TS-621

Multinomial Logit, Discrete Choice Modeling

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

Warren F. Kuhfeld
SAS Institute Inc.

January 1, 2000

Contents

Multinomial Logit, Discrete Choice Modeling

This report shows 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). The purpose of this report is to illustrate 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 the January 1, 2000 edition, and it is a major revision of the May 1996 report and other earlier reports. This report uses macros and features of the SAS System that are new in Version 8, whereas the May 1996 and earlier reports were written for Version 6 of the SAS System. This report is available as a PDF file from the Technical Support web site as http://ftp.sas.com/techsup/download/technote/ts621.pdf. It is also available via anonymous FTP from ftp.sas.com, file techsup/download/technote/ts621.pdf. SAS code examples are available via WWW by connecting to http://www.sas.com/techsup/download/stat/ and getting mlogit8.sas; via anonymous ftp, from ftp.sas.com, get techsup/download/stat/mlogit8.sas. This information is provided by SAS Institute Inc 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.

If you are familiar with the May 1996 or earlier editions of this report, you will see several important differences. Much of our work is now done with autocall macros. See page 261 for more information on autocall macros.

- We now use the autocall macro %MKTDES to generate most of our experimental designs. It is easier to use and usually produces better results than the methods suggested in earlier reports. The macro is documented starting on page 261 of this report.
- We use the %MKTRUNS autocall macro to suggest design sizes. See page 268 for documentation.
- We use the %CHOICEFF autocall macro to generate certain specialized choice designs. See page 270 for documentation.
- We use the autocall macros %MKTROLL, %MKTMERGE, and %MKTALLO to prepare the data and design for analysis. These macros are not part of the autocall library for Version 8.0 but will be for Version 8.10 and subsequent releases of the SAS System. For Version 8.0, you can obtain these macros by writing saswfk@wnt.sas.com or by getting the code as described previously. See pages 281, 285, and 286 for documentation.
- We use PROC TRANSREG to do all of our design coding. With Version 7 and Version 8 of the SAS® System, PROC TRANSREG has new options and long names and labels, which makes it well suited for coding choice models.
- We use the autocall macro %PHCHOICE to customize our printed output. This macro uses PROC TEMPLATE and ODS (Output Delivery System) to customize the output of PROC PHREG, which fits the multinomial logit model. See page 288 for documentation.

Several examples are discussed including some new ones.*

- 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, 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.

*All of the sample data sets are artificially generated.

- The first vacation example is a larger, symmetric example that discusses designing the choice experiment, how PROC OPTEX works, 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, coding down, pseudo-factors, using a tabled design as a candidate set, evaluating the efficiency of a given design, 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, 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 next section contains miscellaneous examples including designs with many factors, improving an existing design, when some choice sets are fixed in advance, six-level factors, and ten-level factors.

This report 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.

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 73 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 variables that have two or more fixed values, or *levels*. Experiments are performed to study the effects of the factor levels on the dependent variable. In a 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 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	Brand 2	Brand3
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

The most obvious example of an experimental design is the *full-factorial design*, which consists of all possible combinations of the levels of the factors. For example, with five factors, two at 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 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 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. 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 $e+1$. For example, for resolution III designs, all main effects are estimable free of each other, but some of them are confounded with two-factor interactions. For resolution V designs, all main effects and two-factor interactions are estimable free of each other. Higher resolutions require larger designs. Orthogonal arrays come in specific numbers of runs (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 used in practice, is imprecise. It refers to designs that are both orthogonal and balanced, and hence optimal. It also refers to designs that are orthogonal but not balanced, and hence potentially nonoptimal. A design is *balanced* when each level occurs equally often within each factor, which means the intercept is orthogonal to each effect. Imbalance is a generalized form of nonorthogonality, which increases the variances of the parameter estimates.

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 $\boldsymbol{\beta}$ in a least-squares analysis is proportional to $(\mathbf{X}'\mathbf{X})^{-1}$. An efficient design will have a "small" variance matrix, and the eigenvalues of $(\mathbf{X}'\mathbf{X})^{-1}$ provide measures of its "size." The two most prominent efficiency measures are based on the idea of quantifying 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 (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 (for a suitably coded $\mathbf{X}$):

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

These efficiencies measure the goodness of the design relative to hypothetical orthogonal designs that may be far from possible, so they are not useful as absolute measures of design efficiency. Instead, they should be used relatively, to compare one design to another for the same situation. Efficiencies that are not near 100 may be perfectly satisfactory. Throughout this report, we will use the %MKTDES macro and PROC OPTEX 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 variances of the parameter estimates $\hat{\boldsymbol{\beta}}$ are proportional to $(\mathbf{X}'\mathbf{X})^{-1}$. In contrast, the variances of the parameter estimates $\hat{\boldsymbol{\beta}}$ 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 might 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.

Full-Profile Conjoint Design	
Brand	Price
1	1.99
1	2.99
2	1.99
2	2.99
3	1.99
3	2.99

For the same problem, we might use linear model theory to create a "linear" design from which we will construct a choice design. This design has three factors: brand 1 price, brand 2 price, and brand 3 price.

Linear Design		
Brand 1	Brand 2	Brand3
1.99	1.99	1.99
1.99	2.99	2.99
2.99	1.99	2.99
2.99	2.99	1.99

When we fit the choice model, we will construct a choice design from the linear design that looks quite different. See the left panel of the next table. When we code the design, it could look something like the right panel.

Choice Design		Choice Design Coding					
					Brand 1	Brand 2	Brand 3
Brand	Price	Brand 1	Brand 2	Brand 3	Price	Price	Price
1	1.99	1	0	0	1.99	0.00	0.00
2	1.99	0	1	0	0.00	1.99	0.00
3	1.99	0	0	1	0.00	0.00	1.99
1	1.99	1	0	0	1.99	0.00	0.00
2	2.99	0	1	0	0.00	2.99	0.00
3	2.99	0	0	1	0.00	0.00	2.99
1	2.99	1	0	0	2.99	0.00	0.00
2	1.99	0	1	0	0.00	1.99	0.00
3	2.99	0	0	1	0.00	0.00	2.99
1	2.99	1	0	0	2.99	0.00	0.00
2	2.99	0	1	0	0.00	2.99	0.00
3	1.99	0	0	1	0.00	0.00	1.99

Each group of three rows in the choice design forms one choice set. The linear design has one factor for each attribute of each alternative (or brand), and brand is not a factor in the linear design. Brand is a "bin" into which the other factors are collected. In the choice design, brand and price are both factors, but they have 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 do all these things many

times in the examples. The point now is to notice that the design matrix for a linear model is different from the design matrix for a choice model. They aren't even the same size!

We make a good design for a linear model by picking our **x**'s to 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 239−249, 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. 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 array the design in the linear fashion (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. The design techniques discussed in this book are based on this assumption and have been used quite successfully in the field for many years.

In most of the examples, we will use the %MKTDES macro or PROC OPTEX 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 %MKTDES macro or PROC OPTEX 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 261 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 of 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 288 for more information on the %PHCHOICE macro.

Candy Example

We begin with a very simple example. 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. Then 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

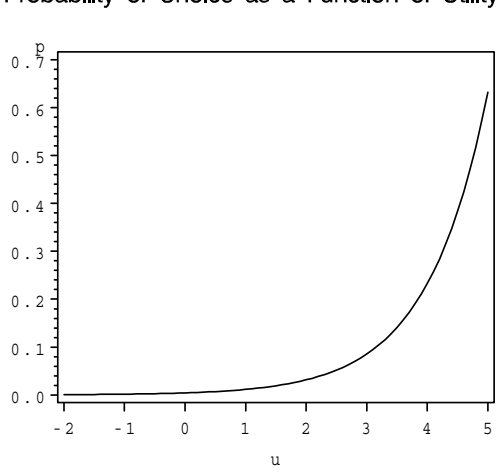
p(Milk, Chewy, No Nuts)	=	exp(0)	/ 234.707	=	0.004
p(Milk, Chewy, Nuts)	=	exp(1)	/ 234.707	=	0.012
p(Milk, Soft, No Nuts)	=	exp(-2)	/ 234.707	=	0.001
p(Milk, Soft, Nuts)	=	exp(-1)	/ 234.707	=	0.002
p(Dark, Chewy, No Nuts)	=	exp(4)	/ 234.707	=	0.233
p(Dark, Chewy, Nuts)	=	exp(5)	/ 234.707	=	0.632
p(Dark, 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 '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 more data may follow on the same line.

```
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.

```
* 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 number.     */
   Subj = _n_; Set = 1;            /* Store subject, choice set number. */
   do i = 1 to 8;                  /* Loop over alternatives.           */
      c = 2 - (i eq choice);       /* Designate chosen alternative.     */
      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 in 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. So 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. The entire eight observations in the COMBOS data set is 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 further 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 the SAS System, 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 are 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 exceed 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 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 with 1 equals choice and 2 equals unchosen, 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.† The **label** statement is added since we are using a template that assumes each variable has a label.

Multinomial Logit Model Results

The output is shown next. Recall that we used **%phchoice(on)** on page 71 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

*This 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.

†Do not specify any other **ties=** options; **ties=breslow** specifies the most efficient and always appropriate way to fit the multinomial logit model.

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 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 degrees of freedom 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 degree of freedom. 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. So for example, 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, creating the questionnaire, inputting the raw data, creating the data set for analysis, and fitting the discrete choice model. We assume the reader is familiar with the experimental design issues discussed in Kuhfeld, Tobias, and Garratt (1994). Some of these concepts are reviewed starting on page 68.

Set Up

The study involves four fictitious fabric softener brand names *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 261. 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.

```
title 'Choice of Fabric Softener';

%mktruns( 3 3 3 3 )
```

The output tells us the size of the saturated design, which is the number of parameters in the linear design, and suggests design sizes.

Choice of Fabric Softener

Some Reasonable Design Sizes (Saturated=9)	Violations	Cannot Be Divided By
9	0	
18	0	
27	0	
36	0	
45	0	
54	0	
63	0	
72	0	
81	0	
90	0	

The output from this macro tells us that the saturated design has nine runs. This is shown by the "(Saturated=9)" in the listing. It also tells us that 9, 18, 27, ..., 90 are optimal design sizes with zero violations. There are zero violations because all of these sizes can be divided by 3 and $3 \times 3 = 9$. In this problem, the %MKTRUNS macro reports ten different sizes with no violations.† 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

*Of course real studies would use real brands. Since we have not collected real data, we cannot use real brand names.

†For more realistic problems we will see violations. For example, for this problem, a sample size of 12 is considered, but it has 6 violations. Six times, $the 4(4-1)/2 = 6$ pairs of the four threes, 12 cannot be divided by $3 \times 3 = 9$.

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 in a format. Our design will have values for price of 1, 2, and 3. We use a format to map these values to 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 alternatives 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 %MKTDES to create most of our designs. (All of the autocall macros used in this report are documented starting on page 261.) When you invoke the %MKTDES macro for a simple problem, you only need to specify the factors, number of levels, and number of runs. The macro does the rest. The macro has two primary steps: it first creates a candidate set of potential choice sets using either PROC PLAN or PROC FACTEX, then it uses PROC OPTEX to construct the design by selecting a good subset of the potential choice sets. PROC PLAN generates full-factorial candidate sets, PROC FACTEX generates fractional-factorial candidate sets, and PROC OPTEX searches the candidate set for an optimal design. The macro displays the code it generates so you can better understand what it is doing for you. Here is the %MKTDES macro usage for this example:

```
%mktdes(factors=x1-x4=3, n=&n)
```

This example has four factors, **x1** through **x4** all with three levels. A design with 18 runs is requested. The **factors=** option lists the factors followed by an equal sign and the number of levels. 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 you can reproduce these results.* In practice, particularly for small problems such as this, specifying a seed is not necessary. The **procopts=** option is used to specify options for the PROC OPTEX statement, in this case the **seed=** option. Here is the macro usage with the random number seed specified:

```
%mktdes(factors=x1-x4=3, n=&n, procopts=seed=7654321)

proc print; run;
```

Here are the results.

*However, due to machine differences, some results may not be exactly reproducible on any particular machine, though at most the differences should be slight.

Choice of Fabric Softener

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	100.0000	100.0000	100.0000	0.7071
2	100.0000	100.0000	100.0000	0.7071
3	100.0000	100.0000	100.0000	0.7071
4	100.0000	100.0000	100.0000	0.7071
5	100.0000	100.0000	100.0000	0.7071

Choice of Fabric Softener

Obs	x1	x2	x3	x4
1	3	3	2	3
2	3	3	2	2
3	3	2	3	3
4	3	2	1	1
5	3	1	3	2
6	3	1	1	1
7	2	3	3	1
8	2	3	1	3
9	2	2	2	2
10	2	2	2	1
11	2	1	3	2
12	2	1	1	3
13	1	3	3	1
14	1	3	1	2
15	1	2	3	3
16	1	2	1	2
17	1	1	2	3
18	1	1	2	1

More will be said about these results starting on page 89. For now, notice that 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.

Examining the Design

It is good to run basic checks on all designs. The following statements use PROC SUMMARY and PROC PRINT to display all one-way frequencies for all attributes, all two-way frequencies, and all n-way frequencies (in this case four-way) frequencies. What we hope to see is equal or at least nearly equal one-way and two-way frequencies, and we want to see that each combination occurs only once.

```
proc summary data=design;
   class _all_;
   ways 1 2 4;
   output out=sum;
   run;

proc print; by _type_; run;
```

Choice of Fabric Softener

TYPE=1

Obs	x1	x2	x3	x4	_FREQ_
1	.	.	.	1	6
2	.	.	.	2	6
3	.	.	.	3	6

TYPE=2

Obs	x1	x2	x3	x4	_FREQ_
4	.	.	1	.	6
5	.	.	2	.	6
6	.	.	3	.	6

TYPE=3

Obs	x1	x2	x3	x4	_FREQ_
7	.	.	1	1	2
8	.	.	1	2	2
9	.	.	1	3	2
10	.	.	2	1	2
11	.	.	2	2	2
12	.	.	2	3	2
13	.	.	3	1	2
14	.	.	3	2	2
15	.	.	3	3	2

TYPE=4

Obs	x1	x2	x3	x4	_FREQ_
16	.	1	.	.	6
17	.	2	.	.	6
18	.	3	.	.	6

TYPE=5

Obs	x1	x2	x3	x4	_FREQ_
19	.	1	.	1	2
20	.	1	.	2	2
21	.	1	.	3	2
22	.	2	.	1	2
23	.	2	.	2	2
24	.	2	.	3	2
25	.	3	.	1	2
26	.	3	.	2	2
27	.	3	.	3	2

---------- _TYPE_=6 ----------

Obs	x1	x2	x3	x4	_FREQ_
28	.	1	1	.	2
29	.	1	2	.	2
30	.	1	3	.	2
31	.	2	1	.	2
32	.	2	2	.	2
33	.	2	3	.	2
34	.	3	1	.	2
35	.	3	2	.	2
36	.	3	3	.	2

---------- _TYPE_=8 ----------

Obs	x1	x2	x3	x4	_FREQ_
37	1	.	.	.	6
38	2	.	.	.	6
39	3	.	.	.	6

---------- _TYPE_=9 ----------

Obs	x1	x2	x3	x4	_FREQ_
40	1	.	.	1	2
41	1	.	.	2	2
42	1	.	.	3	2
43	2	.	.	1	2
44	2	.	.	2	2
45	2	.	.	3	2
46	3	.	.	1	2
47	3	.	.	2	2
48	3	.	.	3	2

---------- _TYPE_=10 ----------

Obs	x1	x2	x3	x4	_FREQ_
49	1	.	1	.	2
50	1	.	2	.	2
51	1	.	3	.	2
52	2	.	1	.	2
53	2	.	2	.	2
54	2	.	3	.	2
55	3	.	1	.	2
56	3	.	2	.	2
57	3	.	3	.	2

---------- _TYPE_=12 ----------

Obs	x1	x2	x3	x4	_FREQ_
58	1	1	.	.	2
59	1	2	.	.	2
60	1	3	.	.	2
61	2	1	.	.	2
62	2	2	.	.	2
63	2	3	.	.	2
64	3	1	.	.	2
65	3	2	.	.	2
66	3	3	.	.	2

```
---------------------------------- _TYPE_=15 ----------------------------------

                    Obs    x1    x2    x3    x4    _FREQ_

                     67     1     1     2     1       1
                     68     1     1     2     3       1
                     69     1     2     1     2       1
                     70     1     2     3     3       1
                     71     1     3     1     2       1
                     72     1     3     3     1       1
                     73     2     1     1     3       1
                     74     2     1     3     2       1
                     75     2     2     2     1       1
                     76     2     2     2     2       1
                     77     2     3     1     3       1
                     78     2     3     3     1       1
                     79     3     1     1     1       1
                     80     3     1     3     2       1
                     81     3     2     1     1       1
                     82     3     2     3     3       1
                     83     3     3     2     2       1
                     84     3     3     2     3       1
```

This design is perfect. However, there are other 100% efficient designs with duplicate observations that can be produced with different seeds. The last part of the output, the n-way frequencies contains some 2's for those designs. Sometimes simply changing the seed results in a better design.

Understanding the %MKTDES Macro

For simple problems such as this, the macro can find an optimal design quite easily, but for more complicated problems you may have to try more than one approach to find a good design. For this reason, it is important to understand the steps the macro takes in creating an efficient design. To help you better understand what the macro is doing, it shows you some of the code it generates. For this problem, the macro first uses PROC PLAN to create a full-factorial design with 4 three-level factors (four brands each at three prices). Here is the PROC PLAN step that the %MKTDES macro generated.

```
proc plan ordered;
   factors
      x1=3
      x2=3
      x3=3
      x4=3
      / noprint;
   output out=Cand1;
   run; quit;
```

The **factors** statement specifies that **x1** has 3 levels, **x2** has 3 levels, **x3** has 3 levels, and **x4** has 3 levels. The full-factorial design consists of all possible $3 \times 3 \times 3 \times 3 = 81$ combinations of the factor levels, the first 10 of which are shown. The **ordered** option specifies that the design is not to be randomized – the choice sets are not sorted into a random order yet. The **output** statement outputs the design to a SAS data set, CAND1.

```
proc print data=Cand1(obs=10);
   title2 'The First 10 Observations of the Full-Factorial Design';
   run; quit;
```

Here are the first ten observations of the full-factorial design.

Choice of Fabric Softener
The First 10 Observations of the Full-Factorial Design

Obs	x1	x2	x3	x4
1	1	1	1	1
2	1	1	1	2
3	1	1	1	3
4	1	1	2	1
5	1	1	2	2
6	1	1	2	3
7	1	1	3	1
8	1	1	3	2
9	1	1	3	3
10	1	2	1	1

Each row of this design is a potential choice set. The next step shows the first ten potential choice sets by applying formats and labels.

```
proc print data=Cand1(obs=10) label;
   title2 'Ten Potential Choice Sets';
   format x1-x4 price.;
   label x1 = 'Sploosh' x2 = 'Plumbbob' x3 = 'Platter' x4 = 'Moosey';
   run; quit;
```

Choice of Fabric Softener
Ten Potential Choice Sets

Obs	Sploosh	Plumbbob	Platter	Moosey
1	$1.49	$1.49	$1.49	$1.49
2	$1.49	$1.49	$1.49	$1.99
3	$1.49	$1.49	$1.49	$2.49
4	$1.49	$1.49	$1.99	$1.49
5	$1.49	$1.49	$1.99	$1.99
6	$1.49	$1.49	$1.99	$2.49
7	$1.49	$1.49	$2.49	$1.49
8	$1.49	$1.49	$2.49	$1.99
9	$1.49	$1.49	$2.49	$2.49
10	$1.49	$1.99	$1.49	$1.49

So for example, the first potential choice set consists of the four brands, all at $1.49, and a constant alternative which is not shown. The eighth choice set consists of *Sploosh* at $1.49, *Plumbob* at $1.49, *Platter* at $2.49, *Moosey* at $1.99, and the constant alternative.

Eighty-one choice sets are too many to get reliable data, so a subset of the choice sets must be selected. The full-factorial design is used as a candidate set of points from which the final design is chosen. The macro calls PROC OPTEX to create an efficient design.* This specification asks for an efficient main-effects design for 4 three-level factors in 18 runs (18 choice sets). PROC OPTEX independently generates ten designs, each time trying to optimize the D-efficiency criterion, which is a measure of design goodness. The best five designs are kept, and the designs are then ordered from most to least efficient. The **output** statement outputs the most efficient design to a SAS data set. Here is the PROC OPTEX step that the %MKTDES macro generated.

*See the section "Candidate Sets and How PROC OPTEX Works" on page 111, the vacation example on page 109, and subsequent examples for more information on PROC OPTEX.

```
title 'Choice of Fabric Softener';

proc optex data=Cand1 seed=7654321;
   class x1-x4 / param=orthref;
   model x1-x4;
   generate n=18 iter=10 keep=5 method=m_federov;
   output out=Design;
   run; quit;
```

Here are the results.

Choice of Fabric Softener

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	100.0000	100.0000	100.0000	0.7071
2	100.0000	100.0000	100.0000	0.7071
3	100.0000	100.0000	100.0000	0.7071
4	100.0000	100.0000	100.0000	0.7071
5	100.0000	100.0000	100.0000	0.7071

In this (easy) case, PROC OPTEX has little trouble finding a design with perfect, 100% efficiency. Small examples such as this should run in at most a few seconds on most modern computers. In this example, the new Version 8 option **param=orthref** was specified in the **class** statement with PROC OPTEX. This option generates an orthogonal coding for the design matrix so that D-efficiency is scaled to range from 0 to 100. The sometimes complicated strategies that were needed to get a 0 to 100 scale using previous releases of the SAS System are no longer needed. PROC OPTEX options are explained in more detail starting on page 109.

Randomizing the Design, Postprocessing

It is a good idea to randomize the design, that is to present the choice sets in random order. We can use PROC PLAN to generate the integers in the range 1 to 18 sorted into random order to sort the design. The DATA step reads the ORDER data set, which contains the variable **Set**, and then uses **point=** to directly access the appropriate observation from SASUSER.DES. For example, the first value of **Set** is 16, so the sixteenth observation of SASUSER.DES is read first. Each observation of SASUSER.DES is read once in a random order. Also in the next steps, the constant *Another* alternative is created and formats assigned.

```
proc plan seed=7654321;          /* random order to sort design       */
   factors set=&n / noprint;     /* integers, 1 - &n in random order  */
   output out=order;             /* output to data set order          */
   run; quit;

data sasuser.des;                /* create randomly sorted design     */
   set order;                    /* read order of observations        */
   set design point=set;         /* sort into random order            */
   format x1-x&mm1 price.;       /* assign formats, labels            */
   label x1 = 'Sploosh' x2 = 'Plumbbob' x3 = 'Platter' x4 = 'Moosey';
   run;
```

This is the final design. Note that it is stored in a permanent SAS data set so that it will still be available after the data are collected.

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

Choice of Fabric Softener
Efficient Design

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

These previous steps could be simplified. Later, we will need to use PROC PLAN and the **point=** option in more complicated examples, so we chose to illustrate them here in a simpler problem first. Here is a simpler alternative to the previous two steps. Note that the final order for these two approaches are not the same, even with the same seeds.

```
data temp;                          /* create randomly sorted design     */
   set design;                      /* read current design               */
   format x1-x&mm1 price.;          /* assign formats, labels            */
   label x1 = 'Sploosh' x2 = 'Plumbbob' x3 = 'Platter' x4 = 'Moosey';
   r = uniform(7654321);            /* random variable to sort on        */
   run;

proc sort;
   by r;
   run;
```

Generating the Questionnaire

A questionnaire based on the design is printed using the DATA step. The statement **array brands[&mm] $ _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 $1.49.

   2) Plumbbob brand at $1.99.

   3) Platter brand at $1.49.

   4) Moosey brand at $1.99.

   5) Another brand at $1.99.

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

   1) Sploosh brand at $1.99.

   2) Plumbbob brand at $1.99.

   3) Platter brand at $1.99.

   4) Moosey brand at $1.99.

   5) Another brand at $1.99.
```

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 $1.49) 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.

```
data results;                    /* read choice data set              */
   input Subj (choose1-choose&n) (1.) @@;
   datalines;
  1 333542334333314443  2 333212344333333345  3 333212333333313333
  4 133242144334414453  5 335242134333513443  6 333242234333314443
  7 333432334332323443  8 333242234334414443  9 333432331352313343
 10 325222235332333443 11 333232334333313343 12 333242234333313453
 13 533212334332213443 14 142242144333213443 15 333222335333313345
 16 333434235333315343 17 533242234352313443 18 343445534332414543
 19 333342335332313443 20 333242234332315543 21 333252534333513443
 22 333242354333313543 23 333242333333313443 24 525222234332223443
 25 353342234333213343 26 333245545332313443 27 333352534333353343
 28 333232334333333343 29 333422534335353443 30 333252334533313443
 31 353342334332313443 32 353222234333334443 33 333222234352313345
 34 332244134333313443 35 343552234353413445 36 333244534333313443
 37 333244234334514443 38 353232334333353543 39 333252334333313543
 40 343234134332413343 41 333444244432413443 42 333232234332314443
 43 333242254333333443 44 333242234332313443 45 312252544432414443
 46 132242235433514443 47 543242534332413443 48 335452334333323453
 49 333542134334313443 50 333222334332314443
;
```

Processing the Data

Our next step is to prepare the experimental design for analysis. Our design, stored in the data set SASUSER.DES, is stored with one row per choice set. While this is convenient for generating the questionaire, it is not the right form for analysis. For analysis, we need a design with one row for each alternative of each choice set. We will use the macro %MKTROLL to "roll out" the design into the proper form. First, we must create a data set that describes how the design is to be processed. The next DATA step shows that we want a design with two factors, **Brand** and **Price**. **Brand** has values "Sploosh", "Plumbbob", "Platter", "Moosey", and "Another". **Price** is created from **x1** for Sploosh, **x2** for Plumbbob, **x3** for Platter, **x4** for Moosey, and no attribute for Another (the constant alternative). The variables **Brand** and **Price** are logically quite different. **Brand** will be named on the **alt=** macro option as the alternative variable, so its values will literally come out of the **key=** data set. **Price** will not be named on the **alt=** macro option, so its values start out as variable names and (as we will see later) will end up as the values of those variables.

```
data key;
   input Brand $ Price $;
   datalines;
Sploosh   x1
Plumbbob  x2
Platter   x3
Moosey    x4
Another   .
;

proc print; run;
```

Choice of Fabric Softener

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 is blank (character missing). The period in the in-stream data set is simply a place holder used with list input to read both character and numeric missing data. A period is not stored with the data. Next, we use the %MKTROLL macro to process the design.

```
%mktroll(design=sasuser.des, key=key, alt=brand, out=rolled)
```

This step processes the **design=sasuser.des** 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. 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 original design matrix.

```
proc print data=sasuser.des(obs=3); run;
```

Choice of Fabric Softener

Obs	x1	x2	x3	x4
1	$1.49	$1.99	$1.49	$1.99
2	$1.99	$1.99	$1.99	$1.99
3	$1.99	$1.49	$1.49	$2.49

These observations define the first three choice sets. Here are those same observations, arrayed for analysis.

```
proc print data=rolled(obs=15); format price price.; run;
```

Choice of Fabric Softener

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

This data set has a choice set variable **Set**, an alternative name variable **Brand**, and a price variable **Price**.

The prices come from the design, and the price for "Another" is a constant $1.99. 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** experimental 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.

```
proc print data=res2(obs=15); run;
```

Choice of Fabric Softener

Obs	Subj	Set	Brand	Price	c
1	1	1	Sploosh	1	2
2	1	1	Plumbbob	2	2
3	1	1	Platter	1	1
4	1	1	Moosey	2	2
5	1	1	Another	.	2
6	1	2	Sploosh	2	2
7	1	2	Plumbbob	2	2
8	1	2	Platter	2	1
9	1	2	Moosey	2	2
10	1	2	Another	.	2
11	1	3	Sploosh	2	2
12	1	3	Plumbbob	1	2
13	1	3	Platter	1	1
14	1	3	Moosey	3	2
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 three choice sets. 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. However, instead we will use the **put** function to write the 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 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. The **design** option can optionally 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**. An alternative is to code **by subj**, but this is less efficient because block size is so small.

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 a constant variable is created by the coding, it is not to 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 when 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 are to be coded. The specification **class(brand / zero=none order=data)** specifies that the variable **Brand** is a classification variable and requests a binary coding. The **zero=none** option specifies that one binary variable should be created for all categories. The **order=data** option sorts the values 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. So 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 suppresses 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;
   title 'Choice of Fabric Softener';
   title2 'First 15 Observations of Analysis Data Set';
   id subj set c;
   run;
```

Choice of Fabric Softener
First 15 Observations of Analysis Data Set

Subj	Set	c	Sploosh	Plumbbob	Platter	Moosey	Another	Price	Brand
1	1	2	1	0	0	0	0	1.49	Sploosh
1	1	2	0	1	0	0	0	1.99	Plumbbob
1	1	1	0	0	1	0	0	1.49	Platter
1	1	2	0	0	0	1	0	1.99	Moosey
1	1	2	0	0	0	0	1	1.99	Another
1	2	2	1	0	0	0	0	1.99	Sploosh
1	2	2	0	1	0	0	0	1.99	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	1.99	Sploosh
1	3	2	0	1	0	0	0	1.49	Plumbbob
1	3	1	0	0	1	0	0	1.49	Platter
1	3	2	0	0	0	1	0	2.49	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;
   title2 'Discrete Choice Model';
   model c*c(2) = &_trgind / ties=breslow;
   strata subj set;
   run;
```

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 practical problems, so multiple choices are collected for each subject. Then 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 71 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

Stratum	Subj	Set	Number of Alternatives	Chosen Alternatives	Not Chosen
1	1	1	5	1	4
2	1	2	5	1	4
3	1	3	5	1	4
4	1	4	5	1	4
5	1	5	5	1	4
6	1	6	5	1	4
7	1	7	5	1	4
8	1	8	5	1	4
9	1	9	5	1	4
10	1	10	5	1	4
11	1	11	5	1	4
12	1	12	5	1	4
13	1	13	5	1	4
14	1	14	5	1	4
15	1	15	5	1	4
16	1	16	5	1	4
17	1	17	5	1	4
18	1	18	5	1	4
19	2	1	5	1	4
20	2	2	5	1	4
.					
.					
.					
898	50	16	5	1	4
899	50	17	5	1	4
900	50	18	5	1	4
Total			4500	900	3600

Convergence Status

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

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	2896.988	1445.310
AIC	2896.988	1455.310
SBC	2896.988	1479.322

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	1451.6781	5	<.0001
Score	1309.4957	5	<.0001
Wald	666.3325	5	<.0001

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Sploosh	1	-1.22716	0.21083	33.8783	<.0001
Plumbbob	1	-0.25760	0.17245	2.2315	0.1352
Platter	1	1.97660	0.14592	183.4752	<.0001
Moosey	1	0.52028	0.16253	10.2467	0.0014
Another	0	0	.	.	.
Price	1	-4.50980	0.20231	496.8925	<.0001

The procedure output begins with information about each of the 900 strata. The first 20 and last three strata are shown. Each subject and choice set combination consists of a total of five observations, one that was chosen (the event) and four that are unchosen (censored). The **nosummary** option can be used to suppress this output, but it is a good idea to *not* specify **nosummary**, since the summary information can help verify that the data are arrayed correctly. We will see on page 129 how to print a compact summary of the summary table.

The most to least preferred brands are: *Platter*, *Moosey*, *Another*, *Plumbbob*, and *Sploosh*. Increases in price have a negative utility. For example, the predicted utility of *Platter* brand at \$1.99 is $\mathbf{x}_i\boldsymbol{\beta}$ which is $(0 \quad 0 \quad 1 \quad 0 \quad 0 \quad \$1.99) \quad (-1.23 \quad -0.26 \quad 1.98 \quad 0.52 \quad 0 \quad -4.51)' = 1.98 + 1.99 \times -4.51 = -6.99$. 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 $1.98 + 1.89 \times -4.51 = -6.54$, which is a $\$0.10 \times 4.51 = 0.45$ 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. Nevertheless, choice modeling is a useful and popular marketing research technique.

First, PROC SCORE is used to compute the predicted utility for each alternative.

```
proc score data=coded(where=(subj=1) drop=c) score=betas type=parms out=p;
   var &_trgind;
   run;
```

The data set to be scored is named with the **data=** option, and the coefficients are specified in the option **score=beta**. Note that we only need to read all of the choice sets once, since the parameter estimates were computed in an aggregate analysis. This is why we specified where=(subj=1). We do not need $\mathbf{x}_j\hat{\boldsymbol{\beta}}$ for each of the different subjects. We dropped the variable **c** from the CODED data set since this name will be used by PROC SCORE for the results ($\mathbf{x}_j\hat{\boldsymbol{\beta}}$). The option **type=parms** specifies that the **score=** data set contains the parameters in **_TYPE_ = 'PARMS'** observations. The output data set with the predicted utilities is named P. Scoring is based on the coded variables from PROC TRANSREG, whose names are contained in the macro variable **&_trgind**. The next step exponentiates $\mathbf{x}_j\hat{\boldsymbol{\beta}}$.

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

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

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

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

```
data p;
   merge p2 s(keep=set sp);
   by set;
   p = p / sp;
   keep brand set price p;
   run;
```

Here are the results for the first three choice sets.

```
proc print data=p(obs=15);
   title2 'Choice Probabilities for the First 3 Choice Sets';
   run;
```

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

Obs	Price	Brand	Set	p
1	1.49	Sploosh	1	0.03723
2	1.99	Plumbbob	1	0.01030
3	1.49	Platter	1	0.91674
4	1.99	Moosey	1	0.02241
5	1.99	Another	1	0.01332

6	1.99	Sploosh	2	0.02673
7	1.99	Plumbbob	2	0.07048
8	1.99	Platter	2	0.65819
9	1.99	Moosey	2	0.15342
10	1.99	Another	2	0.09119
11	1.99	Sploosh	3	0.00377
12	1.49	Plumbbob	3	0.09489
13	1.49	Platter	3	0.88619
14	2.49	Moosey	3	0.00227
15	1.99	Another	3	0.01288

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 107.

First, the macro variable **&forms** is created. It contains the number of separate questionnaires (or forms or subjects). We can use PROC PLAN to create random orders for the choice sets and alternatives. The data set created by PROC PLAN has $50 \times 18 \times 4$ observations for 50 people, 18 choice sets, and 4 alternatives, not counting the constant alternative. There are $18 \times 4 = 72$ observations for subject 1, followed by 72 observations for subject 2, ..., followed by 72 observations for subject 50. The **Form** variable is ordered due to the **ordered** option in the **factors** statement. 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. The **set=** specification in the **factors** statement creates the random choice set order, and **alt=** creates the random alternative 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;
title 'Create 50 Custom Questionnaires';

proc plan seed=7654321;
   factors Form=&forms ordered Set=&n Alt=&mm1 / noprint;
   output out=sasuser.orders;
   run; quit;

proc print data=sasuser.orders(obs=16);
   run;
```

The first 16 observations in this data set are shown next.

Create 50 Custom Questionnaires

Obs	Form	Set	Alt
1	1	16	2
2	1	16	3
3	1	16	1
4	1	16	4
5	1	18	4
6	1	18	2
7	1	18	3
8	1	18	1

```
 9      1        8      1
10      1        8      4
11      1        8      3
12      1        8      2
13      1        4      1
14      1        4      4
15      1        4      2
16      1        4      3
```

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

Obs	Form	Set	COL1	COL2	COL3	COL4
1	1	16	2	3	1	4
2	1	18	4	2	3	1
3	1	8	1	4	3	2
4	1	4	1	4	2	3
5	1	11	1	3	2	4
6	1	13	2	4	3	1
7	1	9	1	4	3	2
8	1	3	2	3	1	4
9	1	5	2	1	3	4
10	1	12	2	1	3	4
11	1	10	4	1	3	2
12	1	17	1	4	2	3
13	1	7	2	3	4	1
14	1	14	2	3	4	1
15	1	2	1	4	3	2
16	1	15	1	4	3	2
17	1	6	4	3	1	2
18	1	1	1	4	2	3

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] = 'Plumbob'**, 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) Plumbbob brand at $1.99.

    2) Platter brand at $1.99.

    3) Sploosh brand at $1.99.

    4) Moosey brand at $1.49.

    5) Another brand at $1.99.
```

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

   1) Moosey brand at $1.99.

   2) Plumbbob brand at $2.49.

   3) Platter brand at $1.49.

   4) Sploosh brand at $1.49.

   5) Another brand at $1.99.
```

Processing the Data for Custom Questionnaires

Here are the data. (Actually, these are the data that would have been collected if the same people as in the previous situation made the same choices, without error and uninfluenced by order effects.) Before these data are analyzed, the original order must be restored.

```
data results;                    /* read choice data set              */
   input Subj (choose1-choose&n) (1.) @@;
   datalines;
  1 433523224332243244  2 111134542142124243  3 414223142333323422
  4 143512112224441441  5 312513434513443233  6 444412234223112421
  7 123131242243411222  8 134312124143421411  9 142244314432511242
 10 124121151135114211 11 242144434121413233 12 351311124421241341
 13 153244214412414433 14 422322441221123324 15 211452423534132431
 16 213241243354321353 17 231424441351422145 18 123554323344445414
 19 434224442513232132 20 244415443124532223 21 341521125414222253
 22 521322422113145241 23 441314334433322221 24 434215341425341111
 25 341242445212232234 26 453513242124211541 27 244434142234533535
 28 324122444414142212 29 221313543351425441 30 535414424143141343
 31 244414242334451134 32 233444112314125124 33 113434111124545334
 34 442311112314222421 35 145141335453225333 36 233124413424242135
 37 214441243452324333 38 133134341255222251 39 123513314411443544
 40 441342414112143431 41 313242432133211112 42 111313111321411441
 43 124232133234532322 44 124131332124112243 45 452231431542313121
 46 131444324142443155 47 134133121134525123 48 353422425435444422
 49 524243122341231233 50 132123343334312112
;
```

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 2. The data set SASUSER.ORDERS shows that this choice of 2 corresponds to the third alternative (the second Col variable,

Col2 = 3) of choice set **Set=16**. The listing shows that the original order has been restored. Similarly, the second choice was 3. In the next observation of SASUSER.ORDERS, **Col3=3** and **Set=18**, so the $18th$ choice in the new data set is 3. The third choice was 3. In the next observation of SASUSER.ORDERS, **Col3=3** and **Set=8**, so the $8th$ choice in the new data set is 3. This process continues for the rest of the choices.

This DATA step writes out the data after the original order has been restored. It matches the data on page 94.

```
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 333542334333314443   2 333212344333333345   3 333212333333313333
  4 133242144334414453   5 335242134333513443   6 333242234333314443
  7 333432334332323443   8 333242234334414443   9 333432331352313343
 10 325222235332333443  11 333232334333313343  12 333242234333313453
 13 533212334332213443  14 142242144333213443  15 333223335333313345
 16 333434235333315343  17 533242234352313443  18 343445534332414543
 19 333342335332313443  20 333242234332315543  21 333252534333513443
 22 333242354333313543  23 333242333333313443  24 525222234332223443
 25 353342234333213343  26 333245545332313443  27 333352534333353343
 28 333232334333333343  29 333422534335353443  30 333252334533313443
 31 353342334332313443  32 353222234333334443  33 333222234352313345
 34 332244134333313443  35 343552234353413445  36 333244534333313443
 37 333244234334514443  38 353232334333353543  39 333252334333313543
 40 343234134332413343  41 333444244432413443  42 333232234332314443
 43 333242254333333443  44 333242234332313443  45 312252544432414443
 46 132242235433514443  47 543242534332413443  48 335452334333323453
 49 333542134334313443  50 333222334332314443
```

The data can be combined with the design and analyzed as in the previous example.

