Multinomial Logit, Discrete Choice Modeling

An Introduction to Designing Choice Experiments, and Collecting, Processing, and Analyzing Choice Data with SAS®

Warren F. Kuhfeld
SAS

August 31, 2002

Contents

Multinomial Logit, Discrete Choice Modeling

This report shows you how to use the multinomial logit model (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). We will discuss designing a choice experiment, preparing the questionnaire, inputting and processing the data, performing the analysis, and interpreting the results. 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.

This report is TS-677E, the August 31, 2002 edition for SAS Version 9.0. It is a revision of the April 1, 2001 report for Version 8.2 and other previous editions. This edition uses SAS macros and features that are new in Version 9, including the new experimental design macro, **%MktEx** . This report heavily relies on a number of macros and procedures.

- We use the **%MktRuns** autocall macro to suggest design sizes. See page 360 for documentation. The **%MktRuns** macro has been revised since the 2001 book.
- We use the **%MktEx** autocall macro to generate most of our experimental designs. It is easier to use and usually produces better results than the methods suggested in earlier reports. See page 327 for documentation. The **%MktEx** macro is new with the 2002 book.
- We use the **%MktEval** autocall macro to evaluate our designs. See page 325 for documentation. The **%MktEval** macro has been revised since the 2001 book.
- We use the **%ChoiceEff** autocall macro to generate certain specialized choice designs. See page 288 for documentation.
- We use the autocall macros **%MktRoll**, **%MktMerge**, and **%MktAllo** to prepare the data and design for analysis. See pages 356, 353, and 303 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 364 for documentation.
- The **%MktBal** macro can be used to make perfectly balanced designs. See page 305 for documentation. The **%MktBal** macro is new with the 2002 book.
- The **%MktBlock** macro can be used to block a linear or choice design. See page 307 for documentation. The **%MktBlock** macro is new with the 2002 book.
- The **%MktDes** experimental design macro, which was heavily used in previous reports, is called by the **%MktEx** macro, and it can still be called directly. See page 307 for documentation.
- The **%MktDups** macro can be used to search for duplicate runs or choice sets. See page 319 for documentation. The **%MktDups** macro is new with the 2002 book.
- The **%MktLab** macro can be used to assign different variable names, labels and levels to experimental designs and to add an intercept. See page 345 for documentation. The **%MktLab** macro is new with the 2002 book.
- The **%MktOrth** macro can be used to list orthogonal experimental designs that the **%MktEx** macro can produce. See page 354 for documentation. The **%MktOrth** macro is new with the 2002 book.

All of these macros are distributed with Version 9.0 of SAS as autocall macros (see page 287 for more information on autocall macros). Note however, that Version 9.0 was finished before the macros were finalized and this book finished. Hence there are a few differences between the macros used in this book and those shipped with Version 9.0 of SAS. If you are running version 9.0 or any earlier version of SAS, get the latest macros from the web or by writing Warren.Kuhfeld@sas.com. This report and the macros are available from the Technical Support web site at http://www.sas.com/service/techsup/tnote/tnote_stat.html . This information is provided by SAS as a service to its users. It is provided "as is." There are no warranties, expressed or implied, as to merchantability or fitness for a particular purpose regarding the accuracy of the materials or code contained herein.

Several examples are discussed.*

- The candy example 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 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 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.
- The second vacation example 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 is a small example that discusses the processing of aggregate data, the mother logit model, and the likelihood function.
- The food product example is a medium sized example that discusses asymmetry, coding, checking the design to ensure that all effects are estimable, availability cross effects, interactions, overnight design searches, modeling subject attributes, and designs when balance is of primary importance.
- The drug allocation example is a small example that discusses data processing for studies where respondents potentially make multiple choices.
- The chair example is a purely generic-attributes study, and it uses the **`%ChoicEff`** macro to create experimental designs.
- The last example sections contains miscellaneous examples including improving an existing design, augmenting a design with some choice sets are fixed in advance, and partial profiles.

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 sample data sets are artificially generated.

Preliminaries

This section defines some design terms that we will use later and shows how to customize the multinomial logit output listing. Impatient readers may skip ahead to the candy example on page 83 and refer back to this section as needed.

Experimental Design Terminology

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 called *runs* and correspond to product profiles in a full-profile conjoint study or choice sets in a discrete choice study. Experiments are performed to study the effects of the factor levels on the dependent variable. In a discrete-choice study, the factors are the attributes of the hypothetical products or services, and the response is choice. For example, the following table contains an experimental design in 8 runs with three factors, Brand 1 price, Brand 2 price, and Brand 3 price. Each factor has two levels, $1.99 and $2.99.

Linear Design
For a Choice Model

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

This is an example of a *full-factorial design*. It consists of all possible combinations of the levels of the factors. 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 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.

A special type of fractional-factorial design is the *orthogonal array*. An orthogonal array or orthogonal design is one 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. 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. Orthogonal arrays come in specific numbers of runs (such as 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 is correctly used to refer to designs that are both orthogonal and balanced, and hence optimal. 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. A design is *balanced* when each level occurs equally often within each factor, which that means the intercept is orthogonal to each effect. Imbalance is a generalized form of nonorthogonality, hence it increases the variances of the parameter estimates and decreases efficiency.

Efficiency of an Experimental Design

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 $\hat{\beta}$ 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 trace $((\mathbf{X}'\mathbf{X})^{-1})/p$. (The trace is the sum of the diagonal elements of a matrix, which is the sum of the eigenvalues.) *D-efficiency* is a function of the geometric mean of the eigenvalues, which is given by $|(\mathbf{X}'\mathbf{X})^{-1}|^{1/p}$. (The determinant, $|(\mathbf{X}'\mathbf{X})^{-1}|$, is the product of the eigenvalues of $(\mathbf{X}'\mathbf{X})^{-1}$.) 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, $\frac{1}{N_D}\mathbf{I}$, 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.

These measures of efficiency can be scaled to range from 0 to 100 (see page 80 for the orthogonal coding of $\mathbf{X}$ that must be used with these formulas):

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

These efficiencies measure the goodness of the design relative to hypothetical orthogonal designs that may 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. Throughout this report, we will use the **`%MktEx`** macro to find good, efficient experimental designs.

Efficiency of a Choice Design

All of the theory in the preceding section concerned linear models. In linear models, the parameter estimates $\hat{\beta}$ have variances proportional to $(\mathbf{X}'\mathbf{X})^{-1}$. In contrast, the variances of the parameter estimates in the 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

We will often create experimental designs for choice models using efficiency criteria for linear models. Consider an extremely simple example of three brands and two prices. We could use linear model theory to create a design for a full-profile conjoint study. The full-profile conjoint design has two factors, one for brand and one for price. For the same brands and prices, we could instead use linear model theory to create a *linear design* from which we will construct a *choice design* to use in a discrete choice study. The linear 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.

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

Before we fit the choice model, we will construct a choice design from the linear design and code the choice design. See the three tables below.

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

Choice Design

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

Choice Design Coding

Brand 1	Brand 2	Brand 3	Brand 1 Price	Brand 2 Price	Brand 3 Price
1	0	0	1.99	0	0
0	1	0	0	1.99	0
0	0	1	0	0	1.99
1	0	0	1.99	0	0
0	1	0	0	2.99	0
0	0	1	0	0	2.99
1	0	0	2.99	0	0
0	1	0	0	1.99	0
0	0	1	0	0	2.99
1	0	0	2.99	0	0
0	1	0	0	2.99	0
0	0	1	0	0	1.99

The linear design has one row per choice set. The choice design has three rows for each choice set. 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.

We will go through how to construct linear and choice designs many times in the examples. 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.

We make a good design for a linear model by picking $\mathbf{x}$'s that minimize functions of $(\mathbf{X}'\mathbf{X})^{-1}$. In the choice model, ideally we would like to minimize functions of

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

We cannot do this unless we know β, and if we knew β, we would not need to do the experiment. (However, in the chair example on pages 252−267, we will see how to make an efficient choice design when we are willing to make assumptions about β.)

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 that are 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 safe in the sense that you have enough choice sets and collect enough 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.

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 287 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 364 for more information on the **%PhChoice** macro.

Orthogonal Coding, Efficiency, Balance, and Orthogonality

We mentioned on page 77 that we use a special orthogonal coding of $\mathbf{X}$ when computing design efficiency. This section shows that coding. All but the most dedicated readers may skip ahead to the candy example on page 83.

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

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

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

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 trace $((\mathbf{X}'\mathbf{X})^{-1})/p = |(\mathbf{X}'\mathbf{X})^{-1}|^{1/p} = 1/N_D$. In this case, the two denominator terms cancel and efficiency is 100%. As the average variance increases, efficiency decreases.

Here are the orthogonal codes for two-level through five-level factors.

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 coding of the two-level factor is 2; for all of the columns of the three-level factor, the sums of squares are 3; for the four-level factor, the sums of squares are all 4; and for 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.

This example shows the coding of a 2×6 full-factorial design in 12 runs using a coding function that requires that the factors 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 trace $((\mathbf{X}'\mathbf{X})^{-1})/p$ 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 report, we will never consider an experimental design with fewer runs than a saturated design. A *saturated design* has as many runs as there are parameters. The number of parameters in a main-effects model is 1 (for the intercept) plus the sum of the numbers of levels of all of the factors, minus the number of factors. 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.

Candy Example

We begin with a very simple example. In this example, we will discuss the multinomial logit model, data input and processing, analysis, results, interpretation, and probability of choice. 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 **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 $\boldsymbol{\beta}' = (4 \quad -2 \quad 1)$. 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

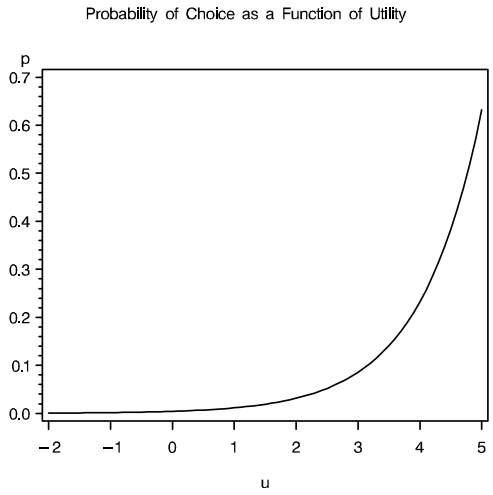
The denominator of the probability formula, $\sum_{j=1}^{m} \exp(\mathbf{x}_j\boldsymbol{\beta})$, is $\exp(0) + \exp(1) + \exp(-2) + \exp(-1) + \exp(4) + \exp(5) + \exp(2) + \exp(3) = 234.707$. The probability that each alternative is chosen, $\exp(\mathbf{x}_i\boldsymbol{\beta}) / \sum_{j=1}^{m} \exp(\mathbf{x}_j\boldsymbol{\beta})$,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

```
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. 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.

Fitting the Multinomial Logit Model

The data are now in the right form for analysis. 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. 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 unchosen. 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 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 79 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

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 next table, 'Convergence Status', 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 covariance 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

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 the reader is familiar with the experimental design issues discussed in Kuhfeld, Tobias, and Garratt (1994), on page 25. Some of these concepts are reviewed starting on page 76.

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 report are documented starting on page 287. 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 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

Some Reasonable Design Sizes	Violations	Cannot Be 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.**

*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.

Choice of Fabric Softener

n	Design	Reference
9	3 ** 4	Fractional-factorial
18	2 ** 1 3 ** 7	Taguchi, 1987
18	3 ** 6 6 ** 1	Taguchi, 1987

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 threes) 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 remember 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 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 report are documented starting on page 287.) 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 report, random number seeds are explicitly specified with the **seed=** option so that the results will be reproducible.* Here is the macro

*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.

call with the random number seed specified:

```
%mktex(3 ** 4, n=&n, seed=7654321)

proc print; run;
```

Here are the results.

```
                    Choice of Fabric Softener

                     Algorithm Search History

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

                    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

                                                          Average
                                                       Prediction
Design                                                   Standard
Number     D-Efficiency     A-Efficiency     G-Efficiency      Error
-------------------------------------------------------------------
     1       100.0000         100.0000         100.0000          0.7071 p

                    Choice of Fabric Softener

                   Obs    x1    x2    x3    x4

                     1     1     1     1     1
                     2     1     1     2     3
                     3     1     2     1     3
                     4     1     2     3     2
                     5     1     3     2     2
                     6     1     3     3     1
                     7     2     1     1     2
                     8     2     1     3     3
                     9     2     2     2     2
```

```
10     2     2     3     1
11     2     3     1     3
12     2     3     2     1
13     3     1     2     1
14     3     1     3     2
15     3     2     1     1
16     3     2     2     3
17     3     3     1     2
18     3     3     3     3
```

The macro found a perfect, orthogonal and balanced, 100% 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 123 for more information on how the **%MktEx** macro works.

Examining the Design

It is good to run basic checks on all designs. 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). 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
```

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. 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 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 126, 162, and 336). 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. In this case, there are no correlations.

The macro also prints one-way, two-way and n-way frequencies. 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 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** to ensure that there are no duplicates.

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
There are 0 Canonical Correlations Greater Than 0.316

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

Randomizing the Design, 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 has a first row consisting 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 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.DES so that it will still be available after the data are collected.

```
proc print data=randomized; run;

data sasuser.des;
   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.des label; /* print final design */
   title2 'Efficient Design';
   run;

title2;
```

Choice of Fabric Softener
Efficient Design

Obs	Sploosh	Plumbbob	Platter	Moosey
1	$2.49	$1.99	$1.99	$2.49
2	$1.49	$2.49	$1.99	$1.99
3	$1.49	$1.99	$1.99	$1.49
4	$2.49	$2.49	$2.49	$1.49
5	$1.99	$1.49	$1.99	$2.49
6	$1.49	$1.49	$1.49	$2.49
7	$1.49	$1.99	$1.49	$1.99
8	$1.49	$1.49	$2.49	$1.49
9	$2.49	$1.49	$1.49	$1.49
10	$1.99	$2.49	$1.99	$1.49
11	$2.49	$1.49	$1.99	$1.99
12	$1.99	$1.99	$2.49	$2.49
13	$1.99	$2.49	$1.49	$1.99
14	$2.49	$1.99	$2.49	$1.99
15	$2.49	$2.49	$1.49	$2.49
16	$1.99	$1.49	$2.49	$1.99
17	$1.49	$2.49	$2.49	$2.49
18	$1.99	$1.99	$1.49	$1.49

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.des;

   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 $2.49.

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 $1.49.

2) Plumbbob brand at $2.49.

3) Platter brand at $1.99.

4) Moosey brand at $1.99.

5) Another brand at $1.99.

In practice, data collection may be much more elaborate than this. It may involve art work, photographs, and the choice sets may be presented and data may be collected 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.

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 $1.99) 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 334523313433343413  2 314423343433333215  3 333423323323333233
 4 314432343444343254  5 315423313421543213  6 334433344423323233
 7 334433343432343413  8 334423323434343214  9 333433313351343213
10 345423325322343233 11 333433343433323213 12 314423343423333253
13 534423313322343213 14 214421144423323214 15 334423345433343235
16 333433345333335313 17 534423343452343413 18 344435544432343513
19 334433345432343433 20 311423343422325513 21 334453543423543213
22 314423353323333513 23 333423343333333433 24 545422323322323213
25 354433343433333313 26 314425525422343214 27 334353523423353213
28 334433313333333233 29 333423543335353234 30 334453343533343433
31 354423344322333413 32 354422343323333213 33 314423343352343215
34 334423343443333213 35 314553344453343215 36 333433544433343233
37 314423343424543214 38 353433324423353533 39 333453323333323513
40 314433343422333214 41 334423344442343444 42 334433323422323213
43 333423354433343213 44 314423323422333213 45 314452544422343214
46 414423345423543214 47 544423544442343414 48 335453343423323453
49 314523344424333214 50 334423343332343413
;
```

Processing the Data

Next, we prepare the experimental design for analysis. Our design, stored in the data set SASUSER.DES, is stored with one row per choice set. We call this the *linear design* (see page 78). The linear design, which came directly from the **%MktEx** macro, is conveniently arrayed for generating the questionnaire, however it is not in the right form for analysis. For analysis, we need a *choice design* with one row for each alternative of each choice set. We will use the macro **%MktRoll** to "roll out" the linear design into the choice design, which is in 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;

title2;
```

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 placeholder, 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.des, key=key, alt=brand, out=rolled)
```

The **%MktRoll** step processes the **design=sasuser.des** 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 ROLLED, which contains the choice design. The input **design=sasuser.des** data set has 18 observations, one per choice set, and the output **out=rolled** 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.des(obs=3); run;

title2;
```

Choice of Fabric Softener
Linear Design (First 3 Sets)

Obs	x1	x2	x3	x4
1	$2.49	$1.99	$1.99	$2.49
2	$1.49	$2.49	$1.99	$1.99
3	$1.49	$1.99	$1.99	$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=rolled(obs=15); format price price.; run;

title2;
```

Choice of Fabric Softener
Choice Design (First 3 Sets)

Obs	Set	Brand	Price
1	1	Sploosh	$2.49
2	1	Plumbbob	$1.99
3	1	Platter	$1.99
4	1	Moosey	$2.49
5	1	Another	$1.99
6	2	Sploosh	$1.49
7	2	Plumbbob	$2.49
8	2	Platter	$1.99
9	2	Moosey	$1.99
10	2	Another	$1.99
11	3	Sploosh	$1.49
12	3	Plumbbob	$1.99
13	3	Platter	$1.99
14	3	Moosey	$1.49
15	3	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;
```

The next step merges the choice data with the choice design using the **%MktMerge** macro.

```
%mktmerge(design=rolled, data=results, out=res2,
          nsets=&n, nalts=&m, setvars=choose1-choose&n)
```

This step reads the **design=rolled** 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); run;

title2;
```

Choice of Fabric Softener
Choice Design and Data (First 3 Sets)

Obs	Subj	Set	Brand	Price	c
1	1	1	Sploosh	3	2
2	1	1	Plumbbob	2	2
3	1	1	Platter	2	1
4	1	1	Moosey	3	2
5	1	1	Another	.	2
6	1	2	Sploosh	1	2
7	1	2	Plumbbob	3	2
8	1	2	Platter	2	1
9	1	2	Moosey	2	2
10	1	2	Another	.	2
11	1	3	Sploosh	1	2
12	1	3	Plumbbob	2	2
13	1	3	Platter	2	2
14	1	3	Moosey	1	1
15	1	3	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=rolled** data set, and the variable **c**, which indicates which alternative was chosen. The variable **c** indicates the chosen alternatives: 1 for first choice and 2 for second or subsequent choice. This subject chose the third alternative, *Platter*, for each of the first two choice sets, and Moosey for 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 of 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 numbers 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. A *binary* or zero-one design matrix must be coded 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 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. 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 they were first encountered in the data set. The specification **identity(price)** specifies that **Price** is a quantitative factor that should be analyzed as is (not expanded into dummy 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. 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'**.

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;
   run;

title2;
```

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	2.49	Sploosh
1	1	2	0	1	0	0	0	1.99	Plumbbob
1	1	1	0	0	1	0	0	1.99	Platter
1	1	2	0	0	0	1	0	2.49	Moosey
1	1	2	0	0	0	0	1	1.99	Another

```
1    2   2     1        0        0        0        0      1.49  Sploosh
1    2   2     0        1        0        0        0      2.49  Plumbbob
1    2   1     0        0        1        0        0      1.99  Platter
1    2   2     0        0        0        1        0      1.99  Moosey
1    2   2     0        0        0        0        1      1.99  Another
1    3   2     1        0        0        0        0      1.49  Sploosh
1    3   2     0        1        0        0        0      1.99  Plumbbob
1    3   2     0        0        1        0        0      1.99  Platter
1    3   1     0        0        0        1        0      1.49  Moosey
1    3   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;

title2;
```

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 79 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

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	1567.162
AIC	2896.988	1577.162
SBC	2896.988	1601.174

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	1329.8263	5	<.0001
Score	1197.0568	5	<.0001
Wald	608.1816	5	<.0001

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Sploosh	1	-1.11167	0.20114	30.5462	<.0001
Plumbbob	1	-0.16186	0.16951	0.9117	0.3397
Platter	1	1.95389	0.14609	178.8686	<.0001
Moosey	1	0.78650	0.15517	25.6909	<.0001
Another	0	0	.	.	.
Price	1	-4.25545	0.19663	468.3659	<.0001

The procedure output begins with information about the data set, variables, and options. 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 226) 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 $\mathbf{x}_i\beta$ which is $(0 \quad 0 \quad 1 \quad 0 \quad 0 \quad \$1.99) \quad (-1.11 \quad -0.16 \quad 1.95 \quad 0.79 \quad 0 \quad -4.26)' = 1.95 + 1.99 \times -4.26 = -6.53$. 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 $1.95 + 1.89 \times -4.26 = -6.10$, which is a $\$0.10 \times 4.26 = 0.43$ 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 $\mathbf{x}_j\hat{\beta}$ 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 ($\mathbf{x}_j\hat{\beta}$). 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 $\mathbf{x}_j\hat{\beta}$.

```
data p2;
   set p;
   p = exp(c);
   run;
```

Next, $\exp(\mathbf{x}_j\hat{\beta})$ is summed for each choice set.

```
proc means data=p2 noprint;
   output out=s sum(p) = sp;
   by set;
   run;
```

Finally, each $\mathbf{x}_j\hat{\beta}$ is divided by $\sum_{j=1}^{m} \mathbf{x}_j\hat{\beta}$.

```
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;

title2;
```

Choice of Fabric Softener
Choice Probabilities for the First 3 Choice Sets

Obs	Price	Brand	Set	p
1	2.49	Sploosh	1	0.00426
2	1.99	Plumbbob	1	0.09238
3	1.99	Platter	1	0.76635
4	2.49	Moosey	1	0.02840
5	1.99	Another	1	0.10861
6	1.49	Sploosh	2	0.21061
7	2.49	Plumbbob	2	0.00772
8	1.99	Platter	2	0.53800
9	1.99	Moosey	2	0.16741
10	1.99	Another	2	0.07625
11	1.49	Sploosh	3	0.09176
12	1.99	Plumbbob	3	0.02825
13	1.99	Platter	3	0.23439
14	1.49	Moosey	3	0.61237
15	1.99	Another	3	0.03322

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 116.

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.

```
%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(7);
   r2 = uniform(7);
   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	4
2	1	4	2
3	1	4	3
4	1	4	1
5	1	8	4
6	1	8	2
7	1	8	1
8	1	8	3
9	1	7	4
10	1	7	3
11	1	7	2
12	1	7	1
13	1	14	4
14	1	14	1
15	1	14	3
16	1	14	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	4	2	3	1
2	1	8	4	2	1	3
3	1	7	4	3	2	1
4	1	14	4	1	3	2
5	1	9	2	1	4	3
6	1	11	4	1	2	3
7	1	17	3	2	4	1
8	1	1	4	1	2	3
9	1	12	1	3	2	4
10	1	13	2	1	4	3
11	1	18	3	1	4	2
12	1	3	3	1	4	2
13	1	2	3	4	2	1
14	1	5	3	4	1	2
15	1	15	1	3	4	2
16	1	6	3	2	1	4
17	1	16	2	1	4	3
18	1	10	4	1	3	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.des 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.DES 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=16`**, **`Col1=2`**, **`Col2=3`**, **`Col3=1`**, and **`Col4=4`**. The first brand, is **`c[brnds] = c[1] = col1 = 2`**, so **`brands[c[brnds]] = brands[c[1]] = brands[2] = 'Plumbbob'`**, and the price, from observation **`Set=16`** of SASUSER.DES, is **`x[c[brnds]] = x[2] = $1.99`**. The second brand, is **`c[brnds] = c[2] = col2 = 3`**, so **`brands[c[brnds]] = brands[c[2]] = brands[3] = 'Platter'`**, and the price, from observation **`Set=16`** of SASUSER.DES, is **`x[c[brnds]] = x[3] = $1.49`**.

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) Moosey brand at $1.49.

    2) Plumbbob brand at $2.49.

    3) Platter brand at $2.49.

    4) Sploosh brand at $2.49.

    5) Another brand at $1.99.
```

```
2) Circle your choice of one of the following fabric softeners:

   1) Moosey brand at $1.49.

   2) Plumbbob brand at $1.49.

   3) Sploosh brand at $1.49.

   4) Platter brand at $2.49.

   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 532144442413142131  2 512111233124212431  3 122111234233443322
  4 141153221441433422  5 114112352113441523  6 243134342341213233
  7 324132423331441234  8 233333444114324111  9 132244342211532121
 10 123313551442431124 11 432413112334322232 12 344144441133222325
 13 241112415231331332 14 234311233124243122 15 431114535444121411
 16 145431412414225122 17 352214223453322244 18 153214343455431432
 19 311424145334343233 20 322242435132322453 21 422313351152215321
 22 144434352124221354 23 431322144242341234 24 234313521411125323
 25 222341432223153413 26 452315344524342414 27 512514432245223241
 28 223133334224321424 29 134244525425322423 30 442434215232335231
 31 425133234223121332 32 125322244221121142 33 341335431241533111
 34 423142124231241323 35 524415342341154511 36 431241423531214311
 37 151222121421141413 38 342143531454133541 39 342124325132141235
 40 442421422321313223 41 411341134114122434 42 124333224232434133
 43 342222124412221521 44 331422224311344343 45 523222441214141453
 46 332325444322322225 47 442244212532412252 48 235451235314313322
 49 133114422252312124 50 242434413113342131
;
```

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* alternative. The second choice was 3. The data set SASUSER.ORDERS shows in the second observation that this choice of 3 corresponds to the first alternative (in the third column variable, **Col3 = 1**) of choice set **Set= 8**. This DATA step writes out the data after the original order has been restored. It matches the data on page 102.

```
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 334523313433343413   2 314423343433333215   3 333423323323333233
  4 314432343444343254   5 315423313421543213   6 334433344423323233
  7 334433343432343413   8 334423323434343214   9 333433313351343213
 10 345423325322343233  11 333433343433323213  12 314423343423333253
 13 534423313322343213  14 214421144423323214  15 334423345433343235
 16 333433345333335313  17 534423343452343413  18 344435544432343513
 19 334433345432343433  20 311423343422325513  21 334453543423543213
 22 314423353323333513  23 333423343333333433  24 545422323322323213
 25 354433343433333313  26 314425525422343214  27 334353523423353213
 28 334433313333333233  29 333423543335353234  30 334453343533343433
 31 354423344322333413  32 354422343323333213  33 314423343352343215
 34 334423343443333213  35 314553344453343215  36 333433544433343233
 37 314423343424543214  38 353433324423353533  39 333453323333323513
 40 314433343422333214  41 334423344442343444  42 334433323422323213
 43 333423354433343213  44 314423323422333213  45 314452544422343214
 46 414423345423543214  47 544423544442343414  48 335453343423323453
 49 314523344424333214  50 334423343332343413
```

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 problem. 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.

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, with factors grouped by attribute and 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 report are documented starting on page 287.) 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	Taguchi, 1987
54	3 ** 24 6 ** 1	Hedayat, Sloane, and Stufken, 1999
54	3 ** 18 18 ** 1	Hedayat, Sloane, and Stufken, 1999

```
72    2 ** 23  3 ** 24                        Dey, 1985
72    2 ** 20  3 ** 24  4 **  1               Wang, 1996
72    2 ** 16  3 ** 25                        Wang, 1996
72    2 ** 14  3 ** 24  6 **  1               Wang, 1996
72    2 ** 13  3 ** 25  4 **  1               Wang, 1996
72    2 ** 12  3 ** 24 12 **  1               Hedayat, Sloane, and Stufken, 1999
72    2 ** 11  3 ** 24  4 **  1  6 **  1      Wang, 1996
72             3 ** 25  8 **  1               Hedayat, Sloane, and Stufken, 1999
72             3 ** 24 24 **  1               Hedayat, Sloane, and Stufken, 1999
```

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) and the number of choice sets must be a multiple of 9. Note however, that zero violations does not imply that a 100% efficient design exists. It just means that 100% efficiency is not precluded by unequal frequencies. In fact. the **%MktEx** orthogonal design catalogue 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 an 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 nonorthogonal designs.

We can see what orthogonal designs with three-level factors are available in 36 runs as follows. The macro **%MktOrth** 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**, one of the (sometimes many) references for 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 can sort this data set, excluding all but the 36-run designs, such that designs with the most three-level factors are printed first.

```
%mktorth;

proc sort data=mktdeslev out=list(drop=x:);
   by descending x3;
   where n = 36;
   run;

proc print; run;
```

Vacation Example

Obs	n	Design		Reference
1	36	2 ** 4	3 ** 13	Taguchi, 1987
2	36		3 ** 13 4 ** 1	Dey, 1985
3	36	2 ** 11	3 ** 12	Taguchi, 1987
4	36	2 ** 2	3 ** 12 6 ** 1	Wang and Wu, 1991
5	36		3 ** 12 12 ** 1	Wang and Wu, 1991
6	36	2 ** 1	3 ** 8 6 ** 2	Zhang, Lu, and Pang, 1999
7	36		3 ** 7 6 ** 3	Finney, 1982
8	36	2 ** 13	3 ** 4	Suen, 1989
9	36	2 ** 4	3 ** 3 6 ** 1	Hedayat, Sloane, and Stufken, 1999
10	36	2 ** 20	3 ** 2	Hedayat, Sloane, and Stufken, 1999
11	36	2 ** 11	3 ** 2 6 ** 1	Hedayat, Sloane, and Stufken, 1999
12	36	2 ** 2	3 ** 2 6 ** 2	Hedayat, Sloane, and Stufken, 1999
13	36	2 ** 27	3 ** 1	Hedayat, Sloane, and Stufken, 1999
14	36	2 ** 18	3 ** 1 6 ** 1	Hedayat, Sloane, and Stufken, 1999
15	36	2 ** 9	3 ** 1 6 ** 2	Hedayat, Sloane, and Stufken, 1999
16	36	2 ** 35		Hadamard
17	36	2 ** 13	9 ** 1	Suen, 1989
18	36	2 ** 2	18 ** 1	Hedayat, Sloane, and Stufken, 1999
19	36	2 ** 1	6 ** 3	SAS Procedure OPTEX

There are 13 two-level factors available in 36 runs, and we need 15, 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=7654321)
```

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. 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 123.

