates, $\mathbf{D_c}^{-1}\mathbf{B}$, provide a CA based on the row profile matrix. The *principal* row coordinates, $\mathbf{D_r}^{-1}\mathbf{A}\mathbf{D_u}$, and *standard* column coordinates, $\mathbf{D_c}^{-1}\mathbf{B}$, provide a decomposition of $\mathbf{D_r}^{-1}\mathbf{A}\mathbf{D_u}\mathbf{B}'\mathbf{D_c}^{-1} = \mathbf{D_r}^{-1}\mathbf{P}\mathbf{D_c}^{-1} = \mathbf{R}\mathbf{D_c}^{-1}$. Since $\mathbf{D_r}^{-1}\mathbf{A}\mathbf{D_u} = \mathbf{R}\mathbf{D_c}^{-1}\mathbf{B}$, the row coordinates are weighted centroids of the column coordinates. Each column point, with coordinates scaled to standard coordinates, defines a vertex in $(n_c - 1)$-dimensional space. All of the principal row coordinates are located in the space defined by the standard column coordinates. Distances among row points have meaning, but distances among column points and distances between row and column points are not interpretable.

The formulas for the analysis of the *column profile* matrix can easily be derived by applying the row profile formulas to the transpose of $\mathbf{P}$. The principal column coordinates $\mathbf{D_c}^{-1}\mathbf{B}\mathbf{D_u}$ are weighted centroids of the standard row coordinates $\mathbf{D_r}^{-1}\mathbf{A}$. Each row point, with coordinates scaled to standard coordinates, defines a vertex in $(n_r - 1)$-dimensional space. All of the principal column coordinates are located in the space defined by the standard row coordinates. Distances among column points have meaning, but distances among row points and distances between row and column points are not interpretable.

The usual sets of coordinates[§] are given by $\mathbf{D_r}^{-1}\mathbf{A}\mathbf{D_u}$ and $\mathbf{D_c}^{-1}\mathbf{B}\mathbf{D_u}$. One advantage of using these coordinates is that both sets are postmultiplied by the diagonal matrix $\mathbf{D_u}$, whose diagonal values are all less than or equal to one. When $\mathbf{D_u}$ is a part of the definition of only one set of coordinates, that set forms a tight cluster near the centroid, whereas the other set of points is more widely dispersed. Including $\mathbf{D_u}$ in both sets makes a better graphical display. However, care must be taken in interpreting such a plot. No correct interpretation of distances between row points and column points can be made. Less specific statements, such as "two points are on the same side of the plot" have meaning.

Another property of this choice of coordinates concerns the geometry of distances between points within each set. Distances between row (or column) profiles are computed using a *chi-square metric*. The rationale for computing distances between row profiles using the non-Euclidean chi-square metric is as follows. Each row of the contingency table may be viewed as a realization of a multinomial distribution conditional on its row marginal frequency. The null hypothesis of row and column independence is equivalent to the hypothesis of homogeneity of the row profiles. A significant chi-square statistic is geometrically interpreted as a significant deviation of the row profiles from their centroid, $\mathbf{c}'$. The chi-square metric is the Mahalanobis metric between row profiles based on their estimated covariance matrix under the homogeneity assumption (Greenacre and Hastie 1987). A parallel argument can be made for the column profiles.

[§]This set is often referred to as the French standardization due to its popularity in France.

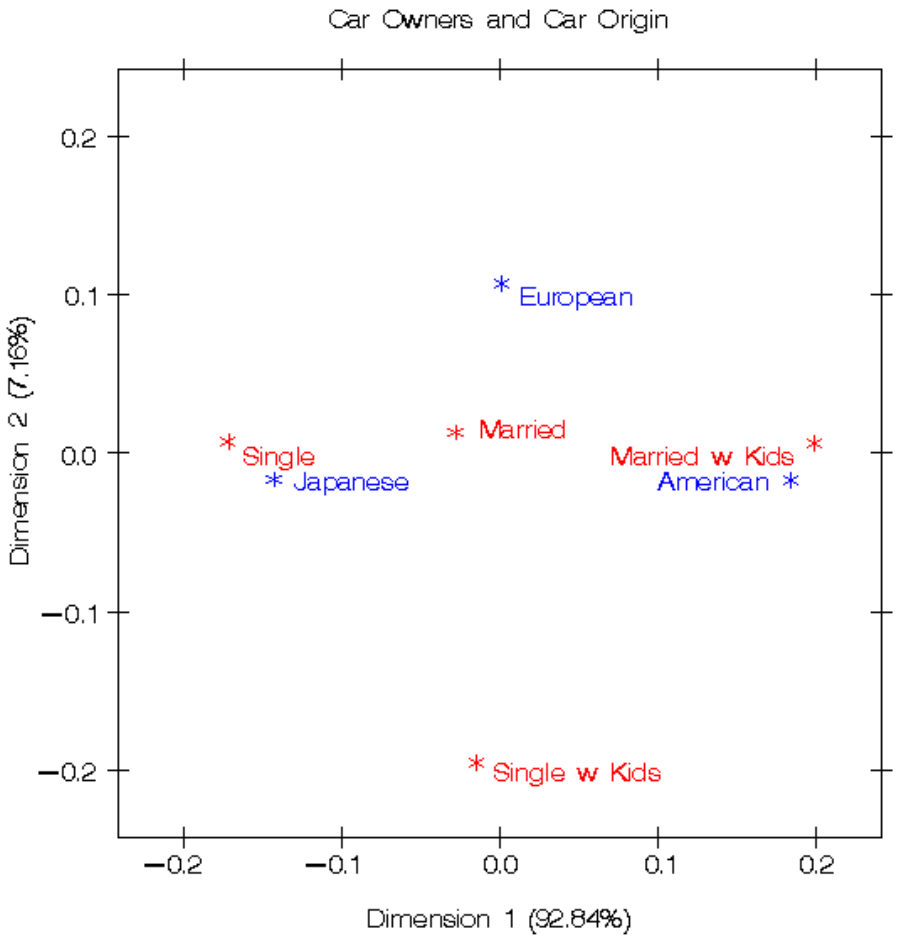


Figure 4. Simple Correspondence Analysis

The row coordinates are $\mathbf{D_r}^{-1}\mathbf{A}\mathbf{D_u} = \mathbf{D_r}^{-1}\mathbf{A}\mathbf{D_u}\mathbf{B}'\mathbf{D_c}^{-1}\mathbf{B} = (\mathbf{D_r}^{-1}\mathbf{P})(\mathbf{D_c}^{-1/2})(\mathbf{D_c}^{-1/2}\mathbf{B})$. They are row profiles $\mathbf{D_r}^{-1}\mathbf{P}$ rescaled by $\mathbf{D_c}^{-1/2}$ (rescaled so that distances between profiles are transformed from a chi-square metric to a Euclidean metric), then orthogonally rotated with $\mathbf{D_c}^{-1/2}\mathbf{B}$ to a principal axes orientation. Similarly, the column coordinates are column profiles rescaled to a Euclidean metric and orthogonally rotated to a principal axes orientation.

CA Example. Figure 4 contains a plot of the results of a simple CA of a survey of car owners. The questions included origin of the car (American, Japanese, European), and marital/family status (single, married, single and living with children, and married living with children). Both variables are categorical. Table 1 contains the crosstabulation and the observed minus expected frequencies. It can be seen from the observed minus expected frequencies that four cells have values appreciably different from zero (Married w Kids/American, Single/American, Married w Kids/Japanese, Single/Japanese). More people who are married with children drive American cars than would be expected if the rows and columns are independent, and more people who are single with no children drive Japanese cars than would be expected if the rows and columns are independent.

Table 1
Simple Correspondence Example Input

	Contingency Table			Observed Minus Expected Values		
	American	European	Japanese	American	European	Japanese
Married	37	14	51	-1.5133	0.4602	1.0531
Married w Kids	52	15	44	10.0885	0.2655	-10.3540
Single	33	15	63	-8.9115	0.2655	8.6460
Single w Kids	6	1	8	0.3363	-0.9912	0.6549

CA graphically shows the information in the observed minus expected frequencies. The right side of Figure 4 shows the association between being married with children and owning an American Car. The left side of the plot shows the association between being single and owning a Japanese Car. This interpretation is based on points being located in approximately the same direction from the origin and in approximately the same region of the space. Distances between row points and column points are not defined.

Multiple Correspondence Analysis. Multiple correspondence analysis (MCA) is a generalization of simple CA for more than two variables. The input is a *Burt table*, which is a partitioned symmetric matrix containing all pairs of crosstabulations among a set of categorical variables. Each diagonal partition is a diagonal matrix containing marginal frequencies (a crosstabulation of a variable with itself). Each off-diagonal partition is an ordinary contingency table. Each contingency table above the diagonal has a transposed counterpart below the diagonal. A Burt table is the inner product of a partitioned design matrix. There is one partition per categorical variable, and each partition is a binary design matrix. Each design matrix has one column per category, and a single 1 in each row. The partitioned design matrix has exactly m ones in each row, where m is the number of categorical variables. The results of a MCA of a Burt table, $\mathbf{N}$, are the same as the column results from a simple CA of the design matrix whose inner product is the Burt table. MCA is not a simple CA of the Burt table. The coordinates for MCA are $\mathbf{D_c}^{-1}\mathbf{B}\mathbf{D_u}$, from $(1/f)\mathbf{N} = \mathbf{P} = \mathbf{P}' = \mathbf{B}\mathbf{D_u}^2\mathbf{B}'$, where $\mathbf{B}'\mathbf{D_c}^{-1}\mathbf{B} = \mathbf{I}$.

MCA Example. Figure 5 contains a plot of the results of an MCA of a survey of car owners. The questions included origin of the car (American, Japanese, European), size of car (small, medium, large), type of car (family, sporty, work vehicle), home ownership (owns, rents), marital/family status (single, married, single and living with children, and married living with children), and sex (male, female). The variables are all categorical.