Vacation Example, Big Designs

This example illustrates creating a design when the full-factorial is too large to use as a candidate set. A researcher is interested in studying choice of vacation destinations. There are five destinations (alternatives) of interest: Hawaii, Alaska, Mexico, California, and Maine. 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). This problem requires a design with 15 three-level factors, denoted 3^{15}. The design has three factors, one per attribute, for each of the five destinations. 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. 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

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 261.) 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.

```
title 'Vacation Example, Strategies for Big Designs';

%mktruns( 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 )
```

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, Strategies for Big Designs

Some Reasonable Design Sizes (Saturated=31)	Violations	Cannot Be Divided By
36	0	
45	0	
54	0	
63	0	
72	0	
81	0	
90	0	
99	0	
108	0	
117	0	

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, any size of at least 36 choice sets and multiples of 9 choice sets can be optimal. Zero violations does not imply that we will always find a 100% efficient design. It just means that optimality is not precluded by unequal frequencies. We will create an efficient experimental design with 36 choice sets using the %MKTDES macro.

Designing the Choice Experiment

The following code creates a design.

```
%mktdes(factors=x1-x15=3, n=36, procopts=seed=7654321)
```

The **factors=** option specifies a design with 15 factors, **x1**−**x15**, each with three levels. A design with 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.

In the soap example, the macro first ran PROC PLAN to create a full-factorial candidate set. That will not work in this situation. The full-factorial design has $3^{15} = 14,348,907$ observations, which is way too many to use as a candidate set. The full-factorial design will require over 1642 mega-bytes of disk. It is often the case that the full-factorial design, even if it is small enough to create and store, is too big to use as a candidate set.

When is the candidate set too big? It depends on how fast your computer is, how much memory it has, how much free disk space it has, how much time you have, and so on. Often, you will find your best designs with small candidate sets. Here are some approximate guidelines. Under 2000 runs is small. $2000-5000$, while not small, may be very reasonable. $5000-10{,}000$ is big. Candidate sets with sizes closer to 5000 may work fine in a reasonable amount of time, but as the candidate set gets bigger, the software has a harder time finding an optimal design. You should only consider candidate sets with over 10,000 candidates when you have a lot of time to wait

for searches that will usually produce suboptimal designs. By default, the macro will not create a full-factorial candidate set that is bigger than $2188 = \max(2^{11}, 3^7) + 1$ runs. This is controlled with the **big=** option.

We will often create a candidate set whose size is a power of a prime number.

2^{12}	=	$4,096$	is reasonable although smaller sizes should be tried
2^{13}	=	$8,192$	is probably too big
2^{14}	=	$16,384$	is almost certainly too big
3^7	=	$2,187$	is reasonable
3^8	=	$6,561$	is probably too big
3^9	=	$19,683$	is almost certainly too big
5^5	=	$3,125$	is reasonable
5^6	=	$15,625$	is almost certainly too big
7^4	=	$2,401$	is reasonable
7^5	=	$16,807$	is almost certainly too big

Returning to the example, the macro first runs PROC FACTEX to create a fractional-factorial design. It creates a resolution III design for a candidate set. In a resolution III design, all main effects are estimable free from each other. Resolution III designs are usually much smaller than the full-factorial. See pages 68 and 111 for more detail about resolution. The macro writes PROC FACTEX code using an algorithm well-suited for much more complicated problems such as models with interactions and a mix of levels. This flexibility makes the code more complicated than is ideal for an introductory example, so first we will start more simply, and later we will look at the actual macro-generated code (page 140 at the very end of this example). Let's consider the PROC FACTEX code that we might have written for the problem if we were solving the design problem without the use of the %MKTDES macro.

```
proc factex;
   factors x1-x15 / nlev=3;
   size design=min;
   model res=3;
   output out=Cand1;
   run; quit;
```

The **factors** statement names the factors, **x1-x15**, and **nlev=** specifies the (constant) number of levels for each. The statements **size design=min** and **model res=3** combine to create a resolution III design in the minimum number of runs. The **output** statement with **out=** puts the design in a SAS data set named CAND1. The resulting candidate set has 81 runs (possible choice sets). The only real difference between this code and the macro generated code is the macro creates factors with values of (1, 2, 3), whereas in the code above and by default, PROC FACTEX creates factors with values of (-1, 0, 1).

After the candidate set is generated, the macro runs PROC OPTEX to find the final design. Here is the PROC OPTEX code that the macro generated.

```
proc optex data=Cand1 seed=7654321;
   class x1-x15 / param=orthref;
   model x1-x15;
   generate n=36 iter=10 keep=5 method=m_federov;
   output out=design;
   run; quit;
```

The PROC statement option **data=** names the candidate data set, and the **seed=** option specifies the random number seed. The **class** statement designates the variables **x1** through **x15** as classification variables, which means they are nominal or categorical as opposed to linear or quantitative. The **param=orthref** option generates an orthogonal coding for the design matrix. With this coding, the efficiency values range from 0 (some effects are not estimable) to 100 (a perfect design). The **model** statement specifies that we want to find a good design for a main effects model (no interactions). The **generate** statement requests the generation of an efficient design with **n=36** choice sets. The options **iter=10** and **keep=5** request that ten designs be independently generated and five be kept. The option **method=m_federov** specifies the modified Federov

algorithm (Federov (1972) and Cook and Nachtsheim (1980)), which is usually the most reliable. See page 111 for more information on the modified Federov algorithm. The **output** statement outputs the most efficient design to the SAS data set DESIGN. Here are the results.

Vacation Example, Strategies for Big Designs

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	84.4436	65.6780	67.2558	1.1450
2	84.4436	65.6780	67.2558	1.1450
3	84.4436	65.6780	67.2558	1.1450
4	84.4436	65.6780	67.2558	1.1450
5	84.4436	65.6780	67.2558	1.1450

The best design had a D-Efficiency of 84.4%, and the macro ran in under two seconds. The macro ran fast because the candidate set only had 81 candidates. When the candidate set is this small and run time is this fast, it is usually good to try again with bigger candidate sets to see if more efficient designs can be found. Since the factors all have three levels, larger candidate sets can be created by specifying a set size of the minimum (81) times 3, 9, 27, 81, and so on. Of course the actual numbers, 243 and 729 and so on, could have been specified.

```
%mktdes(factors=x1-x15=3, size=81*3, n=36, procopts=seed=7654321)
```

With $81 \times 3 = 243$ candidates, D-efficiency is better, 86.5650. PROCs FACTEX and OPTEX combined ran in under four seconds. Let's try again with a bigger candidate set.

```
%mktdes(factors=x1-x15=3, size=81*9, n=36, procopts=seed=7654321)
```

With $81 \times 9 = 729$ candidates, D-efficiency is again better, 89.4777. PROCs FACTEX and OPTEX combined ran in under twelve seconds. Let's try again with a bigger candidate set.

```
%mktdes(factors=x1-x15=3, size=81*27, n=36, procopts=seed=7654321)
```

With $81 \times 27 = 2187$ candidates, D-efficiency is again better, 92.2205. PROCs FACTEX and OPTEX combined ran in under 35 seconds. This candidate set is starting to get large, so this would be a reasonable stopping point. Still, it would not hurt to try one more time.

```
%mktdes(factors=x1-x15=3, size=81*81, n=36, procopts=seed=7654321)
```

With $81 \times 81 = 6561$ candidates, D-efficiency was this time a little worse, 91.6423. PROCs FACTEX and OPTEX combined ran in 2 minutes and 15 seconds. When the candidate set is large, PROC OPTEX has a harder time finding the best designs.

Candidate Sets and How PROC OPTEX Works

On page 109, our goal was to find a good design in 36 runs from an 81 run candidate set. The number of possible designs for this problem is $81!/(36!(81-36)!) = 1.3 \times 10^{23}$. Even if your computer could evaluate one billion designs a second, it would take over four million years to evaluate all possible designs. Furthermore, this is a small problem; researchers frequently have much larger designs and candidate sets. Exhaustive search is impossible, so PROC OPTEX uses heuristics to look for good designs.

The details of how PROC OPTEX works vary by algorithm, but typically and the way we use it with the %MKTDES macro, it starts by randomly selecting a design from the candidate set. The efficiency of the random design is evaluated. Then points that are in the design are considered for removal, and points that are not in the design are considered for inclusion. The effects on efficiency of these removals and inclusions are evaluated. If swapping a design point with a candidate point increases efficiency, it is done. This process continues until efficiency quits improving. The result is one of the designs that is printed in the efficiency table. (By default, no report is generated of the iterations that led to that design.) The process is repeated again starting with a new random design and iteratively refining it. The result is another of the designs in the efficiency table. This process occurs n times, from **iter=**n in the **generate** statement, which by default is 10. The designs are sorted by decreasing efficiency, and the efficiency table is printed. The table is not an iteration history. It is a list of information about n independently generated designs, ordered from best to worst. Because exhaustive searches are impossible, PROC OPTEX may fail to find *the* optimal design. However, the procedure invariably finds efficient designs.

We will always use the modified Federov algorithm (Federov (1972) and Cook and Nachtsheim (1980)) with PROC OPTEX when it is called from the %MKTDES macro. The modified Federov algorithm iteratively refines a design by considering swapping each candidate point in place of each design point. Consider a candidate set with n points and a design with m points. Each iteration of the modified Federov algorithm considers all pairs of $n \times m$ swaps $-$ each design point is removed and the effect of replacing it by each of the n candidates is evaluated. Candidates are swapped in and design points swapped out whenever efficiency improves. The process is repeated until $n \times m$ swaps are considered but nothing changes. The *original* Federov algorithm considers all $n \times m$ possible swaps then performs the *single* swap that leads to the greatest improvement in efficiency. This process is repeated until no swap increases efficiency. In contrast, the *modified* Federov algorithm performs *every* swap that increases efficiency. Other algorithms are available, however we will not use them since the modified Federov algorithm works so well.

As we saw on pages 109 through 110, sometimes you can find a more efficient design by using a larger candidate set or by letting PROC OPTEX run longer. Starting on page 109 we used a resolution III candidate set (all main effects are estimable free of each other). We started with small resolution III candidate sets and worked our way up to larger sets. We could also try resolution IV candidate sets (all main effects are estimable free of each other and free of all two-factor interactions, but some two-factor interactions are confounded with each other), or resolution V candidate sets (all main effects and two-way interactions are estimable free of each other). Other resolutions can be tried as well. The size of an orthogonal design is directly related to resolution. Specifying a smaller resolution will create a candidate set with fewer choice sets, and a larger resolution will create a candidate set with more choice sets. The size of the candidate set should be small relative to the full-factorial design but larger than the final desired design.

Before the development of the %MKTDES macro, we recommended the following strategy.

- Try using a resolution III design as the candidate set.
- Try using a resolution IV design as the candidate set.
- Try using a resolution V design as the candidate set.
- Try using a resolution III design, concatenated with a resolution IV and resolution V design as the candidate set.
- Try using a full-factorial design as the candidate set if it is not too big.

In fact the strategy outlined starting on page 109 seems to usually be superior. Page 109 suggests using the macro with increasingly larger values of **size=**. The macro is faster, more convenient, and usually does a better job than writing PROC FACTEX code with different resolutions. Still, it is good to know about and try other strategies sometimes. Searching for an efficient experimental design is like a box of chocolates. You never know what you'll get.

The choice of the size of the candidate set involves balancing the richness of the candidate set versus the computational difficulty of the search for an optimal design. Increasing the size of the candidate set gives PROC OPTEX more combinations to work with and hence usually increases the efficiency of the best design that can be constructed from the candidate set. However, as candidate set size increases, it becomes more difficult to find the best designs. This is because as the number of possible designs increases, the probability that the search will get stuck in a local optimum also increases. To compensate for this, more but slower searches may be necessary with larger candidate sets.

To envision how PROC OPTEX works, imagine a bunch of blind-folded kangaroos hopping around, looking for the top of Mt. Everest. The search for an efficient design is like a kangaroo jumping around until it reaches a place where it can only go down. We want to find the top of Mt. Everest, but we would be happy with K2, which is almost as high as Everest. We might also make do with other Himalayan peaks or even with Mt. McKinley. However, local optima such as underwater mountain peaks and the highest point in Nebraska are not good answers. Using a full-factorial design as a candidate set is like parachuting the kangaroos into random places on the planet. Most will drown, freeze or meet some other unpleasant fate, but occasionally, a kangaroo will find the top of a mountain. Since the kangaroos are being parachuted over the entire planet, some kangaroo will find Mt. Everest, given enough kangaroos and enough time. However, it may take a *very* long time. Using a minimum-sized resolution III candidate set is like parachuting kangaroos into *some* mountain range. They will find a peak very quickly, but you do not know if it is Everest because you may have dropped them in the wrong mountain range. Using increasingly larger candidate sets is like parachuting the kangaroos into increasingly larger areas: a region, country, continent, hemisphere, and planet. As the size of the candidate set increases, the chance that you will find the optimum or a very good local optimum increases, however each search takes longer and has a lower probability of success, so more searches may be necessary.

Generating the Final Design

Let's return to the 2187 run candidate set and ask PROC OPTEX to generate more than 10 designs, say **iter=50**. This should take less than $(50/10) \times 35$ seconds = 2 minutes and 35 seconds, about as much time as we spent with 6561 candidates. Also, the seed was changed so that the same first ten designs as before would not be generated. It is unlikely we will do a lot better with this strategy, but it is worth 2 1/2 minutes of computer time to find out.

```
%mktdes(factors=x1-x15=3, size=81*27, n=36, iter=50,
        procopts=seed=72555)
```

In fact it took about 2 minutes and 21 seconds and at D-efficiency = 91.3630 did a little worse than our previous best. This is a good place to stop and regenerate our best design. There are many other strategies that could be tried: specify bigger values for **iter=** before going to lunch or home for the evening, try concatenating the different candidate sets, try creating candidate sets other ways. Our experience suggests that it is unlikely you will do significantly better than we have already done using the simple strategy outlined here. It is instructive to compare the PROC OPTEX D-efficiency results for the different candidate set sizes.

	81 Candidates D-Efficiency	243 Candidates D-Efficiency	729 Candidates D-Efficiency	2187 Candidates D-Efficiency
1	84.4436	86.5650	89.4777	92.2205
2	84.4436	86.5550	89.4575	91.4788
3	84.4436	86.2203	89.4471	90.9939
4	84.4436	86.2174	89.2366	90.7937
5	84.4436	86.1137	89.2189	90.7782

With 81 candidates, the top five designs all have the same D-efficiency. This suggests that PROC OPTEX has probably found the best design that can be constructed from that candidate set. In contrast, as the candidate set gets larger, the D-efficiency of the top five designs becomes more variable. In those cases we are less certain that the optimal design that can be found in the candidates has been found. In particular, with the 2187 candidates, it is quite likely that there are better designs out there. However, they are probably only a little better, and it would probably take a long time to find them. Going from 81 candidates to 243, then 729, and 2187 is easy and it does not take much computer or analyst time. Furthermore, there is a substantial gain in efficiency from 84.4% to 92.2%. It is unlikely that performing more searches or using larger candidate sets will help much. Our experience suggests that the following points are good heuristics for design search.

- Keep candidate set sizes under 5000 runs.
- It is unlikely that more than 10 or 20 searches of a particular candidate set will be a lot better than just 10 or 20 searches.

Increasing the candidate set size or the number of searches can greatly increase search time, typically with a very diminished return, that is, typically with little or no increase in the efficiency of the final design. See page 192 for an illustration of this.

Here are the results for our most efficient design.

```
%mktdes(factors=x1-x15=3, size=81*27, n=36, procopts=seed=7654321)
```

Vacation Example, Strategies for Big Designs

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	92.2205	85.3867	77.0109	1.0042
2	91.4788	83.8273	74.0857	1.0135
3	90.9939	82.8629	74.2882	1.0194
4	90.7937	81.8787	71.7165	1.0255
5	90.7782	82.0349	74.1813	1.0245

Examining the Design

Before you use a design, you should always look at its characteristics. First, let's look at the one-way frequencies – the number of times each level appears. Since we have all three-level factors and 36 choice sets, we would like to see frequencies of all 12, though 11's and 13's will certainly be acceptable. We will also look at the n-way frequencies to ensure that we do not have duplicate choice sets. We will use PROC SUMMARY with the **ways** statement along with PROC PRINT to display the frequencies. In the interest of space, the two-way frequencies are not printed, They could have been requested by adding "**2**" to the **ways** statement, **ways 1 2 15**.

```
proc summary data=design;
   class _all_;
   ways 1 15;
   output out=sum;
   run;

proc print; by _type_; run;
```

Vacation Example, Strategies for Big Designs

TYPE=1

Obs	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	_FREQ_
1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	10
2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	2	14
3	.	.	.	.	.	.	.	.	.	.	.	.	.	.	3	12

TYPE=2

Obs	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	_FREQ_
4	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	11
5	.	.	.	.	.	.	.	.	.	.	.	.	.	2	.	13
6	.	.	.	.	.	.	.	.	.	.	.	.	.	3	.	12

TYPE=4

Obs	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	_FREQ_
7	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	14
8	.	.	.	.	.	.	.	.	.	.	.	.	2	.	.	12
9	.	.	.	.	.	.	.	.	.	.	.	.	3	.	.	10

TYPE=8

Obs	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	_FREQ_
10	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	12
11	.	.	.	.	.	.	.	.	.	.	.	2	.	.	.	11
12	.	.	.	.	.	.	.	.	.	.	.	3	.	.	.	13

TYPE=16

Obs	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	_FREQ_
13	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	11
14	.	.	.	.	.	.	.	.	.	.	2	.	.	.	.	11
15	.	.	.	.	.	.	.	.	.	.	3	.	.	.	.	14

TYPE=32

Obs	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	_FREQ_
16	.	.	.	.	.	.	.	.	.	1	.	.	.	.	.	13
17	.	.	.	.	.	.	.	.	.	2	.	.	.	.	.	9
18	.	.	.	.	.	.	.	.	.	3	.	.	.	.	.	14

TYPE=64

Obs	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	_FREQ_
19	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.	12
20	.	.	.	.	.	.	.	.	2	.	.	.	.	.	.	12
21	.	.	.	.	.	.	.	.	3	.	.	.	.	.	.	12

TYPE=128

Obs	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	_FREQ_
22	.	.	.	.	.	.	.	1	.	.	.	.	.	.	.	12
23	.	.	.	.	.	.	.	2	.	.	.	.	.	.	.	13
24	.	.	.	.	.	.	.	3	.	.	.	.	.	.	.	11

TYPE=256

Obs	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	_FREQ_
25	.	.	.	.	.	.	1	.	.	.	.	.	.	.	.	12
26	.	.	.	.	.	.	2	.	.	.	.	.	.	.	.	14
27	.	.	.	.	.	.	3	.	.	.	.	.	.	.	.	10

---------- _TYPE_=512 ----------

Obs	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	_FREQ_
28	.	.	.	.	.	1	.	.	.	.	.	.	.	.	.	12
29	.	.	.	.	.	2	.	.	.	.	.	.	.	.	.	11
30	.	.	.	.	.	3	.	.	.	.	.	.	.	.	.	13

---------- _TYPE_=1024 ----------

Obs	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	_FREQ_
31	.	.	.	.	1	.	.	.	.	.	.	.	.	.	.	14
32	.	.	.	.	2	.	.	.	.	.	.	.	.	.	.	10
33	.	.	.	.	3	.	.	.	.	.	.	.	.	.	.	12

---------- _TYPE_=2048 ----------

Obs	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	_FREQ_
34	.	.	.	1	.	.	.	.	.	.	.	.	.	.	.	11
35	.	.	.	2	.	.	.	.	.	.	.	.	.	.	.	13
36	.	.	.	3	.	.	.	.	.	.	.	.	.	.	.	12

---------- _TYPE_=4096 ----------

Obs	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	_FREQ_
37	.	.	1	.	.	.	.	.	.	.	.	.	.	.	.	12
38	.	.	2	.	.	.	.	.	.	.	.	.	.	.	.	13
39	.	.	3	.	.	.	.	.	.	.	.	.	.	.	.	11

---------- _TYPE_=8192 ----------

Obs	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	_FREQ_
40	.	1	.	.	.	.	.	.	.	.	.	.	.	.	.	11
41	.	2	.	.	.	.	.	.	.	.	.	.	.	.	.	12
42	.	3	.	.	.	.	.	.	.	.	.	.	.	.	.	13

---------- _TYPE_=16384 ----------

Obs	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	_FREQ_
43	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	11
44	2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	12
45	3	.	.	.	.	.	.	.	.	.	.	.	.	.	.	13

```
-------------------------------- _TYPE_=32767 --------------------------------

 Obs  x1  x2  x3  x4  x5  x6  x7  x8  x9  x10  x11  x12  x13  x14  x15  _FREQ_

  46   1   1   2   3   1   3   2   1   1   1    1    1    1    1    1      1
  47   1   1   2   3   2   3   2   3   3   3    3    3    3    3    3      1
  48   1   1   2   3   3   2   3   2   2   2    2    2    2    2    2      1
  49   1   2   1   1   1   1   1   3   3   3    2    2    2    1    1      1
  50   1   2   1   1   2   1   1   2   2   2    1    1    1    3    3      1
  51   1   2   1   2   1   2   1   1   1   1    3    3    3    2    2      1
  52   1   2   2   1   3   1   3   1   1   1    3    3    3    2    2      1
  53   1   3   1   2   3   2   2   3   3   3    1    1    1    2    2      1
  54   1   3   3   2   1   3   2   2   2   2    3    3    3    1    1      1
  55   1   3   3   2   2   3   2   1   1   1    2    2    2    3    3      1
  56   1   3   3   3   1   3   3   3   3   3    1    1    1    2    2      1
  57   2   1   2   1   1   2   2   3   2   1    3    2    1    3    2      1
  58   2   1   3   1   2   2   1   2   1   3    2    1    3    2    1      1
  59   2   1   3   1   3   2   1   1   3   2    1    3    2    1    3      1
  60   2   1   3   3   3   1   1   3   2   1    3    2    1    3    2      1
  61   2   2   1   2   3   3   2   2   1   3    1    3    2    3    2      1
  62   2   2   2   2   2   3   3   1   3   2    3    2    1    2    1      1
  63   2   2   2   2   3   3   3   3   2   1    2    1    3    1    3      1
  64   2   2   2   3   1   1   1   2   1   3    1    3    2    3    2      1
  65   2   3   1   3   1   1   2   1   3   2    2    1    3    3    2      1
  66   2   3   1   3   1   2   3   2   1   3    3    2    1    1    3      1
  67   2   3   1   3   2   1   2   3   2   1    1    3    2    2    1      1
  68   2   3   2   2   3   1   2   2   1   3    3    2    1    1    3      1
  69   3   1   1   1   1   3   2   2   3   1    2    3    1    2    3      1
  70   3   1   1   2   2   1   3   1   2   3    1    2    3    1    2      1
  71   3   1   1   2   3   1   3   3   1   2    3    1    2    3    1      1
  72   3   1   3   2   1   1   1   2   3   1    2    3    1    2    3      1
  73   3   2   1   3   3   3   1   1   2   3    3    1    2    2    3      1
  74   3   2   3   1   1   1   2   1   2   3    3    1    2    2    3      1
  75   3   2   3   3   2   2   2   3   1   2    2    3    1    1    2      1
  76   3   2   3   3   3   2   2   2   3   1    1    2    3    3    1      1
  77   3   3   2   1   1   3   1   3   1   2    1    2    3    2    3      1
  78   3   3   2   1   3   3   1   1   2   3    2    3    1    3    1      1
  79   3   3   2   2   1   2   3   1   2   3    2    3    1    3    1      1
  80   3   3   2   2   2   2   1   2   3   1    3    1    2    1    2      1
  81   3   3   3   1   2   3   3   2   3   1    3    1    2    1    2      1
```

We see a few 9's, 10's, and 14's that we would rather not see, but overall, balance looks pretty good. The following program summarizes the two-way frequencies. PROC SUMMARY with the **ways 2** statement generates all two way frequencies and outputs them to a SAS data set. All frequencies are stored in the variable **_freq_**. Rather than print all $15 * (15-1)/2 = 105$ tables, we summarize the frequencies with a PROC FREQ step.

```
proc summary noprint data=design;
   ways 2;
   class x:;
   output out=res(keep=_freq_);
   run;

proc freq; run;
```

Ideally, we would like all two-way frequencies to be $36/(3 \times 3) = 4$, but we would certainly expect some threes and fives. Here are the results.

The FREQ Procedure

FREQ	Frequency	Percent	Cumulative Frequency	Cumulative Percent
2	7	0.74	7	0.74
3	278	29.42	285	30.16
4	400	42.33	685	72.49
5	229	24.23	914	96.72
6	30	3.17	944	99.89
7	1	0.11	945	100.00

The results look pretty good. $(278 + 400 + 229)/954$ or 96% of the frequencies are 3, 4, or 5. If the results were not acceptable, you could run the macro again with different seeds until you found a design you liked better. Some researchers are willing to even sacrifice a little bit of efficiency for better balance (see page 221).

You could also rerun the PROC OPTEX step, copying the generated code and adding an **`examine i`** statement, to print the information matrix, which is the covariance matrix of the parameter estimates. You hope to see all of the off-diagonal elements, the covariances, are small relative to the variances on the diagonal.

```
proc optex data=Cand1 seed=7654321;
   class x1-x15 / param=orthref;
   model x1-x15;
   generate n=36 iter=10 keep=5 method=m_federov;
   output out=design;
   examine i;
   run; quit;
```

Information Matrix

	Intercept	x11	x12	x21	x22	x31	x32	x41
Intercept	36.0	-2.1	-1.2	-2.1	-1.2	0.0	2.4	-2.1
x11	-2.1	34.5	0.9	-1.5	0.9	1.5	0.9	-1.5
x12	-1.2	0.9	37.5	0.9	1.5	0.9	4.5	-4.3
x21	-2.1	-1.5	0.9	34.5	0.9	-3.0	-1.7	3.0
x22	-1.2	0.9	1.5	0.9	37.5	3.5	0.0	3.5
x31	0.0	1.5	0.9	-3.0	3.5	36.0	-1.7	-3.0
x32	2.4	0.9	4.5	-1.7	0.0	-1.7	36.0	-1.7
x41	-2.1	-1.5	-4.3	3.0	3.5	-3.0	-1.7	34.5
x42	1.2	-0.9	-4.5	-3.5	-3.0	1.7	3.0	-0.9
x51	4.2	3.0	-1.7	-1.5	-4.3	1.5	0.9	3.0
x52	-2.4	1.7	-3.0	-0.9	-4.5	-0.9	-4.5	1.7
x61	0.0	-3.0	3.5	1.5	6.1	9.0	-1.7	1.5
x62	-2.4	-3.5	3.0	4.3	1.5	1.7	-3.0	-0.9
x71	0.0	-3.0	-1.7	1.5	6.1	0.0	-1.7	10.5
x72	4.9	1.7	3.0	-0.9	-1.5	1.7	-6.0	-0.9
x81	0.0	1.5	-4.3	-3.0	3.5	0.0	3.5	1.5
x82	2.4	-4.3	-1.5	-1.7	0.0	-1.7	-3.0	0.9
x91	0.0	1.5	6.1	-3.0	3.5	0.0	3.5	-3.0
x92	0.0	-2.6	1.5	-0.0	3.0	-0.0	6.0	-0.0
x101	2.1	0.0	-1.7	4.5	0.9	-6.0	-1.7	0.0
x102	-6.1	1.7	-0.0	4.3	1.5	-3.5	-3.0	1.7
x111	-2.1	3.0	3.5	-1.5	0.9	6.0	-1.7	-1.5
x112	-3.7	0.0	-3.0	2.6	-1.5	0.0	-3.0	2.6
x121	0.0	1.5	-4.3	-3.0	-1.7	4.5	-4.3	1.5
x122	-2.4	-0.9	4.5	1.7	-3.0	-0.9	1.5	-0.9
x131	4.2	-1.5	0.9	3.0	-6.9	-3.0	3.5	-1.5
x132	2.4	-4.3	-1.5	-1.7	-0.0	3.5	-3.0	0.9
x141	-2.1	-1.5	0.9	-1.5	-4.3	-3.0	-1.7	-1.5
x142	1.2	4.3	-4.5	-0.9	1.5	1.7	-3.0	4.3
x151	-4.2	0.0	-1.7	-0.0	-1.7	-1.5	0.9	0.0
x152	2.4	3.5	3.0	-1.7	-0.0	0.9	1.5	-6.9

Information Matrix

	x42	x51	x52	x61	x62	x71	x72	x81
Intercept	1.2	4.2	-2.4	0.0	-2.4	0.0	4.9	0.0
x11	-0.9	3.0	1.7	-3.0	-3.5	-3.0	1.7	1.5
x12	-4.5	-1.7	-3.0	3.5	3.0	-1.7	3.0	-4.3
x21	-3.5	-1.5	-0.9	1.5	4.3	1.5	-0.9	-3.0
x22	-3.0	-4.3	-4.5	6.1	1.5	6.1	-1.5	3.5
x31	1.7	1.5	-0.9	9.0	1.7	0.0	1.7	0.0
x32	3.0	0.9	-4.5	-1.7	-3.0	-1.7	-6.0	3.5
x41	-0.9	3.0	1.7	1.5	-0.9	10.5	-0.9	1.5
x42	37.5	-3.5	-0.0	-0.9	-1.5	-0.9	-4.5	4.3
x51	-3.5	39.0	1.7	1.5	-0.9	1.5	4.3	1.5
x52	-0.0	1.7	33.0	-0.9	-1.5	-0.9	1.5	-0.9
x61	-0.9	1.5	-0.9	36.0	1.7	4.5	-0.9	0.0
x62	-1.5	-0.9	-1.5	1.7	36.0	4.3	-1.5	-3.5
x71	-0.9	1.5	-0.9	4.5	4.3	36.0	-3.5	-0.0
x72	-4.5	4.3	1.5	-0.9	-1.5	-3.5	36.0	-3.5
x81	4.3	1.5	-0.9	0.0	-3.5	-0.0	-3.5	36.0
x82	4.5	0.9	1.5	-1.7	3.0	3.5	0.0	-1.7
x91	1.7	1.5	-0.9	0.0	1.7	0.0	1.7	0.0
x92	0.0	-2.6	-4.5	5.2	0.0	-0.0	-3.0	5.2
x101	1.7	0.0	1.7	-1.5	-0.9	-1.5	4.3	-1.5
x102	0.0	-3.5	3.0	-0.9	1.5	-0.9	-1.5	-0.9
x111	-6.1	-1.5	-0.9	1.5	-0.9	1.5	4.3	-3.0
x112	-1.5	2.6	1.5	-2.6	1.5	2.6	1.5	0.0
x121	-0.9	-3.0	1.7	0.0	-3.5	0.0	-3.5	0.0
x122	-1.5	-3.5	-0.0	1.7	3.0	-3.5	-6.0	-3.5
x131	-0.9	7.5	-0.9	-3.0	1.7	-3.0	1.7	-3.0
x132	-1.5	-4.3	-1.5	3.5	0.0	3.5	0.0	-1.7
x141	4.3	-1.5	4.3	-3.0	1.7	-3.0	-3.5	-3.0
x142	1.5	4.3	1.5	-3.5	0.0	1.7	-6.0	1.7
x151	1.7	0.0	1.7	-1.5	-0.9	-1.5	-0.9	3.0
x152	-3.0	-1.7	-0.0	0.9	10.5	-4.3	-4.5	-1.7

Information Matrix

	x82	x91	x92	x101	x102	x111	x112	x121
Intercept	2.4	0.0	0.0	2.1	-6.1	-2.1	-3.7	0.0
x11	-4.3	1.5	-2.6	0.0	1.7	3.0	0.0	1.5
x12	-1.5	6.1	1.5	-1.7	-0.0	3.5	-3.0	-4.3
x21	-1.7	-3.0	-0.0	4.5	4.3	-1.5	2.6	-3.0
x22	0.0	3.5	3.0	0.9	1.5	0.9	-1.5	-1.7
x31	-1.7	0.0	-0.0	-6.0	-3.5	6.0	0.0	4.5
x32	-3.0	3.5	6.0	-1.7	-3.0	-1.7	-3.0	-4.3
x41	0.9	-3.0	-0.0	0.0	1.7	-1.5	2.6	1.5
x42	4.5	1.7	0.0	1.7	0.0	-6.1	-1.5	-0.9
x51	0.9	1.5	-2.6	0.0	-3.5	-1.5	2.6	-3.0
x52	1.5	-0.9	-4.5	1.7	3.0	-0.9	1.5	1.7
x61	-1.7	0.0	5.2	-1.5	-0.9	1.5	-2.6	0.0
x62	3.0	1.7	0.0	-0.9	1.5	-0.9	1.5	-3.5
x71	3.5	0.0	-0.0	-1.5	-0.9	1.5	2.6	0.0
x72	0.0	1.7	-3.0	4.3	-1.5	4.3	1.5	-3.5
x81	-1.7	0.0	5.2	-1.5	-0.9	-3.0	0.0	0.0
x82	36.0	3.5	-3.0	0.9	-1.5	-1.7	0.0	-1.7
x91	3.5	36.0	0.0	-1.5	-0.9	1.5	-2.6	-4.5
x92	-3.0	0.0	36.0	-2.6	-1.5	-2.6	-1.5	-2.6
x101	0.9	-1.5	-2.6	37.5	4.3	-4.5	-2.6	-1.5
x102	-1.5	-0.9	-1.5	4.3	34.5	-0.9	1.5	-0.9
x111	-1.7	1.5	-2.6	-4.5	-0.9	34.5	2.6	1.5
x112	0.0	-2.6	-1.5	-2.6	1.5	2.6	37.5	-2.6
x121	-1.7	-4.5	-2.6	-1.5	-0.9	1.5	-2.6	36.0
x122	-3.0	-0.9	1.5	-0.9	1.5	-0.9	-4.5	1.7
x131	-1.7	-3.0	0.0	0.0	-3.5	-1.5	2.6	-3.0
x132	3.0	-1.7	0.0	-1.7	-3.0	0.9	-1.5	3.5
x141	3.5	1.5	-2.6	0.0	1.7	-1.5	-2.6	1.5
x142	0.0	-0.9	-4.5	1.7	-0.0	-0.9	-1.5	4.3
x151	-1.7	-1.5	2.6	-3.0	1.7	0.0	5.2	-1.5
x152	-3.0	0.9	-1.5	3.5	0.0	3.5	-3.0	0.9

Information Matrix

	x122	x131	x132	x141	x142	x151	x152
Intercept	-2.4	4.2	2.4	-2.1	1.2	-4.2	2.4
x11	-0.9	-1.5	-4.3	-1.5	4.3	0.0	3.5
x12	4.5	0.9	-1.5	0.9	-4.5	-1.7	3.0
x21	1.7	3.0	-1.7	-1.5	-0.9	-0.0	-1.7
x22	-3.0	-6.9	-0.0	-4.3	1.5	-1.7	-0.0
x31	-0.9	-3.0	3.5	-3.0	1.7	-1.5	0.9
x32	1.5	3.5	-3.0	-1.7	-3.0	0.9	1.5
x41	-0.9	-1.5	0.9	-1.5	4.3	0.0	-6.9
x42	-1.5	-0.9	-1.5	4.3	1.5	1.7	-3.0
x51	-3.5	7.5	-4.3	-1.5	4.3	0.0	-1.7
x52	-0.0	-0.9	-1.5	4.3	1.5	1.7	-0.0
x61	1.7	-3.0	3.5	-3.0	-3.5	-1.5	0.9
x62	3.0	1.7	0.0	1.7	0.0	-0.9	10.5
x71	-3.5	-3.0	3.5	-3.0	1.7	-1.5	-4.3
x72	-6.0	1.7	0.0	-3.5	-6.0	-0.9	-4.5
x81	-3.5	-3.0	-1.7	-3.0	1.7	3.0	-1.7
x82	-3.0	-1.7	3.0	3.5	0.0	-1.7	-3.0
x91	-0.9	-3.0	-1.7	1.5	-0.9	-1.5	0.9
x92	1.5	0.0	0.0	-2.6	-4.5	2.6	-1.5
x101	-0.9	0.0	-1.7	0.0	1.7	-3.0	3.5
x102	1.5	-3.5	-3.0	1.7	-0.0	1.7	0.0
x111	-0.9	-1.5	0.9	-1.5	-0.9	0.0	3.5
x112	-4.5	2.6	-1.5	-2.6	-1.5	5.2	-3.0
x121	1.7	-3.0	3.5	1.5	4.3	-1.5	0.9
x122	36.0	1.7	0.0	4.3	-1.5	-0.9	-1.5
x131	1.7	39.0	-1.7	-1.5	-0.9	0.0	-1.7
x132	0.0	-1.7	33.0	0.9	-1.5	-1.7	-0.0
x141	4.3	-1.5	0.9	34.5	-0.9	0.0	-1.7
x142	-1.5	-0.9	-1.5	-0.9	37.5	-3.5	-3.0
x151	-0.9	0.0	-1.7	0.0	-3.5	33.0	-1.7
x152	-1.5	-1.7	-0.0	-1.7	-3.0	-1.7	39.0

This design looks reasonable.

Blocking and Randomizing the Design

Thirty-six choice sets may be too many for one person to rate. Hence, before the design is used, it should be blocked. We can create two blocks of size 18 so no person has to make more than 18 choices. The following code does the blocking and outputs the results to a SAS data set BLOCKDES. The goal is to take the observations in an existing design and optimally sort them into blocks. No swapping between the candidate set and the design is performed. The **generate** statement options **initdesign=design method=sequential** name the design to block, DESIGN, and the sequential method since no swapping in or out is performed. The **blocks** statement option **structure=(2)18** asks for 2 blocks of size 18, **init=chain** specifies no swapping from the candidate set during the initialization, and **noexchange** specifies no swapping from the candidate set during the iterations.

```
proc optex data=design seed=72343;
   title3 'Blocking an Existing Design';
   class x1-x15 / param=orthref;
   model x1-x15;
   generate initdesign=design method=sequential;
   blocks structure=(2)18 init=chain noexchange iter=1;
   output out=blockdes;
   run; quit;

proc freq data=blockdes;
   tables block * (x:);
   run;
```

The PROC FREQ step prints all of the one-way frequencies within blocks. In the interest of space, they are not shown here. However, they should be examined to ensure that each level is well represented in each block.

Before the design is used, the order of the choice sets should be randomized within blocks.

```
%let m   = 6;                    /* m alternatives including constant */
%let mm1 = %eval(&m - 1);        /* m - 1                             */
%let n   = 18;                   /* number of choice sets             */
%let blocks = 2;                 /* number of blocks                  */

proc plan seed=7654321;
   factors block=&blocks ordered set=&n / noprint;
   output out=orders;
   run; quit;

data sasuser.blockdes;
   set orders;
   set = (block - 1) * &n + set;
   set blockdes point=set;
   run;
```

We can use PROC PLAN to create a data set with a variable **Block** with **&blocks=2** values, the integers 1 through 2. Within each block, the integers 1 through 18 are generated in a random order. These orders are read in a DATA step and used to read the BLOCKDES data set in the random order within blocks. The **point=** specification specifies that the variable **Set** contains the order in which to read the observations. The assignment statement **set = (block - 1) * 18 + set** works as follows. The **(block - 1) * 18** produces the observation number before the start of each block, 0 or 18. With the addition of **Set**, which contains the random choice set order within each block, we get the observation numbers within each set. Note that the final design is stored in a permanent SAS data set, SASUSER.BLOCKDES, so it will still exist after the data are collected.