```
NOTE: Generating the candidate set.
NOTE: Performing 20 searches of 81 candidates, full-factorial=28,697,814.
NOTE: Generating the tabled design, n=36.
```

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	82.2172	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		
3	Start	79.1390		Tab,Ran
3	19 15	98.1101	98.1101	
3	19 15	98.6368	98.6368	
3	End	98.6368		
.				
.				
.				
5	Start	79.6983		Tab,Ran
5	30 14	98.8933	98.8933	
5	End	98.8933		
.				
.				
.				
11	Start	80.5274		Tab,Ran
11	End	98.4043		
12	Start	51.8915		Ran,Mut,Ann
12	End	93.0214		
.				
.				
.				
21	Start	47.6990		Ran,Mut,Ann
21	End	93.6096		

Vacation Example

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		
2	Start	82.0528		Tab,Ran
2	End	98.7672		
.				
.				
.				
43	Start	80.4332		Tab,Ran
43	35 14	98.9438	98.9438	
43	End	98.9438		
.				
.				
.				
62	Start	74.7712		Tab,Ran
62	16 15	98.9438	98.9438	
62	End	98.9438		
.				
.				
.				
84	Start	81.9302		Tab,Ran
84	9 15	98.9438	98.9438	
84	End	98.9438		
.				
.				
.				
124	Start	85.2024		Tab,Ran
124	21 14	98.9438	98.9438	
124	End	98.9438		
.				
.				
.				
149	Start	78.9082		Tab,Ran
149	31 14	98.9438	98.9438	
149	End	98.9438		

NOTE: Stopping since it appears that no improvement is possible.

Vacation Example

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		
.				
.				
.				
10	Start	90.7390		Pre,Mut,Ann
10	End	92.3775		

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

Vacation Example

The OPTEX Procedure

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 5.24 minutes and found a design that is almost 99% 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 and PROC OPTEX and are nothing to worry about.)

The %MktEx Macro Algorithm

The **%MktEx** macro creates 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% 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.

The **%MktEx** macro can directly generate, without iterations, hundreds of different 100% D-efficient, orthogonal and balanced, tabled designs. It does this using its design catalogue and many different general and ad hoc algorithms. See page 328 for a list of some of the orthogonal designs that the **%MktEx** macro knows how to make. 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 design is a *candidate set* that in the second part is searched by PROC OPTEX using the modified Federov algorithm. A design is built from a selection of the rows of the candidate set (Federov, 1972; Cook and Nachtsheim, 1980). The modified Federov 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 28 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 82.2172% 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 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 columns can be initialized with 14 columns of $2^4 3^{13}$, and the other two columns are randomly initialized. Levels that are not orthogonally initialized may be exchanged for other levels if the exchange increases efficiency. For example, the iteration history for this example shows that the macro exchanged levels in row 5 column 14, row 6 column 14, row 6 column 15, and so on.

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 every 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 small (up to several thousand), the modified Federov algorithm is usually superior to coordinate exchange. When the full-factorial design is larger, coordinate exchange is usually the superior approach. However, heuristics like this 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 98.0624% efficient (Design 2, **End**). In design 3 and subsequent designs, the macro uses this same approach, but different random initializations of the remaining two columns. In design 5, the **%MktEx** macro finds a design that is 98.8933% 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 all of the other levels, one level of 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 efficiency of 98.8933. In designs 43, 62, 84, 124, and 149, it finds a design that is 98.9438% efficient. After iteration 149, 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.

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 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. In this case the macro fails to improve the design. 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 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 swaps increases. For design 1, for the first level of the first factor, by default, the macro may execute a swap (say change a 2 to a 1), that makes the design worse, with probability 0.05. As more and more swaps occur, this probability decreases so at the end of the processing of design 1, swaps 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. Recall however, that 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 less than 6 minutes. 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 20 to 45 minutes. Since the tabled initialization worked best, run time was on the order of several minutes. 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=**, 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     6 3 3 3 6 3 3 3 6
          x13 x16     6 6 6 6 6 6
  *       x14 x15     3 6 3 6 3 3 3 3 6
          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 covariance 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

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	98.9099	97.8947	98.9418	0.9280

Information Matrix

	Intercept	x11	x12	x21	x22	x31	x32	x41	x42	x51	x52
Intercept	36	0	0	0	0	0	0	0	0	0	0
x11	0	36	-0	-0	0	0	-0	0	-0	0	0
x12	0	-0	36	0	-0	-0	0	0	-0	0	-0
x21	0	-0	0	36	0	-0	-0	0	0	0	-0
x22	0	0	-0	0	36	0	-0	-0	-0	0	-0
x31	0	0	-0	-0	0	36	-0	-0	0	0	0
x32	0	-0	0	-0	-0	-0	36	0	0	0	-0
x41	0	0	0	0	-0	-0	0	36	-0	-0	-0
x42	0	-0	-0	0	-0	0	0	-0	36	0	-0
x51	0	0	0	0	0	0	0	-0	0	36	-0

.
.
.

Information Matrix

	x112	x121	x122	x131	x132	x141	x142	x151	x152
x112	36	-0	-0	0	0	0	-0	-0	-0
x121	-0	36	-0	-0	-0	-0	-0	0	0
x122	-0	-0	36	-0	0	-0	-0	-0	-0
x131	0	-0	-0	36	-0	-5	-8	9	-0
x132	0	-0	0	-0	36	8	-5	-0	-9
x141	0	-0	-0	-5	8	36	-0	-5	-8
x142	-0	-0	-0	-8	-5	-0	36	-8	5
x151	-0	0	-0	9	-0	-5	-8	36	-0
x152	-0	0	-0	-0	-9	-8	5	-0	36

Variance Matrix

	Intercept	x11	x12	x21	x22	x31
Intercept	0.0278	-0.0000	-0.0000	-0.0000	-0.0000	-0.0000
x11	-0.0000	0.0278	0.0000	0.0000	-0.0000	-0.0000
x12	-0.0000	0.0000	0.0278	-0.0000	0.0000	0.0000
x21	-0.0000	0.0000	-0.0000	0.0278	-0.0000	0.0000
x22	-0.0000	-0.0000	0.0000	-0.0000	0.0278	-0.0000
x31	-0.0000	-0.0000	0.0000	0.0000	-0.0000	0.0278

.
.
.

Variance Matrix

	x131	x132	x141	x142	x151	x152
x131	0.0309	0.0000	0.0031	0.0053	-0.0062	0.0000
x132	0.0000	0.0309	-0.0053	0.0031	0.0000	0.0062
x141	0.0031	-0.0053	0.0309	0.0000	0.0031	0.0053
x142	0.0053	0.0031	0.0000	0.0309	0.0053	-0.0031
x151	-0.0062	0.0000	0.0031	0.0053	0.0309	0.0000
x152	0.0000	0.0062	0.0053	-0.0031	0.0000	0.0309

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** make their variances larger than the other factors (0.0309 versus 0.0278).

This variance matrix is a little hard to look at. All of the 0.0000 and -0.0000's tend to obscure the nonzeros. We can use ODS along with PROC FORMAT and PROC PRINT to make a better display. The variance matrix is excluded from the printed output and instead is output to a SAS data set. The **persist** options are used since the ODS statements need to persist through the macro steps until the macro reaches the PROC OPTEX step. In Version 9.0 and previous SAS versions, the **match_all** option must be specified with **persist** on the **ods output** statement. PROC FORMAT is used to construct a format so that the values within rounding error of zero print as '0'. PROC PRINT is called to print the results. The label statement gives the row ID variable, **rowname** a null header.

```
ods exclude 'variance matrix'(persist);
ods output 'variance matrix'(persist match_all)=v;
%mktex(3 ** 15,  n=&n * &blocks, init=randomized(drop=x16),
       options=check, examine=v)

proc format;
   value zer -1e-8 - 1e-8 = ' 0     ';
   run;

proc print label data=v(drop=_:);
   format _numeric_ zer7.4;
   label rowname = '00'x;
   id rowname;
   run;
```

Vacation Example

	Intercept	x11	x12	x21	x22	x31	x32	x41
Intercept	0.0278	0	0	0	0	0	0	0
x11	0	0.0278	0	0	0	0	0	0
x12	0	0	0.0278	0	0	0	0	0
x21	0	0	0	0.0278	0	0	0	0
x22	0	0	0	0	0.0278	0	0	0
x31	0	0	0	0	0	0.0278	0	0
x32	0	0	0	0	0	0	0.0278	0
x41	0	0	0	0	0	0	0	0.0278

.
.
.

	x122	x131	x132	x141	x142	x151	x152
x112	0	0	0	0	0	0	0
x121	0	0	0	0	0	0	0
x122	0.0278	0	0	0	0	0	0
x131	0	0.0309	0	0.0031	0.0053	-0.0062	0
x132	0	0	0.0309	-0.0053	0.0031	0	0.0062
x141	0	0.0031	-0.0053	0.0309	0	0.0031	0.0053
x142	0	0.0053	0.0031	0	0.0309	0.0053	-0.0031
x151	0	-0.0062	0	0.0031	0.0053	0.0309	0
x152	0	0	0.0062	0.0053	-0.0031	0	0.0309

These next steps use the **%MktLab** macro to reassign the variable names, store the design in a permanent SAS data set, SASUSER.BLOCKDES, and then use the **%MktEx** macro to check the results. 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**. We made the correlated variables correspond to the least important attributes in different alternatives (in this case the scenery factors for Alaska, Mexico, and Maine).

```
%mktlab(data=randomized, vars=x1-x5 x11-x15 x6 x9 x7 x8 x10 Block,
        out=sasuser.blockdes)

%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
                .
                .
                .
                x9 x7         4 4 4 4 4 4 4 4 4
                x9 x8         4 4 4 4 4 4 4 4 4
                x9 x10        4 4 4 4 4 4 4 4 4
              *  x7 x8        3 3 6 6 3 3 3 6 3
              *  x7 x10       6 3 3 3 3 6 3 6 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
```

Generating the Questionnaire

This next DATA step prints the questionnaires. They are then copied and the data are collected.

```
title;
proc sort data=sasuser.blockdes; 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.blockdes;
   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 shorthand 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.

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 $1249.

2) Alaska, staying in a Hotel near a Lake,
with a package cost of $1499.

3) Mexico, staying in a Cabin near the Mountains,
with a package cost of $1249.

4) California, staying in a Hotel near the Beach,
with a package cost of $1249.

5) Maine, staying in a Cabin near the Beach,
with a package cost of $1249.

6) Stay at home this year.

2) Circle your choice of vacation destinations:

1) Hawaii, staying in a Hotel near the Mountains,
with a package cost of $999.

2) Alaska, staying in a Bed & Breakfast near a Lake,
with a package cost of $1249.

3) Mexico, staying in a Hotel near the Mountains,
with a package cost of $999.

4) California, staying in a Cabin near the Mountains,
with a package cost of $999.

5) Maine, staying in a Hotel near the Beach,
with a package cost of $999.

6) Stay at home this year.

In practice, data collection may be much more elaborate than this. It may involve art work, photographs, and the choice sets may be presented and data may be collected over the web. However the choice sets are presented and the data collected, the essential ingredients remain the same. Subjects are shown sets of alternatives and 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.

```
data results;
   input Subj Form (choose1-choose&n) (1.) @@;
   datalines;
  1  1 151312311511353143  2  2 414435155112312411  3  1 131311331154453453
  4  2 424434451134315111  5  1 131311111112442443  6  2 354434253152315111
  7  1 141543311511154143  8  2 414415211434353511  9  1 111512331151412153
 10  2 424333133154311511 11  1 151511331543123143 12  2 424155153114312511
 13  1 151542331412114143 14  2 323414141152313111 15  1 151511311511413113
 16  2 313435133154353113 17  1 154511311554152143 18  2 423434153121353111
 19  1 111311131315144413 20  2 413435331112413111 21  1 131341331533413143
.
.
.
;
```

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, **x1**, **x2**, **x3**, ... **x15**. We can run the **%MktKey** macro to get these names in the SAS log, for cutting and pasting into the program without typing them.

```
%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
```

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.blockdes, key=key, alt=place, out=rolled)
```

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.BLOCKDES 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 ROLLED from SASUSER.BLOCKDES 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.blockdes** data set will not appear in the final design. The purpose of the variable **Block** (sorting the design into blocks) has already been achieved. 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.blockdes(obs=2);
   id Block;
   var x1-x15;
   run;

proc print data=rolled(obs=12); run;
```

Vacation Example

Block	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15
1	2	3	1	3	1	2	2	1	3	3	2	3	2	2	2
1	3	2	3	1	3	1	2	1	1	3	1	2	1	1	1

Vacation Example

Obs	Set	Place	Lodge	Scene	Price
1	1	Hawaii	2	2	2
2	1	Alaska	3	2	3
3	1	Mexico	1	1	2
4	1	California	3	3	2
5	1	Maine	1	3	2
6	1	Home	.	.	.
7	2	Hawaii	3	1	1
8	2	Alaska	2	2	2
9	2	Mexico	3	1	1
10	2	California	1	1	1
11	2	Maine	3	3	1
12	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 rolled2;
   set rolled;
   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=rolled2(obs=12); run;
```

Vacation Example

Obs	Set	Place	Lodge	Scene	Price
1	1	Hawaii	Bed & Breakfast	Lake	1249
2	1	Alaska	Hotel	Lake	1499
3	1	Mexico	Cabin	Mountains	1249
4	1	California	Hotel	Beach	1249
5	1	Maine	Cabin	Beach	1249
6	1	Home	Home	Home	0
7	2	Hawaii	Hotel	Mountains	999
8	2	Alaska	Bed & Breakfast	Lake	1249
9	2	Mexico	Hotel	Mountains	999
10	2	California	Cabin	Mountains	999
11	2	Maine	Hotel	Beach	999
12	2	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. 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=rolled2, data=results, out=res2, blocks=form,
          nsets=&n, nalts=&m, setvars=choose1-choose&n)

proc print data=res2(obs=12); run;
```

This macro takes the **design=rolled2** 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 21600 observations (200 subjects who each saw 18 choice sets with 6 alternatives). Here are the first two choice sets.

Vacation Example

Obs	Subj	Form	Set	Place	Lodge	Scene	Price	c
1	1	1	1	Hawaii	Bed & Breakfast	Lake	1249	1
2	1	1	1	Alaska	Hotel	Lake	1499	2
3	1	1	1	Mexico	Cabin	Mountains	1249	2
4	1	1	1	California	Hotel	Beach	1249	2
5	1	1	1	Maine	Cabin	Beach	1249	2
6	1	1	1	Home	Home	Home	0	2
7	1	1	2	Hawaii	Hotel	Mountains	999	2
8	1	1	2	Alaska	Bed & Breakfast	Lake	1249	2
9	1	1	2	Mexico	Hotel	Mountains	999	2
10	1	1	2	California	Cabin	Mountains	999	2
11	1	1	2	Maine	Hotel	Beach	999	1
12	1	1	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 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. 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(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	1249	Lake	Bed & Breakfast
Alaska	1	1	1	2	1499	Lake	Hotel
Mexico	1	1	1	2	1249	Mountains	Cabin
California	1	1	1	2	1249	Beach	Hotel
Maine	1	1	1	2	1249	Beach	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	0	0	1	0	Lake
Alaska	0	0	1	0	Lake
Mexico	0	0	0	1	Mountains
California	1	0	0	0	Beach
Maine	1	0	0	0	Beach
Home	0	1	0	0	Home

Vacation Example

Place	Bed & Breakfast	Cabin	Home	Hotel	Lodge	0	999	1249	1499	Price
Hawaii	1	0	0	0	Bed & Breakfast	0	0	1	0	1249
Alaska	0	0	0	1	Hotel	0	0	0	1	1499
Mexico	0	1	0	0	Cabin	0	0	1	0	1249
California	0	0	0	1	Hotel	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 Bed & Breakfast column has a 1 for Hawaii since Hawaii has B & B lodging in this choice set. The Bed & Breakfast column has a 0 for Alaska since Alaska does not have B & B 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 146, 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 79 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

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	6365.337
AIC	12900.668	6387.337
SBC	12900.668	6455.412

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	6535.3316	11	<.0001
Score	5858.0723	11	<.0001
Wald	2158.0226	11	<.0001

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Hawaii	1	3.59233	0.41717	74.1524	<.0001
Alaska	1	0.73317	0.42650	2.9552	0.0856
Mexico	1	2.73979	0.41843	42.8735	<.0001
California	1	2.04754	0.42037	23.7249	<.0001
Maine	1	1.34308	0.42439	10.0153	0.0016
Home	0	0	.	.	.
0	0	0	.	.	.
999	1	3.51899	0.08910	1559.9275	<.0001
1249	1	1.23293	0.07617	261.9979	<.0001
1499	0	0	.	.	.
Beach	1	1.41301	0.07173	388.0235	<.0001
Home	0	0	.	.	.
Lake	1	0.71264	0.06292	128.2840	<.0001
Mountains	0	0	.	.	.
Bed & Breakfast	1	0.66322	0.05778	131.7652	<.0001
Cabin	1	-1.53715	0.07178	458.5709	<.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-zero parameter estimates in the 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

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	6445.698
AIC	12900.668	6465.698
SBC	12900.668	6527.585

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	6454.9702	10	<.0001
Score	5633.7761	10	<.0001
Wald	2243.9574	10	<.0001

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Hawaii	1	14.21266	0.46894	918.5650	<.0001
Alaska	1	11.46095	0.45555	632.9626	<.0001
Mexico	1	13.38866	0.46367	833.7971	<.0001
California	1	12.70294	0.46109	758.9939	<.0001
Maine	1	12.12909	0.45754	702.7502	<.0001
Price	1	-0.00729	0.0001785	1670.5323	<.0001
Beach	1	1.31418	0.06946	357.9800	<.0001
Lake	1	0.68484	0.06252	120.0016	<.0001
Mountains	0	0	.	.	.
Bed & Breakfast	1	0.64743	0.05770	125.9040	<.0001
Cabin	1	-1.47644	0.06929	453.9843	<.0001
Hotel	0	0	.	.	.

The results of the two different analyses are similar. The coefficients for the destinations all increase by a non-constant amount (approximately 10.65) 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 = 6454.9702 , compared to the previous value of 6535.3316 (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

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	6365.337
AIC	12900.668	6387.337
SBC	12900.668	6455.412

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	6535.3316	11	<.0001
Score	5858.0723	11	<.0001
Wald	2158.0226	11	<.0001

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Hawaii	1	4.82525	0.41472	135.3715	<.0001
Alaska	1	1.96610	0.42218	21.6877	<.0001
Mexico	1	3.97272	0.41540	91.4641	<.0001
California	1	3.28047	0.41666	61.9889	<.0001
Maine	1	2.57601	0.42133	37.3804	<.0001
Price 1	1	-1.75950	0.04455	1559.9275	<.0001
Price 2	1	0.52657	0.05879	80.2229	<.0001
Beach	1	1.41301	0.07173	388.0235	<.0001
Lake	1	0.71264	0.06292	128.2840	<.0001
Mountains	0	0	.	.	.
Bed & Breakfast	1	0.66322	0.05778	131.7652	<.0001
Cabin	1	-1.53715	0.07178	458.5709	<.0001
Hotel	0	0	.	.	.

The fit is exactly the same as when price was treated as qualitative, Chi-Square = 6535.3316. 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 1.23293. The coefficients for the factors after price are unchanged. The part-worth utility for \$999 is $-1.75950 \times (999-1249)/250 + 0.52657 \times ((999-1249)/250)^2 = 2.28607$, the part-worth utility for \$1249 is $-1.75950 \times (1249-1249)/250 + 0.52657 \times ((1249-1249)/250)^2 = 0$, and the part-worth utility for \$1499 is $-1.75950 \times (1499-1249)/250 + 0.52657 \times ((1499-1249)/250)^2 = -1.23293$, which differ from the coefficients when price was treated as qualitative, by a constant -1.23293.

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 139, 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 dummy 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 dummy 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 dummy variables for missing **class** levels. 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;
```

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
1	1	0	0	0	0	0	0	0	1	1
2	0	1	0	0	0	1	1	0	1	-1
3	0	0	1	0	0	0	0	-1	-1	0
4	0	0	0	1	0	0	0	1	0	-1
5	0	0	0	0	1	0	0	1	0	0
6	0	0	0	0	0	0	0	0	0	0

Obs	Cabin	Place	Price	Scene	Lodge	Subj	Set	Form	c
1	0	Hawaii	0	Lake	Bed & Breakfast	1	1	1	1
2	-1	Alaska	1	Lake	Hotel	1	1	1	2
3	1	Mexico	0	Mountains	Cabin	1	1	1	2
4	-1	California	0	Beach	Hotel	1	1	1	2
5	1	Maine	0	Beach	Cabin	1	1	1	2
6	0	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

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	6365.337
AIC	12900.668	6387.337
SBC	12900.668	6455.412

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	6535.3316	11	<.0001
Score	5858.0723	11	<.0001
Wald	2158.0226	11	<.0001

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Hawaii	1	5.24249	0.41160	162.2273	<.0001
Alaska	1	2.38334	0.41768	32.5603	<.0001
Mexico	1	4.38996	0.41216	113.4464	<.0001
California	1	3.69771	0.41325	80.0633	<.0001
Maine	1	2.99325	0.41631	51.6947	<.0001
Price 1	1	-1.75950	0.04455	1559.9275	<.0001
Price 2	1	0.52657	0.05879	80.2229	<.0001
Beach	1	0.70446	0.03931	321.2158	<.0001
Lake	1	0.00409	0.03390	0.0146	0.9040
Bed & Breakfast	1	0.95453	0.04069	550.3877	<.0001
Cabin	1	-1.24584	0.04754	686.7095	<.0001

It is instructive to compare the results of this analysis to the previous analysis on page 142. 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 0.41724, the price coefficients are the same, the scenery coefficients differ by a constant -0.70855, and the lodging coefficients differ by a constant 0.29131. Notice that $0.41724 + 0 + -0.70855 + 0.29131 = 0$, 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. Our new model allows for different price, scenery and lodging effects for each destination. The coding can be done with PROC TRANSREG and 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 133 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.blockdes, key=key, alt=place, out=rolled)

%mktmerge(design=rolled, 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 133. The **%MktMerge** macro usage differs from page 135 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	Bed & Breakfast	Lake	1249	1
2	1	1	1	Alaska	Hotel	Lake	1499	2
3	1	1	1	Mexico	Cabin	Mountains	1249	2
4	1	1	1	California	Hotel	Beach	1249	2
5	1	1	1	Maine	Cabin	Beach	1249	2
6	1	1	1		.	.	.	2
7	1	1	2	Hawaii	Hotel	Mountains	999	2
8	1	1	2	Alaska	Bed & Breakfast	Lake	1249	2
9	1	1	2	Mexico	Hotel	Mountains	999	2
10	1	1	2	California	Cabin	Mountains	999	2
11	1	1	2	Maine	Hotel	Beach	999	1
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. 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	0	0	1
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	Maine, 999	Maine, 1249	Maine, 1499
0	0	0	0	1	0	0	0	0
0	0	0	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	1	0
0	0	0	0	0	0	0	0	0

Mexico, 999	Mexico, 1249	Mexico, 1499	Alaska, Beach	Alaska, Lake	Alaska, Mountains	California, Beach	California, Lake	California, Mountains
0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0
0	1	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	0	0	0	0

Hawaii, Beach	Hawaii, Lake	Hawaii, Mountains	Maine, Beach	Maine, Lake	Maine, Mountains	Mexico, Beach	Mexico, Lake	Mexico, Mountains
0	1	0	0	0	0	0	0	0
0	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	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Alaska, Bed & Breakfast	Alaska, Cabin	Alaska, Hotel	California, Bed & Breakfast	California, Cabin	California, Hotel
0	0	0	0	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

Hawaii, Bed & Breakfast	Hawaii, Cabin	Hawaii, Hotel	Maine, Bed & Breakfast	Maine, Cabin	Maine, Hotel	Mexico, Bed & Breakfast	Mexico, Cabin	Mexico, Hotel
1	0	0	0	0	0	0	0	0
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	1	0	0	0	0
0	0	0	0	0	0	0	0	0

	Price	Scene	Lodge	Subj	Set	Form	c
Place							
	1249	Lake	Bed & Breakfast	1	1	1	1
Hawaii	1499	Lake	Hotel	1	1	1	2
Alaska	1249	Mountains	Cabin	1	1	1	2
Mexico	1249	Beach	Hotel	1	1	1	2
California	1249	Beach	Cabin	1	1	1	2
Maine	.	.	.	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

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	6340.136
AIC	12900.668	6410.136
SBC	12900.668	6626.741

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	6560.5318	35	<.0001
Score	6461.9024	35	<.0001
Wald	2047.3514	35	<.0001

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Hawaii	1	3.48997	0.42964	65.9821	<.0001
Alaska	1	-0.14055	0.67506	0.0434	0.8351
Mexico	1	2.84951	0.44410	41.1695	<.0001
California	1	2.08314	0.46841	19.7780	<.0001
Maine	1	1.24023	0.50709	5.9818	0.0145
Alaska, 999	1	4.14782	0.46236	80.4767	<.0001
Alaska, 1249	1	1.93389	0.49549	15.2336	<.0001
Alaska, 1499	0	0	.	.	.
California, 999	1	3.59002	0.19604	335.3481	<.0001
California, 1249	1	1.04103	0.22174	22.0424	<.0001
California, 1499	0	0	.	.	.

Hawaii, 999	1	3.61719	0.13710	696.0668	<.0001
Hawaii, 1249	1	1.24098	0.13098	89.7622	<.0001
Hawaii, 1499	0	0	.	.	.
Maine, 999	1	3.86257	0.25424	230.8154	<.0001
Maine, 1249	1	1.72816	0.28715	36.2202	<.0001
Maine, 1499	0	0	.	.	.
Mexico, 999	1	3.54171	0.15019	556.0958	<.0001
Mexico, 1249	1	1.22636	0.15974	58.9413	<.0001
Mexico, 1499	0	0	.	.	.
Alaska, Beach	1	1.67842	0.25920	41.9321	<.0001
Alaska, Lake	1	0.94543	0.24197	15.2667	<.0001
Alaska, Mountains	0	0	.	.	.
California, Beach	1	1.43286	0.16370	76.6131	<.0001
California, Lake	1	0.63691	0.16172	15.5103	<.0001
California, Mountains	0	0	.	.	.
Hawaii, Beach	1	1.60035	0.12782	156.7505	<.0001
Hawaii, Lake	1	1.02525	0.12842	63.7366	<.0001
Hawaii, Mountains	0	0	.	.	.
Maine, Beach	1	1.40565	0.18525	57.5772	<.0001
Maine, Lake	1	0.34761	0.19683	3.1189	0.0774
Maine, Mountains	0	0	.	.	.
Mexico, Beach	1	1.32202	0.14113	87.7431	<.0001
Mexico, Lake	1	0.63658	0.13770	21.3708	<.0001
Mexico, Mountains	0	0	.	.	.
Alaska, Bed & Breakfast	1	0.87845	0.21590	16.5553	<.0001
Alaska, Cabin	1	-1.72057	0.25842	44.3295	<.0001
Alaska, Hotel	0	0	.	.	.
California, Bed & Breakfast	1	0.64269	0.13784	21.7398	<.0001
California, Cabin	1	-1.32720	0.16997	60.9734	<.0001
California, Hotel	0	0	.	.	.
Hawaii, Bed & Breakfast	1	0.55551	0.12773	18.9151	<.0001
Hawaii, Cabin	1	-1.63720	0.12169	181.0058	<.0001
Hawaii, Hotel	0	0	.	.	.
Maine, Bed & Breakfast	1	0.60884	0.15443	15.5423	<.0001
Maine, Cabin	1	-1.76896	0.21614	66.9816	<.0001
Maine, Hotel	0	0	.	.	.
Mexico, Bed & Breakfast	1	0.64489	0.12357	27.2366	<.0001
Mexico, Cabin	1	-1.75349	0.17161	104.4079	<.0001
Mexico, Hotel	0	0	.	.	.

There are zero coefficients for the reference alternative. 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 $6560.5318 - 6535.3316 = 25.2002$ is distributed χ^2 with $35-11 = 24$ *df* ($p < 0.395$). This more complicated model does not account for significantly more variance than the simpler model.

Vacation Example, with Alternative-Specific Attributes

A researcher is interested in studying choice of vacation destinations. 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, 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. Here are two summaries of the design, with factors grouped by attribute and 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	$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

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.

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 report are documented starting on page 287.) 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	Wang, 1996
72	2 ** 13 3 ** 25 4 ** 1	Wang, 1996
72	2 ** 11 3 ** 24 4 ** 1 6 ** 1	Wang, 1996

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=7654321)
%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

Vacation Example with Asymmetry

The OPTEX Procedure

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% efficient design in 5.5 minutes. 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 x5*x15,
       seed=7654321)
```

We immediately get this message.

```
ERROR: More parameters than runs.
       If you really want to do this, specify RIDGE=.
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 also run the **%MktRuns** macro to help us choose the number of choice sets. However, the **%MktRuns** does not have a special syntax for interactions, you have to specify the main effects and interactions of two factors as if it were a single factor. For example, for the interaction of 2 three-level factors, you specify 9 in the list. For the interaction of a three-level factor and a four-level factor, you specify 12 in the list. Do not specify '3 3 9' or '3 4 12'; just specify '3' and '12'. In this example, we specify four 9's for the four accommodation/price interactions involving only three-level factors, one 12 for the California accommodation/price interaction, five 3's for scenery, and two 2's for the side trips. 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(9 9 9 9 12 3 3 3 3 3 2 2, max=45)
```

Vacation Example with Asymmetry

Design Summary

Number of Levels	Frequency
2	2
3	5
9	4
12	1

Vacation Example with Asymmetry

Saturated = 56
Full Factorial = 76,527,504

Some Reasonable Design Sizes	Violations	Cannot Be Divided By
72	30	27 81 108
81	33	2 4 6 12 18 24 36 108
90	39	4 12 24 27 36 81 108
96	57	9 18 27 36 81 108
60	59	9 18 24 27 36 81 108
63	59	2 4 6 12 18 24 27 36 81 108
84	59	9 18 24 27 36 81 108
99	59	2 4 6 12 18 24 27 36 81 108
66	61	4 9 12 18 24 27 36 81 108
78	61	4 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 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=7654321,
       interact=x1*x11 x2*x12 x3*x13 x4*x14 x5*x15)
```

The macro printed these notes to the log.

```
NOTE: Performing 20 searches of 243 candidates, full-factorial=76,527,504.
NOTE: Generating the tabled 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: $((123) \Rightarrow (121))$. The four-level factor in the candidate set is made from 2 three-level factors and coding down. $((1\ 2\ 3) \times (1\ 2\ 3) \Rightarrow (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9) \Rightarrow (1\ 2\ 3\ 4\ 1\ 2\ 3\ 4\ 1))$. 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		85.1188	85.1188	Can
1	End		85.1188		
2	Start		65.6743		Tab
2	46	11	85.1337	85.1337	
2	59	13	85.1617	85.1617	
2	64	1	85.1630	85.1630	
2	69	3	85.1663	85.1663	
2	1	2	85.3358	85.3358	
2	2	2	85.3706	85.3706	
2	2	12	85.4221	85.4221	
2	3	13	85.4374	85.4374	
2	14	1	85.4819	85.4819	
2	25	15	85.4824	85.4824	
2	28	5	85.5177	85.5177	
2	32	3	85.5350	85.5350	
2	42	5	85.5801	85.5801	
2	51	4	85.5826	85.5826	
2	59	13	85.6058	85.6058	
2	66	1	85.6455	85.6455	
2	7	15	85.6703	85.6703	
2	12	2	85.7176	85.7176	
2	40	1	85.7682	85.7682	
2	End		85.7682		

```
 3      Start       65.6743               Tab
 3     40   1       85.7682       85.7682
 3        End       85.7682

 4      Start       65.6743               Tab
 4     40   1       85.7682       85.7682
 4        End       85.7682

 .
 .
 .

11      Start       65.6743               Tab
11     40   1       85.7682       85.7682
11        End       85.7682

12      Start       56.4650               Ran,Mut,Ann
12     48  14       85.7817       85.7817
12     48  16       85.7888       85.7888
12     49   6       85.7888       85.7888
 .
 .
 .
12     40  17       89.5461       89.5461
12     58  17       89.5503       89.5503
12        End       89.5503

13      Start       56.9380               Ran,Mut,Ann
13     12  16       89.5646       89.5646
13     15  16       89.5674       89.5674
 .
 .
 .
13     62  17       89.8765       89.8765
13        End       89.8765

 .
 .
 .

16      Start       59.8285               Ran,Mut,Ann
16        End       89.0880

NOTE: Quitting the algorithm search step after 10.40 minutes and 16 designs.

                    Vacation Example with Asymmetry

                         Design Search History

                                 Current          Best
      Design    Row,Col  D-Efficiency  D-Efficiency  Notes
      ------------------------------------------------------
           0    Initial       89.8765       89.8765  Ini

           1      Start       57.5919               Ran,Mut,Ann
           1        End       89.2836

           .
           .
           .
```

```
              14      Start       58.7160                Ran,Mut,Ann
              14        End       88.7760

NOTE: Quitting the design search step after 20.95 minutes and 14 designs.

                      Vacation Example with Asymmetry

                         Design Refinement History

                                  Current          Best
          Design    Row,Col  D-Efficiency  D-Efficiency  Notes
          ------------------------------------------------------
               0    Initial       89.8765       89.8765  Ini

               1      Start       81.3493                Pre,Mut,Ann
               1        End       89.7066

               .
               .
               .

               4      Start       81.5586                Pre,Mut,Ann
               4        End       89.1031

NOTE: Quitting the refinenent step after 6.33 minutes and 4 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
```

Vacation Example with Asymmetry

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	89.8765	80.0119	95.5854	0.8819

The algorithm search history shows that the candidate-set approach (**Can**) used in design 1 found a design that was 85.1188% 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 111 used the coordinate-exchange algorithm with a tabled design initialization (**Tab**). This process found a design that was 85.7682% efficient. 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. In design 13, the macro finds a design with 89.8765% efficiency. After 16 iterations, the macro quit because the run time for the algorithm search exceeded 10 minutes (which is the default first value of the **maxtime=** option). The macro only checks the elapsed time after it finishes making a design. This is why huge problems with restrictions can take much longer.

