

# Discrete Choice

Warren F. Kuhfeld

## Abstract

Discrete choice modeling is a popular technique in marketing research, transportation, and other areas. It is used to help researchers understand people's stated choice of alternative products and services. We discuss designing a choice experiment, preparing the questionnaire, inputting and processing the data, performing the analysis, and interpreting the results.\* Most of the discussion is on designing the choice experiment.

## Introduction

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

The design and analysis of a discrete choice experiment is explained in the context of a series examples.<sup>†</sup> There are also several very basic introductory examples starting on page 127 in the introduction to experimental design chapter, which starts on page 53. Be sure to read the design chapter before proceeding to the examples in this chapter. The examples are as follows:

- The candy example (page 289) is a first, very simple example that discusses the multinomial logit model, the input data, analysis, results, and computing the probability of choice.
- The fabric softener example (page 302) is a small, somewhat more realistic example that discusses designing the choice experiment, randomization, generating the questionnaire, entering and processing the data, analysis, results, probability of choice, and custom questionnaires.

---

\*Copies of this chapter (MR-2010F), the other chapters, sample code, and all of the macros are available on the Web [http://support.sas.com/resources/papers/tnote/tnote\\_marketresearch.html](http://support.sas.com/resources/papers/tnote/tnote_marketresearch.html). Specifically, sample code is here <http://support.sas.com/techsup/technote/mr2010f.sas>. For help, please contact SAS Technical Support. See page 25 for more information. This document would not be possible without the help of Randy Tobias who contributed to the discussion of experimental design and Ying So who contributed to the discussion of analysis. Randy Tobias wrote PROC FACTEX and PROC OPTEX. Ying So wrote PROC PHREG. Warren F. Kuhfeld wrote PROC TRANSREG and all of the macros.

<sup>†</sup>All of the example data sets are artificially generated.

- The first vacation example (page 339) is a larger, symmetric example (all factors have the same number of levels) that discusses designing the choice experiment, blocks, randomization, generating the questionnaire, entering and processing the data, coding, and alternative-specific effects.
- The second vacation example (page 410) is a larger, asymmetric example (not all factors have the same number of levels) that discusses designing the choice experiment, blocks, blocking an existing design, interactions, generating the questionnaire, generating artificial data, reading, processing, and analyzing the data, aggregating the data to save time and memory.
- The brand choice example (page 444) is a small example that discusses the processing of aggregate data, the mother logit model, and the likelihood function.
- The food product example (page 468) is a medium sized example that discusses asymmetry, coding, checking the design to ensure that all effects are estimable, price cross-effects, availability cross-effects, interactions, overnight design searches, modeling subject attributes, and designs when balance is of primary importance.
- The drug allocation example (page 535) is a small example that discusses data processing for studies where respondents potentially make multiple choices.
- The chair example (page 556) is a purely generic-attributes study, and it uses the `%ChoiceEff` macro to create experimental designs.
- The next example section (page 580) shows how to improve an existing design and augmenting a design with some choice sets are fixed in advance.
- The last example section (page 595) discusses partial-profile designs and designs with restrictions. Also see page 1079 for an example of a choice design with a complicated set of restrictions.

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

- The `%MktRuns` autocall macro suggests design sizes. See page 1159 for documentation.
- The `%MktEx` autocall macro generates designs for linear models. These designs are directly used for conjoint studies. After post-processing they can also be used as choice designs. They are also used to make candidate sets for directly constructing choice designs. Most examples use the `%MktEx` macro in some capacity. See page 1017 for documentation.
- The `%MktEval` autocall macro evaluates linear model designs. See page 1012 for documentation.
- The `%ChoiceEff` autocall macro both generates and evaluates choice designs. See page 806 for documentation.
- The autocall macros `%MktKey`, `%MktRoll`, `%MktMerge`, and `%MktAllo` prepare the data and design for analysis. See pages 1090, 1153, 1125, and 956 for documentation.
- PROC TRANSREG codes our designs, which puts the data into the final form for analysis.
- The `%PHChoice` autocall macro customizes our displayed output. This macro uses PROC TEMPLATE and ODS (Output Delivery System) to customize the output from PROC PHREG, which fits the multinomial logit model. See page 1173 for documentation.
- The `%MktBal` macro makes perfectly balanced designs for main effects models. See page 959 for documentation.

- The `%MktBlock` macro block a linear or choice design into sets of alternatives or choice sets. See page 979 for documentation.
- The `%MktDups` macro searches a design for duplicate runs or choice sets. See page 1004 for documentation.
- The `%MktLab` macro assigns variable names, labels and levels to experimental designs and adds an intercept. See page 1093 for documentation.
- The `%MktOrth` macro lists the orthogonal experimental designs that the `%MktEx` macro can produce. See page 1128 for documentation.
- The `%MktPPro` macro makes certain partial-profile choice designs. See page 1145 for documentation.
- The `%MktBIBD` macro makes balanced incomplete block designs, unbalanced block designs, and more generally, incomplete block designs, which are useful in constructing certain partial-profile designs and MaxDiff designs. See page 963 for documentation.
- The `%MktMDiff` macro processes and analyzes data from MaxDiff (best-worst) studies. See page 1105 for documentation.

All of these macros are distributed with SAS 9.2 as autocall macros (see page 803 for more information about autocall macros), however, you should get the latest versions of the macros from the Web [http://support.sas.com/resources/papers/tnote/tnote\\_marketresearch.html](http://support.sas.com/resources/papers/tnote/tnote_marketresearch.html).

## Experimental Design

Experimental design is a fundamental component of choice modeling. A discrete choice experimental design consists of sets of products, and subjects choose a product from each set. Often, the most challenging part of the entire study is making the design. There are many examples of making choice designs in this chapter. Before you read them, be sure to read the design chapter beginning on page 53. There are also a number of design examples with the macro documentation. After you become familiar with the design chapter, you should check out the `%ChoiceEff` macro examples starting on page 808.

## Customizing the Multinomial Logit Output

You can fit the multinomial logit model for discrete choice experiments by using the SAS/STAT procedure PHREG (proportional hazards regression), with the `ties=breslow` option. The likelihood function for the multinomial logit model has the same form as a survival analysis model fit by PROC PHREG. PROC PHREG and its output are primarily designed for survival-analysis studies. Before you fit the multinomial logit model with PROC PHREG, you can customize its output templates to make them more appropriate for choice experiments by using the `%PHChoice` autocall macro. See page 803 for information about autocall macros. You can run the following macro to customize PROC PHREG output:

```
%phchoice(on)
```

The macro uses PROC TEMPLATE and ODS (Output Delivery System) to customize the output from PROC PHREG. Running this step edits two of the PROC PHREG templates and stores copies in the `sasuser` library. The default templates that SAS provides are stored in the library `sashelp.tmplmst`. By default, if you modify a template, it is stored in the library `sasuser.templat`. By default, ODS searches `sasuser.templat` for templates, and then it searches `sashelp.tmplmst` if it does not find the requested template in `sasuser.templat`. You can see the list of template libraries by submitting the following statement:

```
ods path show;
```

The template changes made by the `%PHChoice` macro do not affect the numerical output from PROC PHREG, but they do change the format and some of the titles, labels, and column headers that are used to label the output. 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 display variable names. If you are coding with PROC TRANSREG, this is usually the case. These changes remain in effect until you delete them. To return to the default output from PROC PHREG, run the following macro:

```
%phchoice(off)
```

More generally, you can run the following step to delete the entire `sasuser.templat` library of customized templates so that ODS uses only the SAS supplied templates:

```
proc datasets library=sasuser;  
  delete templat(memtype=itemstor);  
run;
```

If you only use PROC PHREG for choice modeling, and if you do not delete the contents of the `sasuser` or the template library, you only have to run the `%PHChoice` macro once, and the template changes are there for all subsequent steps that use the same template library in the `sasuser` library. Alternatively, you can choose to run `phchoice(on)` before running PROC PHREG and `phchoice(off)` after you are done. If you plan on using PROC PHREG for survival analysis, you should be sure to run `phchoice(off)` first so that your output is labeled appropriately for a survival study. See page 1173 for more information about the `%PHChoice` macro.

# Candy Example

We begin with a very simple introductory example. In this example, we discuss the multinomial logit model, data input and processing, analysis, results, interpretation, and probability of choice. Many aspects of this example, the experimental design in particular, are simpler than almost all realistic choice studies. Still, it is useful to start with a simple choice study with no experimental design issues to consider. In this example, each of ten subjects is 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 coded attributes and  $\boldsymbol{\beta}$  is a vector of unknown attribute parameters. At the heart- of this formula is a linear part-worth utility function,  $\mathbf{x}_i\boldsymbol{\beta}$  (like a linear regression function, but wrapped in a more complicated nonlinear model).  $U(c_i) = \mathbf{x}_i\boldsymbol{\beta}$  is the utility for alternative  $c_i$ , which is a linear function of the attributes. The probability that an individual will choose one of the  $m$  alternatives,  $c_i$ , from choice set  $C$  is the exponential of the utility of the alternative divided by the sum of all of the exponentiated utilities.

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

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

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

U(Milk, Chewy, No Nuts)	=	$0 \times 4$	+	$0 \times -2$	+	$0 \times 1$	=	0
U(Milk, Chewy, Nuts )	=	$0 \times 4$	+	$0 \times -2$	+	$1 \times 1$	=	1
U(Milk, Soft, No Nuts)	=	$0 \times 4$	+	$1 \times -2$	+	$0 \times 1$	=	-2
U(Milk, Soft, Nuts )	=	$0 \times 4$	+	$1 \times -2$	+	$1 \times 1$	=	-1
U(Dark, Chewy, No Nuts)	=	$1 \times 4$	+	$0 \times -2$	+	$0 \times 1$	=	4
U(Dark, Chewy, Nuts )	=	$1 \times 4$	+	$0 \times -2$	+	$1 \times 1$	=	5
U(Dark, Soft, No Nuts)	=	$1 \times 4$	+	$1 \times -2$	+	$0 \times 1$	=	2
U(Dark, Soft, Nuts )	=	$1 \times 4$	+	$1 \times -2$	+	$1 \times 1$	=	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

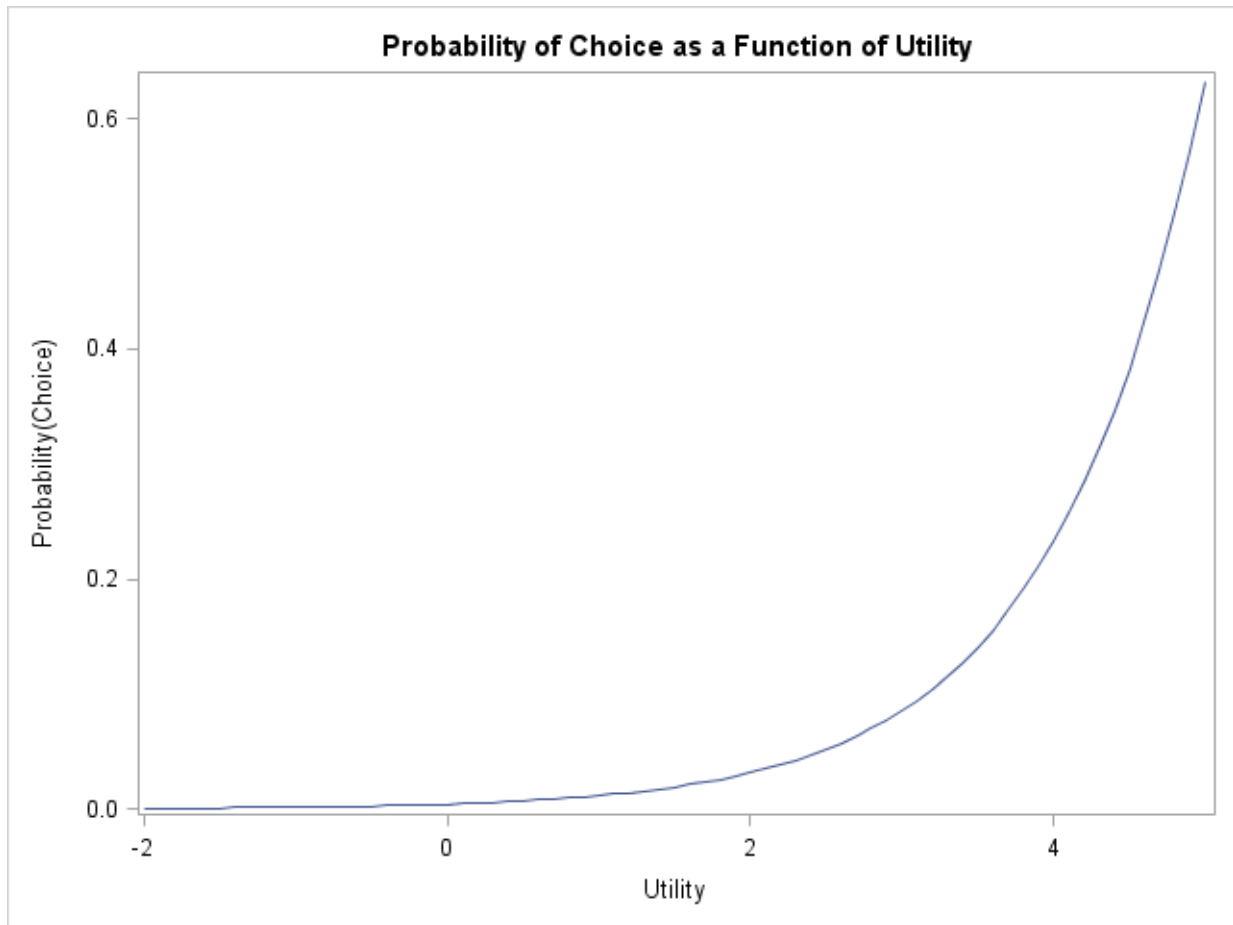
p(Milk, Chewy, No Nuts)	=	$\exp(0) / 234.707$	=	0.004
p(Milk, Chewy, Nuts )	=	$\exp(1) / 234.707$	=	0.012
p(Milk, Soft, No Nuts)	=	$\exp(-2) / 234.707$	=	0.001
p(Milk, Soft, Nuts )	=	$\exp(-1) / 234.707$	=	0.002
p(Dark, Chewy, No Nuts)	=	$\exp(4) / 234.707$	=	0.233
p(Dark, Chewy, Nuts )	=	$\exp(5) / 234.707$	=	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 does not change the probability of choice, however multiplying by a constant will.

Probability of choice is a nonlinear and increasing function of utility. The plot produced by the following steps 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;
  label p = 'Probability(Choice)' u = 'Utility';
run;

proc sgplot data=x;
  title 'Probability of Choice as a Function of Utility';
  series y=p x=u;
run;
```



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  $\mathbf{x}_2$  is chosen 2 times,  $\mathbf{x}_5$  is chosen 2 times,  $\mathbf{x}_6$  is chosen 5 times, and  $\mathbf{x}_7$  is 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 is chosen, and the set of alternatives from which it is 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 or three variables are used to identify the choice sets. These variables include subject ID and choice set number. In addition, larger studies might have a block ID variable when subjects see only a block of choice sets instead of every choice set. 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 more 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 (as a second or subsequent choice).

Both the chosen and the 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. The following statements read the data and store them in a SAS data set:

```

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
;

```



In this 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 the data entry in the way that requires the fewest programming statements. Each execution of the `input` statement reads information about one alternative. The `@@` in the `input` statement specifies that SAS should not automatically go to a new input data set line when it reads the next row of data. This specification is needed here because each line in the input data set contains the data for four output data set rows. The data from the first two subjects is displayed as follows:

```
proc print data=chocs noobs;
  where subj <= 2;
  var subj set c dark soft nuts;
run;
```

The data for the first two subjects is as follows:

---

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 and the data are stored in separate data sets. Then they are merged and processed to produce the same results as the preceding steps. The process of merging the experimental design and the data is explicitly illustrated here with a DATA step program. In practice, and in all of the other examples, we use the `%MktMerge` macro to do this. The following steps read the design, merge it with the data, and designate first and subsequent choices:

```

title 'Choice of Chocolate Candies';

* Alternative Form of Data Entry;

data combos;                                /* Read the design matrix.    */
  input Dark Soft Nuts;
  datalines;
0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1
;

data chocs;                                  /* Create the data set.      */
  input Choice @@; drop choice; /* Read the chosen combo num. */
  Subj = _n_; Set = 1;          /* Store subj, choice set num. */
  do i = 1 to 8;                /* Loop over alternatives.    */
    c = 2 - (i eq choice);      /* Designate chosen alt.      */
    set combos point=i;         /* Read design matrix.        */
    output;                      /* Output the results.        */
  end;
  datalines;
5 6 7 5 2 6 2 6 6 6
;

```

The variable `Choice` is the number of the chosen alternative. For each choice set, each of the eight observations in the experimental design is read. The `point=` option in the `set` statement is used to read the *i*th observation of the data set `Combos`. When *i* (the alternative index) equals `Choice` (the number of the chosen alternative), the logical expression `(i eq choice)` equals 1; otherwise it is 0. The statement `c = 2 - (i eq choice)` sets *c* to 1 (two minus one) when the alternative is chosen and 2 (two minus zero) otherwise. All eight observations in the `Combos` data set are read 10 times, once per subject. The resulting data set is the same as the one we created previously. As we mentioned previously, in all of the remaining examples, we simplify this process by using the `%MktMerge` macro to merge the design and data and flag the chosen and unchosen alternatives. Still, it is good to know what the `%MktMerge` step does. 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, and one observation per alternative per choice set per subject is written.

## Choice and Survival Models

In SAS, the multinomial logit model is fit with the SAS/STAT procedure PHREG (proportional hazards regression), with the `ties=breslow` option. The likelihood function of the multinomial logit model has

the same form as a survival-analysis model fit by PROC PHREG.

In a discrete choice study, subjects are presented with sets of alternatives and are asked to choose the most preferred alternative. The data for one choice set consist of one alternative that is chosen and  $m - 1$  alternatives that are not chosen. First choice is 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 have occurred after the censored time. In a discrete choice study, first choice occurs at time one, and all subsequent choices (second choice, third choice, and so on) are unobserved or censored. The survival and choice models are the same.

## Fitting the Multinomial Logit Model

The preceding steps arranged the data into the right form for analysis. You use PROC PHREG to fit the multinomial logit model 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 is 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 of the unchosen alternatives.

In the left side of the `model` statement, you specify the variables that indicate which alternatives are chosen and not chosen. While this could be two different variables, we 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 is chosen. The second `c` specifies that `c` indicates which alternatives are not chosen, and (2) means that observations with values of 2 are not chosen. When `c` is set up such that 1 indicates the chosen alternative and 2 indicates the unchosen alternatives, always specify `c*c(2)` on the left of the equal sign in the `model` statement. The attribute variables are specified after the equal sign. Specify `ties=breslow` after a slash to explicitly specify the likelihood function for the multinomial logit model. (Do not specify any other `ties=` options; `ties=breslow` specifies the most computationally efficient and always appropriate way to fit the multinomial logit model.) The `label` statement is added since we are using a template that assumes each variable has a label.

Note that the `c*c(n)` syntax allows second choice (`c=2`) and subsequent choices (`c=3`, `c=4`, ...) to be entered. Just enter in parentheses one plus the number of choices actually made. For example, with first and second choice data specify `c*c(3)`. Note, however, that most experts believe that second and subsequent choice data are much less reliable than first choice data (or last choice data).

## Multinomial Logit Model Results

Recall that we specified %phchoice(on) on page 287 to customize the output from PROC PHREG. The results are as follows:

---

### Choice of Chocolate Candies

#### The PHREG Procedure

#### Model Information

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

Number of Observations Read	80
Number of Observations Used	80

#### Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Stratum	Subj	Set	Number of Alternatives	Chosen Alternatives	Not Chosen
1	1	1	8	1	7
2	2	1	8	1	7
3	3	1	8	1	7
4	4	1	8	1	7
5	5	1	8	1	7
6	6	1	8	1	7
7	7	1	8	1	7
8	8	1	8	1	7
9	9	1	8	1	7
10	10	1	8	1	7
Total			80	10	70

#### Convergence Status

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

#### Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	41.589	28.727
AIC	41.589	34.727
SBC	41.589	35.635

## Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	12.8618	3	0.0049
Score	11.6000	3	0.0089
Wald	8.9275	3	0.0303

## Choice of Chocolate Candies

## The PHREG Procedure

## 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 next table shows that 80 observations are read from the input SAS data set, and all 80 of them are used in the analysis.

The `Summary of Subjects, Sets, and Chosen and Unchosen Alternatives` table is displayed by default and should be used to check the data entry. There are as many strata as there are combinations of the `Subj` and `Set` variables.\* In this case, there are ten strata. Each stratum must be composed of  $m$  alternatives. In this case, there are eight alternatives. The number of chosen alternatives should be 1, and the number of unchosen alternatives is  $m - 1$  (in this case, 7). **Always check the summary table to ensure that the data are arrayed correctly.**

The `Convergence Status` table shows that the iterative algorithm successfully converged. The next tables, `Model Fit Statistics` and `Testing Global Null Hypothesis: BETA=0` contain the overall fit of the model. The  $-2 \text{ LOG L}$  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 reject the null hypothesis of no relationship between choice and the attributes. Note that 41.589 ( $-2 \text{ LOG L Without Covariates}$ , which is  $-2 \text{ LOG L}$  for a model with no explanatory variables) minus 28.727 ( $-2 \text{ LOG L With Covariates}$ , which is  $-2 \text{ LOG L}$  for a model with all explanatory variables) equals 12.8618 (Model Chi-Square, which is used to test the effects of the explanatory variables).

The `Multinomial Logit Parameter Estimates` table is next. For each effect, it contains the maximum likelihood parameter estimate, its estimated standard error (the square root of the corresponding diagonal element of the estimated variance matrix), the Wald Chi-Square statistic (the square of the parameter estimate divided by its standard error), the  $df$  of the Wald Chi-Square statistic (1 unless the corresponding parameter is redundant or infinite, in which case the value is 0), and the  $p$ -value of the Chi-Squared statistic with respect to a chi-squared distribution with one  $df$ . The parameter estimate

---

\*More generally, there are as many strata as there are combinations of the `Subj`, `Set`, and block ID variable. In this case, there is only one block and no blocking variable.

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 perform the same analysis as before, only now, coefficients for both levels of the three attributes are displayed:

```
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;
```

Binary variables for the missing levels are created by subtracting the existing binary variables from 1. The output is as follows:

---

### Choice of Chocolate Candies

#### The PHREG Procedure

#### Model Information

Data Set	WORK.CHOC2
Dependent Variable	c
Censoring Variable	c
Censoring Value(s)	2
Ties Handling	BRESLOW
Number of Observations Read	80
Number of Observations Used	80

## Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Stratum	Subj	Set	Number of Alternatives	Chosen Alternatives	Not Chosen
1	1	1	8	1	7
2	2	1	8	1	7
3	3	1	8	1	7
4	4	1	8	1	7
5	5	1	8	1	7
6	6	1	8	1	7
7	7	1	8	1	7
8	8	1	8	1	7
9	9	1	8	1	7
10	10	1	8	1	7
-----					
Total			80	10	70

## Convergence Status

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

## Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	41.589	28.727
AIC	41.589	34.727
SBC	41.589	35.635

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	12.8618	3	0.0049
Score	11.6000	3	0.0089
Wald	8.9275	3	0.0303

## Choice of Chocolate Candies

## The PHREG Procedure

## 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 displayed. 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 is 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 is 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) to the macro variable '&sum';
if eof then call symputx('sum', sum);
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;

```

The results are as follows:

---

Choice of Chocolate Candies				
Obs	Dark	Soft	Nuts	p
1	Dark	Chewy	Nuts	0.50400
2	Dark	Chewy	No Nuts	0.21600
3	Milk	Chewy	Nuts	0.12600
4	Dark	Soft	Nuts	0.05600
5	Milk	Chewy	No Nuts	0.05400
6	Dark	Soft	No Nuts	0.02400
7	Milk	Soft	Nuts	0.01400
8	Milk	Soft	No Nuts	0.00600

---

The three most preferred alternatives are Dark/Chewy/Nuts, Dark/Chewy/No Nuts, and Milk/Chewy/Nuts.