The top-right quadrant of the plot shows that the categories single, single with kids, 1 income, and renting a home are associated. Proceeding clockwise, the categories sporty, small, and Japanese are associated. In the bottom-left quadrant we see the association between being married, owning your own home, and having two incomes. Having children is associated with owning a large American family car. Such information could be used in marketing research to identify target audiences for advertisements. This interpretation is based on points being located in approximately the same direction from the origin

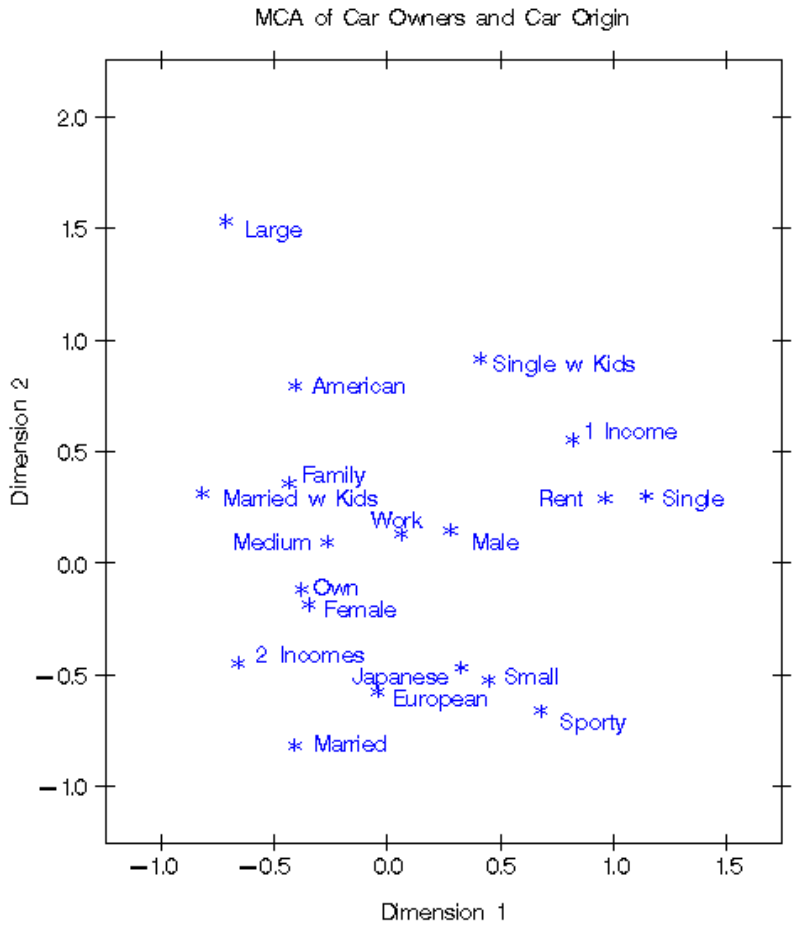


Figure 5. Multiple Correspondence Analysis

and in approximately the same region of the space. Distances between points are not interpretable in MCA.

Other CA Standardizations. Other standardizations have been proposed for CA by several authors. The usual goal is to provide a standardization that avoids the problem of row and column distances being undefined. Unfortunately, this problem remains unsolved. Carroll, Green, and Schaffer (1986) proposed that simple CA coordinates should be transformed to MCA coordinates before they are plotted. They argued that all distances are then comparable, but Greenacre (1989) showed that their assertion was incorrect. Others have also claimed to have discovered a method of defining the between row and column differences, but so far no method has been demonstrated to be correct.

Notes

The Geometry of the Scatter Plots. All of the scatterplots in this chapter were created with the axes equated so that a centimeter on the y-axis represents the same data range as a centimeter on the x-axis. *This is important.* Distances, angles between vectors, and projections are evaluated to interpret the plots. When the axes are equated, distances and angles are correctly represented in the plot. When axes are scaled independently, for example to fill the page, then the correct geometry is not presented. The important step of equating the axes is often overlooked in practice.

In a true biplot, $\mathbf{A} = \mathbf{U}\mathbf{D}^r$ and $\mathbf{B} = \mathbf{V}\mathbf{D}^{(1-r)}$ are plotted, and the elements of $\mathbf{Y}$ can be approximated from $y_{ij} \approx \mathbf{a}_i'\mathbf{b}_j$. For MDPREF and PREFMAP, the absolute lengths of the vectors are not important since the goal is to project points on vectors, not look at scalar products of row points and column vectors. It is often necessary to change the lengths of *all* of the vectors to improve the graphical display. If all of the vectors are relatively short with end points clustered near the origin, the display will not look good. To avoid this problem in Figure 1, *both* the x-axis and y-axis coordinates were multiplied by the constant 2.5, to lengthen all vectors by the same relative amount. The coordinates must not be scaled independently.

Software. All data analyses were performed with Release 8.00 of the SAS System. MDPREF is performed with PROC PRINQUAL, simple and multiple correspondence analysis are performed with PROC CORRESP, and PREFMAP is performed with PROC TRANSREG. The plots are prepared with the SAS `%PlotIt` autocall macro. If your site has installed the autocall libraries supplied by SAS Institute and uses the standard configuration of SAS software supplied by the Institute, you need only to ensure that the SAS system option `mautosource` is in effect to begin using the autocall macros. See pages 597–599 and pages 753–783.

Conclusions

Marketing research helps marketing decision makers understand their customers and their competition. Correspondence analysis compactly displays survey data to aid in determining what kinds of consumers are buying certain products. Multidimensional preference analysis shows product positioning, group preferences, and individual preferences. The plots may suggest how to reposition products to appeal to a broader audience. They may also suggest new groups of customers to target. Preference mapping is used as an aid in understanding MDPREF and multidimensional scaling results. PREFMAP displays product attributes in the same plot as the products. The insight gained from perceptual mapping methods can be a valuable asset in marketing decision making. These techniques can help marketers gain insight into their products, their customers, and their competition.

Concluding Remarks

I hope you like this book and the new macros. In particular, I hope you find the `%MktEx` macro to be very powerful and useful. My goal in writing this book and tool set is to help you do better research and do it more quickly and more easily. I would like to hear what you think. Many of my examples and enhancements to the software are based on feedback from people like you. This book has already been revised many times, and future revisions are likely. If you have comments or suggestions for future revisions, write Warren F. Kuhfeld, (Warren.Kuhfeld@sas.com) at SAS Institute Inc. My goal to provide you with enough examples so that you can easily adapt aspects of one or more examples to fit your particular needs. When I do not succeed, tell me about it and I will try to add a new example to the next revision. Please direct questions to the technical support division and suggestions to me. Please email me. I would like to hear from you!

I would like to put in a plug for the American Marketing Association's Advanced Research Techniques Forum (ART Forum), which is a conference held each year in June. I have been to every one since 1991, although I will probably miss the 2005 meeting due to my daughter's high school graduation. It is a great place to meet academic researchers and top practitioners in the areas of conjoint, choice, and other branches of marketing research. It always draws a diverse and international crowd. There are a number of great tutorials, and most years there is one by Don Anderson and myself on choice designs.

I leave you with this old Irish blessing.

May the road rise up
to meet you
may the wind be
always at your back
May the sun shine warm
on your face
And the rain fall soft
upon your fields

... along with this additional thought ...

May your designs always be efficient
and your standard errors small

The Kuhfeld Conundrum

What do all of the random number seeds used in the "Conjoint Analysis Examples," "Multinomial Logit, Discrete Choice Modeling," and "Experimental Design and Choice Modeling Macros" have in common? Send answers to Warren.Kuhfeld@sas.com. I will send a small prize to the first person to send me the answer that I have in mind. I first put this challenge out there quite a few years ago now. I keep hoping that one of these days, one of you will send me the answer.* Hints: The relationship is not mathematical. An answer like "they are all less than 619," while true, is not what I have in mind. Think about the order of a row, and heed this admonition: stick to the middle of the road.

*If you can't get that one, here is an easy one—no prize for this one though. Find the joke embedded in the index.

References

Addelman, S. (1962a), "Orthogonal Main-Effects Plans for Asymmetrical Factorial Experiments," *Technometrics*, 4, 21–46.

Addelman, S. (1962b), "Symmetrical and Asymmetrical Fractional Factorial Plans," *Technometrics*, 4, 47–58.

Agresti, A. (1990), *Categorical Data Analysis*. New York: John Wiley and Sons.

Anderson, D.A. and Wiley, J.B. (1992), "Efficient Choice Set Designs for Estimating Cross-Effects Models," *Marketing Letters*, 3, 357–370.

Anderson, D.A. (2003), personal communication.

de Boor, C. (1978), *A Practical Guide to Splines*, New York: Springer Verlag.

Bose, R.C. (1947), "Mathematical Theory of the Symmetrical Factorial Design," *Sankhya*, 8, 107–166.

Booth, K.H.V. and Cox, D.R. (1962), "Some Systematic Super-Saturated Designs," *Technometrics*, 4, 489–495.

Breiman, L. and Friedman, J.H. (1985), "Estimating Optimal Transformations for Multiple Regression and Correlation," (with discussion), *Journal of the American Statistical Association*, 77, 580–619.

Breslow, N. and Day, N.E. (1980), *Statistical Methods in Cancer Research, Vol. II: The Design and Analysis of Cohort Studies,* Lyon: IARC.

Bunch, D.S., Louviere, J.J., and Anderson, D.A. (1996), "A Comparison of Experimental Design Strategies for Choice-Based Conjoint Analysis with Generic-Attribute Multinomial Logit Models," Working Paper, Graduate School of Management, University of California at Davis.

van der Burg, E. and de Leeuw, J. (1983), "Non-linear Canonical Correlation," *British Journal of Mathematical and Statistical Psychology*, 36, 54–80.

Carmone, F.J. and Green, P.E. (1981), "Model Misspecification in Multiattribute Parameter Estimation," *Journal of Marketing Research*, 18 (February), 87–93.

Carroll, J.D. (1972), "Individual Differences and Multidimensional Scaling," in *Multidimensional Scaling: Theory and Applications in the Behavioral Sciences (Volume 1)*, in Shepard, R.N., Romney, A.K., and Nerlove, S.B. (ed.), New York: Seminar Press.

Carroll, J.D, Green, P.E., and Schaffer, C.M. (1986), "Interpoint Distance Comparisons in Correspondence Analysis," *Journal of Marketing Research*, 23, 271–280.

Carson, R.T., Louviere, J.J, Anderson, D.A., Arabie, P., Bunch, D., Hensher, D.A., Johnson, R.M., Kuhfeld, W.F., Steinberg, D., Swait, J., Timmermans, H., and Wiley, J.B. (1994), "Experimental Analysis of Choice," *Marketing Letters,* 5(4), 351–368.

Chakravarti, I.M. (1956), "Fractional Replication in Asymmetrical Factorial Designs and Partially Balanced Arrays," *Sankhya*, 17, 143–164.

Chrzan, K. and Elrod, T. (1995), "Partial Profile Choice Experiments: A Choice-Based Approach for Handling Large Numbers of Attributes," paper presented at the AMA's 1995 Advanced Research Techniques Forum, Monterey, CA.

Cook, R.D. and Nachtsheim, C.J. (1980), "A Comparison of Algorithms for Constructing Exact *D*-optimal Designs," *Technometrics*, 22, 315–324.

Cook, R.D. and Nachtsheim, C.J. (1989), "Computer-Aided Blocking of Factorial and Response-Surface Designs," *Technometrics* 31 (August), 339–346.

Coolen, H., van Rijckevorsel, J., and de Leeuw, J. (1982), "An Algorithm for Nonlinear Principal Components with B-splines by Means of Alternating Least Squares," in H. Caussinus, P. Ettinger, and R. Tomassone (ed.), *COMPUSTAT 1982*, Part 2, Vienna: Physica Verlag.

Dawson, J. (1985), "A Construction for Generalized Hadamard Matrices, GH(4q, EA(q))," *Journal of Statistical Planning and Inference*, 11, 103–110.

De Cock, D. and Stufken, J. (2000), "On Finding Mixed Orthogonal Arrays of Strength 2 With Many 2-Level Factors," *Statistics and Probability Letters*, 50, 383–388.

de Launey, W. (1986), "A Survey of Generalized Hadamard Matrices and Difference Matrices D(k, λ; G) with Large k," *Utilitas Mathematica*, 30, 5–29.

de Launey, W. (1987), *(O,G)-Designs and Applications, Dissertation*, University of Sydney.

de Launey, W. (1987), "On Difference Matrices, Transversal Designs, Resolvable Traversal Designs and Large Sets of Mutually Orthogonal F-Squares", *Journal of Statistical Planning and Inference*, 16, 107–125.