The algorithm search phase picked the coordinate-exchange algorithm with a random initialization and 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 = 89.8765) found so far. The design search phase starts out with the initial design (**Ini**) found in the algorithm search phase. Usually, you will see improvement in the design search phase, however, in this case you do not. After 14 iterations, the macro quit because the run time for the algorithm search exceeded 20 minutes (which is the default second value of the **maxtime=** option).

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 efficiency of the starting design worse than the previous best efficiency. After 4 iterations, the macro quit because the run time for the algorithm search exceeded 5 minutes (which is the default third value of the **maxtime=** option).

All together, the macro used 37.78 minutes, which is more than the sum of the three reported times, because not all of the macro's calculations (most notably time spent in PROC FACTEX and OPTEX) are timed. Run time was just slightly more than the 10 + 20 + 5 = 35 maximum time specified by **maxtime=**. Recall that the macro stated that it ran 20 OPTEX iterations on 243 candidates. This will be very fast. If the full-factorial design had been smaller (a few thousand runs) the macro may have done more iterations using PROC OPTEX. This could have brought the run time up closer to an hour. When the full-factorial design is too big to search, and the macro uses a fractional-factorial design, it does not spend much time using OPTEX, because PROC OPTEX is probably not going to be the best approach. When the full-factorial design is manageable, the macro will spend more time in OPTEX, because there is a good chance that it will be the best approach. The macro picks the number of OPTEX iterations based on the size of the candidate set and the value of the **maxtime=** option.

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    0.13    0.18    0.11    0.07    0.15    0.07    0.09
x2      0.11    1       0.19    0.11    0.05    0.11    0.08    0.12    0.11
x3      0.13    0.19    1       0.17    0.09    0.06    0.12    0.09    0.07
x4      0.18    0.11    0.17    1       0.05    0.09    0.05    0.06    0.11
x5      0.11    0.05    0.09    0.05    1       0.09    0.11    0.05    0.08
x6      0.07    0.11    0.06    0.09    0.09    1       0.07    0.09    0.09
x7      0.15    0.08    0.12    0.05    0.11    0.07    1       0.11    0.09
x8      0.07    0.12    0.09    0.06    0.05    0.09    0.11    1       0.07
x9      0.09    0.11    0.07    0.11    0.08    0.09    0.09    0.07    1
.
.
.

                    Vacation Example with Asymmetry
              There are 0 Canonical Correlations Greater Than 0.316
                        * - Indicates Unequal Frequencies

                             Frequencies

              *    x1        25 23 24
              *    x2        22 24 26
              *    x3        21 26 25
              *    x4        24 22 26
                   x5        24 24 24
              *    x6        22 24 26
              *    x7        26 23 23
              *    x8        24 25 23
              *    x9        26 23 23
              *    x10       23 25 24
                   x11       24 24 24
              *    x12       23 24 25
              *    x13       23 24 25
              *    x14       19 16 18 19
              *    x15       22 25 25
              *    x16       34 38
              *    x17       38 34
```

```
*    x1 x2      7 8 10 6 9 8 9 7 8
*    x1 x3      8 8 9 8 8 7 5 10 9
*    x1 x4      10 5 10 6 9 8 8 8 8
*    x1 x5      10 7 8 7 8 8 7 9 8
*    x1 x6      8 8 9 7 7 9 7 9 8
*    x1 x7      9 10 6 8 6 9 9 7 8
*    x1 x8      8 9 8 8 7 8 8 9 7
*    x1 x9      9 8 8 7 8 8 10 7 7
*    x1 x10     8 8 9 9 8 6 6 9 9
*    x1 x11     8 8 9 7 8 8 9 8 7
*    x1 x12     7 8 10 8 8 7 8 8 8
*    x1 x13     8 9 8 7 7 9 8 8 8
*    x1 x14     7 6 6 6 5 4 7 7 7 6 5 6
*    x1 x15     6 9 10 8 6 9 8 10 6
*    x1 x16     12 13 10 13 12 12
*    x1 x17     13 12 12 11 13 11
*    x12 x13    8 7 8 6 9 9 9 8 8
*    x12 x14    7 6 5 5 6 4 7 7 6 6 6 7
*    x12 x15    7 7 9 6 10 8 9 8 8
*    x12 x16    10 13 11 13 13 12
*    x12 x17    12 11 12 12 14 11
*    x13 x14    7 4 5 7 7 4 7 6 5 8 6 6
*    x13 x15    8 7 8 9 7 8 5 11 9
*    x13 x16    10 13 13 11 11 14
*    x13 x17    12 11 13 11 13 12
*    x14 x15    5 7 7 5 4 7 4 7 7 8 7 4
*    x14 x16    9 10 7 9 9 9 9 10
*    x14 x17    10 9 9 7 9 9 10 9
*    x15 x16    11 11 13 12 10 15
*    x15 x17    11 11 13 12 14 11
*    x16 x17    19 15 19 19
     .
     .
     .
    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 1
```

We can 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)
```

In the interest of space, the results from this step are not shown.

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.blockdes, seed=114)
```

This step took 8.45 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. Perfect balance is impossible for the three level factors.

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	0.06	0.13	0.11	0.07	0.10	0.08	0.13	0.08
x1	0.06	1	0.11	0.13	0.18	0.11	0.07	0.15	0.07
x2	0.13	0.11	1	0.19	0.11	0.05	0.11	0.08	0.12
x3	0.11	0.13	0.19	1	0.17	0.09	0.06	0.12	0.09
x4	0.07	0.18	0.11	0.17	1	0.05	0.09	0.05	0.06
x5	0.10	0.11	0.05	0.09	0.05	1	0.09	0.11	0.05
x6	0.08	0.07	0.11	0.06	0.09	0.09	1	0.07	0.09
x7	0.13	0.15	0.08	0.12	0.05	0.11	0.07	1	0.11
x8	0.08	0.07	0.12	0.09	0.06	0.05	0.09	0.11	1

Vacation Example with Asymmetry
Summary of Frequencies
There are 0 Canonical Correlations Greater Than 0.316
*** - Indicates Unequal Frequencies**

	Block	18 18 18 18
*	x1	25 24 23
*	x2	22 24 26
*	x3	25 26 21
*	x4	22 26 24
	x5	24 24 24
*	x6	24 26 22
*	x7	23 26 23
*	x8	24 25 23
*	x9	23 23 26
*	x10	23 25 24
	x11	24 24 24
*	x12	24 23 25
*	x13	23 24 25
*	x14	18 16 19 19
*	x15	22 25 25
*	x16	34 38
*	x17	34 38

```
*    Block x1     6 6 6 6 6 6 7 6 5 6 6 6
*    Block x2     7 6 5 5 6 7 5 7 6 5 5 8
*    Block x3     7 6 5 6 6 6 7 7 4 5 7 6
*    Block x4     6 6 6 5 7 6 5 7 6 6 6 6
*    Block x5     7 5 6 5 7 6 6 6 6 6 6 6
*    Block x6     6 7 5 5 7 6 6 6 6 7 6 5
*    Block x7     4 7 7 6 6 6 7 6 5 6 7 5
*    Block x8     6 6 6 6 6 6 5 7 6 7 6 5
*    Block x9     6 6 6 5 6 7 6 5 7 6 6 6
*    Block x10    6 6 6 5 7 6 5 6 7 7 6 5
*    Block x11    6 6 6 6 6 6 5 6 7 7 6 5
*    Block x12    6 6 6 7 5 6 5 7 6 6 5 7
*    Block x13    7 5 6 5 7 6 5 6 7 6 6 6
*    Block x14    4 4 6 4 4 4 4 6 5 4 5 4 5 4 4 5
*    Block x15    5 7 6 6 5 7 6 6 6 5 7 6
*    Block x16    7 11 8 10 9 9 10 8
*    Block x17    8 10 9 9 9 9 8 10
     .
     .
     .

     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
```

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.blockdes;
   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 Cabin near the Beach,
       with a package cost of $1,749, and an optional helicopter
       flight to an active volcano for an additional $200.

    2) Alaska, staying in a Cabin near a Lake,
       with a package cost of $1,249.

    3) Mexico, staying in a Cabin near a Lake,
       with a package cost of $1,249.

    4) California, staying in a Bed & Breakfast near the Beach,
       with a package cost of $999.

    5) Maine, staying in a Cabin near the Beach,
       with a package cost of $1,499.

    6) Stay at home this year.
```

```
2) Circle your choice of vacation destinations:

   1) Hawaii, staying in a Bed & Breakfast near the Mountains,
      with a package cost of $1,249, and an optional helicopter
      flight to an active volcano for an additional $200.

   2) Alaska, staying in a Bed & Breakfast near a Lake,
      with a package cost of $1,249.

   3) Mexico, staying in a Bed & Breakfast near the Mountains,
      with a package cost of $999.

   4) California, staying in a Hotel near a Lake,
      with a package cost of $1,499.

   5) Maine, staying in a Hotel near the Mountains,
      with a package cost of $1,499.

   6) Stay at home this year.
```

In practice, data collection may be much more elaborate than this. It may involve art work, photographs, and the choice sets may be presented and data may be collected over the web. However the choice sets are presented and the data collected, the essential ingredients remain the same. Subjects are shown sets of alternatives and asked to make a choice, and 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 occuring after the data have already been collected. See page 215 for an alternative method of testing your design. This step generates data for 300 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.blockdes point=n;
            do dest = 1 to &mm1;
               u[dest] = dests[dest] + lodging[x[dest]] +
                         scenes[x[&mm1 + dest]] -
                         x[2 * &mm1 + dest] +
                         2 * normal(7);
               end;
```

```
            u[1] = u[1] + (x16 = 1);
            u[3] = u[3] + (x17 = 1);
            u&m  = -3 + 3 * normal(7);
            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(7)**. 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(7)**. 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 431313344341313133   2 2 314113115514415413   3 3 331313331143431411
  4 4 431133311134311143   5 1 431151131141311333   6 2 111113113514335111
  7 3 341513331311451134   8 4 431113211334311511   9 1 411121131141311153
.
.
.
;
```

The analysis proceeds in a fashion similar to before. Formats and the key to processing the design are created.

```
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  .
.           .   .    .    .
;
```

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 133 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 133 and 135. 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 page 211 for an example of handling more complicated asymmetries.

```
data temp;
   set sasuser.blockdes;
   x11 + 1;
   x12 + 1;
   run;

%mktroll(design=temp, key=key, alt=place, out=rolled)

%mktmerge(design=rolled, 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); run;
```

Here are the first three choice sets.

Vacation Example with Asymmetry

Obs	Subj	Form	Set	Place	Lodge	Scene	Price	Side	c
1	1	1	1	Hawaii	Cabin	Beach	1749	No	2
2	1	1	1	Alaska	Cabin	Lake	1249	.	2
3	1	1	1	Mexico	Cabin	Lake	1249	Side Trip	2
4	1	1	1	California	Bed & Breakfast	Beach	999	.	1
5	1	1	1	Maine	Cabin	Beach	1499	.	2
6	1	1	1		.	.	.	.	2
7	1	1	2	Hawaii	Bed & Breakfast	Mountains	1249	No	2
8	1	1	2	Alaska	Bed & Breakfast	Lake	1249	.	2
9	1	1	2	Mexico	Bed & Breakfast	Mountains	999	Side Trip	1
10	1	1	2	California	Hotel	Lake	1499	.	2
11	1	1	2	Maine	Hotel	Mountains	1499	.	2
12	1	1	2		.	.	.	.	2
13	1	1	3	Hawaii	Hotel	Lake	1499	Side Trip	1
14	1	1	3	Alaska	Bed & Breakfast	Mountains	1249	.	2
15	1	1	3	Mexico	Cabin	Mountains	1499	No	2
16	1	1	3	California	Cabin	Beach	1499	.	2
17	1	1	3	Maine	Cabin	Beach	1249	.	2
18	1	1	3		.	.	.	.	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 dummy 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 dummy 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 177.

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	0	1	1	0
2	0	1	0	0	0	0	1	0	0	0	1
3	0	0	1	0	0	0	1	0	0	0	1
4	0	0	0	1	0	1	0	0	0	1	0
5	0	0	0	0	1	0	0	1	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	0	1	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0
3	0	0	1	0	0	0	0	0	1
4	0	1	0	0	0	0	0	0	0
5	0	0	1	0	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	1749	Beach	Cabin	No	1	1	1	2
2	Alaska	1249	Lake	Cabin	.	1	1	1	2
3	Mexico	1249	Lake	Cabin	Side Trip	1	1	1	2
4	California	999	Beach	Bed & Breakfast	.	1	1	1	1
5	Maine	1499	Beach	Cabin	.	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 79 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

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	12829.273
AIC	25801.336	12857.273
SBC	25801.336	12953.618

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	12972.0636	14	<.0001
Score	12091.5568	14	<.0001
Wald	5019.1086	14	<.0001

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Hawaii	1	3.42197	0.21382	256.1254	<.0001
Alaska	1	-1.11090	0.25638	18.7745	<.0001
Mexico	1	2.09610	0.22046	90.4029	<.0001
California	1	1.32260	0.21993	36.1656	<.0001
Maine	1	0.61641	0.22344	7.6105	0.0058
999	1	2.08485	0.07072	868.9811	<.0001
1249	1	1.43088	0.06177	536.5227	<.0001
1499	1	0.72192	0.05905	149.4626	<.0001
1749	0	0	.	.	.
Beach	1	1.41944	0.04504	993.3905	<.0001
Lake	1	0.81476	0.04675	303.7324	<.0001
Mountains	0	0	.	.	.
Bed & Breakfast	1	0.74857	0.04146	325.9274	<.0001
Cabin	1	-1.38428	0.04908	795.6207	<.0001
Hotel	0	0	.	.	.
Alaska Side Trip	0	0	.	.	.
California Side Trip	0	0	.	.	.
Hawaii Side Trip	1	0.69748	0.05815	143.8627	<.0001
Maine Side Trip	0	0	.	.	.
Mexico Side Trip	1	0.58787	0.06192	90.1210	<.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 or unchosen. 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 726 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. The next example 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

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
.					
.					
.					
71	4	71	600	100	500
72	4	72	600	100	500
Total			36000	6000	30000

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 in parentheses, 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 179.

```
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;

title2;
```

Here are the results. (Recall that we used **`%phchoice(on)`** on page 79 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

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 dummy 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

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 185, 192, 379, and 383 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 print data=coded(obs=5) label; var subj set c brand price; run;
proc print data=coded(obs=5) label; var Brand:; run;
proc print data=coded(obs=5) label; var p1B:; id brand; run;
proc print data=coded(obs=5) label; var p2B:; id brand; run;
proc print data=coded(obs=5) label; var p3B:; id brand; run;
proc print data=coded(obs=5) label; var p4B:; id brand; run;
proc print data=coded(obs=5) label; var p5B:; 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	0.00	0.00	0.00	0.00	Brand 1
2	0	1	0	0	0	0.00	5.99	0.00	0.00	0.00	Brand 2
3	0	0	1	0	0	0.00	0.00	3.99	0.00	0.00	Brand 3
4	0	0	0	1	0	0.00	0.00	0.00	5.99	0.00	Brand 4
5	0	0	0	0	1	0.00	0.00	0.00	0.00	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	0.00	0.00	0.00	0.00
Brand 2	0.00	3.99	0.00	0.00	0.00
Brand 3	0.00	0.00	3.99	0.00	0.00
Brand 4	0.00	0.00	0.00	3.99	0.00
Other	0.00	0.00	0.00	0.00	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	0.00	0.00	0.00	0.00
Brand 2	0.00	5.99	0.00	0.00	0.00
Brand 3	0.00	0.00	5.99	0.00	0.00
Brand 4	0.00	0.00	0.00	5.99	0.00
Other	0.00	0.00	0.00	0.00	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	0.00	0.00	0.00	0.00
Brand 2	0.00	3.99	0.00	0.00	0.00
Brand 3	0.00	0.00	3.99	0.00	0.00
Brand 4	0.00	0.00	0.00	3.99	0.00
Other	0.00	0.00	0.00	0.00	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	0.00	0.00	0.00	0.00
Brand 2	0.00	5.99	0.00	0.00	0.00
Brand 3	0.00	0.00	5.99	0.00	0.00
Brand 4	0.00	0.00	0.00	5.99	0.00
Other	0.00	0.00	0.00	0.00	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	0.00	0.00	0.00	0.00
Brand 2	0.00	4.99	0.00	0.00	0.00
Brand 3	0.00	0.00	4.99	0.00	0.00
Brand 4	0.00	0.00	0.00	4.99	0.00
Other	0.00	0.00	0.00	0.00	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

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 179, 192, 379, and 383 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;

title2;
```

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

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

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

The PHREG Procedure

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 is the choice example from Kuhfeld, Tobias, and Garratt (1994), on page 25. 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 179, 185, 379, and 383 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-cross-effects 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.

	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

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, $\text{floor}(22 \times 3/4) = 16$; for the competitor and 32 runs, $\text{floor}(32 \times 2/3) = 21$. 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).

Designing the Choice Experiment

This example originated with Kuhfeld, Tobias, and Garratt (1994), long before the **%MktRuns** macro was programmed. 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 report are documented starting on page 287.) 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. This 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 page 280 for other examples of restrictions.)

We must use IML logical operators, which are not as rich as DATA step operators:

$=$	equals	not: EQ
$\wedge =$ or $\neg =$	not equals	not: NE
$<$	less than	not: LT
$<=$	less than or equal to	not: LE
$>$	greater than	not: GT
$>=$	greater than or equal to	not: GE
&	and	not: AND
|	or	not: OR
$\wedge$ or $\neg$	not	not: NOT

To restrict the design, we must specify **restrictions=***macro-name*, in this case **restrictions=bad**, that names the macro that quantifies badness. The first statement counts up the number of available alternatives. The second sets the actual badness values. If bad (the number available) is less than two or greater than 4, then the Boolean expression **((bad < 2) | (bad > 4))** is true or 1. When the expression is true, 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**. We could just set **bad** to zero when everything is fine and one otherwise, but it is better to help the macro by letting it know that when it switches from zero available to one available, it is going in the right direction. Here is the code.

```
title 'Consumer Food Product Example';

%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, seed=377, outr=sasuser.choicdes )
```

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 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 not tabled design that is very good for this problem, this is the kind of problem where we would expect the PROC OPTEX 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% 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	49.1839		Tab,Unb,Ran
2	1 1	77.7982		Conforms
2	End	79.6170		
.				
.				
.				
11	Start	48.5995		Tab,Ran
11	23 1	76.9197		Conforms
11	End	79.8631		
12	Start	25.4775		Ran,Mut,Ann
12	18 1	65.5812		Conforms
12	End	81.4087		
.				
.				
.				
21	Start	36.9535		Ran,Mut,Ann
21	1 1	73.6168		Conforms
21	End	83.1951		

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	85.4271		Can
1	2 1	85.4271	85.4271	Conforms
1	End	85.4271		

Consumer Food Product Example

Design Refinement History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
0	Initial	85.4271	85.4271	Ini
1	Start	73.6369		Pre,Mut,Ann
1	2 1	71.6187		Conforms
1	End	82.1633		
.				
.				
.				
10	Start	62.6495		Pre,Mut,Ann
10	2 1	69.0658		Conforms
10	End	82.0098		

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
x8	3	1 2 3

Consumer Food Product Example

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	85.4271	72.0480	98.0611	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 (49.1839 in design 2), the D-efficiency when the restrictions are met (77.7982, **Conforms**), and the D-efficiency for the final design (79.6170). 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 85.4271% D-efficiency. The design refinement step fails to improve on the best design. This step took 15 minutes.

As we will see in the next section, it turns out that this is a *very* good design for this example.* Usually, for this problem, we see final D-efficiencies less than 85.3. How did we happen to find a design this good? We just got lucky. The random number seed that happened to go into the second OPTEX run, which was a function of all of the random numbers generated previously by both OPTEX and the coordinate-exchange algorithm, was a really good one.

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. 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 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 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 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,

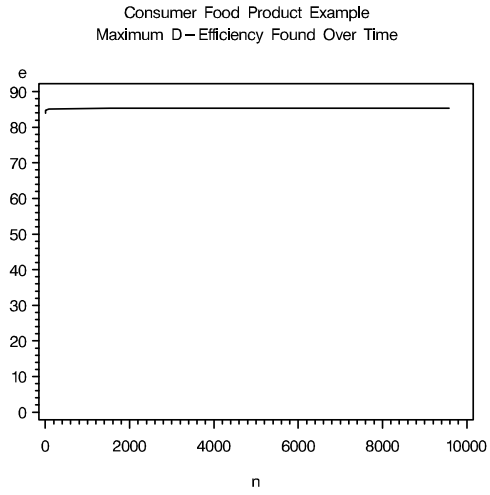
*Author's note: I do not know if this design is *the* optimal design. I know it is the best I have seen, and I frequently use this example as a test case. If I had to guess, I would guess it is not the optimal design, but it is really close.

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;

title2;
```

The plot of maximum D-efficiency as a function of PROC OPTEX run number clearly shows that the gain in 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 give 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 327) 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 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, 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.choicdes);
```

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	0.37	0.11	0.11	0.42	0.27	0.18	0.25
x2	0.37	1	0	0	0.38	0.50	0.25	0.08
x3	0.11	0	1	0.08	0.09	0.19	0.15	0.09
x4	0.11	0	0.08	1	0.25	0	0	0.09
x5	0.42	0.38	0.09	0.25	1	0.14	0.24	0.18
x6	0.27	0.50	0.19	0	0.14	1	0.10	0.29
x7	0.18	0.25	0.15	0	0.24	0.10	1	0.19
x8	0.25	0.08	0.09	0.09	0.18	0.29	0.19	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	x6	0.50	0.25
x1	x5	0.42	0.17
x2	x5	0.38	0.14
x1	x2	0.37	0.13

```
                Consumer Food Product Example
                    Summary of Frequencies
There are 4 Canonical Correlations Greater Than 0.316
          * - Indicates Unequal Frequencies

                   Frequencies

*    x1         6 5 7 8
*    x2         6 6 6 8
     x3         13 13
     x4         13 13
*    x5         8 9 9
*    x6         10 8 8
*    x7         14 12
*    x8         9 9 8
*    x1 x2      2 2 0 2 1 0 2 2 1 2 2 2 2 2 2 2
*    x1 x3      3 3 3 2 3 4 4 4
*    x1 x4      3 3 3 2 3 4 4 4
*    x1 x5      2 1 3 2 1 2 2 2 3 2 5 1
*    x1 x6      2 2 2 1 2 2 3 1 3 4 3 1
*    x1 x7      4 2 3 2 3 4 4 4
*    x1 x8      2 3 1 2 1 2 2 2 3 3 3 2
*    x2 x3      3 3 3 3 3 3 4 4
*    x2 x4      3 3 3 3 3 3 4 4
*    x2 x5      1 3 2 2 2 2 1 3 2 4 1 3
*    x2 x6      0 3 3 4 1 1 2 1 3 4 3 1
*    x2 x7      3 3 2 4 4 2 5 3
*    x2 x8      2 2 2 2 2 2 2 2 2 3 3 2
*    x3 x4      7 6 6 7
*    x3 x5      4 5 4 4 4 5
*    x3 x6      4 4 5 6 4 3
*    x3 x7      6 7 8 5
*    x3 x8      5 4 4 4 5 4
*    x4 x5      5 3 5 3 6 4
*    x4 x6      5 4 4 5 4 4
*    x4 x7      7 6 7 6
*    x4 x8      5 4 4 4 5 4
*    x5 x6      3 3 2 3 3 3 4 2 3
*    x5 x7      3 5 5 4 6 3
*    x5 x8      3 2 3 3 3 3 3 4 2
*    x6 x7      6 4 4 4 4 4
*    x6 x8      2 5 3 3 2 3 4 2 2
*    x7 x8      4 6 4 5 3 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
```

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 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

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, 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. The **%MktRuns** macro cannot provide us with much guidance with the interactions. We "tricked" it in the past by substituting products of levels, like $9 = 3 \times 3$, but in this case, factors like **x2**, **x3**, and **x4** interact multiple times, so it would not be that simple. We will try making a design in 36 runs, and see how it looks.

```
title 'Consumer Food Product Example';

%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=36, interact=x2*x3 x2*x4 x3*x4 x6*x7,
        restrictions=bad, seed=377, outr=sasuser.choicdes )

%mkteval;
```

Here is the last part of the output from the **%MktEx** macro.

Consumer Food Product Example

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Prediction Standard Error
1	94.6814	89.3664	94.3049	0.8333

D-efficiency at 94.68% 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	0.20	0	0	0.18	0.17	0.16	0.18
x2	0.20	1	0	0	0.13	0.39	0.28	0.18
x3	0	0	1	0.11	0.07	0.07	0	0.07
x4	0	0	0.11	1	0.07	0.07	0	0.07
x5	0.18	0.13	0.07	0.07	1	0.05	0.07	0.15
x6	0.17	0.39	0.07	0.07	0.05	1	0.07	0.08
x7	0.16	0.28	0	0	0.07	0.07	1	0.07
x8	0.18	0.18	0.07	0.07	0.15	0.08	0.07	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	8 10 8 10
*	x2	10 8 8 10
	x3	18 18
	x4	18 18
*	x5	9 11 16
*	x6	13 11 12
	x7	18 18
*	x8	12 9 15
	.	
	.	
	.	

The correlations are better, although we still have one a bit larger than we would like. However, the biggest problem is that the balance is much worse than we would like. 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.

```
%mktex( 4 4 2 2 3 3 2 3, n=36, interact=x2*x3 x2*x4 x3*x4 x6*x7,
       restrictions=bad, seed=377, outr=sasuser.choicdes, balance=2 )
%mkteval;
```

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	94.0824	88.3946	93.3172	0.8333

The D-efficiency looks good. It is a little lower than before, but not much. Here is the first part of the output from the **%MktEval** macro.

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	0.24	0.08	0.08	0.18	0.21	0.13	0.11
x2	0.24	1	0	0	0.24	0.17	0.10	0.32
x3	0.08	0	1	0	0.12	0.12	0.17	0
x4	0.08	0	0	1	0.07	0.07	0.06	0
x5	0.18	0.24	0.12	0.07	1	0.17	0.10	0.14
x6	0.21	0.17	0.12	0.07	0.17	1	0.04	0.10
x7	0.13	0.10	0.17	0.06	0.10	0.04	1	0.08
x8	0.11	0.32	0	0	0.14	0.10	0.08	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	x8	0.32	0.10

The canonical correlations look good. One is just large enough to be flagged $(0.32 > 0.316)$, but r^2 is only 0.1. Here is the last part of the output from the **%MktEval** macro.

```
            Consumer Food Product Example
                Summary of Frequencies
There is 1 Canonical Correlation Greater Than 0.316
          * - Indicates Unequal Frequencies

                Frequencies

*     x1          8 10 9 9
*     x2          8 8 10 10
      x3          18 18
      x4          18 18
*     x5          11 13 12
*     x6          11 12 13
*     x7          17 19
      x8          12 12 12
*     x1 x2       2 2 2 2 2 2 2 4 2 3 2 2 2 1 4 2
*     x1 x3       4 4 5 5 5 4 4 5
*     x1 x4       4 4 5 5 5 4 4 5
*     x1 x5       3 2 3 3 4 3 2 3 4 3 4 2
*     x1 x6       2 3 3 2 4 4 3 2 4 4 3 2
*     x1 x7       3 5 5 5 4 5 5 4
*     x1 x8       3 2 3 3 4 3 3 3 3 3 3 3
*     x2 x3       4 4 4 4 5 5 5 5
*     x2 x4       4 4 4 4 5 5 5 5
*     x2 x5       2 3 3 2 3 3 4 2 4 3 5 2
*     x2 x6       2 3 3 2 2 4 4 3 3 3 4 3
*     x2 x7       4 4 3 5 5 5 5 5
*     x2 x8       2 2 4 2 2 4 4 4 2 4 4 2
      x3 x4       9 9 9 9
*     x3 x5       5 6 7 6 7 5
*     x3 x6       5 7 6 6 5 7
*     x3 x7       7 11 10 8
      x3 x8       6 6 6 6 6 6
*     x4 x5       6 6 6 5 7 6
*     x4 x6       6 6 6 5 6 7
*     x4 x7       8 10 9 9
      x4 x8       6 6 6 6 6 6
*     x5 x6       4 4 3 3 4 6 4 4 4
*     x5 x7       6 5 6 7 5 7
*     x5 x8       4 3 4 4 4 5 4 5 3
*     x6 x7       5 6 6 6 6 7
*     x6 x8       4 4 3 4 4 4 4 4 5
*     x7 x8       6 6 5 6 6 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**, but for this problem, the price in efficiency is too high. We do want 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. It is balanced in this design. If it had not been, we could have switched it with another two-level factor or tried again with a different seed. If nothing else worked, we could have added it after the fact by blocking (running the **%MktBlock** macro as if we were adding a blocking factor).

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. Most of our restrictions are imposed within each row. Those kinds of restrictions do not pose a problem for the macro. Balance restrictions are imposed across rows within a column. We know of better ways to impose balance, but they tend to be very slow. This is an area where more research is needed, and the way this option works will quite likely be different in future releases. If perfect balance is critical, 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 report 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.choicdes(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	66.9776	54.6662	82.8184	0.6939

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	73.7719	66.4421	87.3561	0.7071

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	69.9539	59.0519	83.0652	0.7360

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	70.8672	57.5401	80.8955	0.7518

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	65.7047	50.8521	84.5539	0.7360

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.choicdes;
   y = 1;
   run;
proc glm data=temp;
   model y = x1-x8 x2*x3 x2*x4 x3*x4 x6*x7 / e aliasing;
   run;
```

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, 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**. This list will generate all of the effects of interest, **x1-x8 x2*x3 x2*x4 x3*x4 x6*x7**, and all of the two-way and three-way interactions. It is not a problem that some of the interactions 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;
```