## Fabric Softener Example

In this example, subjects are asked to choose among fabric softeners. This example shows all of the steps in a discrete choice study, including experimental design creation and evaluation, creating the questionnaire, inputting the raw data, creating the data set for analysis, coding, fitting the discrete choice model, interpretation, and probability of choice. In addition, custom questionnaires are discussed. We assume that you are familiar with the experimental design issues that are discussed starting on page 53.

### Set Up

The study involves four fictitious fabric softeners *Sploosh*, *Plumbbob*, *Platter*, and *Moosey*.<sup>\*</sup> There are 50 subjects, each of which see the same choice sets.

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. The total number of alternatives for this study is four, since there are four brands, and each consists of one attribute, namely price. Since the constant alternative does not vary, it does not enter into the experimental design at this stage. We use an approach in this example that is discussed in detail in the experimental design chapter starting on page 53. Namely, we start by making a “linear arrangement of a choice design,” with one factor for every attribute of every alternative, and use that to make our final choice design.

We can use the `%MktRuns` autocall macro to help us choose the number of choice sets. All of the autocall macros used in this book are documented starting on page 803. To use this macro, you specify the number of levels for each of the factors. With four price factors (one for each of the four brands) each with three prices, you specify four 3’s (or equivalently, `3 ** 4`). The `%MktRuns` macro is invoked as follows:

```
title 'Choice of Fabric Softener';

%mktruns(3 3 3 3)
```

The output from this macro is as follows:

---

#### Choice of Fabric Softener

##### Design Summary

Number of Levels	Frequency
3	4

---

<sup>\*</sup>Of course real studies use real brands. We have not collected real data, so we cannot use real brand names with artificial data. We picked these silly names so no one would confuse our artificial data with real data.

## Choice of Fabric Softener

Saturated = 9  
Full Factorial = 81

Some Reasonable Design Sizes	Violations	Cannot Be Divided By
9 *S	0	
18 *	0	
12	6	9
15	6	9
10	10	3 9
11	10	3 9
13	10	3 9
14	10	3 9
16	10	3 9
17	10	3 9

\* - 100% Efficient design can be made with the MktEx macro.  
S - Saturated Design - The smallest design that can be made.

## Choice of Fabric Softener

n	Design	Reference
9	3 ** 4	Fractional-Factorial
18	2 ** 1 3 ** 7	Orthogonal Array
18	3 ** 6 6 ** 1	Orthogonal Array

The output first tells us that we specified a design with four factors, each with three levels. The next table reports the size of the saturated design, which is the number of parameters in the linear model based on this design. After that, design sizes are suggested.

The output from this macro tells us that the saturated design has nine runs and the full-factorial design has 81 runs. It also tells us that 9 and 18 are optimal design sizes with zero violations. The macro tells us that in nine runs, an orthogonal design with 4 three-level factors is available, and in 18 runs, two orthogonal and balanced designs are available: one with a two-level factor and 7 three-level factors, and one with 6 three-level factors and a six-level factor. There are zero violations with these designs because these sizes can be divided by 3 and  $3 \times 3 = 9$ . Twelve and 15 are also reported as potential design sizes, but each has 6 violations. Six times (the  $4(4-1)/2 = 6$  pairs of the four 3's) the design sizes of 12 and 15 cannot be divided by  $3 \times 3 = 9$ . Ideally, we would like to have a manageable number of choice sets for people to evaluate and a design that is both orthogonal and balanced. When violations are reported, orthogonal and balanced designs are not possible. While orthogonality and balance are not required, they are nice properties to have. With 4 three-level factors, the number of choice sets in all orthogonal and balanced designs must be divisible by  $3 \times 3 = 9$ .

Nine choice sets is a bit small. Furthermore, there are no error  $df$ . We set the number of choice sets to 18 since it is small enough for each person to see all choice sets, large enough to have reasonable error  $df$ , and an orthogonal and balanced design is available. It is important to understand, however, that the concept of number of parameters and error  $df$  discussed here applies to the linear arrangement of the choice design and not to the choice model.\* We could use the nine-run design for a discrete choice model and have error  $df$  in the choice model. If we were to instead use this design for a full-profile conjoint (not recommended), there would be no error  $df$ .

To make the code easier to modify for future use, the number of choice sets and alternatives are stored in macro variables and the prices are put into a format. Our design, in raw form, has values for price of 1, 2, and 3. We use a format to assign the actual prices: \$1.49, \$1.99, and \$2.49. The format also creates a price of \$1.99 for missing, which is used for the constant alternative. The following statements create the macro variables and format:

```
%let n = 18;                /* n choice sets          */
%let m = 5;                /* m alternative including constant */
%let mm1 = %eval(&m - 1);  /* m - 1                  */

proc format;                /* create a format for the price */
  value price 1 = '$1.49' 2 = '$1.99' 3 = '$2.49' . = '$1.99';
run;
```

## Designing the Choice Experiment

The next step creates an efficient experimental design. We use the autocall macro `%MktEx` to create many of our designs. (All of the autocall macros used in this book are documented starting on page 803.) When you invoke the `%MktEx` macro for a simple problem, you only need to specify the numbers of levels and the number of runs. The macro does the rest. The `%MktEx` macro usage for this example is as follows:

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

The syntax `'n ** m'` means  $m$  factors each at  $n$  levels. This example has four factors, `x1` through `x4`, all with three levels. The `n=` option specifies the number of runs. The specification `n=&n` is equivalent to `n=18`, and it requests a design in 18 runs. These are all the options that are needed for a simple problem such as this one. However, throughout this book, random number seeds are explicitly specified with the `seed=` option so that the results are reproducible.\* The macro call with the random number seed specified is as follows:

```
%mktex(3 ** 4, n=&n, seed=17)
```

---

\*See page 67 for an explanation of the linear arrangement of a choice design versus the arrangement of a choice design that is more suitable for analysis.

\*By specifying a random number seed, results should be reproducible within a SAS release for a particular operating system and for a particular version of the macro. However, due to machine and macro differences, some results might not be exactly reproducible everywhere. For most orthogonal and balanced designs, the results should be reproducible. When computerized searches are done, it is likely that you will not get the same design as the one in the book, although you would expect the efficiency differences to be slight.

The results are as follows:

---

Choice of Fabric Softener				
Algorithm Search History				
Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
1	Start	100.0000	100.0000	Tab
1	End	100.0000		
Choice of Fabric Softener				
The OPTEX Procedure				
Class Level Information				
Class	Levels	Values		
x1	3	1	2	3
x2	3	1	2	3
x3	3	1	2	3
x4	3	1	2	3
Choice of Fabric Softener				
The OPTEX Procedure				
Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	100.0000	100.0000	100.0000	0.7071

---

The following step displays the design:

```
proc print data=design; run;
```

The design is as follows:

---

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

---

The macro found a perfect, orthogonal and balanced, 100%  $D$ -efficient design consisting of 4 three-level factors,  $x1$ - $x4$ . The levels are the integers 1 to 3. For this problem, the macro generated the design directly. For other problems, the macro might have to use a computerized search. See page 347 for more information about how the `%MktEx` macro works.

## Examining the Design

You should run basic checks on all designs, even orthogonal designs such as this one. You can use the `%MktEval` macro to display information about the design. The macro first displays a matrix of canonical correlations between the factors. We hope to see an identity matrix (a matrix of ones on the diagonal and zeros everywhere else) which means the design is orthogonal. Next, the macro displays all one-way frequencies for all attributes, all two-way frequencies, and all  $n$ -way frequencies (in this case four-way frequencies). We hope to see equal or at least nearly equal one-way and two-way frequencies, and we want to see that each combination occurs only once. The following step creates the design:

```
%mkteval(data=design)
```

The results are as follows:

---

Choice of Fabric Softener  
Canonical Correlations Between the Factors  
There are 0 Canonical Correlations Greater Than 0.316

	x1	x2	x3	x4
x1	1	0	0	0
x2	0	1	0	0
x3	0	0	1	0
x4	0	0	0	1

Choice of Fabric Softener  
Summary of Frequencies  
There are 0 Canonical Correlations Greater Than 0.316

Frequencies

x1	6 6 6
x2	6 6 6
x3	6 6 6
x4	6 6 6
x1 x2	2 2 2 2 2 2 2 2 2
x1 x3	2 2 2 2 2 2 2 2 2
x1 x4	2 2 2 2 2 2 2 2 2
x2 x3	2 2 2 2 2 2 2 2 2
x2 x4	2 2 2 2 2 2 2 2 2
x3 x4	2 2 2 2 2 2 2 2 2
N-Way	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

---

A *canonical correlation* is the maximum correlation between linear combinations of the coded factors (see page 101). All zeros off of the diagonal show that this design is orthogonal for main effects. If any off-diagonal canonical correlations are greater than 0.316 ( $r^2 > 0.1$ ), the macro lists them in a separate table. The last title line tells you that none of them is this large. The criterion  $r > 0.316$  or ( $r^2 > 0.1$ ) is purely arbitrary, and you can specify any other value you choose by using the `list=` option.

For nonorthogonal designs and designs with interactions, the canonical-correlation matrix is not a substitute for looking at the variance matrix (with `examine=v`, discussed on pages 351, 425, and 1058). It just provides a quick and more-compact picture of the correlations between the factors. The variance matrix is sensitive to the actual model specification and the actual coding. The canonical-correlation matrix just tells you if there is some correlation between the main effects. In this case, there are no correlations.

The equal one-way frequencies show you that this design is balanced. The equal two-way frequencies show you that this design is orthogonal. The  $n$ -way frequencies, all equal to one, show there are no duplicate profiles. This is a perfect design for a main-effects model.

You should always check the  $n$ -way frequencies to ensure that you do not have duplicates. For this situation for example, a 100%  $D$ -efficient design exists where each run appears twice. It consists of two copies of the fractional-factorial design  $3^4$  in 9 runs. When you get duplicates, specify `options=nodups` in the `%MktEx` macro, or sometimes you can just change the random number seed. Most designs do not have duplicates, so it is better to specify `options=nodups` only after you have found a design with duplicates. The no-duplicates constraint greatly slows down the algorithm.

The `%MktEval` macro produces a very compact summary of the design, hence some information, for example, the levels to which the frequencies correspond, is not shown. You can use the `print=freqs` option to get a less compact and more detailed display as follows:

```
%mkteval(data=design, print=freqs)
```

Portions of the results are as follows:

---

Choice of Fabric Softener						
Frequencies						
Effects	Frequency	x1	x2	x3	x4	
x1	6	1	.	.	.	
	6	2	.	.	.	
	6	3	.	.	.	
x2	6	.	1	.	.	
	6	.	2	.	.	
	6	.	3	.	.	
.						
.						
.						
x1 x2	2	1	1	.	.	
	2	1	2	.	.	
	2	1	3	.	.	
	2	2	1	.	.	
	2	2	2	.	.	
	2	2	3	.	.	
	2	3	1	.	.	
	2	3	2	.	.	
	2	3	3	.	.	
.						
.						
.						



x3	x4	2	.	.	1	1
		2	.	.	1	2
		2	.	.	1	3
		2	.	.	2	1
		2	.	.	2	2
		2	.	.	2	3
		2	.	.	3	1
		2	.	.	3	2
		2	.	.	3	3
N-Way		1	1	1	1	1
		1	1	1	2	2
		1	1	2	1	3
		1	1	2	3	1
		1	1	3	2	3
		1	1	3	3	2
		1	2	1	1	3
		1	2	1	3	1
		1	2	2	2	2
		1	2	2	3	3
		1	2	3	1	2
		1	2	3	2	1
		1	3	1	2	3
		1	3	1	3	2
		1	3	2	1	2
		1	3	2	2	1
		1	3	3	1	1
		1	3	3	3	3

## The Randomized Design and Postprocessing

The design we just looked at and examined is in the default output data set, **Design**. The **Design** data set is the last data set created by the **%MktEx** macro, so it is the data set that you see if you run **PROC PRINT** or the **%MktEval** macro without specifying the **data=** option. The **Design** data set is sorted, and often the first row consists entirely of ones. For these reasons, you should actually use the *randomized* design. In the randomized design, the choice sets are presented in a random order and the levels have been randomly reassigned. Neither of these operations affects the design *D*-efficiency, balance, or orthogonality, so the **%MktEval** results from the **Design** data set are valid, even if you ultimately use the **Randomized** data set. The macro automatically randomizes the design and stores the results in a data set called **Randomized**. The next steps assign formats and labels and store the results in a SAS data set **sasuser.SoftenerLinDes**. It is important to store the design in a permanent SAS data set or in some other permanent form so that it is available for analysis after the data are collected.

Every SAS data set has a two-level name of the form **libref.filename**. You can always reference a file with its two-level name. However, you can also use a one-level name, and then that data set is stored in temporary SAS data library with a libref of **Work**. Temporary data sets are deleted at

the end of your SAS session, so any data that must be saved needs to be stored in a permanent SAS data set. The libref called `sasuser` is automatically available for permanent storage in most SAS installations. Furthermore, you can make your own libref using a `libname` statement. You might wish to create a separate library for each project. The latter approach of using a `libname` statement is usually preferable, but for our purposes, mainly to avoid discussing issues of host-specific paths and file names, we use `sasuser`. See your BASE SAS documentation and SAS Companion for your operating system for more information about data libraries, `libref`, and `libname`. The following step creates a permanent SAS data set:

```
data sasuser.SoftenerLinDes;
  set randomized;
  format x1-x&mm1 price.;
  label x1 = 'Sploosh' x2 = 'Plumbbob' x3 = 'Platter' x4 = 'Moosey';
run;
```

The following step displays the final design:

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

The results are as follows:

---

Choice of Fabric Softener				
Efficient Design				
Obs	Sploosh	Plumbbob	Platter	Moosey
1	\$1.99	\$1.99	\$1.99	\$2.49
2	\$2.49	\$1.49	\$1.49	\$1.99
3	\$1.49	\$2.49	\$2.49	\$1.49
4	\$2.49	\$1.99	\$2.49	\$1.99
5	\$1.49	\$1.49	\$1.49	\$2.49
6	\$1.49	\$2.49	\$1.99	\$1.99
7	\$2.49	\$1.99	\$1.99	\$1.49
8	\$2.49	\$2.49	\$1.49	\$1.49
9	\$1.99	\$1.49	\$2.49	\$1.49
10	\$1.49	\$1.49	\$1.99	\$1.49
11	\$1.99	\$2.49	\$1.49	\$2.49
12	\$1.49	\$1.99	\$1.49	\$1.99
13	\$1.99	\$1.99	\$1.49	\$1.49
14	\$1.49	\$1.99	\$2.49	\$2.49
15	\$2.49	\$1.49	\$2.49	\$2.49
16	\$1.99	\$2.49	\$2.49	\$1.99
17	\$1.99	\$1.49	\$1.99	\$1.99
18	\$2.49	\$2.49	\$1.99	\$2.49

---

## From the Linear Arrangement to a Choice Design

The randomized design is now in a useful form for making the questionnaire, which is discussed in the next section. However, it is not in the final choice-design form that is needed for analysis and for the last evaluation that we should perform before collecting data. In this section, we convert our linear arrangement to a choice design and evaluate its goodness for a choice model.

Our linear arrangement, which we stored in a permanent SAS data set, `sasuser.SoftenerLinDes`, is arranged with one row per choice set. For analysis, we need a *choice design* with one row for each alternative of each choice set. We call the randomized design a *linear arrangement* (see page 67) because we used the `%MktEx` macro to create it optimizing  $D$ -efficiency for a linear model. We use the macro `%MktRoll` to “roll out” the linear arrangement into the choice design, which is in the proper form for analysis. First, we must create a data set that describes how the design is processed. We call this data set the *design key*.

In this example, we want a choice design with two factors, **Brand** and **Price**. **Brand** has levels *Sploosh*, *Plumbbob*, *Platter*, *Moosey*, and *Another*. **Price** has levels \$1.49, \$1.99, and \$2.49. **Brand** and **Price** are created by different processes. The **Price** attribute is constructed from the factors of the linear arrangement matrix. In contrast, there is no **Brand** factor in the linear arrangement. Each brand is a bin into which its factors are collected. The variable **Brand** is named in the `alt=` option of the `%MktRoll` macro as the alternative variable, so its values are read directly out of the `Key` data set. **Price** is not named in the `alt=` macro option, so its values in the `Key` data set are variable names from the linear arrangement data set. The values of **Price** in the final choice design are read from the named variables in the linear arrangement data set. The **Price** attribute in the choice design is created from the four linear arrangement factors (`x1` for *Sploosh*, `x2` for *Plumbbob*, `x3` for *Platter*, `x4` for *Moosey*, and no attribute for *Another*, the constant alternative). The `Key` data set is created in the next step. The **Brand** factor levels and the **Price** linear arrangement factors are stored in the `Key` data set as follows:

```

title2 'Key Data Set';

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

proc print; run;

```

The results are as follows:

---

Choice of Fabric Softener  
Key Data Set

Obs	Brand	Price
1	Sploosh	x1
2	Plumbbob	x2
3	Platter	x3
4	Moosey	x4
5	Another	

---

Note that the value of `Price` for alternative *Another* in the `Key` data set is blank (character missing). The period in the in-stream data set is simply a place holder, used with list input to read both character and numeric missing data. A period is not stored with the data. Next, we use the `%MktRoll` macro to process the design as follows:

```
%mktroll(design=sasuser.SoftenerLinDes, key=key, alt=brand,
         out=sasuser.SoftenerChDes)
```

The `%MktRoll` step processes the `design=sasuser.SoftenerLinDes` linear arrangement data set using the rules specified in the `key=key` data set, naming the `alt=brand` variable as the alternative name variable, and creating an output SAS data set called `out=sasuser.SoftenerChDes`, which contains the choice design. The input `design=sasuser.SoftenerLinDes` data set has 18 observations, one per choice set, and the output `out=sasuser.SoftenerChDes` data set has  $5 \times 18 = 90$  observations, one for each alternative of each choice set. The following step displays the first three observations of the linear arrangement data set:

```
title2 'Linear Arrangement (First 3 Sets)';

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

The results are as follows:

---

Choice of Fabric Softener  
Linear Arrangement (First 3 Sets)

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

---

These observations define the first three choice sets.

The following step displays those same observations, arrayed for analysis in the choice design data set:

```
title2 'Choice Design (First 3 Sets)';

proc print data=sasuser.SoftenerChDes(obs=15);
  format price price.;
  id set; by set;
run;
```

The results are as follows:

---

Choice of Fabric Softener  
Choice Design (First 3 Sets)

Set	Brand	Price
1	Sploosh	\$1.99
	Plumbbob	\$1.99
	Platter	\$1.99
	Moosey	\$2.49
	Another	\$1.99
2	Sploosh	\$2.49
	Plumbbob	\$1.49
	Platter	\$1.49
	Moosey	\$1.99
	Another	\$1.99
3	Sploosh	\$1.49
	Plumbbob	\$2.49
	Platter	\$2.49
	Moosey	\$1.49
	Another	\$1.99

---

The choice design data set has a choice set variable **Set**, an alternative name variable **Brand**, and a price variable **Price**. The prices come from the linear arrangement, and the price for *Another* is a constant \$1.99. Recall that the prices are assigned by the following format:

```
proc format;                                /* create a format for the price */
  value price 1 = '$1.49' 2 = '$1.99' 3 = '$2.49' . = '$1.99';
run;
```

## Testing the Design Before Data Collection

Collecting data is time consuming and expensive. It is always good practice to make sure that the design works with the most complicated model that we anticipate fitting. The following step evaluates the choice design:

```

title2 'Evaluate the Choice Design';