Generating the Questionnaire

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

```
title;
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 generic attributes. Each destination name is accessed from the array **dests**. Note that destination is not a factor in the design; it is a "bin" into which the attributes are grouped. The factors in the design are named in the statement **array x[15]**, which is a short-hand notation for **array x[15] x1-x15**. The first five factors are used for the lodging attribute of the five destinations. The actual descriptions of lodging are accessed by **lodging[x[dest]]**. The variable **Dest** varies from 1 to 5 destinations, so **x[dest]** extracts the levels for the **Dest** destination. Similarly for scenery, **scenes[x[&mm1 + dest]]** extracts the descriptions of the scenery. The index **&mm1 + dest** accesses factors 6 through 10, and **x[&mm1 + dest]** indexes the **scenes** array. For prices, **prices[x[2 * &mm1 + dest]]**, the index **2 * &mm1 + dest** accesses the factors 11 through 15. Here are the first two choice sets.

```
                                        Form: 1  Subject: ________


 1) Circle your choice of vacation destinations:

    1) Hawaii, staying in a Hotel near a Lake,
       with a package cost of $1249.

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

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

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

    5) Maine, staying in a Bed & Breakfast near a Lake,
       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 Beach,
      with a package cost of $1499.

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

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

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

   5) Maine, staying in a Hotel near the Beach,
      with a package cost of $1499.

   6) Stay at home this year.
```

Entering and Processing the Data

Here are some of the input data. Data from a total of 200 subjects were collected, 100 per form.

```
title 'Vacation Example, Strategies for Big Designs';

data results;
   input Subj Form (choose1-choose&n) (1.) @@;
   datalines;
  1  1 321512533111543443   2  2 435421113312413133   3  1 331311531112543413
  4  2 431131321114411133   5  1 341111531113143443   6  2 1413412133121111133
  7  1 341513531312145414   8  2 434111213344453114   9  1 341514131113145424
 10  2 444313233322411113  11  1 321511131143123443  12  2 435151213413311134
 13  1 321512531112135443  14  2 433111213342113133  15  1 141211111113143414
 16  2 545433233323451133  17  1 344511533114145443  18  2 5434311223321451115
 19  1 325511131115143413  20  2 435431233112413133  21  1 1313115311113145443
.
.
.
;
```

These next steps prepare the design for analysis. First, we create a data set KEY that describes how the factors in our design will be used for analysis.

```
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); run;

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

Vacation Example, Strategies for Big Designs

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

Vacation Example, Strategies for Big Designs

Obs	Set	Place	Lodge	Scene	Price
1	1	Hawaii	3	2	2
2	1	Alaska	2	2	3
3	1	Mexico	3	3	1
4	1	California	3	1	1
5	1	Maine	2	2	2
6	1	Home	.	.	.
7	2	Hawaii	3	3	3
8	2	Alaska	2	1	1
9	2	Mexico	1	1	2
10	2	California	3	2	2
11	2	Maine	3	3	3
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, Strategies for Big Designs

Obs	Set	Place	Lodge	Scene	Price
1	1	Hawaii	Hotel	Lake	1249
2	1	Alaska	Bed & Breakfast	Lake	1499
3	1	Mexico	Hotel	Beach	999
4	1	California	Hotel	Mountains	999
5	1	Maine	Bed & Breakfast	Lake	1249
6	1	Home	Home	Home	0
7	2	Hawaii	Hotel	Beach	1499
8	2	Alaska	Bed & Breakfast	Mountains	999
9	2	Mexico	Cabin	Mountains	1249
10	2	California	Hotel	Lake	1249
11	2	Maine	Hotel	Beach	1499
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, Strategies for Big Designs

Obs	Subj	Form	Set	Place	Lodge	Scene	Price	c
1	1	1	1	Hawaii	Hotel	Lake	1249	2
2	1	1	1	Alaska	Bed & Breakfast	Lake	1499	2
3	1	1	1	Mexico	Hotel	Beach	999	1
4	1	1	1	California	Hotel	Mountains	999	2
5	1	1	1	Maine	Bed & Breakfast	Lake	1249	2
6	1	1	1	Home	Home	Home	0	2
7	1	1	2	Hawaii	Hotel	Beach	1499	2
8	1	1	2	Alaska	Bed & Breakfast	Mountains	999	1
9	1	1	2	Mexico	Cabin	Mountains	1249	2
10	1	1	2	California	Hotel	Lake	1249	2
11	1	1	2	Maine	Hotel	Beach	1499	2
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. The **design** option can optionally 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**. An alternative is to code **by subj**, but this is less efficient because block size is so small.

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 a constant variable is created by the coding, it is not to 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 when 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 are to be coded. The specification **class(place / zero=none order=data)** specifies that the variable **Place** is a classification variable and requests a binary coding. The **zero=none** option specifies that one binary variable should be created for all categories. The **order=data** option sorts the values 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 / zero=none order=formatted)** 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. So 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 suppresses 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, Strategies for Big Designs

Place	Subj	Set	Form	c	Price	Scene	Lodge
Hawaii	1	1	1	2	1249	Lake	Hotel
Alaska	1	1	1	2	1499	Lake	Bed & Breakfast
Mexico	1	1	1	1	999	Beach	Hotel
California	1	1	1	2	999	Mountains	Hotel
Maine	1	1	1	2	1249	Lake	Bed & Breakfast
Home	1	1	1	2	0	Home	Home

Vacation Example, Strategies for Big Designs

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, Strategies for Big Designs

place	Beach	Home	Lake	Mountains	Scene
Hawaii	0	0	1	0	Lake
Alaska	0	0	1	0	Lake
Mexico	1	0	0	0	Beach
California	0	0	0	1	Mountains
Maine	0	0	1	0	Lake
Home	0	1	0	0	Home

Vacation Example, Strategies for Big Designs

Place	Bed & Breakfast	Cabin	Home	Hotel	Lodge	0	999	1249	1499	Price
Hawaii	0	0	0	1	Hotel	0	0	1	0	1249
Alaska	1	0	0	0	Bed & Breakfast	0	0	0	1	1499
Mexico	0	0	0	1	Hotel	0	1	0	0	999
California	0	0	0	1	Hotel	0	1	0	0	999
Maine	1	0	0	0	Bed & Breakfast	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 0 for Hawaii since Hawaii has hotel lodging in this choice set. The Bed & Breakfast column has a 1 for Alaska since Alaska has Bed & Breakfast lodging in this choice set. These binary

variables will form the independent variables in the analysis.

PROC PHREG is then run in the usual way to fit the choice model.

```
proc phreg data=coded;
   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 71 to customize the output from PROC PHREG.) In the interest of space, only a few lines of the summary table are printed. It is important to check the summary table to help ensure that the data were entered correctly. The number of alternatives, number of chosen alternatives, and the number not chosen should be constant in an example like this one. We will see on page 129 in the next part of this example how to print a compact summary of the summary table. Here are the results.

Vacation Example, Strategies for Big Designs

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

Stratum	Subj	Set	Number of Alternatives	Chosen Alternatives	Not Chosen
1	1	1	6	1	5
2	1	2	6	1	5
3	1	3	6	1	5
4	1	4	6	1	5
5	1	5	6	1	5
.					
.					
.					
3595	200	13	6	1	5
3596	200	14	6	1	5
3597	200	15	6	1	5
3598	200	16	6	1	5
3599	200	17	6	1	5
3600	200	18	6	1	5
Total			21600	3600	18000

Convergence Status

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

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	12900.668	6325.661
AIC	12900.668	6347.661
SBC	12900.668	6415.736

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	6575.0076	11	<.0001
Score	5977.4609	11	<.0001
Wald	2308.1976	11	<.0001

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Hawaii	1	3.41528	0.38699	77.8830	<.0001
Alaska	1	0.50991	0.39691	1.6504	0.1989
Mexico	1	2.59644	0.38889	44.5752	<.0001
California	1	1.96867	0.39076	25.3825	<.0001
Maine	1	1.27411	0.39313	10.5036	0.0012
Home	0	0	.	.	.
0	0	0	.	.	.
999	1	3.51098	0.09241	1443.5439	<.0001
1249	1	1.32425	0.08383	249.5542	<.0001
1499	0	0	.	.	.
Beach	1	1.48358	0.06898	462.5029	<.0001
Home	0	0	.	.	.
Lake	1	0.67406	0.06123	121.1979	<.0001
Mountains	0	0	.	.	.
Bed & Breakfast	1	0.66936	0.06081	121.1516	<.0001
Cabin	1	-1.41692	0.07118	396.3019	<.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. So 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. One way to do this is to print the independent variable list and copy and edit it, removing the **Price** 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 getting rid of 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 variable **Price** is designated as an **identity** variable − a do-nothing transformation. The **identity** specification simply copies the variable **Price** as is into the output data set and adds **Price** 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;
```

The summary table is huge, so we would rather not print the entire thing, yet we would like to use it to check the data entry and processing. We could use the **nosummary** option in the PROC statement to suppress the table. Instead, we will use ODS statements to output the summary table to a data set, then we will summarize the table and print the summary. First, the **ods exclude CensoredSummary** excludes the summary table from the output listing. Then the **ods output CensoredSummary=CS** statement outputs the table to a SAS data set CS. PROC FREQ is run to list the combinations of event and censored (the number of chosen alternatives and the number that were not chosen). If data entry is correct for this study, the resulting table will have one line showing that 3600 times, one alternative was chosen and five were not chosen. The **where n(stratum)** statement is used to exclude the last line of the table, the "Total" line, where **Stratum** is missing in the output data set.

```
ods exclude CensoredSummary;
ods output  CensoredSummary=CS;

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

proc freq;
   tables event * censored / list;
   where n(stratum);
   run;
```

Here are the results.

Vacation Example, Strategies for Big Designs

The PHREG Procedure

Model Information

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

Convergence Status

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

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	12900.668	6379.323
AIC	12900.668	6399.323
SBC	12900.668	6461.210

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	6521.3450	10	<.0001
Score	5654.4615	10	<.0001
Wald	2261.0883	10	<.0001

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Hawaii	1	14.28398	0.45505	985.3388	<.0001
Alaska	1	11.45659	0.43750	685.7353	<.0001
Mexico	1	13.49398	0.44557	917.1550	<.0001
California	1	12.80977	0.44323	835.2537	<.0001
Maine	1	12.16993	0.44151	759.7996	<.0001
Price	1	-0.00747	0.0001828	1668.6766	<.0001
Beach	1	1.49144	0.06902	466.9792	<.0001
Lake	1	0.69070	0.06082	128.9826	<.0001
Mountains	0	0	.	.	.
Bed & Breakfast	1	0.68812	0.06057	129.0788	<.0001
Cabin	1	-1.37471	0.07011	384.5218	<.0001
Hotel	0	0	.	.	.

```
                Vacation Example, Strategies for Big Designs

                              The FREQ Procedure

                                                          Cumulative    Cumulative
Event    Censored    Frequency     Percent     Frequency      Percent
-----------------------------------------------------------------------------
    1           5        3600      100.00          3600       100.00
```

The results of the two different analyses are similar. The coefficients for the destinations all increase by a non-constant amount (approximately 10.6) 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 = 6521.3450, compared to the previous value of 6575.0076 (bigger values mean better fit), because price has one fewer parameter.

Quadratic Price Effect

In a previous example 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. 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). Then 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 nosummary;
   model c*c(2) = &_trgind / ties=breslow;
   strata subj set;
   run;
```

This step produced the following results.

Vacation Example, Strategies for Big Designs

The PHREG Procedure

Model Information

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

Convergence Status

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

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	12900.668	6325.661
AIC	12900.668	6347.661
SBC	12900.668	6415.736

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	6575.0076	11	<.0001
Score	5977.4609	11	<.0001
Wald	2308.1976	11	<.0001

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Hawaii	1	4.73953	0.38552	151.1393	<.0001
Alaska	1	1.83416	0.39258	21.8288	<.0001
Mexico	1	3.92069	0.38610	103.1154	<.0001
California	1	3.29292	0.38608	72.7475	<.0001
Maine	1	2.59837	0.38913	44.5871	<.0001
Price 1	1	-1.75549	0.04620	1443.5439	<.0001
Price 2	1	0.43124	0.05933	52.8369	<.0001
Beach	1	1.48358	0.06898	462.5029	<.0001
Lake	1	0.67406	0.06123	121.1979	<.0001
Mountains	0	0	.	.	.
Bed & Breakfast	1	0.66936	0.06081	121.1516	<.0001
Cabin	1	-1.41692	0.07118	396.3019	<.0001
Hotel	0	0	.	.	.

The fit is exactly the same as when price was treated as qualitative, Chi-Square = 6575.0076. 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 1.32425. The coefficients for the factors after price are unchanged. The part-worth utility for \$999 is $-1.75549 \times (999-1249)/250 + 0.43124 \times ((999-1249)/250)^2 = 2.18673$, the part-worth utility for \$1249 is $-1.75549 \times (1249-1249)/250 + 0.43124 \times ((1249-1249)/250)^2 =$

0, and the part-worth utility for \$1499 is $-1.75549 \times (1499-1249)/250 + 0.43124 \times ((1499-1249)/250)^2 = -1.32425$, which differ from the coefficients when price was treated as qualitative, by a constant -1.32425.

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 128, some of which is 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. Then 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, Strategies for Big Designs

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	-1	1	1	0	-1
4	0	0	0	1	0	-1	1	-1	-1	-1
5	0	0	0	0	1	0	0	0	1	1
6	0	0	0	0	0	0	0	0	0	0

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

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

Vacation Example, Strategies for Big Designs

The PHREG Procedure

Model Information

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

Convergence Status

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

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	12900.668	6325.661
AIC	12900.668	6347.661
SBC	12900.668	6415.736

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	6575.0076	11	<.0001
Score	5977.4609	11	<.0001
Wald	2308.1976	11	<.0001

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Hawaii	1	5.20955	0.38305	184.9688	<.0001
Alaska	1	2.30419	0.38868	35.1442	<.0001
Mexico	1	4.39072	0.38328	131.2299	<.0001
California	1	3.76294	0.38324	96.4090	<.0001
Maine	1	3.06839	0.38657	63.0031	<.0001
Price 1	1	-1.75549	0.04620	1443.5439	<.0001
Price 2	1	0.43124	0.05933	52.8369	<.0001
Beach	1	0.76436	0.03857	392.6592	<.0001
Lake	1	-0.04515	0.03393	1.7708	0.1833
Bed & Breakfast	1	0.91854	0.03916	550.2758	<.0001
Cabin	1	-1.16773	0.04460	685.4980	<.0001

It is instructive to compare the results of this analysis to the previous analysis on page 131. First, model fit and chi-square statistics are the same indicating the models are equivalent. The coefficients for the destinations differ by a constant 0.47002, the price coefficients are the same, the scenery coefficients differ by a constant -0.71921, and the lodging coefficients differ by a constant 0.24919. Notice that $0.47002 + 0.24919 + -0.71921 = 0$, so the utility for each alternative is unchanged by the different but equivalent codings.

Alternative-Specific Effects

In all of the analyses presented in this example so far, 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, lets 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 122 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 122. The %MKTMERGE macro differs from page 124 in that instead of assigning labels and recoding price in a separate DATA step, we instead 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, Strategies for Big Designs

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

Notice that the attributes for the constant alternative are all missing. Next, we code with PROC TRANSREG.

Since we are using missing values for the constant alternative, we must specify the **nozeroconstant** 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;
    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, Strategies for Big Designs

Obs	Hawaii	Alaska	Mexico	California	Maine	Alaska, 999	Alaska, 1249	Alaska, 1499	California, 999
1	1	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	1	0
3	0	0	1	0	0	0	0	0	0
4	0	0	0	1	0	0	0	0	1
5	0	0	0	0	1	0	0	0	0
6	0	0	0	0	0	0	0	0	0

Obs	California, 1249	California, 1499	Hawaii, 999	Hawaii, 1249	Hawaii, 1499	Maine, 999	Maine, 1249	Maine, 1499
1	0	0	0	1	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	1	0
6	0	0	0	0	0	0	0	0

Obs	Mexico, 999	Mexico, 1249	Mexico, 1499	Alaska, Beach	Alaska, Lake	Alaska, Mountains	California, Beach	California, Lake
1	0	0	0	0	0	0	0	0
2	0	0	0	0	1	0	0	0
3	1	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0

Obs	California, Mountains	Hawaii, Beach	Hawaii, Lake	Hawaii, Mountains	Maine, Beach	Maine, Lake	Maine, Mountains	Mexico, Beach
1	0	0	1	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	1
4	1	0	0	0	0	0	0	0
5	0	0	0	0	0	1	0	0
6	0	0	0	0	0	0	0	0

Obs	Mexico, Lake	Mexico, Mountains	Alaska, Bed & Breakfast	Alaska, Cabin	Alaska, Hotel	California, Bed & Breakfast	California, Cabin
1	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0
3	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0

Obs	California, Hotel	Hawaii, Bed & Breakfast	Hawaii, Cabin	Hawaii, Hotel	Maine, Bed & Breakfast	Maine, Cabin	Maine, Hotel	Mexico, Bed & Breakfast
1	0	0	0	1	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0
5	0	0	0	0	1	0	0	0
6	0	0	0	0	0	0	0	0

Obs	Mexico, Cabin	Mexico, Hotel	Place	Price	Scene	Lodge	Subj	Set	Form	c
1	0	0	Hawaii	1249	Lake	Hotel	1	1	1	2
2	0	0	Alaska	1499	Lake	Bed & Breakfast	1	1	1	2
3	0	1	Mexico	999	Beach	Hotel	1	1	1	1
4	0	0	California	999	Mountains	Hotel	1	1	1	2
5	0	0	Maine	1249	Lake	Bed & Breakfast	1	1	1	2
6	0	0		.			1	1	1	2

Analysis proceeds by running PROC PHREG as before.

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

The PHREG Procedure

Model Information

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

Convergence Status

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

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	12900.668	6296.939
AIC	12900.668	6366.939
SBC	12900.668	6583.543

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	6603.7295	35	<.0001
Score	6626.5116	35	<.0001
Wald	2272.7813	35	<.0001

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Hawaii	1	3.28017	0.40142	66.7713	<.0001
Alaska	1	0.88221	0.51968	2.8819	0.0896
Mexico	1	2.65936	0.42318	39.4917	<.0001
California	1	1.92812	0.42686	20.4034	<.0001
Maine	1	1.18481	0.48444	5.9815	0.0145
Alaska, 999	1	3.45444	0.26284	172.7323	<.0001
Alaska, 1249	1	0.73751	0.38764	3.6198	0.0571
Alaska, 1499	0	0	.	.	.
California, 999	1	3.63995	0.21968	274.5414	<.0001
California, 1249	1	1.46294	0.21528	46.1777	<.0001
California, 1499	0	0	.	.	.
Hawaii, 999	1	3.45605	0.13258	679.5428	<.0001
Hawaii, 1249	1	1.42337	0.12282	134.3047	<.0001
Hawaii, 1499	0	0	.	.	.
Maine, 999	1	3.75491	0.25232	221.4693	<.0001
Maine, 1249	1	1.20395	0.29647	16.4910	<.0001
Maine, 1499	0	0	.	.	.
Mexico, 999	1	3.37240	0.16292	428.4889	<.0001
Mexico, 1249	1	1.22668	0.17987	46.5120	<.0001
Mexico, 1499	0	0	.	.	.
Alaska, Beach	1	1.41262	0.21525	43.0704	<.0001
Alaska, Lake	1	0.29267	0.22784	1.6501	0.1989
Alaska, Mountains	0	0	.	.	.
California, Beach	1	1.59118	0.15540	104.8399	<.0001
California, Lake	1	0.44899	0.16023	7.8518	0.0051
California, Mountains	0	0	.	.	.
Hawaii, Beach	1	1.44565	0.12351	137.0010	<.0001
Hawaii, Lake	1	0.85820	0.13430	40.8340	<.0001
Hawaii, Mountains	0	0	.	.	.

Maine, Beach	1	1.43838	0.20887	47.4229	<.0001
Maine, Lake	1	0.64057	0.22338	8.2232	0.0041
Maine, Mountains	0	0	.	.	.
Mexico, Beach	1	1.54452	0.14136	119.3759	<.0001
Mexico, Lake	1	0.73442	0.12926	32.2801	<.0001
Mexico, Mountains	0	0	.	.	.
Alaska, Bed & Breakfast	1	0.40383	0.19122	4.4600	0.0347
Alaska, Cabin	1	-1.78198	0.31993	31.0240	<.0001
Alaska, Hotel	0	0	.	.	.
California, Bed & Breakfast	1	0.62159	0.13647	20.7470	<.0001
California, Cabin	1	-1.61786	0.19845	66.4643	<.0001
California, Hotel	0	0	.	.	.
Hawaii, Bed & Breakfast	1	0.65588	0.12580	27.1844	<.0001
Hawaii, Cabin	1	-1.23939	0.11982	106.9906	<.0001
Hawaii, Hotel	0	0	.	.	.
Maine, Bed & Breakfast	1	0.65929	0.17686	13.8963	0.0002
Maine, Cabin	1	-1.54305	0.20860	54.7205	<.0001
Maine, Hotel	0	0	.	.	.
Mexico, Bed & Breakfast	1	0.70823	0.13084	29.3001	<.0001
Mexico, Cabin	1	-1.45232	0.13758	111.4373	<.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. The null hypothesis that the two models are not significantly different can be tested by comparing the likelihoods for the two models. The difference between two $-2\log(\mathcal{L}_C)$'s (the number reported under "With Covariates" in the output) has a chi-square distribution. The degrees of freedom for the test is the difference between the two *df* for the two likelihoods. The difference $6603.7295 - 6575.0076 = 28.7219$ is distributed χ^2 with $35 - 11 = 24$ *df* ($p < 0.23$). So this more complicated model does not account for significantly more variance than the simpler model.

PROC FACTEX Code Generated by the %MKTDES Macro

In the first part of this example, starting on page 108, we used the %MKTDES macro as follows:

```
%mktdes(factors=x1-x15=3, n=36, procopts=seed=7654321)
```

We did not show the PROC FACTEX code actually generated by the macro at that point because it is more complex than is really necessary for a problem this simple. This is because the macro uses an algorithm well-suited for much more complicated problems such as models with interactions and a mix of levels. The macro makes no attempt to simplify the code it produces for simple examples such as this one. The %MKTDES macro generated the following PROC FACTEX code:

```
proc factex;
   factors _1-_15 / nlev=3;
   size design=min;
   model estimate=(
      _1
      _2
      _3
      _4
      _5
      _6
      _7
      _8
      _9
      _10
      _11
      _12
      _13
      _14
      _15
      );
   output out=Cand1(drop=_:)
      [_1]=x1 nvals=(1 2 3)
      [_2]=x2 nvals=(1 2 3)
      [_3]=x3 nvals=(1 2 3)
      [_4]=x4 nvals=(1 2 3)
      [_5]=x5 nvals=(1 2 3)
      [_6]=x6 nvals=(1 2 3)
      [_7]=x7 nvals=(1 2 3)
      [_8]=x8 nvals=(1 2 3)
      [_9]=x9 nvals=(1 2 3)
      [_10]=x10 nvals=(1 2 3)
      [_11]=x11 nvals=(1 2 3)
      [_12]=x12 nvals=(1 2 3)
      [_13]=x13 nvals=(1 2 3)
      [_14]=x14 nvals=(1 2 3)
      [_15]=x15 nvals=(1 2 3)
      ;
   run; quit;
```

The result of this step is a design with 15 factors, **x1-x15**, each with values 1, 2, and 3. This can be seen by the **x1 nvals=(1 2 3)** through **x15 nvals=(1 2 3)** specifications. Internally, PROC FACTEX creates 15 factors, **_1 - _15**, which it renames to **x1-x15** when the output data set CAND1 is created and the level values are assigned. The statement **factors _1-_15 / nlev=3** specifies that we are creating a design with 15 three-level factors. The statement **size design=min** specifies that PROC FACTEX should create a minimum sized design for this problem. The statement **model estimate=(...)** specifies the effects that must be estimated, in this case the main effects. Since only main effects are requested, PROC FACTEX creates a resolution III design. In this example, there is a one-to-one mapping from the underscore variables to the final "**x**" variables. In more complicated examples, each factor may be composed of the main-effects and interactions of two or more underscore variables. Also, the **nvals=** option can be used to create factors with differing numbers of levels.

We stated on page 109 that the preceding PROC FACTEX code is almost the same as the simpler PROC FACTEX code shown next.

```
proc factex;
   factors x1-x15 / nlev=3;
   size design=min;
   model res=3;
   output out=Cand1;
   run; quit;
```

Here is some code that is exactly equivalent to the PROC FACTEX code that the macro generates.

```
proc factex;
   factors x1-x15 / nlev=3;
   size design=min;
   model res=3;
   output out=Cand1
      x1 nvals=(1 2 3)
      x2 nvals=(1 2 3)
      x3 nvals=(1 2 3)
      x4 nvals=(1 2 3)
      x5 nvals=(1 2 3)
      x6 nvals=(1 2 3)
      x7 nvals=(1 2 3)
      x8 nvals=(1 2 3)
      x9 nvals=(1 2 3)
      x10 nvals=(1 2 3)
      x11 nvals=(1 2 3)
      x12 nvals=(1 2 3)
      x13 nvals=(1 2 3)
      x14 nvals=(1 2 3)
      x15 nvals=(1 2 3)
      ;
   run; quit;
```

The **nvals=** option is used to code each factor with the numeric values 1, 2, and 3 instead of the default -1, 0, and 1. In this example, the statement **model res=3** is equivalent to **model estimate=(x1-x15)**. Both create a main-effects only or resolution III design.

The thing that distinguishes the code that the macro generates from the code we might have written from scratch is the use of *pseudo-factors* in the macro. The factors **_1-_15**, are pseudo-factors and are not of direct interest. They are used to create the *derived factors* **x1 - x15**, the three-level factors of interest. This example is a simple problem. All factors have the same numbers of levels, the number of levels is a power of a prime, and there are no interactions. In a simple example such as this, pseudo-factors are not needed. However, pseudo-factors are needed for more complicated problems, so it is easier for the macro to simply always use pseudo-factors. For this problem, the %MKTDES macro generates PROC FACTEX code that creates a three-level pseudo-factor for each factor of interest, renames the pseudo-factor creating the specified derived factor, and maps the values to 1, 2, and 3. The mapping of pseudo-factors to derived factors and original levels (-1, 0, 1) to actual levels (1, 2, 3) is completely one to one.

Vacation Example, Big Designs and Asymmetry

A researcher is interested in studying choice of vacation destinations. 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. Here is a summary of the design.

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	Mexico	Side Trip	Yes, No
X17	Hawaii	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
X17		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
X16		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

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.

This example illustrates many of the techniques that are used in the search for a good design including

- creating a candidate set with asymmetry by coding down,
- creating a candidate set with asymmetry by using pseudo-factors,
- using a tabled design to create a candidate set,
- efficiently blocking an existing design,
- generating artificial data to test the design.
- aggregating the data.

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 261.) As before, we specify a list containing the number of levels of each factor.

```
title 'Vacation Example with Asymmetry';

%mktruns( 3 3 3 3 3 3 3 3 3 3 3 3 3 3 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

Some Reasonable Design Sizes (Saturated=34)	Violations	Cannot Be Divided By
72	0	
144	0	
216	0	
36	2	8
108	2	8
180	2	8
54	18	4 12 8
90	18	4 12 8
126	18	4 12 8
162	18	4 12 8

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. However, 36 is pretty good. It 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, and 36 is much more manageable than 72, so again we chose 36 choice sets. 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. We will look at several ways of approaching this problem. First, we will run the %MKTDES macro specifying **factors=x1-x13 x15=3 x14=4 x16 x17=2** which indicates that **x1-x13 x15** are three-level-factors, **x14** is a four-level-factor, and **x16 x17** are two-level-factors. We also specify a random number seed.

```
%mktdes(factors=x1-x13 x15=3 x14=4 x16 x17=2, n=36,
        procopts=seed=7654321)
```

The macro generates the following PROC FACTEX code.

```
proc factex;
   factors _1-_32 / nlev=2;
   size design=min;
   model estimate=(
      _1|_2
      _3|_4
      _5|_6
      _7|_8
      _9|_10
      _11|_12
      _13|_14
      _15|_16
      _17|_18
      _19|_20
      _21|_22
      _23|_24
      _25|_26
      _27|_28
      _29|_30
      _31
      _32
      );
   output out=Cand1(drop=_:)
      [_1 _2]=x1 nvals=(1 2 3 1)
      [_3 _4]=x2 nvals=(1 2 3 1)
      [_5 _6]=x3 nvals=(1 2 3 1)
      [_7 _8]=x4 nvals=(1 2 3 1)
      [_9 _10]=x5 nvals=(1 2 3 1)
      [_11 _12]=x6 nvals=(1 2 3 1)
      [_13 _14]=x7 nvals=(1 2 3 1)
      [_15 _16]=x8 nvals=(1 2 3 1)
      [_17 _18]=x9 nvals=(1 2 3 1)
      [_19 _20]=x10 nvals=(1 2 3 1)
      [_21 _22]=x11 nvals=(1 2 3 1)
      [_23 _24]=x12 nvals=(1 2 3 1)
      [_25 _26]=x13 nvals=(1 2 3 1)
      [_27 _28]=x15 nvals=(1 2 3 1)
      [_29 _30]=x14 nvals=(1 to 4)
      [_31]=x16 nvals=(1 2)
      [_32]=x17 nvals=(1 2)
      ;
   run; quit;
```

The statement **factors _1-_32 / nlev=2** generates a design with 32 two-level factors. The statement **size design=min** requests the smallest design in which all the effects specified in the **model** statement are

estimable. The first factor we want from this design is **x1** with three levels. We see in the **model** statement that **_1 | _2** must be estimable, that is the **_1** main effect, the **_2** main effect, and the **_1*_2** interaction must be estimable. From the 1 *df* for **_1**, the 1 *df* for **_2**, and the 1 *df* for **_1*_2**, we can construct the three-level factor **x1** with $3 - 1$ *df* and one *df* to spare. The **output** statement specification **[_1 _2]=x1 nvals=(1 2 3 1)** constructs the factor **x1** from the values of **_1** and **_2**. The (1,1), (1,2), (2,1), (2,2) values are coded 1, 2, 3, and 1 creating a three level factor. The three-level factors **x2** through **x13** and **x15** are created in the same manner from **_2** through **_28**. The factor **x14** has four levels, and is created from **_29** and **_30** and their interaction. The factors **x16** and **x17** have two levels, and are created from **_31** and **_32**, respectively.

The factors **_1-_32** are pseudo-factors and not of direct interest. They are used to create the *derived factors* **x1 - x17**, the two-level, the three-level, and the four-level factors of interest. This example constructed three-level factors from 2 two-level factors and their interaction by *coding down*. Four pairs of values (1,1), (1,2), (2,1), (2,2) could become four levels 1, 2, 3, 4, but instead are mapped to 1, 2, 3, 1 by coding down. The value 1 maps to 1, the value 2 maps to 2, the value 3 maps to 3, and the value 4 maps to 1. This creates a candidate set with imbalance (twice as many ones as twos or threes), but we hope that PROC OPTEX will be able to find a reasonably balanced design.

The macro generates the following PROC OPTEX code.

```
proc optex data=Cand1 seed=7654321;
   class x1-x13 x15 x14 x16 x17 / param=orthref;
   model x1-x13 x15 x14 x16 x17;
   generate n=36 iter=10 keep=5 method=m_federov;
   output out=Design;
   run; quit;
```

The **class** and **model** statements contain the factors in the same order they appeared in the **factors=x1-x13 x15=3 x14=4 x16 x17=2** option. The other aspects of the PROC OPTEX syntax are no different than we saw in previous examples.

The macro finds a design with D-efficiency = 76.0622. Since the design is based on two-level factors, we can try larger candidate sets, each time doubling the size.

```
%mktdes(factors=x1-x13 x15=3 x14=4 x16 x17=2, n=36, size=128,
        procopts=seed=7654321)
%mktdes(factors=x1-x13 x15=3 x14=4 x16 x17=2, n=36, size=256,
        procopts=seed=7654321)
%mktdes(factors=x1-x13 x15=3 x14=4 x16 x17=2, n=36, size=512,
        procopts=seed=7654321)
%mktdes(factors=x1-x13 x15=3 x14=4 x16 x17=2, n=36, size=1024,
        procopts=seed=7654321)
%mktdes(factors=x1-x13 x15=3 x14=4 x16 x17=2, n=36, size=2048,
        procopts=seed=7654321)
%mktdes(factors=x1-x13 x15=3 x14=4 x16 x17=2, n=36, size=4096,
        procopts=seed=7654321)
```

The slowest of these steps with **size=4096**, takes over 1.5 minutes and produced a design with D-efficiency = 83.1150.

Since we have a mix of factor levels, would could also try basing the design on three-level factors. The **nlev=3** option is specified to use three-level pseudo-factors. The sizes now increase by a factor of three. The **factors=** and **seed=** options are the same.

```
%mktdes(factors=x1-x13 x15=3 x14=4 x16 x17=2, nlev=3, n=36,
        procopts=seed=7654321)
%mktdes(factors=x1-x13 x15=3 x14=4 x16 x17=2, nlev=3, n=36, size=243,
        procopts=seed=7654321)
%mktdes(factors=x1-x13 x15=3 x14=4 x16 x17=2, nlev=3, n=36, size=729,
        procopts=seed=7654321)
%mktdes(factors=x1-x13 x15=3 x14=4 x16 x17=2, nlev=3, n=36, size=2187,
        procopts=seed=7654321)
```

The first of the preceding macro invocations generated the following PROC FACTEX code.

```
proc factex;
   factors _1-_18 / nlev=3;
   size design=min;
   model estimate=(
      _1
      _2
      _3
      _4
      _5
      _6
      _7
      _8
      _9
      _10
      _11
      _12
      _13
      _14
      _15|_16
      _17
      _18
      );
   output out=Cand1(drop=_:)
      [_1]=x1 nvals=(1 2 3)
      [_2]=x2 nvals=(1 2 3)
      [_3]=x3 nvals=(1 2 3)
      [_4]=x4 nvals=(1 2 3)
      [_5]=x5 nvals=(1 2 3)
      [_6]=x6 nvals=(1 2 3)
      [_7]=x7 nvals=(1 2 3)
      [_8]=x8 nvals=(1 2 3)
      [_9]=x9 nvals=(1 2 3)
      [_10]=x10 nvals=(1 2 3)
      [_11]=x11 nvals=(1 2 3)
      [_12]=x12 nvals=(1 2 3)
      [_13]=x13 nvals=(1 2 3)
      [_14]=x15 nvals=(1 2 3)
      [_15 _16]=x14 nvals=(1 to 4 1 to 4 1)
      [_17]=x16 nvals=(1 2 1)
      [_18]=x17 nvals=(1 2 1)
      ;
   run; quit;
```

Now the design is generated from 18 pseudo-factors. The three-level factors **x1-x13 x15** are created directly from **_1-_14**. The four-level factor **x14** is created from **_15** and **_16** and their interaction. The nine values are coded down into: 1 2 3 4 1 2 3 4 1. The two-level factors **x16 x17** are created from **_17 _18** coding down the three values to 1, 2, 1. The last of the preceding macro invocations with **size=2187** produces a design with D-efficiency = 84.6249 and took about 40 seconds.

We could also try a multi-step process. The first step, designated by **step=1**, runs only PROC FACTEX and creates a candidate set for only the three-level factors. The random number seed is not specified since PROC OPTEX is not run in this step. The next step, **step=2**, generates a candidate set for the two-level and four-level factors and crosses it with the candidate set from step 1. Then PROC OPTEX is run on the resulting candidate set. The advantage of doing this is the candidate set is balanced and orthogonal.

```
%mktdes(factors=x1-x13 x15=3,    run=factex,       step=1)
%mktdes(factors=x14=4 x16 x17=2, run=factex optex, step=2, n=36,
        procopts=seed=7654321)
```

The first step generated the following PROC FACTEX code.

```
proc factex;
   factors _1-_14 / nlev=3;
   size design=min;
   model estimate=(
      _1
      _2
      _3
      _4
      _5
      _6
      _7
      _8
      _9
      _10
      _11
      _12
      _13
      _14
      );
   output out=Cand1(drop=_:)
      [_1]=x1 nvals=(1 2 3)
      [_2]=x2 nvals=(1 2 3)
      [_3]=x3 nvals=(1 2 3)
      [_4]=x4 nvals=(1 2 3)
      [_5]=x5 nvals=(1 2 3)
      [_6]=x6 nvals=(1 2 3)
      [_7]=x7 nvals=(1 2 3)
      [_8]=x8 nvals=(1 2 3)
      [_9]=x9 nvals=(1 2 3)
      [_10]=x10 nvals=(1 2 3)
      [_11]=x11 nvals=(1 2 3)
      [_12]=x12 nvals=(1 2 3)
      [_13]=x13 nvals=(1 2 3)
      [_14]=x15 nvals=(1 2 3)
      ;
   run; quit;
```

This is standard PROC FACTEX for three-level factors. There are no interactions or coding down. This candidate set has 81 runs. The second macro generates the following PROC FACTEX code.

```
proc factex;
   factors _1-_4 / nlev=2;
   size design=min;
   model estimate=(
      _1|_2
      _3
      _4
      );
   output out=Cand2(drop=_:)
      pointrep=Cand1
      [_1 _2]=x14 nvals=(1 to 4)
      [_3]=x16 nvals=(1 2)
      [_4]=x17 nvals=(1 2)
      ;
   run; quit;
```

This PROC FACTEX step generates a design in CAND2 from the previously generated design for the three-level factors CAND1 and another design it is generating for the two-level and four-level factors. The only aspect of this code that is new is the **`pointrep=Cand1`** option. PROC FACTEX creates a design with **`x14 x16 x17`**

in it in 8 runs and crosses it with the 81 runs in the CAND1 data set creating a candidate set with $8 \times 81 = 648$ runs. Each run in the three-level factor design (CAND1) appears with each run in the two-level factor design.

The macro generated the following PROC OPTEX code. The only difference in the PROC OPTEX code from the one-step method is now the **class** and **model** statements have the factors grouped by step.

```
proc optex data=Cand2 seed=7654321;
   class
      x1-x13 x15
      x14 x16 x17
      / param=orthref;
   model
      x1-x13 x15
      x14 x16 x17
      ;
   generate n=36 iter=10 keep=5 method=m_federov;
   output out=Design;
   run; quit;
```

The resulting design had a D-efficiency of 82.4738. Like before, we can try this again with larger sizes.

```
%mktdes(factors=x1-x13 x15=3,    run=factex,       step=1, size=243)
%mktdes(factors=x14=4 x16 x17=2, run=factex optex, step=2, size=16, n=36,
        procopts=seed=7654321)
```

This approach creates a candidate set with $16 \times 243 = 3888$ runs. The design has D-efficiency = 86.1054 and took about 1.5 minutes.