Dey, A. (1985), *Orthogonal Fractional Factorial Designs*, New York: Wiley.

DuMouchell, W. and Jones, B. (1994), "A Simple Bayesian Modification of *D*-Optimal Designs to Reduce Dependence on an Assumed Model," *Technometrics* 36 (February), 37–47.

Dykstra, O. (1971), "The Augmentation of Experimental Data to Maximize $|(\mathbf{X}'\mathbf{X})^{-1}|$," *Technometrics*, 13 (August), 682–688.

Eckart, C. and Young, G. (1936), "The Approximation of One Matrix by Another of Lower Rank," *Psychometrika*, 1, 211–218.

Elrod, T., Louviere, J.J, and Davey, K.S. (1992), "An Empirical Comparison of Ratings-Based and Choice-Based Conjoint Models," *Journal of Marketing Research*, 29 (August), 368–377.

Fedorov, V.V. (1972), *Theory of Optimal Experiments*, translated and edited by W.J. Studden and E.M. Klimko, New York: Academic Press.

Finkbeiner, C.T. (1988), "Comparison of Conjoint Choice Simulators," Sawtooth Software Conference Proceedings.

Fisher, R. (1938), *Statistical Methods for Research Workers*, 10th Edition, Edinburgh: Oliver and Boyd Press.

Gabriel, K.R. (1981), "Biplot Display of Multivariate Matrices for Inspection of Data and Diagnosis," *Interpreting Multivariate Data*, V. Barnett (ed.), London: John Wiley and Sons, Inc.

Gail, M.H., Lubin, J.H., and Rubinstein, L.V. (1981), "Likelihood Calculations for Matched Case-control Studies and Survival Studies with Tied Death Times," *Biometrika*, 68, 703–707.

Gifi, A. (1981), *Nonlinear Multivariate Analysis*, Department of Data Theory, The University of Leiden, The Netherlands.

Gifi, A. (1990), *Nonlinear Multivariate Analysis*, New York: Wiley.

Green, P.E. (1974), "On the Design of Choice Experiments involving Multifactor Alternatives," *Journal of Consumer Research*, 1, 61–68.

Green, P.E. and Rao, V.R. (1971), "Conjoint Measurement for Quantifying Judgmental Data," *Journal of Marketing Research*, 8, 355–363.

Green, P.E. and Srinivasan, V. (1990), "Conjoint Analysis in Marketing: New Developments with Implications for Research and Practice," *Journal of Marketing*, 54, 3–19.

Green, P.E. and Wind, Y. (1975), "New Way to Measure Consumers' Judgments," *Harvard Business Review*, July–August, 107–117.

Green, P.E. and Rao, V.R. (1971), "Conjoint Measurement for Quantifying Judgmental Data," *Journal of Marketing Research*, 8, 355–363.

Greenacre, M.J. (1984), *Theory and Applications of Correspondence Analysis*, London: Academic Press.

Greenacre, M.J. (1989), "The Carroll-Green-Schaffer Scaling in Correspondence Analysis: A Theoretical and Empirical Appraisal," *Journal of Market Research*, 26, 358–365.

Greenacre, M.J. and Hastie, T. (1987), "The Geometric Interpretation of Correspondence Analysis," *Journal of the American Statistical Association*, 82, 437–447.

Hadamard, J. (1893), "Resolution d'une Question Relative aux Determinants," *Bull. des Sciences Math*, (2), 17, 240–246.

Hastie, T. and Tibshirani, R. (1986), "Generalized Additive Models," *Statistical Science*, 3, 297–318.

Hedayat, A.S., Sloane, N.J.A., and Stufken, J. (1999), *Orthogonal Arrays*, New York: Springer.

Hoffman, D.L., and Franke, G.R. (1986), "Correspondence Analysis: Graphical Representation of Categorical Data in Marketing Research," *Journal of Marketing Research*, 23, 213–227.

Hoffman, S.D. and Duncan, G.J. (1988), "Multinomial and Conditional Logit Discrete-choice Models in Demography," *Demography,* 25 (3), 415–427.

Huber, J. and Zwerina, K. (1996), "The Importance of Utility Balance in Efficient Choice Designs," *Journal of Marketing Research*, 33, 307–317.

Huber, J., Wittink, D.R., Fiedler, J.A., and Miller, R. (1993), "The Effectiveness of Alternative Preference Elicitation Procedures in Predicting Choice," *Journal of Marketing Research*, 30 (February), 105–114.

Kharaghania, H., and Tayfeh-Rezaiea, B. (2004), "A Hadamard Matrix of Order 428," [http://math.ipm.ac.ir/tayfeh-r/papersandpreprints/h428.pdf].

Kirkpatrick, S., Gellat, C.D., and Vecchi, M.P. (1983), "Optimization by Simulated Annealing," *Science*, 220, 671–680.

Krieger, A.B. and Green, P.E. (1991), "Designing Pareto Optimal Stimuli for Multiattribute Choice Experiments," *Marketing Letters*, 2, 337–348.

Kruskal, J.B. and Wish, M. (1978), *Multidimensional Scaling*, Sage University Paper series on Quantitative Applications in the Social Sciences, 07–011, Beverly Hills and London: Sage Publications.

Kruskal, J.B. and Shepard, R.N. (1974), "A Nonmetric Variety of Linear Factor Analysis," *Psychometrika*, 38, 123–157.

Kuhfeld, W.F. (1991), "A Heuristic Procedure for Label Placement in Scatter Plots," Presented to the joint meeting of the Psychometric Society and Classification Society of North America, Rutgers University, New Brunswick NJ, June 13–16, 1991.

Kuhfeld, W.F. (2005), "Marketing Research Methods in SAS," [http://support.sas.com/techsup/tnote/tnote_stat.html#market].

Kuhfeld, W.F. (1990), *SAS Technical Report R-108: Algorithms for the PRINQUAL and TRANSREG Procedures*, Cary NC: SAS Institute Inc.

Kuhfeld, W.F. (2005), "Difference Schemes via Computerized Searches," *Journal of Statistical Planning and Inference*, 127, 1-2, 341–346.

Kuhfeld, W.F., Tobias, R.D., and Garratt, M. (1994), "Efficient Experimental Design with Marketing Research Applications," *Journal of Marketing Research*, 31, 545–557.

Kuhfeld, W.F. and Garratt, M. (1992), "Linear Models and Conjoint Analysis with Nonlinear Spline Transformations," Paper presented to the American Marketing Association Advanced Research Techniques Forum, Lake Tahoe, Nevada.

Lazari, A.G. and Anderson, D.A. (1994), "Designs of Discrete Choice Set Experiments for Estimating Both Attribute and Availability Cross Effects," *Journal of Marketing Research*, 31, 375–383.

Lebart, L., Morineau, A., and Warwick, K.M. (1984), *Multivariate Descriptive Statistical Analysis: Correspondence Analysis and Related Techniques for Large Matrices*, New York: Wiley.

de Leeuw, J. (1986), "Regression with Optimal Scaling of the Dependent Variable," Department of Data Theory, The University of Leiden, The Netherlands.

de Leeuw, J., Young, F.W., and Takane, Y. (1976), "Additive Structure in Qualitative Data: an Alternating Least Squares Method with Optimal Scaling features," *Psychometrika*, 41, 471–503.

Louviere, J.J. (1988), *Analyzing Decision Making, Metric Conjoint Analysis*, Sage University Papers, Beverly Hills: Sage.

Louviere, J.J. (1991), "Consumer Choice Models and the Design and Analysis of Choice Experiments," Tutorial presented to the American Marketing Association Advanced Research Techniques Forum, Beaver Creek, Colorado.

Louviere, J.J. and Woodworth, G. (1983), "Design and Analysis of Simulated Consumer Choice of Allocation Experiments: A Method Based on Aggregate Data," *Journal of Marketing Research*, 20 (November), 350–367.

Louviere, J.J. (1991), "Consumer Choice Models and the Design and Analysis of Choice Experiments," Tutorial presented to the American Marketing Association Advanced Research Techniques Forum, Beaver Creek, Colorado.

Manski, C.F. and McFadden, D. (1981), *Structural Analysis of Discrete Data with Econometric Applications*. Cambridge: MIT Press.

Mardia, K.V., Kent, J.T., and Bibby, J.M. (1979), Multivariate Analysis, New York: Academic Press.

McFadden, D. (1974), "Conditional logit Analysis Of Qualitative Choice Behavior," in P. Zarembka (ed.) *Frontiers in Econometrics,* New York: Academic Press, 105–142.

McKelvey, R.D. and Zavoina, W. (1975), "A Statistical Model for the Analysis Of Ordinal Level Dependent Variables," *Journal of Mathematical Sociology,* 4, 103–120.

Meyer, R.K. and Nachtsheim, C.J. (1995), "The Coordinate-Exchange Algorithm for Constructing Exact Optimal Experimental Designs," *Technometrics*, 37, 60–69.

Mitchell, T.J. and Miller, F.L. Jr. (1970), "Use of Design Repair to Construct Designs for Special Linear Models," *Math. Div. Ann. Progr. Rept. (ORNL-4661)*, 130–131, Oak Ridge, TN: Oak Ridge National Laboratory.

Mitchell, T.J. (1974), "An Algorithm for the Construction of *D*-optimal Experimental Designs," *Technometrics*, 16 (May), 203–210.

Nishisato, S. (1980), *Analysis of Categorical Data: Dual Scaling and Its Applications*, Toronto: University of Toronto Press.

Paley, R.E.A.C (1933), "On Orthogonal Matrices," *J. Math. Phys*, 12, 311–320.

Perreault, W.D. and Young, F.W. (1980), "Alternating Least Squares Optimal Scaling: Analysis of Nonmetric Data in Marketing Research," *Journal of Marketing Research*, 17, 1–13.

Raktoe, B.L., Hedayat, A.S., and Federer, W.T. (1981), *Factorial Designs*, New York: John Wiley and Sons.

Ramsay, J.O. (1988), "Monotone Regression Splines in Action," *Statistical Science*, 3, 425–461.

Rao, C.R. (1947), "Factorial Experiments Derivable from Combinatorial Arrangements of Arrays," *Journal of the Royal Statistical Society*, Suppl., 9, 128–139.

van Rijckevorsel, J. (1982), "Canonical Analysis with B-splines," in H. Caussinus, P. Ettinger, and R. Tomassone (ed.), *COMPUSTAT 1982*, Part I, Vienna: Physica Verlag.

Schiffman, S.S., Reynolds, M.L., and Young, F.W. (1981), *Introduction to Multidimensional Scaling*, New York: Academic Press.

Sloane, N.J.A. (2004), "A Library of Orthogonal Arrays," [http://www.research.att.com/∼njas/oadir].

Smith, P.L. (1979), "Splines as a Useful and Convenient Statistical Tool," *The American Statistician*, 33, 57–62.