%choicEff(data=sasuser.SoftenerChDes,/* candidate set of choice sets      */
init=sasuser.SoftenerChDes(keep=set), /* select these sets            */
intiter=0, /* evaluate without internal iterations */
/* main effects with ref cell coding */
/* ref level for brand is 'Another' */
/* ref level for price is $1.99 */
model=class(brand price / zero='Another' '$1.99') /
lprefix=0 /* lpr=0 labels created from just levels*/
cprefix=0%str(;) /* cpr=0 names created from just levels */
format price price.,/* trick: format passed in with model */

nsets=&n, /* number of choice sets */
nalts=&m, /* number of alternatives */
beta=zero) /* assumed beta vector, Ho: b=0 */

```

The `%ChoiEff` macro has two uses. You can use it to search for an efficient choice design, or you can use it to evaluate a choice design including designs that are generated using other methods such as the `%MktEx` and `%MktRoll` macros. It is this latter use that is illustrated here.

The way you check a design like this is to first name it in the `data=` option. This is the candidate set that contains all of the choice sets that we will consider. In addition, the same design is named in the `init=` option. The full specification is `init=sasuser.SoftenerChDes(keep=set)`. Just the variable `Set` is kept. It is used to bring in just the indicated choice sets from the `data=` design, which in this case is all of them. The option `nsets=&n` specifies that there are `&n=18` choice sets, and `nalts=&m` specifies that there are `&m=5` alternatives. The option `beta=zero` specifies that we are assuming for design evaluation purposes the null hypothesis that all of the betas or part-worth utilities are zero. You can evaluate the design for other parameter vectors by specifying a list of numbers after `beta=`. This will change the variances and standard errors. We also specify `intiter=0` which specifies zero internal iterations. We use zero internal iterations when we want to evaluate an initial design, but not attempt to improve it. The `model=` option option specifies the model.

The model specification contains everything that appears in the TRANSREG procedure's `model` statement for coding the design. The specification `class(brand price / zero='Another' '$1.99')` names the `brand` and `price` variable as a classification variables and asks for coded variables for every level except `'Another'` for `brand` and `'$1.99'` for `price`. The levels `'Another'` and `'$1.99'` are the reference levels for the two attributes. In a  $p$ -level factor, there are at most  $p - 1$  nonzero parameters.

The `lprefix=0` option specifies that when labels are created for the binary variables, zero characters of the original variable name should be used as a prefix. This means that the labels are created only from the level values. For example, `'Sploosh'` and `'Plumbbob'` are created as labels not `'Brand Sploosh'` and `'Brand Plumbbob'`. The `cprefix=0` option specifies that when names are created for the binary variables, zero characters of the original variable name should be used as a prefix. This means that the names are created only from the level values. The `c` in `cprefix` stands for `class`.

The code following the `cprefix=` specification is a bit of a trick. The `%ChoiEff` macro generates a `model` statement for PROC TRANSREG using the specified value like this:

```
model &model;
```

By adding a semicolon, enclosed in `%str( )` and a format statement, we can send a format statement to the PROC TRANSREG coding step. The semicolon must be in the `%str( )` macro function so that it is passed into the macro and is not treated as the end of the macro specification. The following model specification adds these two statements to PROC TRANSREG in the `%ChoiceEff` macro:

```
model class(brand price / zero='Another' '$1.99') / lprefix=0 cprefix=0;
format price price.;
```

Alternatively, we could have just created a separate data set and added the format statement that way.

The results are as follows:

---

```

Choice of Fabric Softener
Evaluate the Choice Design

      n   Name      Beta   Label
-----
      1   Moosey      0     Moosey
      2   Platter      0     Platter
      3   Plumbbob     0     Plumbbob
      4   Sploosh      0     Sploosh
      5   _1_49        0     $1.49
      6   _2_49        0     $2.49

Choice of Fabric Softener
Evaluate the Choice Design

Design  Iteration  D-Efficiency      D-Error
-----
      1         0         2.34223 *      0.42694

Choice of Fabric Softener
Evaluate the Choice Design

Final Results

Design           1
Choice Sets      18
Alternatives     5
Parameters       6
Maximum Parameters 72
D-Efficiency     2.3422
D-Error          0.4269
```

Choice of Fabric Softener  
Evaluate the Choice Design

n	Variable Name	Label	Variance	DF	Standard Error
1	Moosey	Moosey	0.72917	1	0.85391
2	Platter	Platter	0.72917	1	0.85391
3	Plumbbob	Plumbbob	0.72917	1	0.85391
4	Sploosh	Sploosh	0.72917	1	0.85391
5	_1_49	\$1.49	0.52083	1	0.72169
6	_2_49	\$2.49	0.52083	1	0.72169
				==	
				6	

---

The first table provides the name, specified value, and label for each parameter. The second table is the iteration history. There is just one line in the table since zero internal iterations are requested. The third table summarizes the design. The first design has 18 choice sets, 5 alternatives, a  $D$ -efficiency of 2.3422 and a  $D$ -error of 0.4269.  $D$ -error =  $1 / D$ -efficiency. Note that for this evaluation,  $D$ -efficiency and  $D$ -error are computed on a scale with an unknown maximum, so unlike the values that come out of the `%MktEx` macro,  $D$ -efficiency is not on a percentage or zero to 100 scale in this set of results. Later in this example, we change that. For now, the  $D$ -efficiency is not what really interests us. We are most interested in the final table. It shows the names and labels for the parameters as well as their variances, standard errors, and  $df$ . We see that the parameters for all four brands have the same standard errors. Similarly, the standard errors for the two prices are the same. They are different for the two attributes since both have a different number of levels. In both sets, however, they are all approximately the same order of magnitude. Sometimes you might see wildly varying parameters. This is usually a sign of a problematic design, perhaps one with too few choice sets for the number of parameters. This design looks good.

It is a really good idea to perform this step before designing the questionnaire and collecting data. Data collection is expensive, so it is good to make sure that the design can be used for the most complicated model that you intend to fit. Much more is said about evaluating the standard errors in the later and more complicated examples.



You can also evaluate the choice design using the standardized orthogonal contrast coding as follows:

```

title2 'Evaluate the Choice Design';
title3 'Standardized Orthogonal Contrast Coding';

%choiceff(data=sasuser.SoftenerChDes,/* candidate set of choice sets      */
          init=sasuser.SoftenerChDes(keep=set), /* select these sets          */
          intiter=0, /* evaluate without internal iterations */
                                     /* model with stdz orthogonal coding */
                                     /* ref level for brand is 'Another'  */
                                     /* ref level for price is $1.99      */
          model=class(brand price / zero='Another' '$1.99' sta) /
          lprefix=0 /* lpr=0 labels created from just levels*/
          cprefix=0%str(;) /* cpr=0 names created from just levels */
          format price price.,/* trick: format passed in with model */

          nsets=&n, /* number of choice sets          */
          nalts=&m, /* number of alternatives          */
          options=relative, /* display relative D-efficiency */
          beta=zero) /* assumed beta vector, Ho: b=0   */

```

The `sta*` (short for `standorth`) option in the `model=` option is new with SAS 9.2 and requests a standardized orthogonal contrast coding. You must specify this coding if you want to see relative *D*-efficiency on a 0 to 100 scale. Relative *D*-efficiency is not displayed by default since it is only meaningful for a limited class of designs. You must request it with `options=relative`.

The last part of the output from the `%ChoiceEff` macro's evaluation of the design is as follows:

---

```

                Choice of Fabric Softener
                Evaluate the Choice Design
Standardized Orthogonal Contrast Coding

```

Final Results

Design	1
Choice Sets	18
Alternatives	5
Parameters	6
Maximum Parameters	72
D-Efficiency	15.5119
Relative D-Eff	86.1774
D-Error	0.0645
1 / Choice Sets	0.0556

---

\*This option is first available with SAS 9.2. It will not be recognized, and it will cause an error in earlier SAS releases.

Choice of Fabric Softener  
Evaluate the Choice Design  
Standardized Orthogonal Contrast Coding

9

n	Variable Name	Label	Variance	DF	Standard Error
1	Moosey	Moosey	0.055556	1	0.23570
2	Platter	Platter	0.055556	1	0.23570
3	Plumbbob	Plumbbob	0.055556	1	0.23570
4	Sploosh	Sploosh	0.055556	1	0.23570
5	_1_49	\$1.49	0.086806	1	0.29463
6	_2_49	\$2.49	0.086806	1	0.29463
				==	
				6	

In the first table, we see  $D$ -efficiency equals 15.5119 (as opposed to 2.3422 previous).  $D$ -error is always equal to  $1 / D$ -efficiency. If this were a perfect choice design like we can achieve for some generic designs, then  $D$ -efficiency would equal 18, and the number of choice sets and  $D$ -error and all of the variances would equal one over the number of choice sets (0.0556). Relative  $D$ -efficiency equals 100 times  $D$ -efficiency divided by the number of choice sets. It is 100 for perfect designs. In this case, relative  $D$ -efficiency = 86.1774, and all of the variances for the price parameters are larger than 0.556. Note the relative  $D$ -efficiency = 86.1774 provides a pessimistic view of the goodness of this design. This value is computed relative to the value you would get for an unrestricted design (one that does not have a constant alternative). 100% relative  $D$ -efficiency is not possible with a constant alternative and using the number of choice sets as a base line since all alternatives must vary optimally to achieve 100% efficiency. Furthermore, this design has five alternatives. There are five brands, one for each alternative, which is ideal. Hence, the variances for the brand parameters are at the minimum. However, there are only three prices. Since 3 does not evenly divide the 5 alternatives, there is no way we can achieve 100% efficiency by scaling  $D$ -efficiency relative to the number of choice sets. Hence, the variances for the price parameters are greater than their minimum. This design looks reasonable because the variances are not too far removed from one over the number of choice sets.

The `%ChoiceEff` macro displays the variances. You can display the full matrix of variances and covariances, which the `%ChoiceEff` macro stores in a SAS data set, as follows:

```
proc print data=bestcov label;
  title3 'Variance-Covariance Matrix';
  id __label;
  label __label = '00'x;
  var Moosey -- _2_49;
run;
```

The results are as follows:

---

Choice of Fabric Softener Evaluate the Choice Design Variance-Covariance Matrix						
	Moosey	Platter	Plumbbob	Sploosh	\$1.49	\$2.49
Moosey	0.055556	-0.000000	-0.000000	0.000000	0.000000	-0.000000
Platter	-0.000000	0.055556	0.000000	-0.000000	-0.000000	0.000000
Plumbbob	-0.000000	0.000000	0.055556	0.000000	-0.000000	0.000000
Sploosh	0.000000	-0.000000	0.000000	0.055556	0.000000	-0.000000
\$1.49	0.000000	-0.000000	-0.000000	0.000000	0.086806	-0.000000
\$2.49	-0.000000	0.000000	0.000000	-0.000000	-0.000000	0.086806

---

All of the covariances are zero, which is good. The larger variances are due to the within-choice-set imbalance and perhaps inefficiency in price.

There is one more test that should be run before a design is used. The following %MktDups macro step checks the design to see if any choice sets are duplicates of any other choice sets:

```
title2 'Evaluate the Choice Design, Check for Duplicates';

%mktdups(branded, data=sasuser.SoftenerChDes, nalts=&m, factors=brand price)
```

The results are as follows:

---

Design:	Branded
Factors:	brand price
	Brand
	Price
Duplicate Sets:	0

---

The first line of the table tells us that this is a branded design as opposed to a generic design (bundles of attributes with no brands). The second line tells us the factors as specified in the `factors=` option. These are followed by the actual variable names for the factors. The last line reports the number of duplicates. In this case, there are no duplicate choice sets. When there are duplicate choice sets, changing the random number seed might help. Changing other aspects of the design or the approach for making the design might also help.

## Evaluating the Design Relative to the Optimal Design

In the previous section, we saw that our design had a  $D$ -efficiency = 86.1774 relative to a design with no constant alternative restriction, which provided a pessimistic view of the goodness of this design. This problem is small enough, with only  $3^4 = 81$  possible choice sets that we can be reasonably confident

that the %ChoiceEff macro will find *the* optimal design given a candidate set of all possible choice sets. We can then use the *D*-efficiency of the optimal design in the relative *D*-efficiency computations for our design to measure the goodness of our design for this particular model. This is illustrated in this section. It proceeds very similarly to the steps shown previously, only this time, rather than giving the %ChoiceEff macro a set of choice sets to evaluate, it is given a set of candidate choice sets from which to construct a design. The following steps create a candidate set of all possible choice sets:

```
%mktex(3 ** 4, n=81)

data TestLinDes;
  set design;
  format x1-x&mm1 price.;
  label x1 = 'Sploosh' x2 = 'Plumbbob' x3 = 'Platter' x4 = 'Moosey';
  run;

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

%mktroll(design=TestLinDes, key=key, alt=brand, out=TestChDes)
```

The following step searches for an optimal design:

```
%choiceff(data=TestChDes,          /* candidate set of choice sets      */
           /* model with stdz orthogonal coding */
           /* ref level for brand is 'Another'   */
           /* ref level for price is $1.99      */
           model=class(brand price / zero='Another' '$1.99' sta) /
           lprefix=0                /* lpr=0 labels created from just levels*/
           cprefix=0%str(;)         /* cpr=0 names created from just levels */
           format price price.,/* trick: format passed in with model */

           seed=205,                /* random number seed                  */
           maxiter=50,              /* maximum number of designs to make   */
           nsets=&n,                /* number of choice sets               */
           nalts=&m,                /* number of alternatives               */
           options=relative,        /* display relative D-efficiency       */
           beta=zero)               /* assumed beta vector, Ho: b=0        */
```

Some of the results are as follows:

---

Choice of Fabric Softener  
Evaluate the Choice Design, Check for Duplicates

Final Results

Design	1
Choice Sets	18
Alternatives	5
Parameters	6
Maximum Parameters	72
D-Efficiency	16.4264
Relative D-Eff	91.2581
D-Error	0.0609
1 / Choice Sets	0.0556

Choice of Fabric Softener  
Evaluate the Choice Design, Check for Duplicates

n	Variable Name	Label	Variance	DF	Standard Error
1	Moosey	Moosey	0.055556	1	0.23570
2	Platter	Platter	0.055556	1	0.23570
3	Plumbbob	Plumbbob	0.055556	1	0.23570
4	Sploosh	Sploosh	0.055556	1	0.23570
5	_1_49	\$1.49	0.073099	1	0.27037
6	_2_49	\$2.49	0.073099	1	0.27037
				==	
				6	

---

The iteration histories for the 50 designs that were created (not shown) all have the same  $D$ -efficiency = 16.4264. With small problems like this one, this is a good sign that the optimal design was found.

The following step evaluates our design from the previous section using the  $D$ -efficiency from what is probably the optimal design in the `rscale=` option:

```
%choicereff(data=sasuser.SoftenerChDes,/* candidate set of choice sets      */
  init=sasuser.SoftenerChDes(keep=set), /* select these sets              */
  intiter=0, /* evaluate without internal iterations */
  rscale=16.4264, /* optimal D-efficiency          */
  /* model with stdz orthogonal coding */
  /* ref level for brand is 'Another' */
  /* ref level for price is $1.99    */
  model=class(price / zero='Another' '$1.99' sta) /
  lprefix=0 /* lpr=0 labels created from just levels*/
  cprefix=0%str(;) /* cpr=0 names created from just levels */
  format price price.,/* trick: format passed in with model */

  nsets=&n, /* number of choice sets          */
  nalts=&m, /* number of alternatives          */
  options=relative, /* display relative D-efficiency */
  beta=zero) /* assumed beta vector, Ho: b=0   */
```

Some of the results are as follows:

---

Choice of Fabric Softener  
Evaluate the Choice Design, Check for Duplicates

Final Results

Design	1
Choice Sets	18
Alternatives	5
Parameters	6
Maximum Parameters	72
D-Efficiency	15.5119
Relative D-Eff	94.4329
D-Error	0.0645
1 / Choice Sets	0.0556

---

This shows that relative to what is probably the optimal design, our design is 94.4329%  $D$ -efficient. Of course, usually we cannot know this. We will continue with this example using the original design as planned, although for this particular model and beta specification, the design we just found is a little better. Of course, we do not know the true beta, although we are sure it is *not* our null hypothesis zero vector, so in practice, we do not in fact know which design is most optimal. Our original design is good in that all of the prices are balanced and orthogonal (at least outside the constant alternative) across all alternatives. In contrast, the design we just found is good in a different way since it was optimized for this model and beta specification.

## Generating the Questionnaire

A questionnaire based on the design is produced using the following DATA step:

```

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.SoftenerLinDes;

  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;

```

The following statement creates a constant array:

```

array brands[&m] $ _temporary_ ('Sploosh' 'Plumbbob' 'Platter'
                                'Moosey' 'Another')

```

The array reference `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.

In the interest of space, only the first two choice sets are shown as follows:

---

Subject: \_\_\_\_\_

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

- 1) Sploosh brand at \$1.99.
- 2) Plumbbob brand at \$1.99.
- 3) Platter brand at \$1.99.
- 4) Moosey brand at \$2.49.
- 5) Another brand at \$1.99.

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

- 1) Sploosh brand at \$2.49.
  - 2) Plumbbob brand at \$1.49.
  - 3) Platter brand at \$1.49.
  - 4) Moosey brand at \$1.99.
  - 5) Another brand at \$1.99.
- 

The questionnaire is printed, copied, and the data are collected.

In practice, data collection is typically much more elaborate than this. It might involve art work or photographs, and the choice sets might be presented and the data might be collected through personal interview or over the Web. However the choice sets are presented and the data are collected, the essential elements remain the same. Subjects are shown a set of alternatives and are asked to make a choice, then they go on to the next set.

## Entering the Data

The data consist of a subject number followed by 18 integers in the range 1 to 5. These are the alternatives that are chosen for each choice set. For example, the first subject chose alternative 3 (*Platter* brand at \$1.99) in the first choice set, alternative 3 (*Platter* brand at \$1.49) in the second choice set, and so on. In the interest of space, data from three subjects appear on one line. The data are read in the following step:



```

title 'Choice of Fabric Softener';

data results;                                /* read choice data set */
  input Subj (choose1-choose&n) (1.) @@;
  datalines;
  1 334533434233312433  2 334213442433333325  3 333333333333313333
  4 334431444434412453  5 335431434233512423  6 334433434433312433
  7 334433434433322433  8 334433434433412423  9 334433332353312433
 10 325233435233332433 11 334233434433313333 12 334331334433312353
 13 534333334333312323 14 134421444433412423 15 334333435433312335
 16 334433435333315333 17 534333432453312423 18 334435544433412543
 19 334333335433313433 20 331431434233315533 21 334353534433512323
 22 334333452233312523 23 334333332333312433 24 525221444233322423
 25 354333434433312333 26 334435545233312323 27 334353534233352323
 28 33433333233332333 29 334433534335352423 30 334453434533313433
 31 354333334333312433 32 354331332233332423 33 334424432353312325
 34 334433434433312433 35 334551444453412325 36 334234534433312433
 37 334431434433512423 38 354333334433352523 39 334351334333312533
 40 324433334433412323 41 334433444433412443 42 334433434433312423
 43 334434454433332423 44 334433434233312423 45 334451544433412424
 46 434431435433512423 47 524434534433412433 48 335453334433322453
 49 334533434133312433 50 334433332333312423
;

```

## Processing the Data

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

```

proc format;
  value price 1 = '$1.49' 2 = '$1.99' 3 = '$2.49' . = '$1.99';
run;

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

```

This step reads the `design=sasuser.SoftenerChDes` choice design and the `data=results` data set and creates the `out=res2` output data set. The data are from an experiment with `nsets=&n` choice sets, `nalts=&m` alternatives, with variables `setvars=choose1-choose&n` containing the numbers of the chosen alternatives. The following step displays the first 15 observations:

```

title2 'Choice Design and Data (First 3 Sets)';

proc print data=res2(obs=15);
  id subj set; by subj set;
run;

```

The results are as follows:

---

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

Subj	Set	Brand	Price	c
1	1	Sploosh	2	2
		Plumbbob	2	2
		Platter	2	1
		Moosey	3	2
		Another	.	2
1	2	Sploosh	3	2
		Plumbbob	1	2
		Platter	1	1
		Moosey	2	2
		Another	.	2
1	3	Sploosh	1	2
		Plumbbob	3	2
		Platter	3	2
		Moosey	1	1
		Another	.	2

---

The data set contains the subject ID variable `Subj` from the `data=results` data set, the `Set`, `Brand`, and `Price` variables from the `design=sasuser.SoftenerChDes` data set, and the variable `c`, which indicates which alternative is chosen. The variable `c` contains: 1 for first choice and 2 for second or subsequent choice. This subject chose the third alternative, *Platter* in the first choice set, *Platter* in the second, and *Moosey* in the third. This data set has 4500 observations: 50 subjects times 18 choice sets times 5 alternatives.

Since we did not specify a format, we see in the design the raw design values for `Price`: 1, 2, 3 and missing for the constant alternative. If we were going to treat `Price` as a categorical variable for analysis, this would be fine. We would simply assign our price format to `Price` and designate it as a `class` variable. However, in this analysis we are going to treat price as quantitative and use the actual prices in the analysis. Hence, we must convert our design values from 1, 2, 3, and . to 1.49, 1.99, 2.49, and 1.99. We cannot do this by simply assigning a format. Formats create character strings that are displayed in place of the original value. We need to convert a numeric variable from one set of numeric values to another. We could use `if` and assignment statements. We could also use the `%MktLab` macro, which is used in later examples. However, instead we can use the `put` function to write the formatted value into a character string, then we read it back using a dollar format and the `input` function. For example, the expression `put(price, price.)` converts a number, say 2, into a string (in this case '\$1.99'), then the `input` function reads the string and converts it to a numeric 1.99. The following step also assigns a label to the variable `Price`:

```

data res3; /* Create a numeric actual price */
  set res2;
  price = input(put(price, price.), dollar5.);
  label price = 'Price';
run;

```

## Binary Coding

One more thing must be done to these data before they can be analyzed. The factors must be coded. In this example, we use a *binary* or zero-one coding for the brand effect. This can be done with PROC TRANSREG as follows:

```

proc transreg design=5000 data=res3 nozeroconstant norestoremissing;
  model class(brand / zero=none order=data)
    identity(price) / lprefix=0;
  output out=coded(drop=_type_ _name_ intercept);
  id subj set c;
run;

```

The `design` option specifies that no model is fit; the procedure is just being used to code a design. When `design` is specified, dependent variables are not required. Optionally, `design` can be followed by “=*n*” where *n* is the number of observations to process at one time. By default, PROC TRANSREG codes all observations in one big group. For very large data sets, this can consume large amounts of memory and time. Processing blocks of smaller numbers of observations is more computationally efficient. The option `design=5000` processes observations in blocks of 5000. For smaller computers, try something like `design=1000`.

The `nozeroconstant` and `norestoremissing` options are not necessary for this example, but they are included here, because sometimes they are very helpful in coding choice models. The `nozeroconstant` option specifies that if the coding creates a constant variable, it should not be zeroed. The `nozeroconstant` option should always be specified when you specify `design=n` because the last group of observations might be small and might contain constant variables. The `nozeroconstant` option is also important if you do something like coding by `subj set` because sometimes an attribute is constant within a choice set. The `norestoremissing` option specifies that missing values should not be restored when the `out=` data set is created. By default, the coded `class` variable contains a row of missing values for observations in which the `class` variable is missing. When you specify the `norestoremissing` option, these observations contain a row of zeros instead. This option is useful when there is a constant alternative indicated by missing values. Both of these options, like almost all options in PROC TRANSREG, can be abbreviated to three characters (`noz` and `nor`).

The `model` statement names the variables to code and provides information about how they should be coded. The specification `class(brand / ...)` specifies that the variable `Brand` is a classification variable and requests a binary coding. The `zero=none` option creates binary variables for all categories. In contrast, by default, a binary variable is not created for the last category—the parameter for the last category is a structural zero. The `zero=none` option is used when there are no structural zeros or when you want to see the structural zeros in the multinomial logit parameter estimates table. See page 78 for more information about the `zero=` option. The `order=data` option sorts the levels into the order that they are first encountered in the data set. Alternatively, the levels could be sorted based on the formatted or unformatted values. The specification `identity(price)` specifies that `Price` is a

quantitative attribute that should be analyzed as is (not expanded into indicator variables).

The `lprefix=0` option specifies that when labels are created for the binary variables, zero characters of the original variable name should be used as a prefix.

An output statement names the output data set and drops variables that are not needed. These variables do not have to be dropped. However, since they are variable names that are often found in special data set types, PROC PHREG displays warnings when it finds them. Dropping the variables prevents the warnings. Finally, the `id` statement names the additional variables that we want copied from the input to the output data set. The next steps display the first three coded choice sets:

```
proc print data=coded(obs=15) label;
  title2 'First 15 Observations of Analysis Data Set';
  id subj set c;
  by subj set;
  run;
```

The results are as follows:

---

Choice of Fabric Softener									
First 15 Observations of Analysis Data Set									
Subj	Set	c	Sploosh	Plumbbob	Platter	Moosey	Another	Price	Brand
1	1	2	1	0	0	0	0	1.99	Sploosh
		2	0	1	0	0	0	1.99	Plumbbob
		1	0	0	1	0	0	1.99	Platter
		2	0	0	0	1	0	2.49	Moosey
		2	0	0	0	0	1	1.99	Another
1	2	2	1	0	0	0	0	2.49	Sploosh
		2	0	1	0	0	0	1.49	Plumbbob
		1	0	0	1	0	0	1.49	Platter
		2	0	0	0	1	0	1.99	Moosey
		2	0	0	0	0	1	1.99	Another
1	3	2	1	0	0	0	0	1.49	Sploosh
		2	0	1	0	0	0	2.49	Plumbbob
		2	0	0	1	0	0	2.49	Platter
		1	0	0	0	1	0	1.49	Moosey
		2	0	0	0	0	1	1.99	Another

---

## Fitting the Multinomial Logit Model

The next step fits the discrete choice, multinomial logit model:

```
proc phreg data=coded outest=betas brief;
  title2 'Discrete Choice Model';
  model c*c(2) = &_trgind / ties=breslow;
  strata subj set;
run;
```

The `brief` option requests a brief summary for the strata. As with the candy example, `c*c(2)` designates the chosen and unchosen alternatives in the `model` statement. We specify the `&_trgind` macro variable for the `model` statement independent variable list. PROC TRANSREG automatically creates this macro variable. It contains the list of coded independent variables generated by the procedure. This is so you do not have to figure out what names TRANSREG created and specify them. In this case, PROC TRANSREG sets `&_trgind` to contain the following list:

```
BrandSploosh BrandPlumbbob BrandPlatter BrandMoosey BrandAnother Price
```

The `ties=breslow` option specifies a PROC PHREG model that has the same likelihood as the multinomial logit model for discrete choice. The `strata` statement specifies that the combinations of `Set` and `Subj` indicate the choice sets. This data set has 4500 observations consisting of  $18 \times 50 = 900$  strata and five observations per stratum.

Each subject rated 18 choice sets, but the multinomial logit model assumes each stratum is independent. That is, the multinomial logit model assumes each person makes only one choice. The option of collecting only one datum from each subject is too expensive to consider for many problems, so multiple choices are collected from each subject, and the repeated measures aspect of the problem is ignored. This practice is typical, and it usually works well, because the parameter estimates are still unbiased.

## Multinomial Logit Model Results

Recall that we specified `%phchoice(on)` on page 287 to customize the output from PROC PHREG. The results are as follows:

---

```

Choice of Fabric Softener
Discrete Choice Model

The PHREG Procedure

Model Information

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

Number of Observations Read 4500  
 Number of Observations Used 4500

Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Pattern	Number of Choices	Number of Alternatives	Chosen Alternatives	Not Chosen
1	900	5	1	4

Convergence Status

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

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	2896.988	1439.457
AIC	2896.988	1449.457
SBC	2896.988	1473.469

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	1457.5310	5	<.0001
Score	1299.7889	5	<.0001
Wald	635.9093	5	<.0001

Choice of Fabric Softener  
 Discrete Choice Model

The PHREG Procedure

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Sploosh	1	-1.30565	0.21097	38.3017	<.0001
Plumbbob	1	-0.49090	0.18035	7.4091	0.0065
Platter	1	2.08485	0.14592	204.1402	<.0001
Moosey	1	0.62183	0.15503	16.0884	<.0001
Another	0	0	.	.	.
Price	1	-4.60150	0.21608	453.5054	<.0001

---

The procedure output begins with information about the data set, variables, options, and number of observations read. This is followed by information about the 900 strata. Since the `brief` option is specified, this table contains one row for each stratum pattern. In contrast, the default table has 900 rows, one for each choice set and subject combination. Each subject and choice set combination consists of a total of five observations, one that is chosen and four that are not chosen. This pattern is observed 900 times. This table provides a check on data entry. Unless we have an availability or allocation study (page 524) or a nonconstant number of alternatives in different choice sets, we expect to see one pattern of results where one of the  $m$  alternatives is chosen for each choice set. If you do not observe this for a study like this, there is probably a mistake in the data entry or processing.

The most to least preferred brands are: *Platter*, *Moosey*, *Another*, *Plumbbob*, and *Sploosh*. Increases in price have a negative utility. For example, the predicted utility of *Platter* brand at \$1.99 is  $\mathbf{x}_i\boldsymbol{\beta}$  which is  $(0 \ 0 \ 1 \ 0 \ 0 \ \$1.99) \ (-1.31 \ -0.49 \ 2.08 \ 0.62 \ 0 \ -4.60)' = 2.08 + 1.99 \times -4.60 = -7.07$ . Since `Price` is analyzed as a quantitative attribute, we can see for example that the utility of *Platter* at \$1.89, which is not in any choice set, is  $2.08 + 1.89 \times -4.60 = -6.61$ , which is a  $\$0.10 \times 4.60 = 0.46$  increase in utility.

## Probability of Choice

These next steps compute the expected probability that each alternative is chosen within each choice set. These steps 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 mirror true market share.

First, PROC SCORE is used to compute the predicted utility for each alternative as follows:

```
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 are 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 is 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,  $\exp(\mathbf{x}_j\hat{\beta})$  is summed for each choice set as follows:

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

Finally, each  $\mathbf{x}_j\hat{\beta}$  is divided by  $\sum_{j=1}^m \mathbf{x}_j\hat{\beta}$  in the following step:

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

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

The results for the first three choice sets are as follows:

---

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

Set	Price	Brand	p
1	1.99	Sploosh	0.02680
	1.99	Plumbbob	0.06052
	1.99	Platter	0.79535
	2.49	Moosey	0.01845
	1.99	Another	0.09888
2	2.49	Sploosh	0.00030
	1.49	Plumbbob	0.06843
	1.49	Platter	0.89921
	1.99	Moosey	0.02086
	1.99	Another	0.01120
3	1.49	Sploosh	0.11679
	2.49	Plumbbob	0.00265
	2.49	Platter	0.03479
	1.49	Moosey	0.80260
	1.99	Another	0.04318

---



## Custom Questionnaires

In this part of the example, a custom questionnaire is displayed 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 displayed last. If you have no interest in custom questionnaires, you can skip ahead to page 339.

First, the macro variable `&forms` is created. It contains the number of separate questionnaires (or forms or subjects, in this case 50). We can use the `%MktEx` macro to create a data set with one observation for each alternative of each choice set for each person. The specification `%mktex(&forms &n &mm1, n=&forms * &n * &mm1)` is `%mktex(50 18 4, n=50 * 18 * 4)` and creates a  $50 \times 18 \times 4$  full-factorial design. Note that the `n=` specification allows expressions. The macro `%MktLab` is then used to assign the variable names `Form`, `Set`, and `Alt` instead of the default `x1 - x3`. The data set is sorted by `Form`. Within `Form`, the choice sets are sorted into a random order, and within choice set, the alternatives are sorted into a random order. The 72 observations for each choice set contain 18 blocks of 4 observations—one block per choice set in a random order and the 4 alternatives within each choice set, again in a random order. Note that we store these in a permanent SAS data set so they are available after the data are collected. See page 309 for more information about permanent SAS data sets. The following steps create the design:

```
%let forms = 50;
title2 'Create 50 Custom Questionnaires';

*---Make the design---;
%mktex(&forms &n &mm1, n=&forms * &n * &mm1)

*---Assign Factor Names---;
%mktlab(data=design, vars=Form Set Alt)

*---Set up for Random Ordering---;
data sasuser.orders;
  set final;
  by form set;
  retain r1;
  if first.set then r1 = uniform(17);
  r2 = uniform(17);
run;

*---Random Sort---;
proc sort out=sasuser.orders(drop=r:); by form r1 r2; run;

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

The first 16 observations in this data set are as follows:

---

Choice of Fabric Softener  
Create 50 Custom Questionnaires

Obs	Form	Set	Alt
1	1	4	3
2	1	4	1
3	1	4	2
4	1	4	4
5	1	8	2
6	1	8	3
7	1	8	1
8	1	8	4
9	1	16	1
10	1	16	2
11	1	16	3
12	1	16	4
13	1	1	3
14	1	1	1
15	1	1	4
16	1	1	2

---

The data set is transposed, so the resulting data set contains  $50 \times 18 = 900$  observations, one per subject per choice set. The alternatives are in the variables Col1-Col14. The first 18 observations, which contain the ordering of the choice sets for the first subject, are displayed by the following steps:

```
proc transpose data=sasuser.orders out=sasuser.orders(drop=_name_);
  by form notsorted set;
run;

proc print data=sasuser.orders(obs=18);
run;
```

The results are as follows:

---

Choice of Fabric Softener  
Create 50 Custom Questionnaires

Obs	Form	Set	COL1	COL2	COL3	COL4
1	1	4	3	1	2	4
2	1	8	2	3	1	4
3	1	16	1	2	3	4
4	1	1	3	1	4	2
5	1	6	2	4	1	3
6	1	7	4	1	3	2

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

---

The following DATA step displays the 50 custom questionnaires:

```

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.SoftenerLinDes 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;
    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.SoftenerLinDes` using the `point=` option in the `set` statement. The order of the alternatives is in the `c` array and variables `col1-col&mm1` from the `sasuser.Orders` data set. In the first observation of `sasuser.Orders`, `Set=4`, `Col1=3`, `Col2=1`, `Col3=2`, and `Col4=4`. The first brand, is `c[brnds] = c[1] = col1 = 3`, so `brands[c[brnds]] = brands[c[1]] = brands[3] = 'Platter'`, and the price, from observation `Set=4` of `sasuser.SoftenerLinDes`, is `x[c[brnds]] = x[3] = $2.49`. The second brand, is `c[brnds] = c[2] = col2 = 1`, so `brands[c[brnds]] = brands[c[2]] = brands[1] = 'Sploosh'`, and the price, from observation `Set=4` of `sasuser.SoftenerLinDes`, is `x[c[brnds]] = x[1] = $2.49`.

In the interest of space, only the first two choice sets are displayed. Note that the subject number is displayed on the form. This information is needed to restore all data to the original order. The first two choice sets are as follows:

Subject: 1

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

- 1) Platter brand at \$2.49.
- 2) Sploosh brand at \$2.49.
- 3) Plumbbob brand at \$1.99.
- 4) Moosey brand at \$1.99.
- 5) Another brand at \$1.99.

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

- 1) Plumbbob brand at \$2.49.
- 2) Platter brand at \$1.49.
- 3) Sploosh brand at \$2.49.
- 4) Moosey brand at \$1.49.
- 5) Another brand at \$1.99.

## Processing the Data for Custom Questionnaires

The data are entered next. (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 as follows:

```

title 'Choice of Fabric Softener';

data results;                                /* read choice data set */
  input Subj (choose1-choose&n) (1.) @@;
  datalines;
  1 524141141211421241  2 532234223321321311  3 223413221434144231
  4 424413322222544331  5 123324312534444533  6 233114423441143321
  7 123243224422433312  8 312432241121112412  9 315432222144111124
 10 511432445343442414 11 331244123342421432 12 323234114312123245
 13 312313434224435334 14 143433332142334114 15 234423133531441145
 16 425441421454434414 17 234431535341441432 18 235224352241523311
 19 134331342432542243 20 335331253334232433 21 513453254214134224
 22 212241213544214125 23 133444341431414432 24 453424142151142322
 25 324424431252444221 26 244145452131443415 27 553254131423323121
 28 233423242432231424 29 322454324541433543 30 323433433135133542
 31 412422434342513222 32 243144343352123213 33 441113141133454445
 34 131114113312342312 35 325222444355122522 36 342133254432124342
 37 511322324114234222 38 522153113442344541 39 211542232314512412
 40 244432222212213211 41 241411341323123213 42 314334342111232114
 43 422351321313343332 44 124243444234124432 45 141251113314352121
 46 414215225442424413 47 333452434454311222 48 334325341342552344
 49 335124122444243112 50 244412331342433332
;

```

The data set is transposed, and the original order is restored as follows:

```

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[&mmm1] col1-col&mmm1;
  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 is 5, which is the *Another* constant alternative. Since the first subject saw the fourth choice set first, the fourth data value for the first subject in the processed data set has a value of 5. The choice in the second choice set for the first subject is 2, and the second alternative the subject saw is *Platter*. The data

set `sasuser.Orders` shows in the second observation that this choice of 2 corresponds to the third (original) alternative (in the second column variable, `Col2 = 3`) of choice set `Set= 8`. In the original ordering, *Platter* is the third alternative. Hence the eighth data value in the processed data set has a value of 3. The following DATA step writes out the data after the original order has been restored:

```
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;
```

The results are as follows:

---

1	334533434233312433	2	33421344243333325	3	33333333333313333
4	334431444434412453	5	335431434233512423	6	334433434433312433
7	334433434433322433	8	334433434433412423	9	334433332353312433
10	32523343523332433	11	334233434433313333	12	334331334433312353
13	534333334333312323	14	134421444433412423	15	334333435433312335
16	334433435333315333	17	534333432453312423	18	334435544433412543
19	334333335433313433	20	331431434233315533	21	334353534433512323
22	334333452233312523	23	334333332333312433	24	525221444233322423
25	354333434433312333	26	334435545233312323	27	334353534233352323
28	33433333233332333	29	334433534335352423	30	334453434533313433
31	354333334333312433	32	35433133223332423	33	334424432353312325
34	334433434433312433	35	334551444453412325	36	334234534433312433
37	334431434433512423	38	354333334433352523	39	334351334333312533
40	324433334433412323	41	334433444433412443	42	334433434433312423
43	33443445443332423	44	334433434233312423	45	334451544433412424
46	434431435433512423	47	524434534433412433	48	335453334433322453
49	334533434133312433	50	334433332333312423		

---

The results match the data on page 325.

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

## Vacation Example

This example illustrates the design and analysis for a larger choice experiment. We discuss designing a choice experiment, evaluating the design, generating the questionnaire, processing the data, binary coding, generic attributes, quantitative price effects, quadratic price effects, effects coding, alternative-specific effects, analysis, and interpretation of the results. In this example, a researcher is interested in studying choice of vacation destinations. There are five destinations (alternatives) of interest: Hawaii, Alaska, Mexico, California, and Maine. Two summaries of the design are displayed next, one with factors first grouped by attribute and one grouped by destination:

Factor	Destination	Attribute	Levels
X1	Hawaii	Accommodations	Cabin, Bed & Breakfast, Hotel
X2	Alaska	Accommodations	Cabin, Bed & Breakfast, Hotel
X3	Mexico	Accommodations	Cabin, Bed & Breakfast, Hotel
X4	California	Accommodations	Cabin, Bed & Breakfast, Hotel
X5	Maine	Accommodations	Cabin, Bed & Breakfast, Hotel
X6	Hawaii	Scenery	Mountains, Lake, Beach
X7	Alaska	Scenery	Mountains, Lake, Beach
X8	Mexico	Scenery	Mountains, Lake, Beach
X9	California	Scenery	Mountains, Lake, Beach
X10	Maine	Scenery	Mountains, Lake, Beach
X11	Hawaii	Price	\$999, \$1249, \$1499
X12	Alaska	Price	\$999, \$1249, \$1499
X13	Mexico	Price	\$999, \$1249, \$1499
X14	California	Price	\$999, \$1249, \$1499
X15	Maine	Price	\$999, \$1249, \$1499

Factor	Destination	Attribute	Levels
X1	Hawaii	Accommodations	Cabin, Bed & Breakfast, Hotel
X6		Scenery	Mountains, Lake, Beach
X11		Price	\$999, \$1249, \$1499
X2	Alaska	Accommodations	Cabin, Bed & Breakfast, Hotel
X7		Scenery	Mountains, Lake, Beach
X12		Price	\$999, \$1249, \$1499
X3	Mexico	Accommodations	Cabin, Bed & Breakfast, Hotel
X8		Scenery	Mountains, Lake, Beach
X13		Price	\$999, \$1249, \$1499
X4	California	Accommodations	Cabin, Bed & Breakfast, Hotel
X9		Scenery	Mountains, Lake, Beach
X14		Price	\$999, \$1249, \$1499
X5	Maine	Accommodations	Cabin, Bed & Breakfast, Hotel
X10		Scenery	Mountains, Lake, Beach
X15		Price	\$999, \$1249, \$1499

Each alternative is composed of three factors: package cost (\$999, \$1,249, \$1,499), scenery (mountains, lake, beach), and accommodations (cabin, bed & breakfast, and hotel). There are five destinations, each with three attributes, for a total of 15 factors. This problem requires a design with 15 three-level factors, denoted  $3^{15}$ . Each row of the design matrix contains the description of the five alternatives in one choice set. Note that the levels do not have to be the same for all destinations. For example, the cost for Hawaii and Alaska could be different from the other destinations. However, for this example, each destination has the same attributes.

## Set Up

We can use the `%MktRuns` autocall macro to suggest design sizes. (All of the autocall macros used in this book are documented starting on page 803.) To use this macro, you specify the number of levels for each of the factors. With 15 attributes each with three prices, you specify fifteen 3's (`3 3 3 3 3 3 3 3 3 3 3 3 3 3 3`), or you can use the more compact syntax of `3 ** 15` as follows:

```
title 'Vacation Example';

%mktruns(3 ** 15)
```

The results are as follows:

---

```

                Vacation Example
                Design Summary
                Number of
                Levels      Frequency
                3           15
                Vacation Example
Saturated      = 31
Full Factorial = 14,348,907
Some Reasonable Design Sizes      Violations      Cannot Be
                                     Divided By
                36                   0
                45                   0
                54 *                  0
                63                   0
                72 *                  0
```



33	105	9
39	105	9
42	105	9
48	105	9
51	105	9
31 S	120	3 9

\* - 100% Efficient design can be made with the MktEx macro.  
 S - Saturated Design - The smallest design that can be made.  
 Note that the saturated design is not one of the recommended designs for this problem. It is shown to provide some context for the recommended sizes.