Using a Tabled Design as a Candidate Set

Another alternative for creating a candidate set is to use a standard tabled design, such as the L_{36}, instead of using PROC PLAN or PROC FACTEX. The L_{36} has 11 two-level factors and 12 three-level factors in 36 runs. In this example, we cross part of this design with a full-factorial design with 2 three-level factors and 1 four-level factor, creating a candidate set in $36 \times 3 \times 3 \times 4 = 1296$ runs. The resulting candidate set is orthogonal and balanced.

```
data l36;   /* a1-a11 - two-level, b1-b12 - three-level */
   input (a1-a11 b1-b12) (23*1.) @@;
   if mod(_n_, 3) = 0 then input;
   datalines;
11111111112223322321121111111111333113313223111111111111111221121331
21211122212222212121223212111222123333232323312121112222121111313131 12
22121112221223221312232222121112221331332123313221211122211121132311 21
12212111222221123223322122121112223322313311331221211122211331211 2211
21222121112223112233121121221211122312233112322212212111221233112 23133
22122121112231313111132221221211123121212222132212212111212323323 33321
22212212111232211213311222122121111313322321122222122121111211331322 33
12221221211233131122121122212212111311212233232122212212111223233113 13
11222122121213132211333112221221321213322111112222122121132321133222
11122212212213223133113111222122123213312112211112222122121321121323 32
21112222122121131133232321112222122132212211313121112222122113323322 12
12111222122212333323231121112222122323111131312121112222122131222212 123
;
```

```
data cand1(drop=a: b: i);
   set l36;
   x16 = a1; x17 = a2;
   array x[12]; array b[12];
   do i = 1 to 12; x[i] = b[i]; end;
   do x13 = 1 to 3;
      do x14 = 1 to 4;
         do x15 = 1 to 3;
            output;
            end;
         end;
      end;
   run;
```

For each of the 36 input observations, the 36 combinations of **x13**, **x14**, and **x15** are created and output, creating a total of 1296 possible choice sets in the candidate set. Next, we use the %MKTDES macro to run only PROC OPTEX. It prints some generated PROC FACTEX code even though it does not run it.

```
%mktdes(factors=x1-x13 x15=3 x14=4 x16 x17=2,
        run=optex, n=36, procopts=seed=7654321)
```

This step took 14 seconds and produced the following results.

Vacation Example with Asymmetry

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	98.8874	97.5943	95.3827	0.9837
2	98.8874	97.5943	95.3827	0.9837
3	98.4728	96.6825	92.1954	0.9884
4	98.2861	96.3742	90.7878	0.9899
5	98.2861	96.1470	91.7095	0.9911

The best design has an efficiency of 98.8874, which is quite a bit better than the previous high of 86.1054. While there is no guarantee that this approach will be better than other approaches, these results are not surprising. The L_{36} is a very good, specialized design, and crossing it with a full-factorial design does nothing to diminish its goodness. Also, the size of the candidate set, 1296, is very reasonable, so we would not expect search time or local optima to be serious problems.

Ensuring that Certain Key Interactions are Estimable

Next, we will ensure that certain key interactions are estimable. Specifically, it might 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 use the %MKTDES macro in two steps to find a design. Recall that when we first used a two-step approach, we used the following code.

```
%mktdes(factors=x1-x13 x15=3,    run=factex,       step=1)
%mktdes(factors=x14=4 x16 x17=2, run=factex optex, step=2, n=36,
        procopts=seed=7654321)
```

This time, we will add to the first step an **interact=x1*x11 x2*x12 x3*x13 x5*x15** option naming the interactions that we want to be estimable. We did not specify **x4*x14** yet since **x14** is not created in the first step. In the second step, we specify our remaining interaction, **otherint=x4*x14**. The **otherint=** option is used for interactions that are composed of factors created in different steps. Interactions that appear in **interact=** appear in the PROC FACTEX and PROC OPTEX **model** statements. In contrast, interactions that appear in **otherint=** appear only in the PROC OPTEX **model** statement. The option **n=saturated** is specified so that PROC OPTEX will create the smallest possible design. This is an easy way for you to find out what the smallest size is without doing all the parameter calculations.

```
%mktdes(factors=x1-x13 x15=3, run=factex, step=1,
        interact=x1*x11 x2*x12 x3*x13 x5*x15)
%mktdes(factors=x14=4 x16 x17=2, run=factex optex, step=2,
        otherint=x4*x14, n=saturated, procopts=seed=7654321)
```

PROC OPTEX tells us **NOTE: Number of design points set to 56.** 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. So we need at least 56 choice sets, and ideally for this design with 2, 3, and 4 level factors, we would like the number 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 choices 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. So 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 56 up to 100.

```
%mktruns(9 9 9 9 12 3 3 3 3 3 2 2, max=45)
```

Vacation Example with Asymmetry

Some Reasonable Design Sizes (Saturated=56)	Violations	Cannot Be Divided By
72	30	81 108 27
81	33	12 108 36 2 18 24 6 4
90	39	81 12 108 27 36 24 4
96	57	9 81 108 27 36 18
60	59	9 81 108 27 36 18 24
63	59	81 12 108 27 36 2 18 24 6 4
84	59	9 81 108 27 36 18 24
99	59	81 12 108 27 36 2 18 24 6 4
66	61	9 81 12 108 27 36 18 24 4
78	61	9 81 12 108 27 36 18 24 4

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.

We will run the %MKTDES macro again with **n=60** specified.

```
%mktdes(factors=x1-x13 x15=3, run=factex, step=1,
        interact=x1*x11 x2*x12 x3*x13 x5*x15)
%mktdes(factors=x14=4 x16 x17=2, run=factex optex, step=2,
        otherint=x4*x14, n=60, procopts=seed=7654321)
```

The candidate set has $81 \times 8 = 648$ candidates, and the D-efficiency of the best design is 80.3563. Since making a candidate set six times bigger should not be a problem, we can run again with **size=243** and **size=16**.

```
%mktdes(factors=x1-x13 x15=3, run=factex, step=1, size=243,
        interact=x1*x11 x2*x12 x3*x13 x5*x15)
%mktdes(factors=x14=4 x16 x17=2, run=factex optex, step=2, size=16,
        otherint=x4*x14, n=60, procopts=seed=7654321)
```

The candidate set has $243 \times 16 = 3888$ candidates, and the D-efficiency of the best design is 77.4934. Since this is worse than we saw previously, we could try again with **size=243** and **size=8**.

```
%mktdes(factors=x1-x13 x15=3, run=factex, step=1, size=243,
        interact=x1*x11 x2*x12 x3*x13 x5*x15)
%mktdes(factors=x14=4 x16 x17=2, run=factex optex, step=2, size=8,
        otherint=x4*x14, n=60, procopts=seed=7654321)
```

The candidate set has $243 \times 8 = 1944$ candidates, and the D-efficiency of the best design is 76.3911. Since this is worse than we saw previously, we could try again with **size=81** and **size=16**.

```
%mktdes(factors=x1-x13 x15=3, run=factex, step=1, size=81,
        interact=x1*x11 x2*x12 x3*x13 x5*x15)
%mktdes(factors=x14=4 x16 x17=2, run=factex optex, step=2, size=16,
        otherint=x4*x14, n=60, procopts=seed=7654321)
```

The candidate set has $81 \times 16 = 1296$ candidates, and the D-efficiency of the best design is 80.8125. This is better than we saw previously. We could try again with more iterations.

```
%mktdes(factors=x1-x13 x15=3, run=factex, step=1, size=81,
        interact=x1*x11 x2*x12 x3*x13 x5*x15)
%mktdes(factors=x14=4 x16 x17=2, run=factex optex, step=2, size=16, iter=20,
        otherint=x4*x14, n=60, procopts=seed=072555)
```

This gave us a small gain in D-efficiency, 81.3949. This step took 2 minutes and 15 seconds.

Examining the Design

We can use PROC SUMMARY and PROC FREQ as a first step in evaluating the goodness of this design.

```
proc summary data=design;
   class _all_;
   ways 1 17;
   output out=sum;
   run;

proc print; by _type_; run;

proc freq;
   tables x1*x11 x2*x12 x3*x13 x4*x14 x5*x15;
   run;
```

```
                Vacation Example, Strategies for Big Designs

---------------------------------- _TYPE_=1 ----------------------------------

     Obs x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x15 x14 x16 x17 _FREQ_

       1  .  .  .  .  .  .  .  .  .  .   .   .   .   .   .   .   1    30
       2  .  .  .  .  .  .  .  .  .  .   .   .   .   .   .   .   2    30

---------------------------------- _TYPE_=2 ----------------------------------

     Obs x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x15 x14 x16 x17 _FREQ_

       3  .  .  .  .  .  .  .  .  .  .   .   .   .   .   .   1   .    31
       4  .  .  .  .  .  .  .  .  .  .   .   .   .   .   .   2   .    29

---------------------------------- _TYPE_=4 ----------------------------------

     Obs x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x15 x14 x16 x17 _FREQ_

       5  .  .  .  .  .  .  .  .  .  .   .   .   .   .   1   .   .    17
       6  .  .  .  .  .  .  .  .  .  .   .   .   .   .   2   .   .    14
       7  .  .  .  .  .  .  .  .  .  .   .   .   .   .   3   .   .    15
       8  .  .  .  .  .  .  .  .  .  .   .   .   .   .   4   .   .    14

---------------------------------- _TYPE_=8 ----------------------------------

     Obs x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x15 x14 x16 x17 _FREQ_

       9  .  .  .  .  .  .  .  .  .  .   .   .   .   1   .   .   .    19
      10  .  .  .  .  .  .  .  .  .  .   .   .   .   2   .   .   .    21
      11  .  .  .  .  .  .  .  .  .  .   .   .   .   3   .   .   .    20

--------------------------------- _TYPE_=16 ----------------------------------

     Obs x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x15 x14 x16 x17 _FREQ_

      12  .  .  .  .  .  .  .  .  .  .   .   .   1   .   .   .   .    20
      13  .  .  .  .  .  .  .  .  .  .   .   .   2   .   .   .   .    20
      14  .  .  .  .  .  .  .  .  .  .   .   .   3   .   .   .   .    20

--------------------------------- _TYPE_=32 ----------------------------------

     Obs x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x15 x14 x16 x17 _FREQ_

      15  .  .  .  .  .  .  .  .  .  .   .   1   .   .   .   .   .    21
      16  .  .  .  .  .  .  .  .  .  .   .   2   .   .   .   .   .    20
      17  .  .  .  .  .  .  .  .  .  .   .   3   .   .   .   .   .    19
```

```
---------------------------------- _TYPE_=64 ----------------------------------

     Obs x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x15 x14 x16 x17 _FREQ_

      18  .  .  .  .  .  .  .  .  .  .   1   .   .   .   .   .   .    21
      19  .  .  .  .  .  .  .  .  .  .   2   .   .   .   .   .   .    20
      20  .  .  .  .  .  .  .  .  .  .   3   .   .   .   .   .   .    19

---------------------------------- _TYPE_=128 ---------------------------------

     Obs x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x15 x14 x16 x17 _FREQ_

      21  .  .  .  .  .  .  .  .  .  1   .   .   .   .   .   .   .    20
      22  .  .  .  .  .  .  .  .  .  2   .   .   .   .   .   .   .    21
      23  .  .  .  .  .  .  .  .  .  3   .   .   .   .   .   .   .    19

---------------------------------- _TYPE_=256 ---------------------------------

     Obs x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x15 x14 x16 x17 _FREQ_

      24  .  .  .  .  .  .  .  .  1  .   .   .   .   .   .   .   .    21
      25  .  .  .  .  .  .  .  .  2  .   .   .   .   .   .   .   .    19
      26  .  .  .  .  .  .  .  .  3  .   .   .   .   .   .   .   .    20

---------------------------------- _TYPE_=512 ---------------------------------

     Obs x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x15 x14 x16 x17 _FREQ_

      27  .  .  .  .  .  .  .  1  .  .   .   .   .   .   .   .   .    21
      28  .  .  .  .  .  .  .  2  .  .   .   .   .   .   .   .   .    19
      29  .  .  .  .  .  .  .  3  .  .   .   .   .   .   .   .   .    20

---------------------------------- _TYPE_=1024 --------------------------------

     Obs x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x15 x14 x16 x17 _FREQ_

      30  .  .  .  .  .  .  1  .  .  .   .   .   .   .   .   .   .    20
      31  .  .  .  .  .  .  2  .  .  .   .   .   .   .   .   .   .    20
      32  .  .  .  .  .  .  3  .  .  .   .   .   .   .   .   .   .    20

---------------------------------- _TYPE_=2048 --------------------------------

     Obs x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x15 x14 x16 x17 _FREQ_

      33  .  .  .  .  .  1  .  .  .  .   .   .   .   .   .   .   .    19
      34  .  .  .  .  .  2  .  .  .  .   .   .   .   .   .   .   .    19
      35  .  .  .  .  .  3  .  .  .  .   .   .   .   .   .   .   .    22

---------------------------------- _TYPE_=4096 --------------------------------

     Obs x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x15 x14 x16 x17 _FREQ_

      36  .  .  .  .  1  .  .  .  .  .   .   .   .   .   .   .   .    20
      37  .  .  .  .  2  .  .  .  .  .   .   .   .   .   .   .   .    20
      38  .  .  .  .  3  .  .  .  .  .   .   .   .   .   .   .   .    20

---------------------------------- _TYPE_=8192 --------------------------------

     Obs x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x15 x14 x16 x17 _FREQ_

      39  .  .  .  1  .  .  .  .  .  .   .   .   .   .   .   .   .    20
      40  .  .  .  2  .  .  .  .  .  .   .   .   .   .   .   .   .    20
      41  .  .  .  3  .  .  .  .  .  .   .   .   .   .   .   .   .    20

---------------------------------- _TYPE_=16384 -------------------------------

     Obs x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x15 x14 x16 x17 _FREQ_

      42  .  .  1  .  .  .  .  .  .  .   .   .   .   .   .   .   .    21
      43  .  .  2  .  .  .  .  .  .  .   .   .   .   .   .   .   .    19
      44  .  .  3  .  .  .  .  .  .  .   .   .   .   .   .   .   .    20
```

---------- _TYPE_=32768 ----------

Obs	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x15	x14	x16	x17	_FREQ_
45	.	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	19
46	.	2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	21
47	.	3	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	20

---------- _TYPE_=65536 ----------

Obs	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x15	x14	x16	x17	_FREQ_
48	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	19
49	2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	19
50	3	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	22

---------- _TYPE_=131071 ----------

Obs	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x15	x14	x16	x17	_FREQ_
51	1	1	1	2	3	3	3	3	3	3	3	3	1	3	3	2	2	1
52	1	1	1	3	2	2	2	2	2	2	2	2	1	2	3	2	1	1
53	1	1	2	1	3	3	3	3	3	3	3	2	3	2	2	1	1	1
54	1	1	2	3	1	1	1	1	1	1	1	3	3	3	3	2	2	1
55	1	1	3	1	2	2	2	2	2	2	2	3	2	3	3	1	2	1
56	1	1	3	2	1	1	1	1	1	1	1	2	2	2	4	1	1	1
57	1	2	1	1	3	3	3	2	2	2	1	3	3	3	4	1	1	1
58	1	2	1	3	1	1	1	3	3	3	2	1	3	1	1	2	2	1
59	1	2	2	1	2	2	2	1	1	1	3	1	2	1	4	1	2	1
60	1	2	2	2	1	1	1	3	3	3	2	3	2	3	3	1	1	1
61	1	2	3	2	3	3	3	2	2	2	1	1	1	1	3	2	2	1
62	1	2	3	3	2	2	2	1	1	1	3	3	1	3	1	2	1	1
63	1	3	1	1	2	2	2	3	3	3	1	2	2	2	1	1	2	1
64	1	3	1	2	1	1	1	2	2	2	3	1	2	1	1	1	1	1
65	1	3	1	3	3	3	3	1	1	1	2	3	2	3	4	1	2	1
66	1	3	2	2	3	3	3	1	1	1	2	2	1	2	1	2	2	1
67	1	3	2	3	2	2	2	3	3	3	1	1	1	1	4	2	1	1
68	1	3	3	1	3	3	3	1	1	1	2	1	3	1	1	2	1	1
69	1	3	3	3	1	1	1	2	2	2	3	2	3	2	4	2	2	1
70	2	1	1	1	3	2	1	3	2	1	3	3	3	2	1	2	1	1
71	2	1	1	3	1	3	2	1	3	2	1	1	3	3	2	2	2	1
72	2	1	2	1	2	1	3	2	1	3	2	1	2	3	1	2	2	1
73	2	1	2	2	1	3	2	1	3	2	1	3	2	2	1	1	1	1
74	2	1	3	1	1	3	2	1	3	2	1	2	1	1	1	1	1	1
75	2	1	3	2	3	2	1	3	2	1	3	1	1	3	1	1	2	1
76	2	1	3	3	2	1	3	2	1	3	2	3	1	2	2	1	1	1
77	2	2	1	1	2	1	3	1	3	2	3	2	2	1	3	2	2	1
78	2	2	1	2	1	3	2	3	2	1	2	1	2	3	4	1	1	1
79	2	2	2	2	3	2	1	2	1	3	1	2	1	1	4	1	2	1
80	2	2	2	3	2	1	3	1	3	2	3	1	1	3	3	1	1	1
81	2	2	3	1	3	2	1	2	1	3	1	1	3	3	3	2	1	1
82	2	2	3	3	1	3	2	3	2	1	2	2	3	1	3	2	2	1
83	2	3	1	2	3	2	1	1	3	2	2	3	1	2	2	1	2	1
84	2	3	1	3	2	1	3	3	2	1	1	2	1	1	1	1	1	1
85	2	3	2	1	3	2	1	1	3	2	2	2	3	1	4	2	1	1
86	2	3	2	3	1	3	2	2	1	3	3	3	3	2	1	2	2	1
87	2	3	3	1	2	1	3	3	2	1	1	3	2	2	4	2	2	1
88	2	3	3	2	1	3	2	2	1	3	3	2	2	1	2	1	1	1
89	3	1	1	1	2	3	1	2	3	1	2	2	2	3	4	2	2	1
90	3	1	1	2	1	2	3	1	2	3	1	1	2	2	2	2	1	1
91	3	1	2	2	3	1	2	3	1	2	3	2	1	3	2	2	2	1
92	3	1	2	3	2	3	1	2	3	1	2	1	1	2	1	1	1	1
93	3	1	3	1	3	1	2	3	1	2	3	1	3	2	4	1	1	1
94	3	1	3	3	1	2	3	1	2	3	1	2	3	3	1	1	2	1
95	3	2	1	1	1	2	3	3	1	2	2	1	1	2	2	2	2	1
96	3	2	1	2	3	1	2	2	3	1	1	3	1	1	1	1	2	1
97	3	2	1	3	2	3	1	1	2	3	3	2	1	3	2	1	1	1
98	3	2	2	1	3	1	2	2	3	1	1	2	3	3	2	1	1	1
99	3	2	2	2	2	3	1	1	2	3	3	1	3	2	2	1	2	1
100	3	2	2	3	1	2	3	3	1	2	2	3	3	1	2	1	2	1
101	3	2	3	1	2	3	1	1	2	3	3	3	2	1	2	2	2	1
102	3	2	3	2	1	2	3	3	1	2	2	2	2	3	1	2	1	1
103	3	2	3	3	3	1	2	2	3	1	1	1	2	2	2	2	1	1
104	3	3	1	1	3	1	2	1	2	3	2	3	3	1	3	1	1	1
105	3	3	1	2	2	3	1	3	1	2	1	2	3	3	3	1	1	1
106	3	3	1	3	1	2	3	2	3	1	3	1	3	2	3	1	2	1
107	3	3	2	1	2	3	1	3	1	2	1	1	2	2	3	2	2	1
108	3	3	2	2	1	2	3	2	3	1	3	3	2	1	4	2	1	1
109	3	3	3	2	3	1	2	1	2	3	2	1	1	2	4	2	2	1
110	3	3	3	3	2	3	1	3	1	2	1	3	1	1	3	1	1	1

Table of x1 by x11

Frequency Percent Row Pct Col Pct	1	2	3	Total
1	6 10.00 31.58 28.57	7 11.67 36.84 35.00	6 10.00 31.58 31.58	19 31.67
2	7 11.67 36.84 33.33	6 10.00 31.58 30.00	6 10.00 31.58 31.58	19 31.67
3	8 13.33 36.36 38.10	7 11.67 31.82 35.00	7 11.67 31.82 36.84	22 36.67
Total	21 35.00	20 33.33	19 31.67	60 100.00

Table of x2 by x12

x2 x12

Frequency Percent Row Pct Col Pct	1	2	3	Total
1	6 10.00 31.58 28.57	7 11.67 36.84 35.00	6 10.00 31.58 31.58	19 31.67
2	9 15.00 42.86 42.86	6 10.00 28.57 30.00	6 10.00 28.57 31.58	21 35.00
3	6 10.00 30.00 28.57	7 11.67 35.00 35.00	7 11.67 35.00 36.84	20 33.33
Total	21 35.00	20 33.33	19 31.67	60 100.00

Table of x3 by x13

x3 x13

Frequency Percent Row Pct Col Pct	1	2	3	Total
1	7 11.67 33.33 35.00	7 11.67 33.33 35.00	7 11.67 33.33 35.00	21 35.00
2	6 10.00 31.58 30.00	6 10.00 31.58 30.00	7 11.67 36.84 35.00	19 31.67
3	7 11.67 35.00 35.00	7 11.67 35.00 35.00	6 10.00 30.00 30.00	20 33.33
Total	20 33.33	20 33.33	20 33.33	60 100.00

Table of x4 by x14

x4 / x14	1	2	3	4	Total
1	5	4	5	6	20
	8.33	6.67	8.33	10.00	33.33
	25.00	20.00	25.00	30.00	
	29.41	28.57	33.33	42.86	
2	6	5	4	5	20
	10.00	8.33	6.67	8.33	33.33
	30.00	25.00	20.00	25.00	
	35.29	35.71	26.67	35.71	
3	6	5	6	3	20
	10.00	8.33	10.00	5.00	33.33
	30.00	25.00	30.00	15.00	
	35.29	35.71	40.00	21.43	
Total	17	14	15	14	60
	28.33	23.33	25.00	23.33	100.00

Table of x5 by x15

x5 / x15	1	2	3	Total
1	7	7	6	20
	11.67	11.67	10.00	33.33
	35.00	35.00	30.00	
	36.84	33.33	30.00	
2	6	7	7	20
	10.00	11.67	11.67	33.33
	30.00	35.00	35.00	
	31.58	33.33	35.00	
3	6	7	7	20
	10.00	11.67	11.67	33.33
	30.00	35.00	35.00	
	31.58	33.33	35.00	
Total	19	21	20	60
	31.67	35.00	33.33	100.00

The frequencies look good and each choice set appears only once.

We can use PROC OPTEX to evaluate an existing design and print additional information such as the information matrix. The options **method=sequential** and **initdesign=** are used to evaluate an existing design. With an initial design, the sequential algorithm has nothing to do, so the efficiency and optionally other characteristics of the existing design are printed.

```
proc optex data=cand2;
   title2 'Evaluating the Efficiency of a Given Design';
   class x1-x17 / param=orthref;
   model x1-x17 x1*x11 x2*x12 x3*x13 x4*x14 x5*x15;
   generate method=sequential initdesign=design;
   examine i;
   run; quit;
```

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

Blocking an Existing Design

An existing design is blocked as follows.

```
proc optex data=design seed=72343;
   title2 'Design with Interactions, 60-Runs';
   title3 'Blocking an Existing Design';
   class x1-x17 / param=orthref;
   model x1-x17 x1*x11 x2*x12 x3*x13 x4*x14 x5*x15;
   generate initdesign=design method=sequential;
   blocks structure=(3)20 init=chain noexchange iter=1;
   output out=sasuser.blckdes;
   run; quit;
```

This step took 21 seconds. The goal is to take the observations in an existing design and optimally sort them into blocks. No swapping between the candidate set and the design is performed. The **`generate`** statement options **`initdesign=design`** and **`method=sequential`** name the design to block, DESIGN, and the sequential method since no swapping in and out is performed. The **`blocks`** statement option **`structure=`** asks for 3 blocks of size 20, **`init=chain`** specifies no swapping from the candidate set during the initialization, and **`noexchange`** specifies no swapping from the candidate set during the iterations.

The table shows the frequencies for the different frequency patterns across blocks and factors. The design is nearly balanced in most of the factors and blocks. Perfect balance is impossible for the three level factors.

Level Frequency				Frequency
8	12			2
9	11			3
10	10			1
3	8	9		1
4	7	9		1
4	8	8		1
5	6	9		3
5	7	8		15
6	6	8		11
6	7	7		10
4	4	5	7	1
4	4	6	6	2

Generating the Questionnaire

These next steps randomize SASUSER.BLCKDES, the blocked design we just created, within blocks and print the questionnaire.

```
%let m   = 6;                    /* m alternatives including constant */
%let mm1 = %eval(&m - 1);        /* m - 1                             */
%let n   = 20;                   /* number of choice sets             */
%let blocks = 3;                 /* number of blocks                  */

proc plan seed=7654321;
   factors block=&blocks ordered set=&n / noprint;
   output out=orders;
   run; quit;
```

```
data sasuser.blckdes;
   set orders;
   set = (block - 1) * &n + set;
   set sasuser.blckdes point=set;
   run;

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.blckdes;
   by block;

   p1 = 1499 + (x[11] - 2) * 250;
   p2 = 1499 + (x[12] - 2) * 250;
   p3 = 1249 + (x[13] - 2) * 250;
   p4 = 1374 + (x[14] - 2.5) * 250;
   p5 = 1249 + (x[15] - 2) * 250;

   if first.block then do;
      choice = 0;
      put _page_;
      put @50 'Form: ' block  ' Subject: ________' //;
      end;
   choice + 1;

   if ll < (19 + (x16 = 1) + (x17 = 1)) then put _page_;
   put choice 2. ') Circle your choice of '
       'vacation destinations:' /;

   do dest = 1 to &mm1;
      price = left(put(p[dest], dollar6.));
      put '    ' dest 1. ') ' dests[dest]
          +(-1) ', staying in a ' lodging[x[dest]]
          'near ' scenes[x[&mm1 + dest]] +(-1) ',' /
          +7 'with a package cost of ' price +(-1) @@;
      if dest = 3 and x16 = 1 then
         put ', and an optional visit' / +7
             'to archaeological sites for an additional $100' @@;
      else if dest = 1 and x17 = 1 then
         put ', and an optional helicopter' / +7
             'flight to an active volcano for an additional $200' @@;
      put '.' /;
      end;
   put "    &m) Stay at home this year." /;
   run;
```

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

Form: 1 Subject: _____

1) Circle your choice of vacation destinations:

   1) Hawaii, staying in a Bed & Breakfast near the Beach,
      with a package cost of $1,499.

   2) Alaska, staying in a Bed & Breakfast near a Lake,
      with a package cost of $1,499.

   3) Mexico, staying in a Hotel near the Beach,
      with a package cost of $1,499.

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

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

   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 $1,499.

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

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

   4) California, staying in a Bed & Breakfast near a Lake,
      with a package cost of $1,749.

   5) Maine, staying in a Hotel near the Beach,
      with a package cost of $1,249.

   6) Stay at home this year.

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. 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.blckdes 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 13113435444151313134   2 2 43331151114114133313   3 3 13331111451411131133
  4 1 11113133144113311114   5 2 31415434431113453111   6 3 13133111341343151311
  7 1 15413435141111133134   8 2 33115411141111413113   9 3 15153113431113131311
.
.
.
;
```

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 122 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 is similar to what was done on pages 122 and 124. One difference is now there are asymmetries in **`Price`**. For Hawaii's price, **`x11`**, we need to map 1, 2, 3 to \$1249, \$1499, \$1749; for Alaska's price, **`x12`**, we need to map 1, 2, 3 to \$1249, \$1499, \$1749; for Mexico's price, **`x13`**, we need to map 1, 2, 3 to \$999, \$1249, \$1499; for California's price, **`x14`**, we need to map 1, 2, 3, 4 to \$999, \$1249, \$1499, \$1749; for Maine's price, **`x11`**, we need to map 1, 2, 3 to \$999, \$1249, \$1499. We can simplify the problem by adding 1 to **`x11`** and **`x12`**, the factors that start at \$1249 instead of \$999, then we can use a common format to set the price.

```
data temp;
   set sasuser.blckdes;
   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	Bed & Breakfast	Beach	1499	No	1
2	1	1	1	Alaska	Bed & Breakfast	Lake	1499		2
3	1	1	1	Mexico	Hotel	Beach	1499	No	1
4	1	1	1	California	Hotel	Lake	1499		2
5	1	1	1	Maine	Cabin	Mountains	999		2
6	1	1	1		.	.	.		2
7	1	1	2	Hawaii	Hotel	Mountains	1499	No	2
8	1	1	2	Alaska	Hotel	Lake	1249		2
9	1	1	2	Mexico	Hotel	Mountains	999	No	1
10	1	1	2	California	Bed & Breakfast	Lake	1749		2
11	1	1	2	Maine	Hotel	Beach	1249		2
12	1	1	2		.	.	.		2
13	1	1	3	Hawaii	Cabin	Mountains	1499	Side Trip	1
14	1	1	3	Alaska	Bed & Breakfast	Mountains	1749		2
15	1	1	3	Mexico	Bed & Breakfast	Beach	1249	Side Trip	2
16	1	1	3	California	Bed & Breakfast	Beach	1499		2
17	1	1	3	Maine	Cabin	Beach	1499		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 a constant variable is created by the coding, it is not to 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 are to be coded. The specification **class(place / zero=none order=data)** specifies that the variable **Place** is a classification variable and requests a binary coding. The **zero=none** option specifies that one binary variable should be created for all categories. The **order=data**

option sorts the values 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 / zero=' ' 'No' separators=" ' ')** creates alternative-specific side 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 **separators=** option (explained in more detail on page 172) 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".

The **lprefix=0** option specifies that when labels are created for the binary variables, zero characters of the original variable name should be used as a prefix. This means that the labels are created only from the level values. An **output** statement names the output data set and drops variables that are not needed. Finally, the **id** statement names the additional variables that we want copied from the input to the output data set.

Vacation Example with Asymmetry

Obs	Hawaii	Alaska	Mexico	California	Maine	999	1249	1499	1749	Beach	Lake
1	1	0	0	0	0	0	0	1	0	1	0
2	0	1	0	0	0	0	0	1	0	0	1
3	0	0	1	0	0	0	0	1	0	1	0
4	0	0	0	1	0	0	0	1	0	0	1
5	0	0	0	0	1	1	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0

Obs	Mountains	Bed & Breakfast	Cabin	Hotel	Alaska Side Trip	California Side Trip	Hawaii Side Trip	Maine Side Trip	Mexico Side Trip
1	0	1	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0
4	0	0	0	1	0	0	0	0	0
5	1	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	1499	Beach	Bed & Breakfast	No	1	1	1	1
2	Alaska	1499	Lake	Bed & Breakfast	.	1	1	1	2
3	Mexico	1499	Beach	Hotel	No	1	1	1	2
4	California	1499	Lake	Hotel	.	1	1	1	2
5	Maine	999	Mountains	Cabin	.	1	1	1	2
6		.	.	.	.	1	1	1	2

The PROC PHREG specification is the same as we have seen before. (Recall that we used **%phchoice(on)** on page 71 to customize the output from PROC PHREG.)

```
ods exclude CensoredSummary;
ods output  CensoredSummary=CS;

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

```
proc freq;
   tables event * censored / list;
   where n(stratum);
   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

Convergence Status

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

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	21501.114	10600.873
AIC	21501.114	10628.873
SBC	21501.114	10722.666

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	10900.2405	14	<.0001
Score	10070.1027	14	<.0001
Wald	4295.0918	14	<.0001

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Hawaii	1	3.74345	0.24610	231.3706	<.0001
Alaska	1	-0.98353	0.29398	11.1930	0.0008
Mexico	1	2.28200	0.25137	82.4127	<.0001
California	1	1.73953	0.25052	48.2158	<.0001
Maine	1	0.96882	0.25545	14.3842	0.0001
999	1	1.98569	0.07826	643.7935	<.0001
1249	1	1.33922	0.06689	400.8888	<.0001
1499	1	0.61453	0.06335	94.0995	<.0001
1749	0	0	.	.	.
Beach	1	1.40373	0.05181	733.9404	<.0001
Lake	1	0.73358	0.05306	191.1124	<.0001
Mountains	0	0	.	.	.

Bed & Breakfast	1	0.66600	0.04334	236.1449	<.0001
Cabin	1	-1.50871	0.05375	787.7420	<.0001
Hotel	0	0	.	.	.
Alaska Side Trip	0	0	.	.	.
California Side Trip	0	0	.	.	.
Hawaii Side Trip	1	0.65428	0.06331	106.7980	<.0001
Maine Side Trip	0	0	.	.	.
Mexico Side Trip	1	0.85246	0.06886	153.2310	<.0001

Vacation Example with Asymmetry

The FREQ Procedure

Event	Censored	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	5	6000	100.00	6000	100.00

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 36,000 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 is used to store the number of times each unique observation appears in the variable **_freq_**. Then PROC PHREG is run with a **freq** statement. Now, instead of analyzing a data set with 36,000 observations and 6000 strata, we analyze a data set with 600 observations and 60 strata.

```
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 under 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
.					
.					
.					
59	3	59	600	100	500
60	3	60	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 174.

```
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 a constant variable is created by the coding, it is not to 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 are to 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 specifies that one binary variable should be created 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.

```
ods exclude CensoredSummary;
ods output  CensoredSummary=CS;

proc phreg data=coded;
   title2 'Discrete Choice with Common Price Effect';
   model c*c(2) = &_trgind / ties=breslow;
   strata subj set;
   run;

proc freq;
   tables event * censored / list;
   where n(stratum);
   run;
```

Here are the results. (Recall that we used **%phchoice(on)** on page 71 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

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
Brand 5	0	0	.	.	.
Price	1	0.14966	0.04406	11.5379	0.0007

Brand Choice Example, Multinomial Logit Model
Discrete Choice with Common Price Effect

The FREQ Procedure

Event	Censored	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	4	800	100.00	800	100.00

Alternative-Specific Price Effects

In the next steps, the multinomial logit model is coded and fit with brand by price effects. 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.

```
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 **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.

```
proc print data=coded(obs=10) label;
   title2 'Discrete Choice with Brand by Price Effects';
   var subj set c brand price &_trgind;
   run;

ods exclude CensoredSummary;
ods output  CensoredSummary=CS;

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

proc freq;
   tables event * censored / list;
   where n(stratum);
   run;
```

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

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	.	.	.

Brand Choice Example, Multinomial Logit Model
Discrete Choice with Brand by Price Effects

The FREQ Procedure

Event	Censored	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	4	800	100.00	800	100.00

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. The null hypothesis that the two models are not significantly different is tested by comparing the likelihoods for the two models. The difference between two $-2\log(\mathcal{L}_C)$'s (the number reported under "With Covariates" in the output) has a chi-square distribution. The degrees of freedom for the test is the difference between 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 first part of the input data set. The first five rows containing the first choice set are printed.

```
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

It 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 alternative.

In all of the previous examples, we have seen models 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 180, 187, 301, and 305 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 / zero=none separators=" ' ') | identity(price)** specification in the **model** statement is the same as the previous analysis. The additional terms, **identity(p1-p&m) * class(brand / zero=none lprefix=0 separators=" ' on ')** 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 brand/price effects, followed by the asterisk, followed by **class** and the brand effect. This influences the order in which the labels are written. 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 will be 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

"Brand 1" through "Other" are binary brand effect variables. They indicate the brand for each alternative. "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

"Brand 1 on Brand 1" through "Brand 1 on Other" are the first five cross effects. They represent the effect on the utility of each alternative of Brand 1 at its price appearing in the choice set. 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. This value is constant across the alternatives in the first choice set since the price of Brand 1 is constant within the first choice set. Notice that "Brand 1 on Brand 1", which is the effect on the utility of Brand 1 at its price on Brand 1, is the same as "Brand 1 Price" (shown in the previous output), which is the effect of the Brand 1 price on the utility for the Brand 1 alternative. In other words, the "Brand 1 on Brand 1" cross effect is the same as the "Brand 1 Price" effect. Because of this, when we do the analysis, we will see that the coefficient for the effect of "Brand 1 on Brand 1" will be zero.

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

"Brand 2 on Brand 1" through "Brand 2 on Other" are the next five cross effects. They represent the effect on the utility of each alternative of Brand 2 at its price appearing in the choice set. For the first choice set, these five cross effects consist entirely of zeros and $4.99's, where $4.99 is the price of Brand 2 in this choice set. This value is constant across the alternatives in the first choice set since the price of Brand 2 is constant within the

first choice set. Notice that "Brand 2 on Brand 2", which is the effect on the utility of Brand 2 at its price on Brand 2, is the same as "Brand 2 Price" (shown in a previous output), which is the effect of the Brand 2 price on the utility for the Brand 2 alternative. In other words, the "Brand 2 on Brand 2" cross effect is the same as the "Brand 2 Price" effect. Because of this, when we do the analysis, we will see that the coefficient for the effect of "Brand 2 on Brand 2" will be zero.

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

"Brand 3 on Brand 1" through "Brand 2 on Other" are the next five cross effects. They represent the effect on the utility of each alternative of Brand 3 at its price appearing in the choice set. 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 in this choice set. This value is constant across the alternatives in the first choice set since the price of Brand 3 is constant within the first choice set. Notice that "Brand 3 on Brand 3", which is the effect on the utility of Brand 3 at its price on Brand 3, is the same as "Brand 3 Price" (shown in a previous output), which is the effect of the Brand 3 price on the utility for the Brand 3 alternative. In other words, the "Brand 3 on Brand 3" cross effect is the same as the "Brand 3 Price" effect. Because of this, when we do the analysis, we will see that the coefficient for the effect of "Brand 3 on Brand 3" will be zero.

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

Here are the remaining cross effects. They follow the same pattern that was described for the cross effects above.

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

            Other on    Other on    Other on    Other on    Other on
  Brand      Brand 1     Brand 2     Brand 3     Brand 4      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 seen many times previously.

```
ods exclude CensoredSummary;
ods output  CensoredSummary=CS;

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

proc freq;
   tables event * censored / list;
   where n(stratum);
   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

                     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	.	.	.

```
Brand Choice Example, Multinomial Logit Model
Discrete Choice with Cross Effects, Mother Logit

The FREQ Procedure

                                                    Cumulative    Cumulative
Event    Censored    Frequency     Percent     Frequency      Percent
-----------------------------------------------------------------------
    1           4          800      100.00           800       100.00
```

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 175, 187, 301, and 305 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;
```

This step 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

                      Parameter      Standard
            DF         Estimate         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 those seen 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). 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 entrees. The client has one nationally branded competitor, a regional competitor in each of three regions, and a profusion of private label products at the grocery chain level. The product 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 may be set up as a discrete choice model in which a respondent's choice among brands, given choice set C_a of available brands, will correspond to the brand with the highest utility. For each brand i, the utility U_i is the sum of a systematic component V_i and a random component e_i. The probability of choosing brand i from choice set C_a is therefore:

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

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. One way to capture departures from IIA is to use the mother logit formulation of McFadden (1974). In these models, the utility for brand i is a function of both the attributes of brand i and the attributes of other brands. The effect of one brand's attributes on another is termed a cross effect. In the case of designs in which only subsets C_a of the full shelf set C appear, the effect of the presence/absence of one brand on the utility of another is termed an availability cross effect. (See pages 175, 180, 301, and 305 for other discussions of IIA.)

Set Up

In the frozen entree example, there are five alternatives: the client, the client's line extension, a national branded competitor, a regional brand and a private label brand. Several regional and private labels can be tested in each market, then aggregated for the final model. Note that the line extension is treated as a separate alternative rather than as a level of the client brand. This enables us to model the source of volume for the new entry and to quantify any cannibalization that occurs. Each brand is shown at either two or three price points. Additional price points are included so that quadratic models of price elasticity may 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	3 prices + absent
2	X2	4	Client Line Extension	3 prices + absent
	X3	2		microwave/stove-top
	X4	2		shelf-talker yes/no
3	X5	3	Regional	2 prices + absent
4	X6	3	Private Label	2 prices + absent
	X7	2		microwave/stove-top
5	X8	3	Competitor	2 prices + absent

In addition to intercepts and main effects, we also require that all two-way interactions within alternatives be estimable: **`x2*x3, x2*x4, x3*x4`** for the line extension and **`x6*x7`** for private labels. This will enable us to test for different price elasticities by form (stove-top versus microwaveable) and to see if the promotion works better combined with a low price or with different forms. Using a linear model for **`x1-x8`**, the total number of parameters including the intercept, all main effects, and two-way interactions with brand is 25. This assumes that price is treated as qualitative. The actual number of parameters in the choice model is larger than this because of the inclusion of cross effects. Using indicator variables to code availability, the systematic component of utility for brand i can be expressed as:

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

where

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

The x_{ik} and x_{jl} might be expanded to include interaction and polynomial terms. In an availability-cross-effects design, each brand is present in only a fraction of choice sets. The size of this fraction or subdesign is a function of the number of levels of the alternative-specific variable that is used to code availability (usually price). For 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 algorithm model specification, we designate all variables as classification variables and require that all main effects and two-way interactions within brands be estimable. The number of runs to use follows from a calculation of the number of parameters that we wish to estimate in the various submatrices $\mathbf{X}_i$ of $\mathbf{X}$. Assuming that there is a None alternative used as a reference level, the numbers of parameters required for various alternatives are shown in the next table along with the size of submatrices (rounded down) for various numbers of runs. Parameters for quadratic price models are given in parentheses. Note that the effect of private label being in a microwaveable or stove-top form (stove/micro cross effect) is an explicit parameter under the client line extension.