Steckel, J.H., DeSarbo, W.S., and Mahajan, V. (1991), "On the Creation of Acceptable Conjoint Analysis Experimental Designs," *Decision Sciences*, 22, 435–442.

Suen, C.Y. (1989a), "A Class of Orthogonal Main Effects Plans," *Journal of Statistical Planning and Inference*, 21, 391–394.

Suen, C.Y. (1989b), "Some Resolvable Orthogonal Arrays with Two Symbols," *Communications in Statistics, Theory and Methods*, 18, 3875–3881.

Suen, C.-Y. (2003a), "Construction of Mixed Orthogonal Arrays by Juxtaposition," paper in review.

Suen, C.-Y. (2003b), "Table of Orthogonal Arrays," personal communication.

Suen, C.-Y. (2003c), "Some Mixed Orthogonal Arrays Obtained by Orthogonal Projection Matrices," to be submitted.

Suen, C.-Y. and Kuhfeld, W.F. (2005), "Some Mixed Orthogonal Arrays Obtained by Kronecker Sums," to appear in *Journal of Statistical Planning and Inference*.

Taguchi, G. (1987), *System of Experimental Design: Engineering Methods to Optimize Quality and Minimize Costs.* White Plains, NY: UNIPIB, and Dearborn, MI: American Supplier Institute.

Tenenhaus, M. and Young, F.W. (1985), "An Analysis and Synthesis of Multiple Correspondence Analysis, Optimal Scaling, Dual Scaling, Homogeneity Analysis, and Other Methods of Quantifying Categorical Multivariate Data," *Psychometrika*, 50, 91–119.

Wang, J.C. (1996a), "Mixed Difference Matrices and the Construction of Orthogonal Arrays," *Statist. Probab. Lett.*, 28, 121–126.

Wang, J.C. (1996b), *A Recursive Construction of Orthogonal Arrays*, preprint.

Wang, J.C. and Wu, C.F.J. (1989), "An Approach to the Construction of Asymmetrical Orthogonal Arrays," *IIQP Research Report RR-89-01*, University of Waterloo.

Wang, J.C. and Wu, C.F.J. (1991), "An Approach to the Construction of Asymmetrical Orthogonal Arrays," *Journal of the American Statistical Association*, 86, 450–456.

Williamson, J. (1944), "Hadamard's Determinant Theorem and the Sum of Four Squares," Duke Math. J., 11, 65–81.

Winsberg, S. and Ramsay, J.O. (1980), "Monotonic Transformations to Additivity Using Splines," *Biometrika*, 67, 669–674.

Wittink, D.R. and Cattin, P. (1989), "Commercial Use of Conjoint Analysis: An Update," *Journal of Marketing*, 53 (July), 91–96.

Wittink, D.R., Krishnamurthi, L., and Reibstein, D.J. (1989), "The Effect of Differences in the Number of Attribute Levels in Conjoint Results," *Marketing Letters*, *1:2*, 113–123.

Xu, H. (2002), "An Algorithm for Constructing Orthogonal and Nearly Orthogonal Arrays with Mixed Levels and Small Runs," *Technometrics*, 44, 356–368.

Young, F.W. (1981), "Quantitative Analysis of Qualitative Data," *Psychometrika*, 46, 357–388.

Young, F.W. (1987), *Multidimensional Scaling: History, Theory, and Applications*, R.M. Hamer (ed.), Hillsdale, NJ: Lawrence Erlbaum Associates.

Young, F.W., de Leeuw, J., and Takane, Y. (1976), "Regression with Qualitative and Quantitative Variables: an Alternating Least Squares Approach with Optimal Scaling Features," *Psychometrika*, 41, 505–529.

Zhang, Y.S., Lu, Y., and Pang, S. (1999), "Orthogonal Arrays Obtained by Orthogonal Decompositions of Projection Matrices," *Statistica Sinica*, 9, 595–604.

Zhang, Y.S., Pang, S., and Wang, Y. (2001), "Orthogonal Arrays Obtained by Generalized Hadamard Product," *Discrete Mathematics*, 238, 151–170.

Zhang, Y.S., Duan, L., Lu. Y., Zheng, Z. (2002), "Construction of Generalized Hadamard Matrices," *Journal of Statistical Planning and Inference*, 104, 239–258.

Zhang, Y.S., Weiguo, L., Meixia, M., and Zheng, Z. (2005), "Orthogonal Arrays Obtained by Generalized Kronecker Product", in review, *Journal of Statistical Planning and Inference*.

Index

+- defined 686
3 defined 686
512 defined 686 729 746
@@ 146
A-efficiency 53 102
abbreviating option names 585
accept defined 684
accept option 450 678 686 689 707-709
active observations 592
Addelman, S. 110 121 125 668 831
adjust1= defined 779
adjust2= defined 780
adjust3= defined 780
adjust4= defined 780
adjust5= defined 780
Age variable 340
aggregate data 259-262 276-280 332-334 353 357 360 480
Agresti, A. 467 831
algorithm options, TRANSREG 586-587
aliased 50 101
aliasing structure 306
allcode defined 652
alligator 55
allocation study 345-357
allocation study, defined 96
ALLOCS data set 633
Alt variable 178 642 645
Alt variable 739
alt= 164-165 644 739
alt= defined 644 739
alternative, defined 96
alternative-specific attribute, defined 96
alternative-specific effects 222-224 256 265 270 275 285 321 326 469-470 473-475 607 614
Anderson, D.A. 3 47 99 105 113 121 134 285 305 435 829-831 834
Ann 680
anneal= defined 692
annealfun= defined 699
annealing 193 241 676 692 699
anniter= defined 692
antiidea= defined 772
anti-ideal points 28 823
Arabie, P. 831
arithmetic mean 53
arrays 154 169 180-182 206 251-253 261 276 330 351 378 471-473 476
artificial data 141 253 330
asymmetry 229 283 614-625 720-722
attribute levels, order 586
attribute, defined 96
attributes, design 483-485
augmenting an existing design 385
autocall macros 597
availability cross effects 283-286 305 321 328-329 334
availability cross effects, defined 96
available, not 714
average importance 542-544
bad variable 286-287 411 687-688
badness 688
balance 50 54 157 244 316
balance= 299-300 303-304 393 684-688
balance= defined 682
balanced and orthogonal 51 54 66-68 157
balanced incomplete blocks design 435-461 464-465
__bbad variable 688
Benzécri, J.P. 28 823
bestcov= defined 627
bestout= defined 627
beta= 167 203 248 321 367 601 609-610 628
beta= defined 628
Bibby, J.M. 835
BIBD, see balanced incomplete blocks design
big designs 229
big= 649 652 695
big= defined 649 695
bin of attributes 60 164 206
binary coding 64 173 209 218 256 264
biplot 22 818
Block variable 201 347 357 638 645 717
block= 644
block= defined 644
BLOCKED data set 646
blocking 195-197 244-246 296 304 308-311 346 638-647 665 717-718 723-724
blocking, defined 96
blocks= 208
blocks= defined 665 723
blue bus 284

`blue=` defined 778
Booth, K.H.V. 103 831
`border` defined 767
Bose, R.C. 668 831
Bradley, R.A. 38 549 566-570 573-576
Bradley-Terry-Luce model
compared to other simulators 567
defined 566
market share 575
brand choice (aggregate data) example 261
`Brand` variable 164-166 172-173 264 268-269 316 321 324-328 340 351 354 357 613 616 619 632 735-739
branded design, defined 96
`branded` defined 660
Breiman, L. 786 831
Breslow likelihood 282
Breslow, N. 282 468-471 481 831
`brief` 175-176 211 344
`bright=` defined 776
`britypes=` defined 769
B-splines 793
Bunch, D.S. 121 125 831
bundles of attributes 363 403
Burt table, correspondence analysis 826
bus 284
`by` statement, syntax 591
`c = 2 - (i eq choice)` 148
`c` variable 146-150 172 177 181 208 268-269 333 356 632 723-724
`c*c(2)` 149-150 175
`c*c(3)` 150
`Can` 240 290 680
`cand=` 652
`cand=` defined 649
candidate set 192 285 291 365 370 374 378 403 600 605-606 675 707-709
`canditer=` defined 692
candy example (choice) 144
candy example (conjoint) 489
canonical correlation 70 161 501
canonical initialization 586
cards, printing in a DATA step 502 516 557
Carmone, F.J. 99 831
Carroll, J.D. 24-25 818-820 823 827 831
Carson, R.T. 101 105 131 134 141 481 831
`cat=` 729
`cat=` defined 689
CATMOD procedure, see PROC CATMOD
Cattin, P. 99 836
CB data set 666
`cfill=` defined 720
`cframe=` defined 776
chair (generic attributes) example 363
Chakravarti, I.M. 105 108 117 831
change, market share 576 584
check the data entry 151
`check` defined 652 684
`check` option 196
chi-square metric, correspondence analysis 824
Chi-Square statistic 152
chocolate candy example (choice) 144
chocolate candy example (conjoint) 489
choice design, defined 96
choice design
60-63 89 157 164-166 171 735
efficiency 62-63 89 248 320-330 364-367 370-372 375 379 407 600 604 607-610 613
generation 366 370-372 375 379 600-601 605-613 656-657
generic 89
optimal 89
choice model
141 144-146 154 282-283
coding 68-69 85 173 209 214-216 219 223 255 263-265 269 275-280 333 337 340 357
fitting 174 211 214-216 220 226 257-259 264-266 273 278-280 334 342 357 360 470-471 474-475 478-479
choice probabilities 154
choice set, defined 96
choice sets 47
choice sets, minimum number 363
choice simulators
Bradley-Terry-Luce model 566
compared 567-568
defined 566
example 573-575
logit model 566
macro 570
maximum utility model 566 569
`Choice` variable 148
choice-based conjoint 141 485
`%ChoicEff` macro
5 72-73 82 91-93 130 136 142 166-167 203-204 248 316 320-324 334 364-367 370-381 399-400 403 407 417 420-422 425-428 432 436-440 452-455 458 461 464-465 597 600-