## Vacation Example

n	Design	Reference
54	2 ** 1 3 ** 25	Orthogonal Array
54	2 ** 1 3 ** 21 9 ** 1	Orthogonal Array
54	3 ** 24 6 ** 1	Orthogonal Array
54	3 ** 20 6 ** 1 9 ** 1	Orthogonal Array
54	3 ** 18 18 ** 1	Orthogonal Array
72	2 ** 23 3 ** 24	Orthogonal Array
72	2 ** 22 3 ** 20 6 ** 1	Orthogonal Array
72	2 ** 21 3 ** 16 6 ** 2	Orthogonal Array
72	2 ** 20 3 ** 24 4 ** 1	Orthogonal Array
72	2 ** 19 3 ** 20 4 ** 1 6 ** 1	Orthogonal Array
72	2 ** 18 3 ** 16 4 ** 1 6 ** 2	Orthogonal Array
72	2 ** 16 3 ** 25	Orthogonal Array
72	2 ** 15 3 ** 21 6 ** 1	Orthogonal Array
72	2 ** 14 3 ** 24 6 ** 1	Orthogonal Array
72	2 ** 14 3 ** 17 6 ** 2	Orthogonal Array
72	2 ** 13 3 ** 25 4 ** 1	Orthogonal Array
72	2 ** 13 3 ** 20 6 ** 2	Orthogonal Array
72	2 ** 12 3 ** 24 12 ** 1	Orthogonal Array
72	2 ** 12 3 ** 21 4 ** 1 6 ** 1	Orthogonal Array
72	2 ** 12 3 ** 16 6 ** 3	Orthogonal Array
72	2 ** 11 3 ** 24 4 ** 1 6 ** 1	Orthogonal Array
72	2 ** 11 3 ** 20 6 ** 1 12 ** 1	Orthogonal Array
72	2 ** 11 3 ** 17 4 ** 1 6 ** 2	Orthogonal Array
72	2 ** 10 3 ** 20 4 ** 1 6 ** 2	Orthogonal Array
72	2 ** 10 3 ** 16 6 ** 2 12 ** 1	Orthogonal Array
72	2 ** 9 3 ** 16 4 ** 1 6 ** 3	Orthogonal Array
72	3 ** 25 8 ** 1	Orthogonal Array
72	3 ** 24 24 ** 1	Orthogonal Array

---

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

In this design, there are  $15 \times (3 - 1) + 1 = 31$  parameters, so at least 31 choice sets must be created. With all three-level factors, the number of choice sets in all orthogonal and balanced designs must be divisible by  $3 \times 3 = 9$ . Hence, optimal designs for this problem have at least 36 choice sets (the smallest number  $\geq 31$  and divisible by 9). Note, however, that zero violations does not guarantee that a 100%  $D$ -efficient design exists. It just means that 100%  $D$ -efficiency is not precluded by unequal cell frequencies. In fact, the %MktEx orthogonal design catalog does not include orthogonal designs for this problem in 36, 45, and 63 runs (because they do not exist).

Thirty-six is a good design size (2 blocks of size 18) as is 54 (3 blocks of size 18). Fifty-four is probably the best choice, and that is what we recommend for this study. However, we instead create a  $D$ -efficient experimental design with 36 choice sets using the %MktEx macro. In practice, with more difficult designs, an orthogonal design is not available, and using 36 choice sets lets us see an example of using the %Mkt family of macros to get a nonorthogonal design.

We can see what orthogonal designs with three-level factors are available in 36 runs as follows. The %MktOrth macro creates a data set with information about the orthogonal designs that the %MktEx macro knows how to make. This macro produces a data set called MktDesLev that contains variables **n**, the number of runs; **Design**, a description of the design; and **Reference**, which contains the type of the design. In addition, there are variables: **x1**, the number of 1-level factors (which is always zero); **x2**, the number of 2-level factors; **x3**, the number of 3-level factors; and so on. The following steps use %MktOrth to only output **n=36** run designs and sort this list so that designs with the most three-level factors are displayed first:

```
%mktorth(range=n=36)

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

proc print; run;
```

The results are as follows:

---

Vacation Example

Obs	n	Design	Reference
1	36	2 ** 4 3 ** 13	Orthogonal Array
2	36	3 ** 13 4 ** 1	Orthogonal Array
3	36	2 ** 11 3 ** 12	Orthogonal Array
4	36	2 ** 2 3 ** 12 6 ** 1	Orthogonal Array
5	36	3 ** 12 12 ** 1	Orthogonal Array
6	36	2 ** 3 3 ** 9 6 ** 1	Orthogonal Array
7	36	2 ** 10 3 ** 8 6 ** 1	Orthogonal Array
8	36	2 ** 1 3 ** 8 6 ** 2	Orthogonal Array
9	36	3 ** 7 6 ** 3	Orthogonal Array
10	36	2 ** 2 3 ** 5 6 ** 2	Orthogonal Array

11	36	2 **	16	3 **	4			Orthogonal Array
12	36	2 **	9	3 **	4	6 **	2	Orthogonal Array
13	36	2 **	1	3 **	3	6 **	3	Orthogonal Array
14	36	2 **	20	3 **	2			Orthogonal Array
15	36	2 **	13	3 **	2	6 **	1	Orthogonal Array
16	36	2 **	3	3 **	2	6 **	3	Orthogonal Array
17	36	2 **	27	3 **	1			Orthogonal Array
18	36	2 **	18	3 **	1	6 **	1	Orthogonal Array
19	36	2 **	10	3 **	1	6 **	2	Orthogonal Array
20	36	2 **	4	3 **	1	6 **	3	Orthogonal Array

---

There are 13 two-level factors available in 36 runs, and we need 15, only two more, so we expect to make a pretty good nonorthogonal design.

## Designing the Choice Experiment

The following step creates a design:

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

%mktex(3 ** 15 2, n=&n * &blocks, seed=151)
```

The specification 3 \*\* 15 requests a design with 15 factors, x1---x15, each with three levels. This specification also requests a two-level factor (the 2 following the 3 \*\* 15). This is because 36 choice sets might be too many for one person to rate, so we might want to block the design into two blocks, and we can use a two-level factor to do this. A design with  $18 \times 2 = 36$  runs is requested, which means 36 choice sets. A random number seed is explicitly specified so we can reproduce these exact results.\*

Some of the log messages are as follows:

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

The macro searches a fractional-factorial candidate set of 81 runs, and it also generates an orthogonal array in 36 runs to try as part of the design. This is explained in more detail on page 347.

---

\*By specifying a random number seed, results should be reproducible within a SAS release for a particular operating system and for a particular version of the macro. However, due to machine and macro differences, some results might not be exactly reproducible everywhere. For most orthogonal and balanced designs, the results should be reproducible. When computerized searches are done, it is likely that you will not get the same design as the one in the book, although you expect the efficiency differences to be slight.

Some of the results from the %MktEx macro are as follows:

---

Vacation Example

Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
1	Start	82.2544	82.2544	Can
1	End	82.2544		
2	Start	78.8337		Tab,Ran
2	3 15	82.7741	82.7741	
2	4 14	83.2440	83.2440	
2	4 15	83.6041	83.6041	
2	4 15	83.8085	83.8085	
2	6 14	84.0072	84.0072	
.				
.				
.				
2	End	98.8567		
.				
.				
.				
5	Start	78.5222		Tab,Ran
5	11 14	98.8567	98.8567	
5	End	98.8567		
.				
.				
.				
8	Start	77.5829		Tab,Ran
8	10 15	98.9438	98.9438	
8	End	98.9438		
.				
.				
.				
21	Start	48.8411		Ran,Mut,Ann
21	End	93.1010		

Design Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
0	Initial	98.9438	98.9438	Ini

1	Start	77.8094		Tab,Ran
1	End	98.6368		
2	Start	77.9170		Tab,Ran
2	End	98.5516		
.				
.				
.				
78	Start	79.9023		Tab,Ran
78	24 15	98.9438	98.9438	
78	End	98.9438		
.				
.				
.				
87	Start	74.7014		Tab,Ran
87	4 15	98.9438	98.9438	
87	End	98.9438		
.				
.				
.				
146	Start	78.3794		Tab,Ran
146	19 15	98.9438	98.9438	
146	End	98.9438		
.				
.				
.				
200	Start	84.1995		Tab,Ran
200	End	98.6368		

Vacation Example

Design Refinement History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
0	Initial	98.9438	98.9438	Ini
1	Start	96.5678		Pre,Mut,Ann
1	1 8	98.9438	98.9438	
1	26 14	98.9438	98.9438	
1	30 11	98.9438	98.9438	
1	1 12	98.9438	98.9438	
1	32 5	98.9438	98.9438	
1	18 6	98.9438	98.9438	
1	End	98.9438		

```

.
.
.
6      Start      97.2440      Pre,Mut,Ann
6      33  7      98.9438      98.9438
6      4   3      98.9438      98.9438
6      16 12      98.9438      98.9438
6      3  14      98.9438      98.9438
6      20 15      98.9438      98.9438
6      End      98.6958
    
```

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

Vacation Example

The OPTEX Procedure

Class Level Information

Class	Levels	Values
x1	3	1 2 3
x2	3	1 2 3
x3	3	1 2 3
x4	3	1 2 3
x5	3	1 2 3
x6	3	1 2 3
x7	3	1 2 3
x8	3	1 2 3
x9	3	1 2 3
x10	3	1 2 3
x11	3	1 2 3
x12	3	1 2 3
x13	3	1 2 3
x14	3	1 2 3
x15	3	1 2 3
x16	2	1 2

Vacation Example

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	98.9437	97.9592	98.9743	0.9428

---

The `%MktEx` macro used 30 seconds and found a design that is almost 99%  $D$ -efficient. (Differences in the fourth decimal place between the iteration history and the final table, in this case 98.9438 versus 98.9437, are due to rounding error and differences in ridging strategies between the macro code the generates the design and PROC OPTEX, which evaluates the design, and are nothing to worry about.)

## The `%MktEx` Macro Algorithm

The `%MktEx` macro creates  $D$ -efficient linear experimental designs using several approaches. The macro tries to create an orthogonal array, it searches a set of candidate runs (rows of the design), and it uses a coordinate-exchange algorithm using both a random initial designs and also a partial orthogonal array initialization. The macro stops if at any time it finds a perfect, 100%  $D$ -efficient, orthogonal and balanced design. This first phase is the algorithm search phase. In it, the macro tries a number of methods for this problem. At the end of this phase, the macro chooses the method that has produced the best design and performs another set of iterations using exclusively the chosen approach. Finally, the macro performs a third set of iterations, where it takes the best design it found so far and tries to improve it.

The `%MktEx` macro can directly generate, without iterations, well over one-hundred thousand different 100%  $D$ -efficient, orthogonal and balanced designs. It does this using its design catalog and many different general and ad hoc algorithms. The closest design that the macro knows how to make for this problem is  $2^13^{13}$  in 36 runs.

The candidate-set search has two parts. First, either PROC PLAN is run to create a full-factorial design for small problems, or PROC FACTEX is run to create a fractional-factorial design for large problems. Either way, this larger design is a *candidate set* that in the second part is searched by PROC OPTEX using the modified Fedorov algorithm. A design is built from a selection of the rows of the candidate set (Fedorov 1972; Cook and Nachtsheim 1980). The modified Fedorov algorithm considers each run in the design and each candidate run. Candidate runs are swapped in and design runs are swapped out if the swap improves  $D$ -efficiency. In this case, since the full-factorial design is large (over 14 million runs), the candidate-set search step calls PROC FACTEX to make the candidate set and then PROC OPTEX to do the search. The `Can` line of the iteration history shows that this step found a design that is 82.2544%  $D$ -efficient.

Next, the `%MktEx` macro uses the *coordinate-exchange algorithm*, based on Meyer and Nachtsheim (1995). The coordinate-exchange algorithm considers each level of each factor, and considers the effect on  $D$ -efficiency of changing a level ( $1 \rightarrow 2$ , or  $1 \rightarrow 3$ , or  $2 \rightarrow 1$ , or  $2 \rightarrow 3$ , or  $3 \rightarrow 1$ , or  $3 \rightarrow 2$ , and so on). Exchanges that increase  $D$ -efficiency are performed. In this step, the macro first tries to initialize the design with an orthogonal array (designated by `Tab`, which refers to the orthogonal array table or catalog) and a random design (`Ran`) both. In this case, 14 of the 16 factors can be initialized with the 13 three-level factors and one two-level factor of  $2^43^{13}$ , and the other two factors are randomly initialized. Levels that are not orthogonally initialized might be exchanged for other levels if the exchange increases  $D$ -efficiency. The algorithm search and design search iteration histories for this example show that the macro exchanged levels in factor 14 and 15 only, the ones that are randomly initialized.

The initialization might be more complicated in other problems. Say you asked for the design  $4^15^13^4$  in 18 runs. The macro uses the orthogonal array  $3^66^1$  in 18 runs to initialize the three-level factors orthogonally, and the five-level factor with the six-level factor coded down to five levels (which is unbalanced). The four-level factor is randomly initialized. The macro also tries the same initialization but with a random rather than unbalanced initialization of the five-level factor, as a minor variation on

the first initialization. In the next initialization variation, the macro uses a fully-random initialization. If the number of runs requested is smaller than the number of runs in the initial orthogonal array, the macro initializes the design with just the first  $n$  rows of the orthogonal array. Similarly, if the number of runs requested is larger than the number of runs in the initial orthogonal array, the macro initializes part of the design with the orthogonal array and the remaining rows and columns randomly. The coordinate-exchange algorithm considers each level of each factor that is not orthogonally initialized, and it exchanges a level if the exchange improves  $D$ -efficiency. When the number of runs in the orthogonal array does not match the number of runs desired, none of the design is initialized orthogonally.

The coordinate-exchange algorithm is not restricted by having a candidate set and hence can *potentially* consider any possible design. In practice, however, both the candidate-set-based and coordinate-exchange algorithms consider only a tiny fraction of the possible designs. When the number of runs in the full-factorial design is very small (say 100 or 200 runs), the modified Fedorov algorithm and coordinate-exchange algorithms usually work equally well. When the number of runs in the full-factorial design is small (up to several thousand), the modified Fedorov algorithm is sometimes superior to coordinate exchange, particularly for models with interactions. When the full-factorial design is larger, coordinate exchange is usually the superior approach. However, heuristics like these are sometimes wrong, which is why the macro tries both methods to see which one is really best for each problem.

In the first attempt at coordinate exchange (Design 2), the macro found a design that is 98.8567%  $D$ -efficient (Design 2, **End**). In design 3 and subsequent designs, the macro uses this same approach, but different random initializations of the remaining two factors. In design 8, the `%MktEx` macro finds a design that is 98.9438%  $D$ -efficient. Designs 12 through 21 use a purely random initialization and simulated annealing and are not as good as previous designs. During these iterations, the macro is considering exchanging every level of every factor with every other level, one row and one factor at a time. At this point, the `%MktEx` macro determines that the combination of orthogonal array and random initialization is working best and tries more iterations using that approach. It starts by displaying the initial (**Ini**) best  $D$ -efficiency of 98.9438. In designs 78, 87, 146, and 197 the macro finds a design that is 98.9438%  $D$ -efficient.

Next, the `%MktEx` macro tries to improve the best design it found previously. Using the previous best design as an initialization (**Pre**), and random mutations of the initialization (**Mut**) and simulated annealing (**Ann**), the macro uses the coordinate-exchange algorithm to try to find a better design. This step is important because the best design that the macro found might be an intermediate design, and it might not be the final design at the end of an iteration. Sometimes the iterations deliberately make the designs less  $D$ -efficient, and sometimes, the macro never finds a design as efficient or more efficient again. Hence it is worthwhile to see if the best design found so far can be improved. In this case, the macro fails to improve the design. After iteration 6, the macro stops since it keeps finding the same design over and over. This does not necessarily mean the macro found *the* optimal design; it means it found a very attractive (perhaps local) optimum, and it is unlikely it will do better using this approach. At the end, PROC OPTEX is called to display the levels of each factor and the final  $D$ -efficiency.

*Random mutations* add random noise to the initial design before the iterations start (levels are randomly changed). This might eliminate the perfect balance that is often in the initial design. By default, random mutations are used with designs with fully-random initializations and in the design refinement step; orthogonal initial designs are not mutated.

*Simulated annealing* allows the design to get worse occasionally but with decreasing probability as the number of exchanges increases. For design 1, for the first level of the first factor, by default, the macro might execute an exchange (say change a 2 to a 1), that makes the design worse, with probability



0.05. As more and more exchanges occur, this probability decreases so at the end of the processing of design 1, exchanges that decrease  $D$ -efficiency are hardly ever done. For design 2, this same process is repeated, again starting by default with an annealing probability of 0.05. This often helps the algorithm overcome local efficiency maxima. To envision this, imagine that you are standing on a molehill next to a mountain. The only way you can start going up the mountain is to first step down off the molehill. Once you are on the mountain, you might occasionally hit a dead end, where all you can do is step down and look for a better place to continue going up. Other analogies include cleaning a garage and painting a room. Both have steps where you make things look worse so that in the end they look better. The solitaire game “Spider,” which is available on many PCs, is another example. Sometimes, you need to temporarily break apart those suits that you so carefully put together in order to make progress. Simulated annealing, by occasionally stepping down the efficiency function, often allows the macro to go farther up it than it would otherwise. The simulated annealing is why you sometimes see designs getting worse in the iteration history. However, the macro keeps track of the best design, not the final design in each step. By default, annealing is used with designs with fully-random initializations and in the design refinement step; simulated annealing is not used with orthogonal initial designs.

For this example, the `%MktEx` macro ran in around 30 seconds. If an orthogonal design had been available, run time would have been a few seconds. If the fully-random initialization method had been the best method, run time might have been on the order of 10 to 45 minutes. Since the orthogonal array initialization worked best, run time is much shorter. While it is possible to construct huge problems that take much longer, for any design that most marketing researchers are likely to encounter, run time should be less than one hour. One of the macro options, `maxtime=`, typically ensures this.

## Examining the Design

Before you use a design, you should always look at its characteristics. We can use the `%MktEval` macro to do this as follows:

```
%mkteval(data=randomized)
```

The results are as follows:

---

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

#### Vacation Example

##### Summary of Frequencies

There are 0 Canonical Correlations Greater Than 0.316

\* - Indicates Unequal Frequencies

##### Frequencies

x1	12	12	12
x2	12	12	12
x3	12	12	12
x4	12	12	12
x5	12	12	12
x6	12	12	12
x7	12	12	12
x8	12	12	12
x9	12	12	12
x10	12	12	12
x11	12	12	12
x12	12	12	12
x13	12	12	12
x14	12	12	12
x15	12	12	12
x16	18	18	

x1 x2	4 4 4 4 4 4 4 4 4
x1 x3	4 4 4 4 4 4 4 4 4
x1 x4	4 4 4 4 4 4 4 4 4
x1 x5	4 4 4 4 4 4 4 4 4
x1 x6	4 4 4 4 4 4 4 4 4
x1 x7	4 4 4 4 4 4 4 4 4
x1 x8	4 4 4 4 4 4 4 4 4
x1 x9	4 4 4 4 4 4 4 4 4
x1 x10	4 4 4 4 4 4 4 4 4
x1 x11	4 4 4 4 4 4 4 4 4
x1 x12	4 4 4 4 4 4 4 4 4
x1 x13	4 4 4 4 4 4 4 4 4
x1 x14	4 4 4 4 4 4 4 4 4
x1 x15	4 4 4 4 4 4 4 4 4
x1 x16	6 6 6 6 6 6
.	
.	
.	
* x13 x14	3 6 3 6 3 3 3 3 6
* x13 x15	3 3 6 3 6 3 6 3 3
x13 x16	6 6 6 6 6 6
* x14 x15	3 6 3 3 3 6 6 3 3
x14 x16	6 6 6 6 6 6
x15 x16	6 6 6 6 6 6
N-Way	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

---

This design looks great! The factors x1-x13 form an orthogonal design, x14 and x15 are slightly correlated with each other and with x13. The blocking factor x16 is orthogonal to all the other factors. All of the factors are perfectly balanced. The N-Way frequencies show that each choice set appears once.

What if there had been some larger canonical correlations? Would this be a problem? That depends. You have to decide this for yourself based on your particular study. You do not want large correlations between your most important factors. If you have high correlations between the wrong factors, you can swap them with other factors with the same number of levels, or try to make a new design with a different seed, or change the number of choice sets, and so on. While this design looks great, we should make one minor adjustment based on these results. Since our correlations are in the factors we originally planned to make price factors, we should change our plans slightly and use those factors for less important attributes like scenery.

You can run the %MktEx macro to provide additional information about a design, for example, asking to examine the information matrix (I) and its inverse (V), which is the variance matrix of the parameter estimates. You hope to see that all of the off-diagonal elements of the variance matrix, the covariances, are small relative to the variances on the diagonal. When options=check is specified, the macro evaluates an initial design instead of generating a design. The option init=randomized names the design to evaluate, and the examine= option displays the information and variance matrices. The blocking variable is dropped.

The following step creates the design:

```
%mktex(3 ** 15,          /* all attrs of all alternatives */
      n=&n * &blocks,     /* total number of choice sets */
      init=randomized(drop=x16), /* initial design */
      options=check,      /* check initial design efficiency */
      examine=i v)       /* show information & variance matrices */
```

A small part of the output is as follows:

---

Vacation Example

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

Vacation Example  
Information Matrix

	Intercept	x11	x12	x21	x22	x31	x32	x41
Intercept	36.000	0	0	0	0	0	0	0
x11	0	36.000	0	0	0	0	0	0
x12	0	0	36.000	0	0	0	0	0
x21	0	0	0	36.000	0	0	0	0
x22	0	0	0	0	36.000	0	0	0
x31	0	0	0	0	0	36.000	0	0
x32	0	0	0	0	0	0	36.000	0
x41	0	0	0	0	0	0	0	36.000
.								
.								
.								
x122	36.000	0	0	0	0	0	0	0
x131	0	36.000	0	9.000	0	-4.500	-7.794	-7.794
x132	0	0	36.000	0	9.000	-7.794	4.500	4.500
x141	0	9.000	0	36.000	0	-4.500	-7.794	-7.794
x142	0	0	9.000	0	36.000	-7.794	4.500	4.500
x151	0	-4.500	-7.794	-4.500	-7.794	36.000	0	0
x152	0	-7.794	4.500	-7.794	4.500	0	36.000	36.000

Vacation Example Variance Matrix								
	Intercept	x11	x12	x21	x22	x31	x32	x41
Intercept	0.028	0	0	0	0	0	0	0
x11	0	0.028	0	0	0	0	0	0
x12	0	0	0.028	0	0	0	0	0
x21	0	0	0	0.028	0	0	0	0
x22	0	0	0	0	0.028	0	0	0
x31	0	0	0	0	0	0.028	0	0
x32	0	0	0	0	0	0	0.028	0
x41	0	0	0	0	0	0	0	0.028
.								
.								
.								
x122	0.028	0	0	0	0	0	0	0
x131	0	0.031	0	-0.006	0	0.003	0.005	0.005
x132	0	0	0.031	0	-0.006	0.005	-0.003	-0.003
x141	0	-0.006	0	0.031	0	0.003	0.005	0.005
x142	0	0	-0.006	0	0.031	0.005	-0.003	-0.003
x151	0	0.003	0.005	0.003	0.005	0.031	0	0
x152	0	0.005	-0.003	0.005	-0.003	0	0.031	0.031

This design still looks good. The  $D$ -efficiency for the design excluding the blocking factor is 98.9099%. We can see that the nonorthogonality between x13-x15 makes their variances larger than the other factors (0.031 versus 0.028).

These next steps use the %MktLab macro to reassign the variable names, store the design in a permanent SAS data set, sasuser.VacationLinDesBlckd, and then use the %MktEx macro to check the results. See page 309 for more information about permanent SAS data sets. We need to make the correlated variables correspond to the least important attributes in different alternatives (in this case the scenery factors for Alaska, Mexico, and Maine). The vars= option provides the new variable names: the first variable (originally x1) becomes x1 (still), ..., the fifth variable (originally x5) becomes x5 (still), the sixth variable (originally x6) becomes x11, ... the tenth variable (originally x10) becomes x15, the eleventh through fifteenth original variables become x6, x9, x7, x8, x10, and finally the last variable becomes Block. The following PROC SORT step sorts the design into blocks:

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

proc sort data=sasuser.VacationLinDesBlckd; by block; run;

%mkteval(blocks=block)
```