```
Intercept - 174.72*x4*x6 - 476.67*x1*x4*x6 - 490.37*x2*x4*x6 - 359.77*x3*x4*x6
+ 204.67*x5*x6 + 742.7*x1*x5*x6 + 698.48*x2*x5*x6 + 444.15*x3*x5*x6 -
30.82*x4*x5*x6 - 30.013*x1*x7 + 174.13*x2*x7 + 412.08*x1*x2*x7 + 149.33*x3*x7
+ 291.03*x1*x3*x7 + 748.79*x2*x3*x7 + 33.824*x4*x7 - 34.861*x1*x4*x7 +
453.21*x2*x4*x7 + 260.2*x3*x4*x7 + 197.6*x5*x7 + 430.1*x1*x5*x7 +
871.72*x2*x5*x7 + 623.82*x3*x5*x7 + 288.4*x4*x5*x7 + 59.967*x1*x6*x7 +
335.16*x2*x6*x7 + 313.37*x3*x6*x7 - 202.24*x4*x6*x7 + 701.86*x5*x6*x7 -
28.098*x1*x8 + 35.022*x2*x8 + 190.57*x1*x2*x8 - 49.406*x3*x8 - 134.53*x1*x3*x8
- 158.84*x2*x3*x8 - 138.5*x4*x8 - 505.81*x1*x4*x8 - 454.8*x2*x4*x8 -
347.43*x3*x4*x8 + 190.55*x5*x8 + 659.05*x1*x5*x8 + 470.76*x2*x5*x8 +
344.34*x3*x5*x8 - 19.226*x4*x5*x8 + 55.161*x6*x8 + 32.377*x1*x6*x8 +
260.89*x2*x6*x8 - 48.524*x3*x6*x8 - 547.3*x4*x6*x8 + 829.99*x5*x6*x8 -
120.95*x7*x8 - 338.09*x1*x7*x8 + 48.151*x2*x7*x8 + 112.34*x3*x7*x8 -
290.99*x4*x7*x8 + 417.15*x5*x7*x8 - 61.492*x6*x7*x8

x1 + 36.198*x4*x6 + 98.299*x1*x4*x6 + 102.07*x2*x4*x6 + 71.034*x3*x4*x6 -
28.544*x5*x6 - 111.22*x1*x5*x6 - 98.978*x2*x5*x6 - 68.98*x3*x5*x6 +
28.763*x4*x5*x6 + 1.2817*x1*x7 - 25.266*x2*x7 - 74.585*x1*x2*x7 - 29.591*x3*x7
- 69.14*x1*x3*x7 - 131.14*x2*x3*x7 - 8.8034*x4*x7 - 10.134*x1*x4*x7 -
78.619*x2*x4*x7 - 51.281*x3*x4*x7 - 46.501*x5*x7 - 118.36*x1*x5*x7 -
172.7*x2*x5*x7 - 132.56*x3*x5*x7 - 72.363*x4*x5*x7 - 23.906*x1*x6*x7 -
36.943*x2*x6*x7 - 59.657*x3*x6*x7 + 38.937*x4*x6*x7 - 132.19*x5*x6*x7 +
15.948*x1*x8 + 3.4064*x2*x8 + 29.796*x1*x2*x8 + 6.4151*x3*x8 + 27.738*x1*x3*x8
+ 40.475*x2*x3*x8 + 25.464*x4*x8 + 108.13*x1*x4*x8 + 99.485*x2*x4*x8 +
61.056*x3*x4*x8 - 37.732*x5*x8 - 111.6*x1*x5*x8 - 65.647*x2*x5*x8 -
70.116*x3*x5*x8 + 3.9099*x4*x5*x8 - 13.614*x6*x8 + 7.0483*x1*x6*x8 -
30.651*x2*x6*x8 + 1.3753*x3*x6*x8 + 106.47*x4*x6*x8 - 137.07*x5*x6*x8 +
18.312*x7*x8 + 59.267*x1*x7*x8 + 11.27*x2*x7*x8 - 34.681*x3*x7*x8 +
40.508*x4*x7*x8 - 106.81*x5*x7*x8 - 9.4494*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.choicdes, out=sasuser.blockdes, nblocks=2, seed=289)
```

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 is 1 Canonical Correlation Greater Than 0.316

	Block	x1	x2	x3	x4	x5	x6	x7	x8
Block	1	0.08	0	0	0	0.07	0.07	0.06	0.14
x1	0.08	1	0.24	0.08	0.08	0.18	0.21	0.13	0.11
x2	0	0.24	1	0	0	0.24	0.17	0.10	0.32
x3	0	0.08	0	1	0	0.12	0.12	0.17	0
x4	0	0.08	0	0	1	0.07	0.07	0.06	0
x5	0.07	0.18	0.24	0.12	0.07	1	0.17	0.10	0.14
x6	0.07	0.21	0.17	0.12	0.07	0.17	1	0.04	0.10
x7	0.06	0.13	0.10	0.17	0.06	0.10	0.04	1	0.08
x8	0.14	0.11	0.32	0	0	0.14	0.10	0.08	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 is 1 Canonical Correlation Greater Than 0.316
*** - Indicates Unequal Frequencies**

		Frequencies
	Block	18 18
*	x1	8 10 9 9
*	x2	8 8 10 10
	x3	18 18
	x4	18 18
*	x5	11 13 12
*	x6	11 12 13
*	x7	17 19
	x8	12 12 12
*	Block x1	4 5 4 5 4 5 5 4
*	Block x2	4 4 5 5 4 4 5 5
	Block x3	9 9 9 9
	Block x4	9 9 9 9
*	Block x5	6 6 6 5 7 6
*	Block x6	6 6 6 5 6 7
*	Block x7	9 9 8 10
*	Block x8	5 6 7 7 6 5

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	3	2	1	1	1	1	2	3
	2	4	4	1	2	2	2	1	3
	3	3	4	2	2	1	3	1	2
	4	4	3	1	2	1	1	1	1
	5	4	4	2	1	3	1	2	1
	6	3	1	2	1	2	2	2	3
	7	1	1	1	2	1	2	2	3
	8	2	1	1	1	3	3	2	3
	9	2	2	1	2	3	3	1	3
	10	1	3	2	2	3	2	1	2
	11	2	2	2	1	2	3	1	2
	12	2	4	1	2	2	1	2	1
	13	1	1	2	1	1	3	1	1
	14	3	3	1	2	3	1	2	2
	15	4	3	2	2	2	3	1	3
	16	2	4	2	1	2	2	1	2
	17	4	3	2	1	1	1	2	2
	18	1	2	1	1	3	2	2	1

Consumer Food Product Example

Block	Run	x1	x2	x3	x4	x5	x6	x7	x8
2	1	4	2	2	2	1	2	1	1
	2	3	1	1	2	3	2	1	1
	3	4	1	2	2	2	3	2	3
	4	3	4	1	1	2	3	1	1
	5	2	3	1	1	1	2	1	3
	6	2	3	2	2	1	3	2	1
	7	4	3	1	1	3	2	2	2
	8	2	4	1	1	1	2	2	2
	9	3	3	2	1	3	1	1	1
	10	1	3	1	1	2	3	2	1
	11	2	4	2	2	2	2	2	1
	12	2	1	2	2	3	1	1	2
	13	3	2	1	2	2	3	2	2
	14	3	2	2	1	3	3	2	3
	15	1	4	2	1	1	1	1	3
	16	4	1	1	1	2	1	1	2
	17	1	2	2	2	2	1	2	3
	18	1	4	1	2	3	3	2	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.

The Final Design

The following code creates the final choice design, stored in SASUSER.FINCHDES, 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, make 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. You may *not* use ordinary missing **'.'** for levels. 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 **._**.*

The **%MktLab** macro specification names the input SAS data set with the design. By default, it looks for an input **key=** data set named KEY and 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;

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.blockdes, key=key)

proc sort out=sasuser.finchdes(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
```

*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 ...**.

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	$2.09	$1.89	Micro	$1.99	$1.49	Stove	N
		N	N	Stove	N	$1.49	Stove	$1.99
		$2.09	$1.39	Stove	$2.49	$2.29	Stove	N
		$1.69	$1.39	Micro	N	N	Stove	N
		$1.69	$1.89	Stove	$2.49	N	Micro	$2.39
		$1.29	$1.39	Stove	$1.99	N	Micro	$1.99
		$1.69	N	Stove	$2.49	$2.29	Micro	$2.39
		N	$2.39	Stove	$1.99	$1.49	Stove	$2.39
		$1.29	$1.89	Micro	N	$2.29	Stove	$1.99
1	Talker	N	N	Micro	$2.49	$2.29	Micro	N
		$2.09	N	Stove	$1.99	N	Micro	$2.39
		N	$2.39	Micro	$1.99	$1.49	Micro	$1.99
		$1.29	$1.39	Micro	$1.99	$2.29	Stove	N
		$1.69	$1.89	Micro	N	N	Micro	N
		$1.29	$2.39	Stove	N	$2.29	Micro	$2.39
		$1.69	N	Micro	$2.49	$1.49	Stove	$1.99
		$2.09	$2.39	Micro	N	$1.49	Stove	$2.39
		N	$2.39	Stove	$2.49	N	Micro	N
2	No	$2.09	N	Micro	$2.49	N	Micro	$1.99
		$1.69	$2.39	Micro	$1.99	$2.29	Micro	N
		N	$2.39	Micro	N	$2.29	Stove	$2.39
		$1.69	N	Micro	$1.99	$2.29	Stove	$2.39
		$2.09	$2.39	Stove	N	$1.49	Micro	$1.99
		$1.29	$2.39	Micro	$2.49	N	Stove	$1.99
		$2.09	$1.89	Stove	N	N	Stove	N
		$1.29	N	Stove	$1.99	$1.49	Micro	N
		N	$1.39	Micro	$2.49	$1.49	Micro	$2.39
2	Talker	N	$1.89	Stove	$1.99	$2.29	Micro	$1.99
		$2.09	$1.39	Micro	N	$2.29	Micro	$1.99
		N	$1.39	Stove	$2.49	N	Stove	N
		$1.69	$2.39	Stove	$1.99	N	Stove	$1.99
		$1.69	N	Stove	$2.49	$2.29	Stove	$1.99
		$1.69	$1.39	Stove	N	$1.49	Micro	$2.39
		$2.09	$1.89	Micro	$2.49	N	Stove	$2.39
		$1.29	$1.89	Stove	$2.49	$1.49	Stove	N
		$1.29	N	Micro	N	N	Stove	$2.39

In contrast, here are the actual values without formats and labels.

```
proc print data=sasuser.finchdes; format _numeric_; run;
```

Consumer Food Product Example

Obs	x1	x2	x3	x4	x5	x6	x7	x8	Block
1	2.09	1.89	1	1	1.99	1.49	2	N	1
2	N	N	2	1	N	1.49	2	1.99	1
3	2.09	1.39	2	1	2.49	2.29	2	N	1
4	1.69	1.39	1	1	N	N	2	N	1
5	1.69	1.89	2	1	2.49	N	1	2.39	1
6	1.29	1.39	2	1	1.99	N	1	1.99	1
7	1.69	N	2	1	2.49	2.29	1	2.39	1
8	N	2.39	2	1	1.99	1.49	2	2.39	1
9	1.29	1.89	1	1	N	2.29	2	1.99	1
10	N	N	1	2	2.49	2.29	1	N	1
11	2.09	N	2	2	1.99	N	1	2.39	1
12	N	2.39	1	2	1.99	1.49	1	1.99	1
13	1.29	1.39	1	2	1.99	2.29	2	N	1
14	1.69	1.89	1	2	N	N	1	N	1
15	1.29	2.39	2	2	N	2.29	1	2.39	1
16	1.69	N	1	2	2.49	1.49	2	1.99	1
17	2.09	2.39	1	2	N	1.49	2	2.39	1
18	N	2.39	2	2	2.49	N	1	N	1
19	2.09	N	1	1	2.49	N	1	1.99	2
20	1.69	2.39	1	1	1.99	2.29	1	N	2
21	N	2.39	1	1	N	2.29	2	2.39	2
22	1.69	N	1	1	1.99	2.29	2	2.39	2
23	2.09	2.39	2	1	N	1.49	1	1.99	2
24	1.29	2.39	1	1	2.49	N	2	1.99	2
25	2.09	1.89	2	1	N	N	2	N	2
26	1.29	N	2	1	1.99	1.49	1	N	2
27	N	1.39	1	1	2.49	1.49	1	2.39	2
28	N	1.89	2	2	1.99	2.29	1	1.99	2
29	2.09	1.39	1	2	N	2.29	1	1.99	2
30	N	1.39	2	2	2.49	N	2	N	2
31	1.69	2.39	2	2	1.99	N	2	1.99	2
32	1.69	N	2	2	2.49	2.29	2	1.99	2
33	1.69	1.39	2	2	N	1.49	1	2.39	2
34	2.09	1.89	1	2	2.49	N	2	2.39	2
35	1.29	1.89	2	2	2.49	1.49	2	N	2
36	1.29	N	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 five 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 unestimable 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 and availability cross effects. Collecting data is time consuming and expensive. It is good practice, particularly when there are cross effects, to make sure that the design will work with the most complicated model that we anticipate fitting. We saw, starting on page 166, an example of generating artificial data to test the design before collecting real data. Here, we will explore an alternative approach. Before we collect any data, we will convert the linear design to a choice design and use the **%ChoiceEff** macro to evaluate its efficiency for a multinomial logit model, with 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.FINCHDES 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 effects. Normally, they would be dropped.

```
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.finchdes, key=key, alt=brand, out=rolled,
         keep=x1 x2 x5 x6 x8)

proc print data=sasuser.finchdes(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	$2.09	$1.89	Micro	No	$1.99	$1.49	Stove	N	1
2	N	N	Stove	No	N	$1.49	Stove	$1.99	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	$2.09	.	.	$2.09	$1.89	$1.99	$1.49	N
	Extension	$1.89	Micro	No	$2.09	$1.89	$1.99	$1.49	N
	Regional	$1.99	.	.	$2.09	$1.89	$1.99	$1.49	N
	Private	$1.49	Stove	.	$2.09	$1.89	$1.99	$1.49	N
	National	N	.	.	$2.09	$1.89	$1.99	$1.49	N
	None	.	.	.	$2.09	$1.89	$1.99	$1.49	N
2	Client	N	.	.	N	N	N	$1.49	$1.99
	Extension	N	Stove	No	N	N	N	$1.49	$1.99
	Regional	N	.	.	N	N	N	$1.49	$1.99
	Private	$1.49	Stove	.	N	N	N	$1.49	$1.99
	National	$1.99	.	.	N	N	N	$1.49	$1.99
	None	.	.	.	N	N	N	$1.49	$1.99

Set 1, Alternative 1

Brand	=	'Client'	the brand for this alternative
Price	=	**x1** = $2.09	the price of this alternative
Micro	=	.	does not apply to this brand
Shelf	=	.	does not apply to this brand
x1	=	$2.09	the price of the client brand in this choice set
x2	=	$1.89	the price of the extension in this choice set
x5	=	$1.99	the price of the regional competitor in this choice set
x6	=	$1.49	the price of the private label in this choice set
x8	=	N	national competitor unavailable in this choice set

Set 1, Alternative 2

Brand	=	'Extension'	the brand for this alternative
Price	=	**x2** = $1.89	the price of this alternative
Micro	=	Micro	Microwave version
Shelf	=	No	Shelf Talker, No
x1	=	$2.09	the price of the client brand in this choice set
x2	=	$1.89	the price of the extension in this choice set
x5	=	$1.99	the price of the regional competitor in this choice set
x6	=	$1.49	the price of the private label in this choice set
x8	=	N	national competitor unavailable in this choice set

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.

We need to do a few more things to this design before we are ready to use it. Since we will be treating all of the price factors as a quantitative (not a **class** variable), we need to convert the missing prices to zero. We also need to convert the missings for when **Micro** and **Shelf** do not apply to 2 for 'Stove' and 1 for 'No'. We also need to assign formats. 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 alternative (**w** = 1).

```
data sasuser.choicedes(drop=i);
   set rolled;
   array x[6] price x1 -- x8;
   do i = 1 to 6; if nmiss(x[i]) then x[i] = 0; end;
   if nmiss(micro) then micro = 2;
   if nmiss(shelf) then shelf = 1;
   w = brand eq 'None' or price ne 0;
   format price dollar5.2 shelf yn. micro micro.;
   run;

proc print data=sasuser.choicedes(obs=12); by set; id set; run;
```

Here are the first two choice sets.

Consumer Food Product Example

Set	Brand	Price	Micro	Shelf	x1	x2	x5	x6	x8	w
1	Client	$2.09	Stove	No	$2.09	$1.89	$1.99	$1.49	$0.00	1
	Extension	$1.89	Micro	No	$2.09	$1.89	$1.99	$1.49	$0.00	1
	Regional	$1.99	Stove	No	$2.09	$1.89	$1.99	$1.49	$0.00	1
	Private	$1.49	Stove	No	$2.09	$1.89	$1.99	$1.49	$0.00	1
	National	$0.00	Stove	No	$2.09	$1.89	$1.99	$1.49	$0.00	0
	None	$0.00	Stove	No	$2.09	$1.89	$1.99	$1.49	$0.00	1
2	Client	$0.00	Stove	No	$0.00	$0.00	$0.00	$1.49	$1.99	0
	Extension	$0.00	Stove	No	$0.00	$0.00	$0.00	$1.49	$1.99	0
	Regional	$0.00	Stove	No	$0.00	$0.00	$0.00	$1.49	$1.99	0
	Private	$1.49	Stove	No	$0.00	$0.00	$0.00	$1.49	$1.99	1
	National	$1.99	Stove	No	$0.00	$0.00	$0.00	$1.49	$1.99	1
	None	$0.00	Stove	No	$0.00	$0.00	$0.00	$1.49	$1.99	1

Now our choice design is done except for the final coding for the analysis. We can now use the **%ChoicEff** macro to evaluate it for a choice model. 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. 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 cross effects. For now, let's concentrate on everything else.

```
%choiceff(data=sasuser.choicedes,
          model= ..., /* model specification skipped for now */
          nsets=36, nalts=6, weight=w,
          beta=zero, init=sasuser.choicedes(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.choicedes(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.choicedes,
          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 ') /
                lprefix=0 order=data,
          nsets=36, nalts=6, weight=w,
          beta=zero, init=sasuser.choicedes(keep=set),
          intiter=0);
```

The specification **class(brand / zero='None')** specifies the brand effects. This specification will create dummy 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'). Dummy 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 dummy 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 dummy 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 dummy 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 cross effects. The **separators=** option is specified with a second value of **' on '** to create cross effect labels like **'Client on Extension'**. 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 twice.

```
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 on two occasions **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	9.5192	1	3.08532
2	BrandExtension	Extension	9.5688	1	3.09335
3	BrandRegional	Regional	32.4993	1	5.70081
4	BrandPrivate	Private	11.9475	1	3.45651
5	BrandNational	National	35.8218	1	5.98513
6	BrandClientPrice	Client Price	2.3237	1	1.52437
7	BrandExtensionPrice	Extension Price	1.5338	1	1.23845
8	BrandRegionalPrice	Regional Price	4.5450	1	2.13191
9	BrandPrivatePrice	Private Price	1.7607	1	1.32693
10	BrandNationalPrice	National Price	6.4173	1	2.53323
11	ShelfTalker	Shelf Talker	0.9420	1	0.97055
12	MicroMicro	Micro	0.4983	1	0.70587
13	x1BrandClient	Client Brand on Client	.	0	.
14	x1BrandExtension	Client Brand on Extension	0.5613	1	0.74923
15	x1BrandRegional	Client Brand on Regional	0.5515	1	0.74265
16	x1BrandPrivate	Client Brand on Private	0.5311	1	0.72879
17	x1BrandNational	Client Brand on National	0.6016	1	0.77565
18	x2BrandClient	Client Line Extension on Client	0.3960	1	0.62926
19	x2BrandExtension	Client Line Extension on Extension	.	0	.
20	x2BrandRegional	Client Line Extension on Regional	0.4368	1	0.66095
21	x2BrandPrivate	Client Line Extension on Private	0.4050	1	0.63643
22	x2BrandNational	Client Line Extension on National	0.3484	1	0.59029
23	x5BrandClient	Regional Brand on Client	0.2629	1	0.51270
24	x5BrandExtension	Regional Brand on Extension	0.2971	1	0.54504
25	x5BrandRegional	Regional Brand on Regional	.	0	.
26	x5BrandPrivate	Regional Brand on Private	0.4222	1	0.64978
27	x5BrandNational	Regional Brand on National	0.3078	1	0.55477
28	x6BrandClient	Private Label on Client	0.3090	1	0.55590
29	x6BrandExtension	Private Label on Extension	0.3491	1	0.59086
30	x6BrandRegional	Private Label on Regional	0.4039	1	0.63554
31	x6BrandPrivate	Private Label on Private	.	0	.
32	x6BrandNational	Private Label on National	0.4439	1	0.66624
33	x8BrandClient	National Competitor on Client	0.2766	1	0.52595
34	x8BrandExtension	National Competitor on Extension	0.2918	1	0.54015
35	x8BrandRegional	National Competitor on Regional	0.3185	1	0.56435
36	x8BrandPrivate	National Competitor on Private	0.4331	1	0.65809
37	x8BrandNational	National Competitor on National	.	0	.
				==	
				32	

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 are single *df* effects for the shelf talker and the microwave option. Next we see five sets of cross effects, each consisting of four effects of a brand on another brand, plus one more zero *df* cross effect of a bran 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 the alternative-specific price effects. These results look fine. Everything that should be estimable is, and everything that should not 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	$2.09	$1.89	Micro	$1.99	$1.49	Stove	N
		N	N	Stove	N	$1.49	Stove	$1.99
		$2.09	$1.39	Stove	$2.49	$2.29	Stove	N
		$1.69	$1.39	Micro	N	N	Stove	N
		$1.69	$1.89	Stove	$2.49	N	Micro	$2.39

The coded design that the **%ChoicEff** 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=21) label;
   var Brand Price Shelf Micro;
   where w;
   run;
```

Consumer Food Product Example

Obs	Brand	Price	Shelf	Micro
1	Client	$2.09	No	Stove
2	Extension	$1.89	No	Micro
3	Regional	$1.99	No	Stove
4	Private	$1.49	No	Stove
6	None	$0.00	No	Stove
10	Private	$1.49	No	Stove
11	National	$1.99	No	Stove
12	None	$0.00	No	Stove
13	Client	$2.09	No	Stove
14	Extension	$1.39	No	Stove
15	Regional	$2.49	No	Stove
16	Private	$2.29	No	Stove
18	None	$0.00	No	Stove
19	Client	$1.69	No	Stove
20	Extension	$1.39	No	Micro
24	None	$0.00	No	Stove
25	Client	$1.69	No	Stove
26	Extension	$1.89	No	Stove
27	Regional	$2.49	No	Stove
29	National	$2.39	No	Stove
30	None	$0.00	No	Stove

Unlike all previous examples, the number of alternatives is not the same in all of the choice sets. The first choice set consists of five alternatives including 'None'. The national competitor is not available in this choice set. The second choice set consists of three alternatives including 'None'. The client brand, extension, and regional competitors are not available in this choice set. The third choice set consists of five alternatives including 'None', and so on.

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 print data=tmp_cand(obs=5) label;
   id Brand Price;
   var BrandClientPrice -- BrandNationalPrice;
   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
Private	0	0	0	1	0
None	0	0	0	0	0

Consumer Food Product Example

Brand	Price	Client Price	Extension Price	Regional Price	Private Price	National Price
Client	$2.09	2.09	0.00	0.00	0.00	0
Extension	$1.89	0.00	1.89	0.00	0.00	0
Regional	$1.99	0.00	0.00	1.99	0.00	0
Private	$1.49	0.00	0.00	0.00	1.49	0
None	$0.00	0.00	0.00	0.00	0.00	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 national competitor. Note that **Brand** **Price** are just an 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	$2.09	No	Stove	0	0
Extension	$1.89	No	Micro	0	1
Regional	$1.99	No	Stove	0	0
Private	$1.49	No	Stove	0	0
None	$0.00	No	Stove	0	0

The following code prints the cross effects along with **Brand** and **Price** for the first choice set.

```
proc print data=tmp_cand(obs=5) label;
   id Brand Price;
   var x1Brand:;
   where w;
   run;
proc print data=tmp_cand(obs=5) label;
   id Brand Price;
   var x2Brand:;
   where w;
   run;
proc print data=tmp_cand(obs=5) label;
   id Brand Price;
   var x5Brand:;
   where w;
   run;
proc print data=tmp_cand(obs=5) label;
   id Brand Price;
   var x6Brand:;
   where w;
   run;
proc print data=tmp_cand(obs=5) label;
   id Brand Price;
   var x8Brand:;
   where w;
   run;
```

The cross effects are printed in panels. This first panel shows the terms that capture the effect of the client brand being available at $2.09 on the utility of the other brands. The last panel shows that the national competitor, which is unavailable, has no effect on any other brand's utility in this choice set.

Consumer Food Product Example

Brand	Price	Client Brand on Client	Client Brand on Extension	Client Brand on Regional	Client Brand on Private	Client Brand on National
Client	$2.09	2.09	0.00	0.00	0.00	0
Extension	$1.89	0.00	2.09	0.00	0.00	0
Regional	$1.99	0.00	0.00	2.09	0.00	0
Private	$1.49	0.00	0.00	0.00	2.09	0
None	$0.00	0.00	0.00	0.00	0.00	0

Consumer Food Product Example

Brand	Price	Client Line Extension on Client	Client Line Extension on Extension	Client Line Extension on Regional	Client Line Extension on Private	Client Line Extension on National
Client	$2.09	1.89	0.00	0.00	0.00	0
Extension	$1.89	0.00	1.89	0.00	0.00	0
Regional	$1.99	0.00	0.00	1.89	0.00	0
Private	$1.49	0.00	0.00	0.00	1.89	0
None	$0.00	0.00	0.00	0.00	0.00	0

Consumer Food Product Example

Brand	Price	Regional Brand on Client	Regional Brand on Extension	Regional Brand on Regional	Regional Brand on Private	Regional Brand on National
Client	$2.09	1.99	0.00	0.00	0.00	0
Extension	$1.89	0.00	1.99	0.00	0.00	0
Regional	$1.99	0.00	0.00	1.99	0.00	0
Private	$1.49	0.00	0.00	0.00	1.99	0
None	$0.00	0.00	0.00	0.00	0.00	0

Consumer Food Product Example

Brand	Price	Private Label on Client	Private Label on Extension	Private Label on Regional	Private Label on Private	Private Label on National
Client	$2.09	1.49	0.00	0.00	0.00	0
Extension	$1.89	0.00	1.49	0.00	0.00	0
Regional	$1.99	0.00	0.00	1.49	0.00	0
Private	$1.49	0.00	0.00	0.00	1.49	0
None	$0.00	0.00	0.00	0.00	0.00	0

Consumer Food Product Example

Brand	Price	National Competitor on Client	National Competitor on Extension	National Competitor on Regional	National Competitor on Private	National Competitor on National
Client	$2.09	0	0	0	0	0
Extension	$1.89	0	0	0	0	0
Regional	$1.99	0	0	0	0	0
Private	$1.49	0	0	0	0	0
None	$0.00	0	0	0	0	0

A column like **'Private Label on Client'** in the second last panel, for example captured the effect of the private label brand being available at $1.49 on the utility of the client brand. In the previous pane, **'Regional Brand on Extension'** captures the effect of the regional brand being available at $1.99 on the utility of the extension.

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 we know how to specify the model to get all the right codings. 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. 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 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.finchdes point=j;
         do brand = 1 to &m; u[brand] = brands[brand] + 2 * normal(7); 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 252212542412222122622122115222212221   2 252222521452221422122122212222212226
  3 242221122412222422122122212222112221   4 242222122312222122122112212222252211
  5 241222122452222122124152232522212221   6 251222122452222122122122212222212521
  .
  .
  .
297 242222122412221122122522242422252221 298 251122141315222522122121212222112221
299 352222122412222112112511212222212221 300 242222122452222252251212212222112211
;
```

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=** alternative and expects a constant number of alternatives in each choice set.

```
%mktmerge(design=sasuser.choicedes, 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.

Subj	Set	Brand	Price	Micro	Shelf	x1	x2	x5	x6	x8	w	c
1	1	Client	$2.09	Stove	No	$2.09	$1.89	$1.99	$1.49	$0.00	1	2
		Extension	$1.89	Micro	No	$2.09	$1.89	$1.99	$1.49	$0.00	1	1
		Regional	$1.99	Stove	No	$2.09	$1.89	$1.99	$1.49	$0.00	1	2
		Private	$1.49	Stove	No	$2.09	$1.89	$1.99	$1.49	$0.00	1	2
		National	$0.00	Stove	No	$2.09	$1.89	$1.99	$1.49	$0.00	0	2
		None	$0.00	Stove	No	$2.09	$1.89	$1.99	$1.49	$0.00	1	2
1	2	Client	$0.00	Stove	No	$0.00	$0.00	$0.00	$1.49	$1.99	0	2
		Extension	$0.00	Stove	No	$0.00	$0.00	$0.00	$1.49	$1.99	0	2
		Regional	$0.00	Stove	No	$0.00	$0.00	$0.00	$1.49	$1.99	0	2
		Private	$1.49	Stove	No	$0.00	$0.00	$0.00	$1.49	$1.99	1	2
		National	$1.99	Stove	No	$0.00	$0.00	$0.00	$1.49	$1.99	1	1
		None	$0.00	Stove	No	$0.00	$0.00	$0.00	$1.49	$1.99	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 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 292. 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	c	_FREQ_
1	1	Client	$2.09	No	Stove	$2.09	$1.89	$1.99	$1.49	$0.00	1	42
2	1	Client	$2.09	No	Stove	$2.09	$1.89	$1.99	$1.49	$0.00	2	258
3	1	Extension	$1.89	No	Micro	$2.09	$1.89	$1.99	$1.49	$0.00	1	255
4	1	Extension	$1.89	No	Micro	$2.09	$1.89	$1.99	$1.49	$0.00	2	45
5	1	None	$0.00	No	Stove	$2.09	$1.89	$1.99	$1.49	$0.00	2	300
6	1	Private	$1.49	No	Stove	$2.09	$1.89	$1.99	$1.49	$0.00	1	1
7	1	Private	$1.49	No	Stove	$2.09	$1.89	$1.99	$1.49	$0.00	2	299
8	1	Regional	$1.99	No	Stove	$2.09	$1.89	$1.99	$1.49	$0.00	1	2
9	1	Regional	$1.99	No	Stove	$2.09	$1.89	$1.99	$1.49	$0.00	2	298

In the first choice set, the client brand was chosen (**c** = 1) a total of **_freq_** = 42 times and not chosen (**c** = 2) a total of **_freq_** = 258 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 217. 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 ') /
         lprefix=0;
   output out=coded(drop=_type_ _name_ intercept);
   id set c _freq_;
   label x1 = 'CE, Client'
         x2 = 'CE, Extension'
         x5 = 'CE, Regional'
         x6 = 'CE, Private'
         x8 = 'CE, National'
         shelf = 'Shelf Talker'
         micro = 'Microwave';
   run;
```

Note that like we saw in the **%ChoicEff** macro, PROC TRANSREG produces the following warning twice.

```
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 on two occasions **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.

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 79 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

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	1500	300	1200
4	4	900	300	600
5	5	1500	300	1200
6	6	1500	300	1200
7	7	1500	300	1200
8	8	1500	300	1200
9	9	1500	300	1200
10	10	900	300	600
11	11	1200	300	900
12	12	1500	300	1200
13	13	1500	300	1200
14	14	900	300	600
15	15	1500	300	1200
16	16	1500	300	1200
17	17	1500	300	1200
18	18	900	300	600
19	19	1200	300	900
20	20	1500	300	1200
21	21	1200	300	900
22	22	1500	300	1200
23	23	1500	300	1200
24	24	1500	300	1200
25	25	900	300	600
26	26	1200	300	900
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	1500	300	1200
34	34	1500	300	1200
35	35	1500	300	1200
36	36	900	300	600
Total		48000	10800	37200

Convergence Status

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

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	154978.05	135268.63
AIC	154978.05	135332.63
SBC	154978.05	135565.83

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	19709.4180	32	<.0001
Score	22091.0481	32	<.0001
Wald	7080.2597	32	<.0001

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Client	1	4.24512	0.51680	67.4740	<.0001
Extension	1	5.48777	0.54501	101.3878	<.0001
National	1	2.69212	0.74517	13.0519	0.0003
Private	1	2.93103	0.57137	26.3150	<.0001
Regional	1	2.58486	1.17748	4.8191	0.0281
Client Price	1	0.22857	0.32495	0.4948	0.4818
Extension Price	1	0.28959	0.27805	1.0848	0.2976
National Price	1	-0.73909	0.35790	4.2646	0.0389
Private Price	1	-0.50607	0.26142	3.7476	0.0529
Regional Price	1	-1.12619	0.46588	5.8435	0.0156
Shelf Talker	1	0.63195	0.06938	82.9781	<.0001
Micro	1	0.67436	0.06136	120.7997	<.0001
CE, Client on Client	0	0	.	.	.
CE, Client on Extension	1	0.97241	0.31232	9.6938	0.0018
CE, Client on National	1	0.78460	0.30882	6.4546	0.0111
CE, Client on Private	1	0.81216	0.30519	7.0820	0.0078
CE, Client on Regional	1	0.62573	0.34844	3.2250	0.0725
CE, Extension on Client	1	1.13135	0.26635	18.0422	<.0001
CE, Extension on Extension	0	0	.	.	.
CE, Extension on National	1	0.89205	0.26587	11.2574	0.0008
CE, Extension on Private	1	0.75865	0.26804	8.0110	0.0046
CE, Extension on Regional	1	0.81508	0.29154	7.8160	0.0052
CE, Regional on Client	1	-0.14977	0.21001	0.5086	0.4757
CE, Regional on Extension	1	-0.09684	0.21131	0.2100	0.6467
CE, Regional on National	1	-0.19080	0.21256	0.8057	0.3694
CE, Regional on Private	1	-0.21540	0.21625	0.9921	0.3192
CE, Regional on Regional	0	0	.	.	.

CE, Private on Client	1	0.39355	0.19335	4.1429	0.0418
CE, Private on Extension	1	0.38933	0.19579	3.9543	0.0468
CE, Private on National	1	0.35647	0.19772	3.2504	0.0714
CE, Private on Private	0	0	.	.	.
CE, Private on Regional	1	0.23482	0.22396	1.0994	0.2944
CE, National on Client	1	-0.37645	0.23758	2.5107	0.1131
CE, National on Extension	1	-0.33986	0.23888	2.0241	0.1548
CE, National on National	0	0	.	.	.
CE, National on Private	1	-0.32288	0.23670	1.8608	0.1725
CE, National on Regional	1	-0.27579	0.26135	1.1136	0.2913

Since the number of alternatives is not constant within each choice set, the summary table has nonconstant numbers of alternatives and numbers 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 first choice set has 1500 alternatives, 5 available times 300 subjects; whereas the fifth choice set has 900 alternatives, 3 available times 300 subjects.

The most to least preferred brands are: client line extension, client brand, private label, national brand, and regional competitor, and finally the none alternative (with an implicit part-worth utility of zero). The price effects are mostly negative, and the positive effects are nonsignificant. Both the shelf-talker and the microwaveable option have positive utility. The cross effects are mostly nonsignificant. The most significant cross effect is the effect of the extension on the original client brand.

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 respondent age and income (in thousands of dollars). The data from two subjects appear on one line.