616 620 626-627 638 641 644 656-657 720 731-732
alternative swapping 374 381
documentation 598-631
set swapping 378 381
versus the %MktEx macro 381
choose 695
Choose variable 183
chosen alternative 148
Chrzan, K. 397 686 831
cirsegs= defined 780
class 172-173 205 209 213 218-219 223 256 264-266 269 305 319-322 332-334 354 363 366 489 492 505 521 562 566 579-581 594-596 609 751
class PROC TRANSREG syntax 589
class statement 305 650 653
classopts= defined 650
Client variable 714
close defined 767
cluster analysis 546
coded defined 629
coding down 237 650 694
coding the choice model 68-69 85 173 209 214-216 219 223 255 263-265 269 275-280 333 337 340 357
coding
binary 64 173 209 256 264
effects 64 218
orthogonal 64-65 68-69
price 68-69
coding= defined 650
coefficients 521 562 566 579-581
coefficients PROC TRANSREG syntax 588
Color variable 736-737
color= defined 776
colors= defined 769
column profiles, correspondence analysis 824
column defined 752
column statement 487 750
combinations
printing in a DATA step 502 516 557
unrealistic 552
Conforms 680
confounded 50 101
conjoint analysis
31 35 483-596 797
defined 484
model 491
typical options 594
conjoint measurement 31 483
constant alternative 156 178 261
constrained part-worth utilities 595
CONTENTS procedure, see PROC CONTENTS
converge= 608
converge= defined 628
converge= PROC TRANSREG syntax 586
convergence criterion 586
Cook, R.D. 104 109 124 192 288 366 675 832
Coolen, H. 786 832
coordinate-exchange algorithm 192 675
CORR data set 666
CORR procedure, see PROC CORR
correlations 535
CORRESP procedure, see PROC CORRESP
correspondence analysis 28 40 823-826
Count variable 351-353 632
cov= defined 628
Cox, D.R. 103 831
cprefix= 167 204 249
cross effects 263 268-276 283-286 305 321-322 326-327 332-334 337
cross effects, defined 96
cursegs= defined 780
curve fitting 797
curvecol= defined 777
customizing
multinomial logit output 143
PROC PHREG output 143 748-751
TRANSREG output 486
cyclic design 603
D-efficiency 53 102
D-efficiency, 0 to 100 scale 54 65-67 305
D-optimality 667
data collection
rank data 516
data entry
checking 151
choice 146-148 171 182 208 254 261 276 331 338 351 468
rank data 503 516
rating-scale data 489 561
simulation 577 581
data
collection, rank data 516
generating artificial 253 330
processing 183 219 378 473 476 489 503 516 561

validation, rank data 516
data= 149 166-167 171-172 177 203 248 320 347 356 367 521 562 566 579-581 594-596 601 629 633 645 661 714-715 720 723-724
data= defined 628 633 644 661 665 720 724 764
data= PROC TRANSREG syntax 585
datatype= defined 765
Davey, K.S. 99 109 832
Dawson, J.E. 668 832
Day, N.E. 468 831
de Boor, C. 786 793 831
De Cock, D. 668 832
de Launey, W. 3 668 832
de Leeuw, J. 786 796 831-836
debug= defined 781
define statement 487
degree, splines 590
degree= 216
degree= PROC TRANSREG syntax 590
demographic information 338
DenDF variable 565
depend variable 534 587
depvar variable 534-537 541-542 564 579-581
derivatives
splines 790
DeSarbo, W.S. 108 835
DESIGN data set 636 652 675 691
design
attributes 483-485
differences 498 630 637 647 653 689
efficient choice 62-63 89 248 320-330 364-367 370-372 375 379 407 600 604 607-610 613
efficient linear 53-54 60-61 101-102 498 509-512 520 549 552-553 667
evaluating efficiency 196 243 684
evaluation 160 194 197 232 242 292 297 300 303-305 348 499 509 552 635
example 498 509 552
factors 47 101
fractional-factorial 50 101
full-factorial 49-50 101-102 192 285 519 600
generation 158 178 188 196 232-235 243 287 292 297 300 303 346 363-365 374 378 383-386 389 600-601 605-606 609-611 638 641 656-660 667-669 675 712 715-721 736-737
holdouts 511
key 79 164-165 200 222 246 316 353 378 605 609-611 624 641 656-658 710 735-739
levels 47 101 110
methods compared 381
nonorthogonal 511 552-553
runs 47 100 156 185 230 234 294-296 345 364 496 550 740
saturated 68
shifted 89-91
size 156 185 230 234 294-296 345 364 496 550 682 740-743
testing 166 203 248 316
design 173 209 255 264
Design variable 187 369
design= 165 171-172 201 723 735 738-739
design= defined 724 734 739
Dest variable 206
detail defined 629
detfuzz= defined 699
Dey, A. 668 832
diag defined 767
differences (machine) in designs 630 637 647 653 689
different designs 498
diminishing returns on iterations 291
discontinuous functions
sample specification 596
splines 791
discrete choice 141-465
dog analogy 412
dolist= defined 720
dollar format 347
drop= 608 628
drop= defined 628
&droplist variable 564-566 577 581
dropping variables 174 209
Duan, L. 837
dummy variables 61
dummy PROC TRANSREG syntax 586
DuMouchell, W. 115 832
Duncan, G.J. 476 833
duplicate runs 678
dups defined 729
Dykstra, O. 104 832
Eckart, C. 23 818 832
edit statement 487 750
effects coding 64 218-219 609-610 626
effects 218 609
efficiency
choice design 62-63 89 248 320-330 364-367 370-372 375 379 407 600 604 607-610 613
evaluating for a given design 196 243 684

linear design 53-54 60-61 101-102 498 509-512 520 549 552-553 667
`Efficiency` variable 369
eigenvalues 52-53
Elrod, T. 99 109 397 686 831-832
errors in running macros 784
`%EvalEff` macro 305
evaluation, design 499 509 552
evenly spaced knots 590
`evenly` PROC TRANSREG syntax 590
`examine=` 161 196 501 650 683
`examine=` defined 650 683
examining the design 160 194 197 232 242 292 297 300 303-305 348 635
example
Bradley-Terry-Luce model 575
brand choice (aggregate data) 261
candy (choice) 144
candy (conjoint) 489
chair (generic attributes) 363
chocolate candy (choice) 144
chocolate candy (conjoint) 489
design 498 509 552
fabric softener 156
food product (availability) 283
frozen diet entrées (advanced) 509
frozen diet entrées (basic) 496
logit model 575
market share 569 575-576
maximum utility model 569
metric conjoint analysis 489 521 562
new products 576 583
nonmetric conjoint analysis 492 505
nonorthogonal design 549
partial profiles 397
prescription drugs (allocation) 345
simulation 569 575-576
spaghetti sauce 549
stimulus creation 502 516 557
vacation 184
vacation (alternative-specific) 229
`exchange=` 430 686-688 693
`exchange=` defined 695
`excolors=` defined 777
existing design, improving 383
`expand` defined 767
expansion
`class` 589
polynomial spline 590
experimental design
47-139
defined 47
evaluation 160 194 197 232 242 292 297 300 303-305 348 635
generation 158 178 188 196 232-235 243 287 292 297 300 303 346 363-365 374 378 383-386 389 600-601 605-606 609-611 638 641 656-660 667-669 675 712 715-721 736-737
saturated 68
shifted 89-91
size 156 185 230 234 294-296 345 364 496 550 682 740-743
testing 166 203 248 316
`extend=` defined 772
external attributes 338
external unfolding 820
`extraobs=` defined 771
extreme value type I distribution 283
`exttypes=` defined 769
`f` variable 385
`f1` variable 616
`f2` variable 616
fabric softener example 156
`facopts=` defined 650
FACTEX procedure, see PROC FACTEX
factors, design 47 101
`factors` statement 652
`factors=` 650-651 657
`factors=` defined 645 650 661 665
failed initialization 677
FASTCLUS procedure, see PROC FASTCLUS
Federer, W.T. 123 835
Fedorov, modified 192 600 675
Fedorov, V.V. 104-105 110 115 124 192 288 296-298 366-367 600 651 675-676 832
Fiedler, J.A. 833
`file` defined 684
`file` statement 169
`filepref=` defined 764
FINAL data set 721
Finkbeiner, C.T. 568 832
Fisher, R. 796 832
fitting the choice model 87 149 152 174 211 214-216 220 226 257-259 264-266 273 278-280 334 342 357 360 470-471 474-475 478-479
fixed choice sets 385
`fixed=` 385

`fixed=` defined 628 695
`flags=` 367 379 601 626-628
`flags=` defined 627
`font=` defined 773
food product (availability) example 283
football 52
`Form` variable 178 208 259
FORMAT procedure, see PROC FORMAT
`format` statement 223 724
`format=` defined 665
formats 155-158 163 180 246 255 261 276 312 319
formatting a `weight` variable 520 592
`&forms` variable 178
fractional-factorial design 50 101
`framecol=` defined 777
Franke, G.R. 28 823 833
FREQ data set 666
FREQ procedure, see PROC FREQ
`freq` statement 259 278-280 334 357 360
`Freq` variable 278
`_FREQ_` variable 259 333-334
`freq=` 356 633
`freq=` defined 633
`freqs=` 665
`freqs=` defined 665
frequencies, n-way 665
frequency variable 259-261 276-280
Friedman, J.H. 786 831
frozen diet entrées (advanced) example 509
frozen diet entrées (basic) example 496
FSUM data set 666
full-factorial design 49-50 101-102 192 285 519 600
G-efficiency 54 102
Gabriel, K.R. 22-23 818 832
Gail, M.H. 481 832
GANNO procedure, see PROC GANNO
Garratt, M. 3 58 99 122 286 785 834
`gdesc=` defined 764
Gellat, C.D. 676 833
general linear univariate model 786
`generate` statement 651
`generate=` defined 651
generic attribute, defined 96
generic attributes 211
generic design 89 363-370 374-381
generic design, defined 97
`generic` defined 661
geometric mean 53 102
Gifi, A. 28 484 786 823 832-833
GLM procedure, see PROC GLM
glossary 96
`gname=` defined 765
`gopplot=` defined 763
`gopprint=` defined 763
`gopts2=` defined 764
`gopts=` defined 764
`gout=` defined 765
GPLOT procedure, see PROC GPLOT
graphical scatter plots 753-783 803-828
Green, P.E. 31 99 105 118 121 483 797 827 831-833
`green=` defined 778
Greenacre, M.J. 28 823-824 827 833
Hadamard matrices 668 685 712-713
Hadamard, J. 668 833
Hastie, T. 28 786 823-824 833
Hayashi, C. 28 823
`header` statement 487
Hedayat, A.S. 123 668 833-835
Hensher, D.A. 831
`hminor=` defined 773
`hnobs=` defined 779
Hoffman, D.L. 28 823 833
Hoffman, S.D. 476 833
holdouts
  design 511
  validation 535
`holdouts=` 385 389
`holdouts=` defined 696
host differences 498 630 637 647 653 689
`hpos=` defined 781
`href=` defined 773
`HRLowerCL` 751
`HRUpperCL` 751
`hsize=` defined 781
Huber, J. 3 99 121-124 128 133-134 366 833
`(i eq choice)` 148
`i` variable 687 701
`ibd=` 435
`ibd=` defined 734
`id` statement 174 209 256 264
`ID` statement, syntax 591
`id=` defined 645
ideal point model 27 822
identity attribute, sample specification 595
`identity` 173 214 264-265 269 322 334 489 521 562 566 579-581 593-596 751