The output from the %MktLab macro, which shows the correspondence between the original and new variable names is as follows:

---

Variable Mapping:

x1 : x1  
 x2 : x2  
 x3 : x3  
 x4 : x4  
 x5 : x5  
 x6 : x11  
 x7 : x12  
 x8 : x13  
 x9 : x14  
 x10 : x15  
 x11 : x6  
 x12 : x9  
 x13 : x7  
 x14 : x8  
 x15 : x10  
 x16 : Block

---

Some of the output from the %MktEval macro is as follows:

---

Vacation Example  
 Canonical Correlations Between the Factors  
 There are 0 Canonical Correlations Greater Than 0.316

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

Vacation Example  
Summary of Frequencies

There are 0 Canonical Correlations Greater Than 0.316

\* - Indicates Unequal Frequencies

Frequencies

Block	18 18
x1	12 12 12
x2	12 12 12
x3	12 12 12
x4	12 12 12
x5	12 12 12
x11	12 12 12
x12	12 12 12
x13	12 12 12
x14	12 12 12
x15	12 12 12
x6	12 12 12
x9	12 12 12
x7	12 12 12
x8	12 12 12
x10	12 12 12
Block x1	6 6 6 6 6 6
Block x2	6 6 6 6 6 6
Block x3	6 6 6 6 6 6
Block x4	6 6 6 6 6 6
Block x5	6 6 6 6 6 6
Block x11	6 6 6 6 6 6
Block x12	6 6 6 6 6 6
Block x13	6 6 6 6 6 6
Block x14	6 6 6 6 6 6
Block x15	6 6 6 6 6 6
Block x6	6 6 6 6 6 6
Block x9	6 6 6 6 6 6
Block x7	6 6 6 6 6 6
Block x8	6 6 6 6 6 6
Block x10	6 6 6 6 6 6
x1 x2	4 4 4 4 4 4 4 4 4
x1 x3	4 4 4 4 4 4 4 4 4
x1 x4	4 4 4 4 4 4 4 4 4
x1 x5	4 4 4 4 4 4 4 4 4
.	
.	
.	

```

*   x7 x8      6 3 3 3 6 3 3 3 6
*   x7 x10     3 3 6 3 6 3 6 3 3
*   x8 x10     3 3 6 3 6 3 6 3 3
      N-Way    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
              1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

```

---

## From a Linear Arrangement to a Choice Design

These next steps prepare the design for analysis and further evaluation. We need to convert our linear arrangement into a choice design.\* We need to create a data set `Key` that describes how the factors in our linear arrangement are used to make the choice design for analysis. The `Key` data set contains all of the factor names, `x1`, `x2`, `x3`, ... `x15`. We can run the `%MktKey` macro to get these names, for cutting and pasting into the program without typing them. The following step requests 5 rows, 3 columns and the results transposed so names progress down each column instead of across each row:

```
%mktkey(5 3 t)
```

The `%MktKey` macro produces the following data set:

---

```

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

```

---

\*See page 67 for an explanation of the linear arrangement of a choice design versus the arrangement of a choice design that is more suitable for analysis.



The following steps make the Key data set and processes the design:

```

title 'Vacation Example';

data key;
  input Place $ 1-10 (Lodge Scene Price) ($);
  datalines;
Hawaii      x1      x6      x11
Alaska      x2      x7      x12
Mexico      x3      x8      x13
California  x4      x9      x14
Maine       x5      x10     x15
Home        .       .       .
;

%mktroll(design=sasuser.VacationLinDesBlckd, key=key, alt=place,
         out=sasuser.VacationChDes)

```

For analysis, the design has 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 are constructed from the `sasuser.VacationLinDesBlckd` data set. `Lodge` is created from `x1` for Hawaii, `x2` for Alaska, ..., `x5` for Maine, and no attribute for Home. Similarly, `Scene` is created from `x6-x10`, and `Price` is created from `x11-x15`. The macro `%MktRoll` is used to create the data set `sasuser.VacationChDes` from `sasuser.VacationLinDesBlckd` using the mapping in `Key` and using the variable `Place` as the alternative ID variable.

The macro displays the following warning:

```

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.VacationLinDesBlckd` data set does not appear in the final design. The purpose of the variable `Block` (sorting the design into blocks) has already been achieved. You can specify `options=nowarn` if you want to suppress this warning.

These next steps show the results for the first two choice sets:

```

proc print data=sasuser.VacationLinDesBlckd(obs=2);
  id Block;
  var x1-x15;
run;

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

```

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.

The results are as follows:

---

Vacation Example

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

Vacation Example

Set	Place	Lodge	Scene	Price
1	Hawaii	1	3	1
	Alaska	3	1	1
	Mexico	3	3	2
	California	1	2	2
	Maine	1	1	2
	Home	.	.	.
	2	Hawaii	1	3
Alaska		2	2	3
Mexico		1	2	1
California		2	1	1
Maine		3	3	2
Home		.	.	.

---

The following steps assign formats, convert the variable `Price` to contain actual prices, and recode the constant alternative:

```
proc format;
  value price 1 = ' 999'      2 = '1249'
             3 = '1499'      0 = '  0';
  value scene 1 = 'Mountains' 2 = 'Lake'
             3 = 'Beach'      0 = 'Home';
  value lodge 1 = 'Cabin'     2 = 'Bed & Breakfast'
             3 = 'Hotel'     0 = 'Home';
run;

data sasuser.VacationChDes;
  set sasuser.VacationChDes;
  if place = 'Home' then do; lodge = 0; scene = 0; price = 0; end;
  price = input(put(price, price.), 5.);
  format scene scene. lodge lodge.;
run;

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

The results are as follows:

---

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

---

It is not necessary to recode the missing values for the constant alternative. In practice, we usually do not do this step. However, for this first analysis, we want all nonmissing values of the attributes so we can see all levels in the final displayed output. We also recode `Price` so that for a later analysis, we can analyze `Price` as a quantitative effect. For example, the expression `put(price, price.)` converts a number, say 2, into a string (in this case '1249'), then the `input` function reads the string and converts it to a numeric 1249.

## Testing the Design Before Data Collection

Collecting data is time consuming and expensive. It is always good practice to make sure that the design works with the most complicated model that we anticipate fitting. The following steps evaluate the choice design:

```

title2 'Evaluate the Choice Design';

%choicEff(data=sasuser.VacationChDes,/* candidate set of choice sets      */
init=sasuser.VacationChDes(keep=set), /* select these sets              */
intiter=0,                            /* evaluate without internal iterations */
                                        /* alternative-specific effects model   */
                                        /* zero=none - use all levels of place */
                                        /* order=data - do not sort levels    */
model=class(place / zero=none order=data)
                                        /* place * price ... - interactions or */
                                        /*   alternative-specific effects      */
class(place * price place * scene place * lodge /
zero=none /* zero=none - use all levels of place */
order=formatted) / /* order=formatted - sort levels */
lprefix=0 /* lpr=0 labels created from just levels*/
cprefix=0 /* cpr=0 names created from just levels */
separators=' ' ', /* use comma sep to build interact terms*/

nsets=36, /* number of choice sets          */
nalts=6, /* number of alternatives            */
beta=zero) /* assumed beta vector, Ho: b=0    */

```

The `%ChoiEff` macro has two uses. You can use it to search for an efficient choice design, or you can use it to evaluate a choice design including designs that are generated using other methods such as the `%MktEx` macro. It is this latter use that is illustrated here.

The way you check a design like this is to first name it in the `data=` option. This is the candidate set that contains all of the choice sets that we will consider. In addition, the same design is named in the `init=` option. Just the variable `Set` is kept. It is used to bring in just the indicated choice sets from the `data=` design, which in this case is all of them. The option `nsets=` specifies that there are 36 choice sets, and `nalts=` specifies that there are 6 alternatives. The option `beta=zero` specifies that we are assuming for design evaluation purposes the null hypothesis that all of the betas or part-worth utilities are zero. You can evaluate the design for other parameter vectors by specifying a list of numbers after `beta=`. This will change the variances and standard errors. We also specify `intiter=0` which specifies zero internal iterations. We use zero internal iterations when we want to evaluate an initial design, but not attempt to improve it. The last option specifies the model.

The model specification contains everything that appears in the TRANSREG procedure's `model` statement for coding the design. Some of these options are familiar from the previous example. The specification `class(place / zero=none order=data)` names the `place` variable as a classification variables and asks for coded variables for every level including the constant, stay-at-home alternative. The specification `class(place * price place * scene place * lodge / zero=none order=formatted)` asks for alternative-specific effects for price, lodging, and scenery. The alternative-specific effects permit the part-worth utilities to be different for each of the destinations. This is accomplished by requesting interactions between the destination and the attributes. `Class` levels are sorted by their formatted values, and for all factors. The `zero=none` option is used so that for now we can see all of the levels of all of the factors. Many of these will not correspond to estimable parameters, and we can eliminate them later. See page 78 for more information about the `zero=` option.

The `lprefix=0` option specifies that when labels are created for the binary variables, zero characters of the original variable name should be used as a prefix. The `cprefix=0` option specifies that when names are created for the binary variables, zero characters of the original variable name should be used as a prefix. The `separators=' ' ', '` option provides two strings (one blank and the other a comma followed by a blank) that let you specify label component separators for the main effect and interaction terms. By specifying a comma and a blank for the second value, we request labels for the side trip effects like `'Alaska, 999'` instead of the default `'Alaska * 999'`. This option is explained in more detail on page 449.

The last tables from the `%ChoiceEff` macro, which are the ones in which we are most interested, is as follows:

---

Vacation Example  
Evaluate the Choice Design

Final Results

Design	1
Choice Sets	36
Alternatives	6
Parameters	35
Maximum Parameters	180
D-Efficiency	0
D-Error	.

Vacation Example  
Evaluate the Choice Design

n	Variable Name	Label	Variance	DF	Standard Error
1	Hawaii	Hawaii	1.53333	1	1.23828
2	Alaska	Alaska	1.53545	1	1.23913
3	Mexico	Mexico	1.53545	1	1.23913
4	California	California	1.53333	1	1.23828
5	Maine	Maine	1.53545	1	1.23913
6	Home	Home	.	0	.
7	Alaska_0	Alaska, 0	.	0	.
8	Alaska_999	Alaska, 999	1.20000	1	1.09545
9	Alaska_1249	Alaska, 1249	1.20000	1	1.09545
10	Alaska_1499	Alaska, 1499	.	0	.
11	California_0	California, 0	.	0	.
12	California_999	California, 999	1.20000	1	1.09545
13	California_1249	California, 1249	1.20000	1	1.09545
14	California_1499	California, 1499	.	0	.
15	Hawaii_0	Hawaii, 0	.	0	.
16	Hawaii_999	Hawaii, 999	1.20000	1	1.09545
17	Hawaii_1249	Hawaii, 1249	1.20000	1	1.09545
18	Hawaii_1499	Hawaii, 1499	.	0	.
19	Home_0	Home, 0	.	0	.
20	Home_999	Home, 999	.	0	.
21	Home_1249	Home, 1249	.	0	.
22	Home_1499	Home, 1499	.	0	.
23	Maine_0	Maine, 0	.	0	.
24	Maine_999	Maine, 999	1.20000	1	1.09545
25	Maine_1249	Maine, 1249	1.20000	1	1.09545
26	Maine_1499	Maine, 1499	.	0	.
27	Mexico_0	Mexico, 0	.	0	.
28	Mexico_999	Mexico, 999	1.20000	1	1.09545
29	Mexico_1249	Mexico, 1249	1.20000	1	1.09545
30	Mexico_1499	Mexico, 1499	.	0	.
31	AlaskaBeach	Alaska, Beach	1.20635	1	1.09834
32	AlaskaHome	Alaska, Home	.	0	.
33	AlaskaLake	Alaska, Lake	1.20635	1	1.09834
34	AlaskaMountains	Alaska, Mountains	.	0	.
35	CaliforniaBeach	California, Beach	1.20000	1	1.09545
36	CaliforniaHome	California, Home	.	0	.
37	CaliforniaLake	California, Lake	1.20000	1	1.09545
38	CaliforniaMountains	California, Mountains	.	0	.
39	HawaiiBeach	Hawaii, Beach	1.20000	1	1.09545
40	HawaiiHome	Hawaii, Home	.	0	.
41	HawaiiLake	Hawaii, Lake	1.20000	1	1.09545
42	HawaiiMountains	Hawaii, Mountains	.	0	.

43	HomeBeach	Home, Beach	.	0	.
44	HomeHome	Home, Home	.	0	.
45	HomeLake	Home, Lake	.	0	.
46	HomeMountains	Home, Mountains	.	0	.
47	MaineBeach	Maine, Beach	1.20635	1	1.09834
48	MaineHome	Maine, Home	.	0	.
49	MaineLake	Maine, Lake	1.20635	1	1.09834
50	MaineMountains	Maine, Mountains	.	0	.
51	MexicoBeach	Mexico, Beach	1.20635	1	1.09834
52	MexicoHome	Mexico, Home	.	0	.
53	MexicoLake	Mexico, Lake	1.20635	1	1.09834
54	MexicoMountains	Mexico, Mountains	.	0	.
55	AlaskaBed___Breakfast	Alaska, Bed & Breakfast	1.20000	1	1.09545
56	AlaskaCabin	Alaska, Cabin	1.20000	1	1.09545
57	AlaskaHome2	Alaska, Home	.	0	.
58	AlaskaHotel	Alaska, Hotel	.	0	.
59	CaliforniaBed___Breakfast	California, Bed & Breakfast	1.20000	1	1.09545
60	CaliforniaCabin	California, Cabin	1.20000	1	1.09545
61	CaliforniaHome2	California, Home	.	0	.
62	CaliforniaHotel	California, Hotel	.	0	.
63	HawaiiBed___Breakfast	Hawaii, Bed & Breakfast	1.20000	1	1.09545
64	HawaiiCabin	Hawaii, Cabin	1.20000	1	1.09545
65	HawaiiHome2	Hawaii, Home	.	0	.
66	HawaiiHotel	Hawaii, Hotel	.	0	.
67	HomeBed___Breakfast	Home, Bed & Breakfast	.	0	.
68	HomeCabin	Home, Cabin	.	0	.
69	HomeHome2	Home, Home	.	0	.
70	HomeHotel	Home, Hotel	.	0	.
71	MaineBed___Breakfast	Maine, Bed & Breakfast	1.20000	1	1.09545
72	MaineCabin	Maine, Cabin	1.20000	1	1.09545
73	MaineHome2	Maine, Home	.	0	.
74	MaineHotel	Maine, Hotel	.	0	.
75	MexicoBed___Breakfast	Mexico, Bed & Breakfast	1.20000	1	1.09545
76	MexicoCabin	Mexico, Cabin	1.20000	1	1.09545
77	MexicoHome2	Mexico, Home	.	0	.
78	MexicoHotel	Mexico, Hotel	.	0	.

==  
35

---

We see estimable parameters for the five destinations, but not for the stay at home alternative. This is because the last level, the home alternative, is a reference level. For each destination/attribute combination, which are the alternative-specific effects, we see two estimable parameters. We also see two more lines with zero degrees of freedom. One corresponds to the last level of the attribute. This is the usual reference term. It is a linear combination of terms that come before. For example, the reference level for price of Alaska, `Alaska_1499` is equal to `Alaska - Alaska_999 - Alaska_1249`. The other zero degree of freedom term corresponds to staying home. Consider, for example, the price of 0 (the price of staying home) and the Alaska destination. The indicator variable for this term consists

of all zeros since the price of 0 never happens with Alaska. So there is no parameter to estimate. You can prove these things to yourself by examining the coded design. The %ChoiceEff macro makes it available in a SAS data set called tmp\_cand. This data set is not considered to be one of the “normal” output data sets from the macro, so no note is created about its creation. However, it is always there in case you want to look at it. You can verify that Alaska\_1499 is equal to Alaska - Alaska\_999 - Alaska\_1249 as follows:

```
proc reg data=tmp_cand;
  model Alaska_1499 = Alaska Alaska_999 Alaska_1249 / noint;
run; quit;
```

The results are as follows:

---

Vacation Example  
Evaluate the Choice Design

The REG Procedure

Model: MODEL1

Dependent Variable: Alaska\_1499 Alaska, 1499

Number of Observations Read	216
Number of Observations Used	216

NOTE: No intercept in model. R-Square is redefined.

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	3	12.00000	4.00000	Infty	<.0001
Error	213	0	0		
Uncorrected Total	216	12.00000			

Root MSE	0	R-Square	1.0000
Dependent Mean	0.05556	Adj R-Sq	1.0000
Coeff Var	0		

Parameter Estimates

Variable	Label	DF	Parameter Estimate	Standard Error	t Value	Pr >  t
Alaska	Alaska	1	1.00000	0	Infty	<.0001
Alaska_999	Alaska, 999	1	-1.00000	0	-Infty	<.0001
Alaska_1249	Alaska, 1249	1	-1.00000	0	-Infty	<.0001

---



You can verify that Alaska\_0 is all zero as follows:

```
proc freq data=tmp_cand;
  tables Alaska_0;
run;
```

The results are as follows:

---

Vacation Example				
Evaluate the Choice Design				
The FREQ Procedure				
Alaska, 0				
Alaska_0	Frequency	Percent	Cumulative Frequency	Cumulative Percent
-----				
0	216	100.00	216	100.00

---

You can similarly verify the causes of the 0 *df* for the other terms in the model.

The standard errors for most of the alternative-specific effects are 1.09545, but a few are a bit higher. They correspond to the scenery attributes for Alaska, Maine, and Mexico, which are our nonorthogonal factors. This design looks quite good. Everything that should be estimable in an alternative-specific effects model is estimable, and all of the standard errors are of a similar magnitude.