```
data results;
   input Subj (choose1-choose&n) (1.) age income;
   datalines;
  1 252212542412222122622122115222212221 33  44
  2 252222521452221422122122212222212226 52  82
  3 242221122412222422122122212222112221 51 136
  4 242222122312222122122112212222252211 60 108
  .
  .
  .
299 352222122412222112112511212222212221 48  49
300 242222122452222522512112212222112211 38  51
;
```

Merging the data and design is no different from what we saw previously.

```
%mktmerge(design=sasuser.choicedes, 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	x1	x2	x5	x6	x8	w	c
1	1	33	44	Client	$2.09	Stove	No	$2.09	$1.89	$1.99	$1.49	$0.00	1	2
		33	44	Extension	$1.89	Micro	No	$2.09	$1.89	$1.99	$1.49	$0.00	1	1
		33	44	Regional	$1.99	Stove	No	$2.09	$1.89	$1.99	$1.49	$0.00	1	2
		33	44	Private	$1.49	Stove	No	$2.09	$1.89	$1.99	$1.49	$0.00	1	2
		33	44	National	$0.00	Stove	No	$2.09	$1.89	$1.99	$1.49	$0.00	0	2
		33	44	None	$0.00	Stove	No	$2.09	$1.89	$1.99	$1.49	$0.00	1	2
2	2	52	82	Client	$0.00	Stove	No	$0.00	$0.00	$0.00	$1.49	$1.99	0	2
		52	82	Extension	$0.00	Stove	No	$0.00	$0.00	$0.00	$1.49	$1.99	0	2
		52	82	Regional	$0.00	Stove	No	$0.00	$0.00	$0.00	$1.49	$1.99	0	2
		52	82	Private	$1.49	Stove	No	$0.00	$0.00	$0.00	$1.49	$1.99	1	2
		52	82	National	$1.99	Stove	No	$0.00	$0.00	$0.00	$1.49	$1.99	1	1
		52	82	None	$0.00	Stove	No	$0.00	$0.00	$0.00	$1.49	$1.99	1	2
3	3	51	136	Client	$2.09	Stove	No	$2.09	$1.39	$2.49	$2.29	$0.00	1	2
		51	136	Extension	$1.39	Stove	No	$2.09	$1.39	$2.49	$2.29	$0.00	1	1
		51	136	Regional	$2.49	Stove	No	$2.09	$1.39	$2.49	$2.29	$0.00	1	2
		51	136	Private	$2.29	Stove	No	$2.09	$1.39	$2.49	$2.29	$0.00	1	2
		51	136	National	$0.00	Stove	No	$2.09	$1.39	$2.49	$2.29	$0.00	0	2
		51	136	None	$0.00	Stove	No	$2.09	$1.39	$2.49	$2.29	$0.00	1	2
300	36	38	51	Client	$1.29	Stove	No	$1.29	$0.00	$0.00	$0.00	$2.39	1	1
		38	51	Extension	$0.00	Micro	Talker	$1.29	$0.00	$0.00	$0.00	$2.39	0	2
		38	51	Regional	$0.00	Stove	No	$1.29	$0.00	$0.00	$0.00	$2.39	0	2
		38	51	Private	$0.00	Stove	No	$1.29	$0.00	$0.00	$0.00	$2.39	0	2
		38	51	National	$2.39	Stove	No	$1.29	$0.00	$0.00	$0.00	$2.39	1	2
		38	51	None	$0.00	Stove	No	$1.29	$0.00	$0.00	$0.00	$2.39	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')
         identity(x1 x2 x5 x6 x8) *
            class(brand / zero='None' separators=' ' ' on ') /
         lprefix=0 order=data;
```

```
   output out=coded(drop=_type_ _name_ intercept);
   where w; /* exclude unavailable, w = 0 */
   id subj set c;
   label x1 = 'CE, Client'
         x2 = 'CE, Extension'
         x5 = 'CE, Regional'
         x6 = 'CE, Private'
         x8 = 'CE, National'
         shelf = 'Shelf Talker'
         micro = 'Microwave';
   run;
```

The **Age** and **Income** variables are incorporated into the analysis by interacting them with **Brand**. Demographic variables must be interacted with 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=5) label;
   id brand price;
   var BrandClient -- BrandPrivate Shelf Micro c;
   run;
proc print data=coded(obs=5 drop=Age) label;
   id brand price;
   var Age:;
   run;
proc print data=coded(obs=5 drop=Income) label;
   id brand price;
   var Income:;
   run;
proc print data=coded(obs=5) label;
   id brand price;
   var BrandClientPrice -- BrandPrivatePrice;
   format BrandClientPrice -- BrandPrivatePrice best4.;
   run;
proc print data=coded(obs=5 drop=x1) label;
   id brand price; var x1:; format x1: best4.;
   run;
proc print data=coded(obs=5 drop=x2) label;
   id brand price; var x2:; format x2: best4.;
   run;
proc print data=coded(obs=5 drop=x5) label;
   id brand price; var x5:; format x5: best4.;
   run;
proc print data=coded(obs=5 drop=x6) label;
   id brand price; var x6:; format x6: best4.;
   run;
proc print data=coded(obs=5 drop=x8) label;
   id brand price; var x8:; format x8: 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	$2.09	1	0	0	0	No	Stove	2
Extension	$1.89	0	1	0	0	No	Micro	1
Regional	$1.99	0	0	1	0	No	Stove	2
Private	$1.49	0	0	0	1	No	Stove	2
None	$0.00	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	$2.09	33	0	0	0	0
Extension	$1.89	0	33	0	0	0
Regional	$1.99	0	0	33	0	0
Private	$1.49	0	0	0	33	0
None	$0.00	0	0	0	0	0

Here are the income by brand effects.

Consumer Food Product Example

Brand	Price	Income, Client	Income, Extension	Income, Regional	Income, Private	Income, National
Client	$2.09	44	0	0	0	0
Extension	$1.89	0	44	0	0	0
Regional	$1.99	0	0	44	0	0
Private	$1.49	0	0	0	44	0
None	$0.00	0	0	0	0	0

Here are the alternative-specific price effects.

Consumer Food Product Example

Brand	Price	Client Price	Extension Price	Regional Price	Private Price
Client	$2.09	2.09	0	0	0
Extension	$1.89	0	1.89	0	0
Regional	$1.99	0	0	1.99	0
Private	$1.49	0	0	0	1.49
None	$0.00	0	0	0	0

Here are the client cross effects.

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	$2.09	2.09	0	0	0	0
Extension	$1.89	0	2.09	0	0	0
Regional	$1.99	0	0	2.09	0	0
Private	$1.49	0	0	0	2.09	0
None	$0.00	0	0	0	0	0

Here are the extension cross effects.

Consumer Food Product Example

Brand	Price	CE, Extension on Client	CE, Extension on Extension	CE, Extension on Regional	CE, Extension on Private	CE, Extension on National
Client	$2.09	1.89	0	0	0	0
Extension	$1.89	0	1.89	0	0	0
Regional	$1.99	0	0	1.89	0	0
Private	$1.49	0	0	0	1.89	0
None	$0.00	0	0	0	0	0

Here are the regional competitor cross effects.

Consumer Food Product Example

Brand	Price	CE, Regional on Client	CE, Regional on Extension	CE, Regional on Regional	CE, Regional on Private	CE, Regional on National
Client	$2.09	1.99	0	0	0	0
Extension	$1.89	0	1.99	0	0	0
Regional	$1.99	0	0	1.99	0	0
Private	$1.49	0	0	0	1.99	0
None	$0.00	0	0	0	0	0

Here are the private label cross effects.

Consumer Food Product Example

Brand	Price	CE, Private on Client	CE, Private on Extension	CE, Private on Regional	CE, Private on Private	CE, Private on National
Client	$2.09	1.49	0	0	0	0
Extension	$1.89	0	1.49	0	0	0
Regional	$1.99	0	0	1.49	0	0
Private	$1.49	0	0	0	1.49	0
None	$0.00	0	0	0	0	0

Here are the national competitor cross effects.

Consumer Food Product Example

Brand	Price	CE, National on Client	CE, National on Extension	CE, National on Regional	CE, National on Private	CE, National on National
Client	$2.09	0	0	0	0	0
Extension	$1.89	0	0	0	0	0
Regional	$1.99	0	0	0	0	0
Private	$1.49	0	0	0	0	0
None	$0.00	0	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

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	1200	4	1	3
3	7200	5	1	4

Convergence Status

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

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	31776.351	12053.607
AIC	31776.351	12137.607
SBC	31776.351	12443.674

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	19722.7440	42	<.0001
Score	22093.7119	42	<.0001
Wald	7064.8884	42	<.0001

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Client	1	5.15121	0.64435	63.9112	<.0001
Extension	1	6.34979	0.67179	89.3407	<.0001
Regional	1	3.68135	1.26766	8.4335	0.0037
Private	1	3.72866	0.69082	29.1321	<.0001
National	1	3.58319	0.84306	18.0643	<.0001
Age, Client	1	-0.03218	0.01306	6.0733	0.0137
Age, Extension	1	-0.03271	0.01338	5.9778	0.0145
Age, Regional	1	-0.04536	0.01657	7.4964	0.0062
Age, Private	1	-0.03551	0.01317	7.2717	0.0070
Age, National	1	-0.02887	0.01346	4.6048	0.0319
Income, Client	1	0.00773	0.00548	1.9941	0.1579
Income, Extension	1	0.00876	0.00560	2.4448	0.1179
Income, Regional	1	0.01336	0.00676	3.9103	0.0480
Income, Private	1	0.01154	0.00551	4.3830	0.0363
Income, National	1	0.00579	0.00564	1.0539	0.3046
Client Price	1	0.23736	0.32667	0.5280	0.4675
Extension Price	1	0.29648	0.27957	1.1247	0.2889
Regional Price	1	-1.12989	0.46647	5.8670	0.0154
Private Price	1	-0.50333	0.26187	3.6944	0.0546
National Price	1	-0.74511	0.35881	4.3124	0.0378
Shelf Talker	1	0.63199	0.06938	82.9766	<.0001
Micro	1	0.67453	0.06136	120.8327	<.0001
CE, Client on Client	0	0	.	.	.
CE, Client on Extension	1	0.98125	0.31408	9.7608	0.0018
CE, Client on Regional	1	0.63475	0.35006	3.2880	0.0698
CE, Client on Private	1	0.82032	0.30695	7.1425	0.0075
CE, Client on National	1	0.79361	0.31055	6.5306	0.0106
CE, Extension on Client	1	1.13830	0.26794	18.0478	<.0001
CE, Extension on Extension	0	0	.	.	.
CE, Extension on Regional	1	0.82281	0.29302	7.8852	0.0050
CE, Extension on Private	1	0.76600	0.26958	8.0737	0.0045
CE, Extension on National	1	0.89865	0.26742	11.2925	0.0008
CE, Regional on Client	1	-0.15260	0.21106	0.5228	0.4697
CE, Regional on Extension	1	-0.09966	0.21236	0.2202	0.6389
CE, Regional on Regional	0	0	.	.	.
CE, Regional on Private	1	-0.21835	0.21734	1.0093	0.3151
CE, Regional on National	1	-0.19370	0.21362	0.8221	0.3646
CE, Private on Client	1	0.39602	0.19388	4.1723	0.0411
CE, Private on Extension	1	0.39183	0.19632	3.9837	0.0459
CE, Private on Regional	1	0.23759	0.22445	1.1205	0.2898
CE, Private on Private	0	0	.	.	.
CE, Private on National	1	0.35882	0.19823	3.2766	0.0703

```
CE, National on Client       1    -0.38253     0.23881      2.5657      0.1092
CE, National on Extension    1    -0.34594     0.24010      2.0760      0.1496
CE, National on Regional     1    -0.28219     0.26249      1.1557      0.2824
CE, National on Private      1    -0.32953     0.23791      1.9185      0.1660
CE, National on National     0           0           .           .           .
```

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.

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 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 report are documented starting on page 287.) 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 Design Sizes		Violations	Cannot Be 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	Taguchi, 1987
36	2 ** 4 3 ** 13	Taguchi, 1987
36	2 ** 2 3 ** 12 6 ** 1	Wang and Wu, 1991
36	3 ** 13 4 ** 1	Dey, 1985
36	3 ** 12 12 ** 1	Wang and Wu, 1991
54	2 ** 1 3 ** 25	Taguchi, 1987
54	3 ** 24 6 ** 1	Hedayat, Sloane, and Stufken, 1999
54	3 ** 18 18 ** 1	Hedayat, Sloane, and Stufken, 1999

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=7654321)
```

The macro finds a 100% D-efficient design.

Allocation of Prescription Drugs

Algorithm Search History

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

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.allocdes; 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	$100	$75	$100	$100	$75	$100	$50	$50	$75	$100
	$50	$100	$75	$100	$100	$100	$100	$75	$75	$75
	$75	$50	$50	$100	$50	$100	$75	$100	$75	$50
	$75	$75	$100	$50	$75	$75	$100	$75	$50	$75
	$75	$100	$75	$75	$100	$50	$50	$50	$100	$100
	$50	$50	$50	$50	$50	$75	$50	$50	$50	$100
	$100	$50	$50	$75	$50	$50	$100	$75	$100	$75
	$100	$100	$75	$50	$100	$75	$75	$100	$50	$50
	$50	$75	$100	$75	$75	$50	$75	$100	$100	$50
2	$75	$100	$50	$75	$75	$75	$75	$75	$75	$100
	$100	$50	$100	$75	$100	$75	$50	$100	$75	$75
	$100	$100	$50	$50	$75	$100	$100	$50	$100	$50
	$100	$75	$75	$100	$50	$50	$75	$75	$50	$100
	$50	$100	$50	$100	$75	$50	$50	$100	$50	$75
	$75	$75	$75	$50	$50	$100	$50	$100	$100	$75
	$50	$75	$75	$75	$50	$75	$100	$50	$75	$50
	$50	$50	$100	$50	$100	$100	$75	$75	$100	$100
	$75	$50	$100	$100	$100	$50	$100	$50	$50	$50
3	$100	$75	$50	$100	$100	$75	$100	$100	$100	$100
	$75	$100	$100	$75	$50	$100	$100	$100	$50	$100
	$100	$50	$75	$75	$75	$100	$75	$50	$50	$75
	$100	$100	$100	$50	$50	$50	$50	$75	$75	$50
	$75	$75	$50	$50	$100	$50	$75	$50	$75	$75
	$75	$50	$75	$100	$75	$75	$50	$75	$100	$50
	$50	$100	$100	$100	$50	$75	$75	$50	$100	$75
	$50	$75	$50	$75	$100	$100	$50	$75	$50	$50
	$50	$50	$75	$50	$75	$50	$100	$100	$75	$100

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

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   2 1  1 1 0 2 0 3 0 1 1 1
2   2 2  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 ($(10 + 1) \times 9 \times 3 \times 100 = 29700$ 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 × 9 sets × 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 in the SAS log 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.allocdes, key=key, alt=brand, out=rolled)

proc print data=rolled(obs=11); format price dollar4.; run;
```

Allocation of Prescription Drugs

Obs	Set	Brand	Price
1	1	Brand 1	$100
2	1	Brand 2	$75
3	1	Brand 3	$100
4	1	Brand 4	$100
5	1	Brand 5	$75
6	1	Brand 6	$100
7	1	Brand 7	$50
8	1	Brand 8	$50
9	1	Brand 9	$75
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	$100	103
	Brand 2	$75	58
	Brand 3	$100	318
	Brand 4	$100	99
	Brand 5	$75	54
	Brand 6	$100	83
	Brand 7	$50	71
	Brand 8	$50	58
	Brand 9	$75	100
	Brand 10	$100	56
---			-----
1			1000
2		.	10
	Brand 1	$50	73
	Brand 2	$100	76
	Brand 3	$75	342
	Brand 4	$100	55
	Brand 5	$100	50
	Brand 6	$100	77
	Brand 7	$100	95
	Brand 8	$75	71
	Brand 9	$75	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	$100	103	1
4	1	Brand 1	$100	897	2
5	1	Brand 2	$75	58	1
6	1	Brand 2	$75	942	2
7	1	Brand 3	$100	318	1
8	1	Brand 3	$100	682	2
9	1	Brand 4	$100	99	1
10	1	Brand 4	$100	901	2
11	1	Brand 5	$75	54	1
12	1	Brand 5	$75	946	2
13	1	Brand 6	$100	83	1
14	1	Brand 6	$100	917	2
15	1	Brand 7	$50	71	1
16	1	Brand 7	$50	929	2
17	1	Brand 8	$50	58	1
18	1	Brand 8	$50	942	2
19	1	Brand 9	$75	100	1
20	1	Brand 9	$75	900	2
21	1	Brand 10	$100	56	1
22	1	Brand 10	$50	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 unchosen. 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. Dummy 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 79 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

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.

Allocation of Prescription Drugs

The PHREG Procedure

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	502505.13	489061.18
AIC	502505.13	489085.18
SBC	502505.13	489183.62

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	13443.9511	12	<.0001
Score	18342.3475	12	<.0001
Wald	14088.6926	12	<.0001

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Brand 1	1	2.09201	0.06766	955.9229	<.0001
Brand 2	1	2.08414	0.06769	947.9356	<.0001
Brand 3	1	3.53501	0.06484	2972.3245	<.0001
Brand 4	1	2.09005	0.06767	953.9288	<.0001
Brand 5	1	2.07819	0.06771	941.9267	<.0001
Brand 6	1	2.02826	0.06790	892.2654	<.0001
Brand 7	1	2.06215	0.06777	925.8259	<.0001
Brand 8	1	2.07868	0.06771	942.4280	<.0001
Brand 9	1	2.11000	0.06760	974.2866	<.0001
Brand 10	1	2.05658	0.06779	920.2659	<.0001
$50	1	0.02024	0.01627	1.5466	0.2136
$75	1	0.00665	0.01632	0.1660	0.6837
$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

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.

Allocation of Prescription Drugs

The PHREG Procedure

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	37816.553	36472.158
AIC	37816.553	36496.158
SBC	37816.553	36566.970

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	1344.3951	12	<.0001
Score	1834.2348	12	<.0001
Wald	1408.8693	12	<.0001

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Brand 1	1	2.09201	0.21397	95.5923	<.0001
Brand 2	1	2.08414	0.21406	94.7936	<.0001
Brand 3	1	3.53501	0.20504	297.2324	<.0001
Brand 4	1	2.09005	0.21399	95.3929	<.0001
Brand 5	1	2.07819	0.21413	94.1927	<.0001
Brand 6	1	2.02826	0.21472	89.2265	<.0001
Brand 7	1	2.06215	0.21432	92.5826	<.0001
Brand 8	1	2.07868	0.21412	94.2428	<.0001
Brand 9	1	2.11000	0.21377	97.4287	<.0001
Brand 10	1	2.05658	0.21438	92.0266	<.0001
$50	1	0.02024	0.05146	0.1547	0.6941
$75	1	0.00665	0.05160	0.0166	0.8975
$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. 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)

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=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 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 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 from the 'Preliminaries' section starting on page 76. Instead 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 β. This approach uses the **%ChoicEff** macro.

Generic Attributes, Alternative Swapping, Large Candidate Set

This part of the example illustrates using the **%ChoiceEff** 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

                    Generic Chair Attributes

          Saturated      = 11
          Full Factorial = 243

          Some Reasonable                      Cannot Be
             Design Sizes        Violations    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              Taguchi, 1987
     18            3 **  6  6 **  1      Taguchi, 1987
     27            3 ** 13              Fractional-factorial
     27            3 **  9  9 **  1      Fractional-factorial
     36    2 ** 11  3 ** 12              Taguchi, 1987
     36    2 **  4  3 ** 13              Taguchi, 1987
     36    2 **  2  3 ** 12  6 **  1     Wang and Wu, 1991
     36    2 **  1  3 **  8  6 **  2     Zhang, Lu, and Pang, 1999
     36            3 ** 13  4 **  1      Dey, 1985
     36            3 ** 12 12 **  1      Wang and Wu, 1991
     36            3 **  7  6 **  3      Finney, 1982
```

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 should give us 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;
```

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

*Later, we will see we could have chosen 18.

Next, we will search that candidate set for an efficient design for the model specification **class(x1-x5)** and the assumption $\beta = 0$. We will use the **%ChoiceEff** autocall macro to do this. (All of the autocall macros used in this report are documented starting on page 287.) 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 β. The **%ChoiceEff** macro uses a modified Federov algorithm (Federov, 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 $\beta = 0$. A vector of numbers like **beta=-1 0 -1 0 -1 0 -1 0 -1 0 -1 0** could be specified. 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 β. It is very important to check this list and make sure it is correct. In particular, when you are explicitly specifying the β 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 **%ChoiceEff** macro uses PROC IML and a modified Federov algorithm to iteratively improve the efficiency of the choice design given the specified candidates, model, and β. 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

Design   Iteration  D-Efficiency       D-Error
-----------------------------------------------
   2         0          0.792568      1.261721
             1          1.513406      0.660761
             2          1.732051      0.577350
             3          1.732051      0.577350

   .
   .
   .

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 efficiency and D-Error (D-Efficiency = 1 / D-Error).

```
Final Results:   Design     = 34
                 Efficiency = 1.1547005384
                 D-Error    = 0.8660254038
```

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 efficiency, **Efficiency** - the 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 β, **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 $\times$ 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

Generic Chair Attributes

Final Results: Design = 1
Efficiency = 1.1547005384
D-Error = 0.8660254038

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 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. We have frequently found 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 β 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.

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)

%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	25	0.33333	3	3	1	1	2
	2	0.33333	1	1	2	3	3
	15	0.33333	2	2	3	2	1
2	10	0.33333	2	1	1	2	1
	23	0.33333	3	2	2	1	2
	9	0.33333	1	3	3	3	3
3	24	0.33333	3	2	3	3	1
	11	0.33333	2	1	2	1	3
	7	0.33333	1	3	1	2	2

```
4       13     0.33333     2     2     1     1     3
         3     0.33333     1     1     3     2     2
        26     0.33333     3     3     2     3     1

5       20     0.33333     3     1     2     2     3
         6     0.33333     1     2     3     1     1
        16     0.33333     2     3     1     3     2

6        8     0.33333     1     3     2     1     1
        22     0.33333     3     2     1     2     3
        12     0.33333     2     1     3     3     2

7        5     0.33333     1     2     2     2     2
        18     0.33333     2     3     3     1     3
        19     0.33333     3     1     1     3     1

8        1     0.33333     1     1     1     1     1
        14     0.33333     2     2     2     3     2
        27     0.33333     3     3     3     2     3

9       17     0.33333     2     3     2     2     1
         4     0.33333     1     2     1     3     3
        21     0.33333     3     1     3     1     2
```

Notice that like before, the variances are constant, 2/3, and each candidate appears once. This is an optimal design in 9 choice sets.

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)

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.

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 $244th$ 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 **%ChoiceEff** 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
Efficiency = 1.0033975924
D-Error = 0.9966139121

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	1.13102	1	1.06350
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	18	0.25	291	1	1	2	3	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 $\beta = 0$. With other assumptions about β, 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 **%ChoiceEff** 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 numeric, 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 unestimable 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 **%ChoiceEff** 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 267). 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(x1-x15)

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 line, which we copied, pasted, and edited to make the KEY data set.

```
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	3	2	2	1	2
	2	3	2	1	1	2
	3	2	1	3	3	1
	4	1	1	1	1	1
100	1	3	3	2	2	3
	2	3	1	3	3	2
	3	1	3	3	3	1
	4	1	1	1	1	1
1000	1	3	2	2	2	2
	2	3	3	3	2	1
	3	1	2	3	2	1
	4	1	1	1	1	1
5000	1	1	2	2	3	3
	2	3	3	3	3	2
	3	2	2	1	3	3
	4	1	1	1	1	1
6561	1	3	3	1	3	2
	2	3	2	1	2	1
	3	1	1	1	1	1
	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	0.518092	1.930159
	1	0.800630	1.249017
	2	0.861910	1.160214
	3	0.861910	1.160214
.			
.			
.			

Design	Iteration	D-Efficiency	D-Error
8	0	0.523312	1.910907
	1	0.850993	1.175097
	2	0.878594	1.138182
	3	0.878594	1.138182
.			
.			
.			