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, $\mathrm{floor}(22 \times 3/4) = 16$; for the competitor and 32 runs, $\mathrm{floor}(32 \times 2/3) = 21$. 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 submatrices for more complex models (Lazari and Anderson 1994).

Designing the Choice Experiment

We will use the %MKTDES autocall macro to create the design. (All of the autocall macros used in this report are documented starting on page 261.) We will start by trying small candidate sets first, as we did in the other examples. The following code shows our first attempt at finding an efficient design.

```
title 'Consumer Food Product Example';

%mktdes(factors=x1 x2=4 x3 x4=2 x5 x6=3 x7=2 x8=3, n=26,
        interact=x2*x3 x2*x4 x3*x4 x6*x7,
        where=(2 <= ((x1 < 4) + (x2 < 4) + (x5 < 3) +
               (x6 < 3) + (x8 < 3)) <= 4),
        procopts=seed=7654321)
```

The option **factors=x1 x2=4 x3 x4=2 x5 x6=3 x7=2 x8=3** names **x1** and **x2** as four-level factors, **x3**, **x4**, and **x7** as two-level factors, and **x5**, **x6** and **x8** as three-level factors. The number of choice sets is **n=26**. The option **interact=x2*x3 x2*x4 x3*x4 x6*x7** names the two-way interactions of interest. The option **procopts=seed=7654321** names the random number seed. This example also uses a **where=** option that we have not seen in previous examples. The **where=** option is used to exclude certain combinations from the candidate set.

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. The goal is to create choice sets with either 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. The Boolean term **(x1 < 4)** is one when **x1** is less than 4 and zero otherwise. Hence,

((x1 < 4) + (x2 < 4) + (x5 < 3) + (x6 < 3) + (x8 < 3)) is the number of available alternatives. The **where=** option keeps all of the choice sets in which either 2, 3, or 4 alternatives are available. The value of the **where=** option must be a valid SAS System **where** clause. This step does not work. PROC OPTEX exits with the messages:

```
ERROR: Can't estimate the model parameters from the given candidates.
ERROR: No design to output.
```

The problem is the candidate set had only 64 runs before the exclusions and only 48 after. The exclusions were severe enough that it is not possible from this candidate set to create a design in 26 runs in which all effects are estimable. As in previous examples, we can try different sizes and a two-step process.

```
%mktdes(factors=x1 x2=4 x3 x4=2 x5 x6=3 x7=2 x8=3, n=26, size=1024,
        interact=x2*x3 x2*x4 x3*x4 x6*x7,
        where=(2 <= ((x1 < 4) + (x2 < 4) + (x5 < 3) +
               (x6 < 3) + (x8 < 3)) <= 4),
        procopts=seed=7654321)

%mktdes(factors=x1 x2=4 x3 x4=2 x5 x6=3 x7=2 x8=3, n=26, size=2048,
        interact=x2*x3 x2*x4 x3*x4 x6*x7,
        where=(2 <= ((x1 < 4) + (x2 < 4) + (x5 < 3) +
               (x6 < 3) + (x8 < 3)) <= 4),
        procopts=seed=7654321)

%mktdes(factors=x1 x2=4 x3 x4 x7=2, run=factex, step=1, size=64,
        interact=x2*x3 x2*x4 x3*x4)
%mktdes(factors=x5 x6 x8=3, n=26, run=factex optex, step=2, size=27,
        otherint=x6*x7,
        where=(2 <= ((x1 < 4) + (x2 < 4) + (x5 < 3) +
               (x6 < 3) + (x8 < 3)) <= 4),
        procopts=seed=7654321)
```

The second step produces a design with D-efficiency=82.2623. Since the full-factorial design has 3,456 runs for this problem, it is reasonable to try it. We specify **big=5000**, a number larger than the size of the full-factorial so that the macro will use PROC PLAN to create the full factorial instead of PROC FACTEX.

```
title 'Consumer Food Product Example';

%mktdes(factors=x1 x2=4 x3 x4=2 x5 x6=3 x7=2 x8=3, n=26, big=5000,
        interact=x2*x3 x2*x4 x3*x4 x6*x7,
        where=(2 <= ((x1 < 4) + (x2 < 4) + (x5 < 3) +
               (x6 < 3) + (x8 < 3)) <= 4),
        procopts=seed=7654321)
```

After exclusions, the candidate set has 2776 runs. This size, while well within the reasonable range is large enough that local optima will likely be a problem. That is, PROC OPTEX will have a tough time finding *the* optimal design, although as always, we would expect it to come very close very quickly. The PROC PLAN step took less than a second, and the PROC OPTEX step took about 28 seconds to create 10 designs.

Consumer Food Product Example

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	83.7773	68.5707	60.5171	1.1782
2	83.5351	66.5636	56.3773	1.1973
3	83.4898	66.6937	53.3434	1.1887
4	83.0419	66.7546	57.1022	1.1854
5	82.9902	67.1331	58.7965	1.1883

When You Have a Long Time to Search for an Efficient Design

With a moderate to large candidate set such as this one, it is likely that we can do better with more iterations. The next steps show how to repeatedly run PROC OPTEX from a macro, each time with a different (clock determined) seed. This job is run in batch over lunch, overnight, or over the weekend. First, the %MKTDES macro is run once to create the candidate set. The ad hoc macro %DOIT runs PROC OPTEX up to 10,000 times. Each time, the best efficiency found so far and the random number seed used to find it are reported and stored.

```
%mktdes(factors=x1 x2=4 x3 x4=2 x5 x6=3 x7=2 x8=3, n=26, big=5000,
        interact=x2*x3 x2*x4 x3*x4 x6*x7,
        where=(2 <= ((x1 < 4) + (x2 < 4) + (x5 < 3) +
               (x6 < 3) + (x8 < 3)) <= 4),
        run=plan)

%macro doit;
   %let best = 0;
   %let seed = 5;

   %do i = 1 %to 10000;
      title "&i"; %put &i;
      proc optex data=Cand1 seed=&seed;
         ods output efficiencies=e;
         class x1 x2 x3 x4 x5 x6 x7 x8 / param=orthref;
         model x1 x2 x3 x4 x5 x6 x7 x8 x2*x3 x2*x4 x3*x4 x6*x7;
         generate n=26 iter=20 keep=1 method=m_federov;
         output out=Design;
         run; quit;

      data _null_;
         set e;
         if DCriterion > &best then do;
            call symput('best', put(DCriterion, 8.4));
            call symput('bestseed', symget('seed'));
            end;
         seed = abs(1e6 * (1e-5 + time() - floor(time())));
         call symput('seed', compress(put(seed, 12.0)));
         run;

      data sasuser.results;
         DCriterion = &best;
         Seed       = &bestseed;
         run;
```

```
        %put seed=&seed, best=&best, best seed=&bestseed;
        %end;

%mend;

%doit
```

The macro variable **Best**, the best efficiency found, is initialized to zero. The random number seed is initialized to an arbitrary value, in this case 5. The macro do loop loops over the PROC OPTEX step and the DATA steps that report and store the results. The PROC OPTEX code came from copying the code that the macro wrote, modifying the random number seed to use a macro variable and modifying **iter=** and **keep=** to iterate 20 times and report the best design. In addition, an **ods output efficiencies=e** statement is added so the efficiency of the best design will be available in a SAS data set.

The **data _null_** step reads the D-efficiency for the latest PROC OPTEX step, and if it is better than the previous best, stores it in a macro variable along with the random number seed. This step also generates a new random number seed from the clock time. Finally, the results are stored in a permanent SAS data set SASUSER.RESULTS. This approach is preferred over simply specifying something like **iter= 100000** with PROC OPTEX because intermediate results are quickly available. If there is a power failure or other problem while the job runs, you should still get useful information. On some operating systems, you can even monitor the progress of the job. If this macro was stored in a file **macro.sas** and run by typing **sas macro.sas**, then the results are in the files **macro.lst** and **macro.log**. For example, on a work station running UNIX, you can look at these files as the job is running. Since clock time is used to make random number seeds, these results will not exactly be reproduced if this same code is run again.

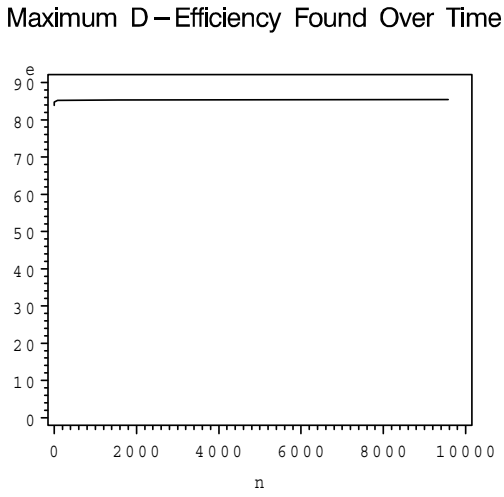
Looking at the end of the list file, we find the maximum efficiency of 85.3985 in run 9576 of PROC OPTEX. The next table shows the PROC OPTEX run where the best efficiency changed and the percent improvement. This table shows efficiency ranging from the best D-efficiency found in the first 20 iterations to the largest D-efficiency found anywhere.

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 numbers of iterations. 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. Six minutes into the search, in the first six passes through the macro ($6 \times 20 = 120$ total iterations), we find 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 above the best design found in 120 iterations was 0.75952%, about three-quarters of a percent. These numbers will change with other problems and other seeds. However, as these results show, usually the first few iterations will give you a good, efficient design, and usually, subsequent iterations will give you slight improvements but with a cost of much greater run times.

```
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 'Maximum D-Efficiency Found Over Time';
   plot e * n / vaxis=axis1;
   symbol i=join;
   axis1 order=(0 to 90 by 10);
   run; quit;
```

The plot of maximum D-efficiency as a function of PROC OPTEX run number clearly shows that the gain is slight over increased iterations.



Recreating the Best Design

From the log file or from looking at SASUSER.RESULTS we see that the most efficient design was found with **seed=41611**.

```
title 'Consumer Food Product Example';

proc print data=sasuser.results; run;
```

Consumer Food Product Example

Obs	DCriterion	Seed
1	85.3985	41611

This design can be recreated by explicitly specifying this seed.

```
%mktdes(factors=x1 x2=4 x3 x4=2 x5 x6=3 x7=2 x8=3, n=26, big=5000,
        out=sasuser.choicdes,
        interact=x2*x3 x2*x4 x3*x4 x6*x7,
        where=(2 <= ((x1 < 4) + (x2 < 4) + (x5 < 3) +
               (x6 < 3) + (x8 < 3)) <= 4),
        procopts=seed=41611, iter=20, keep=1)
```

The PROC OPTEX step took 57 seconds and produced the following results.

Consumer Food Product Example

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	85.3985	71.4573	63.7881	1.1461

Examining the Design

Here is the code that prints the frequency of occurrence of each level and each two-way interaction.

```
proc summary data=sasuser.choicdes;
   class _all_;
   ways 1 8;
   output out=sum;
   run;
proc print; by _type_; run;

proc freq data=sasuser.choicdes;
   tables x1-x8 x2*x3 x2*x4 x3*x4 x6*x7;
   run;
```

Here are the results, summarized in a table.

Level	X1	X2	X3	X4	X5	X6	X7	X8	X2 by X3		X2 by X4		X3 by X4		X6 by X7	
1	7	6	13	13	10	7	14	9	3	3	3	3	6	7	4	3
2	6	6	13	13	9	9	12	7	3	3	3	3	7	6	4	5
3	7	6			7	10		10	3	3	3	3			6	4
4	6	8							4	4	4	4				

Examining the Submatrices

We mentioned previously, "it is sometimes necessary and good practice to check the ranks of the submatrices for more complex models (Lazari and Anderson 1994)." Here is a way to do that. For convenience, we use a macro since PROC OPTEX 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 use the same **model** statement as before but not the same **class** statement. This is because we only have enough runs to consider linear price effects within each availability group. Hence, the price variables are no longer designated **class**.

```
%macro evaleff(where);
proc optex data=sasuser.choicdes;
   class x3 x4 x7 / param=orthref;
   model x1-x8 x2*x3 x2*x4 x3*x4 x6*x7;
   generate method=sequential initdesign=sasuser.choicdes;
   where &where;
   run; quit;
%mend;

%evaleff(x1 ne 4)
%evaleff(x2 ne 4)
%evaleff(x5 ne 3)
%evaleff(x6 ne 3)
%evaleff(x8 ne 3)
```

Each PROC OPTEX 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 the results.

Consumer Food Product Example

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	62.3892	38.6926	91.6206	0.8062

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	62.6819	40.2703	86.9954	0.8498

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	66.9433	49.6260	85.7154	0.8272

```
                                                                  Average
                                                                Prediction
Design                                                            Standard
Number     D-Efficiency     A-Efficiency     G-Efficiency          Error
-------------------------------------------------------------------------
     1         73.0870          57.7546          90.5666          0.9014

                                                                  Average
                                                                Prediction
Design                                                            Standard
Number     D-Efficiency     A-Efficiency     G-Efficiency          Error
-------------------------------------------------------------------------
     1         58.6329          38.7890          90.8714          0.9014
```

Examining the Information and Variance Matrices

It is also a good idea to look at the information and variance matrices. This is done by adding the statement **`examine i v`** to any PROC OPTEX run.

```
proc optex data=sasuser.choicdes;
   class x3 x4 x7 / param=orthref;
   model x1-x8 x2*x3 x2*x4 x3*x4 x6*x7;
   generate method=sequential initdesign=sasuser.choicdes;
   where x1 ne 4;
   examine i v;
   run; quit;
```

In the interest of space, only some of the results are shown.

```
                    Consumer Food Product Example

                         The OPTEX Procedure

                                                                  Average
                                                                Prediction
Design                                                            Standard
Number     D-Efficiency     A-Efficiency     G-Efficiency          Error
-------------------------------------------------------------------------
     1         62.3892          38.6926          91.6206          0.8062

                    Consumer Food Product Example

                         The OPTEX Procedure
                   Examining Design Number 1

      Log determinant of the information matrix      3.2811E+01
      Maximum prediction variance over candidates        0.7743
      Average prediction variance over candidates        0.6500
      Average variance of coefficients                   0.1292
      D-Efficiency                                      62.3892
      A-Efficiency                                      38.6926
```

Information Matrix

	Intercept	x1	x2	x3	x4	x5	x6
Intercept	20.0	0.0	1.3	0.0	0.0	-2.0	2.0
x1	0.0	14.0	2.0	2.0	2.0	2.0	0.0
x2	1.3	2.0	11.1	0.0	0.0	0.7	-4.7
x3	0.0	2.0	0.0	20.0	0.0	2.0	0.0
x4	0.0	2.0	0.0	0.0	20.0	-2.0	0.0
x5	-2.0	2.0	0.7	2.0	-2.0	14.0	-1.0
x6	2.0	0.0	-4.7	0.0	0.0	-1.0	14.0
x7	0.0	-2.0	-8.0	0.0	0.0	0.0	0.0
x8	2.0	-3.0	-4.7	0.0	-4.0	1.0	0.0
x2*x3	0.0	4.0	0.0	1.3	0.0	-0.7	0.0
x2*x4	0.0	0.0	0.0	0.0	1.3	0.7	1.3
x3*x4	0.0	0.0	0.0	0.0	0.0	2.0	-6.0
x6*x7	0.0	-2.0	0.0	-2.0	2.0	-1.0	0.0

.
.
.

Variance Matrix

	Intercept	x1	x2	x3	x4	x5	x6
Intercept	0.0639	-0.0040	-0.0687	-0.0005	-0.0074	0.0131	-0.0334
x1	-0.0040	0.0913	0.0037	-0.0044	-0.0083	-0.0168	0.0001
x2	-0.0687	0.0037	0.4145	-0.0004	0.0410	-0.0297	0.1582
x3	-0.0005	-0.0044	-0.0004	0.0522	-0.0014	-0.0070	0.0007
x4	-0.0074	-0.0083	0.0410	-0.0014	0.0605	0.0063	0.0134
x5	0.0131	-0.0168	-0.0297	-0.0070	0.0063	0.0819	-0.0087
x6	-0.0334	0.0001	0.1582	0.0007	0.0134	-0.0087	0.1482
x7	-0.0380	0.0137	0.2157	-0.0012	0.0232	-0.0194	0.0851
x8	-0.0464	0.0207	0.2223	-0.0025	0.0397	-0.0205	0.0717
x2*x3	-0.0090	-0.0289	0.0513	-0.0052	0.0135	0.0083	0.0197
x2*x4	0.0003	0.0071	-0.0044	-0.0002	-0.0053	-0.0085	-0.0154
x3*x4	-0.0054	0.0005	0.0278	0.0028	-0.0054	-0.0084	0.0431
x6*x7	-0.0019	0.0111	0.0044	0.0069	-0.0073	0.0006	0.0111

.
.
.

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
```

Contrast this with a specification that includes all simple effects and two-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-factor interactions using the notation **x1|x2|x3|x4|x5|x6|x7|x8@2**. This list will generate all of the effects of interest, **x1-x8 x2*x3 x2*x4 x3*x4 x6*x7**, and all the other two-way interactions, but that is not a problem 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@2 / e aliasing;
   run;
```

```
Intercept - 7.2928*x2*x7 + 3.9403*x3*x7 - 3.8348*x4*x7 + 8.943*x5*x7 -
3.5093*x1*x8 - 10.789*x2*x8 - 3.0971*x3*x8 - 2.6715*x4*x8 - 3.4108*x5*x8 -
1.2232*x6*x8 - 14.437*x7*x8

x1 - 1.4634*x2*x7 + 1.8795*x3*x7 - 3.7098*x4*x7 + 6.1087*x5*x7 - 1.5265*x1*x8 +
4.388*x2*x8 + 0.9191*x3*x8 - 0.2793*x4*x8 + 1.8472*x5*x8 + 0.816*x6*x8 -
8.4327*x7*x8

x2 + 2.2518*x2*x7 - 0.5586*x3*x7 + 1.8045*x4*x7 - 2.6429*x5*x7 + 5.345*x1*x8 +
0.8605*x2*x8 + 1.722*x3*x8 + 0.425*x4*x8 + 0.0336*x5*x8 + 1.2263*x6*x8 +
2.7259*x7*x8
.
.
.
x2*x3 - 0.355*x2*x7 + 0.4404*x3*x7 - 0.9841*x4*x7 + 0.6352*x5*x7 - 2.0611*x1*x8
- 0.1886*x2*x8 - 0.4661*x3*x8 - 0.3117*x4*x8 + 0.8003*x5*x8 + 0.2153*x6*x8 -
1.1787*x7*x8

x2*x4 + 0.1913*x2*x7 + 0.2884*x3*x7 + 1.2156*x4*x7 - 0.3771*x5*x7 + 2.5548*x1*x8
+ 1.365*x2*x8 + 0.4744*x3*x8 + 0.2408*x4*x8 + 0.0595*x5*x8 - 0.7713*x6*x8 +
1.219*x7*x8
.
.
.
```

Again, we have a list of linear combinations that are estimable. This shows that the Intercept cannot be estimated independently of the **x2*x7** interaction, the **x3*x7** interaction, the **x4*x7** interaction, ..., and the **x7*x8** interaction. Similarly, **x1** is confounded with lots of two way interactions, and so on. This is why we want to be estimable the two-way interactions between factors that are combined to create an attribute. 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.

The Final Design

This code creates the final choice design, stored in SASUSER.FINCHDES, sorted by the shelf-talker variable, and randomized within shelf talker.

```
proc format;
   value f1_ 1 = '$1.29' 2 = '$1.69' 3 = '$2.09' 4 = 'N/A';
   value f2_ 1 = '$1.39' 2 = '$1.89' 3 = '$2.39' 4 = 'N/A';
   value f3_ 1 = 'micro' 2 = 'stove';
   value f5_ 1 = '$1.99' 2 = '$2.49' 3 = 'N/A';
   value f6_ 1 = '$1.49' 2 = '$2.29' 3 = 'N/A';
   value f8_ 1 = '$1.99' 2 = '$2.39' 3 = 'N/A';
   value yn  1 = 'Yes'   2 = 'No';
   run;

data choicdes;
   length b1-b5 $ 12;
   set sasuser.choicdes;
   r = uniform(7); /* for randomization within shelf talker */
   b1 = put(x1,f1_.);
   b2 = put(x2,f2_.);
   if b2 ne 'N/A' then b2 = trim(b2)||'/'||put(x3,f3_.);
   b3 = put(x5,f5_.);
   b4 = put(x6,f6_.);
   if b4 ne 'N/A' then b4 = trim(b4)||'/'||put(x7,f3_.);
   b5 = put(x8,f8_.);
   label b1 = 'Client Brand'
         b2 = 'Client Line Extension'
         b3 = 'Regional Brand'
         b4 = 'Private Label'
         b5 = 'National Competitor'
         x4 = 'Shelf Talker';
   format x4 yn.;
   run;
proc sort out=sasuser.finchdes(drop=r); by descending x4 r; run;

proc print label; var b:; by descending x4; run;
```

Here is the design.

Consumer Food Product Example

Shelf Talker=No

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

------------------------------ Shelf Talker=Yes ------------------------------

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

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 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 four 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. In fact our best design when we tried this (not shown) had the shelf-talker in only 10 of 26 choice sets.
- Move the choice sets with client line extension unavailable to the no-shelf-talker block and rerandomize. Then the shelf-talker is 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.

Generating Artificial Data

This DATA step generates some artificial data.

```
%let m   = 6;
%let mm1 = %eval(&m - 1);
%let n   = 26;
data _null_;
   array brands[&m] _temporary_ (5 7 1 2 3 -2);
   array u[&m];
   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;
         if x4 = 1 then u2 = u2 + 1; /* shelf-talker */
         if x3 = 1 then u2 = u2 + 1; /* microwave    */
         if x7 = 1 then u4 = u4 + 1; /* microwave    */
         * pull prices out of the formats;
         if x1 ne 4 then u1 = u1 - input(substr(put(x1,f1_.),2),4.);
                    else u1 = .;
         if x2 ne 4 then u2 = u2 - input(substr(put(x2,f2_.),2),4.);
                    else u2 = .;
         if x5 ne 3 then u3 = u3 - input(substr(put(x5,f5_.),2),4.);
                    else u3 = .;
         if x6 ne 3 then u4 = u4 - input(substr(put(x6,f6_.),2),4.);
                    else u4 = .;
         if x8 ne 3 then u5 = u5 - input(substr(put(x8,f8_.),2),4.);
                    else u5 = .;
         * Choose the most preferred alternative.;
         m = max(of u1-u&m);
         if      n(u1) and abs(u1 - m) < 1e-4 then c = 1;
         else if n(u2) and abs(u2 - m) < 1e-4 then c = 2;
         else if n(u3) and abs(u3 - m) < 1e-4 then c = 3;
         else if n(u4) and abs(u4 - m) < 1e-4 then c = 4;
         else if n(u5) and abs(u5 - m) < 1e-4 then c = 5;
         else                                      c = 6;
         put +(-1) c @@;
         end;
      end;
   stop;
   run;
```

This DATA step reads the data.

```
data results;
   input Subj (choose1-choose&n) (1.) @@;
   datalines;
  1 22251224212125422245212222   2 51111224232221622211222522
  3 22251123512221422216222121   4 11211224242215422211222212
  5 22211223552225422241222222   6 12211224252211425214222122
  .
  .
  .
297 22441223242211322214221122 298 22251224512221422245222112
299 52211224242251422211212122 300 22211224262221422251222212
;
```

Processing the Data

The analysis proceeds in a fashion similar to before. For analysis, the design will have four factors as shown by the variables in the data set KEY. **Brand** is the alternative name; its values are directly read from the 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.

```
data key;
   input Brand $ 1-10 (Price Micro Shelf) ($);
   datalines;
Client     x1 .  .
Extension  x2 x3 x4
Regional   x5 .  .
Private    x6 x7 .
National   x8 .  .
None       .  .  .
;
```

The **Price** factors are different for each alternative. Therefore, we will map the factors in the design from values of 1, 2, ... directly to prices before a common price factor is created. Not available will be coded as a price of zero.

```
proc format;
   value p1_ 1 = '1.29' 2 = '1.69' 3 = '2.09' 4 = '0';
   value p2_ 1 = '1.39' 2 = '1.89' 3 = '2.39' 4 = '0';
   value p5_ 1 = '1.99' 2 = '2.49' 3 = '0';
   value p6_ 1 = '1.49' 2 = '2.29' 3 = '0';
   value p8_ 1 = '1.99' 2 = '2.39' 3 = '0';
   run;

data temp;
   set sasuser.finchdes;
   x1 = input(put(x1, p1_.), 5.);
   x2 = input(put(x2, p2_.), 5.);
   x5 = input(put(x5, p5_.), 5.);
   x6 = input(put(x6, p6_.), 5.);
   x8 = input(put(x8, p8_.), 5.);
   keep x1-x8;
   run;
```

The design is converted from one row per choice set to one row for each alternative of each choice set using the %MKTROLL macro. The macro %MKTROLL is used to create the data set ROLLED from TEMP using the mapping in KEY and using the variable **Brand** as the alternative ID variable. By default, all of the original factors in the **design=** data set are dropped after they are used to create the factors in the **out=** data set. In this case, we specify **keep=x1 x2 x5 x6 x8** because we need to keep all of the original price factors to use in cross effects.

```
%mktroll(design=temp, key=key, alt=brand, out=rolled, keep=x1 x2 x5 x6 x8)
```

These next steps show the input and 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=temp(obs=2); run;

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

Consumer Food Product Example

Obs	x1	x2	x3	x4	x5	x6	x7	x8
1	1.29	1.89	1	No	1.99	0.00	1	1.99
2	1.29	1.39	2	No	1.99	2.29	1	0.00

Consumer Food Product Example

Obs	Set	Brand	Price	Micro	Shelf	x1	x2	x5	x6	x8
1	1	Client	1.29	.	.	1.29	1.89	1.99	0.00	1.99
2	1	Extension	1.89	1	2	1.29	1.89	1.99	0.00	1.99
3	1	Regional	1.99	.	.	1.29	1.89	1.99	0.00	1.99
4	1	Private	0.00	1	.	1.29	1.89	1.99	0.00	1.99
5	1	National	1.99	.	.	1.29	1.89	1.99	0.00	1.99
6	1	None	.	.	.	1.29	1.89	1.99	0.00	1.99
7	2	Client	1.29	.	.	1.29	1.39	1.99	2.29	0.00
8	2	Extension	1.39	2	2	1.29	1.39	1.99	2.29	0.00
9	2	Regional	1.99	.	.	1.29	1.39	1.99	2.29	0.00
10	2	Private	2.29	1	.	1.29	1.39	1.99	2.29	0.00
11	2	National	0.00	.	.	1.29	1.39	1.99	2.29	0.00
12	2	None	.	.	.	1.29	1.39	1.99	2.29	0.00

Consider the first two observations in the data set ROLLED.

Set 1, Alternative 1

Brand	=	"Client"	the brand for this alternative
Price	=	**x1** = 1.29	the price of this alternative
Micro	=	.	does not apply to this brand
Shelf	=	.	does not apply to this brand
x1	=	1.29	the price of the client brand in this choice set
x2	=	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	=	0,	private label unavailable in this choice set
x8	=	1.99	the price of the national competitor 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	=	1	Microwave, yes
Shelf	=	2	Shelf Talker, No
x1	=	1.29	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	=	0,	private label unavailable in this choice set
x8	=	1.99	the price of the national competitor 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.

The data and design are merged in the usual way using the %MKTMERGE macro.

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

proc print data=res2(obs=18); run;
```

Here are the data and design for the first two choice sets for the first subject.

Consumer Food Product Example

Obs	Subj	Set	Brand	Price	Micro	Shelf	x1	x2	x5	x6	x8	c
1	1	1	Client	1.29	.	.	1.29	1.89	1.99	0.00	1.99	2
2	1	1	Extension	1.89	1	2	1.29	1.89	1.99	0.00	1.99	1
3	1	1	Regional	1.99	.	.	1.29	1.89	1.99	0.00	1.99	2
4	1	1	Private	0.00	1	.	1.29	1.89	1.99	0.00	1.99	2
5	1	1	National	1.99	.	.	1.29	1.89	1.99	0.00	1.99	2
6	1	1	None	.	.	.	1.29	1.89	1.99	0.00	1.99	2
7	1	2	Client	1.29	.	.	1.29	1.39	1.99	2.29	0.00	2
8	1	2	Extension	1.39	2	2	1.29	1.39	1.99	2.29	0.00	1
9	1	2	Regional	1.99	.	.	1.29	1.39	1.99	2.29	0.00	2
10	1	2	Private	2.29	1	.	1.29	1.39	1.99	2.29	0.00	2
11	1	2	National	0.00	.	.	1.29	1.39	1.99	2.29	0.00	2
12	1	2	None	.	.	.	1.29	1.39	1.99	2.29	0.00	2

We need to do a few more things before we are ready to code. Since we will be treating **`Price`** as a quantitative factor (not a **`class`** variable), we need to convert the missing price for the constant "None" alternative to zero. We also need to convert the missings for when **`Micro`** and **`Shelf`** do not apply to 2 for "No". Finally, we need to output just the alternatives that are available (those with a nonzero price and also the none alternative).

```
data res3;
   set res2;
   if price = . then price = 0;
   if micro = . then micro = 2;
   if shelf = . then shelf = 2;
   if brand = 'None' or price ne 0;
   format micro shelf yn.;
   run;

proc print data=res3(obs=10); run;
```

Consumer Food Product Example

Obs	Subj	Set	Brand	Price	Micro	Shelf	x1	x2	x5	x6	x8	c
1	1	1	Client	1.29	No	No	1.29	1.89	1.99	0.00	1.99	2
2	1	1	Extension	1.89	Yes	No	1.29	1.89	1.99	0.00	1.99	1
3	1	1	Regional	1.99	No	No	1.29	1.89	1.99	0.00	1.99	2
4	1	1	National	1.99	No	No	1.29	1.89	1.99	0.00	1.99	2
5	1	1	None	0.00	No	No	1.29	1.89	1.99	0.00	1.99	2
6	1	2	Client	1.29	No	No	1.29	1.39	1.99	2.29	0.00	2
7	1	2	Extension	1.39	No	No	1.29	1.39	1.99	2.29	0.00	1
8	1	2	Regional	1.99	No	No	1.29	1.39	1.99	2.29	0.00	2
9	1	2	Private	2.29	Yes	No	1.29	1.39	1.99	2.29	0.00	2
10	1	2	None	0.00	No	No	1.29	1.39	1.99	2.29	0.00	2

Cross Effects

These next steps code the design for analysis.

```
proc transreg data=res3 design=5000 nozeroconstant norestoremissing;
   model class(brand / zero='None')
         class(brand / zero='None' separators='' ' ') * identity(price)
         class(shelf micro / lprefix=12 12 zero='No' 'No')
         identity(x1 x2 x5 x6 x8) *
         class(brand / zero='None' separators=' ' ' on ') /
         lprefix=0;
   output out=coded(drop=_type_ _name_ intercept
                    where=(brand = 'None' or price ne 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 specification **`class(brand / zero='None')`** creates the brand effects for each brand except the none alternative. The specification **`class(brand / zero='None' separators=" ' ') * identity(price)`** creates the alternative-specific price effects. The **`zero='None'`** option, like **`zero='Home'`** and other **`zero=`***'literal-string'* options we have seen in previous examples, names the actual formatted value of the **`class`** variable that is to 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 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 **`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 / lprefix=12 12 zero='No' 'No')`** names the shelf talker and microwave variables as categorical variables and creates dummy variables for the "Yes" categories, not the "No" categories. The first **`'No'`** applies to **`Shelf`** and the second **`'No'`** applies to **`Micro`**. The specification **`identity(x1 x2 x5 x6 x8) * class(brand / zero='None' separators=' ' ' on ')`** creates the cross effects. The **`separators=`** option is specified with a second value of **`' on '`** to create cross effect labels like "CE, Client on Extension" (where CE means cross effect). 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 might 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 next. Before we look at the coded data, recall that the design for the first choice sets is as follows.

Obs	Client Brand	Client Line Extension	Regional Brand	Private Label	National Competitor
1	$1.29	$1.89/micro	$1.99	N/A	$1.99
2	$1.29	$1.39/stove	$1.99	$2.29/micro	N/A
3	$2.09	$2.39/micro	$1.99	$2.29/stove	N/A
4	$1.69	N/A	$1.99	$1.49/stove	$2.39
5	$1.29	N/A	N/A	$2.29/stove	N/A

We will look at the coded data set in several ways. First, here are the **Brand**, **Price**, **c**, microwave and shelf talker variables, printed by choice set. Information from just the first subject and the first five choice sets is printed.

```
proc print data=coded(obs=23) label;
   var Brand Price c Shelf Micro;
   by set;
   run;
```

Consumer Food Product Example

---------- Set=1 ----------

Obs	Brand	Price	c	Shelf Talker	Microwave
1	Client	1.29	2	No	No
2	Extension	1.89	1	No	Yes
3	Regional	1.99	2	No	No
4	National	1.99	2	No	No
5	None	0.00	2	No	No

---------- Set=2 ----------

Obs	Brand	Price	c	Shelf Talker	Microwave
6	Client	1.29	2	No	No
7	Extension	1.39	1	No	No
8	Regional	1.99	2	No	No
9	Private	2.29	2	No	Yes
10	None	0.00	2	No	No

---------- Set=3 ----------

Obs	Brand	Price	c	Shelf Talker	Microwave
11	Client	2.09	2	No	No
12	Extension	2.39	1	No	Yes
13	Regional	1.99	2	No	No
14	Private	2.29	2	No	No
15	None	0.00	2	No	No

```
------------------------------------ Set=4 -------------------------------------

                                              Shelf
              Obs    Brand       Price    c   Talker    Microwave

               16    Client       1.69    2     No         No
               17    Regional     1.99    2     No         No
               18    Private      1.49    2     No         No
               19    National     2.39    1     No         No
               20    None         0.00    2     No         No

------------------------------------ Set=5 -------------------------------------

                                             Shelf
              Obs    Brand      Price    c   Talker    Microwave

               21    Client      1.29    1     No         No
               22    Private     2.29    2     No         No
               23    None        0.00    2     No         No
```

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 private label brand is not available in this choice set. The second choice set consists of five alternatives including "None". The national competitor are not available in this choice set. The fifth choice set consists of three alternatives including "None". The extension, regional competitor, and national competitor are not available in this choice set.

Here are the brand effects and alternative-specific price effects.

```
proc print data=coded(obs=5) label;
   id Brand;
   var BrandClient -- BrandPrivate;
   run;

proc print data=coded(obs=5) label;
   id Brand Price;
   var BrandClientPrice --BrandPrivatePrice;
   run;
```

```
                        Consumer Food Product Example

   Brand         Client    Extension    Regional    National    Private

   Client           1           0           0           0          0
   Extension        0           1           0           0          0
   Regional         0           0           1           0          0
   National         0           0           0           1          0
   None             0           0           0           0          0

                        Consumer Food Product Example

                      Client    Extension    Regional    National    Private
Brand         Price    Price      Price        Price       Price      Price

Client         1.29    1.29        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
National       1.99    0.00        0.00         0.00        1.99         0
None           0.00    0.00        0.00         0.00        0.00         0
```

The brand effects and alternative-specific price effects are similar to what we have seen previously. The difference is the presence of all zero columns for unavailable alternatives, in this case the private label. This following code prints the cross effects along with **Brand** and **Price** for the first choice set.

```
proc print data=coded(obs=5) label;
   id Brand Price;
   var x1BrandClient -- x1BrandPrivate;
   run;

proc print data=coded(obs=5) label;
   id Brand Price;
   var x2BrandClient -- x2BrandPrivate;
   run;

proc print data=coded(obs=5) label;
   id Brand Price;
   var x5BrandClient -- x5BrandPrivate;
   run;

proc print data=coded(obs=5) label;
   id Brand Price;
   var x6BrandClient -- x6BrandPrivate;
   run;

proc print data=coded(obs=5) label;
   id Brand Price;
   var x8BrandClient -- x8BrandPrivate;
   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 $1.29 on the utility of the other brands. These terms represent the interaction of the continuous and constant within choice set variable **x1 = 1.29** and the binary coded brand effects. The term "CE, Client on Extension" represents the cross effect of the client brand at its price on the utility of the extension. The term "CE, Client on Client" represents in some sense the cross effect of the client brand at its price on the utility of the client brand. In other words, it is exactly the same as the client price effect and hence will have a zero coefficient in the analysis.

Consumer Food Product Example

Brand	Price	CE, Client on Client	CE, Client on Extension	CE, Client on Regional	CE, Client on National	CE, Client on Private
Client	1.29	1.29	0.00	0.00	0.00	0
Extension	1.89	0.00	1.29	0.00	0.00	0
Regional	1.99	0.00	0.00	1.29	0.00	0
National	1.99	0.00	0.00	0.00	1.29	0
None	0.00	0.00	0.00	0.00	0.00	0

This next panel shows the terms that capture the effect of the extension being available at $1.89 on the utility of the other brands. These terms represent the interaction of the continuous and constant within choice set variable **x2 = 1.89** and the binary coded brand effects. The term "CE, Extension on Regional" represents the cross effect of the client brand extension at its price on the utility of the regional competitor. The term "CE, Extension on Extension" represents in some sense the cross effect of the extension at its price on the utility of the extension. In other words, it is exactly the same as the extension price effect and hence will have a zero coefficient in the analysis.

Consumer Food Product Example

Brand	Price	CE, Extension on Client	CE, Extension on Extension	CE, Extension on Regional	CE, Extension on National	CE, Extension on Private
Client	1.29	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
National	1.99	0.00	0.00	0.00	1.89	0
None	0.00	0.00	0.00	0.00	0.00	0

Similarly, this next panel shows the terms that capture the effect of the regional competitor being available at $1.99 on the utility of the other brands.

Consumer Food Product Example

Brand	Price	CE, Regional on Client	CE, Regional on Extension	CE, Regional on Regional	CE, Regional on National	CE, Regional on Private
Client	1.29	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
National	1.99	0.00	0.00	0.00	1.99	0
None	0.00	0.00	0.00	0.00	0.00	0

This next panel shows the private label, unavailable in this choice set, has no effect on the utility of the brands in this set.

Consumer Food Product Example

Brand	Price	CE, Private on Client	CE, Private on Extension	CE, Private on Regional	CE, Private on National	CE, Private on Private
Client	1.29	0	0	0	0	0
Extension	1.89	0	0	0	0	0
Regional	1.99	0	0	0	0	0
National	1.99	0	0	0	0	0
None	0.00	0	0	0	0	0

This next panel shows the terms that capture the effect of the national competitor being available at $1.99 on the utility of the other brands.

Consumer Food Product Example

Brand	Price	CE, National on Client	CE, National on Extension	CE, National on Regional	CE, National on National	CE, National on Private
Client	1.29	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
National	1.99	0.00	0.00	0.00	1.99	0
None	0.00	0.00	0.00	0.00	0.00	0

Coding and Fitting the Cross Effects Model

These next steps aggregate, code, and perform the analysis. This is the logical order to do things: aggregate then code and analyze the data. The data set size is greatly reduced by the aggregation, which makes both the TRANSREG and the PHREG steps run in just a few seconds. Aggregation is faster too since the aggregation is based on a smaller number of uncoded variables. We coded before aggregating previously just to make it easier to show the results of the coding.