You can run the %ChoiceEff macro one more time and get closer to just a listing of the estimable terms as follows:

```
title2 'Evaluate the Choice Design';
%choiceff(data=sasuser.VacationChDes, /* candidate set of choice sets      */
  init=sasuser.VacationChDes(keep=set), /* select these sets          */
  intiter=0, /* evaluate without internal iterations */
  /* alternative-specific effects model */
  /* ref level for place is 'Home'    */
  /* order=data - do not sort levels  */
  model=class(place / zero='Home' order=data)
  /* ref level for place is 'Home'    */
  /* ref level for price is 0         */
  /* ref level for scene is 'Home'    */
  /* ref level for lodge is 'Home'    */
  /* order=formatted - sort levels    */
  class(place * price place * scene place * lodge /
    zero='Home' '0' 'Home' 'Home' order=formatted) /
  lprefix=0 /* lpr=0 labels created from just levels*/
  cprefix=0 /* cpr=0 names created from just levels */
  separators=' ' ', ' /* use comma sep to build interact terms*/
  nsets=36, /* number of choice sets */
  nalts=6, /* number of alternatives */
  beta=zero) /* assumed beta vector, Ho: b=0 */
```

The results of this step are not shown, but there are 35 model *df* like before and many fewer zeros. The macro displays a list of the terms with zero *df*. In this case, the list is as follows:

```
Alaska_1499 California_1499 Hawaii_1499 Maine_1499 Mexico_1499 AlaskaMountains
CaliforniaMountains HawaiiMountains MaineMountains MexicoMountains AlaskaHotel
CaliforniaHotel HawaiiHotel MaineHotel MexicoHotel
```

You can incorporate this list into the macro as follows:

```
title2 'Evaluate the Choice Design';

%choiceff(data=sasuser.VacationChDes,/* candidate set of choice sets      */
init=sasuser.VacationChDes(keep=set), /* select these sets              */
intiter=0,                            /* evaluate without internal iterations */
                                        /* alternative-specific effects model  */
                                        /* ref level for place is 'Home'      */
                                        /* order=data - do not sort levels    */
model=class(place / zero='Home' order=data)
                                        /* ref level for place is 'Home'     */
                                        /* ref level for price is 0           */
                                        /* ref level for scene is 'Home'     */
                                        /* ref level for lodge is 'Home'     */
                                        /* order=formatted - sort levels     */
class(place * price place * scene place * lodge /
zero='Home' '0' 'Home' 'Home' order=formatted) /
lprefix=0                               /* lpr=0 labels created from just levels*/
cprefix=0                               /* cpr=0 names created from just levels */
separators=' ' ', ', /* use comma sep to build interact terms*/

nsets=36,                               /* number of choice sets              */
nalts=6,                                /* number of alternatives              */
                                        /* extra model terms to drop from model */
drop=Alaska_1499 California_1499 Hawaii_1499 Maine_1499
Mexico_1499 AlaskaMountains CaliforniaMountains
HawaiiMountains MaineMountains MexicoMountains AlaskaHotel
CaliforniaHotel HawaiiHotel MaineHotel MexicoHotel,
beta=zero)                               /* assumed beta vector, Ho: b=0      */
```

Most of the results are as follows:

---

Vacation Example  
Evaluate the Choice Design

Design	Iteration	D-Efficiency	D-Error
1	0	1.18690 *	0.84253

Vacation Example  
Evaluate the Choice Design

Final Results

Design	1
Choice Sets	36
Alternatives	6
Parameters	35
Maximum Parameters	180
D-Efficiency	1.1869
D-Error	0.8425

Vacation Example  
Evaluate the Choice Design

n	Variable Name	Label	Variance	DF	Standard Error
1	Hawaii	Hawaii	1.53333	1	1.23828
2	Alaska	Alaska	1.53545	1	1.23913
3	Mexico	Mexico	1.53545	1	1.23913
4	California	California	1.53333	1	1.23828
5	Maine	Maine	1.53545	1	1.23913
6	Alaska_999	Alaska, 999	1.20000	1	1.09545
7	Alaska_1249	Alaska, 1249	1.20000	1	1.09545
8	California_999	California, 999	1.20000	1	1.09545
9	California_1249	California, 1249	1.20000	1	1.09545
10	Hawaii_999	Hawaii, 999	1.20000	1	1.09545
11	Hawaii_1249	Hawaii, 1249	1.20000	1	1.09545
12	Maine_999	Maine, 999	1.20000	1	1.09545
13	Maine_1249	Maine, 1249	1.20000	1	1.09545
14	Mexico_999	Mexico, 999	1.20000	1	1.09545
15	Mexico_1249	Mexico, 1249	1.20000	1	1.09545
16	AlaskaBeach	Alaska, Beach	1.20635	1	1.09834
17	AlaskaLake	Alaska, Lake	1.20635	1	1.09834
18	CaliforniaBeach	California, Beach	1.20000	1	1.09545
19	CaliforniaLake	California, Lake	1.20000	1	1.09545
20	HawaiiBeach	Hawaii, Beach	1.20000	1	1.09545
21	HawaiiLake	Hawaii, Lake	1.20000	1	1.09545
22	MaineBeach	Maine, Beach	1.20635	1	1.09834
23	MaineLake	Maine, Lake	1.20635	1	1.09834
24	MexicoBeach	Mexico, Beach	1.20635	1	1.09834
25	MexicoLake	Mexico, Lake	1.20635	1	1.09834
26	AlaskaBed___Breakfast	Alaska, Bed & Breakfast	1.20000	1	1.09545
27	AlaskaCabin	Alaska, Cabin	1.20000	1	1.09545
28	CaliforniaBed___Breakfast	California, Bed & Breakfast	1.20000	1	1.09545
29	CaliforniaCabin	California, Cabin	1.20000	1	1.09545

30	HawaiiBed__Breakfast	Hawaii, Bed & Breakfast	1.20000	1	1.09545
31	HawaiiCabin	Hawaii, Cabin	1.20000	1	1.09545
32	MaineBed__Breakfast	Maine, Bed & Breakfast	1.20000	1	1.09545
33	MaineCabin	Maine, Cabin	1.20000	1	1.09545
34	MexicoBed__Breakfast	Mexico, Bed & Breakfast	1.20000	1	1.09545
35	MexicoCabin	Mexico, Cabin	1.20000	1	1.09545
				==	
				35	

---

Now we have all of the final 35 parameters. We used the `drop=` option to drop the extra terms. It is often the case with a choice model with a constant alternative that it is hard to write a model that precisely creates all of the right terms with nothing extra. The `drop=` option gives you a way to to remove the extra terms.

These results look good. The variances for Alaska, Mexico, and Maine are slightly larger than the variances for Hawaii and California. Similarly, the variances for the scenery parameters for Alaska, Mexico, and Maine are larger than the variances for the other attributes. This is because we have a small amount of nonorthogonality in those attributes. You can see that this effect is slight.

There is one more test that should be run before a design is used. The following `%MktDups` macro step checks the design to see if any choice sets are duplicates of any other choice sets:

```
%mktDups(branded, data=sasuser.VacationChDes,
          nalts=6, factors=place price scene lodge)
```

The results are as follows:

---

```
Design:          Branded
Factors:         place price scene lodge
                 Place
                 Lodge Price Scene
Duplicate Sets:  0
```

---

The first line of the table tells us that this is a branded design as opposed to a generic design (bundles of attributes with no brands). The second line tells us the factors as specified in the `factors=` option. These are followed by the actual variable names for the factors. The last line reports the number of duplicates. In this case, there are no duplicate choice sets. If there are duplicate choice sets, then changing the random number seed might help. Changing other aspects of the design or the approach for making the design might also help.

## Generating the Questionnaire

The following DATA step produces the questionnaires:

```

title;
proc sort data=sasuser.VacationLinDesBlckd; by block; run;

options ls=80 ps=60 nodate nonumber;

data _null_;
  array dests[&mm1] $ 10 _temporary_ ('Hawaii' 'Alaska' 'Mexico'
                                     'California' 'Maine');
  array prices[3] $ 5 _temporary_ ('$999' '$1249' '$1499');
  array scenes[3] $ 13 _temporary_
    ('the Mountains' 'a Lake' 'the Beach');
  array lodging[3] $ 15 _temporary_
    ('Cabin' 'Bed & Breakfast' 'Hotel');
  array x[15];
  file print linesleft=11;

  set sasuser.VacationLinDesBlckd;
  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;

```

They are then copied and the data are collected. In this design, there are five destinations, and each destination has three attributes. Each destination name is accessed from the array `dests`. Note that destination is not a factor in the design; it is a bin into which the attributes are grouped. The factors in the design are named in the statement `array x[15]`, which is a short-hand notation for `array x[15] x1-x15`. The first five factors are used for the lodging attribute of the five destinations. The actual descriptions of lodging are accessed by `lodging[x[dest]]`. The variable `Dest` varies from 1 to 5 destinations, so `x[dest]` extracts the levels for the `Dest` destination. Similarly for scenery,

`scenes[x[&mm1 + dest]]` extracts the descriptions of the scenery. The index `&mm1 + dest` accesses factors 6 through 10, and `x[&mm1 + dest]` indexes the `scenes` array. For prices, `prices[x[2 * &mm1 + dest]]`, the index `2 * &mm1 + dest` accesses the factors 11 through 15.

The first two choice sets are as follows:

---

Vacation Example

Form: 1 Subject: \_\_\_\_\_

1) Circle your choice of vacation destinations:

- 1) Hawaii, staying in a Cabin near the Beach,  
with a package cost of \$999.
- 2) Alaska, staying in a Hotel near the Mountains,  
with a package cost of \$999.
- 3) Mexico, staying in a Hotel near the Beach,  
with a package cost of \$1249.
- 4) California, staying in a Cabin near a Lake,  
with a package cost of \$1249.
- 5) Maine, staying in a Cabin near the Mountains,  
with a package cost of \$1249.
- 6) Stay at home this year.

2) Circle your choice of vacation destinations:

- 1) Hawaii, staying in a Cabin near the Beach,  
with a package cost of \$999.
  - 2) Alaska, staying in a Bed & Breakfast near a Lake,  
with a package cost of \$1499.
  - 3) Mexico, staying in a Cabin near a Lake,  
with a package cost of \$999.
  - 4) California, staying in a Bed & Breakfast near the Mountains,  
with a package cost of \$999.
  - 5) Maine, staying in a Hotel near the Beach,  
with a package cost of \$1249.
  - 6) Stay at home this year.
-

In practice, data collection is typically much more elaborate than this. It might involve art work or photographs, and the choice sets might be presented and the data might be collected through personal interview or over the Web. However the choice sets are presented and the data are collected, the essential elements remain the same. Subjects are shown a set of alternatives and are asked to make a choice, then they go on to the next set.

## Entering and Processing the Data

The data are read as follows:

```

title 'Vacation Example';

data results;
  input Subj Form (choose1-choose&n) (1.) @@;
  datalines;
  1  1 111353313351554151  2  2 344113155513111413  3  1 132353331151534151
  4  2 341133131523331143  5  1 142153111151334143  6  2 344114111543131151
  7  1 141343111311154154  8  2 344113111343121111  9  1 141124131151342155
 10  2 344113131523131141 11  1 311423131353524144 12  2 332123151413331151
 13  1 311244331352134155 14  2 3411141111543131153 15  1 141253111351344151
 16  2 344135131323331143 17  1 142123313154132141 18  2 542113151323131141
 19  1 145314111311144111 20  2 344111131313431143 21  1 133343131313432145
  .
  .
  .
  ;

```

Data from a total of 200 subjects are collected, 100 per form.

Next, we use the %MktMerge macro as follows to combine the data and design and create the variable `c`, indicating whether each alternative is a first choice or a subsequent choice:

```

%mktmerge(design=sasuser.VacationChDes, data=results, out=res2, blocks=form,
          nsets=&n, nalts=&m, setvars=choose1-choose&n)

proc print data=res2(obs=12);
  id subj form set; by subj form set;
run;

```

This macro takes the `design=sasuser.VacationChDes` experimental design, merges it with the `data=result` data set, creating the `out=res2` output data set. The `Results` data set contains the variable `Form` that contains the block number. Since there are two blocks, this variable must have values of 1 and 2. This variable must be specified in the `blocks=` option. The experiment has `nsets=&n` choice sets, `nalts=6` alternatives, and the variables `setvars=choose1-choose&n` contain the numbers of the chosen alternatives. The output data set `Res2` has 21,600 observations (200 subjects who each saw 18 choice sets with 6 alternatives).

The first two choice sets are as follows:

---

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

---

## Binary Coding

One more thing must be done to these data before they can be analyzed. The binary design matrix is coded for each effect. This can be done with PROC TRANSREG as follows:

```
proc transreg design=5000 data=res2 nozeroconstant norestoremismissing;
  model class(place / zero=none order=data)
         class(price scene lodge / zero=none order=formatted) /
         lprefix=0;
  output out=coded(drop=_type_ _name_ intercept);
  id subj set form c;
run;
```

The `design` option specifies that no model is fit; the procedure is just being used to code a design. When `design` is specified, dependent variables are not required. Optionally, `design` can be followed by “=*n*” where *n* is the number of observations to process at one time. By default, PROC TRANSREG codes all observations in one big group. For very large data sets, this can consume large amounts of memory and time. Processing blocks of smaller numbers of observations is more computationally efficient. The option `design=5000` processes observations in blocks of 5000. For smaller computers, try something like `design=1000`.

The `nozeroconstant` and `norestoremismissing` options are not necessary for this example but are included here because sometimes they are very helpful in coding choice models. The `nozeroconstant` option specifies that if the coding creates a constant variable, it should not be zeroed. The `nozeroconstant` option should always be specified when you specify `design=n` because the last group of observations might be small and might contain constant variables. The `nozeroconstant` option is also important if you do something like coding by `subj set` because sometimes an attribute is constant within a



choice set. The `norestoremissing` option specifies that missing values should not be restored when the `out=` data set is created. By default, the coded `class` variable contains a row of missing values for observations in which the `class` variable is missing. With the `norestoremissing` option, these observations contain a row of zeros instead. This option is useful when there is a constant alternative indicated by missing values. Both of these options, like almost all options in PROC TRANSREG, can be abbreviated to three characters (`noz` and `nor`).

The `model` statement names the variables to code and provides information about how they should be coded. The specification `class(place / ...)` specifies that the variable `Place` is a classification variable and requests a binary coding. The `zero=none` option creates binary variables for all categories. The `order=data` option sorts the levels into the order they are encountered in the data set. It is specified so 'Home' is the last destination in the analysis, as it is in the data set. The `class(price scene lodge / ...)` specification names the variables `Price`, `Scene`, and `Lodge` as categorical variables and creates binary variables for all of the levels of all of the variables. The levels are sorted into order based on their formatted values. The `lprefix=0` option specifies that when labels are created for the binary variables, zero characters of the original variable name should be used as a prefix. This means that the labels are created only from the level values. For example, 'Mountains' and 'Bed & Breakfast' are created as labels not 'Scene Mountains' and 'Lodge Bed & Breakfast'.

An `output` statement names the output data set and drops variables that are not needed. These variables do not have to be dropped. However, since they are variable names that are often found in special data set types, PROC PHREG displays warnings when it finds them. Dropping the variables prevents the warnings. Finally, the `id` statement names the additional variables that we want copied from the input to the output data set. The following steps display 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;
```

The results are as follows:

---

Vacation Example

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

Vacation Example

Obs	Hawaii	Alaska	Mexico	California	Maine	Home	Place
1	1	0	0	0	0	0	Hawaii
2	0	1	0	0	0	0	Alaska
3	0	0	1	0	0	0	Mexico
4	0	0	0	1	0	0	California
5	0	0	0	0	1	0	Maine
6	0	0	0	0	0	1	Home

Vacation Example

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

Vacation Example

Place	Bed & Breakfast	Cabin	Home	Hotel	Lodge	0	999	1249	1499	Price
Hawaii	0	1	0	0	Cabin	0	1	0	0	999
Alaska	0	0	0	1	Hotel	0	1	0	0	999
Mexico	0	0	0	1	Hotel	0	0	1	0	1249
California	0	1	0	0	Cabin	0	0	1	0	1249
Maine	0	1	0	0	Cabin	0	0	1	0	1249
Home	0	0	1	0	Home	1	0	0	0	0

---

The coded design consists of binary variables for destinations Hawaii — Home, scenery Beach — Mountains, lodging Bed & Breakfast — Hotel, and price 0 — 1499. For example, in the last panel of the first choice set, the Cabin column has a 1 for Hawaii since Hawaii has Cabin lodging in this choice set. The Cabin column has a 0 for Alaska since Alaska does not have Cabin lodging in this choice set. These binary variables form the independent variables in the analysis.

Note that we are fitting a model with *generic attributes*. Generic attributes are assumed to be the same for all alternatives. For example, our model is structured so that the part-worth utility for being on a lake is the same for Hawaii, Alaska, and all of the other destinations. Similarly, the part-worth utilities for the different prices do not depend on the destinations. In contrast, on page 386, using the same data, we code alternative-specific effects where the part-worth utilities are allowed by the model to be different for each of the destinations.

PROC PHREG is run to fit the choice model as follows:

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

We specify the `&_trgind` macro variable for the `model` statement independent variable list. PROC TRANSREG automatically creates this macro variable. It contains the list of coded independent variables generated by the procedure. This is so you do not have to figure out what names TRANSREG created and specify them. In this case, PROC TRANSREG sets `&_trgind` to contain the following list:

```
PlaceHawaii PlaceAlaska PlaceMexico PlaceCalifornia PlaceMaine PlaceHome
Price0 Price999 Price1249 Price1499 SceneBeach SceneHome SceneLake
SceneMountains LodgeBed___Breakfast LodgeCabin LodgeHome LodgeHotel
```

The analysis is stratified by subject and choice set. Each stratum consists of a set of alternatives from which a subject made one choice. In this example, each stratum consists of six alternatives, one of which is chosen and five of which are not chosen. (Recall that we specified `%phchoice(on)` on page 287 to customize the output from PROC PHREG.) The full table of the strata is quite large with one line for each of the 3600 strata, so the `brief` option is specified in the PROC PHREG statement. This option produces a brief summary of the strata. In this case, we see there are 3600 choice sets that all fit one response pattern. Each consisted of 6 alternatives, 1 of which is chosen and 5 of which are not chosen. There should be one pattern for all choice sets in an example like this one—the number of alternatives, number of chosen alternatives, and the number not chosen should be constant.

The results are as follows:

---

Vacation Example

The PHREG Procedure

Model Information

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

Number of Observations Read	21600
Number of Observations Used	21600

Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Pattern	Number of Choices	Number of Alternatives	Chosen Alternatives	Not Chosen
1	3600	6	1	5

Convergence Status

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

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	12900.668	6257.752
AIC	12900.668	6279.752
SBC	12900.668	6347.827

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	6642.9164	11	<.0001
Score	5858.3798	11	<.0001
Wald	2482.5118	11	<.0001

## Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Hawaii	1	3.50429	0.45819	58.4934	<.0001
Alaska	1	0.62029	0.46624	1.7699	0.1834
Mexico	1	2.81487	0.45955	37.5193	<.0001
California	1	2.13549	0.46027	21.5263	<.0001
Maine	1	1.53470	0.46220	11.0253	0.0009
Home	0	0	.	.	.
0	0	0	.	.	.
999	1	3.56656	0.08849	1624.2978	<.0001
1249	1	1.40145	0.08293	285.6189	<.0001
1499	0	0	.	.	.
Beach	1	1.34191	0.06410	438.2880	<.0001
Home	0	0	.	.	.
Lake	1	0.67993	0.06981	94.8542	<.0001
Mountains	0	0	.	.	.
Bed & Breakfast	1	0.64972	0.05363	146.7874	<.0001
Cabin	1	-1.41463	0.07581	348.1654	<.0001
Home	0	0	.	.	.
Hotel	0	0	.	.	.

The destinations, from most preferred to least preferred, are Hawaii, Mexico, California, Maine, Alaska, and then stay at home. The utility for lower price is greater than the utility for higher price. The beach is preferred over a lake, which is preferred over the mountains. A bed & breakfast is preferred over a hotel, which is preferred over a cabin. Notice that the coefficients for the constant alternative (home and zero price) are all zero. Also notice that for each factor, destination, price, scenery and accommodations, the coefficient for the last level is always zero. This always occurs when we code with `zero=none`. The last level of each factor is a reference level, and the other coefficients have values relative to this zero. For example, all of the coefficients for the destination are positive relative to the zero for staying at home. For scenery, all of the coefficients are positive relative to the zero for the mountains. For accommodations, the coefficient for cabin is less than the zero for hotel, which is less than the coefficient for bed & breakfast. In some sense, each `class` variable in a choice model with a constant alternative has levels that 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 eliminate these levels from the output.

## Quantitative Price Effect

These data can also be analyzed in a different way. The `Price` variable can be specified directly as a quantitative variable, instead of with indicator variables for a qualitative price effect. You could display the independent variable list and copy and edit it, removing the `Price` indicator variables and adding `Price`.

The following statement displays the list:

```
%put &_trgind;
```

Alternatively, you could run PROC TRANSREG again with the new coding. We use this latter approach, because it is easier, and it lets us illustrate other options. In the previous analysis, there are a number of structural zeros in the parameter estimate results due to the usage of the `zero=none` option in the PROC TRANSREG coding. This is a good thing, particularly for a first attempt at the analysis. It is good to specify `zero=none` and check the results and make sure you have the right pattern of zeros and nonzeros. Later, you can run again excluding some of the structural zeros. This time, we explicitly specify the 'Home' level in the `zero=` option as the reference level so it is omitted from the `&_trgind` variable list. The first `class` specification specifies `zero='Home'` since there is one variable. The second `class` specification specifies `zero='Home'` 'Home' specifying the reference level for each of the two variables. The variable `Price` is designated as an `identity` variable. The `identity` transformation is the no-transformation option, which is used for variables that need to enter the model with no further manipulations. The `identity` variables are simply copied into the output data set and added to the `&_trgind` variable list. The statement `label price = 'Price'` is specified to explicitly set a label for the `identity` variable `price`. This is because we explicitly modified PROC PHREG output using `%phchoice(on)` so that the rows of the parameter estimate table are 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. The following steps perform the coding and the analysis:

```
proc transreg design data=res2 nozeroconstant norestoremisning;
  model class(place / zero='Home' order=data) identity(price)
         class(scene lodge / zero='Home' 'Home' order=formatted) /
         lprefix=0;
  output out=coded(drop=_type_ _name_ intercept);
  label price = 'Price';
  id subj set form c;
  run;

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

The results are as follows:

---

#### Vacation Example

#### The PHREG Procedure

#### Model Information

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

Number of Observations Read 21600  
 Number of Observations Used 21600

## Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Pattern	Number of Choices	Number of Alternatives	Chosen Alternatives	Not Chosen
1	3600	6	1	5

## Convergence Status

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

## Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	12900.668	6295.152
AIC	12900.668	6315.152
SBC	12900.668	6377.039

## Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	6605.5164	10	<.0001
Score	5750.9220	10	<.0001
Wald	2483.9241	10	<.0001

## Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Hawaii	1	14.27118	0.50198	808.2623	<.0001
Alaska	1	11.44532	0.49063	544.1855	<.0001
Mexico	1	13.56216	0.49955	737.0457	<.0001
California	1	12.94025	0.49430	685.3359	<.0001
Maine	1	12.36405	0.49553	622.5618	<.0001
Price	1	-0.00740	0.0001770	1747.2333	<.0001
Beach	1	1.33978	0.06458	430.4561	<.0001
Lake	1	0.71161	0.07131	99.5777	<.0001
Mountains	0	0	.	.	.
Bed & Breakfast	1	0.66233	0.05319	155.0604	<.0001
Cabin	1	-1.33467	0.07353	329.4356	<.0001
Hotel	0	0	.	.	.

---

The results of the two different analyses are similar. The coefficients for the destinations all increase by a nonconstant amount (approximately 10.8) but the pattern is the same. There is still a negative effect for price. Also, the fit of this model is slightly worse, Chi-Square = 6605.5164, compared to the previous value of 6642.9164 (bigger values mean better fit), because price has one fewer parameter.