Design	Iteration	D-Efficiency	D-Error
10	0	0.552471	1.810051
	1	0.833732	1.199427
	2	0.844183	1.184577
	3	0.844183	1.184577

Generic Chair Attributes

```
Final Results:   Design     = 8
                 Efficiency = 0.8785943904
                 D-Error    = 1.1381816352
```

Generic Chair Attributes

n	Variable Name	Label	Variance	DF	Standard Error
1	x11	x1 1	1.23879	1	1.11301
2	x12	x1 2	1.99174	1	1.41129
3	x21	x2 1	1.11908	1	1.05787
4	x22	x2 2	1.84621	1	1.35875
5	x31	x3 1	1.34469	1	1.15961
6	x32	x3 2	1.87653	1	1.36987
7	x41	x4 1	1.40455	1	1.18514
8	x42	x4 2	1.47021	1	1.21252
9	x51	x5 1	1.51281	1	1.22996
10	x52	x5 2	1.29878	1	1.13964
				==	
				10	

This design is less 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 report in the context of this problem. There are $3^{3\times 5} = 14,348,907$ 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 β vector. They can be used to produce smaller, more specialized, and better designs. However, if the β 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% 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)
```

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	91.2636	83.9694	97.8111	0.9747

Here are the results from the second step.

Try to Improve an Existing Design

Design Refinement History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
0	Initial	91.2636	91.2636	Ini
1	Start	81.8822		Pre,Mut,Ann
1	End	89.5862		
2	Start	82.3429		Pre,Mut,Ann
2	End	89.2088		

NOTE: Quitting the refinement step after 7.33 minutes and 2 designs.

The macro skips the normal first steps, algorithm search and design search, and goes straight into the design refinement search. No improvements were found, which is usually the case.

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. Stated differently, 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. Later 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 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

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	1	1	1	2	1	1	1	2	1	2	1	3	2	1	2	3	3	1	1	1	3	1	3	1
2	1	1	2	1	2	1	2	1	1	1	1	3	1	1	3	2	3	3	2	1	1	3	2	1
3	2	1	1	2	2	2	2	2	2	1	1	2	2	1	1	2	1	2	3	1	2	3	3	1
4	1	2	1	2	1	2	2	1	1	1	2	1	3	3	3	2	3	1	1	2	2	3	3	1
5	2	2	1	1	1	1	1	1	2	1	1	2	3	2	3	3	1	1	2	1	2	1	2	1
6	2	2	2	2	2	2	1	1	1	2	1	3	3	3	1	1	3	3	3	1	2	1	1	1
7	1	2	2	2	2	1	1	2	2	1	2	1	1	3	2	3	1	1	3	1	1	3	1	1
8	2	1	2	1	1	2	1	2	1	1	2	2	1	3	3	1	1	3	1	1	3	2	3	1
9	1	1	1	1	2	2	1	1	2	2	2	2	2	2	3	3	3	3	3	2	3	3	1	1
10	1	1	1	1	2	2	1	1	2	2	2	1	3	1	1	1	2	1	2	1	1	2	3	1

11	1	1	1	2	1	1	1	2	1	2	1	2	3	3	3	1	2	2	3	3	1	3	2	1
12	2	2	1	1	1	1	1	1	2	1	1	1	1	1	1	1	3	2	1	3	3	3	1	1
13	1	1	1	2	1	1	1	2	1	2	1	1	1	2	1	2	1	3	2	2	2	2	1	1
14	1	1	2	1	2	1	2	1	1	1	1	2	2	3	1	3	2	1	1	3	2	2	1	1
15	2	1	2	1	1	2	1	2	1	1	2	3	3	1	2	3	2	2	2	2	2	3	1	1
16	1	2	1	2	1	2	2	1	1	1	2	2	2	1	2	1	1	3	2	3	1	1	1	1
17	1	2	2	2	2	1	1	2	2	1	2	3	2	2	3	1	3	2	2	3	2	2	3	1
18	2	1	1	2	2	2	2	2	2	1	1	1	3	3	2	3	3	3	2	3	3	2	2	1
19	1	1	2	1	2	1	2	1	1	1	1	1	3	2	2	1	1	2	3	2	3	1	3	1
20	2	1	2	1	1	2	1	2	1	1	2	1	2	2	1	2	3	1	3	3	1	1	2	1
21	2	2	2	2	2	2	1	1	1	2	1	1	2	1	3	3	1	2	1	2	1	2	2	1
22	2	1	2	2	1	1	2	1	2	2	2	2	3	2	2	2	3	2	1	1	1	2	1	1
23	2	2	1	1	2	1	2	2	1	2	2	3	3	1	3	2	1	1	3	3	3	2	1	1
24	2	1	2	2	1	1	2	1	2	2	2	3	2	3	1	1	1	1	2	2	3	3	2	1
25	1	2	1	2	1	2	2	1	1	1	2	3	1	2	1	3	2	2	3	1	3	2	2	1
26	2	2	2	2	2	2	1	1	1	2	1	2	1	2	2	2	2	1	2	3	3	3	3	1
27	1	2	2	1	1	2	2	2	2	2	1	3	3	2	1	3	1	3	1	3	1	3	3	1
28	1	2	2	1	1	2	2	2	2	2	1	1	2	3	3	2	2	2	2	1	3	1	1	1
29	2	2	1	1	2	1	2	2	1	2	2	2	1	3	1	3	3	2	2	2	1	1	3	1
30	1	1	1	1	2	2	1	1	2	2	2	3	1	3	2	2	1	2	1	3	2	1	2	1
31	2	2	1	1	2	1	2	2	1	2	2	1	2	2	2	1	2	3	1	1	2	3	2	1
32	2	1	1	2	2	2	2	2	2	1	1	3	1	2	3	1	2	1	1	2	1	1	1	1
33	1	2	2	2	2	1	1	2	2	1	2	2	3	1	1	2	2	3	1	2	3	1	2	1
34	2	1	2	2	1	1	2	1	2	2	2	1	1	1	3	3	2	3	3	3	2	1	3	1
35	2	2	1	1	1	1	1	1	2	1	1	3	2	3	2	2	2	3	3	2	1	2	3	1
36	1	2	2	1	1	2	2	2	2	2	1	2	1	1	2	1	3	1	3	2	2	2	2	1
37	1	2	2	1	1	2	2	2	2	2	1	2	1	1	2	1	3	1	3	2	2	2	2	.
38	1	2	2	1	1	2	2	2	2	2	1	2	1	1	2	1	3	1	3	2	2	2	2	.
39	1	2	2	1	1	2	2	2	2	2	1	2	1	1	2	1	3	1	3	2	2	2	2	.
40	1	2	2	1	1	2	2	2	2	2	1	2	1	1	2	1	3	1	3	2	2	2	2	.

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	97.0381	97.0381	Ini
1	Start	97.0328		Pre,Mut,Ann
1	37 3	97.0612	97.0612	
1	37 7	97.0618	97.0618	
1	37 8	97.0810	97.0810	
1	37 9	97.0994	97.0994	
1	37 12	97.1028	97.1028	
1	37 16	97.1074	97.1074	
1	38 5	97.1116	97.1116	
1	38 9	97.1220	97.1220	
1	38 10	97.1224	97.1224	
1	38 12	97.1224	97.1224	

```
1     38  14       97.1225       97.1225
1     39   2       97.1460       97.1460
1     39   3       97.1569       97.1569
1     39   5       97.1629       97.1629
1     39   6       97.1815       97.1815
1     39   7       97.1878       97.1878
1     39   8       97.1899       97.1899
1     39   9       97.1965       97.1965
1     39  11       97.1986       97.1986
1     39  12       97.1986       97.1986
1     39  13       97.2002       97.2002
1     39  17       97.2002       97.2002
1     40  10       97.2002       97.2002
1     40  13       97.2002       97.2002
1     37   1       97.2023       97.2023
1     37   6       97.2043       97.2043
1        End       97.2033

2      Start       97.1737                 Pre,Mut,Ann
2        End       97.2038

3      Start       97.1949                 Pre,Mut,Ann
3     39   5       97.2043       97.2043
3     39  14       97.2043       97.2043
3     39  23       97.2043       97.2043
3     40  10       97.2043       97.2043
3     40  16       97.2043       97.2043
3     37   3       97.2043       97.2043
3        End       97.2033

4      Start       97.1918                 Pre,Mut,Ann
4        End       97.2033

5      Start       97.1825                 Pre,Mut,Ann
5        End       97.2033

6      Start       97.1940                 Pre,Mut,Ann
6     38   9       97.2043       97.2043
6     38  13       97.2043       97.2043
6     38  18       97.2043       97.2043
6     39  13       97.2049       97.2049
6     39  23       97.2049       97.2049
6     37  14       97.2049       97.2049
6     37  19       97.2049       97.2049
6     37  12       97.2049       97.2049
6     37  21       97.2049       97.2049
6     39  15       97.2049       97.2049
6        End       97.2033

7      Start       97.1923                 Pre,Mut,Ann
7     39   6       97.2049       97.2049
7     39  11       97.2049       97.2049
7     39  17       97.2049       97.2049
7     39  19       97.2049       97.2049
7     39  22       97.2049       97.2049
7     40  19       97.2054       97.2054
7        End       97.2023

8      Start       97.1839                 Pre,Mut,Ann
8        End       97.2023
```

```
 9      Start       97.1915              Pre,Mut,Ann
 9        End       97.2038

10      Start       97.1862              Pre,Mut,Ann
10        End       97.2038
```

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

Ob	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	x18	x19	x20	x21	x22	x23	f
1	1	1	1	2	1	1	1	2	1	2	1	3	2	1	2	3	3	1	1	1	3	1	3	1
2	1	1	2	1	2	1	2	1	1	1	1	3	1	1	3	2	3	3	2	1	1	3	2	1
3	2	1	1	2	2	2	2	2	2	1	1	2	2	1	1	2	1	2	3	1	2	3	3	1
4	1	2	1	2	1	2	2	1	1	1	2	1	3	3	3	2	3	1	1	2	2	3	3	1
5	2	2	1	1	1	1	1	1	2	1	1	2	3	2	3	3	1	1	2	1	2	1	2	1
6	2	2	2	2	2	2	1	1	1	2	1	3	3	3	1	1	3	3	3	1	2	1	1	1
7	1	2	2	2	2	1	1	2	2	1	2	1	1	3	2	3	1	1	3	1	1	3	1	1
8	2	1	2	1	1	2	1	2	1	1	2	2	1	3	3	1	1	3	1	1	3	2	3	1
9	1	1	1	1	2	2	1	1	2	2	2	2	2	2	3	3	3	3	3	2	3	3	1	1
10	1	1	1	1	2	2	1	1	2	2	2	1	3	1	1	1	2	1	2	1	1	2	3	1
11	1	1	1	2	1	1	1	2	1	2	1	2	3	3	3	1	2	2	3	3	1	3	2	1
12	2	2	1	1	1	1	1	1	2	1	1	1	1	1	1	1	3	2	1	3	3	3	1	1
13	1	1	1	2	1	1	1	2	1	2	1	1	1	2	1	2	1	3	2	2	2	2	1	1
14	1	1	2	1	2	1	2	1	1	1	1	2	2	3	1	3	2	1	1	3	2	2	1	1
15	2	1	2	1	1	2	1	2	1	1	2	3	3	1	2	3	2	2	2	2	2	3	1	1
16	1	2	1	2	1	2	2	1	1	1	2	2	2	1	2	1	1	3	2	3	1	1	1	1
17	1	2	2	2	2	1	1	2	2	1	2	3	2	2	3	1	3	2	2	3	2	2	3	1
18	2	1	1	2	2	2	2	2	2	1	1	1	3	3	2	3	3	3	2	3	3	2	2	1
19	1	1	2	1	2	1	2	1	1	1	1	1	3	2	2	1	1	2	3	2	3	1	3	1
20	2	1	2	1	1	2	1	2	1	1	2	1	2	2	1	2	3	1	3	3	1	1	2	1
21	2	2	2	2	2	2	1	1	1	2	1	1	2	1	3	3	1	2	1	2	1	2	2	1
22	2	1	2	2	1	1	2	1	2	2	2	2	3	2	2	2	3	2	1	1	1	2	1	1
23	2	2	1	1	2	1	2	2	1	2	2	3	3	1	3	2	1	1	3	3	3	2	1	1
24	2	1	2	2	1	1	2	1	2	2	2	3	2	3	1	1	1	1	2	2	3	3	2	1
25	1	2	1	2	1	2	2	1	1	1	2	3	1	2	1	3	2	2	3	1	3	2	2	1
26	2	2	2	2	2	2	1	1	1	2	1	2	1	2	2	2	2	1	2	3	3	3	3	1
27	1	2	2	1	1	2	2	2	2	2	1	3	3	2	1	3	1	3	1	3	1	3	3	1
28	1	2	2	1	1	2	2	2	2	2	1	1	2	3	3	2	2	2	2	1	3	1	1	1
29	2	2	1	1	2	1	2	2	1	2	2	2	1	3	1	3	3	2	2	2	1	1	3	1
30	1	1	1	1	2	2	1	1	2	2	2	3	1	3	2	2	1	2	1	3	2	1	2	1
31	2	2	1	1	2	1	2	2	1	2	2	1	2	2	2	1	2	3	1	1	2	3	2	1
32	2	1	1	2	2	2	2	2	2	1	1	3	1	2	3	1	2	1	1	2	1	1	1	1
33	1	2	2	2	2	1	1	2	2	1	2	2	3	1	1	2	2	3	1	2	3	1	2	1
34	2	1	2	2	1	1	2	1	2	2	2	1	1	1	3	3	2	3	3	3	2	1	3	1
35	2	2	1	1	1	1	1	1	2	1	1	3	2	3	2	2	2	3	3	2	1	2	3	1
36	1	2	2	1	1	2	2	2	2	2	1	2	1	1	2	1	3	1	3	2	2	2	2	1
37	2	2	2	1	2	1	2	2	2	2	1	2	2	3	2	1	3	3	3	1	1	1	3	.
38	2	1	1	1	1	2	2	1	2	1	1	3	3	2	1	2	1	2	2	1	1	1	3	.
39	2	1	2	2	1	2	1	2	2	2	2	3	3	3	2	3	2	3	2	3	3	3	3	.
40	1	2	1	1	1	2	2	2	1	2	2	1	3	1	3	2	3	3	1	2	1	3	3	.

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										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	w
37	2	2	2	1	2	1	2	2	2	2	1	2	2	3	2	1	3	3	3	1	1	1	3	.
38	2	1	1	1	1	2	2	1	2	1	1	3	3	2	1	2	1	2	2	1	1	1	3	.
39	2	1	2	2	1	2	1	2	2	2	2	3	3	3	2	3	2	3	2	3	3	3	3	.
40	1	2	1	1	1	2	2	2	1	2	2	1	3	1	3	2	3	3	1	2	1	3	3	.

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 a 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.

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 a dash.

```
- - - - - - - - - - 1 2 2 2 2 - - - - - -
- - 1 1 1 2 - - - - - - - - - - - - - - 1
- - - - - - - - - 1 1 1 1 1 - - - - - - -
- - - - - - - - - - 2 1 1 - 1 2 - - - - -
- 1 1 2 2 2 - - - - - - - - - - - - - - -
- - - - - - - - - - 2 1 2 - - 1 2 - - - -
- - - - - - - - - - 1 2 1 1 - - 2 - - - -
- - - - - - - - - - - - - - 1 1 - 2 2 1 -
- - - - 2 1 - - 1 2 2 - - - - - - - - - -
- - - - 1 2 - 2 - 2 - 1 - - - - - - - - -
- - - - - - - - - - 1 1 - 2 - - 2 1 - - -
- - - - - - - - - - - - - - 2 1 2 - - 2 2
- - - - - - - - - - 1 1 2 2 2 - - - - - -
- - - - - - - - - - - - 2 2 - 1 2 - 1 - -
- - - - - - - - - - - - - - 1 - 2 1 2 1 -
2 1 - - - - - - - - - - - - - - - - 1 1 1
- 1 - - - - 1 - 2 - 2 1 - - - - - - - - -
- - - - - - - - - - - - - - 2 2 2 2 2 -
1 2 - 1 2 - - - - - - - - - - - 1 - - -
- - - - 2 2 - 2 2 - 1 - - - - - - - - - -
2 2 1 - - - - - - - - - - - - - - 2 - 2
- - - - - - - - - - - - - - 1 2 - - 1 2 2
- - - - - 1 2 2 1 - 2 - - - - - - - - - -
- 2 - - - - - - - - - - - - - 1 - 2 2 1
2 2 1 - - 1 - - - - - - - - - - - - 2 -
- 2 2 2 - - - - - - - - - - - - - 1 - 1 -
- - - - 2 1 - - - - - - - - - - - 2 1 2
- - - 1 1 - 1 - - - - - - - - - - - 1 2
- - - - - 1 - - - 1 - - - 1 2 1 - - - -
- - - - 1 2 2 1 1 - - - - - - - - - - - -
2 - - 2 2 - 1 - - - - - - - - - - - - 1
- - - - - - - - - - - - 1 2 - 1 1 - 2 -
1 - 1 - - - 2 1 - - - - - - - - - - - 1
- - - - - 2 1 1 1 2 - - - - - - - - - - -
2 - 2 1 - - 2 - - - - - - - - - - - - 2
1 - 2 2 1 2 - - - - - - - - - - - - - - -
- - - - - - - - - - 1 2 - - 1 1 1 - - -
- - - 2 1 - - 1 2 2 - - - - - - - - - - -
- - - - - - - - - - 2 - 2 2 1 - - 1 -
- - - - - 2 2 - 2 2 1 - - - - - - - - - -
```

A design like this could be used to make a binary choice experiment. For example, the first run has factors 10 through 14 varying.

```
 - - - - - - - - - 1 2 2 2 2 - - - - - -
```

Assume they are all yes-no factors (1 yes, 2 no). Subjects could be offered a choice between these two profiles:

```
x10 = yes,   x11 = no,   x12 = no,   x13 = no,   x14 = no
x10 = no,    x11 = yes,  x12 = yes,  x13 = yes,  x14 = yes
```

The first profile came directly from the design and the second came from shifting the design: yes → no, and no → yes. Partial-profile designs have become very popular among some researchers.

Here is the code that generated and printed the partial profile design above.

```
%mktex(3 ** 20, n=41, partial=5, seed=292)

%mktlab(values=0 1 2)

proc format; value part 0 = ' -'; run;

data _null_; set final(firstobs=2); put (x1-x20) (part2.); 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 just all attributes not shown (all ones). 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 equal 5 and the number of 1's equal 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 0, 1, 2 where 0 will mean not shown, then a format is written to print zeros as dashes. A DATA step prints the design using the format excluding the constant (all not shown) first row.

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 zero, the code changes 1 to 2 and 2 to 1 and outputs the new profile. The next step uses the **%ChoiceEff** 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 **%ChoiceEff** macro is similar to what we saw in the food product example on page 217. Our choice design is specified on **data=** and the same data set, with just the **Set** variable kept, is specified on the **init=** option. The number of choice sets, 20 (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 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=20, 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	x10	x1 0	.	0	.
2	x11	x1 1	2.4989	1	1.58080
3	x12	x1 2	.	0	.
4	x20	x2 0	.	0	.
5	x21	x2 1	4.3585	1	2.08770
6	x22	x2 2	.	0	.
7	x30	x3 0	.	0	.
8	x31	x3 1	5.7179	1	2.39121
9	x32	x3 2	.	0	.
10	x40	x4 0	.	0	.
11	x41	x4 1	19.0020	1	4.35913
12	x42	x4 2	.	0	.
13	x50	x5 0	.	0	.
14	x51	x5 1	1.4018	1	1.18396
15	x52	x5 2	.	0	.
16	x60	x6 0	.	0	.
17	x61	x6 1	2.9092	1	1.70564
18	x62	x6 2	.	0	.
19	x70	x7 0	.	0	.
20	x71	x7 1	3.6474	1	1.90982
21	x72	x7 2	.	0	.
22	x80	x8 0	.	0	.
23	x81	x8 1	5.5731	1	2.36075
24	x82	x8 2	.	0	.
25	x90	x9 0	.	0	.
26	x91	x9 1	5.6681	1	2.38077
27	x92	x9 2	.	0	.
28	x100	x10 0	.	0	.
29	x101	x10 1	1.2731	1	1.12831
30	x102	x10 2	.	0	.
31	x110	x11 0	.	0	.
32	x111	x11 1	1.0522	1	1.02577
33	x112	x11 2	.	0	.
34	x120	x12 0	.	0	.
35	x121	x12 1	1.5623	1	1.24993
36	x122	x12 2	.	0	.

```
37      x130      x13 0       .          0      .
38      x131      x13 1     7.9449       1    2.81868
39      x132      x13 2       .          0      .
40      x140      x14 0       .          0      .
41      x141      x14 1     3.7175       1    1.92809
42      x142      x14 2       .          0      .
43      x150      x15 0       .          0      .
44      x151      x15 1     3.5820       1    1.89261
45      x152      x15 2       .          0      .
46      x160      x16 0       .          0      .
47      x161      x16 1     4.7737       1    2.18488
48      x162      x16 2       .          0      .
49      x170      x17 0       .          0      .
50      x171      x17 1     5.7060       1    2.38872
51      x172      x17 2       .          0      .
52      x180      x18 0       .          0      .
53      x181      x18 1    10.0597       1    3.17170
54      x182      x18 2       .          0      .
55      x190      x19 0       .          0      .
56      x191      x19 1     6.5273       1    2.55486
57      x192      x19 2       .          0      .
58      x200      x20 0       .          0      .
59      x201      x20 1     6.5214       1    2.55370
60      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. In effect, we have two reference levels, one for the not shown level and the expected one for the no level. The **%ChoicEff** macro prints a list of all redundant variables.

```
Redundant Variables:
x10 x12 x20 x22 x30 x32 x40 x42 x50 x52 x60 x62 x70 x72 x80 x82 x90 x92 x100
x102 x110 x112 x120 x122 x130 x132 x140 x142 x150 x152 x160 x162 x170 x172
x180 x182 x190 x192 x200 x202
```

We can cut this list into our program and drop those terms.

```
%choiceff(data=des,
          model=class(x1-x20 / zero=none),
          nsets=20, nalts=2,
          beta=zero, init=des(keep=set),
          intiter=0, drop=
x10 x12 x20 x22 x30 x32 x40 x42 x50 x52 x60 x62 x70 x72 x80 x82 x90 x92 x100
x102 x110 x112 x120 x122 x130 x132 x140 x142 x150 x152 x160 x162 x170 x172
x180 x182 x190 x192 x200 x202);
```

Here is the last part of the output.

Partial Profiles

n	Variable Name	Label	Variance	DF	Standard Error
1	x11	x1 1	2.4989	1	1.58080
2	x21	x2 1	4.3585	1	2.08770
3	x31	x3 1	5.7179	1	2.39121
4	x41	x4 1	19.0020	1	4.35913
5	x51	x5 1	1.4018	1	1.18396
6	x61	x6 1	2.9092	1	1.70564
7	x71	x7 1	3.6474	1	1.90982
8	x81	x8 1	5.5731	1	2.36075
9	x91	x9 1	5.6681	1	2.38077
10	x101	x10 1	1.2731	1	1.12831
11	x111	x11 1	1.0522	1	1.02577
12	x121	x12 1	1.5623	1	1.24993
13	x131	x13 1	7.9449	1	2.81868
14	x141	x14 1	3.7175	1	1.92809
15	x151	x15 1	3.5820	1	1.89261
16	x161	x16 1	4.7737	1	2.18488
17	x171	x17 1	5.7060	1	2.38872
18	x181	x18 1	10.0597	1	3.17170
19	x191	x19 1	6.5273	1	2.55486
20	x201	x20 1	6.5214	1	2.55370
				==	
				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=201)

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	41.2020	17.9653	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	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	4	2	3	4
3	1	1	1	1	1	1	1	1	4	3	2	3
4	1	1	1	1	1	1	1	2	3	3	4	1
5	1	1	1	1	1	1	2	4	4	2	1	1
6	1	1	1	1	1	1	4	3	4	4	1	1
7	1	1	1	1	1	2	4	1	2	1	4	1
8	1	1	1	1	1	4	3	3	2	1	1	1
9	1	1	1	1	2	4	2	1	3	1	1	1
10	1	1	1	1	3	3	3	1	3	1	1	1
11	1	1	1	1	3	4	4	2	1	1	1	1
12	1	1	1	1	4	1	4	4	3	1	1	1
13	1	1	1	2	2	1	2	3	1	1	1	1
14	1	1	1	2	4	4	1	1	1	1	4	1
15	1	1	1	3	2	2	1	4	1	1	1	1
16	1	1	2	1	4	2	1	3	1	1	1	1
17	1	1	2	4	2	3	1	1	1	1	1	1
18	1	2	1	1	1	1	1	2	2	4	1	1
19	1	2	1	4	1	2	3	1	1	1	1	1
20	1	2	2	3	1	1	2	1	1	1	1	1
21	1	3	3	1	2	1	3	1	1	1	1	1
22	1	3	4	3	1	1	1	1	1	1	1	3
23	1	4	3	1	1	3	1	1	1	1	3	1
24	1	4	3	4	3	1	1	1	1	1	1	1
25	1	4	4	2	1	1	1	1	1	1	1	2
26	2	1	4	1	1	1	1	1	1	1	2	4
27	2	2	3	1	1	1	1	1	1	1	1	2
28	2	3	2	2	1	1	1	1	1	1	1	1
29	2	4	1	3	4	1	1	1	1	1	1	1
30	3	1	1	1	1	1	1	1	3	2	2	1
31	3	1	4	1	1	1	1	1	4	1	3	1
32	3	3	1	1	1	1	1	1	1	3	1	4
33	3	4	1	1	1	1	1	1	1	1	4	3
34	4	1	1	1	1	1	1	1	1	4	3	3
35	4	1	3	1	1	1	1	2	1	1	1	4
36	4	2	4	1	2	1	1	1	1	1	1	1
37	4	3	1	1	1	1	1	1	1	1	2	2

Notice that the first run is constant. For all other runs, exactly four factors vary and have levels not equal to 1.

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 full profile restrictions on each alternative, and you must have the same attributes varying in 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 195 for other examples of restrictions.

In this section, we will construct a partial profile design for a purely generic study (unbranded), 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 **%ChoiceEff** 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] & x[k+10] = x[k+20]);
      end;
   bad = abs(sum - 4);
   %mend;

%mktex(3 ** 30, n=198,
       optiter=0, tabiter=0, maxdesigns=1, mutate=, anneal=, options=accept,
       out=sasuser.cand, restrictions=partprof, seed=201)
```

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 alternatives: **x1 = x11 = x21**, **x2 = x12 = x22**, ..., and **x10 = x20 = x30**. If the number of constant alternatives 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.

Our goal in this step is to make a candidate set of potential partial-profile choice sets, not to make a final experimental design. All we really need is a design with most of the runs defining valid partial-profile candidate choice sets. 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. It is not critical that we allow the macro to spend a great deal of time optimizing linear model D-efficiency. We will specify options so that the macro will just create one candidate design using the coordinate-exchange algorithm with a random initialization.

This is a big problem (30 factors and 198 runs) with restrictions, so **%MktEx** macro will run slowly by default. 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 specify **maxdesigns=1** which will just give us one design using the first method, which will now be coordinate exchange with a random initial design. We also specify **mutate=** and **anneal=** (both with null arguments) so there will be no random mutations or simulated annealing. When we have very strong restrictions like this, the simulated annealing will regularly destroy the structure of the design and make the iterations take longer, so we turn if off. We also specify **options=accept** so that the macro will still output a design even if every choice set does not meet the restrictions. We have used the other options in several previous examples. Here is a small part of the output.

Partial Profiles

Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
1	Start	85.1169		Ran
1	170 1	95.1279	95.1279	Conforms
1	170 3	95.1305	95.1305	
1	171 16	95.1314	95.1314	
1	171 6	95.1346	95.1346	

```
                .
                .
                .
                1     60   7       96.5153       96.5153
                1     66   3       96.5166       96.5166
                1        End       96.5166

                .
                .
                .
```

Partial Profiles

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	96.5165	93.4268	95.9714	0.5551

The macro finds a design that conforms to the restrictions (shown by the **Conforms** note) that is 95.1% D-efficient, and the final design is just over 96.5% D-efficient. This step took almost 28 minutes.

Here is the rest of the code for making the partial-profile choice design.

```
data des(drop=k sum);
   set sasuser.cand;
   array x[30];
   sum = 0;
   do k = 1 to 10;
      sum = sum + (x[k] = x[k+10] & x[k] = x[k+20] & x[k+10] = x[k+20]);
      end;
   if sum eq 4;
   run;

%mktkey(x1-x30)

data key;
   input (x1-x10) ($);
   datalines;
 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
;

%mktroll(design=des, 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=155,
          iter=10, nsets=27, nalts=3, options=nodups, beta=zero)

proc print data=best; id set; by notsorted set; var x1-x10; run;
```

We use a DATA step to check each profile to see if it conforms using essentially the same code we saw in the restrictions macro. The subsetting **if** statement outputs just those choice sets with four constant attributes. This step is not necessary for this particular problem because we saw in the output that the design conformed to the restrictions, however, it could be necessary for other problems. The **%MktKey** macro is run to generate the full list of names in the range **x1 - x30** for pasting into the next step. A KEY data set is created and 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 sets. 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	1	3	1	3	3	3
	2	1	2	2	3	1	2	1	1	2	3
	3	1	3	3	2	1	2	1	2	1	3
2	1	1	1	1	1	3	2	2	1	3	1
	2	3	1	1	1	2	2	1	3	3	3
	3	2	1	3	1	1	2	3	2	3	2
3	1	1	1	1	2	1	3	3	2	3	3
	2	3	1	1	2	3	1	1	1	1	3
	3	2	1	1	2	2	2	2	3	2	3

The **%ChoiceEff** macro is called to search for an efficient choice design. The specification **model=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	2.410695	0.414818
	1	2.846091	0.351359
	2	2.875865	0.347721
	3	2.881470	0.347045
.			
.			
.			

Design	Iteration	D-Efficiency	D-Error
8	0	2.317748	0.431453
	1	2.857337	0.349976
	2	2.902798	0.344495
	3	2.902798	0.344495

```
          .
          .
          .
```

Partial Profiles

```
Final Results:   Design     = 8
                 Efficiency = 2.9027977631
                 D-Error    = 0.3444952358
```

Partial Profiles

n	Variable Name	Label	Variance	DF	Standard Error
1	x11	x1 1	0.43314	1	0.65814
2	x12	x1 2	0.45528	1	0.67474
3	x21	x2 1	0.40651	1	0.63758
4	x22	x2 2	0.38230	1	0.61831
5	x31	x3 1	0.39510	1	0.62857
6	x32	x3 2	0.44795	1	0.66929
7	x41	x4 1	0.45472	1	0.67433
8	x42	x4 2	0.43250	1	0.65765
9	x51	x5 1	0.43399	1	0.65878
10	x52	x5 2	0.43367	1	0.65854
11	x61	x6 1	0.40985	1	0.64020
12	x62	x6 2	0.41572	1	0.64477
13	x71	x7 1	0.45571	1	0.67506
14	x72	x7 2	0.45254	1	0.67271
15	x81	x8 1	0.40567	1	0.63693
16	x82	x8 2	0.36000	1	0.60000
17	x91	x9 1	0.39250	1	0.62649
18	x92	x9 2	0.41535	1	0.64447
19	x101	x10 1	0.43561	1	0.66001
20	x102	x10 2	0.40682	1	0.63783
				==	
				20	

Here is part of the design.

Partial Profiles

Set	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
16	1	1	2	3	2	1	3	3	3	3
	3	1	2	2	1	2	3	1	3	2
	2	1	2	1	3	3	3	2	3	1
196	3	3	3	3	1	2	3	2	1	1
	1	2	3	2	2	3	3	2	1	2
	2	1	3	1	3	1	3	2	1	3
.										
.										
.										
77	2	1	2	2	3	1	1	1	2	2
	1	2	2	1	3	2	1	3	1	2
	3	3	2	3	3	3	1	2	3	2

The choice set number corresponds to the original set numbers in the candidate design.

Advanced Restrictions

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 you might have made. Next, it is called in between calling PROC PLAN or 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 will 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. This step allows the macro to end more gracefully than it would otherwise if there are errors. 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** (and hence **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. 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 IML so you can see the normal IML 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. 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;** and **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** contains the column index, **j2** and **j3** are zero (unless you are using **exchange=**, in which case **j1** and **j2** are indexes of other columns being exchanged), and **i** is the row number.

Restrictions Violations Check In this phase, the macro is used to check the design when there are restrictions and **options=accept**. 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 = 0; j2 = 0; j3 = 0;** 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.

The Macros

The autocall macros that are used in this report are documented in this section on the indicated pages.

Macro	Page	Release	Purpose
%ChoiceEff	288	8.0	efficient choice design
%MktAllo	303	8.1	processing allocation data
%MktBal	305	9.0	balanced main-effects designs
%MktBlock	307	9.0	block a linear or choice design
%MktDes	314	8.0	efficient linear experimental design
%MktDups	319	9.0	identify duplicate choice sets or runs
%MktEval	325	8.1	evaluate an experimental design
%MktEx	327	9.0	efficient linear experimental design
%MktKey	344	9.0	aid creation of **key=** data set
%MktLab	345	9.0	relabel and rename design factors
%MktMerge	353	8.1	merging a choice design with choice data
%MktOrth	354	9.0	list orthogonal designs that **%MktEx** can make
%MktRoll	356	8.1	rolling a linear design into a choice design
%MktRuns	360	8.0	experimental design size
%PhChoice	364	8.0	customizing the printed output from a choice model

The "Release" column indicates the first SAS release in which each macro was distributed. 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 Version 9.0 was finished before the macros were finalized and this book finished. Hence there are a few differences between the macros used in this book and those shipped with Version 9.0 of SAS. If you are running version 8.2 or version 9.0 of SAS, get the latest macros from the web at http://www.sas.com/service/techsup/tnote/tnote_stat.html or by writing Warren.Kuhfeld@sas.com. These macros will not work with Version 6.12. You should install *all* of these macros, not just one. 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** and **%MktDes** macros.

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 or your SAS root directory could be anything, but it is probably something like SAS or SAS\V8. 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**.

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. If you are specifying your own autocall libraries, remember to concatenate the autocall library supplied by SAS with your autocall libraries so that these macros will also be available. For details, refer to your host documentation and SAS macro language documentation.

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 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. 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. If you have problems with a **`%MktEx`** macro restrictions macro, see page 285 for a suggestion on how to diagnose the problem.

The **`%MktRuns`** and **`%PhChoice`** macros use 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, these 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.

%ChoicEff Macro

The **`%ChoicEff`** autocall macro is used to find efficient experimental designs for choice experiments. 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 β.

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 on across alternative restrictions (certain alternatives may not appear with certain other alternatives).

*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.