```
proc summary data=res3 nway;
   class set brand price shelf micro x1 x2 x5 x6 x8 c;
   output out=agg(drop=_type_);
   run;

proc print; where set = 1; run;
```

All of the variables used in the analysis are named as **class** variables in PROC SUMMARY. PROC SUMMARY reduces the data set from 34,500 observations to 209. 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	1.29	No	No	1.29	1.89	1.99	0	1.99	1	68
2	1	Client	1.29	No	No	1.29	1.89	1.99	0	1.99	2	232
3	1	Extension	1.89	No	Yes	1.29	1.89	1.99	0	1.99	1	225
4	1	Extension	1.89	No	Yes	1.29	1.89	1.99	0	1.99	2	75
5	1	National	1.99	No	No	1.29	1.89	1.99	0	1.99	1	6
6	1	National	1.99	No	No	1.29	1.89	1.99	0	1.99	2	294
7	1	None	0.00	No	No	1.29	1.89	1.99	0	1.99	2	300
8	1	Regional	1.99	No	No	1.29	1.89	1.99	0	1.99	1	1
9	1	Regional	1.99	No	No	1.29	1.89	1.99	0	1.99	2	299

In the first choice set, the client brand was chosen (**c** = 1) a total of **_freq_** = 68 times and not chosen (**c** = 2) a total of **_freq_** = 232 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.

```
proc transreg data=agg design=5000 nozeroconstant norestoremissing;
   model class(brand / zero='None')
         class(brand / zero='None' separators='' ' ') * identity(price)
         class(shelf micro / lprefix=12 12 zero='No' 'No')
         identity(x1 x2 x5 x6 x8) *
         class(brand / zero='None' separators=' ' ' on ') /
         lprefix=0  order=data;
   output out=coded(drop=_type_ _name_ intercept
                    where=(brand = 'None' or price ne 0));
   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;

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

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. Analysis is the same as we have seen previously with aggregate data. PROC PHREG is run to fit the mother logit model, complete with availability cross effects.

Multinomial Logit Model Results

These steps produced the following results. (Recall that we used **%phchoice(on)** on page 71 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	1500	300	1200
3	3	1500	300	1200
4	4	1500	300	1200
5	5	900	300	600
6	6	1200	300	900
7	7	1500	300	1200
8	8	900	300	600
9	9	1200	300	900
10	10	1500	300	1200
11	11	900	300	600
12	12	1200	300	900
13	13	1500	300	1200
14	14	1500	300	1200
15	15	900	300	600
16	16	1500	300	1200
17	17	1200	300	900
18	18	1500	300	1200
19	19	1500	300	1200
20	20	1200	300	900
21	21	1200	300	900
22	22	1500	300	1200
23	23	1200	300	900
24	24	1500	300	1200
25	25	1500	300	1200
26	26	1500	300	1200
Total		34500	7800	26700

Convergence Status

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

Consumer Food Product Example

The PHREG Procedure

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	111923.05	98180.822
AIC	111923.05	98244.822
SBC	111923.05	98467.602

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	13742.2313	32	<.0001
Score	15093.5363	32	<.0001
Wald	4823.7301	32	<.0001

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Client	1	5.46733	0.85358	41.0265	<.0001
Extension	1	5.58334	0.86685	41.4861	<.0001
National	1	3.50597	1.13426	9.5541	0.0020
Regional	1	3.80078	1.47533	6.6369	0.0100
Private	1	3.13270	0.71627	19.1287	<.0001
Client Price	1	-0.67421	0.32250	4.3704	0.0366
Extension Price	1	-0.07049	0.31052	0.0515	0.8204
National Price	1	-0.22402	0.38973	0.3304	0.5654
Regional Price	1	-1.18081	0.65386	3.2613	0.0709
Private Price	1	-0.96577	0.28492	11.4894	0.0007
Shelf Talker Yes	1	0.61900	0.08419	54.0569	<.0001
Microwave Yes	1	0.72858	0.07334	98.7000	<.0001
CE, Client on Client	0	0	.	.	.
CE, Client on Extension	1	0.28194	0.31256	0.8137	0.3670
CE, Client on National	1	-0.06563	0.33552	0.0383	0.8449
CE, Client on Regional	1	-0.06821	0.36875	0.0342	0.8532
CE, Client on Private	1	0.30073	0.28794	1.0908	0.2963
CE, Extension on Client	1	0.46570	0.29571	2.4801	0.1153
CE, Extension on Extension	0	0	.	.	.
CE, Extension on National	1	0.19053	0.32421	0.3454	0.5567
CE, Extension on Regional	1	-0.09000	0.35402	0.0646	0.7993
CE, Extension on Private	1	0.14278	0.29837	0.2290	0.6323
CE, Regional on Client	1	0.26312	0.18077	2.1188	0.1455
CE, Regional on Extension	1	0.29864	0.18450	2.6201	0.1055
CE, Regional on National	1	0.33478	0.18863	3.1497	0.0759
CE, Regional on Regional	0	0	.	.	.
CE, Regional on Private	1	0.13963	0.18898	0.5459	0.4600
CE, Private on Client	1	-0.29806	0.28888	1.0646	0.3022
CE, Private on Extension	1	-0.20526	0.28899	0.5045	0.4775
CE, Private on National	1	-0.40419	0.30925	1.7082	0.1912
CE, Private on Regional	1	-0.28243	0.32123	0.7730	0.3793
CE, Private on Private	0	0	.	.	.
CE, National on Client	1	0.02888	0.22009	0.0172	0.8956
CE, National on Extension	1	0.08796	0.22309	0.1555	0.6934
CE, National on National	0	0	.	.	.
CE, National on Regional	1	0.32827	0.29212	1.2628	0.2611
CE, National on Private	1	0.0004496	0.23537	0.0000	0.9985

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, regional competitor, national competitor, private label, and finally the none alternative. The price effects are all negative. Both the shelf-talker and the microwaveable option have positive utility. The cross effects are nonsignificant.

Modeling Subject Attributes

This example uses the same design and data as we have just seen, but this time we have some demographic information about our respondents that we wish to model. The DATA step below 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 22251224212125422245212222  33  44   2 51111224232221622211222522  52  82
  3 22251123512221422216222121  51 136   4 11211224242215422211222212  60 108
  5 22211223552225422241222222  24  34   6 12211224252211425214222122  24  38
  .
  .
  .
297 22441223242211322214221122  31  38 298 22251224512221422245222112  24  20
299 52211224242251422211212122  48  49 300 22211224262221422251222212  38  51
;
```

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

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

proc print data=res2;
   by 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 = 26);
   run;
```

Here is a small sample of the data.

```
                        Consumer Food Product Example

---------------------------------- Subj=1 Set=1 ----------------------------------

       Obs Age Income Brand     Price Micro Shelf  x1   x2   x5  x6  x8  c

         1  33   44   Client     1.29   .     .   1.29 1.89 1.99  0 1.99 2
         2  33   44   Extension  1.89   1     2   1.29 1.89 1.99  0 1.99 1
         3  33   44   Regional   1.99   .     .   1.29 1.89 1.99  0 1.99 2
         4  33   44   Private    0.00   1     .   1.29 1.89 1.99  0 1.99 2
         5  33   44   National   1.99   .     .   1.29 1.89 1.99  0 1.99 2
         6  33   44   None        .     .     .   1.29 1.89 1.99  0 1.99 2

---------------------------------- Subj=2 Set=2 ----------------------------------

       Obs Age Income Brand     Price Micro Shelf  x1   x2   x5   x6  x8 c

       163  52   82   Client     1.29   .     .   1.29 1.39 1.99 2.29  0 1
       164  52   82   Extension  1.39   2     2   1.29 1.39 1.99 2.29  0 2
       165  52   82   Regional   1.99   .     .   1.29 1.39 1.99 2.29  0 2
       166  52   82   Private    2.29   1     .   1.29 1.39 1.99 2.29  0 2
       167  52   82   National   0.00   .     .   1.29 1.39 1.99 2.29  0 2
       168  52   82   None        .     .     .   1.29 1.39 1.99 2.29  0 2
```

```
-------------------------------- Subj=3 Set=3 --------------------------------

      Obs Age Income Brand     Price Micro Shelf  x1   x2   x5   x6  x8 c

      325  51   136  Client     2.09   .     .   2.09 2.39 1.99 2.29  0 2
      326  51   136  Extension  2.39   1     2   2.09 2.39 1.99 2.29  0 1
      327  51   136  Regional   1.99   .     .   2.09 2.39 1.99 2.29  0 2
      328  51   136  Private    2.29   2     .   2.09 2.39 1.99 2.29  0 2
      329  51   136  National   0.00   .     .   2.09 2.39 1.99 2.29  0 2
      330  51   136  None        .     .     .   2.09 2.39 1.99 2.29  0 2

------------------------------ Subj=300 Set=26 -------------------------------

      Obs Age Income Brand     Price Micro Shelf  x1   x2   x5  x6   x8  c

    46795  38   51   Client     2.09   .     .   2.09 1.39 2.49  0 2.39 2
    46796  38   51   Extension  1.39   2     1   2.09 1.39 2.49  0 2.39 1
    46797  38   51   Regional   2.49   .     .   2.09 1.39 2.49  0 2.39 2
    46798  38   51   Private    0.00   1     .   2.09 1.39 2.49  0 2.39 2
    46799  38   51   National   2.39   .     .   2.09 1.39 2.49  0 2.39 2
    46800  38   51   None        .     .     .   2.09 1.39 2.49  0 2.39 2
```

You can see that the demographic information matches the raw data and is constant within subject. The rest of the data processing is virtually the same as well. The only difference is since we have demographic information we won't aggregate. There would have to be ties in both the demographics and choice for aggregation to have any effect.

```
data res3;
   set res2;
   if price = . then price = 0;
   if micro = . then micro = 2;
   if shelf = . then shelf = 2;
   if brand = 'None' or price ne 0;
   format micro shelf yn.;
   run;

proc print data=res3(obs=10); run;
```

```
                        Consumer Food Product Example

 Obs Subj Age Income Set Brand     Price Micro Shelf  x1   x2   x5   x6   x8  c

   1   1   33   44    1  Client     1.29  No    No   1.29 1.89 1.99 0.00 1.99 2
   2   1   33   44    1  Extension  1.89  Yes   No   1.29 1.89 1.99 0.00 1.99 1
   3   1   33   44    1  Regional   1.99  No    No   1.29 1.89 1.99 0.00 1.99 2
   4   1   33   44    1  National   1.99  No    No   1.29 1.89 1.99 0.00 1.99 2
   5   1   33   44    1  None       0.00  No    No   1.29 1.89 1.99 0.00 1.99 2
   6   1   33   44    2  Client     1.29  No    No   1.29 1.39 1.99 2.29 0.00 2
   7   1   33   44    2  Extension  1.39  No    No   1.29 1.39 1.99 2.29 0.00 1
   8   1   33   44    2  Regional   1.99  No    No   1.29 1.39 1.99 2.29 0.00 2
   9   1   33   44    2  Private    2.29  Yes   No   1.29 1.39 1.99 2.29 0.00 2
  10   1   33   44    2  None       0.00  No    No   1.29 1.39 1.99 2.29 0.00 2
```

We use PROC TRANSREG to code, adding **Age** and **Income** to the analysis.

```
proc transreg data=res3 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=12 12 zero='No' 'No')
         identity(x1 x2 x5 x6 x8) *
         class(brand / zero='None' separators=' ' ' on ') /
         lprefix=0  order=data;
   output out=coded(drop=_type_ _name_ intercept
                    where=(brand = 'None' or price ne 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 / zero='None' separators=" ', ')** the coefficients for age and income would be zero. This is because age and income are constant within each choice set and subject combination, that is 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;
   run;

proc print data=coded(obs=5 drop=x1) label; id brand price; var x1:; run;
proc print data=coded(obs=5 drop=x2) label; id brand price; var x2:; run;
proc print data=coded(obs=5 drop=x5) label; id brand price; var x5:; run;
proc print data=coded(obs=5 drop=x6) label; id brand price; var x6:; run;
proc print data=coded(obs=5 drop=x8) label; id brand price; var x8:; run;
```

Here is the coded data set for the first 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	National	Private	Shelf Talker	Microwave	c
Client	1.29	1	0	0	0	0	No	No	2
Extension	1.89	0	1	0	0	0	No	Yes	1
Regional	1.99	0	0	1	0	0	No	No	2
National	1.99	0	0	0	1	0	No	No	2
None	0.00	0	0	0	0	0	No	No	2

Here are the age by brand effects.

Consumer Food Product Example

Brand	Price	Age, Client	Age, Extension	Age, Regional	Age, National	Age, Private
Client	1.29	33	0	0	0	0
Extension	1.89	0	33	0	0	0
Regional	1.99	0	0	33	0	0
National	1.99	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, National	Income, Private
Client	1.29	44	0	0	0	0
Extension	1.89	0	44	0	0	0
Regional	1.99	0	0	44	0	0
National	1.99	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	National Price	Private Price
Client	1.29	1.29	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
National	1.99	0.00	0.00	0.00	1.99	0
None	0.00	0.00	0.00	0.00	0.00	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 National	CE, Client on Private
Client	1.29	1.29	0.00	0.00	0.00	0
Extension	1.89	0.00	1.29	0.00	0.00	0
Regional	1.99	0.00	0.00	1.29	0.00	0
National	1.99	0.00	0.00	0.00	1.29	0
None	0.00	0.00	0.00	0.00	0.00	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 National	CE, Extension on Private
Client	1.29	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
National	1.99	0.00	0.00	0.00	1.89	0
None	0.00	0.00	0.00	0.00	0.00	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 National	CE, Regional on Private
Client	1.29	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
National	1.99	0.00	0.00	0.00	1.99	0
None	0.00	0.00	0.00	0.00	0.00	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 National	CE, Private on Private
Client	1.29	0	0	0	0	0
Extension	1.89	0	0	0	0	0
Regional	1.99	0	0	0	0	0
National	1.99	0	0	0	0	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 National	CE, National on Private
Client	1.29	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
National	1.99	0.00	0.00	0.00	1.99	0
None	0.00	0.00	0.00	0.00	0.00	0

The PROC PHREG specification is the same as we have used before with nonaggregated data.

```
ods exclude CensoredSummary;
ods output  CensoredSummary=CS;

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

proc freq;
   tables event * censored / list;
   where n(stratum);
   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

Convergence Status

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

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	22944.047	9181.362
AIC	22944.047	9265.362
SBC	22944.047	9557.761

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	13762.6845	42	<.0001
Score	15106.4909	42	<.0001
Wald	4810.2570	42	<.0001

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Client	1	5.62995	0.97320	33.4658	<.0001
Extension	1	6.08310	0.99039	37.7259	<.0001
Regional	1	3.57233	1.56325	5.2221	0.0223
National	1	3.95103	1.23858	10.1758	0.0014
Private	1	3.60708	0.84966	18.0227	<.0001
Age, Client	1	0.01298	0.01620	0.6420	0.4230
Age, Extension	1	0.00707	0.01661	0.1812	0.6703
Age, Regional	1	0.02711	0.01777	2.3269	0.1272
Age, National	1	0.00389	0.01736	0.0502	0.8227
Age, Private	1	0.0002107	0.01572	0.0002	0.9893
Income, Client	1	-0.01053	0.00618	2.8990	0.0886
Income, Extension	1	-0.01181	0.00635	3.4582	0.0629
Income, Regional	1	-0.01405	0.00694	4.1040	0.0428
Income, National	1	-0.00898	0.00666	1.8197	0.1773
Income, Private	1	-0.00701	0.00599	1.3711	0.2416
Client Price	1	-0.67517	0.32292	4.3716	0.0365
Extension Price	1	-0.05907	0.31148	0.0360	0.8496
Regional Price	1	-1.18447	0.65400	3.2802	0.0701
National Price	1	-0.22350	0.38996	0.3285	0.5665
Private Price	1	-0.96861	0.28537	11.5207	0.0007
Shelf Talker Yes	1	0.61980	0.08426	54.1031	<.0001
Microwave Yes	1	0.72954	0.07339	98.8078	<.0001
CE, Client on Client	0	0	.	.	.
CE, Client on Extension	1	0.28262	0.31296	0.8155	0.3665
CE, Client on Regional	1	-0.06499	0.36901	0.0310	0.8602
CE, Client on National	1	-0.06432	0.33588	0.0367	0.8481
CE, Client on Private	1	0.30177	0.28829	1.0957	0.2952
CE, Extension on Client	1	0.47752	0.29668	2.5907	0.1075
CE, Extension on Extension	0	0	.	.	.
CE, Extension on Regional	1	-0.07767	0.35478	0.0479	0.8267
CE, Extension on National	1	0.20479	0.32510	0.3968	0.5287
CE, Extension on Private	1	0.15603	0.29926	0.2719	0.6021
CE, Regional on Client	1	0.26281	0.18095	2.1095	0.1464
CE, Regional on Extension	1	0.29836	0.18468	2.6099	0.1062
CE, Regional on Regional	0	0	.	.	.
CE, Regional on National	1	0.33442	0.18881	3.1371	0.0765
CE, Regional on Private	1	0.13920	0.18915	0.5416	0.4618
CE, Private on Client	1	-0.29959	0.28938	1.0718	0.3005
CE, Private on Extension	1	-0.20689	0.28951	0.5107	0.4748
CE, Private on Regional	1	-0.28139	0.32178	0.7647	0.3819
CE, Private on National	1	-0.40533	0.30973	1.7126	0.1906
CE, Private on Private	0	0	.	.	.

```
CE, National on Client       1     0.03005     0.22031      0.0186      0.8915
CE, National on Extension    1     0.08905     0.22332      0.1590      0.6901
CE, National on Regional     1     0.33125     0.29231      1.2842      0.2571
CE, National on National     0           0           .           .           .
CE, National on Private      1     0.00142     0.23558      0.0000      0.9952
```

```
                    Consumer Food Product Example

                          The FREQ Procedure

                                                        Cumulative    Cumulative
  Event    Censored    Frequency     Percent     Frequency      Percent
  ---------------------------------------------------------------------------
      1           2         1200       15.38          1200        15.38
      1           3         2100       26.92          3300        42.31
      1           4         4500       57.69          7800       100.00
```

The coefficients for the age and income variables are not significant in this analysis. In previous examples, when we used PROC FREQ to summarize the summary table, the PROC FREQ output had only one line. In this case, since our choice sets have either 3, 4, or 5 alternatives, we have three rows, one for each size choice set.

When Balance is of Primary Importance

Sometimes it is very important to get a design that is as close to balanced as possible. More generally, sometimes there is some criterion in which you want the design to be good that is not perfectly correlated with efficiency. A strategy for those cases is to generate a number of designs, keep the most efficient few, then use the one that is best in terms of the secondary criterion of interest. In this example, we use a macro to repeatedly generate 1000 designs, output 200 of the most efficient, then evaluate their balance.

```
title 'Consumer Food Product Example';

%mktdes(factors=x1 x2=4 x3 x4=2 x5 x6=3 x7=2 x8=3, n=26, big=5000,
        where=(2 <= ((x1 < 4) + (x2 < 4) + (x5 < 3) +
               (x6 < 3) + (x8 < 3)) <= 4),
        run=plan)

%macro baleval(niter);

%do iter = 1 %to &niter;

   data _null_;
      seed = abs(1e6 * (1e-5 + time() - floor(time())));
      call symput('seed', compress(put(seed, 12.0)));
      run;

   ods output efficiencies(persist=run)=e;

   %mktdes(factors=x1 x2=4 x3 x4=2 x5 x6=3 x7=2 x8=3, n=26, big=5000,
           interact=x2*x3 x2*x4 x3*x4 x6*x7,
           iter=5, keep=1, run=optex, procopts=seed=&seed)
```

```
   proc iml;
      use design; read all into x;
      use e(keep=DCriterion); read all into deff;
      if &iter > 1 then do;
         use balance; read all into balance; close balance;
         end;
      p = ncol(x); n = nrow(x);
      bal = j(1, p, 0);
      do j = 1 to p;
         x1 = design(x[,j])[+,];
         nc = ncol(x1);
         bal[,j] = ssq(x1 - n / nc) / (nc - 1);
         end;
      bal = balance // (&seed || deff || sum(bal) || bal);
      create balance from bal; append from bal;
      quit;

   proc means min data=balance noprint;
      output min(col3)=min out=min;
      run;
   data small(drop=min);
      set balance;
      if _n_ = 1 then set min(keep=min);
      if abs(col3 - min) <= 1e-6;
      run;

   proc sort; by descending col2; run;

   proc print label noobs;
      label col1 = 'Seed' col2 = 'D-Efficiency' col3 = 'Var Sum';
      title2 'Design Balances';
      format col4-col11 5.2;
      run;

   %end;

%mend;
%baleval(200)

proc print label noobs data=balance;
   label col1 = 'Seed' col2 = 'D-Efficiency' col3 = 'Var Sum';
   title2 'Design Balances';
   format col4-col11 5.2;
   run;
```

This code first uses the %MKTDES macro to create the candidate set. Then the %BALEVAL balance evaluation macro is defined. This macro has a loop to repeatedly create designs and evaluate their balance. The DATA step creates a random number seed based on clock time. Hence, no two runs of this macro will produce the same results. The **ods output** statement using **(persist=run)** creates an output SAS data set from the efficiencies table of PROC OPTEX. The %MKTDES macro is run to create five designs and output the best one. The random number seed generated by the DATA step is used. Each design is read into PROC IML to report on balance. The statements **use design** and **read all into x** read the design into PROC IML. The statements

```
if &iter > 1 then do;
   use balance; read all into balance; close balance;
   end;
```

read previous results to which these results are appended. The statements **p = ncol(x)** and **n = nrow(x)**

store the number of columns and rows in the design matrix. The statement **bal = j(1, p, 0)** initializes the results vector BAL to 0. The statements in the **do** loop compute the variance of the frequency of occurrence for each level within each factor. The statement **x1 = design(x[,j])[+,]** counts the number of times each level appears. The statement **bal[,j] = ssq(x1 - n / nc) / (nc - 1)** computes the variance. Note that **n / nc** is the expected frequency for an **nc**-level factor. Then **bal = balance // (&seed || deff || sum(bal) || bal)** stores the random number seed, D-efficiency, sum of the variances and each column's variance in a SAS data set. The PROC MEANS, DATA, PROC SORT and PROC PRINT steps print information about the most balanced design found so far. Outside the macro, all of the balance information is printed. Here are some of the results.

```
                    Consumer Food Product Example
                           Design Balances

  Seed  D-Efficiency Var Sum  COL4  COL5  COL6  COL7  COL8  COL9 COL10 COL11

 381303    84.0781   1.66667  0.33  0.33  0.00  0.00  0.33  0.33  0.00  0.33

                    Consumer Food Product Example
                           Design Balances

  Seed  D-Efficiency Var Sum  COL4  COL5  COL6  COL7  COL8  COL9 COL10 COL11

 375419    83.5921   10.3333  0.33  1.00  2.00  2.00  2.33  0.33  0.00  2.33
 707520    83.3048   12.0000  1.67  0.33  0.00  0.00  2.33  1.33  2.00  4.33
 873439    83.4668   11.3333  0.33  1.00  0.00  0.00  6.33  1.33  0.00  2.33
      .
      .
      .
 381303    84.0781    1.6667  0.33  0.33  0.00  0.00  0.33  0.33  0.00  0.33
      .
      .
      .
  43072    83.8992   28.0000  1.00  1.00  8.00  8.00  0.33  1.33  8.00  0.33
      .
      .
      .
```

The most nearly balanced of the designs has a variance sum of 1.6667. All factors are either perfectly balanced or as close to perfectly balanced as a design with these factors in 26 runs can be. Perfect balance is impossible for the three-level and four-level factors since three and four do not divide 26, so a standard deviation sum of zero is not possible. This design has a D-efficiency of 84.0781 compared with 85.3985 (the D-efficiency for the best design we found previously). This design can be regenerated by running the %MKTDES macro with **procopts=seed=381303** specified.

```
%mktdes(factors=x1 x2=4 x3 x4=2 x5 x6=3 x7=2 x8=3, n=26, big=5000,
        where=(2 <= ((x1 < 4) + (x2 < 4) + (x5 < 3) +
               (x6 < 3) + (x8 < 3)) <= 4),
        interact=x2*x3 x2*x4 x3*x4 x6*x7,
        iter=5, keep=1, procopts=seed=381303)

proc freq; run;
```

Consumer Food Product Example

The FREQ Procedure

x1	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	7	26.92	7	26.92
2	7	26.92	14	53.85
3	6	23.08	20	76.92
4	6	23.08	26	100.00

x2	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	6	23.08	6	23.08
2	6	23.08	12	46.15
3	7	26.92	19	73.08
4	7	26.92	26	100.00

x3	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	13	50.00	13	50.00
2	13	50.00	26	100.00

x4	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	13	50.00	13	50.00
2	13	50.00	26	100.00

x5	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	9	34.62	9	34.62
2	8	30.77	17	65.38
3	9	34.62	26	100.00

x6	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	8	30.77	8	30.77
2	9	34.62	17	65.38
3	9	34.62	26	100.00

x7	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	13	50.00	13	50.00
2	13	50.00	26	100.00

x8	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	8	30.77	8	30.77
2	9	34.62	17	65.38
3	9	34.62	26	100.00

Allocation of Prescription Drugs

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, might be “5 for drug 1, none for drug 2, 3 for drug 3, and 2 for drug 4.”

This example will show how to design, process, code, and analyze an allocation study. 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.

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 261.) We specify ten 3’s for the 10 three-level factors.

```
title 'Allocation of Prescription Drugs';

%mktruns( 3 3 3 3 3 3 3 3 3 3 )
```

Allocation of Prescription Drugs

Some Reasonable Design Sizes (Saturated=21)	Violations	Cannot Be Divided By
27	0	
36	0	
45	0	
54	0	
63	0	
72	0	
81	0	
90	0	
99	0	
108	0	

We need at least 21 choice sets and we see the optimal sizes are all divisible by nine. We will use 27 choice sets.

Next, we use the %MKTDES macro to create the design. In addition, one more factor, **Block**, is added to the design. This factor will be used to block the design into three blocks of size 9. PROC FORMAT is used to assign actual prices of $50, $75, $100 to the levels 1, 2, and 3. A DATA step and PROC SORT are used to assign formats and sort the design into a random order within blocks.

```
%let nalts = 10;

%mktdes(factors=Brand1-Brand&nalts=3 Block=3, n=27, procopts=seed=7654321)

proc format;
   value price 1 = ' $50' 2 = ' $75' 3 = '$100' . = ' ';
   run;
```

```
data sasuser.allocdes;
   r = uniform(7);
   set design;
   format Brand: price.;
   run;

proc sort data=sasuser.allocdes out=sasuser.allocdes(drop=r);
   by block r;
   run;
proc print data=sasuser.allocdes; by block; run;

proc freq;
   tables block * (brand:) / list;
   run;
```

For this problem, there exists a perfectly 100% efficient, orthogonal and balanced design. In fact, PROC FACTEX produces it and the PROC OPTEX step is not actually necessary. Each level of each factor occurs three times in each block.

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
2	100.0000	100.0000	100.0000	0.9230
3	100.0000	100.0000	100.0000	0.9230
4	100.0000	100.0000	100.0000	0.9230
5	100.0000	100.0000	100.0000	0.9230

Allocation of Prescription Drugs

Block=1

Obs	Brand1	Brand2	Brand3	Brand4	Brand5	Brand6	Brand7	Brand8	Brand9	Brand10
1	$50	$75	$50	$100	$100	$100	$75	$75	$75	$50
2	$100	$100	$100	$50	$75	$100	$75	$100	$50	$100
3	$50	$50	$50	$50	$50	$50	$50	$50	$50	$50
4	$75	$75	$75	$50	$100	$75	$100	$75	$50	$75
5	$75	$100	$75	$100	$75	$50	$50	$100	$75	$75
6	$100	$50	$100	$100	$50	$75	$100	$50	$75	$100
7	$100	$75	$100	$75	$100	$50	$50	$75	$100	$100
8	$50	$100	$50	$75	$75	$75	$100	$100	$100	$50
9	$75	$50	$75	$75	$50	$100	$75	$50	$100	$75

---------- Block=2 ----------

Obs	Brand1	Brand2	Brand3	Brand4	Brand5	Brand6	Brand7	Brand8	Brand9	Brand10
10	$50	$50	$100	$75	$75	$75	$75	$75	$75	$75
11	$100	$75	$75	$100	$50	$75	$75	$100	$50	$50
12	$75	$100	$50	$50	$100	$75	$75	$50	$100	$100
13	$75	$75	$50	$75	$50	$100	$50	$100	$75	$100
14	$100	$50	$75	$50	$75	$100	$50	$75	$100	$50
15	$100	$100	$75	$75	$100	$50	$100	$50	$75	$50
16	$75	$50	$50	$100	$75	$50	$100	$75	$50	$100
17	$50	$100	$100	$100	$100	$100	$50	$50	$50	$75
18	$50	$75	$100	$50	$50	$50	$100	$100	$100	$75

---------- Block=3 ----------

Obs	Brand1	Brand2	Brand3	Brand4	Brand5	Brand6	Brand7	Brand8	Brand9	Brand10
19	$75	$50	$100	$50	$100	$75	$50	$100	$75	$50
20	$100	$50	$50	$75	$100	$50	$75	$100	$50	$75
21	$75	$100	$100	$75	$50	$100	$100	$75	$50	$50
22	$50	$100	$75	$50	$50	$50	$75	$75	$75	$100
23	$100	$75	$50	$50	$75	$100	$100	$50	$75	$75
24	$100	$100	$50	$100	$50	$75	$50	$75	$100	$75
25	$50	$75	$75	$75	$75	$75	$50	$50	$50	$100
26	$50	$50	$75	$100	$100	$100	$100	$100	$100	$100
27	$75	$75	$100	$100	$75	$50	$75	$50	$100	$50

Allocation of Prescription Drugs

The FREQ Procedure

Block	Brand1	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	$50	3	11.11	3	11.11
1	$75	3	11.11	6	22.22
1	$100	3	11.11	9	33.33
2	$50	3	11.11	12	44.44
2	$75	3	11.11	15	55.56
2	$100	3	11.11	18	66.67
3	$50	3	11.11	21	77.78
3	$75	3	11.11	24	88.89
3	$100	3	11.11	27	100.00

Block	Brand2	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	$50	3	11.11	3	11.11
1	$75	3	11.11	6	22.22
1	$100	3	11.11	9	33.33
2	$50	3	11.11	12	44.44
2	$75	3	11.11	15	55.56
2	$100	3	11.11	18	66.67
3	$50	3	11.11	21	77.78
3	$75	3	11.11	24	88.89
3	$100	3	11.11	27	100.00

Block	Brand3	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	$50	3	11.11	3	11.11
1	$75	3	11.11	6	22.22
1	$100	3	11.11	9	33.33
2	$50	3	11.11	12	44.44
2	$75	3	11.11	15	55.56
2	$100	3	11.11	18	66.67
3	$50	3	11.11	21	77.78
3	$75	3	11.11	24	88.89
3	$100	3	11.11	27	100.00

Block	Brand4	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	$50	3	11.11	3	11.11
1	$75	3	11.11	6	22.22
1	$100	3	11.11	9	33.33
2	$50	3	11.11	12	44.44
2	$75	3	11.11	15	55.56
2	$100	3	11.11	18	66.67
3	$50	3	11.11	21	77.78
3	$75	3	11.11	24	88.89
3	$100	3	11.11	27	100.00

Block	Brand5	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	$50	3	11.11	3	11.11
1	$75	3	11.11	6	22.22
1	$100	3	11.11	9	33.33
2	$50	3	11.11	12	44.44
2	$75	3	11.11	15	55.56
2	$100	3	11.11	18	66.67
3	$50	3	11.11	21	77.78
3	$75	3	11.11	24	88.89
3	$100	3	11.11	27	100.00

Block	Brand6	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	$50	3	11.11	3	11.11
1	$75	3	11.11	6	22.22
1	$100	3	11.11	9	33.33
2	$50	3	11.11	12	44.44
2	$75	3	11.11	15	55.56
2	$100	3	11.11	18	66.67
3	$50	3	11.11	21	77.78
3	$75	3	11.11	24	88.89
3	$100	3	11.11	27	100.00

Block	Brand7	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	$50	3	11.11	3	11.11
1	$75	3	11.11	6	22.22
1	$100	3	11.11	9	33.33
2	$50	3	11.11	12	44.44
2	$75	3	11.11	15	55.56
2	$100	3	11.11	18	66.67
3	$50	3	11.11	21	77.78
3	$75	3	11.11	24	88.89
3	$100	3	11.11	27	100.00

Block	Brand8	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	$50	3	11.11	3	11.11
1	$75	3	11.11	6	22.22
1	$100	3	11.11	9	33.33
2	$50	3	11.11	12	44.44
2	$75	3	11.11	15	55.56
2	$100	3	11.11	18	66.67
3	$50	3	11.11	21	77.78
3	$75	3	11.11	24	88.89
3	$100	3	11.11	27	100.00

Block	Brand9	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	$50	3	11.11	3	11.11
1	$75	3	11.11	6	22.22
1	$100	3	11.11	9	33.33
2	$50	3	11.11	12	44.44
2	$75	3	11.11	15	55.56
2	$100	3	11.11	18	66.67
3	$50	3	11.11	21	77.78
3	$75	3	11.11	24	88.89
3	$100	3	11.11	27	100.00

Block	Brand10	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	$50	3	11.11	3	11.11
1	$75	3	11.11	6	22.22
1	$100	3	11.11	9	33.33
2	$50	3	11.11	12	44.44
2	$75	3	11.11	15	55.56
2	$100	3	11.11	18	66.67
3	$50	3	11.11	21	77.78
3	$75	3	11.11	24	88.89
3	$100	3	11.11	27	100.00

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
.
.
.
;
```

The first step in creating an analysis data set for an allocation study is to 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 none of the alternatives are 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 alternative 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); 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

Obs	Block	Set	Brand	Count
1	1	1	Brand 1	0
2	1	1	Brand 2	0
3	1	1	Brand 3	8
4	1	1	Brand 4	0
5	1	1	Brand 5	2
6	1	1	Brand 6	0
7	1	1	Brand 7	0
8	1	1	Brand 8	0
9	1	1	Brand 9	0
10	1	1	Brand 10	0
11	1	1		0
12	1	2	Brand 1	0
13	1	2	Brand 2	0
14	1	2	Brand 3	8
15	1	2	Brand 4	0
16	1	2	Brand 5	0
17	1	2	Brand 6	0
18	1	2	Brand 7	2
19	1	2	Brand 8	0
20	1	2	Brand 9	0
21	1	2	Brand 10	0
22	1	2		0
23	1	3	Brand 1	0
24	1	3	Brand 2	0
25	1	3	Brand 3	0
26	1	3	Brand 4	0
27	1	3	Brand 5	0
28	1	3	Brand 6	0
29	1	3	Brand 7	0
30	1	3	Brand 8	0
31	1	3	Brand 9	10
32	1	3	Brand 10	0
33	1	3		0

The next step aggregates the data. It stores in the variable **Count** the number of times each alternative of each choice set was chosen. This creates a data set with 297 observations (3 blocks $\times$ 9 sets $\times$ 11 alternatives = 297).

```
* Aggregate, store the results back in count.;

proc summary data=allocs nway missing;
   class set brand;
   output sum(count)=Count out=allocs(drop=_type_ _freq_);
   run;
```

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 price.; run;
```

Allocation of Prescription Drugs

Obs	Set	Brand	Price
1	1	Brand 1	$50
2	1	Brand 2	$75
3	1	Brand 3	$50
4	1	Brand 4	$100
5	1	Brand 5	$100
6	1	Brand 6	$100
7	1	Brand 7	$75
8	1	Brand 8	$75
9	1	Brand 9	$75
10	1	Brand 10	$50
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 price.;
   run;
proc print data=allocs2(obs=22);
   var brand price count;
   sum count;
   by notsorted 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 \$50 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=1

Obs	Brand	Price	Count
1			0
2	Brand 1	\$50	103
3	Brand 2	\$75	58
4	Brand 3	\$50	318
5	Brand 4	\$100	99
6	Brand 5	\$100	54
7	Brand 6	\$100	83
8	Brand 7	\$75	71
9	Brand 8	\$75	58
10	Brand 9	\$75	100
11	Brand 10	\$50	56
---			-----
Set			1000

Set=2

Obs	Brand	Price	Count
12			10
13	Brand 1	\$100	73
14	Brand 2	\$100	76
15	Brand 3	\$100	342
16	Brand 4	\$50	55
17	Brand 5	\$75	50
18	Brand 6	\$100	77
19	Brand 7	\$75	95
20	Brand 8	\$100	71
21	Brand 9	\$50	72
22	Brand 10	\$100	79
---			-----
Set			1000

At this point, the data set contains 297 observations (27 choice sets times 11 alternatives) showing the number of times each alternative was chosen. This data set must be augmented to also include the number of times each alternative was not chosen. For example, in the first choice set, brand 1 was chosen 103 times, which means it was not chosen $0+58+318+99+54+83+71+58+100+56=897$ times. We use a macro, %MKTALLO for "marketing allocation study" to process the data. We specify the input **data=allocs2** data set, the output **out=allocs3** data set, the number of alternatives including the constant (**nalts=%eval(&nalts + 1)**), the variables in the data set except the frequency variable (**vars=set brand price**), and the frequency variable (**freq=Count**). The macro counts how many times each alternative was chosen and not chosen and writes the results to the **out=** data set along with the usual **c** = 1 for chosen and **c** = 2 for unchosen.