identity PROC TRANSREG syntax 589
IIA 269 275 283 476-480
IIA, defined 97
IML procedure, see PROC IML
imlopts= defined 700
importance
  average 542-544
  defined 492
  inflated 492
  outtest= 585
improving an existing design 383
inc= defined 773
Income variable 340
independence 175
independence from irrelevant alternatives 269 275 283
Index variable 369 608
indicator variables 61 64
individual R-square 542 564-565 579-582
inertia, correspondence analysis 824
infinite, see recursion
inflated, importance 492
information matrix 53 102
informats 577
Ini 680
init= 167 196 203 248 320 383-385 399 604 608 628-629 684 695 709
init= defined 628 690
initblock= defined 645
initialization failed 677
initialization switching 678
initvars= 604 608 628
initvars= defined 629
input data 146
input function 172 202
input statement 146
int defined 684
int= 365 600
int= defined 720
interact= 652
interact= defined 651 683 745
interactions 49 222 233 258 284-285 307-308 316
interpol= defined 773
interval scale of measurement 102 483 589 787 796
intiter= 167 203 248 321 604 629-630
intiter= defined 629
invalid page errors 784
ireplace 505 521 562 566 579-581 594-596
ireplace PROC TRANSREG syntax 588
iter= 407 648 651
iter= defined 629 636 645 651 692
iteration
  history suppressed 587
  history, %MktEx 678-680
  maximum number of 586
  metric conjoint analysis 491
  nonmetric conjoint analysis 492
iterative algorithm 586
j1 variable 688 697 701 707-709
j2 variable 688 697 701 708-709
j3 variable 688 701 708-709
Johnson, R.M 121 831
Jones, B. 3 115 832
justinit defined 684
justparse defined 746
keep= 316 651
keep= defined 651 739
Kendall Tau 535
Kent, J.T. 835
KEY data set
  %MktLab 623 713-720
  %MktRoll 79 605 609-611 624 641 656-658 710 735-739
Key data set
  %MktLab 312 347
  %MktRoll 164-165 200 222 246 316 353 378
key= 165 312 316 347-348 597 710 713-715 718 722 735-739
key= defined 720 739
Kharaghania, H. 668 833
Kirkpatrick, S. 676 833
knots
  defined 787
  evenly spaced 590
  in splines 787-802
  number of 591
  specifications 590
knots= 216 596
knots= PROC TRANSREG syntax 590
Krieger, A.B. 121 833
Krishnamurthi, L. 492 836
Kruskal, J.B. 30 786 796 834
Kuhfeld, W.F. 1-3 21 34-35 46-47 58 99 121-124 130 136 141 286 467 481-483 597 668 785-786 794 803 817 829-831 834-836
labcol= defined 769
label prefix option 586

label separator characters 587
label, variable 149-152 163 167 172 209 214-216 219 256 263-265 269-280 312 321 333 337 340 718-720 750-751
label statement 724
label= defined 774
labelcol= defined 777
labels= 718
labels= defined 720
labelvar= defined 767
labfont= defined 769
labsize= defined 770
large data sets 259 276
largedesign defined 685
largedesign option 410
Lazari, A.G. 99 105 113 121 134 285 305 481 834
Lebart, L. 28 823 834
levels
  design 47 101 110
  order 586
levels= defined 696
libname 163
libref 163
likelihood 143 146 149-150 175 228 259 268 279-282 468-471 479-481
lineage defined 685 729
linear design 60 63 157 164-166 185 200 230 246 312 316-317 363 381 403 735
linear design, defined 97
linear defined 661
linesleft= 169
LIST data set 666
list defined 636 682 711 745
list= defined 645 666
Lodge variable 201 209 223 247 256
-2 LOG L 152 268 279-282 480
LOGISTIC procedure, see PROC LOGISTIC
logit model
  compared to other simulators 567
  defined 566
  market share 575
Louviere, J.J. 3 99 109 121 141 483 831-834
lprefix= 167 174 204 209 249 256 264-266 321 562 566 579-581
lprefix= PROC TRANSREG syntax 586
ls= defined 774
lsinc= defined 774
lsizes= defined 774
Lu, Y. 668 836-837
Lubin, J.H. 481 832
Luce, R.D. 38 549 566-570 573-576
machine differences 498 630 637 647 653 689
macro
  autocall 597
  documentation 597-784
  errors 784
  notes, %MktEx 677
  variables 157 178
macros
  %ChoicEff 5 72-73 82 91-93 130 136 142 166-167 203-204 248 316 320-324 334 364-367 370-381 399-400 403 407 417 420-422 425-428 432 436-440 452-455 458 461 464-465 597-631 638 641 644 656-657 720 731-732
  %EvalEff 305
  macro; 677
  %MktAllo 142 356-357 597 632-634
  %MktBal 142 304 597 635-637 683
  %MktBlock 142 244 304 308-310 597 638-647 717-718
  %MktDes 5 597-598 648-654 691
  %MktDups 142 403 407 416 425 432 438-439 597 630 655-662
  %MktEval 5 72 76 109 142 160-161 194 197-198 232 242 292 297-298 301 347-348 499-501 509-511 552 555 597 635 638 646 663-666 718
  %MktEx 3-6 34 51 61-63 72 75 93-95 99 104 111 142-143 158-161 164-168 178 186-188 191-197 203 232-237 241-243 286-287 292 297-300 303-304 346-347 363-367 374 378 381-386 389-390 393 401-405 410-416 420-425 430-431 435 438-439 445-446 449-455 458-461 464-465 497-501 509-512 519 549-553 597-601 605-611 614 635 638 644 648-649 653 656-660 663 667-696 699-709 712-721 725 728-737 741-744 784 829
  %MktKey 5 72 78 142 200 353 378-379 406 416 425 438 597 624 710-711 735
  %MktLab 72 93 143 172 178 197 311-313 347-348 365 398 498-499 509-511 552 597 600-601 606 623-624 638 657 675 696 712-722
  %MktMerge 73 84 142 148-149 171 208 223 255 331 339 597 723-724
  %MktOrth 91 143 187 597-598 689 725-732
  %MktPPro 143 435 452-455 458 461 464-465 597 731-734
  %MktRoll 5 61 72 79 142 164-165 200-201 223

247 312 316-317 353 363 370 377-378 407 416 425 432 438 597 605-606 609-612 623-624 638 641 644 656-658 710 723-724 735-739
%MktRuns 72-76 103 142 156 185 230 234 286 294-296 345 364 496-497 549-550 597-598 636 651 682 729 740-747 784
%PhChoice 73 87 142-143 150 175 211 214 257 264 334 358 597 748-752 784
%PlotIt 597-598 753-783 803-816 828
%SIM 570 575 580-582
Mahajan, V. 108 835
main effects 49 284-285 307
&main variable 707-709
makefit= defined 781
Manski, C.F. 141 834
Mardia, K.V. 136 835
Market Research Analysis Application 35
market share
Bradley-Terry-Luce model 566 575
change 576 584
example 569
logit model 566 575
maximum utility model 566 569
simulation 569
mass, correspondence analysis 823
mautosource 598
max= 234 742 745
max= defined 745
maxdesigns= 695
maxdesigns= defined 692
maximum number of iterations 586
maximum utility model
compared to other simulators 567
defined 566
example 569
maxiter= 292 367 594-596 601 629 635-636 692-693
maxiter= defined 629 636 645 651 692 774
maxiter= PROC TRANSREG syntax 586
maxlev= 725
maxlev= defined 729 746
maxn= 725
maxn= defined 729
maxokpen= defined 774
maxstages= 410
maxstages= defined 693
maxstages=1 685
maxstarts= 635-636
maxstarts= defined 636
maxtime= 194 292 405 410 430 693-695
maxtime= defined 693
maxtries= 635
maxtries= defined 637
MCA 28 42 826-827
McFadden, D. 112 122 141 283-284 468 834-835
McKelvey, R.D. 467 835
MDPREF 24 43 818-820
MDS 30 44
MEANS procedure, see PROC MEANS
Meixia, M. 668 837
memory, running with less 259
method= 521 562 566 579-581
method= defined 651 764
method= PROC TRANSREG syntax 586
metric conjoint analysis
31
defined 484
example 489-490 521 562
iteration 491
sample specification 594
versus nonmetric 484 594
Meyer, R.K. 104 111 192 675 835
Micro variable 316 319-321 326
Miller, F.L. 104 835
Miller, R. 833
minimum number of choice sets 363
mintry= 299 393
mintry= defined 683
missing 312
missing statement 312 561
Mitchell, T.J. 104 835
%MktAllo macro
142 356-357 597 632-633
documentation 632-634
%MktBal macro
142 304 597 635-636 683
documentation 635-637
%MktBlock macro
142 244 304 308-310 597 638 641 644-646 717-718
documentation 638-647
%MktDes macro
5 597-598 648-653 691
documentation 648-654
MKTDESCAT data set 730
MKTDESLEV data set 730
%MktDups macro

142 403 407 416 425 432 438-439 597 630 655-661
documentation 655-662
%MktEval macro
5 72 76 109 142 160-161 194 197-198 232 242 292 297-298 301 347-348 499-501 509-511 552 555 597 635 638 646 663-665 718
documentation 663-666
%MktEx macro
3-6 34 51 61-63 72 75 93-95 99 104 111 142-143 158-161 164-168 178 186-188 191-197 203 232-237 241-243 286-287 292 297-300 303-304 346-347 363-367 374 378 381-386 389-390 393 401-405 410-416 420-425 430-431 435 438-439 445-446 449-455 458-461 464-465 497-501 509-512 519 549-553 597-601 605-611 614 635 638 644 648-649 653 656-660 663 667-670 675-678 681-689 695 699-707 712-721 725 728-737 741-744 784 829
algorithm 191-194 675-676
documentation 667-696 699-709
notes 677
versus the %ChoicEff macro 381
common options explained 158 188 196
mktex defined 729
%MktKey macro
5 72 78 142 200 353 378-379 406 416 425 438 597 624 710-711 735
documentation 710-711
%MktLab macro
72 93 143 172 178 197 311-313 347-348 365 398 498-499 509-511 552 597 600-601 606 623-624 638 657 675 696 712-721
documentation 712-722
%MktMerge macro
73 84 142 148-149 171 208 223 255 331 339 597 723
documentation 723-724
%MktOrth macro
91 143 187 597-598 689 725 728 732
documentation 725-730
%MktPPro macro
143 435 452-455 458 461 464-465 597 731-734
documentation 731-734
%MktRoll macro
5 61 72 79 142 164-165 200-201 223 247 312 316-317 353 363 370 377-378 407 416 425 432 438 597 605-606 609-612 623-624 638 641 644 656-658 710 723-724 735-738
documentation 735-739
%MktRuns macro
72-76 103 142 156 185 230 234 286 294-296 345 364 496-497 549-550 597-598 636 651 682 729 740-745
documentation 740-747
errors 784
with interactions 234
mktruns defined 729
model comparisons 228 268 281 479-480
model 489 492 505 521 562 566 581 594-596 627
model statement 149-150 173-175 209-211 256 264-265 269 305 377 601 626 652-653
model statement
options, TRANSREG 586
transformation options, TRANSREG 590
transformations 589
model= 367 601 626-628
model= defined 626
monochro= defined 777
monotone spline
589 595 793
sample specification 595
monotone 492 505 592-595
monotone PROC TRANSREG syntax 589
MORALS algorithm 586
morevars= defined 629
Morineau, A. 28 823 834
mother logit 268 275 284 334 478-479
mother logit model, defined 97
mspline 592 595-596
mspline PROC TRANSREG syntax 589
multidimensional preference analysis 24 43 818-820
multidimensional scaling 30 44
multinomial logit 144 149-150 175 263-265 283 467-481
multiple choices 345
multiple correspondence analysis 28 42 826-827
multiple defined 746
multiple2 defined 746
Mut 680
mutate= 692-693
mutate= defined 693
mutations 193 676
mutiter= 694
mutiter= defined 694
.N special missing value 714