The **%ChoiceEff** macro uses a modified Federov algorithm, just like PROC OPTEX and the **%MktEx** macro. First, the **%ChoiceEff** 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. Typically, you use as a candidate set a full-factorial, fractional-factorial, or a tabled design created with the **%MktEx** macro. 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 **%ChoiceEff** macro and the **%MktEx** macro are as follows. The **%ChoiceEff** macro requires you to specify the true (or assumed true) parameters and it optimizes the variance matrix for a multinomial logit model, whereas PROC OPTEX and the **%MktEx** macro optimize 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 multiple intercepts. The option **int=f1-f3** creates three variables with values all one. The default output data set is called FINAL. Next, the **%ChoiceEff** macro is run to find an efficient design for the unbranded, purely generic model assuming $\beta = 0$. Here is the code.

```
%mktex(3 ** 3, n=3**3, seed=238)
%mktlab(int=f1-f3)

%choiceff(data=final, model=class(x1-x3), nsets=9,
          flags=f1-f3, beta=zero, seed=145)

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. Here is 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          0.817815      1.222770
                     1          1.677761      0.596032
                     2          1.702866      0.587245
                     3          1.702866      0.587245

        Design   Iteration  D-Efficiency       D-Error
        ---------------------------------------------
           2         0          0.993579      1.006463
                     1          1.662712      0.601427
                     2          1.710398      0.584659

Final Results:   Design     = 1
                 Efficiency = 1.7028655003
                 D-Error    =   0.58724544
```

n	Variable Name	Label	Variance	DF	Standard Error
1	x11	x1 1	0.68354	1	0.82677
2	x12	x1 2	0.71089	1	0.84314
3	x21	x2 1	0.70177	1	0.83772
4	x22	x2 2	0.67544	1	0.82185
5	x31	x3 1	0.67544	1	0.82185
6	x32	x3 2	0.67544	1	0.82185
				==	
				6	

Set	x1	x2	x3
1	2	1	1
	1	2	2
	3	3	3
2	2	2	3
	3	3	2
	1	1	1
3	2	1	1
	3	2	3
	1	3	2
4	3	2	2
	2	3	1
	1	1	3
5	1	2	1
	3	1	3
	2	3	2
6	1	2	2
	3	3	1
	2	1	3
7	1	3	3
	3	2	1
	2	1	2
8	1	1	2
	2	2	3
	3	3	1
9	3	1	2
	1	3	3
	2	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
                 Efficiency = 1.7320508076
                 D-Error    = 0.5773502692

                    Variable                                  Standard
               n      Name      Label    Variance    DF       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=446);
```

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 **%ChoiceEff** 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=538,
          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) 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.

The option **converge=1e-12** specifies a convergence criterion smaller than the default. 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 the output.

n	Name	Beta	Label
1	Brand1	0	Brand 1
2	Brand2	0	Brand 2
3	Brand3	0	Brand 3
4	Brand1x11	0	Brand 1 * x1 1
5	Brand1x12	0	Brand 1 * x1 2
6	Brand2x11	0	Brand 2 * x1 1
7	Brand2x12	0	Brand 2 * x1 2
8	Brand3x11	0	Brand 3 * x1 1
9	Brand3x12	0	Brand 3 * x1 2
.			
.			
.			

```
Design   Iteration  D-Efficiency       D-Error
-----------------------------------------------
   1         0                0             .
             1                0             .
                       0.300889 (Ridged)
             2                0             .
                       0.304093 (Ridged)
             3                0             .
                       0.304853 (Ridged)
             4                0             .
                       0.304967 (Ridged)
             5                0             .
                       0.304967 (Ridged)

Design   Iteration  D-Efficiency       D-Error
-----------------------------------------------
   2         0                0             .
             1                0             .
                       0.300306 (Ridged)
             .
             .
             .
             6                0             .
                       0.303801 (Ridged)

Final Results:   Design     = 1
                 Efficiency =            0
                 D-Error    =            .

          Variable                                          Standard
 n        Name         Label              Variance    DF      Error

 1        Brand1       Brand 1             3.79498     1     1.94807
 2        Brand2       Brand 2             5.58130     1     2.36248
 3        Brand3       Brand 3               .         0      .
 4        Brand1x11    Brand 1 * x1 1      2.10386     1     1.45047
 5        Brand1x12    Brand 1 * x1 2      2.32036     1     1.52327
 6        Brand2x11    Brand 2 * x1 1      2.51756     1     1.58668
 7        Brand2x12    Brand 2 * x1 2      1.87117     1     1.36791
 8        Brand3x11    Brand 3 * x1 1      2.27177     1     1.50724
 9        Brand3x12    Brand 3 * x1 2      2.08819     1     1.44506
 .
 .
 .
21        Brand3x32    Brand 3 * x3 2      2.19338     1     1.48100
                                                       ==
                                                       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**), which is a structural-zero effect. 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 **%ChoiceEff** macro can still optimize the goodness of the design by optimizing a ridged efficiency criterion. That is what is shown in the iteration history. The option **converge=1e-12** was specified because for this example, iteration stops prematurely with the default convergence criterion. These next steps switch 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,
          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 296). 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 291, the option **initvars=x1-x3** was specified because the initial design was not created by the **%ChoiceEff** 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          0.683297      1.463493
                     1          0.683297      1.463493

Final Results:   Design     = 1
                 Efficiency =  0.683296784
                 D-Error    = 1.4634929117

                 Variable                                          Standard
          n      Name         Label                Variance    DF      Error

          1      Brand1       Brand 1               3.79498     1     1.94807
          2      Brand2       Brand 2               5.58130     1     2.36248
          3      Brand1x11    Brand 1 * x1 1        2.10386     1     1.45047
          4      Brand1x12    Brand 1 * x1 2        2.32036     1     1.52327
          5      Brand2x11    Brand 2 * x1 1        2.51756     1     1.58668
          6      Brand2x12    Brand 2 * x1 2        1.87117     1     1.36791
          7      Brand3x11    Brand 3 * x1 1        2.27177     1     1.50724
          8      Brand3x12    Brand 3 * x1 2        2.08819     1     1.44506
          9      Brand1x21    Brand 1 * x2 1        2.35839     1     1.53570
         10      Brand1x22    Brand 1 * x2 2        2.19604     1     1.48190
         11      Brand2x21    Brand 2 * x2 1        2.42179     1     1.55621
         12      Brand2x22    Brand 2 * x2 2        2.14472     1     1.46449
         13      Brand3x21    Brand 3 * x2 1        2.67577     1     1.63578
         14      Brand3x22    Brand 3 * x2 2        2.29487     1     1.51488
         15      Brand1x31    Brand 1 * x3 1        2.24390     1     1.49797
         16      Brand1x32    Brand 1 * x3 2        2.24520     1     1.49840
         17      Brand2x31    Brand 2 * x3 1        2.05763     1     1.43444
         18      Brand2x32    Brand 2 * x3 2        2.15244     1     1.46712
         19      Brand3x31    Brand 3 * x3 1        2.17488     1     1.47475
         20      Brand3x32    Brand 3 * x3 2        2.19338     1     1.48100
                                                                ==
                                                                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)

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=543,
          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	0.012495	1.00	1	0.11178	8.9462	0.0001
2	Brand2	Brand 2	0.029691	2.00	1	0.17231	11.6068	0.0001
3	Brand1x11	Brand 1 * x1 1	0.012907	-0.50	1	0.11361	-4.4011	0.0001
4	Brand1x12	Brand 1 * x1 2	0.009894	0.50	1	0.09947	5.0267	0.0001
5	Brand2x11	Brand 2 * x1 1	0.012299	-0.75	1	0.11090	-6.7628	0.0001
6	Brand2x12	Brand 2 * x1 2	0.013671	0.75	1	0.11692	6.4144	0.0001
7	Brand3x11	Brand 3 * x1 1	0.022617	-1.00	1	0.15039	-6.6494	0.0001
8	Brand3x12	Brand 3 * x1 2	0.020324	1.00	1	0.14256	7.0145	0.0001
9	Brand1x21	Brand 1 * x2 1	0.010342	-0.50	1	0.10170	-4.9166	0.0001
10	Brand1x22	Brand 1 * x2 2	0.011604	0.50	1	0.10772	4.6416	0.0001
11	Brand2x21	Brand 2 * x2 1	0.010035	-0.75	1	0.10018	-7.4868	0.0001
12	Brand2x22	Brand 2 * x2 2	0.012963	0.75	1	0.11386	6.5873	0.0001
13	Brand3x21	Brand 3 * x2 1	0.020136	-1.00	1	0.14190	-7.0471	0.0001
14	Brand3x22	Brand 3 * x2 2	0.015137	1.00	1	0.12303	8.1279	0.0001
15	Brand1x31	Brand 1 * x3 1	0.011747	-0.50	1	0.10838	-4.6133	0.0001
16	Brand1x32	Brand 1 * x3 2	0.009208	0.50	1	0.09596	5.2105	0.0001
17	Brand2x31	Brand 2 * x3 1	0.012416	-0.75	1	0.11143	-6.7308	0.0001
18	Brand2x32	Brand 2 * x3 2	0.012723	0.75	1	0.11280	6.6491	0.0001
19	Brand3x31	Brand 3 * x3 1	0.016991	-1.00	1	0.13035	-7.6717	0.0001
20	Brand3x32	Brand 3 * x3 2	0.013543	1.00	1	0.11637	8.5931	0.0001
					==			
					20			

These next steps create a design for a cross-effects model with five brands at three prices and a constant alternative. 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 179 and 217 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	

%ChoiceEff Macro Options

The following options can be used with the **%ChoiceEff** macro. 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),
```

nsets= n

specifies the number of choice sets desired.

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.

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.

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**

- 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.

fixed= 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 options 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.

notes
stops the macro from submitting the statement **options nonotes**.

notests
suppresses printing the diagonal of the covariance matrix, and hypothesis tests for this n and β. When β 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 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. However, due to machine differences, some results may not be exactly reproducible on other machines, 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.

%MktAllo Macro

The **%MktAllo** autocall macro is used for manipulating data for an allocation choice experiment. It 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. 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.

%MktBal Macro

The **%MktBal** macro creates linear experimental designs using an algorithm that ensures that the design is perfectly balanced, or in the case 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 hundreds 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. The **%MktBal** macro has several options that can make it run faster for large problems, but at a price of an even further decrease in D-efficiency. 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 hence, by default, will run slowly. You can use the **maxstarts=** and **maxiter=** options to make the macro run more quickly. For example, the second example below 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)
```

%MktBal Macro Options

The following options can be used with the **%MktBal** macro.

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 four 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 as the random number seed. 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, although you would expect the efficiency differences to be slight.

%MktBlock Macro

The **%MktBlock** autocall macro is used to block a choice design or an ordinary linear experimental design. 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 **%ChoiceEff** 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 just 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)

%mktlab(vars=x1-x6 Block)
```

Here is an example of creating a design then blocking it.

```
%mktex(3 ** 6, n=27, seed=350)

%mktblock(data=randomized, nblocks=3, seed=377)
```

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.

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	0.58	0.58	0.58	0.58
	x2	0	1	0	0	0	0
	x3	0.58	0	1	0	0.58	0.58
	x4	0.58	0	0	1	0.58	0.58
	x5	0.58	0	0.58	0.58	1	0
	x6	0.58	0	0.58	0.58	0	1
2	x1	1	0	0.58	0.58	0.58	0.58
	x2	0	1	0	0	0	0
	x3	0.58	0	1	0	0.58	0.58
	x4	0.58	0	0	1	0.58	0.58
	x5	0.58	0	0.58	0.58	1	0
	x6	0.58	0	0.58	0.58	0	1

3	x1	1	0	0	0	0	0
	x2	0	1	0	0	0	0
	x3	0	0	1	0	1.00	0
	x4	0	0	0	1	0	1.00
	x5	0	0	1.00	0	1	0
	x6	0	0	0	1.00	0	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	3	2	2	2	3	3
	2	3	3	1	3	2	1
	3	3	1	2	3	2	3
	4	1	2	1	1	2	2
	5	1	3	3	2	1	3
	6	2	1	3	1	3	1
	7	2	2	3	3	1	1
	8	2	3	2	1	3	2
	9	1	1	1	2	1	2

Block	Run	x1	x2	x3	x4	x5	x6
2	1	3	2	1	1	1	1
	2	3	3	3	2	3	2
	3	3	1	3	1	1	2
	4	2	3	1	3	1	3
	5	1	1	2	3	3	1
	6	1	3	2	1	2	1
	7	1	2	3	3	3	3
	8	2	1	1	2	2	3
	9	1	2	3	2	1	3

Block	Run	x1	x2	x3	x4	x5	x6
3	1	1	3	1	3	3	2
	2	2	2	1	1	3	3
	3	2	1	2	3	1	2
	4	2	3	3	2	2	1
	5	1	1	3	1	2	3
	6	3	2	3	3	2	2
	7	3	1	1	2	3	1
	8	1	2	2	2	1	1
	9	3	3	2	1	1	3

Next, we'll 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 version 9 and later SAS versions, 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
```

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.

alt= *variable*

specifies the variable to contain the alternative number. 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 variable to contain the block number. 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. The default is **data=_last_**. 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 in the **block=**, **set=**, and **alt=** 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 version 9.0 or a later SAS release.

id= *variable-list*

specifies variables in the **data=** data set 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 version 9.0 or a later SAS release.

initblock= *variable*

specifies the name of a variable in the data set that is to be used as the initial blocking variable for the first iteration.

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 swap. The specification **next=1** specifies that the macro should try swapping the level for each run with the level for the next run and all other runs. The specification **next=2** considers swaps with half of the other runs, which makes the algorithm run more quickly. The macro considers swapping the level for run i with run $i+1$ then uses the **next=** value to find the next potential swaps. Other values, including nonintegers can be specified as well. For example **next=1.5** considers swapping 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.

out= *SAS-data-set*

specifies the output data set with the block numbers. The default is **out=blocked**.

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**. Values include:

corr	canonical correlations
list	list of big canonical correlations
freqs	long frequencies list
summ	frequency summaries
block	canonical correlations within blocks
design	blocked design
note	blocking note
all	all of the above
noprint	no printed output
normal	**corr list summ block design note**
short	**corr summ note**

ridge= n

specifies the value to add to the diagonal of $(\mathbf{X}'\mathbf{X})^{-1}$ 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 as the random number seed. 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, although you would expect the efficiency differences to be slight.

set= *variable*

specifies the variable to contain the choice set number. 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.

%MktDes Macro

The **%MktDes** autocall macro creates efficient experimental designs. Throughout this report, we used the **%MktEx** autocall macro, which calls the **%MktDes** macro, to design our experiments, Usually, we will not need to call the **%MktDes** macro directly. At the heart of the **%MktDes** macro are PROC PLAN, PROC FACTEX, and PROC OPTEX. We use macros instead of calling these procedures directly because the macros have a simpler syntax. In extreme cases, a single-line macro call can generate hundreds of lines of otherwise tedious to write procedure code.

The **%MktDes** macro creates efficient experimental designs. 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 also 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 run 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):

```
*---Two 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 other 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.

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 optional. Optionally, the number of pseudo-factors can also be specified. 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 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 is 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 a little 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 n 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_federov** (modified Federov).

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. 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**.

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 as the random number seed. 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, 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;
```

Say you specified **nlev=2** or the macro defaulted to **nlev=2**. Increase the **size=** value by a factor of two each time. For example, 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 multistep 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)
```

Returning to the **%MktDes** macro, the first 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_federov;
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.

%MktDups Macro

The **%MktDups** autocall macro detects duplicate choice sets and duplicate alternatives within generic choice sets. For example, 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 **%ChoiceEff** 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 twice in the design.

The FREQ Procedure

Set	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	3	5.56	3	5.56
3	3	5.56	6	11.11
4	6	11.11	12	22.22
8	3	5.56	15	27.78
10	3	5.56	18	33.33
11	3	5.56	21	38.89
16	3	5.56	24	44.44
19	3	5.56	27	50.00
21	6	11.11	33	61.11
22	6	11.11	39	72.22
24	3	5.56	42	77.78
25	3	5.56	45	83.33
26	3	5.56	48	88.89
27	6	11.11	54	100.00

The output from the **%MktDups** macro contains the following tables:

```
Design:          Branded
Factors:         brand x1-x3
                 Brand
                 x1 x2 x3
Duplicate Sets:  4
```

Choice Set	Duplicate Choice Sets To Delete
1	5
2	12
4	18
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 12 are the same as are 4 and 18, and also 11 and 14. 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=93)
%mktlab(int=f1-f4)

%choiceff(data=final, model=class(x1-x5), seed=109,
          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: 2
Duplicate Sets:  2
```

Choice Set	Duplicate Choice Sets To Delete	
3	36	
5		Alternatives
24		Alternatives
35	42	

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:

Set	x1	x2	x3	x4	x5
3	2	1	1	1	2
	1	1	1	1	1
	1	2	2	2	2
	2	2	2	2	1
5	1	1	2	1	2
	1	1	2	1	2
	2	2	1	2	1
	2	2	1	2	1
24	1	2	2	1	2
	1	2	2	1	2
	2	1	1	2	1
	2	1	1	2	1
35	2	1	2	1	1
	2	2	1	2	2
	1	1	2	1	2
	1	2	1	2	1
36	2	1	1	1	2
	1	1	1	1	1
	1	2	2	2	2
	2	2	2	2	1
42	1	1	2	1	2
	2	1	2	1	1
	1	2	1	2	1
	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


                          Duplicate
                Choice    Choice Sets
                   Set      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=513)

%mktdups(linear, factors=x1-x5);
```

```
Design:          Linear
Factors:         x1-x5
                 x1 x2 x3 x4 x5
Duplicate Runs:  1
```

Run	Duplicate Runs To Delete
9	10

%MktDups Macro Options

The following options can be used with the **%MktDups** macro.

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.

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)
```

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.

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.

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.

%MktEval Macro

The **`%MktEval`** autocall macro helps you evaluate an experimental design. This 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. 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.

%MktEval Macro Options

The following options can be used with the **`%MktEval`** macro.

blocks= *variable*
specifies a blocking variable. This option prints separate canonical correlations within each bloc. 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 create *huge* data sets, whereas **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 with-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= **short|corr|list|freqs|summ|all**

controls the printing of the results. Specify one or more values from the following list.

corr	prints the canonical correlations matrix.
block	prints the canonical correlations within block.
list	prints the list of canonical correlations greater than the **list=** value.
freqs	prints the frequencies, specified by the **freqs=** option.
summ	prints the frequency summaries.
all	prints all of the above.
short	is the default and is equivalent to: **corr list summ block**.
noprint	specifies 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.

%MktEx Macro

The **%MktEx** autocall macro is designed for marketing researchers and any one else who wants to make good, efficient experimental designs. This 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 linear 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 particularly designed to easily create the kinds of linear designs that marketing researches need for conjoint and choice experiments. For any linear design problem, 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.

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 (rows or 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 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% 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 (also 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% 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 of the design matrix (see page 80). See Kuhfeld, Tobias, and Garratt (1994) on page 25for 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. 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.

For certain designs, you can just specify **n=**, which specifies the number of runs or rows in the design, and the macro will give you a 100% efficient design. Certain tabled designs, fractional and full-factorial designs, and Hadamard designs can be requested just by specifying **n=**. The tables show some of the designs up through **n=100** that can be obtained by just specifying **n=**. The macro will also output certain full-factorial designs with just a specification of **n=**. For example **n=162** implies **2 3 ** 4**. The **n=** value is factored into a list of prime factors, and the full-factorial design for that list is created. The tables show some of the 100% efficient designs that the macro can directly create.

Implied Designs When Only **n=** is Specified

N	Design	Reference
4	2^3	Hadamard
8	2^7	Hadamard
9	3^4	Fractional
12	2^{11}	Hadamard
16	2^{15}	Hadamard
18	$2^1 3^7$	Taguchi 1987
20	2^{19}	Hadamard
24	2^{23}	Hadamard
25	5^6	Fractional
27	3^{13}	Fractional
28	2^{27}	Hadamard

N	Design	Reference
32	2^{31}	Hadamard
36	$2^{11} 3^{12}$	Taguchi 1987
40	2^{39}	Hadamard
44	2^{43}	Hadamard
48	$2^{11} 4^{12}$	Suen 1989
49	7^8	Fractional
50	$2^1 5^{11}$	Taguchi 1987
52	2^{51}	Hadamard
54	23^{25}	Taguchi 1987
56	2^{55}	Hadamard
60	2^{59}	Hadamard

N	Design	Reference
64	2^{63}	Hadamard
68	2^{67}	Hadamard
72	$2^{23} 3^{24}$	Dey 1985
76	2^{75}	Hadamard
80	2^{79}	Hadamard
81	3^{40}	Fractional
84	2^{83}	Hadamard
88	2^{87}	Hadamard
92	2^{91}	Hadamard
96	2^{95}	Hadamard
100	2^{99}	Hadamard

Some of the Orthogonal Designs Available from the **%MktEx** Macro

N	Design			Reference
4	2^3			Hadamard
6	2^1	3^1		Full-factorial
8	2^7			Hadamard
9		3^4		Fractional-factorial
10	2^1		5^1	Full-factorial
12	2^{11}			Hadamard
12	2^4	3^1		Hedayat, Sloane & Stufken
12	2^2		6^1	Hedayat, Sloane & Stufken
12		3^1	4^1	Full-factorial
14	2^1		7^1	Full-factorial
15		3^1	5^1	Full-factorial
16	2^{15}			Hadamard
16	2^{12}		4^1	Fractional-factorial
16	2^9		4^2	Fractional-factorial
16	2^6		4^3	Fractional-factorial
16	2^3		4^4	Fractional-factorial
16			4^5	Fractional-factorial
18	2^1	3^7		Taguchi 1987
18		3^6	6^1	Taguchi 1987
20	2^{19}			Hadamard
20	2^8		5^1	Wang & Wu 1992
20	2^2		10^1	Hedayat, Sloane & Stufken
20			$4^1\ 5^1$	Full-factorial
21		3^1	7^1	Full-factorial
22	2^1		11^1	Full-factorial
24	2^{23}			Hadamard
24	2^{20}		4^1	Hadamard
24	2^{16}	3^1		Wang & Wu 1991
24	2^{14}		6^1	Wang & Wu 1991
24	2^{13}	3^1	4^1	Wang & Wu 1991
24	2^{12}		12^1	Hedayat, Sloane & Stufken
24	2^{11}		$4^1\ 6^1$	Wang & Wu 1991
24		3^1	8^1	Full-factorial
25			5^6	Fractional-factorial
26	2^1		13^1	Full-factorial
27	3^{13}			Fractional-factorial
27	3^9		9^1	Fractional-factorial

N	Design			Reference
28	2^{27}			Hadamard
28	2^{12}		7^1	Suen 1989
30	2^1	3^1	5^1	Full-factorial
32	2^{31}			Hadamard
32	2^{28}		4^1	Fractional-factorial
32	2^{25}		4^2	Fractional-factorial
32	2^{24}		8^1	Fractional-factorial
32	2^{22}		4^3	Fractional-factorial
32	2^{21}		$4^1\ 8^1$	Fractional-factorial
32	2^{19}		4^4	Fractional-factorial
32	2^{18}		$4^2\ 8^1$	Fractional-factorial
32	2^{16}		16^1	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
32			$4^8\ 8^1$	Fractional-factorial
33		3^1	11^1	Full-factorial
34	2^1		17^1	Full-factorial
35			$5^1\ 7^1$	Full-factorial
36	2^{35}			Hadamard
36	2^{27}	3^1		Hedayat, Sloane & Stufken
36	2^{20}	3^2		Hedayat, Sloane & Stufken
36	2^{18}	3^1	6^1	Hedayat, Sloane & Stufken
36	2^{13}	3^4		Suen 1989
36	2^{13}		9^1	Suen 1989
36	2^{11}	3^{12}		Taguchi 1987
36	2^{11}	3^2	6^1	Hedayat, Sloane & Stufken
36	2^9	3^1	6^2	Hedayat, Sloane & Stufken
36	2^4	3^{13}		Taguchi 1987
36	2^4	3^3	6^1	Hedayat, Sloane & Stufken

Some of the Orthogonal Designs Available from the **%MktEx** Macro

N	Design			Reference
36	2^2	3^{12}	6^1	Wang & Wu 1991
36	2^2	3^2	6^2	Hedayat, Sloane & Stufken
36	2^2		18^1	Hedayat, Sloane & Stufken
36	2^1	3^8	6^2	Zhang, Lu & Pang 1999
36	2^1		6^3	SAS Procedure OPTEX
36		3^{13}	4^1	Dey 1985
36		3^{12}	12^1	Wang & Wu 1991
36		3^7	6^3	Finney 1982
38	2^1		19^1	Full-factorial
39		3^1	13^1	Full-factorial
40	2^{39}			Hadamard
40	2^{36}		4^1	Hadamard
40	2^{28}		5^1	Hedayat, Sloane & Stufken
40	2^{22}		10^1	Hedayat, Sloane & Stufken
40	2^{20}		20^1	Dey 1985
40	2^{20}		$4^1\ 5^1$	Dey 1985
42	2^1	3^1	7^1	Full-factorial
44	2^{43}			Hadamard
45		3^2	5^1	Full-factorial
46	2^1		23^1	Full-factorial
48	2^{47}			Hadamard
48	2^{44}		4^1	Suen 1989
48	2^{41}		4^2	Suen 1989
48	2^{40}	3^1		Suen 1989
48	2^{38}		6^1	Suen 1989
48	2^{38}		4^3	Suen 1989
48	2^{37}	3^1	4^1	Suen 1989
48	2^{36}		12^1	Suen 1989
48	2^{35}		4^4	Suen 1989
48	2^{35}		$4^1\ 6^1$	Suen 1989
48	2^{34}	3^1	4^2	Suen 1989
48	2^{33}		$4^1\ 12^1$	Suen 1989
48	2^{32}		4^5	Suen 1989
48	2^{32}		$4^2\ 6^1$	Suen 1989
48	2^{31}	3^1	4^3	Suen 1989
48	2^{30}		$4^2\ 12^1$	Suen 1989
48	2^{29}		4^6	Suen 1989
48	2^{29}		$4^3\ 6^1$	Suen 1989
48	2^{28}	3^1	4^4	Suen 1989
48	2^{27}		$4^3\ 12^1$	Suen 1989
48	2^{26}		4^7	Suen 1989
48	2^{26}		$4^4\ 6^1$	Suen 1989
48	2^{25}	3^1	4^5	Suen 1989
48	2^{24}	3^1	8^1	Hedayat, Sloane & Stufken
48	2^{24}		24^1	Hedayat, Sloane & Stufken
48	2^{24}		$4^4\ 12^1$	Suen 1989
48	2^{23}		4^8	Suen 1989
48	2^{23}		$4^5\ 6^1$	Suen 1989
48	2^{22}	3^1	4^6	Suen 1989
48	2^{21}		$4^5\ 12^1$	Suen 1989

N	Design			Reference
48	2^{20}		4^9	Suen 1989
48	2^{20}		$4^6\ 6^1$	Suen 1989
48	2^{19}	3^1	4^7	Suen 1989
48	2^{18}		$4^6\ 12^1$	Suen 1989
48	2^{17}		4^{10}	Suen 1989
48	2^{17}		$4^7\ 6^1$	Suen 1989
48	2^{16}	3^1	4^8	Suen 1989
48	2^{15}		$4^7\ 12^1$	Suen 1989
48	2^{14}		4^{11}	Suen 1989
48	2^{14}		$4^8\ 6^1$	Suen 1989
48	2^{13}	3^1	4^9	Suen 1989
48	2^{12}		$4^8\ 12^1$	Suen 1989
48	2^{11}		4^{12}	Suen 1989
48	2^{11}		$4^9\ 6^1$	Suen 1989
48	2^{10}	3^1	4^{10}	Suen 1989
48	2^9		$4^9\ 12^1$	Suen 1989
48	2^8		$4^{10}\ 6^1$	Suen 1989
48	2^7	3^1	4^{11}	Suen 1989
48	2^6		$4^{10}\ 12^1$	Suen 1989
48	2^5		$4^{11}\ 6^1$	Suen 1989
48	2^4	3^1	4^{12}	Suen 1989
48	2^3		$4^{11}\ 12^1$	Suen 1989
48	2^2		$4^{12}\ 6^1$	Suen 1989
48		3^1	4^{13}	Suen 1989
48			$4^{12}\ 12^1$	Suen 1989
49			7^8	Fractional-factorial
50	2^1		5^{11}	Taguchi 1987
50			$5^{10}\ 10^1$	Hedayat, Sloane & Stufken
52	2^{51}			Hadamard
54	2^1	3^{25}		Taguchi 1987
54		3^{24}	6^1	Hedayat, Sloane & Stufken
54		3^{18}	18^1	Hedayat, Sloane & Stufken
55			$5^1\ 11^1$	Full-factorial
56	2^{55}			Hadamard
56	2^{52}		4^1	Hedayat, Sloane & Stufken
58	2^1		29^1	Full-factorial
60	2^{59}			Hadamard
62	2^1		31^1	Full-factorial
63		3^1	21^1	Full-factorial
64	2^{63}			Hadamard
64	2^{60}		4^1	Fractional-factorial
64	2^{57}		4^2	Fractional-factorial
64	2^{56}		8^1	Fractional-factorial
64	2^{54}		4^3	Fractional-factorial
64	2^{53}		$4^1\ 8^1$	Fractional-factorial
64	2^{51}		4^4	Fractional-factorial
64	2^{50}		$4^2\ 8^1$	Fractional-factorial
64	2^{49}		8^2	Fractional-factorial
64	2^{48}		4^5	Fractional-factorial
64	2^{47}		$4^3\ 8^1$	Fractional-factorial

Some of the Orthogonal Designs Available from the **%MktEx** Macro

N	Design			Reference
64	2^{46}	4^{1}	8^{2}	Fractional-factorial
64	2^{45}	4^{6}		Fractional-factorial
64	2^{44}	4^{4}	8^{1}	Fractional-factorial
64	2^{43}	4^{2}	8^{2}	Fractional-factorial
64	2^{42}	4^{7}		Fractional-factorial
64	2^{42}	8^{3}		Fractional-factorial
64	2^{41}	4^{5}	8^{1}	Fractional-factorial
64	2^{40}	4^{3}	8^{2}	Fractional-factorial
64	2^{39}	4^{8}		Fractional-factorial
64	2^{39}	4^{1}	8^{3}	Fractional-factorial
64	2^{38}	4^{6}	8^{1}	Fractional-factorial
64	2^{37}	4^{4}	8^{2}	Fractional-factorial
64	2^{36}	4^{9}		Fractional-factorial
64	2^{36}	4^{2}	8^{3}	Fractional-factorial
64	2^{35}	4^{7}	8^{1}	Fractional-factorial
64	2^{35}	8^{4}		Fractional-factorial
64	2^{34}	4^{5}	8^{2}	Fractional-factorial
64	2^{33}	4^{10}		Fractional-factorial
64	2^{33}	4^{3}	8^{3}	Fractional-factorial
64	2^{32}	4^{8}	8^{1}	Fractional-factorial
64	2^{32}	4^{1}	8^{4}	Fractional-factorial
64	2^{31}	4^{6}	8^{2}	Fractional-factorial
64	2^{30}	4^{11}		Fractional-factorial
64	2^{30}	4^{4}	8^{3}	Fractional-factorial
64	2^{29}	4^{9}	8^{1}	Fractional-factorial
64	2^{29}	4^{2}	8^{4}	Fractional-factorial
64	2^{28}	4^{7}	8^{2}	Fractional-factorial
64	2^{28}	8^{5}		Fractional-factorial
64	2^{27}	4^{12}		Fractional-factorial
64	2^{27}	4^{5}	8^{3}	Fractional-factorial
64	2^{26}	4^{10}	8^{1}	Fractional-factorial
64	2^{26}	4^{3}	8^{4}	Fractional-factorial
64	2^{25}	4^{8}	8^{2}	Fractional-factorial
64	2^{25}	4^{1}	8^{5}	Fractional-factorial
64	2^{24}	4^{13}		Fractional-factorial
64	2^{24}	4^{6}	8^{3}	Fractional-factorial
64	2^{23}	4^{11}	8^{1}	Fractional-factorial
64	2^{23}	4^{4}	8^{4}	Fractional-factorial
64	2^{22}	4^{9}	8^{2}	Fractional-factorial
64	2^{22}	4^{2}	8^{5}	Fractional-factorial
64	2^{21}	4^{14}		Fractional-factorial
64	2^{21}	4^{7}	8^{3}	Fractional-factorial
64	2^{21}	8^{6}		Fractional-factorial
64	2^{20}	4^{12}	8^{1}	Fractional-factorial
64	2^{20}	4^{5}	8^{4}	Fractional-factorial
64	2^{19}	4^{10}	8^{2}	Fractional-factorial
64	2^{19}	4^{3}	8^{5}	Fractional-factorial
64	2^{18}	4^{15}		Fractional-factorial
64	2^{18}	4^{8}	8^{3}	Fractional-factorial
64	2^{18}	4^{1}	8^{6}	Fractional-factorial