```
%mktallo(data=allocs2, out=allocs3, nalts=%eval(&nalts + 1),
         vars=set brand price, freq=Count)

proc print data=allocs3(obs=22);
   var set brand price count c;
   run;
```

The first 22 records of the allocation data set are shown next.

Allocation of Prescription Drugs

Obs	Set	Brand	Price	Count	c
1	1			0	1
2	1			1000	2
3	1	Brand 1	$50	103	1
4	1	Brand 1	$50	897	2
5	1	Brand 2	$75	58	1
6	1	Brand 2	$75	942	2
7	1	Brand 3	$50	318	1
8	1	Brand 3	$50	682	2
9	1	Brand 4	$100	99	1
10	1	Brand 4	$100	901	2
11	1	Brand 5	$100	54	1
12	1	Brand 5	$100	946	2
13	1	Brand 6	$100	83	1
14	1	Brand 6	$100	917	2
15	1	Brand 7	$75	71	1
16	1	Brand 7	$75	929	2
17	1	Brand 8	$75	58	1
18	1	Brand 8	$75	942	2
19	1	Brand 9	$75	100	1
20	1	Brand 9	$75	900	2
21	1	Brand 10	$50	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.

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 71 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.

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	502505.13	489044.70
AIC	502505.13	489068.70
SBC	502505.13	489167.14

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	13460.4283	12	<.0001
Score	18359.1337	12	<.0001
Wald	14099.9841	12	<.0001

Multinomial Logit Parameter Estimates

		DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Brand	1	1	2.09228	0.06766	956.1751	<.0001
Brand	2	1	2.08440	0.06769	948.1852	<.0001
Brand	3	1	3.53545	0.06484	2973.0894	<.0001
Brand	4	1	2.09032	0.06767	954.1811	<.0001
Brand	5	1	2.07845	0.06771	942.1769	<.0001
Brand	6	1	2.02852	0.06790	892.5031	<.0001
Brand	7	1	2.06241	0.06777	926.0726	<.0001
Brand	8	1	2.07895	0.06771	942.6783	<.0001
Brand	9	1	2.11027	0.06760	974.5421	<.0001
Brand	10	1	2.05684	0.06779	920.5101	<.0001
$50		1	0.04627	0.01617	8.1842	0.0042
$75		1	-0.02108	0.01640	1.6525	0.1986
$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. There is a very small effect for Price = $50.

*We will soon see that this fact should be ignored.

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. Then our frequencies will not be integers. We specify the **notruncate** option on PROC PHREG 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 by the data manipulations so p-values are not interpretable in an allocation study.

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

Criterion	Without Covariates	With Covariates
-2 LOG L	37816.553	36470.511
AIC	37816.553	36494.511
SBC	37816.553	36565.323

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	1346.0428	12	<.0001
Score	1835.9134	12	<.0001
Wald	1409.9984	12	<.0001

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Brand 1	1	2.09228	0.21397	95.6175	<.0001
Brand 2	1	2.08440	0.21406	94.8185	<.0001
Brand 3	1	3.53545	0.20504	297.3089	<.0001
Brand 4	1	2.09032	0.21399	95.4181	<.0001
Brand 5	1	2.07845	0.21413	94.2177	<.0001
Brand 6	1	2.02852	0.21472	89.2503	<.0001
Brand 7	1	2.06241	0.21432	92.6073	<.0001
Brand 8	1	2.07895	0.21412	94.2678	<.0001
Brand 9	1	2.11027	0.21377	97.4542	<.0001
Brand 10	1	2.05684	0.21438	92.0510	<.0001
$50	1	0.04627	0.05114	0.8184	0.3656
$75	1	-0.02108	0.05187	0.1652	0.6844
$100	0	0	.	.	.

Chair Design with Generic Attributes

This study illustrates creating an experimental design for a purely generic choice model. In a purely generic study there are no brands, just bundles of attributes. Say a particular manufacturer is interested in designing one or more new chairs. They can vary the attributes of the chairs, present subjects with descriptions of competing chair designs, and model the effects of the attributes on choice.

Factor	Attribute	Levels
X1	Color	3 Colors
X2	Back	3 Styles
X3	Seat	3 Styles
X4	Arm Rest	3 Styles
X5	Material	3 Materials

Assume subjects will be shown descriptions of three chairs at a time. If we were to model our design after the approach used in previous examples, we would use the %MKTDES 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. Here is how we might have made the design.

```
title 'Generic Chair Attributes';

* This design will not be used;
%mktdes(factors=x1-x15=3, n=33)
```

The %MKTDES 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 30 choice sets is overkill for a problem like this. Since we do not need a complicated model for this example, we will instead use a different approach. Recall the discussion of linear design efficiency, choice model design efficiency, and using linear design efficiency as a surrogate for choice design efficiency from the "Preliminaries" section starting on page 68. Instead of using linear design efficiency as a surrogate for choice design efficiency, we can directly optimize choice design efficiency given an assumed model and β parameter vector.

Purely Generic Attributes, Alternative Swapping

This part of the example will illustrate creating an efficient choice design using an algorithm that swaps individual alternatives (as opposed to entire choice sets) in and out of the design. First, we will use the %MKTDES macro to create a candidate set. It will consist of 5 three-level factors, one for each of the five generic attributes. It will also consist of three flag variables, **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 observations for the different alternatives may be all different, all the same, or any pattern in between depending on the problem. For example, the candidate set may contain one observation that is only used for the last, constant alternative. 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. In this purely generic case, each flag variable consists entirely of ones indicating that any candidate can appear in any alternative. The following code creates the candidates.

```
%mktdes(factors=x1-x5=3 f1-f3=1, run=factex)

proc print; run;
```

We specified the **run=factex** option in the %MKTDES macro. We do not need to run PROC OPTEX since we are just creating a candidate set. The columns **x1-x5** are the generic attributes, and **f1-f3** are the flags. The canidate set has 27 observations, although we certainly could create larger candidate sets for this problem by specifying the **size=** option.

Generic Chair Attributes

Obs	x1	x2	x3	x4	x5	f1	f2	f3
1	1	1	1	1	1	1	1	1
2	1	1	2	3	3	1	1	1
3	1	1	3	2	2	1	1	1
4	1	2	1	3	3	1	1	1
5	1	2	2	2	2	1	1	1
6	1	2	3	1	1	1	1	1
7	1	3	1	2	2	1	1	1
8	1	3	2	1	1	1	1	1
9	1	3	3	3	3	1	1	1
10	2	1	1	3	2	1	1	1
11	2	1	2	2	1	1	1	1
12	2	1	3	1	3	1	1	1
13	2	2	1	2	1	1	1	1
14	2	2	2	1	3	1	1	1
15	2	2	3	3	2	1	1	1
16	2	3	1	1	3	1	1	1
17	2	3	2	3	2	1	1	1
18	2	3	3	2	1	1	1	1
19	3	1	1	2	3	1	1	1
20	3	1	2	1	2	1	1	1
21	3	1	3	3	1	1	1	1
22	3	2	1	1	2	1	1	1
23	3	2	2	3	1	1	1	1
24	3	2	3	2	3	1	1	1
25	3	3	1	3	1	1	1	1
26	3	3	2	2	3	1	1	1
27	3	3	3	1	2	1	1	1

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 %CHOICEFF autocall macro to do this. (All of the autocall macros used in this report are documented starting on page 261.) 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 %CHOICEFF macro uses a modified Federov algorithm (Federov (1972) and Cook and Nachtsheim (1980)) to optimize the choice model variance matrix. This specification requests a generic design with 9 choice sets each consisting of three alternatives.

```
%choiceff(data=cand1, model=class(x1-x5), nsets=9,
          seed=9999, flags=f1-f3, beta=zero);
```

The **`data=cand1`** option names the input data set of candidates. The **`model=class(x1-x5)`** option specifies the most general model that might be considered at analysis time. The **`nsets=9`** 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 %MKTDES linear-design approach. The **`seed=9999`** 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 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. Note that this is not output from PROC OPTEX. The 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. Also note that these efficiencies are not on a 0 to 100 scale. In fact for this design, 1.7320508076 is the optimum. Iterations can be slow, so by default only two sets of iterations are performed. You can increase this by specifying the **maxiter=** option.

Design	Iteration	D-Efficiency	D-Error
1	0	0.79212136	1.2624328172
	1	1.632279118	0.6126403193
	2	1.7320508076	0.5773502692
	3	1.7320508076	0.5773502692

Design	Iteration	D-Efficiency	D-Error
2	0	0.8952159032	1.1170489671
	1	1.5464072418	0.6466601895
	2	1.679907245	0.5952709609
	3	1.679907245	0.5952709609

Next, the macro shows which design it chose and the final efficiency and D-Error (D-Efficiency = 1 / D-Error).

```
Final Results:   Design     = 1
                 Efficiency = 1.7320508076
                 D-Error    = 0.5773502692
```

Next, it shows the variance, standard error, and *df* for each effect. It is important to ensure that each effect is estimable: (*df* = 1).

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	

The data set BEST contains the final, best design found.

```
proc print; run;
```

The data set contains in the variable **Design** the number of the design with the maximum efficiency, in **Efficiency** the efficiency of this design, in **Index** the candidate set observation number, in **Set** the choice set number, in **Prob** the probability that this alternative will be chosen given β, in **n** the observation number, in **x1-x5** the design, and in **f1-f3** the flags.

Generic Chair Attributes

Obs	Design	Efficiency	Index	Set	Prob	n	x1	x2	x3	x4	x5	f1	f2	f3
1	1	1.73205	6	1	0.33333	1	1	2	3	1	1	1	1	1
2	1	1.73205	10	1	0.33333	2	2	1	1	3	2	1	1	1
3	1	1.73205	26	1	0.33333	3	3	3	2	2	3	1	1	1
4	1	1.73205	9	2	0.33333	4	1	3	3	3	3	1	1	1
5	1	1.73205	22	2	0.33333	5	3	2	1	1	2	1	1	1
6	1	1.73205	11	2	0.33333	6	2	1	2	2	1	1	1	1
7	1	1.73205	16	3	0.33333	7	2	3	1	1	3	1	1	1
8	1	1.73205	23	3	0.33333	8	3	2	2	3	1	1	1	1
9	1	1.73205	3	3	0.33333	9	1	1	3	2	2	1	1	1
10	1	1.73205	2	4	0.33333	10	1	1	2	3	3	1	1	1
11	1	1.73205	13	4	0.33333	11	2	2	1	2	1	1	1	1
12	1	1.73205	27	4	0.33333	12	3	3	3	1	2	1	1	1
13	1	1.73205	21	5	0.33333	13	3	1	3	3	1	1	1	1
14	1	1.73205	14	5	0.33333	14	2	2	2	1	3	1	1	1
15	1	1.73205	7	5	0.33333	15	1	3	1	2	2	1	1	1
16	1	1.73205	19	6	0.33333	16	3	1	1	2	3	1	1	1
17	1	1.73205	8	6	0.33333	17	1	3	2	1	1	1	1	1
18	1	1.73205	15	6	0.33333	18	2	2	3	3	2	1	1	1
19	1	1.73205	4	7	0.33333	19	1	2	1	3	3	1	1	1
20	1	1.73205	18	7	0.33333	20	2	3	3	2	1	1	1	1
21	1	1.73205	20	7	0.33333	21	3	1	2	1	2	1	1	1
22	1	1.73205	12	8	0.33333	22	2	1	3	1	3	1	1	1
23	1	1.73205	5	8	0.33333	23	1	2	2	2	2	1	1	1
24	1	1.73205	25	8	0.33333	24	3	3	1	3	1	1	1	1

25	1	1.73205	1	9	0.33333	25	1	1	1	1	1	1	1	1
26	1	1.73205	24	9	0.33333	26	3	2	3	2	3	1	1	1
27	1	1.73205	17	9	0.33333	27	2	3	2	3	2	1	1	1

This design has 27 runs (9 choice sets $\times$ 3 alternatives). This happens to be the same number as the number of candidate alternatives from the fractional-factorial design, although in general these numbers do not have to be the same. Notice the **Index** variable. It contains the candidate set observation number, that is the number of the observation in the candidate set that matches this alterative. Notice that in this problem, each number appears once, so each candidate was selected for inclusion in the design exactly once. For this problem (a generic design with 5 three-level factors, 9 choice sets, three alternatives, and $\beta = 0$) the optimal design can be constructed by optimally sorting the 27 alternatives in a fractional-factorial design. Also notice that in this design, each level occurs exactly once in each factor and each choice set.

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 %MKTDES macro just like before to make a design for the nonconstant alternatives and store the results in a candidate set CAND1. Then we will use the %MKTDES macro again to create the constant alternative. Next, we use a DATA step to combine the two candidate sets.

```
%mktdes(factors=x1-x5=3 f1-f3=1, run=factex)

%mktdes(factors=x1-x5=1 f4=1,    run=plan, cand=cand2)

data cand3; set cand1 cand2; run;

proc print; run;
```

Here is the candidate set.

Generic Chair Attributes

Obs	x1	x2	x3	x4	x5	f1	f2	f3	f4
1	1	1	1	1	1	1	1	1	.
2	1	1	2	3	3	1	1	1	.
3	1	1	3	2	2	1	1	1	.
4	1	2	1	3	3	1	1	1	.
5	1	2	2	2	2	1	1	1	.
6	1	2	3	1	1	1	1	1	.
7	1	3	1	2	2	1	1	1	.
8	1	3	2	1	1	1	1	1	.
9	1	3	3	3	3	1	1	1	.
10	2	1	1	3	2	1	1	1	.
11	2	1	2	2	1	1	1	1	.
12	2	1	3	1	3	1	1	1	.
13	2	2	1	2	1	1	1	1	.
14	2	2	2	1	3	1	1	1	.
15	2	2	3	3	2	1	1	1	.
16	2	3	1	1	3	1	1	1	.
17	2	3	2	3	2	1	1	1	.
18	2	3	3	2	1	1	1	1	.

19	3	1	1	2	3	1	1	1	.
20	3	1	2	1	2	1	1	1	.
21	3	1	3	3	1	1	1	1	.
22	3	2	1	1	2	1	1	1	.
23	3	2	2	3	1	1	1	1	.
24	3	2	3	2	3	1	1	1	.
25	3	3	1	3	1	1	1	1	.
26	3	3	2	2	3	1	1	1	.
27	3	3	3	1	2	1	1	1	.
28	1	1	1	1	1	.	.	.	1

The first 27 observations may be used for any of the first three alternatives and the $28th$ observation may only be used for fourth or constant alternative. In this example, the constant alternative is composed solely from the first level of each factor. Of course this could be changed depending on the situation.

The %CHOICEFF macro invocation is the same as before, except now we have four flags, and now we ask for more iterations.

```
%choiceff(data=cand3, model=class(x1-x5), nsets=9, maxiter=10,
          seed=9999, flags=f1-f4, beta=zero);

proc print; 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

Design	Iteration	D-Efficiency	D-Error
1	0	0.8599706425	1.1628303928
	1	1.3044476431	0.766608001
	2	1.4152340172	0.7065969217
	3	1.4243278149	0.7020855659
	4	1.4282701243	0.7001476702

.
.
.

Design	Iteration	D-Efficiency	D-Error
8	0	0.7424273661	1.3469331085
	1	1.3594472951	0.7355930631
	2	1.436915059	0.6959353608
	3	1.4962970242	0.6683165066
	4	1.4962970242	0.6683165066

.
.
.

Generic Chair Attributes

Final Results: Design = 8
Efficiency = 1.4962970242
D-Error = 0.6683165066

Generic Chair Attributes

n	Variable Name	Label	Variance	DF	Standard Error
1	x11	x1 1	0.76376	1	0.87394
2	x12	x1 2	0.85986	1	0.92729
3	x21	x2 1	0.77177	1	0.87851
4	x22	x2 2	0.88889	1	0.94281
5	x31	x3 1	0.76376	1	0.87394
6	x32	x3 2	0.85986	1	0.92729
7	x41	x4 1	0.76376	1	0.87394
8	x42	x4 2	0.85986	1	0.92729
9	x51	x5 1	0.77177	1	0.87851
10	x52	x5 2	0.88889	1	0.94281
				==	
				10	

Generic Chair Attributes

Obs	Design	Efficiency	Index	Set	Prob	n	x1	x2	x3	x4	x5	f1	f2	f3	f4
1	8	1.49630	26	1	0.25	253	3	3	2	2	3	1	1	1	.
2	8	1.49630	21	1	0.25	254	3	1	3	3	1	1	1	1	.
3	8	1.49630	15	1	0.25	255	2	2	3	3	2	1	1	1	.
4	8	1.49630	28	1	0.25	256	1	1	1	1	1	.	.	.	1
5	8	1.49630	7	2	0.25	257	1	3	1	2	2	1	1	1	.
6	8	1.49630	14	2	0.25	258	2	2	2	1	3	1	1	1	.
7	8	1.49630	21	2	0.25	259	3	1	3	3	1	1	1	1	.
8	8	1.49630	28	2	0.25	260	1	1	1	1	1	.	.	.	1

9	8	1.49630	27	3	0.25	261	3	3	3	1	2	1	1	1	.
10	8	1.49630	2	3	0.25	262	1	1	2	3	3	1	1	1	.
11	8	1.49630	13	3	0.25	263	2	2	1	2	1	1	1	1	.
12	8	1.49630	28	3	0.25	264	1	1	1	1	1	.	.	.	1
13	8	1.49630	6	4	0.25	265	1	2	3	1	1	1	1	1	.
14	8	1.49630	17	4	0.25	266	2	3	2	3	2	1	1	1	.
15	8	1.49630	19	4	0.25	267	3	1	1	2	3	1	1	1	.
16	8	1.49630	28	4	0.25	268	1	1	1	1	1	.	.	.	1
17	8	1.49630	11	5	0.25	269	2	1	2	2	1	1	1	1	.
18	8	1.49630	9	5	0.25	270	1	3	3	3	3	1	1	1	.
19	8	1.49630	22	5	0.25	271	3	2	1	1	2	1	1	1	.
20	8	1.49630	28	5	0.25	272	1	1	1	1	1	.	.	.	1
21	8	1.49630	5	6	0.25	273	1	2	2	2	2	1	1	1	.
22	8	1.49630	12	6	0.25	274	2	1	3	1	3	1	1	1	.
23	8	1.49630	25	6	0.25	275	3	3	1	3	1	1	1	1	.
24	8	1.49630	28	6	0.25	276	1	1	1	1	1	.	.	.	1
25	8	1.49630	23	7	0.25	277	3	2	2	3	1	1	1	1	.
26	8	1.49630	16	7	0.25	278	2	3	1	1	3	1	1	1	.
27	8	1.49630	3	7	0.25	279	1	1	3	2	2	1	1	1	.
28	8	1.49630	28	7	0.25	280	1	1	1	1	1	.	.	.	1
29	8	1.49630	4	8	0.25	281	1	2	1	3	3	1	1	1	.
30	8	1.49630	18	8	0.25	282	2	3	3	2	1	1	1	1	.
31	8	1.49630	20	8	0.25	283	3	1	2	1	2	1	1	1	.
32	8	1.49630	28	8	0.25	284	1	1	1	1	1	.	.	.	1
33	8	1.49630	24	9	0.25	285	3	2	3	2	3	1	1	1	.
34	8	1.49630	8	9	0.25	286	1	3	2	1	1	1	1	1	.
35	8	1.49630	10	9	0.25	287	2	1	1	3	2	1	1	1	.
36	8	1.49630	28	9	0.25	288	1	1	1	1	1	.	.	.	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. Note that missing flags are treated the same as zero. Also notice that some candidate alternatives appear in the design more than once and some do not appear at all.

Generic Attributes, a Constant Alternative, and Choice Set Swapping

The %CHOICEFF macro can be used in a very different way. Instead of providing a candidate set of alternatives to swap in and out of the design, you can provide a candidate set of entire choice sets. For this particular example, swapping alternatives will almost certainly be better. 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 %MKTDES 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). Then we will process the candidates from one row per choice set to one row per alternative per choice set using the %MKTROLL macro. 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.

```
%mktdes(factors=x1-x15=3, run=factex, size=81 * 81)

data key;
   input (x1-x5) ($);
   datalines;
 x1  x2  x3  x4  x5
 x6  x7  x8  x9 x10
x11 x12 x13 x14 x15
  .   .   .   .   .
;

%mktroll(design=cand1, key=key, out=rolled)

* Code the constant alternative;
data cand2;
   set rolled;
   if _alt_ = 4 then do; x1 = 1; x2 = 1; x3 = 1; x4 = 1; end;
   run;

%choiceff(data=cand2, model=class(x1-x5), nsets=9, nalts=4,
          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	0.8366132274	1.1952954689
	1	1.3243418568	0.7550920443
	2	1.3507622396	0.7403227383
	3	1.3683353572	0.730814997
	4	1.3683353572	0.730814997

Design	Iteration	D-Efficiency	D-Error
2	0	0.905865211	1.1039169932
	1	1.3344815671	0.7493546743
	2	1.3793159247	0.7249970671
	3	1.3793159247	0.7249970671

Generic Chair Attributes

```
Final Results:   Design     = 2
                 Efficiency = 1.3793159247
                 D-Error    = 0.7249970671
```

Generic Chair Attributes

n	Variable Name	Label	Variance	DF	Standard Error
1	x11	x1 1	0.80431	1	0.89683
2	x12	x1 2	1.06350	1	1.03126
3	x21	x2 1	0.73761	1	0.85884
4	x22	x2 2	0.96773	1	0.98373
5	x31	x3 1	0.85405	1	0.92415
6	x32	x3 2	0.93511	1	0.96701
7	x41	x4 1	0.97698	1	0.98842
8	x42	x4 2	0.95917	1	0.97937
9	x51	x5 1	0.85520	1	0.92477
10	x52	x5 2	0.99226	1	0.99612
				==	
				10	

Data collection, processing, and analysis is basically the same as before, so we will not go through another example of it. The only difference is that our design data set is already arrayed with one row per alternative, so we will not have to put it in that form during the processing step. However, note that 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, using this design in a branded study and specifying alternative-specific effects like this could lead to a lot of unestimable parameters.

```
* Bad idea for this design!!;
model class(brand)
      class(brand * x1 brand * x2 brand * x3 brand * x4 brand * x5);
```

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 %MKTDES macro and FACTEX/PLAN and OPTEX, 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 nine 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. So the alternative swapping algorithm generally has more freedom to find a good design.

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 %MKTDES 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.

Other Design Strategies

This section illustrates some design strategies that are not in other sections. Included are examples for factors with many levels, when quantitative factors have extra levels, designs with many factors, improving an existing design, adding random choice sets to the candidate set, creating candidate sets by permuting columns, and efficiently augmenting fixed choice sets. We will not actually use any designs from this section.

Very Big Designs

A researcher needs to design a choice experiment with 16 alternatives. Each alternative is composed of 1 eight-level factor, 3 four-level factors, and 15 two-level factors. So the full design is $2^{240}4^{48}8^{16}$ (240 two-level factors, 48 four-level factors, and 16 eight-level factors). Since 2, 4, and 16 are multiples of 2, we can make a design from a design with all two-level factors as follows. This code creates each eight-level factor from the main effects and all interactions of 3 two-level factors and each four-level factor from the main effects and all interactions of 2 two-level factors.

```
%mktdes(factors=x1-x240=2 y1-y48=4 z1-z16=8, run=factex)
```

Unfortunately, on many computers, this step will not run due to insufficient memory.

The fact that we could not directly create a design for this problem means we need to take a less direct approach. We were able to get a design in less than five minutes as follows. First, we use the %MKTDES macro to generate a design for just the four and eight-level factors. In addition, we create nine factors **a b c d e f g h i**, by specifying **otherfac=a|b|c|d|e|f|g|h|i**, the main effects and interactions of 9 two-level variables. By specifying these terms in the **otherfac=** option, we do not ensure that they are estimable. They only get specified in the PROC FACTEX **factors** statement and not in the **estimate=** specification.

```
%let indexes  = a b c d e f g h i;
%let nindexes = 9;
%let indbar   = a|b|c|d|e|f|g|h|i;

%mktdes(otherfac=&indexes, factors=f1-f48=4 e1-e16=8,
        nlev=2, size=512, run=factex)
```

Here are some of the lines of code that the macro generated.

```
proc factex;
   factors a b c d e f g h i _1-_144 / nlev=2;
   size design=512;
   model estimate=(
      _1|_2
      .
      .
      .
      _95|_96
      _97|_98|_99
      .
      .
      .
      _142|_143|_144
      );
```

```
output out=Cand1(drop=_:)
   [_1 _2]=f1 nvals=(1 to 4)
   .
   .
   .
   [_95 _96]=f48 nvals=(1 to 4)
   [_97 _98 _99]=e1 nvals=(1 to 8)
   .
   .
   .
   [_142 _143 _144]=e16 nvals=(1 to 8)
   ;
run; quit;
```

Next, we use a DATA step to add a variable **y** (not of interest) so PROC GLM can tell us which effects in **a|b|c|d|e|f|g|h|i** are estimable, given the four and eight-level factors of interest. The results were stored in the **outstat=** data set RES.

```
data temp; set cand1; y = 1; run;

proc glm data=temp outstat=results noprint;
   class f1-f48 e1-e16;
   model y = f1-f48 e1-e16 &indbar / ss1;
   run;
```

Here are some selected observations in the **outstat=** data set.

Obs	_NAME_	_SOURCE_	_TYPE_	DF	SS	F	PROB
1	y	ERROR	ERROR	0	0	.	.
2	y	f1	SS1	3	0	.	.
49	y	f48	SS1	3	0	.	.
50	y	e1	SS1	7	0	.	.
65	y	e16	SS1	7	0	.	.
66	y	a	SS1	0	0	.	.
67	y	b	SS1	0	0	.	.
68	y	a*b	SS1	0	0	.	.
69	y	c	SS1	0	0	.	.
129	y	g	SS1	1	0	.	.
130	y	a*g	SS1	1	0	.	.
145	y	e*g	SS1	1	0	.	.
149	y	c*e*g	SS1	1	0	.	.
423	y	b*c*f*g*i	SS1	0	0	.	.
424	y	a*b*c*f*g*i	SS1	1	0	.	.
576	y	a*b*c*d*e*f*g*h*i	SS1	0	0	.	.

The four-level factors have 3 *df*, the eight-level factors have 7 *df*, many of the effects in **a|b|c|d|e|f|g|h|i** have 0 *df*, and the remaining effects in **a|b|c|d|e|f|g|h|i** have 1 *df*, which means they are available for creating two-level factors. The next step outputs just those effects with 1 *df* and counts them, storing the result in macro variable **&n**.

```
data res2(keep=_source_);
   set results;
   if df = 1;
   n + 1;
   call symput('n',put(n,3.));
   run;
```

Then we use each of these estimable 1 *df* terms to create the two-level factors. This next DATA step creates 255 two-level factors from the estimable effects in **a|b|c|d|e|f|g|h|i**. For example, when **_SOURCE_ = 'a*g'**, a two-level factor is created from the **a*g** interaction.

```
data candid(drop=&indexes termi);
   set temp;
   array t[&n];
   array terms[&nindexes] &indexes;
   do ind = 1 to &n;
      set res2 point=ind;
      t[ind] = 1;
      do termi = 1 to &nindexes;
         if index(_source_, substr(compress("&indexes"),termi,1))
            then t[ind] = t[ind] * terms[termi];
         end;
      end;
   run;
```

This next step checks the results.

```
proc glm;
   class f1-f48 e1-e16;
   model y = f1-f48 e1-e16 t1-t&n / ss2;
   run;
```

Here are some selected results. If everything is right (and the full listing shows that it is), then all four-level factors have 3 *df*, all eight-level factors have 7 *df*, and all two-level factors have 1 *df*.

Source	DF	Type II SS	Mean Square	F Value	Pr > F
f1	3	0	0	.	.
f48	3	0	0	.	.
e1	7	0	0	.	.
e16	7	0	0	.	.
t1	1	0	0	.	.
t255	1	0	0	.	.

This design is saturated; there are 512 runs and 511 *df* plus the intercept.

Improving an Existing Design

Another useful technique is trying to improve an existing design. In this case, we use the %MKTDES macro to create a design in 80 runs for 25 four-level factors using a 2048 run candidate set.

```
title '25 Factors, Try to Improve an Existing Design';

%mktdes(factors=x1-x25=4, nlev=2, size=2048, n=80, procopts=seed=7654321)
```

25 Factors, Try to Improve an Existing Design

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	82.9917	63.4906	59.2462	1.2232
2	82.2876	61.7434	55.3920	1.2404
3	82.1939	61.4429	55.5984	1.2434
4	82.1919	61.2833	53.8251	1.2451
5	82.1599	60.8513	53.8240	1.2495

Next, we create more potential candidate sets CAND2 through CAND5 of size **min** (128), 256, 512, and 1024. The question we ask is can we improve our design, substituting points from all of our candidate sets?

```
%mktdes(factors=x1-x25=4, nlev=2, size=min,  run=factex, cand=cand2)
%mktdes(factors=x1-x25=4, nlev=2, size=256,  run=factex, cand=cand3)
%mktdes(factors=x1-x25=4, nlev=2, size=512,  run=factex, cand=cand4)
%mktdes(factors=x1-x25=4, nlev=2, size=1024, run=factex, cand=cand5)
```

We can combine the candidate sets, eliminating any duplicates, by sorting them and then merging them on all of the variables.

```
proc sort data=cand1; by x1-x25; run;
proc sort data=cand2; by x1-x25; run;
proc sort data=cand3; by x1-x25; run;
proc sort data=cand4; by x1-x25; run;
proc sort data=cand5; by x1-x25; run;

data cand;
   merge cand1 cand2 cand3 cand4 cand5;
   by x1-x25;
   run;
```

Our new candidate set has 3946 runs. Then we take DESIGN, created from the 2048-run resolution III candidate set and try to improve it using candidate set CAND. We explicitly run PROC OPTEX, specifying **initdesign=design method=m_federov** in the **generate** statement to do this. The %MKTDES macro does not subsume all of the considerable functionality of the FACTEX and OPTEX procedures. Sometimes, for more esoteric problems, we have to run those procedures directly.

```
proc optex seed=123 data=cand;
   title '25 Factors, Try to Improve an Existing Design';
   class x1-x25 / param=orthref;
   model x1-x25;
   generate n=80 initdesign=design method=m_federov;
   output out=des2;
   run; quit;
```

This step produced the following results.

25 Factors, Try to Improve an Existing Design

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	82.9917	63.4906	59.2462	1.2218

Normally, by default there are ten iterations from ten random starts. With **initdesign=** there is only one iteration because there are no random starting points. For this particular problem, we did not do better than we have done previously.

When Some Choice Sets are Fixed in Advance

Another useful technique is creating an efficient design when certain choice sets are fixed in advance and must be included in the design. Stated differently, PROC OPTEX can be used to efficiently augment a small starting design with other choice sets. We will illustrate this in the context of an unrealistic but pedagogical example. In this case we want an efficient design with 9 two-level factors and 12 three-level factors in 36 runs. The best way to create this design is simply to select the relevant columns from the L_{36} tabled design. Instead, we will create a candidate set that contains this design and other choice sets and see if PROC OPTEX can find the L_{36}. We will use the %MKTDES macro to create a candidate set. Then we merge this design with the L_{36}.

```
%let sets     = 36;           /* Number of choice sets               */
%let alts     = 6;            /* Number of alternatives              */
%let factors  = b1-b12 a1-a9; /* Linear Design Factors               */

data l36;   /* a1-a11 - two-level, b1-b12 - three-level */
   input (a1-a11 b1-b12) (23*1.) @@;
   if mod(_n_, 3) = 0 then input;
   datalines;
11111111112223322321121111111111133311331322311111111111111221121331
21211122212222212121223212111222123333232323312121112222121111313113112
22121112221223221312232222121112221331332123313221211122211121132311121
12212111222221123223322122121112223322313311331221211122211331211211
21222121112223112233121112212211122312233112322212212111221233112233133
22122121112231313111132221221211123121212222132212212111212323233332133
22122121111232211213311222122121111313322321122222122121111211331323233
12221221211233131122121122212212111311212233232122212212111223233113133
11222122121213132211333112221221321213322111112222122121132321133322211
11122212212213223133113111222122123213312112211112222122121321132233233211
21112221221211311332323211122212213221221131312111222122113323322121211
12111222122212333323231121112222122323111131312121112222122131222212312
;

* Create candidate set for this situation;

%mktdes(factors=a1-a9=2 , step=1, run=factex)
%mktdes(factors=b1-b12=3, step=2, run=factex)

data both;
   set l36 cand2;
   run;


title 'Use MKTDES to Generate Design';

%mktdes(factors=a1-a9=2 b1-b12=3, procopts=seed=51000,
        iter=500, n=&sets, cand=both, run=optex)

proc optex data=both seed=7654321;
   title 'Evaluate L_36 Design';
   class &factors / param=orthref;
   model &factors;
   generate n=&sets method=sequential initdesign=l36;
   quit;
```

These steps produced the following results.

Use MKTDES to Generate Design

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	100.0000	100.0000	100.0000	0.9718
2	86.4336	74.4076	66.9333	1.1266
3	85.9883	71.9545	65.9474	1.1457
4	85.9390	72.6370	66.5752	1.1403
5	85.9325	72.4818	67.4581	1.1415

Evaluate L_36 Design

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	100.0000	100.0000	100.0000	0.9718

In 500 iterations, PROC OPTEX found the L_{36} only once. The second step shows the L_{36} is 100% efficient. It is not surprising that the optimal design is hard to find. There are over 9×10^{53} designs with 36 runs that can be selected from a 468-run candidate set. They could not all be evaluated in the entire history of the universe even by a trillion computers each evaluating one trillion designs every nanosecond.

Next, we will see what happens if we give PROC OPTEX an initial design with fixed choice sets that must appear in the final design. We sample four choice sets from the L_{36}, store them in the data set INIT, and specify **augment=** in the **generate** statement.

```
data init;
   retain k 4; /* Randomly sample EXACTLY 4 points. */
   set l36;
   if uniform(111) < k / (37 - _n_) then do;
      output;
      k + -1;
      end;
   if k = 0 then stop;
   run;

proc optex data=both seed=72555;
   title "Use OPTEX to Augment 4 Existing Design Points";
   class &factors / param=orthref;
   model &factors;
   generate n=36 method=m_federov augment=init;
   quit;
```

These steps produced the following results.

Use OPTEX to Augment 4 Existing Design Points

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	100.0000	100.0000	100.0000	0.9718
2	100.0000	100.0000	100.0000	0.9718
3	100.0000	100.0000	100.0000	0.9718
4	100.0000	100.0000	100.0000	0.9718
5	84.7174	69.2709	64.6258	1.1677
6	84.5309	69.2777	63.4149	1.1676
7	84.2451	69.3293	66.9899	1.1672
8	84.0934	67.2020	62.6912	1.1855
9	83.6454	67.7776	65.5860	1.1804
10	83.2917	66.2490	64.3089	1.1940

With just four randomly chosen points from the L_{36} fixed, PROC OPTEX achieves 100% efficiency in four out of ten tries.

Six-Level Factors

A researcher needs to create a design with 10 six-level factors. Fractional-factorial designs for six-level factors cannot be created directly by design algorithms like PROC FACTEX uses. These algorithms work with levels that are prime or a power of a prime.

2	–	prime,	can be created directly
3	–	prime,	can be created directly
4	–	prime squared,	can be created directly
5	–	prime,	can be created directly
6	–	not prime,	cannot be created directly
7	–	prime,	can be created directly
8	–	prime cubed,	can be created directly
9	–	prime squared,	can be created directly
10	–	not prime,	cannot be created directly
11	–	prime,	can be created directly
12	–	not prime,	cannot be created directly
13	–	prime,	can be created directly
			and so on

Here is the most obvious approach.

```
title 'Six-Level Factors';

%mktdes(factors=x1-x10=6, n=60, procopts=seed=7654321, size=1024)
```

This approach makes six-level factors from 3 two-level pseudo-factors. For example, **x1** is created from **_1|_2|_3** with the mapping **[_1 _2 _3]=x1 nvals=(1 to 6 1 6)**. The problem with this approach is eight levels are mapped to six, so the candidate set is imbalanced and it is likely the design will be imbalanced. Most factors will have more ones and sixes than twos through fives.

Six-Level Factors

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	82.7201	64.9653	60.6132	1.1047
2	82.6667	65.5906	63.0546	1.1089
3	82.6580	64.7216	59.4145	1.1129
4	82.4430	65.0258	63.0317	1.1052
5	82.2820	64.2499	61.3416	1.1180

The %MKTDES macro provides a special way to handle six-level factors. You can use a two-step process to create a design with two-level and three-level factors. Then you can use the %MKTDES6 macro to combine them into six-level factors. The %MKTDES6 macro is provided with the %MKTDES macro. Use it in a DATA step to combine the **two=** list and the **three=** list, storing the results in the **two=** variables. Then you can use the %MKTDES macro to search for an efficient design.

```
%mktdes(factors=x1-x10=2, step=1, n=60, run=factex, size=32)
%mktdes(factors=b1-b10=3, step=2, n=60, run=factex, size=27)

data cand;
   set cand2;
   *---Create 6-levels from 2-levels and 3-levels---;
   %mktdes6(two=x1-x10, three=b1-b10);
   run;

%mktdes(factors=x1-x10=6, n=60, run=optex, cand=cand, procopts=seed=7654321)
```

Six-Level Factors

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	82.4576	62.2158	60.2928	1.1689
2	81.3020	62.1989	61.4858	1.1690
3	81.0982	61.5894	58.0499	1.1748
4	80.8932	60.3483	56.4664	1.1868
5	80.6477	60.1977	57.6953	1.1883

As you can see from the output, there is no guarantee that using the %MKTDES6 macro will produce a more efficient design, but it should produce a better balanced design.

You could also base the design on only three-level factors by specifying **nlev=3**. This approach makes six-level factors from 2 three-level pseudo-factors. For example, **x1** is created from **_1**|**_2** with the mapping **[_1 _2]=x1 nvals=(1 to 6 1 4 6)**.

```
title 'Six-Level Factors';

%mktdes(factors=x1-x10=6, n=60, nlev=3, procopts=seed=7654321)
```

Six-Level Factors

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	84.6808	67.4270	57.6530	1.0728
2	84.6808	67.4270	57.6530	1.0728
3	84.6808	67.4270	57.6530	1.0728
4	84.6808	67.4270	57.6530	1.0728
5	84.4295	66.5160	55.3460	1.0813

Similarly, you could base the design on five-level factors by specifying **nlev=5**. This approach makes six-level factors from 2 five-level factors. For example, **x1** is created from **_1|_2** with the mapping **[_1 _2]=x1 nvals=(1 to 6 1 to 6 1 to 6 1 to 6 1)**. This should make a much better balanced candidate set than we got using two-level or three-level factors as the base.

```
title 'Six-Level Factors';

%mktdes(factors=x1-x10=6, n=60, nlev=5, procopts=seed=7654321)
```

Six-Level Factors

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	78.4326	58.2505	59.2676	1.2059
2	77.6983	56.7929	58.3282	1.2217
3	77.4057	55.6193	55.9998	1.2343
4	77.2037	55.1903	55.8364	1.2400
5	77.1410	55.5539	55.9045	1.2337

You could base the design on two-level pseudo-factors, but more than the default three per factor. Recall that we can make a six-level factor **x1** from **_1|_2|_3** and **[_1 _2 _3]=x1 nvals=(1 to 6 1 6)**. To get better balance, we could specify **(4)** in **factors=x1-x10=6(4)** to use 4 two-level pseudo-factors for each six-level factor: **_1|_2|_3|_4** with the mapping **[_1 _2 _3 _4]=x1 nvals=(1 to 6 1 to 6 1 3 4 6)**.

```
title 'Six-Level Factors';

%mktdes(factors=x1-x10=6(4), n=60, procopts=seed=7654321, size=1024)
```

Six-Level Factors

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	78.5916	59.1543	60.4969	1.1912
2	78.4422	58.3053	59.3152	1.2020
3	78.1623	57.6540	59.5914	1.2096
4	78.0190	57.6495	56.7703	1.2115
5	77.9914	58.4882	60.7042	1.1992

Of course all of these examples could be tried with different candidate set sizes. It is difficult to predict in advance which approach will work best for any particular problem. However, like the results of this example, we have frequently seen that using more than the minimum number of pseudo-factors tends to increase balance at a cost of decreased efficiency.

Ten-Level Factors

A researcher needs to create a design with 5 ten-level factors. Like six-level factors (discussed starting on page 256), ten-level factors cannot be created directly. Here is the most obvious approach.

```
title 'Ten-Level Factors';

%mktdes(factors=x1-x4=10, n=50, procopts=seed=7654321, size=1024)
```

This approach makes ten-level factors from 4 two-level pseudo-factors. For example, **x1** is created from **_1|_2|_3|_4** with the mapping **[_1 _2 _3 _4]=x1 nvals=(1 to 10 1 3 5 6 8 10)**. The problem with this approach is 16 levels are mapped to ten, so the candidate set is imbalanced and it is likely the design will be imbalanced. Most factors will have more 1's, 3's, 4's, 6's, 8's, and 10's than 2's, 5's, 7's, and 9's.

Ten-Level Factors

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	83.8294	69.3338	67.8612	1.0288
2	83.5641	68.9284	66.4021	1.0269
3	83.2275	68.1464	64.1257	1.0332
4	83.1952	68.1558	65.9666	1.0419
5	82.9385	67.2304	64.1725	1.0406

The %MKTDES macro provides a special way to handle ten-level factors. You can use a two-step process to create a design with two-level and five-level factors. Then you can use the %MKTDES10 macro to combine them into ten-level factors. The %MKTDES10 macro is provided with the %MKTDES macro. Use it in a DATA step to combine the **two=** list and the **five=** list, storing the results in the **two=** variables. Then you can use the %MKTDES macro to search for an efficient design.