n variable 187 369
n= 158 178 235 285 600 610-611 629 651
n= defined 629 636 651 682 746
Nachtsheim, C.J. 104 109-111 124 192 288 366 675 832 835
nalts= 167 171 203 208 248 320 331 356 379 606 626-628 633 644-647 661 723
nalts= defined 627 633 645 661 724
nblocks= defined 645
new products example 576 583
next= defined 645
nfill= 398
nfill= defined 721
Nishisato, S. 28 823 835
nknots= 216 595
nknots= defined 781
nknots= PROC TRANSREG syntax 591
nlev= 650 653
nlev= defined 651
noback defined 767
noborder defined 767
nocenter defined 767
noclip defined 768
nocode defined 629 652 768
nodelete defined 768
nodups defined 630 685
nodups option 161 501 511 684 692
nofinal defined 685
nohistory defined 685 768
nolegend defined 768
nominal scale of measurement 796
nominal variables 589
None alternative 285 316 334 337 342-344
nonlinear transformations 785
nonmetric conjoint analysis
  31
  defined 484
  example 492 505
  iteration 492
  sample specification 594
  versus metric 484 594
nonorthogonal design 511 552-553
noprint 566 579-581
noprint defined 637 661 768
noprint PROC TRANSREG syntax 586
norestoremissing 173 209 219 223 256 264
nosort defined 646 685
nosort option 385 691
not available 714
notests defined 630
notruncate 360
nowarn defined 739
nowarn option 201
nozeroconstant 173 209 255 264
nsets= 167 171 203 208 248 320 367 601 626 723
nsets= defined 627 724
nudge approach 422
number of choice sets, minimum 363
number of runs 47 100 156 185 230 234 294-296 345 364 496 550 740
number of stimuli 496
NumDF variable 565
NUMS data set 746
n-way frequencies 665
ODS 143 485 748
ods exclude statement 486 489 492 505 521 562 579-581 594-596
ods listing statement 535
ods output statement 535
offset= defined 774
onoff defined 752
OPTEX procedure, see PROC OPTEX
optimal choice design 89
options, TRANSREG
  algorithm 586-587
  output 587-588
  transformation 590-591
options defined 660
options= 410
options= defined 629 637 646 652 684 729 739 746 767
options=+- 686
options=3 686
options=512 686 729 746
options=accept 450 678 684-686 689 707-709
options=allcode 652
options=border 767
options=branded 660
options=check 196 652 684
options=close 767
options=coded 629
options=detail 629
options=diag 767
options=dups 729
options=expand 767
options=file 684
options=generic 661
options=int 684

options=justinit 684
options=justparse 746
options=largedesign 410 685
options=lineage 685 729
options=linear 661
options=mktex 729
options=mktruns 729
options=multiple 746
options=multiple2 746
options=noback 767
options=noborder 767
options=nocenter 767
options=noclip 768
options=nocode 629 652 768
options=nodelete 768
options=nodups 161 501 511 630 684-685 692
options=nofinal 685
options=nohistory 685 768
options=nolegend 768
options=noprint 637 661 768
options=nosort 385 646 685 691
options=notests 630
options=nowarn 201 739
options=orthcan 630
options=parent 729
options=progress 637
options=refine 685
options=render 685
options=resrep 410-411 685 701-703
options=source 746
options=square 768
options=textline 768
optiter= 405 410 691-694
optiter= defined 694
order of the spline 596
order= 256 505
order= defined 696
order= PROC TRANSREG syntax 586
order=data 173 209
order=matrix 235 241 297-300
order=random 405 410 421 430 686
order=random=n 430
ordering the attribute levels in the output 586
ordinal scale of measurement 796
ordinal variables 589-590
orthcan defined 630
orthogonal 50 54
orthogonal and balanced 51 54 66-68 157
orthogonal array 50
orthogonal coding 64-69
otherfac= defined 652
otherint= defined 652
out= 165 171-173 209 256 264 347 356 521 562 566 579-581 588 633 646 657 675 684 691 714-715 720 723-724 735-739
out=
  predicted utilities 588
  syntax 588
  transformation 588
out= defined 630 633 636 646 652 661 691 721 724 734 739 746 765
outall= 684 730
outall= defined 691 729
outcat= defined 730
outcb= defined 666
outcorr= defined 666
OUTDUPS data set 661
outest= 149
outfreq= defined 666
outfsum= defined 666
outlev= 727-730
outlev= defined 730
outlist= defined 661 666
output delivery system 143 485 748
output options, TRANSREG 587-588
output 492 521 587 594-596
output statement 174 209 256 264
outr= 675 684 691 713
outr= defined 646 691
outtest= 521 541 562 566 579-581
outtest=
  importance 585
  part-worth utilities 585
  R-square 585
  syntax 585
  utilities 585
outward= defined 781
p 505 521 562 566 579-581 588 594-596
p PROC TRANSREG syntax 588
page errors 784
page, new 181
paint= defined 778
Paley, R.E.A.C 668 835
Pang, S. 668 836
param=orthref 305
parameters 144 149 152-154 216 219 282-285 468-469 473 478 481
parent defined 729

partial profiles 397 402-461 464-465 686
partial= 398 401-405 684-688 691 709
partial= defined 686
part-worth utilities
  constrained 595
  defined 484
  output option 588
  outputting predicted 588
  outtest= 585
  printing 587
  summing to zero 591
part-worth utility 31 144 154 218 254 258
&pass variable 707-709
Pattern variable 737
__pbad variable 688
p_depend_ variable 534-537 587
Pearson r 535
perceptual mapping 22
permanent SAS data set 163-164 499 552 588 627 646 652 661 691 721
Perreault, W.D. 786 796 835
%PhChoice macro
  73 87 142-143 150 175 211 214 257 264 334 358 597 748 751-752 784
  documentation 748-752
  errors 784
PHREG procedure, see PROC PHREG
Place variable 201 209 222-223 247 256
place= defined 774
PLAN procedure, see PROC PLAN
PLOT procedure, see PROC PLOT
%PlotIt macro
  597-598 753-758 761-763 767 780 803-816 828
  documentation 753-783
plotopts= defined 775
plotting the transformation 492
plotvars= defined 768
point labels, scatter plots 753
point= 148 181
polynomial splines 590 785-802
post= defined 765
Pre 680
predicted utilities
  option 588
  out= 588
  variables 534
preference mapping 25 820
prefix, label option 586
prefix= defined 721
PREFMAP 25 820
preproc= defined 771
prescription drugs (allocation) example 345
price
  assigning actual 172 202 216 223 247 255
  coding 68-69
  pricing study 156 614
  quadratic 68-69 216 220 285
  sample specification 596
Price variable 164-166 172-173 177 201-203 209 213-214 223 247 256 264 268-269 284 312 316 321 324-328 357 613 632 735-737
PriceL variable 216
principal row coordinates, correspondence analysis 824
PRINCOMP procedure, see PROC PRINCOMP
PRINQUAL procedure, see PROC PRINQUAL
print= 161 501
print= defined 646 666 734
printing questionnaire 502 516 557
Prob variable 369
probability of choice 144-146 154-155 177-178 283 469-473
PROBIT procedure, see PROC PROBIT
PROC CATMOD 472
PROC CONTENTS 541
PROC CORR 535
PROC CORRESP 754 808-809
PROC DISCRIM 757
PROC FACTEX 192 648-653 675 691 707-708
PROC FASTCLUS 546
PROC FORMAT 81 155 158 171 202 246 261 276 312 499 509-511 520 552 577 592
PROC FREQ 656
PROC GANNO 759
PROC GLM 306-307
PROC GPLOT 145 291 494 568
PROC IML 367 435 443 447 687 731 755
PROC LOGISTIC 467
PROC MEANS 177 542
PROC OPTEX 192 291 305 648-653 675-677 685 691 695 707-708
PROC PHREG 87 143 149 152 174-175 209-211 214-220 226 257-261 264-266 273 278-280 334 342 357 360 470-475 478-481 748
PROC PHREG output, customizing 143 748-751
PROC PHREG, common options explained 149
PROC PLAN 192 648-649 652 675 691 707-708
PROC PLOT 756-758 768 771-775 779-780 783

PROC PRINCOMP 805-807
PROC PRINQUAL 754-755 810-812
PROC PROBIT 467
PROC SCORE 177
PROC SORT 155 183 494 503 507 537 546 583
PROC SUMMARY 259 332 353
PROC TEMPLATE 143 485-487 748-751 784
PROC TRANSPOSE 180-182 503 518 542 561 564 581 622
PROC TRANSREG
85 173-175 209-219 222-223 255 263-265 269 273-280 333 337 340 357 377-378 489 492 505 521 562 566 579-581 585-592 748 751 754 812-813
advanced sample 595
common options explained 173 255
customizing output 486
discontinuous price sample 596
monotone spline sample 595
nonmetric example 492
nonmetric sample 594
rank data sample 594
samples 594-596
simple example 489
specifications 585
syntax 585-592
typical example 521
processing
data 183 219 378 473 476 489 503 516 561
results 507 535-537 541-542 545-546 564 570 573-583
`procopts=` defined 652 775
`progress` defined 637
proportional hazards 143 149 279 470
proportions, analyzing 360
proximity data 30
`ps=` defined 781
pseudo-factors 650
`pspline` 216
`pspline` PROC TRANSREG syntax 590
`put` function 172 202
`put` statement 254
quadratic price effects 68-69 216 220 285
quantitative attributes 68-69
quantitative factor 177 213 216 258
questionnaire
169 178-182 206 251
printing 502 516 557
`radii=` defined 782
Raktoe, B.L. 123 835
Ramsay, J.O. 786 794 835
`Ran` 680
random mutations 193 240 676
random number seeds 158 188 195 291 304 308 367 498 601 630 637 646 653 689
randomization 49 163 178 251 316 499
RANDOMIZED data set 675 691
`range=` 725
`range=` defined 730
rank data
data collection 516
data entry 503 516
data validation 516
`reflect` 594
sample specification 594
versus rating-scale data 594
`rank` PROC TRANSREG syntax 590
`Rank` variable 503-507
Rao, C.R. 668 835
Rao, V.R. 31 483 833
`Rating` variable 490
rating-scale data
data entry 489 561
versus rank data 594
recursion, see infinite
red bus 284
`red=` defined 778
reference level 64 154 213 218-219 285
`Reference` variable 187
`refine` defined 685
`reflect` 505 521 594-596
`reflect`
rank data 594
syntax 591
reflection 505 591
`regdat=` defined 771
`regfun=` defined 782
`regopts=` defined 782
`regprint=` defined 782
Reibstein D.J. 492 836
`render` defined 685
`repeat=` defined 694
replacing independent variables, `ireplace` 588
`residuals` PROC TRANSREG syntax 588
`reslist=` 446 450
`reslist=` defined 687
`resmac=` 446 450
`resmac=` defined 687