N	Design			Reference
64	2^{17}	4^{13}	8^{1}	Fractional-factorial
64	2^{17}	4^{6}	8^{4}	Fractional-factorial
64	2^{16}	4^{11}	8^{2}	Fractional-factorial
64	2^{16}	4^{4}	8^{5}	Fractional-factorial
64	2^{15}	4^{16}		Fractional-factorial
64	2^{15}	4^{9}	8^{3}	Fractional-factorial
64	2^{15}	4^{2}	8^{6}	Fractional-factorial
64	2^{14}	4^{14}	8^{1}	Fractional-factorial
64	2^{14}	4^{7}	8^{4}	Fractional-factorial
64	2^{14}	8^{7}		Fractional-factorial
64	2^{13}	4^{12}	8^{2}	Fractional-factorial
64	2^{13}	4^{5}	8^{5}	Fractional-factorial
64	2^{12}	4^{17}		Fractional-factorial
64	2^{12}	4^{10}	8^{3}	Fractional-factorial
64	2^{12}	4^{3}	8^{6}	Fractional-factorial
64	2^{11}	4^{15}	8^{1}	Fractional-factorial
64	2^{11}	4^{8}	8^{4}	Fractional-factorial
64	2^{11}	4^{1}	8^{7}	Fractional-factorial
64	2^{10}	4^{13}	8^{2}	Fractional-factorial
64	2^{10}	4^{6}	8^{5}	Fractional-factorial
64	2^{9}	4^{18}		Fractional-factorial
64	2^{9}	4^{11}	8^{3}	Fractional-factorial
64	2^{9}	4^{4}	8^{6}	Fractional-factorial
64	2^{8}	4^{16}	8^{1}	Fractional-factorial
64	2^{8}	4^{9}	8^{4}	Fractional-factorial
64	2^{8}	4^{2}	8^{7}	Fractional-factorial
64	2^{7}	4^{14}	8^{2}	Fractional-factorial
64	2^{7}	4^{7}	8^{5}	Fractional-factorial
64	2^{7}	8^{8}		Fractional-factorial
64	2^{6}	4^{19}		Fractional-factorial
64	2^{6}	4^{12}	8^{3}	Fractional-factorial
64	2^{6}	4^{5}	8^{6}	Fractional-factorial
64	2^{5}	4^{17}	8^{1}	Fractional-factorial
64	2^{5}	4^{10}	8^{4}	Fractional-factorial
64	2^{4}	4^{15}	8^{2}	Fractional-factorial
64	2^{4}	4^{8}	8^{5}	Fractional-factorial
64	2^{4}	4^{1}	8^{8}	Fractional-factorial
64	2^{3}	4^{20}		Fractional-factorial
64	2^{3}	4^{13}	8^{3}	Fractional-factorial
64	2^{3}	4^{6}	8^{6}	Fractional-factorial
64		4^{21}		Fractional-factorial
64		4^{7}	8^{6}	Fractional-factorial
65		5^{1}	13^{1}	Full-factorial
66	2^{1} 3^{1}	11^{1}		Full-factorial
66	2^{1}	33^{1}		Full-factorial
68	2^{67}			Hadamard
69	3^{1}	23^{1}		Full-factorial
70	2^{1}	5^{1}	7^{1}	Full-factorial
72	2^{71}			Hadamard
72	2^{68}	4^{1}		Hadamard

Some of the Orthogonal Designs Available from the **%MktEx** Macro

N	Design			Reference
72	2^{63}	3^{1}		Hedayat, Sloane & Stufken
72	2^{56}	3^{2}		Hedayat, Sloane & Stufken
72	2^{54}	3^{1}	6^{1}	Hedayat, Sloane & Stufken
72	2^{49}	3^{4}		Hedayat, Sloane & Stufken
72	2^{49}		9^{1}	Hedayat, Sloane & Stufken
72	2^{47}	3^{12}		Wang 1996
72	2^{47}	3^{2}	6^{1}	Hedayat, Sloane & Stufken
72	2^{45}	3^{1}	6^{2}	Hedayat, Sloane & Stufken
72	2^{40}	3^{13}		Wang & Wu 1991
72	2^{40}	3^{3}	6^{1}	Hedayat, Sloane & Stufken
72	2^{38}	3^{12}	6^{1}	Hedayat, Sloane & Stufken
72	2^{38}	3^{2}	6^{2}	Hedayat, Sloane & Stufken
72	2^{38}		18^{1}	Hedayat, Sloane & Stufken
72	2^{37}	3^{8}	6^{2}	Hedayat, Sloane & Stufken
72	2^{37}		6^{3}	Hedayat, Sloane & Stufken
72	2^{36}	3^{13}	4^{1}	Hedayat, Sloane & Stufken
72	2^{36}	3^{12}	12^{1}	Hedayat, Sloane & Stufken
72	2^{36}	3^{7}	6^{3}	Hedayat, Sloane & Stufken
72	2^{36}		36^{1}	Hedayat, Sloane & Stufken
72	2^{23}	3^{24}		Dey 1985
72	2^{20}	3^{24}	4^{1}	Wang 1996
72	2^{16}	3^{25}		Wang 1996
72	2^{14}	3^{24}	6^{1}	Wang 1996
72	2^{13}	3^{25}	4^{1}	Wang 1996
72	2^{12}	3^{24}	12^{1}	Hedayat, Sloane & Stufken
72	2^{11}	3^{24}	$4^{1}\ 6^{1}$	Wang 1996
72		3^{25}	8^{1}	Hedayat, Sloane & Stufken
72		3^{24}	24^{1}	Hedayat, Sloane & Stufken
74	2^{1}		37^{1}	Full-factorial
75		3^{1}	5^{2}	Full-factorial
76	2^{75}			Hadamard
77			$7^{1}\ 11^{1}$	Full-factorial
78	2^{1}		39^{1}	Full-factorial
80	2^{79}			Hadamard
80	2^{76}		4^{1}	Wang 1996
80	2^{73}		4^{2}	Wang 1996
80	2^{70}		4^{3}	Wang 1996
80	2^{49}		5^{1}	Hedayat, Sloane & Stufken
80	2^{43}		10^{1}	Hedayat, Sloane & Stufken
80	2^{41}		20^{1}	Hedayat, Sloane & Stufken
80	2^{41}		$4^{1}\ 5^{1}$	Hedayat, Sloane & Stufken
81		3^{40}		Fractional-factorial
81		3^{36}	9^{1}	Fractional-factorial
81		3^{32}	9^{2}	Fractional-factorial
81		3^{28}	9^{3}	Fractional-factorial
81		3^{24}	9^{4}	Fractional-factorial
81		3^{20}	9^{5}	Fractional-factorial
81		3^{16}	9^{6}	Fractional-factorial
81		3^{12}	9^{7}	Fractional-factorial
81		3^{8}	9^{8}	Fractional-factorial
81		3^{4}	9^{9}	Fractional-factorial
81			9^{10}	Fractional-factorial
82	2^{1}		41^{1}	Full-factorial
84	2^{83}			Hadamard
85			$5^{1}\ 17^{1}$	Full-factorial
86	2^{1}		43^{1}	Full-factorial
88	2^{87}			Hadamard
90	2^{1}	3^{2}	5^{1}	Full-factorial
92	2^{91}			Hadamard
93		3^{1}	31^{1}	Full-factorial
94	2^{1}		47^{1}	Full-factorial
96	2^{95}			Hadamard
96	2^{92}		4^{1}	Hedayat, Sloane & Stufken
96	2^{89}		4^{2}	Hedayat, Sloane & Stufken
96	2^{86}		4^{3}	Hedayat, Sloane & Stufken
96	2^{83}		4^{4}	Hedayat, Sloane & Stufken
96	2^{80}		4^{5}	Hedayat, Sloane & Stufken
96	2^{77}		4^{6}	Hedayat, Sloane & Stufken
96	2^{74}		4^{7}	Hedayat, Sloane & Stufken
98	2^{1}		49^{1}	Full-factorial
99		3^{2}	11^{1}	Full-factorial
100	2^{99}			Hadamard

Other relevant references include Rao (1947), Addelman (1962); Bose (1947); and Hadamard (1893). Most of these designs were created using information found in Hedayat, Sloane, and Stufken (1999); Dey (1985); and Neil Sloane's very useful web site: http://www.research.att.com/ njas/oadir/. Many of the designs above could have multiple references. Most of the above reference list was derived from Hedayat et. al.

When **n=**n is a multiple of 4, 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	420
424	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 $\mathrm{mod}(n/2, 4) = 2$
- n is a power of 2 (2, 4, 8, 16, ...) times the size of a smaller Hadamard matrix that is available.

For some of these sizes, 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 and $2^{74}\ 4^7$ in 96 runs).

Here is a simple example of using the **%MktEx** macro to request the L_{36} design, 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 get different names and levels.

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 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 will be reassigned (2, 1). Similarly, with three level factors, the mapping (1, 2, 3) will be 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 for some of the tabled designs, which have all ones in the first row.

%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. 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 tabled design, n=.
```

These messages tell you which type of tabled 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.

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 you specify that you do not want duplicate runs, but you also specify **options=accept**, which means you are willing to accept designs that violate the restrictions, the macro will tell you if the restrictions are not met.

```
NOTE: The restrictions were not met.
NOTE: The design has duplicate runs.
```

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.

Vacation Example

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		
.				
.				
.				

Vacation Example

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		
.				
.				
.				

Vacation Example

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 (perhaps externally, perhaps in a previous step) initial design. 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 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

Sometimes, 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.

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 four 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 (see page 328). 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=**. You can use the **%MktRuns** macro to get suggestions for values of **n=**.

Example:

```
%mktruns( 4 2 ** 5 3 ** 5 )
```

In this case, this macro suggests several sizes including an orthogonal design with **n=72** 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 allowed level-frequency range. The **balance=** option allows you to tell the macro that it should make an extra effort to ensure that the design is nearly balanced. By default, the macro does not try to ensure balance beyond the fact that lack of balance decreases D-efficiency. Specify a positive integer, usually 1 or 2, that specifies the degree of imbalance that is acceptable. You may need to also specify **options=accept** with **balance=**. The macro usually does a good job of producing nearly balanced design, but if balance is critically important, and your designs are not balanced enough, you can sometimes achieve better balance by specifying **balance=1**, but usually at the price of worse efficiency, sometimes much worse. The **balance=** option specifies additional restrictions (see **restrictions=**) that help achieve better balance. By default, no additional restrictions are added. The **balance=**n option specifies that for each factor, the difference between the frequencies, for 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. The macro needs the flexibility to have imbalance as it refines the design. Another option is to instead use the **%MktBal** macro, which produces perfectly balanced main effects plans. It is likely that the algorithms used by both the **balance=** option and the **%MktBal** macro will be changed in the future to use some now unknown algorithms that are both faster and better.

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**).

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.

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**.

partial= n

specifies a partial profile design. 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=**). Specifying **options=accept** or **balance=** with **partial=** is *not* a good idea.

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 1 or a larger value. Ideally, set **bad** to the number of violations so that the macro knows if changes to factor levels are moving in the right direction. The macro must consist of PROC IML statements and possibly some macro statements.

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.

Your macro can look at several things in quantifying badness, and must store its results in **bad**.

i - is a scalar that contains the number of the row currently being changed.

x - is a row vector of factor levels, always containing integer values beginning with 1 and continuing on to the number of levels for each factor.

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.

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.

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.

xmat - is the entire **x** matrix. Note that the *ith* row of **xmat** may not be **x** since **x** may contain information on the swaps being considered.

bad - results: 0 - fine, or the number of violations of restrictions.

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 five three-level factors and if it is bad if the level for 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.

$=$	equals	not: EQ
$\wedge =$ or $\neg =$	not equals	not: NE
$<$	less than	not: LT
$<=$	less than or equal to	not: LE
$>$	greater than	not: GT
$>=$	greater than or equal to	not: GE
&	and	not: AND
|	or	not: OR
$\wedge$ or $\neg$	not	not: NOT

Restrictions seriously 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 pages 195, 280, and 285 for more information on restrictions.

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. 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 across different operating systems and different SAS 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.

init= *SAS-data-set*

specifies the initial (input) experimental design. By default, there is no initial design. Use **`init=`** when you want to evaluate the efficiency of a design (along with **`options=check`**) or when you want to try to improve a design.

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 (next) 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.

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 **`out=Randomized`**. Random levels are assigned within factors, and the runs are sorted into a random order. 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.

Iteration Options

These next options control some of the details of the 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 swaps 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 a tabled 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 < 5186 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 tabled 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 .05**. Specify a zero or null 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 a tabled 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 **maxdesign=** 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 tabled 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.

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 much 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 a tabled initial design to using a random initial design. If the macro is not done using tabled 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. A null value, **maxtime=**, removes all time restrictions. 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 (next) for other ways to drastically cut run time for large problems.

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 .05`**. Specify a zero or null 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 a tabled 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.

tabiter= $n1 < n2 >$
specifies the number of times to try to improve a tabled or fractional-factorial initial design. The default is **`tabiter=10 200`**, which means 10 tries in the algorithm search iterations and 200 tries in the design search iterations.

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: $(1\ 2\ 3\ 4) \Rightarrow (1\ 2\ 3\ 3)$ or a two-level factor from a three-level factor: $(1\ 2\ 3) \Rightarrow (1\ 2\ 2)$. 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 this through level swaps).

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=5186 choose**. The default value was chosen, because 5186 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. The default is **exchange=1**, which means that the macro works with one factor at a time. You can specify **exchange=2** to do pair-wise exchanges. Pair-wise exchanges are *much* slower, but may produce better designs.

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.

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.

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
specifies which tabled (or 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 a tabled design that bests matches the specified design.

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.

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. 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. Most users will never need this option.

%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
  .   .   .   .   .
;
```

%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.

%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.

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

The **%MktLab** macro can be used to assign levels of -1 and 1, add an intercept, and change 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; 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 above.

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 Kuhfeld, Tobias, and Garratt (1994). 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 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 macro **%MktLab** takes by default the **data=randomized** data set from **%MktEx** and the **key=key** data set, and combines the information to create the **out=final** data set, which is shown below, 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 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)

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
1	0	1	0	0	1	0	0	1	-1	1	1
1	1	1	1	-1	-1	0	-1	-1	-1	0	-1
-1	0	1	-1	0	0	-1	-1	0	-1	0	0
0	0	0	1	-1	-1	-1	1	1	-1	1	0
0	-1	1	-1	1	-1	1	-1	1	-1	-1	1

This next step creates a design and blocks it. This example shows that it is OK 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)

proc print; id block run; by block; run;
```

Here are the results:

Block	Run	Brand	Price	Size	x4
1	1	Acme	$1.79	6	1
	2	Acme	$1.79	8	3
	3	Acme	$1.99	8	2
	4	Acme	$1.99	12	1
	5	Apex	$1.49	8	3
	6	Apex	$1.49	12	2
	7	Apex	$1.79	6	3
	8	Apex	$1.79	12	1
	9	Apex	$1.99	6	2
2	1	Acme	$1.49	6	2
	2	Acme	$1.49	8	1
	3	Acme	$1.49	12	3
	4	Acme	$1.79	12	2
	5	Acme	$1.99	6	3
	6	Apex	$1.49	6	1
	7	Apex	$1.79	8	2
	8	Apex	$1.99	8	1
	9	Apex	$1.99	12	3

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 the Vacation 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.

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: 1 to 10 or 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 generic choice models, for creating flag variables 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;
proc print label; run;
```

Obs	Sploosh	Plumbob	Platter	Moosey
1	$1.49	$1.49	$1.99	$2.49
2	$2.49	$1.99	$1.49	$2.49
3	$1.49	$1.99	$1.49	$1.99
4	$1.49	$1.99	$1.99	$1.49
.				
.				
.				

out= *SAS-data-set*

specifies the output data set with the final, recoded design. The default is **out=final**.

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	$1.49	$1.49	$1.99	$2.49
2	$2.49	$1.99	$1.49	$2.49
3	$1.49	$1.99	$1.49	$1.99
4	$1.49	$1.99	$1.99	$1.49
.				
.				
.				

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.

%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. 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 $nalts \times nsets$ 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. 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 $nalts \times nsets$ 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 contain 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.

stmts= 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,
          stmts=%str(price = input(put(price, price.), 5.);
                     format scene scene. lodge lodge.;))
```

%MktOrth Macro

The **%MktOrth** macro lists some of the 100% orthogonal main-effects plans that the **%MktEx** macro can generate, up through 100 runs. Here is a typical usage.

```
%mktorth;
```

The macro creates data sets and no printed output.

```
NOTE: The data set WORK.MKTDESLEV has 345 observations and 53 variables.
NOTE: The data set WORK.MKTDESCAT has 345 observations and 3 variables.
```

Here are the first few and last few designs in the marketing design catalogue (data set MKTDESCAT).

```
proc print data=mktdescat(where=(n le 12 or n ge 98)); run;
```

Obs	n	Design	Reference
1	4	2 ** 3	Hadamard
2	6	2 ** 1 3 ** 1	Full-factorial
3	8	2 ** 7	Hadamard
4	9	3 ** 4	Fractional-factorial
5	10	2 ** 1 5 ** 1	Full-factorial
6	12	2 ** 11	Hadamard
7	12	2 ** 4 3 ** 1	Hedayat, Sloane, and Stufken, 1999
8	12	2 ** 2 6 ** 1	Hedayat, Sloane, and Stufken, 1999
9	12	3 ** 1 4 ** 1	Full-factorial
343	98	2 ** 1 49 ** 1	Full-factorial
344	99	3 ** 2 11 ** 1	Full-factorial
345	100	2 ** 99	Hadamard

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**, ..., and the number of fifty-level factors is in **x50**. 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.

Say you are interested in the design $2^5 3^5 4^1$. Here are the ways in which it is available.

```
proc print data=mktdeslev(where=(x2 ge 5 and x3 ge 5 and x4 ge 1));
   var n design reference;
   run;
```

Obs	n	Design	Reference
289	72	2 ** 36 3 ** 13 4 ** 1	Hedayat, Sloane, and Stufken, 1999
294	72	2 ** 20 3 ** 24 4 ** 1	Wang, 1996
297	72	2 ** 13 3 ** 25 4 ** 1	Wang, 1996
299	72	2 ** 11 3 ** 24 4 ** 1 6 ** 1	Wang, 1996

Here is how you can see all the designs in a certain range of sizes.

```
proc print; where 12 le n le 20; run;
```

Obs	n	Design	Reference
6	12	2 ** 11	Hadamard
7	12	2 ** 4 3 ** 1	Hedayat, Sloane, and Stufken, 1999
8	12	2 ** 2 6 ** 1	Hedayat, Sloane, and Stufken, 1999
9	12	3 ** 1 4 ** 1	Full-factorial
10	14	2 ** 1 7 ** 1	Full-factorial
11	15	3 ** 1 5 ** 1	Full-factorial
12	16	2 ** 15	Hadamard
13	16	2 ** 12 4 ** 1	Fractional-factorial
14	16	2 ** 9 4 ** 2	Fractional-factorial
15	16	2 ** 6 4 ** 3	Fractional-factorial
16	16	2 ** 3 4 ** 4	Fractional-factorial
17	16	4 ** 5	Fractional-factorial
18	18	2 ** 1 3 ** 7	Taguchi, 1987
19	18	3 ** 6 6 ** 1	Taguchi, 1987
20	20	2 ** 19	Hadamard

```
21  20  2 **  8          5 **  1          Wang and Wu, 1992
22  20  2 **  2         10 **  1          Hedayat, Sloane, and Stufken, 1999
23  20                   4 **  1  5 **  1  Full-factorial
```

%MktOrth Macro Options

The following options can be used with the **%MktOrth** macro.

outall= *SAS-data-set*

specifies the output data set with all designs. This is like the **outlev=** data set, except larger. The **outall=** data set includes *all* of the **%MktEx** design catalogue, 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**. This data set is not created by default.

outcat= *SAS-data-set*

specifies the output data set with the catalogue 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 more variables: **x2** - number of two-level factors, **x3** - number of three-level factors and so on. The default is **outlev=MktDesLev**.

%MktRoll Macro

The **%MktRoll** autocall macro is used for manipulating the experimental design for choice experiments. It takes as input a SAS data set containing an experimental design with one row per choice set, 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 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.

```
%mktkey(x1-x5)
```

The **%MktKey** macro produced the following line.

```
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
;
```

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 design with one row per choice set:

```
%mktex(3 ** 5, n=12)
```

Here is how you can create the 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:

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)
```

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	1	2	1	1	2	2	2	1	2	1	1

then the data set SASUSER.DESIGN contains the rows:

29	8	A	2	1	2	1
30	8	B	1	2	2	2
31	8	C	1	2	1	1
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)
```

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

then the data set SASUSER.DESIGN contains the rows:

Obs	Set	Brand	Price	Size	Color	Shape	Pattern
29	8	A	1	2	2	1	2
30	8	B	1	2	1	2	.
31	8	C	2	1	2	1	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.

```
%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	1	2	2	1	2	1	1	2
30	8	B	1	2	1	2	.	1	1	2
31	8	C	2	1	2	1	2	1	1	2
32	8	None	.	.	.	.	.	1	1	2

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. 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=**.

notes
do not specify **options nonotes** during most of the macro.

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**.

%MktRuns Macro

The **%MktRuns** autocall macro suggests reasonable sizes for main-effects experimental designs. It 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                    Cannot Be
   Design Sizes        Violations   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                        Suen, 1989
36    2 ** 11  3 ** 12                        Taguchi, 1987
36    2 **  4  3 ** 13                        Taguchi, 1987
72    2 ** 49  3 **  4                        Hedayat, Sloane, and Stufken, 1999
72    2 ** 47  3 ** 12                        Wang, 1996
72    2 ** 40  3 ** 13                        Wang and Wu, 1991
72    2 ** 38  3 ** 12  6 **  1               Hedayat, Sloane, and Stufken, 1999
72    2 ** 37  3 **  8  6 **  2               Hedayat, Sloane, and Stufken, 1999
72    2 ** 36  3 ** 13  4 **  1               Hedayat, Sloane, and Stufken, 1999
72    2 ** 36  3 ** 12 12 **  1               Hedayat, Sloane, and Stufken, 1999
72    2 ** 36  3 **  7  6 **  3               Hedayat, Sloane, and Stufken, 1999
72    2 ** 23  3 ** 24                        Dey, 1985
72    2 ** 20  3 ** 24  4 **  1               Wang, 1996
72    2 ** 16  3 ** 25                        Wang, 1996
72    2 ** 14  3 ** 24  6 **  1               Wang, 1996
72    2 ** 13  3 ** 25  4 **  1               Wang, 1996
72    2 ** 12  3 ** 24 12 **  1               Hedayat, Sloane, and Stufken, 1999
72    2 ** 11  3 ** 24  4 **  1  6 **  1      Wang, 1996
```

The macro reports that the saturated design has 12 runs and that 36 is an optimal design size. 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                    Cannot Be
   Design Sizes      Violations    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. The smallest design that is considered is the saturated design. Usually you will not need to specify the **max=** option. For example, this specification tries 5000 sizes (21 to 5020) and reports that a perfect design can be found with 3600 runs.

```
%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                    Cannot Be
   Design Sizes      Violations    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. Now, instead of specifying **`%mktruns( 2 2 2 3 3 3 3 )`**, we replace two of the 2's with a 4.

```
%mktruns( 2 4 3 3 3 3 )
```

```
                        Design Summary

                       Number of
                       Levels       Frequency

                            2               1
                            3               4
                            4               1

                Saturated      = 13
                Full Factorial = 648

                Some Reasonable                    Cannot Be
                   Design Sizes        Violations   Divided By

                            72 *                0
                           144                  0
                            36                  1     8
                           108                  1     8
                            18                  6     4  8 12
                            24                  6     9
                            48                  6     9
                            54                  6     4  8 12
                            90                  6     4  8 12
                            96                  6     9

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

 n                  Design                  Reference

72    2 ** 36  3 ** 13  4 **  1            Hedayat, Sloane, and Stufken, 1999
72    2 ** 20  3 ** 24  4 **  1            Wang, 1996
72    2 ** 13  3 ** 25  4 **  1            Wang, 1996
72    2 ** 11  3 ** 24  4 **  1  6 **  1   Wang, 1996
```

Now we need 72 runs for perfect balance and orthogonality and there are six violations in 18 runs (4, 4×2, 4×3, 4×3, 4×3, and 4×3).

If you ever get errors running this macro, like invalid page errors, see "Macro Errors" on page 288.

%MktRuns Macro Options

The following options can be used with the **%MktRuns** macro. 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 four 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 design size to evaluate. By default, this option is not specified, and the **max=** option specification provides a range of design sizes to evaluate.

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.

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.

out= *SAS-data-set*

specifies the name of a SAS data set with the suggested sizes. The default is **out=nums**.

%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 288. 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 368.

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 (**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.

Concluding Remarks

This report has illustrated how to design a choice experiment; prepare the questionnaire; input, process, and code the design; perform the analysis; and interpret the results. All examples were artificial. We would welcome any real data sets that we could use in future examples. This report 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. Please direct questions to the technical support division. For more information on discrete choice, see Carson et. al. (1994) and the papers they reference. For information on designing experiments for discrete choice, see Lazari and Anderson (1994), and see Kuhfeld, Tobias, and Garratt (1994) on page 25.

I hope you like the new macros. In particular, I hope you find the new **`%MktEx`** macro to be very powerful and useful. My goal in writing this book is to help you do better research and do it more quickly and more easily. I would like to hear what you think.

For Those Who Like a Challenge
What do most of the random number seeds used in the **Multinomial Logit, Discrete Choice Modeling** report and all of the seeds used in the **Conjoint Analysis Examples** report 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. Hints: Ignore seed 7654321; it has nothing in common with the others. Seeds 201 and 155 almost but not quite fit with the others. Seeds 446, 538, and 543 are part of a still larger group. Answers like “they are all less than 619,” while true, are not what I have in mind.

References

Addelman, S. (1962), "Orthogonal Main-Effects Plans for Asymmetrical Factorial Experiments", *Technometrics*, 4, 21−46.

Bose, R.C. (1947), "Mathematical Theory of the Symmetrical Factorial Design", *Sankhya*, 8, 107−166.

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.

Cook, R.D. and Nachtsheim, C.J. (1980), "A Comparison of Algorithms for Constructing Exact D-optimal Designs", *Technometrics*, 22, 315−324.

Dey, A. (1985), *Orthogonal Fractional Factorial Designs*, New York: Wiley.

Fedorov, V.V. (1972), *Theory of Optimal Experiments*, translated and edited by W.J. Studden and E.M. Klimko, New York: Academic Press.

Finney, D.J. (1982), "Some Enumerations for the 6x6 Latin Squares", *Utilitas Mathematics*, 21, 137−153.

Hadamard, J. (1893), "Resolution d'une question relative aux determinants",*Bull. des Sciences Math*, (2), 17, 240−246.

Hedayat, A.S., Sloane, N.J.A., and Stufken, J. (1999), *Orthogonal Arrays*, New York: Springer.

Huber, J., and Zwerina, K. (1996), "The Importance of Utility Balance in Efficient Choice Designs," *Journal of Marketing Research*, 33, 307−317.

Kuhfeld, W.F., Tobias, R.D., and Garratt, M. (1994), "Efficient Experimental Design with Marketing Research Applications," *Journal of Marketing Research*, 31, 545−557.

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.

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), 350−67.

Manski, C.F., and McFadden, D. (1981), *Structural Analysis of Discrete Data with Econometric Applications*. Cambridge: MIT Press.

Meyer, R.K., and Nachtsheim, C.J. (1995), "The Coordinate-Exchange Algorithm for Constructing Exact Optimal Experimental Designs", *Technometrics*, 37, 60−69.

Paley, R.E.A.C (1933), On Orthogonal Matrices, J. Math. Phys, 12, 311−320.

Rao, C.R. (1947), "Factorial Experiments Derivable from Combinatorial Arrangements of Arrays", *Journal of the Royal Statistical Society*, Suppl., 9, 128−139.

Suen, C.-Y. (1989), "Some Resolvable Orthogonal Arrays with Two Symbols", *Communications in Statistics, Theory and Methods*, 18, 3875−3881.

Suen, C.-Y. (1989), "A Class of Orthogonal Main Effects Plans", *Journal of Statistical Planning and Inference*, 21, 391−394.

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.

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.

Wang, J.C., (1996), "Mixed Difference Matrices and the Construction of Orthogonal Arrays", *Statist. Probab. Lett.*, 28, 121−126.

Wang, J.C., (1996), *A Recursive Construction of Orthogonal Arrays*, Preprint.

Williamson, J. (1944), "Hadamard's Determinant Theorem and the Sum of Four Squares", Duke Math. J., 11, 65−81.

Zhang, Y.S., Lu, Y., and Pang, S., (1999), "Orthogonal Arrays Obtained by Orthogonal Decompositions of Projection Matrices", *Statistica Sinica*, 9, 595−604.

Multinomial Logit Models*

Ying So and Warren F. Kuhfeld

SAS, Cary, NC

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).

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 Days 1980). The conditional logit model was also introduced by McFadden (1973) 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.

*This paper was presented at SUGI 20 by Ying So and can also be found in the SUGI 20 proceedings.

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

$$\Pi_{jk} = \frac{\exp(\boldsymbol{\beta}_k'\mathbf{X}_j)}{\sum_{l=1}^{m}\exp(\boldsymbol{\beta}_l'\mathbf{X}_j)} = \frac{1}{\sum_{l=1}^{m}\exp[(\boldsymbol{\beta}_l - \boldsymbol{\beta}_k)'\mathbf{X}_j)]}$$

$\beta_1, \ldots, \beta_m$ 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, $\beta_m = 0$), the coefficients β_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

$$\Pi_{jk} = \frac{\exp(\boldsymbol{\beta}_k'\mathbf{X}_j + \boldsymbol{\theta}'\mathbf{Z}_{jk})}{\sum_{l=1}^{m} \exp(\boldsymbol{\beta}_l'\mathbf{X}_j + \boldsymbol{\theta}'\mathbf{Z}_{jl})}$$

$\beta_1, \ldots, \beta_{m-1}$ and $\beta_m \equiv 0$ 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 Technical Report P-229). 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:

$$\boldsymbol{\theta} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_{m-1} \end{bmatrix} \quad \mathbf{Z}_{j1} = \begin{bmatrix} \mathbf{X}_j \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \mathbf{Z}_{j2} = \begin{bmatrix} 0 \\ \mathbf{X}_j \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \cdots \quad \mathbf{Z}_{j,m-1} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \mathbf{X}_j \end{bmatrix} \quad \mathbf{Z}_{jm} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

where the 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 179, 185, 192, and 383 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

Reduced Model without Cross-Alternative Effects

The PHREG Procedure

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 179, 185, 192, and 379 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).

References

Agresti, A. (1990) *Categorical Data Analysis*. New York: John Wiley & Sons.

Breslow, N. and Day, N.E. (1980), *Statistical Methods in Cancer Research, Vol. II: The Design and Analysis of Cohort Studies,* Lyon: IARC.

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.

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.

Hoffman, S.D. and Duncan, G.J. (1988), "Multinomial and conditional logit discrete-choice models in Demography," *Demography,* 25 (3), 415-427.

Kuhfeld, W.F., Tobias, R.D., and Garratt, M. (1994), "Efficient Experimental Design with Marketing Research Applications," *Journal of Marketing Research*, 31, 545-557.

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.

McFadden, D. (1973), "Conditional logit analysis of qualitative choice behavior," in P. Zarembka (Ed.) *Frontiers in Econometrics,* New York: Academic Press, Inc.

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.

SAS Institute Inc. (1989), *SAS/STAT User's Guide, Vol. 1, Version 6, Fourth Edition,* Cary: NC: SAS Institute Inc.

SAS Institute Inc. (1992), SAS Technical Report P-229. *SAS/STAT Software: Changes and Enhancements, Release 6.07,* Cary, NC: SAS Institute Inc.

Index