```
%mktdes(factors=x1-x5=2, step=1, n=50, run=factex, size=16)
%mktdes(factors=b1-b5=5, step=2, n=50, run=factex, size=125)
```

```
data cand;
   set cand2;
   *---Create 10-levels from 2-levels and 5-levels---;
   %mktdes10(two=x1-x5, five=b1-b5);
   run;
%mktdes(factors=x1-x5=10, n=50, run=optex, cand=cand, procopts=seed=7654321)
```

Ten-Level Factors

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	74.8865	51.6941	51.6550	1.3341
2	74.5708	52.1228	50.7978	1.3286
3	74.3059	51.0803	50.5599	1.3420
4	74.3042	50.9776	50.4575	1.3434
5	74.1494	50.1395	51.4421	1.3546

As you can see from the output, there is no guarantee that using the %MKTDES10 macro will produce a more efficient design, but it should produce a better balanced design. Of course the techniques illustrated for six-level factors (starting on page 256) could also be used for ten-level factors.

The Macros

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

Macro	Page	Release	Purpose
%MKTDES	261	8.0	efficient linear experimental design
%MKTDES6	266	8.0	making six-level factors
%MKTDES10	267	8.0	making ten-level factors
%MKTRUNS	268	8.0	experimental design size
%CHOICEFF	270	8.0	efficient choice design
%PHCHOICE	288	8.0	customizing the printed output from a choice model
%MKTROLL	281	8.1	rolling a linear design into a choice design
%MKTMERGE	285	8.1	merging a choice design with choice data
%MKTALLO	286	8.1	processing allocation data

The "Release" column indicates the first release of the SAS System in which the macros are distributed. If you are using version 8.0, go to http://www.sas.com/techsup/download/stat/ and get mlogit8.sas, or via anonymous ftp, from ftp.sas.com, get techsup/download/stat/mlogit8.sas. These files contain all of the code used in this report. You can also write saswfk@wnt.sas.com to request the macros. The Version 8.1 macros can be used in Version 8.0; they were just finished too late to be shipped with Version 8.0. All of the macros have been updated at least a little since Version 8.0 was shipped, so if you have not yet installed Version 8.1 or a later release, you should get the latest versions of all of the macros.

If your site has installed the autocall libraries supplied by SAS Institute and uses the standard configuration of SAS software supplied by the Institute, you need only to ensure that the SAS system option **`mautosource`** is in effect to begin using the autocall macros. That is, the macros do *not* have to be included (for example with a **`%include`** statement). They can be called directly. For more information about autocall libraries, refer to *SAS Macro Language: Reference, First Edition*, 1997. On a PC for example, the autocall library may be installed in the **`stat\sasmacro`** directories. On MVS, each macro will be a different member of a PDS. For details on installing autocall macros, consult your host documentation.

%MKTDES Macro Overview

Throughout this report, we used the %MKTDES autocall macro to design our experiments. At the heart of the %MKTDES macro are PROC PLAN, PROC FACTEX, and PROC OPTEX. We often use the macro instead of calling these procedures directly because the macro has a simpler syntax designed for some of the complicated problems marketing researchers face. 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 is small (by default less than 2189 runs) the experimental design problem is straightforward. First, the macro uses PROC PLAN to create a full-factorial candidate set. Then PROC OPTEX searches the full-factorial candidate set and will almost certainly find the optimal design or a very nearly optimal design

given sufficient iteration (for example, specify **iter=100**). Run time will typically be a few seconds or a few minutes, but it could run up to an hour.

When the full-factorial design is larger, the design problem is less straight-forward and requires more thought. Then the macro uses PROC FACTEX to create a fractional-factorial candidate set. The default **size=**, **nlev=**, number of pseudo-factors, and so on will often give a good design. However, changing these values may increase or decrease efficiency, 1, 2, 5, 10, perhaps up to 15%. This is the art of design – finding the options that make the best design. Making a bigger candidate set may help, or it may not. Changing **nlev=** may make the design better; it may make it worse. You have to try several things and see which one is best. Be careful how far you generalize from your experiences. For example, for one problem, doubling the size of the candidate set may increase efficiency a few percent. For the next problem it may *decrease* efficiency.

Start out small. In your first attempt to create a design for a problem, your candidate set should have no more than a few-hundred runs, and run time should be well under a minute. Then you can try bigger candidate sets and larger searches. Keep the size of your candidate set small, usually under 2000 - 5000. See page 108 for a discussion of candidate set size. Here are some typical examples of usage:

```
*---Six three-level factors in 18 runs---;
%mktdes(factors=x1-x6=3, n=18)

*---Two two-level factors and 3 three-level factors in 18 runs---;
%mktdes(factors=x1-x2=2 x3-x5=3, n=18)


*---Mix in levels---;
%mktdes(factors=x2=2 x3=3 x4=4 x5=5 x6=6 x7=7 x8=8,
        size=512, n=32);

*---Mixed 2-, 3-, 5-level factors, done in one step------;
%mktdes(factors=x1-x3=2 x4-x6=3 x7-x9=5, n=30)

*---Mixed 2-, 3-, 5-level factors, done in three steps------;
%mktdes(factors=x1-x3=2, n=30, run=factex,       step=1)
%mktdes(factors=x4-x6=3, n=30, run=factex,       step=2)
%mktdes(factors=x7-x9=5, n=30, run=factex optex, step=3)

*---Five ten-level factors, done in one step---;
%mktdes(factors=x1-x5=10, n=50)

*---Five ten-level factors, done in one step, based on seven-levels---;
%mktdes(factors=x1-x5=10, n=50, nlev=7)

*---Fifteen three-level factors, different candidate set sizes---;
%mktdes(factors=x1-x15=3, n=36,             out=des1, cand=cand1)
%mktdes(factors=x1-x15=3, n=36, size=81*3,  out=des2, cand=cand2)
%mktdes(factors=x1-x15=3, n=36, size=81*9,  out=des3, cand=cand3)
%mktdes(factors=x1-x15=3, n=36, size=81*27, out=des4, cand=cand4)

*---A design with interactions---;
%mktdes(factors=x1-x3=2 x4-x6=3(3) x7-x9=4, n=32, size=1024,
        interact=x1|x2|x3@2 x4*x7)


*---Look at one-way and two-way frequencies---;
proc summary print;
   ways 1 2;
   class x:;
   run;
```

%MKTDES Macro Options

Here are the options you can use 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 is less than 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 a **big=5000** is probably reasonable. However, run time can be very slow with sizes much bigger than the default.

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.

coding= *name*

specifies the PROC OPTEX **coding=** option. If you are using Version 7 or an earlier release of the SAS System, and if you have a balanced and orthogonal candidate set, you may want to specify **coding=orthcan**. With Version 8 of the SAS System, this option is usually not needed.

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 may 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 used to create factors of interest but are not themselves output. So 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))
```

Then 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. Many researchers are willing to sacrifice a little bit of efficiency in order to achieve better balance.

generate= *options*
specifies PROC OPTEX **generate** statement options. By default, additional options are not added to the **generate** statement.

interact= *terms*
specifies interactions that must be estimable. By default interactions are not guaranteed to be estimable. Examples:

```
interact=x1*x2
interact=x1*x2 x3*x4*x5
interact=x1-x5@2
```

Only "@" values of 2 or 3 are allowed. By default, no interactions are guaranteed to be estimable.

iter= 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 PROC OPTEX created design. The default is the PROC OPTEX default and depends on the problem. Typically, you will not want to use the default. Instead pick a value that is divisible by all or most of the numbers of levels and their products. The **n=saturated** option creates a design with the minimum number of runs.

options= *option-list*
specifies general boolean options (currently there is only one):

options=eval evaluates the final efficiency using orthogonal coding. This is the default.

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. The most common usage is **procopts=seed=***n* where *n* is a random seed. 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 omit steps – if for example you only want the code listing – 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 and the value of **big=**. When PLAN is not in the list, the macro generates code for PROC FACTEX.

size= n|**MIN**

specifies the candidate set size. Start with the default **size=min** and see how big that 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 **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. 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)
```

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 stores globally two macro variables **&&class&step** and **&&inter&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 the example above, 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 candidate design, used to restrict the candidates. By default, the candidate design is not restricted.

%MKTDES6 Macro Overview

Also included with the %MKTDES autocall macro is the %MKTDES6 macro. It is used along with %MKTDES to make six-level factors. Since six is not a power of a prime number, six-level factors cannot be created by PROC FACTEX except by coding down or using pseudo-factors. Alternatively, you could create two-level factors in one step and then three-level factors in the next step. Then they could be combined, 2×3, to make six-level factors. You must run the %MKTDES autocall macro before running the %MKTDES6 macro since the %MKTDES6 macro is only loaded when %MKTDES is called. If you do not need to use %MKTDES first, run it with no arguments.

```
%mktdes;
```

The macro will print the message "**ERROR: FACTORS= must be specified.**" and quit, but it will load all three macros first.

%MKTDES6 is used in a DATA step to combine the two-level and three-level factors returning the six-level factors in variables with the same names as the original two-level factors. Then the %MKTDES macro is run on the results to search the candidate set for an efficient design.

```
*---Five six-level factors, done in multiple steps, post-processing---;
%mktdes(factors=x1-x5=2,  n=30, run=factex, size=16, step=1)
%mktdes(factors=x6-x10=3, n=30, run=factex, size=81, step=2)

data cand;
   set cand2;
   *---Create 6-levels from 2-levels and 3-levels---;
   %mktdes6(two=x1-x5, three=x6-x10);
   run;
```

```
%mktdes(factors=x1-x5=6, n=30, run=optex, cand=cand)
```

%MKTDES6 Macro Options

The %MKTDES6 macro has two options.

two= *variable-list*
names the two-level factors.

three= *variable-list*
names the three-level factors.

%MKTDES10 Macro Overview

Also included with the %MKTDES autocall macro is the %MKTDES10 macro. It is used along with %MKTDES to make ten-level factors. Since ten is not a power of a prime number, ten-level factors cannot be created by PROC FACTEX except by coding down or using pseudo-factors. Alternatively, you could create two-level factors in one step and then five-level factors in the next step. Then they could be combined, 2×5, to make ten-level factors. You must run the %MKTDES autocall macro before running the %MKTDES10 macro since the %MKTDES10 macro is only loaded when %MKTDES is called. If you do not need to use %MKTDES first, run it with no arguments.

```
%mktdes;
```

The macro will print the message "**ERROR: FACTORS= must be specified.**" and quit, but it will load all three macros first.

%MKTDES10 is used in a DATA step to combine the two-level and five-level factors returning the ten-level factors in variables with the same names as the original two-level factors. Then the %MKTDES macro is run on the results to search the candidate set for an efficient design.

```
*---Five ten-level factors, done in multiple steps, post-processing---;
%mktdes(factors=x1-x5=2,  n=50, run=factex, step=1)
%mktdes(factors=x6-x10=5, n=50, run=factex, step=2)

data cand;
   set cand2;
   *---Create 10-levels from 2-levels and 5-levels---;
   %mktdes10(two=x1-x5, five=x6-x10);
   run;

%mktdes(factors=x1-x5=10, n=50, run=optex, cand=cand)
```

%MKTDES10 Macro Options

The %MKTDES10 macro has two options:

two= *variable-list*
names the two-level factors.

five= *variable-list*
names the five-level factors.

%MKTRUNS Macro Overview

The %MKTRUNS autocall macro calculates 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. No error checking is performed.

For example, with 3 two-level and 4 three-level factors, specify the macro with three 2's and four 3's, the numbers of levels for all of the factors.

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

The output of the macro in this example is:

Some Reasonable Design Sizes (Saturated=12)	Violations	Cannot Be Divided By
36	0	
72	0	
108	0	
144	0	
180	0	
18	3	4
54	3	4
90	3	4
126	3	4
162	3	4

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. The macro also reports larger sizes. 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:

Some Reasonable Design Sizes (Saturated=21)	Violations	Cannot Be Divided By
120	3	9 16 25
180	6	8 16 25
60	7	9 8 16 25
144	15	5 10 15 20 25
48	16	9 5 10 15 20 25
72	16	16 5 10 15 20 25
80	16	3 6 9 12 15 25
96	16	9 5 10 15 20 25
160	16	3 6 9 12 15 25
192	16	9 5 10 15 20 25

Among the smaller design sizes, 60 or 48 look like a good possibilities. The macro has a second, optional, keyword parameter: **max=**. It specifies the maximum number of sizes to try. Usually you will not need to specify the **max=** option. The smallest design considered is the saturated design. For example, this specification tries 5000 sizes and reports that a perfect design can be found with 3600 runs.

```
%mktruns(2 2 3 3 4 4 5 5, max=5000)
```

Some Reasonable Design Sizes (Saturated=21)	Violations	Cannot Be Divided By
3600	0	
720	1	25
1200	1	9
1440	1	25
1800	1	16
2160	1	25
2400	1	9
2880	1	25
4320	1	25
4800	1	9

Now consider again the problem with 3 two-level and 4 three-level factors, but this time we want to be estimable the interaction of two of the two-level factors. So 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 )
```

Some Reasonable Design Sizes (Saturated=13)	Violations	Cannot Be Divided By
72	0	
144	0	
36	1	8
108	1	8
180	1	8
18	6	4 8 12
24	6	9
48	6	9
54	6	4 8 12
90	6	4 8 12

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).

%MKTRUNS Macro Options

The %MKTDES macro has one positional parameter **list**, that must be specified first. For positional parameters, just a value is specified (unlike keyword parameters which have the form KEY-WORD=**value**). The macro also has a keyword parameter **max=**.

list

specifies the numbers of levels of all of the factors. This parameter must be specified first. For example, with 4 two-level factors, specify a **list** of **2 2 2 2**:

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

max= n

specifies the maximum number of design sizes to try. By default, **max=200**. The macro tries **max=***n* sizes starting with the saturated design and reports the best 10 sizes.

%CHOICEFF Macro Overview

The %CHOICEFF autocall macro is used to find efficient experimental designs for choice experiments. You supply a candidate set of either alternatives or sets of alternatives. The macro searches the candidates for an efficient experimental design – a design in which the variances of the parameter estimates are minimized, given and assumed $\boldsymbol{\beta}$.

There are two primary ways to use the macro:

- You can create a candidate set of alternatives. Then the macro creates 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 (say) four brands, you must create four lists of candidate alternatives, one for each brand.
- You can create a candidate set of choice sets. Then the macro builds a design from the choice sets you supplied.

Typically, you use as a candidate set either a full-factorial or fractional-factorial design, created with the %MKTDES macro. The macro either constructs a random initial design from the candidates or it uses an initial design that you specify. Then it considers swapping out every design alternative/set and replacing it with each candidate alternative/set. Swaps that increase efficiency are performed. Swapping continues until efficiency stabilizes. Then the process is repeated with a different initial design. The best design is output for use.

The macro uses a modified Federov algorithm, just like PROC OPTEX. The key difference between this macro and PROC OPTEX is this macro allows you to specify the true (or assumed true) parameters and optimizes the variance matrix for a multinomial logit model, whereas PROC OPTEX optimizes the variance matrix for a linear model which does not depend on the parameters.

Here are some usage samples. This first example creates a design for a generic model with 3 three-level factors. First, the %MKTDES macro is used to create a candidate set where **x1-x3** are the factors and **f1-f3** are the flags. Since this is a generic model, each alternative can appear anywhere, so all flags are a constant: **f1=1 f2=1 f3=1**. Then the %CHOICEFF macro is run to find an efficient design for the unbranded, purely generic model assuming $\boldsymbol{\beta} = 0$.

```
%mktdes(factors=x1-x3=3 f1-f3=1, run=plan)

%choiceff(data=cand1, model=class(x1-x3), nsets=9,
          flags=f1-f3, beta=zero, seed=145)

proc print; var set x1-x3; run;
```

The option **data=cand1** 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	0.7934651272	1.260294833
	1	1.6792104859	0.5955179582
	2	1.6954565904	0.589811621
	3	1.6954565904	0.589811621

Design	Iteration	D-Efficiency	D-Error
2	0	0.9948981202	1.0051280424
	1	1.6252969583	0.6152721784
	2	1.6779447607	0.5959671757
	3	1.6872641431	0.5926754291
	4	1.6872641431	0.5926754291

Final Results: Design = 1
Efficiency = 1.6954565904
D-Error = 0.589811621

n	Variable Name	Label	Variance	DF	Standard Error
1	x11	x1 1	0.69335	1	0.83267
2	x12	x1 2	0.69335	1	0.83267
3	x21	x2 1	0.67568	1	0.82199
4	x22	x2 2	0.67568	1	0.82199
5	x31	x3 1	0.70166	1	0.83765
6	x32	x3 2	0.72973	1	0.85424
				==	
				6	

Obs	Set	x1	x2	x3
1	1	1	3	1
2	1	2	1	2
3	1	3	2	3
4	2	2	1	3
5	2	3	3	1
6	2	1	2	2
7	3	1	1	3
8	3	3	3	2
9	3	2	2	1
10	4	1	3	1
11	4	2	1	3
12	4	3	2	2
13	5	1	2	1
14	5	2	3	2
15	5	3	1	3

```
16      6       1       2       2
17      6       3       1       1
18      6       2       3       3
19      7       2       2       1
20      7       1       1       2
21      7       3       3       3
22      8       3       1       1
23      8       1       3       3
24      8       2       2       2
25      9       3       2       3
26      9       2       3       2
27      9       1       1       1
```

The output consists of a list of the parameter names, values and labels, followed by two iteration histories, a report in the most efficient design found, the parameter names, variances, *df*, and standard errors. Finally, the design is printed.

These next steps manually create an optimal design for this problem and evaluate its efficiency using the initial design options. The data step creates a cyclic design. In a cyclic design, the factor levels increase cyclically (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
;

%choiceff(data=cand1, 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.

These next steps create a design for this same problem using the candidate set swapping algorithm. The first steps create a candidate set of choice sets.

```
%mktdes(factors=x1-x9=3, size=2187, run=factex)

data key;
   input (x1-x3) ($);
   datalines;
x1 x2 x3
x4 x5 x6
x7 x8 x9
;

%mktroll(design=cand1, key=key, out=rolled)

%choiceff(data=rolled, model=class(x1-x3), nsets=9, nalts=3, beta=zero);
```

The **`nalts=3`** option specifies that there are three alternatives. When we swap choice sets we need to specify

nalts=. The output from these steps is not appreciably different from what we saw previously, so it is not shown.

These next steps create a design for an example with brand effects using the alternative swapping algorithm.

```
%mktdes(factors=x1-x3 Brand=3, run=plan)

data full(drop=i);
   set cand1;
   array f[3];
   do i = 1 to 3; f[i] = (brand eq i); end;
   run;

proc print data=full(obs=6); run;

%choiceff(data=full,
          model=class(brand brand*x1 brand*x2 brand*x3 / zero=' '),
          nsets=15, flags=f1-f3, beta=zero, converge=1e-12);
```

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.

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

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
.			
.			
.			

Design	Iteration	D-Efficiency	D-Error
1	0	0	.
	1	0	.
		0.2968623575 (Ridged)	
	.		
	.		
	.		
	9	0	.
		0.3060697239 (Ridged)	

```
        Design   Iteration  D-Efficiency       D-Error
        ------------------------------------------------
           2         0                0             .
                     1                0             .
                            0.2988838889 (Ridged)
                     .
                     .
                     .
                     7                0             .
                              0.304627967 (Ridged)

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

Redundant Variables:

Brand3

              Variable                                          Standard
         n    Name         Label              Variance    DF      Error

         1    Brand1       Brand 1             4.08854     1     2.02201
         2    Brand2       Brand 2             4.44804     1     2.10904
         3    Brand3       Brand 3               .         0      .
         4    Brand1x11    Brand 1 * x1 1      2.34244     1     1.53050
         .
         .
         .
        21    Brand3x32    Brand 3 * x3 2      2.23043     1     1.49346
                                                          ==
                                                          20
```

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**. This can be seen from both the "Redundant Variables" list and from looking at the variance and *df* table. This makes the efficiency of the design zero. However, the 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. 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 sample usage on page 272, the option **initvars=x1-x3** was specified because the initial design was not created by the %CHOICEFF macro.) Here is some of the output. Notice that now there are no zero parameters so D-efficiency can be directly computed.

```
        Design   Iteration  D-Efficiency       D-Error
        ------------------------------------------------
           1         0        0.685891335  1.4579568934
                     1        0.685891335  1.4579568934
```

```
Final Results:   Design     = 1
                 Efficiency =  0.685891335
                 D-Error    = 1.4579568934

             Variable                                          Standard
          n  Name         Label              Variance    DF      Error

          1  Brand1       Brand 1             4.08854     1    2.02201
          2  Brand2       Brand 2             4.44804     1    2.10904
          3  Brand1x11    Brand 1 * x1 1      2.34244     1    1.53050
          4  Brand1x12    Brand 1 * x1 2      2.49307     1    1.57895
          5  Brand2x11    Brand 2 * x1 1      2.08522     1    1.44403
          6  Brand2x12    Brand 2 * x1 2      2.09687     1    1.44806
          7  Brand3x11    Brand 3 * x1 1      2.34500     1    1.53134
          8  Brand3x12    Brand 3 * x1 2      2.09753     1    1.44829
          9  Brand1x21    Brand 1 * x2 1      2.21583     1    1.48857
         10  Brand1x22    Brand 1 * x2 2      2.15893     1    1.46933
         11  Brand2x21    Brand 2 * x2 1      2.16923     1    1.47283
         12  Brand2x22    Brand 2 * x2 2      1.81748     1    1.34814
         13  Brand3x21    Brand 3 * x2 1      2.17901     1    1.47615
         14  Brand3x22    Brand 3 * x2 2      1.75709     1    1.32555
         15  Brand1x31    Brand 1 * x3 1      2.03717     1    1.42729
         16  Brand1x32    Brand 1 * x3 2      2.08386     1    1.44356
         17  Brand2x31    Brand 2 * x3 1      2.12721     1    1.45850
         18  Brand2x32    Brand 2 * x3 2      2.37880     1    1.54234
         19  Brand3x31    Brand 3 * x3 1      2.13410     1    1.46085
         20  Brand3x32    Brand 3 * x3 2      2.23043     1    1.49346
                                                         ==
                                                         20
```

These next steps handle the same problem only this time using the set swapping algorithm and we will specify a parameter vector that is not zero. At first, we will omit the **beta=** option to just see the coding. We specified the **effects** option in the PROC TRANSREG **class** specificiation of -1, 0, 1 coding.

```
%mktdes(factors=x1-x9=3, size=2187, run=factex)

data key;
   input (Brand x1-x3) ($);
   datalines;
1 x1 x2 x3
2 x4 x5 x6
3 x7 x8 x9
;

%mktroll(design=cand1, 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=' '))
```

Here is the output. This 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 specify the assumed betas on the **beta=** option. Assume we have some good reason for picking these numbers. 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,
          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	0.014207	1.00	1	0.11919	8.3897	0.0001
2	Brand2	Brand 2	0.027615	2.00	1	0.16618	12.0353	0.0001
3	Brand1x11	Brand 1 * x1 1	0.012304	-0.50	1	0.11092	-4.5076	0.0001
4	Brand1x12	Brand 1 * x1 2	0.008256	0.50	1	0.09086	5.5027	0.0001
5	Brand2x11	Brand 2 * x1 1	0.009015	-0.75	1	0.09495	-7.8991	0.0001
6	Brand2x12	Brand 2 * x1 2	0.013755	0.75	1	0.11728	6.3949	0.0001
7	Brand3x11	Brand 3 * x1 1	0.031508	-1.00	1	0.17751	-5.6336	0.0001
8	Brand3x12	Brand 3 * x1 2	0.019487	1.00	1	0.13960	7.1635	0.0001
9	Brand1x21	Brand 1 * x2 1	0.011585	-0.50	1	0.10763	-4.6454	0.0001
10	Brand1x22	Brand 1 * x2 2	0.010046	0.50	1	0.10023	4.9885	0.0001
11	Brand2x21	Brand 2 * x2 1	0.012541	-0.75	1	0.11199	-6.6972	0.0001
12	Brand2x22	Brand 2 * x2 2	0.015142	0.75	1	0.12305	6.0950	0.0001
13	Brand3x21	Brand 3 * x2 1	0.020645	-1.00	1	0.14368	-6.9598	0.0001
14	Brand3x22	Brand 3 * x2 2	0.018324	1.00	1	0.13537	7.3874	0.0001
15	Brand1x31	Brand 1 * x3 1	0.008624	-0.50	1	0.09287	-5.3841	0.0001
16	Brand1x32	Brand 1 * x3 2	0.009486	0.50	1	0.09740	5.1336	0.0001
17	Brand2x31	Brand 2 * x3 1	0.011025	-0.75	1	0.10500	-7.1428	0.0001
18	Brand2x32	Brand 2 * x3 2	0.013454	0.75	1	0.11599	6.4659	0.0001

```
19 Brand3x31 Brand 3 * x3 1 0.023265  -1.00   1  0.15253  -6.5562 0.0001
20 Brand3x32 Brand 3 * x3 2 0.017184   1.00   1  0.13109   7.6286 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.

```
%mktdes(factors=x1-x5=3, run=plan)

data key;
   input (Brand Price) ($);
   datalines;
1 x1
2 x2
3 x3
4 x4
5 x5
. .
;

%mktroll(design=cand1, key=key, alt=brand, out=rolled, keep=x1-x5)

%choiceff(data=rolled,
          model=class(brand brand*price / zero=none)
                class(brand / zero=none) * identity(x1-x5),
          nsets=20, nalts=6, beta=zero);
```

Here is a tiny portion of the output.

```
Redundant Variables:

Brand1Price3 Brand2Price3 Brand3Price3 Brand4Price3 Brand5Price3 Brand1x1
Brand2x2 Brand3x3 Brand4x4 Brand5x5
```

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,
          model=class(brand brand*price / zero=' ')
                class(brand / zero=none) * identity(x1-x5),
          drop=Brand1x1 Brand2x2 Brand3x3 Brand4x4 Brand5x5,
          nsets=20, nalts=6, beta=zero);
```

Here is the last part of the output.

n	Variable Name	Label	Variance	DF	Standard Error
1	Brand1	Brand 1	13.3592	1	3.65503
2	Brand2	Brand 2	11.0073	1	3.31773
3	Brand3	Brand 3	13.5813	1	3.68528
4	Brand4	Brand 4	12.2061	1	3.49372
5	Brand5	Brand 5	12.7614	1	3.57231
6	Brand1Price1	Brand 1 * Price 1	2.8638	1	1.69226
7	Brand1Price2	Brand 1 * Price 2	3.8494	1	1.96198
8	Brand2Price1	Brand 2 * Price 1	2.8912	1	1.70034
9	Brand2Price2	Brand 2 * Price 2	3.6813	1	1.91866

```
10    Brand3Price1    Brand 3 * Price 1      2.8555     1     1.68983
11    Brand3Price2    Brand 3 * Price 2      3.6010     1     1.89764
12    Brand4Price1    Brand 4 * Price 1      2.7100     1     1.64619
13    Brand4Price2    Brand 4 * Price 2      5.2074     1     2.28196
14    Brand5Price1    Brand 5 * Price 1      2.8557     1     1.68987
15    Brand5Price2    Brand 5 * Price 2      3.7769     1     1.94342
16    Brand2x1        Brand 2 * x1           0.7155     1     0.84586
17    Brand3x1        Brand 3 * x1           0.7159     1     0.84611
18    Brand4x1        Brand 4 * x1           0.7155     1     0.84588
19    Brand5x1        Brand 5 * x1           0.7299     1     0.85437
20    Brand1x2        Brand 1 * x2           0.7227     1     0.85012
21    Brand3x2        Brand 3 * x2           0.7372     1     0.85858
22    Brand4x2        Brand 4 * x2           0.7401     1     0.86026
23    Brand5x2        Brand 5 * x2           0.7231     1     0.85034
24    Brand1x3        Brand 1 * x3           0.7141     1     0.84507
25    Brand2x3        Brand 2 * x3           0.7206     1     0.84889
26    Brand4x3        Brand 4 * x3           0.7275     1     0.85292
27    Brand5x3        Brand 5 * x3           0.7141     1     0.84506
28    Brand1x4        Brand 1 * x4           0.6867     1     0.82866
29    Brand2x4        Brand 2 * x4           0.6773     1     0.82298
30    Brand3x4        Brand 3 * x4           0.6774     1     0.82307
31    Brand5x4        Brand 5 * x4           0.6773     1     0.82299
32    Brand1x5        Brand 1 * x5           0.7282     1     0.85334
33    Brand2x5        Brand 2 * x5           0.7137     1     0.84480
34    Brand3x5        Brand 3 * x5           0.7141     1     0.84507
35    Brand4x5        Brand 4 * x5           0.7260     1     0.85204
                                                        ==
                                                        35
```

%CHOICEFF Macro Options

The following options can be used with the %CHOICEFF macro. You must specify both the **model=** and **nsets=** options and either the **flags=** or **nalts=** options. The rest of the options are optional.

You must specify both of these next two options.

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=' ')
      class(b / zero=none) * identity(x1-x5),
```

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 alternatives to be swapped, specify **flags=**. When the candidate set consists of sets of entire choice sets to be swapped, specify **nalts=**.

flags= *variable-list*
flags the alternative(s) for which each candidate may be used. There must be one flag variable per alternative. For example, 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**, alternative 3 candidates are indicated by **f1=0 f2=0 f3=1**.

If every candidate can be used in all alternatives, then the flags are constant: **f1=1 f2=1 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.

beta= *list*
specifies the true parameters. By default, when **beta=** is not specified, the macro just reports on coding. You may specify **beta=zero** to assume all zeros. Otherwise specify a number list: **beta=1 -1 2 -2 1 -1**.

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.

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*
names 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 may 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=** is 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. Then 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
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
Number of observations to use in variance matrix formula. By default, **n=1**.

options= options-list
lists binary options, for example **options=coded tests**. By default, **options=tests**.

- **coded** - prints the coded candidate set.
- **detail** - prints the details of the swaps.
- **tests** - prints the diagonal of covariance matrix, and hypothesis tests for this n and beta. When beta is not zero, the results include a Wald test statistic which is normally distributed (Beta / Standard Error) and the probability of a larger squared Wald statistic.
- **notes** - stops the macro from submitting the statement **options nonotes**.
- **orthcan** - orthogonalizes the candidate set.
- **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).

out= *SAS-data-set*
specifies a name for the output SAS data set with the final designs. By default, this data set is called RESULTS.

seed= n
specifies a random number seed. By default **seed=0** and clock time is used as the random number seed.

set= *variable*
specifies a choice set ID variable. For release 8.0 (or earlier releases) if you are using the 8.0 autocall macro and not an updated macro, when you are using the set swapping algorithm and a large candidate set, specify a variable that identifies the choice sets. Then PROC TRANSREG will code **by &set** which may be more efficient. The 8.1 macro ignores this option and codes up to blocks of 5000 observations at a time.

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 x 4 matrix of **b** and **c** main effects.

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.

%MKTROLL Macro Overview

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 %MKTDES 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=** may 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**.,

```
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:

```
%mktdes(factors=x1-x5=3, n=12)
```

Here is how you can create the design with one row per alternative per choice set:

```
%mktroll(design=design, key=key, out=sasuser.design, alt=brand)
```

For example, if the data set DESIGN contains the row:

Obs	x1	x2	x3	x4	x5
9	1	3	2	3	1

Then the data set SASUSER.DESIGN contains the rows:

Obs	Set	Brand	Price
41	9	A	1
42	9	B	3
43	9	C	2
44	9	D	3
45	9	E	1

The price for Brand A is made from **x1=1**, ..., and the price for Brand E is made from **x5=3**.

Now assume that there are three alternatives, each composed of four factors: **Brand**, **Price**, **Size**, **Color**, and **Shape**. In addition, there is a constant alternative. First, the %MKTDES macro is used to create a design with 12 factors, one for each attribute of each alternative.

```
%mktdes(factors=x1-x12=2, n=16)
```

Then the **key=** data set is created. It shows that there are three brands, A, B, and C, & 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 SAS 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=design, out=sasuser.design, alt=brand)
```

For example, if the data set DESIGN contains the row:

Obs	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12
8	2	1	1	1	1	1	2	2	2	2	1	2

Then the data set SASUSER.DESIGN contains the rows:

Obs	Set	Brand	Price	Size	Color	Shape
29	8	A	2	1	1	1
30	8	B	1	1	2	2
31	8	C	2	2	1	2
32	8	None	.	.	.	.

Now assume like before that there are three alternatives, each composed of four factors: **Brand**, **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 %MKTDES macro is used to create a design with 14 factors, one for each attribue of each alternative.

```
%mktdes(factors=x1-x14=2, n=16)
```

Then 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=design, out=sasuser.design, alt=brand)
```

For example, if the data set DESIGN contains the row:

Obs	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14
8	2	1	1	1	1	1	2	2	2	2	1	2	2	1

Then the data set SASUSER.DESIGN contains the rows:

Obs	Set	Brand	Price	Size	Color	Shape	Pattern
29	8	A	2	1	1	1	2
30	8	B	1	1	2	2	.
31	8	C	2	2	1	2	1
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=design, out=sasuser.design, alt=brand,
         keep=x1 x5 x9)
```

Now the data set also contains the three original price variables.

Obs	Set	Brand	Price	Size	Color	Shape	Pattern	x1	x5	x9
29	8	A	2	1	1	1	2	2	1	2
30	8	B	1	1	2	2	.	2	1	2
31	8	C	2	2	1	2	1	2	1	2
32	8	None	.	.	.	.	.	2	1	2

The macro performs some rudimentary error checking. 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 %CHOICEFF macro. You must specify the **design=** and **key=** options.

alt= *variable*

names 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.

debug= *list*

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

design= *SAS-data-set*
names an input SAS data set with one row per choice set. The **design=** option must be specified.

keep= *variable-list*
names 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*
names 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.

out= *SAS-data-set*
The **out=** option names the output SAS data set. If **out=** is not specified, the DATAn convention is used.

set= *variable*
names the variable in the **out=** data set that will contain the choice set number. By default this variable is named **Set**.

%MKTMERGE Macro Overview

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, one set per block. 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*
either contains a 1 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*
names 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*
names an input SAS data set for a choice model. This data set may have been created for example with the %MKTROLL macro. This option must be specified.

nalts= n
is the number of alternatives. This option must be specified.

nsets= n
is the number of choice sets. This option must be specified.

out= *SAS-data-set*
names 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*
is 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.

%MKTALLO Macro Overview

The %MKTALLO autocall macro is used for manipulating data for a 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 * 11 = 297 observations. Here is an example 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 a sample 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*

names the input SAS data set. By default, the macro uses the last data set created.

freq= *variable*

names the frequency variable, which contains the number of times this alternative was chosen. This option must be specified.

nalts= *n*
names the number of alternatives (including if appropriate the constant alternative). This option must be specified.

out= *SAS-data-set*
names the output SAS data set. The default is **out=allocs**.

vars= *variable-list*
names the variables in the data set that will be used in the analysis but not the **freq=** variable. This option must be specified.

%PHCHOICE Macro Overview

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 of 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)
```

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 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 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 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. As long as we are changing 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 to use the vocabulary of choice models instead of survival analysis models. So the **Stat.Phreg.CensoredSummary** is also edited. 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;
      notes "Number of Events and Censored";
      dynamic ndec;
      column Stratum 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 Stratum;
         header = "Stratum";
         translate _val_=.A into "Total";
         format = 5.0;
         style = RowHeader;
         id;
      end;
      define GenericStrVar;
         generic;
      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;
      col_space_max = 4;
      col_space_min = 1;
      control = control_var;
      use_name;
      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 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**. For positional parameters, just a value is specified (unlike keyword parameters which have the form **KEYWORD=value**).

onoff

ON turns on 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

contains 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, (saswfk@wnt.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 Kuhfeld, Tobias, and Garratt (1994).

References

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. Dennis and Christopher J. Nachtsheim (1980), “A Comparison of Algorithms for Constructing Exact D-optimal Designs,” *Technometrics*, 22 (August), 315−24.

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

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.

Multinomial Logit Models*

Ying So and Warren F. Kuhfeld

SAS Institute Inc, 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

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

$\boldsymbol{\beta}_1,\ldots,\boldsymbol{\beta}_m$ are m vectors of unknown regression parameters (each of which is different, even though $\mathbf{X}_j$ is

constant across alternatives). Since $\sum_{k=1}^{m} \Pi_{jk} = 1$, the m sets of parameters are not unique. By setting the last set of coefficients to null (that is, $\boldsymbol{\beta}_m = 0$), the coefficients $\boldsymbol{\beta}_k$ represent the effects of the $\mathbf{X}$ variables on the probability of choosing the kth alternative over the last alternative. In fitting such a model, you estimate $m - 1$ sets of regression coefficients.

In the conditional logit model, the explanatory variables $\mathbf{Z}$ assume different values for each alternative and the impact of a unit of $\mathbf{Z}$ is assumed to be constant across alternatives. For example, for each of the m travel modes, $\mathbf{Z}_{jk} = (time)'$, and for the first subject, $\mathbf{Z}_{11} = (10)'$, $\mathbf{Z}_{12} = (4.5)'$, and $\mathbf{Z}_{13} = (10.5)'$. The probability that the individual j chooses alternative k is

$$\Pi_{jk} = \frac{\exp(\boldsymbol{\theta}'\mathbf{Z}_{jk})}{\sum_{l=1}^{m} \exp(\boldsymbol{\theta}'\mathbf{Z}_{jl})} = \frac{1}{\sum_{l=1}^{m} \exp[\boldsymbol{\theta}'(\mathbf{Z}_{jl} - \mathbf{Z}_{jk})]}$$

$\boldsymbol{\theta}$ is a single vector of regression coefficients. The impact of a variable on the choice probabilities derives from the difference of its values across the alternatives.

For the mixed logit model that includes both characteristics of the individual and the alternatives, the choice probabilities are

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

$\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_{m-1}$ and $\boldsymbol{\beta}_m \equiv 0$ 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

Effect	Parameter	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	3	-0.0710	0.0652	1.19	0.2762
	4	-0.0500	0.0596	0.70	0.4013

Note that there are two intercept coefficients and two slope coefficients for AGE. The first INTERCEPT and the first AGE coefficients correspond to the effect on the probability of choosing auto over transit, and the second intercept and second age coefficients correspond to the effect of choosing plane over transit.

Let $\mathbf{X}_j$ be a $(p+1)$-vector representing the characteristics of individual j. The generalized logit model can be cast in the framework of a conditional model by defining the global parameter vector $\boldsymbol{\theta}$ and the alternative-specific regressor variables $\mathbf{Z}_{jk}$ as follows:

$$\boldsymbol{\theta} = \begin{bmatrix} \boldsymbol{\beta}_1 \\ \boldsymbol{\beta}_2 \\ \vdots \\ \boldsymbol{\beta}_{m-1} \end{bmatrix} \quad \mathbf{Z}_{j1} = \begin{bmatrix} \mathbf{X}_j \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \mathbf{Z}_{j2} = \begin{bmatrix} 0 \\ \mathbf{X}_j \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \ldots \quad \mathbf{Z}_{j,m-1} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \mathbf{X}_j \end{bmatrix} \quad \mathbf{Z}_{jm} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

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;
```

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	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 travelling 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 175, 180, 187, and 305 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;
   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 175, 180, 187, and 301 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