resolution 50 101
resrep defined 685
resrep option 410-411 701-703
restrictions 286-287 292 297 300 303 397 402-461 464-465 551 687-688 707
restrictions not met 678
restrictions= 287 292 297 300 303 404 410 413 420 423 430 438 684-691
restrictions= defined 687
RESULTS data set 630
results processing 507 535-537 541-542 545-546 564 570 573-583
Reynolds, M.L. 30 835
rgbround= defined 779
rgbtypes= defined 770
ridge= 455
ridge= defined 646 700
ridging 455 608 646 700
rolled out data set 587
row profiles, correspondence analysis 824
RowHeader 750
R-square
  individual 542 564-565 579-582
  outtest= 585
Rubinstein, L.V. 481 832
Run variable 645-647 717
run= defined 652
runs, number of 47 100 156 185 230 234 294-296 345 364 496 550 740
s 769 776
sample specification
  discontinuous functions 596
  identity attribute 595
  metric conjoint analysis 594
  monotone attribute 595
  monotone spline 595
  nonmetric conjoint analysis 594
  price 596
  rank data 594
sasuser 163
saturated design 68 156-157 185 230
scales of measurement 796
scatter plots 753
Scene variable 201 209 223 247 256
Schaffer, C.M. 827 831
Schiffman, S.S. 30 835
SCORE procedure, see PROC SCORE
score= 177
second choice 146 149-150
seed= 158 367 498 601 630 637 646 653 689
seed= defined 630 637 646 653 689
separator characters 587
separators= 204 249 256 265-266 269 321 489 492 505 521 562 566 579-581
separators= PROC TRANSREG syntax 587
sequential algorithm 305
Set 647
set statement 148 181
Set variable 146 149-151 166-167 172 175 178 181-183 203 248 259-261 268-269 278 320 354 369 399 645 739
set= defined 647 739
setvars= 171 208 723
setvars= defined 724
Shape variable 736-737
shelf talker 283 316
Shelf variable 316 319-321 326
shelf-talker 311 316 338
Shepard, R.N. 786 796 834
short 489 505 521 562 566 579-581 594-596
short PROC TRANSREG syntax 587
Side variable 247 256
%SIM macro
  570 575 580-582
simulated annealing 193 240-241 676 692 699
simulation
  data entry 577 581
  example 569 575-576
  market share 569
  observations 519 537 541 592
simulators
  Bradley-Terry-Luce model 566
  compared 567-568
  example 573
  logit model 566
  macro 570
  maximum utility model 566 569
Size variable 736-737
size= defined 653
Sloane, N.J.A. 3 668 833-835
Smith, P.L. 787 835
So, Y.C. 3 141 467
SORT procedure, see PROC SORT
source defined 746
source statement 750
source stat.phreg statement 748
source stat.transreg statement 486
spaghetti sauce example 549

special missing value 312
spline
monotone 589 595
spline 592
spline PROC TRANSREG syntax 590
splines
592 785-802
degree 590
derivatives 790
discontinuous functions 791
monotone 793
order 596
with knots 788
square defined 768
Srinivasan, V. 31 483 797 833
standard column coordinates, correspondence analysis 824
statement
class 305 650 653
column 750
edit 750
factors 652
file 169
format 223 724
freq 259 278 360
generate 651
id 174 209 256 264
input 146
label 724
missing 312
model 149-150 173-175 209-211 256 264-265 269 305 377 601 626 652-653
output 174 209 256 264
put 254
set 148 181
source stat.phreg 748
source stat.transreg 486
source 750
strata 149 175 261 278
where 305 332 357 654
statements= 511
statements= defined 721 724
Statistic variable 542
Steckel, J.H. 108 835
Steinberg, D. 831
step= 652-653
step= defined 653
stimuli, number of 496 549-550
stimulus creation, DATA step 502 516 557
stmts= 223
stopearly= 677
stopearly= defined 699
stopping early 677
Stove variable 312
strata 149-151 175-176 259-261 276 280-282 470-471 481
strata statement 149 175 261 278
structural zeros 154 219 228
Stufken, J. 668 832-833
style= 753 767-769 776
style= defined 777
style=a 753
Style=RowHeader 750
subdesign 285 305
Subj variable 146 149-151 172 175 261-262 268-269 333
subject attributes 338
submat= defined 630
subsequent choice 146 149-150 208 261
Suen, C.Y. 3 668 835-836
suitable orthogonal coding 64
SUMMARY procedure, see PROC SUMMARY
summary table 150-151 279 337
summing to zero, part-worth utilities 591
survival analysis 143 149 470
Swait, J. 831
switching initialization 678
symbols= defined 770
symcol= defined 770
symfont= defined 770
symlen= defined 768
symsize= 700
symsize= defined 770
symtype= defined 770
symvar= defined 768
Tab 240 680
tabiter= 405 410
tabiter= defined 694
tabsize= defined 699
Taguchi, G. 668 836
Takane, Y. 786 796 834-836
target= defined 699
Tayfeh-Rezaiea, R. 668 833
t_depend_ variable 534-537 587
tempdat1= defined 772
tempdat2= defined 772
tempdat3= defined 772
tempdat4= defined 772

tempdat5= defined 772
tempdat6= defined 772
TEMPLATE procedure, see PROC TEMPLATE
template, utilities table 486
temporary 169 616
Tenenhaus, M. 823 836
Terry, M.E. 38 549 566-570 573-576
textline defined 768
Tibshirani, R. 786 833
tickaxes= defined 775
tickcol= defined 777
tickfor= defined 775
ticklen= defined 776
ties=breslow 143 149-150 175 279
time (computer), saving 259
Timmermans, H. 831
titlecol= defined 777
Tobias, R.D. 3 58 99 122 141 286 834
trace 53
trade-offs 483
TRank variable 507
transformation
  class 589
  identity 589
  monotone spline 589 595
  monotone 589
  mspline 589 595
  options, TRANSREG 590-591
  out= 588
  plot 492
  polynomial spline 590
  pspline 590
  rank 590
  regression 785 794-795
  spline 590
TRANSPOSE procedure, see PROC TRANSPOSE
TRANSREG procedure, see PROC TRANSREG
&_trgind variable 175-177 211-216 220 226 257-259 264-266 270 273 278-280 334 342 357 360 534 537 545-546
try variable 687 701
tsize= defined 776
-2 LOG L 152 268 279-282 480
type= 177
types= 630-631
types= defined 630 771
typevar= 630-631
typevar= defined 631 771
typical options, conjoint analysis 594
Unb 680
unbalanced= defined 694
unit= defined 782
UNIVARIATE algorithm 586
unrealistic combinations 552
utilities
  31
  constrained 595
  defined 484 491
  outputting predicted 588
  outtest= 585
  predicted 534
  printing 587
  table, template 486
utilities 489 492 505 521 562 566 579-581 594-596
utilities PROC TRANSREG syntax 587
vacation (alternative-specific) example 229
vacation example 184
validation, holdouts 535
Value variable 542 565
values= 718-722
values= defined 722
van der Burg, E. 786 831
van Rijckevorsel, J. 786 832 835
variable label 149-152 163 167 172 209 214-216 219 256 263-265 269-280 312 321 333 337 340 718-720 750-751
variable
  Age 340
  Alt 178 642 645
  Alt 739
  bad 286-287 411 687-688
  __bbad 688
  Block 201 347 357 638 645 717
  Brand 164-166 172-173 264 268-269 316 321 324-328 340 351 354 357 613 616 619 632 735-739
  c 146-150 172 177 181 208 268-269 333 356 632 723-724
  Choice 148
  Choose 183
  Client 714
  Color 736-737
  Count 351-353 632
  DenDF 565
  depend 534 587
  depvar 534-537 541-542 564 579-581

Design 187 369
Dest 206
&droplist 564-566 577 581
Efficiency 369
f 385
f1 616
f2 616
Form 178 208 259
&forms 178
Freq 278
FREQ 259 333-334
i 687 701
Income 340
Index 369 608
j1 688 697 701 707-709
j2 688 697 701 708-709
j3 688 701 708-709
Lodge 201 209 223 247 256
&main 707-709
Micro 316 319-321 326
n 187 369
NumDF 565
&pass 707-709
Pattern 737
__pbad 688
p_depend_ 534-537 587
Place 201 209 222-223 247 256
Price 164-166 172-173 177 201-203 209 213-214 223 247 256 264 268-269 284 312 316 321 324-328 357 613 632 735-737
PriceL 216
Prob 369
Rank 503-507
Rating 490
Reference 187
Run 645-647 717
Scene 201 209 223 247 256
Set 146 149-151 166-167 172 175 178 181-183 203 248 259-261 268-269 278 320 354 369 399 645 739
Shape 736-737
Shelf 316 319-321 326
Side 247 256
Size 736-737
Statistic 542
Stove 312
Subj 146 149-151 172 175 261-262 268-269 333
t_depend_ 534-537 587
TRank 507
&_trgind 175-177 211-216 220 226 257-259 264-266 270 273 278-280 334 342 357 360 534 537 545-546
try 687 701
Value 542 565
w 319-321 332 511 534
weight 521 577
x 688 701 708
x1 688 701
x2 701
x[j] 688
xmat 688 701 708
variables
interval 589
nominal 589
ordinal 589-590
predicted utilities 534
replacing in output data set 588
residuals 588
variance matrix 53 102
vars= 197 356 633
vars= defined 633 645 661 665 722
Vecchi, M.P. 676 833
vechead= defined 782
vector model 26 820
view= 305
Violations 680
vminor= defined 776
vnobs= defined 779
vpos= defined 782
vref= defined 776
vsize= defined 782
vtoh= defined 783
w variable 319-321 332 511 534
Wang, J.C. 3 668 836
Wang, Y. 668 836
Warwick, K.M. 28 823 834
Watson, W. 3 35
weight format 520 592
weight 521 579-581 594-596
weight statement
holdouts 592
sample specification 595
syntax 592
weight variable 521 577
weight= 321
weight= defined 631
weighted loss function 592
Weiguo, L. 668 837

whack approach 420-422 430
`where` statement 305 332 357 654
`where=` 177
`where=` defined 654
Wiley, J.B. 121 831
Williamson, J. 668 836
Wind, Y. 31 99 105 118 483 833
Winsberg, S. 786 836
Wish, M. 30 834
With Covariates 152 228 268
Wittink, D.R. 99 492 833 836
Woodworth, G. 99 121 141 834
`worksize=` 700
Wu, C.F.J. 668 836
`x` variable 688 701 708
`x1` variable 688 701
`x2` variable 701
`x=` defined 734
`x[j]` variable 688
`xmat` variable 688 701 708
`xmax=` defined 783
Xu, H. 668 836
`ymax=` defined 783
Young, F.W. 3 30 484 786 796 823 834-836
Young, G. 23 818 832
Zavoina, W. 467 835
`zero=` 173 203 209 213-214 218-219 248-249 256 264-265 275 321-322 334 399 489 492 505 521 562 566 579-581 594-596 613
`zero=`*list* 214 256
`zero=` PROC TRANSREG syntax 591
`zero=' '` 256 607
Zhang, Y.S. 3 668 836-837
Zheng, Z. 668 837
Zwerina, K. 3 121-124 128 133-134 366 833