## Quadratic Price Effect

Previously, we saw price treated as a qualitative factor with two parameters and two *df*, then we saw price treated as a quantitative factor with one parameter and one *df*. Alternatively, we could treat price as quantitative and add a *quadratic* price effect (price squared). Like treating price as a qualitative factor, there are two parameters and two *df* for price. First, we create `PriceL`, the linear price term by centering the original price and dividing by the price increment (250). This maps (999, 1249, 1499) to (-1, 0, 1). Next, we run PROC TRANSREG and PROC PHREG with the new price variables as follows:

```
data res3;
  set res2;
  PriceL = price;
  if price then pricel = (price - 1249) / 250;
run;

proc transreg design=5000 data=res3 nozeroconstant norestoremisning;
  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=` is specified) simply expands the variable into a quadratic polynomial.

The following step fits the model:

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



This step produced the following results:

Vacation Example

The PHREG Procedure

Model Information

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

Number of Observations Read	21600
Number of Observations Used	21600

Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Pattern	Number of Choices	Number of Alternatives	Chosen Alternatives	Not Chosen
1	3600	6	1	5

Convergence Status

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

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	12900.668	6257.752
AIC	12900.668	6279.752
SBC	12900.668	6347.827

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	6642.9164	11	<.0001
Score	5858.3798	11	<.0001
Wald	2482.5118	11	<.0001

## Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Hawaii	1	4.90574	0.45379	116.8713	<.0001
Alaska	1	2.02174	0.46010	19.3081	<.0001
Mexico	1	4.21633	0.45427	86.1476	<.0001
California	1	3.53695	0.45507	60.4085	<.0001
Maine	1	2.93615	0.45761	41.1683	<.0001
Price 1	1	-1.78328	0.04425	1624.2978	<.0001
Price 2	1	0.38183	0.06263	37.1732	<.0001
Beach	1	1.34191	0.06410	438.2880	<.0001
Lake	1	0.67993	0.06981	94.8542	<.0001
Mountains	0	0	.	.	.
Bed & Breakfast	1	0.64972	0.05363	146.7874	<.0001
Cabin	1	-1.41463	0.07581	348.1654	<.0001
Hotel	0	0	.	.	.

The fit is exactly the same as when price is treated as qualitative, Chi-Square = 6642.9164. This is because both models are the same except for the different but equivalent 2 *df* codings of price. The coefficients for the destinations in the two models differ by a constant 1.40145. The coefficients for the factors after price are unchanged. The part-worth utility for \$999 is  $-1.78328 \times (999 - 1249)/250 + 0.38183 \times ((999 - 1249)/250)^2 = 2.16511$ , the part-worth utility for \$1249 is  $-1.78328 \times (1249 - 1249)/250 + 0.38183 \times ((1249 - 1249)/250)^2 = 0$ , and the part-worth utility for \$1499 is  $-1.78328 \times (1499 - 1249)/250 + 0.38183 \times ((1499 - 1249)/250)^2 = -1.40145$ , which differ from the coefficients when price is treated as qualitative, by a constant  $-1.40145$ .

## Effects Coding

In the previous analyses, *binary* (1, 0) codings are 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.

Levels	Binary Coding		Effects Coding	
	One	Two	One	Two
1	1	0	1	0
2	0	1	0	1
3	0	0	-1	-1

In this example, we 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 376, a few select lines of which are reproduced next:

---

(Some Lines in the)  
Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Home	0	0	.	.	.
0	0	0	.	.	.
1499	0	0	.	.	.
Home	0	0	.	.	.
Mountains	0	0	.	.	.
Home	0	0	.	.	.
Hotel	0	0	.	.	.

---

Notice that the coefficients for the constant alternative (home and zero price) are all zero. Also notice that for each factor, destination, price, scenery and accommodations, the coefficient for the last level is always zero. In some of the preceding examples, we eliminated the 'Home' levels by specifying `zero=Home`. Next 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 causes PROC TRANSREG to ignore those levels when constructing indicator variables. When you use this strategy, you must specify the `norestoremis` option in the PROC TRANSREG statement. During the first stage of design matrix creation, PROC TRANSREG puts zeros in the indicator variables for observations with missing `class` levels. At the end, it replaces the zeros with missings, "restoring the missing values." When the `norestoremis` option is specified, missing values are not restored and we get zeros in the indicator variables for missing `class` levels, which is usually what we want. The DATA step `if` statements recode the constant levels to missing. Next, in PROC TRANSREG, the reference levels 'Mountains' and 'Hotel' are listed in the `zero=` option in the `class` specification as follows:

```
data res4;
  set res3;
  if scene = 0 then scene = .;
  if lodge = 0 then lodge = .;
run;
```

```

proc transreg design=5000 data=res4 nozeroconstant norestoremismissing;
  model class(place / zero='Home' order=data)
    pspline(pricel / degree=2)
    class(scene lodge /
      effects zero='Mountains' 'Hotel' order=formatted) /
    lprefix=0;
  output out=coded(drop=_type_ _name_ intercept);
  label pricel = 'Price';
  id subj set form c;
run;

```

Next, the coded data and design matrix are displayed 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. The following step displays the lodging terms:

```
proc print data=coded(obs=6) label; run;
```

The results are as follows:

---

Vacation Example											
Obs	Hawaii	Alaska	Mexico	California	Maine	Price 1	Price 2	Beach	Lake	Bed & Breakfast	Cabin
1	1	0	0	0	0	-1	1	1	0	0	1
2	0	1	0	0	0	-1	1	-1	-1	-1	-1
3	0	0	1	0	0	0	0	1	0	-1	-1
4	0	0	0	1	0	0	0	0	1	0	1
5	0	0	0	0	1	0	0	-1	-1	0	1
6	0	0	0	0	0	0	0	0	0	0	0

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

---

The following step fits the choice model:

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

The results are as follows:

---

```

                                Vacation Example

                                The PHREG Procedure

                                Model Information

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

                                Number of Observations Read      21600
                                Number of Observations Used      21600

Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Pattern      Number of      Number of      Chosen      Not
              Choices      Alternatives  Alternatives  Chosen
-----
              1          3600          6          1          5

                                Convergence Status

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

                                Model Fit Statistics

                                Criterion      Without      With
                                Covariates      Covariates

                                -2 LOG L      12900.668    6257.752
                                AIC            12900.668    6279.752
                                SBC            12900.668    6347.827

                                Testing Global Null Hypothesis: BETA=0

Test          Chi-Square      DF      Pr > ChiSq
-----
Likelihood Ratio      6642.9164      11      <.0001
Score                5858.3798      11      <.0001
Wald                  2482.5118      11      <.0001
```

## Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Hawaii	1	5.32472	0.44985	140.1076	<.0001
Alaska	1	2.44072	0.45690	28.5360	<.0001
Mexico	1	4.63531	0.45052	105.8613	<.0001
California	1	3.95593	0.45176	76.6803	<.0001
Maine	1	3.35513	0.45381	54.6610	<.0001
Price 1	1	-1.78328	0.04425	1624.2978	<.0001
Price 2	1	0.38183	0.06263	37.1732	<.0001
Beach	1	0.66796	0.03582	347.7320	<.0001
Lake	1	0.00599	0.03922	0.0233	0.8787
Bed & Breakfast	1	0.90469	0.03471	679.2342	<.0001
Cabin	1	-1.15966	0.04650	621.9367	<.0001

It is instructive to compare the results of this analysis to the previous analysis on page 381. First, the model fit and chi-square statistics are the same indicating the models are equivalent. The coefficients for the destinations differ by a constant  $-0.41898$ , the price coefficients are the same, the scenery coefficients differ by a constant  $0.67395$ , and the lodging coefficients differ by a constant  $-0.25497$ . Notice that  $-0.41898 + 0 + 0.67395 + -0.25497 = 0$ , so the utility for each alternative is unchanged by the different but equivalent codings.

## Alternative-Specific Effects

In all of the analyses presented so far in this example, we have assumed that the effects for price, scenery, and accommodations are generic or constant across the different destinations. Equivalently, we assumed that destination does not interact with the attributes. Next, we show a model with *alternative-specific effects* that does not make this assumption. The alternative-specific model allows for different price, scenery and lodging effects for each destination. The coding can be done with PROC TRANSREG using its syntax for interactions. Before we do the coding, let's go back to the design preparation stage and redo it in a more normal fashion so reference levels are omitted from the analysis.

We start by creating the data set Key as follows:

```
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
.           .   .   .
;
```

This step differs from the one we saw on page 356 only in that now we have a missing value for Place for the constant alternative. Next, we use the %MktRoll macro to process the design and the %MktMerge macro to merge the design and data as follows:

```
%mktroll(design=sasuser.VacationLinDesBlckd, key=key, alt=place,
         out=sasuser.VacationChDes)

%mktmerge(design=sasuser.VacationChDes, 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 356. The %MktMerge macro usage differs from page 371 in that instead of assigning labels and recoding price in a separate DATA step, we now do it directly in the macro. The stmts= option is used to add a price = assignment statement and format statement to the DATA step that merges the two data sets. The statements are included in a %str( ) macro since they contain semicolons. The first two choice sets are as follows:

---

Vacation Example

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

---

Notice that the attributes for the constant alternative are all missing. Next, we code with PROC TRANSREG. Since we are using missing values for the constant alternative, we must specify the `norestoremissing` option in the PROC TRANSREG statement. With the `norestoremissing` option, the indicator variables created for missing class variable values contain all zeros instead of all missings. First, we specify the variable Place as a class variable.

Next, we interact `Place` with all of the attributes, `Price`, `Scene`, and `Lodge`, to create the alternative-specific effects as follows:

```
proc transreg design=5000 data=res2 nozeroconstant norestoremissing;
  model class(place / zero=none order=data)
    class(place * price place * scene place * lodge /
      zero=none order=formatted) / lprefix=0 sep=' ', ';
  output out=coded(drop=_type_ _name_ intercept);
  id subj set form c;
run;

proc print data=coded(obs=6) label noobs; run;
```

The coded design matrix consists of:

- five binary columns, 'Hawaii' through 'Maine', for the five destinations,
- fifteen binary columns (5 destinations times 3 prices), 'Alaska, 999' through 'Mexico, 1499', for the alternative-specific price effects,
- fifteen binary columns (5 destinations times 3 sceneries), 'Alaska, Beach' through 'Mexico, Mountains', for the alternative-specific scenery effects,
- fifteen binary columns (5 destinations times 3 lodgings), 'Alaska, Bed & Breakfast' through 'Mexico, Hotel', for the alternative-specific lodging effects.

The entire sixth row of the coded design matrix, the stay-at-home alternative, consists of zeros. The results are as follows:

---

#### Vacation Example

Hawaii	Alaska	Mexico	California	Maine	Alaska, 999	Alaska, 1249	Alaska, 1499
1	0	0	0	0	0	0	0
0	1	0	0	0	1	0	0
0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0

California, 999	California, 1249	California, 1499	Hawaii, 999	Hawaii, 1249	Hawaii, 1499
0	0	0	1	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0



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

California, Beach	California, Lake	California, Mountains	Hawaii, Beach	Hawaii, Lake	Hawaii, Mountains
0	0	0	1	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Maine, Beach	Maine, Lake	Maine, Mountains	Mexico, Beach	Mexico, Lake	Mexico, Mountains	Alaska, Bed & Breakfast	Alaska, Cabin	Alaska, Hotel
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

California, Bed & Breakfast	California, Cabin	California, Hotel	Hawaii, Bed & Breakfast	Hawaii, Cabin	Hawaii, Hotel
0	0	0	0	1	0
0	0	0	0	0	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

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

---

Analysis proceeds by running PROC PHREG as follows:

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

The results are as follows:

---

#### Vacation Example

#### The PHREG Procedure

#### Model Information

Data Set	WORK.CODED
Dependent Variable	c
Censoring Variable	c
Censoring Value(s)	2
Ties Handling	BRESLOW
Number of Observations Read	21600
Number of Observations Used	21600

## Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Pattern	Number of Choices	Number of Alternatives	Chosen Alternatives	Not Chosen
1	3600	6	1	5

## Convergence Status

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

## Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	12900.668	6239.870
AIC	12900.668	6309.870
SBC	12900.668	6526.474

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	6660.7982	35	<.0001
Score	6601.7928	35	<.0001
Wald	2448.1475	35	<.0001

## Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Hawaii	1	3.49208	0.47222	54.6856	<.0001
Alaska	1	0.17527	0.67139	0.0682	0.7940
Mexico	1	2.93013	0.47932	37.3706	<.0001
California	1	2.17915	0.49725	19.2058	<.0001
Maine	1	1.27770	0.54587	5.4787	0.0192
Alaska, 999	1	4.02423	0.46534	74.7858	<.0001
Alaska, 1249	1	1.81200	0.49473	13.4147	0.0002
Alaska, 1499	0	0	.	.	.
California, 999	1	3.38438	0.19965	287.3498	<.0001
California, 1249	1	1.22372	0.22445	29.7251	<.0001
California, 1499	0	0	.	.	.
Hawaii, 999	1	3.61016	0.14157	650.2879	<.0001
Hawaii, 1249	1	1.45415	0.13050	124.1725	<.0001
Hawaii, 1499	0	0	.	.	.
Maine, 999	1	3.80918	0.26060	213.6577	<.0001
Maine, 1249	1	1.53370	0.27050	32.1475	<.0001
Maine, 1499	0	0	.	.	.

Mexico, 999	1	3.45924	0.15495	498.4209	<.0001
Mexico, 1249	1	1.41406	0.15693	81.1907	<.0001
Mexico, 1499	0	0	.	.	.
Alaska, Beach	1	1.01542	0.21881	21.5355	<.0001
Alaska, Lake	1	0.48168	0.22639	4.5269	0.0334
Alaska, Mountains	0	0	.	.	.
California, Beach	1	1.47681	0.15536	90.3528	<.0001
California, Lake	1	0.84358	0.16138	27.3244	<.0001
California, Mountains	0	0	.	.	.
Hawaii, Beach	1	1.29573	0.12493	107.5692	<.0001
Hawaii, Lake	1	0.61301	0.12299	24.8444	<.0001
Hawaii, Mountains	0	0	.	.	.
Maine, Beach	1	1.59739	0.20874	58.5584	<.0001
Maine, Lake	1	0.64984	0.20203	10.3468	0.0013
Maine, Mountains	0	0	.	.	.
Mexico, Beach	1	1.26780	0.13744	85.0857	<.0001
Mexico, Lake	1	0.67632	0.13589	24.7716	<.0001
Mexico, Mountains	0	0	.	.	.
Alaska, Bed & Breakfast	1	1.00195	0.18862	28.2169	<.0001
Alaska, Cabin	1	-1.33747	0.31958	17.5146	<.0001
Alaska, Hotel	0	0	.	.	.
California, Bed & Breakfast	1	0.67004	0.13195	25.7875	<.0001
California, Cabin	1	-1.50239	0.16734	80.6060	<.0001
California, Hotel	0	0	.	.	.
Hawaii, Bed & Breakfast	1	0.63585	0.11523	30.4508	<.0001
Hawaii, Cabin	1	-1.41004	0.13462	109.7155	<.0001
Hawaii, Hotel	0	0	.	.	.
Maine, Bed & Breakfast	1	0.58532	0.15999	13.3848	0.0003
Maine, Cabin	1	-1.50967	0.22377	45.5166	<.0001
Maine, Hotel	0	0	.	.	.
Mexico, Bed & Breakfast	1	0.54835	0.11802	21.5891	<.0001
Mexico, Cabin	1	-1.40762	0.15033	87.6707	<.0001
Mexico, Hotel	0	0	.	.	.

There are zero coefficients for the reference level. Do we need this more complicated model instead of the simpler model? To answer this, first look at the coefficients. Are they similar across different destinations? In this case, they seem to be. This suggests that the simpler model might be sufficient.

More formally, the two models can be statistically compared. You can test the null hypothesis that the two models are not significantly different by comparing their likelihoods. The difference between two  $-2\log(\mathcal{L}_C)$ 's (the number reported under 'With Covariates' in the output) has a chi-square distribution. We can get the  $df$  for the test by subtracting the two  $df$  for the two likelihoods. The difference  $6257.752 - 6239.870 = 17.882$  is distributed  $\chi^2$  with  $35 - 11 = 24$   $df$  ( $p < 0.80869$ ). This more complicated model does not account for significantly more variance than the simpler model.

## Vacation Example and Artificial Data Generation

This example does not illustrate any new capabilities. Rather, it shows how to make artificial data to test your design and your code for processing and analyzing the real data before the data are collected. This example goes into more detail about how to create artificial data than the other artificial data examples in this chapter. The artificial data creation step assumes that the design is in the final choice design format (not in the linear arrangement that comes from the `%MktEx` macro).

You begin by making a design. This example is based on the vacation example on page 339. The goal is to make an alternative-specific design for vacation destinations. The choice design has a destination attribute along with three other attributes, accommodations, scenery, and price. The design can be constructed from a linear arrangement with the following factors:

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

The design additionally has a constant “stay at home” alternative.

The `%MktEx` macro can be used as follows to make a linear arrangement in 36 runs, which becomes a choice design with 36 choice sets:

```
%mktex(3 ** 15,          /* 15 three-level factors          */
        n=36,           /* 36 rows in linear arrang - 36 ch sets*/
        seed=205)      /* random number seed              */
```

The linear arrangement is blocked into two blocks of size 18 as follows:

```
%mktblock(data=randomized, /* block randomized design          */
            nblocks=2,     /* create two blocks of 18 choice sets */
            out=blocked,  /* output data set for blocked design */
            seed=114)     /* random number seed              */
```

The rules for creating a choice design from a linear arrangement are stored in the following data set:

```

                                /* make Place from the destinations    */
                                /* make Lodge from x1-x5                */
                                /* make Scene from x6-x10              */
                                /* make Price from x11-x15             */
                                /* make 'stay at home' from all missing */
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
.           .   .   .
;

```

The following step creates the choice design and stores it in a permanent SAS data set:

```

%mktrroll(design=blocked,          /* make choice design from blocked    */
          key=key,                 /* linear arrangement from %mktblock  */
          alt=place,               /* use rules in KEY data set          */
          out=sasuser.ChoiceDesign, /* alternative name variable is Place */
          options=nowarn,          /* permanent data set for results     */
          keep=block)              /* don't warn about extra variables   */
                                /* keep the blocking variable         */

```

There are many other ways you could make a choice design for this problem. You could also search a candidate set of choice sets or alternatives with the %ChoiceEff macro. For this example, it does not matter. You can generate artificial data for a choice design stored in choice design format (one row per alternative) no matter how it was created.

The following steps assign formats to the levels of the attributes:

```

proc format;                      /* map 1, 2, 3 levels to actual levels */
  value price 1 = '1499'          2 = '1749'          3 = '1999' . = ' ';
  value scene 1 = 'Mountains'    2 = 'Lake'          3 = 'Beach' . = ' ';
  value lodge 1 = 'Cabin'        2 = 'Bed & Breakfast' 3 = 'Hotel' . = ' ';
run;

data sasuser.ChoiceDesign;        /* assign formats to vars in the design */
  set sasuser.ChoiceDesign;
  format scene scene. lodge lodge. price price.;
run;

```

The choice design data set contains the variable Place, which is a character variable with levels: 'Hawaii', 'Alaska', 'Mexico', 'California', 'Maine', and ' ' (for stay at home). The variables Lodge, Scene, and Price are numeric variables with values 1, 2, and 3 and with formatted values that are more descriptive.

The following step displays the choice design:

```
proc print data=sasuser.ChoiceDesign; /* display sets and check results */
  by block set; id block set;
run;
```

It is important to display the choice design and ensure that it is correct. Some of the results are as follows:

---

Block	Set	Place	Lodge	Scene	Price
1	1	Hawaii	Cabin	Beach	1999
		Alaska	Hotel	Lake	1999
		Mexico	Cabin	Lake	1499
		California	Cabin	Lake	1749
		Maine	Hotel	Beach	1999
1	2	Hawaii	Cabin	Mountains	1499
		Alaska	Bed & Breakfast	Mountains	1999
		Mexico	Bed & Breakfast	Mountains	1999
		California	Hotel	Lake	1749
		Maine	Hotel	Mountains	1749
.					
.					
.					
1	18	Hawaii	Bed & Breakfast	Lake	1999
		Alaska	Hotel	Beach	1749
		Mexico	Cabin	Lake	1999
		California	Bed & Breakfast	Beach	1749
		Maine	Cabin	Lake	1749
.					
.					
.					
2	36	Hawaii	Bed & Breakfast	Lake	1499
		Alaska	Bed & Breakfast	Lake	1999
		Mexico	Hotel	Mountains	1749
		California	Cabin	Beach	1499
		Maine	Cabin	Beach	1999

---

The following step evaluates the choice design by using the %ChoiEff macro:

```

                                /* Evaluate the choice design          */
%choiceff(data=sasuser.ChoiceDesign,/* candidate set of choice sets    */
          init=sasuser.ChoiceDesign(keep=set), /* select these sets          */
          intiter=0, /* evaluate without internal iterations */
                                /* alternative-specific effects model */
                                /* zero=none - no ref levels for place */
                                /* order=data - do not sort levels      */
          model=class(place / zero=none order=data)
                                /* zero=' ' - no ref level for first */
                                /* factor (place), ordinary ref levels */
                                /* for other factors (price -- lodge). */
                                /* order=formatted - sort levels      */
                                /* use blank sep to build interact terms*/
          class(place * price place * scene place * lodge /
                zero=' ' order=formatted separators=''' ') /
                                /* no ref level for place          */
                                /* use blank sep to build interact terms*/
          lprefix=0 /* lpr=0 labels created from just levels*/
          cprefix=0, /* cpr=0 names created from just levels */
          nsets=72, /* number of choice sets          */
          nalts=6, /* number of alternatives          */
          beta=zero) /* assumed beta vector, Ho: b=0    */

```

The list of parameters, variances, and standard errors are as follows:

---

n	Variable Name	Label	Variance	DF	Standard Error
1	Hawaii	Hawaii	0.76667	1	0.87560
2	Alaska	Alaska	0.76667	1	0.87560
3	Mexico	Mexico	0.76717	1	0.87588
4	California	California	0.86667	1	0.93095
5	Maine	Maine	0.76717	1	0.87588
6	Alaska_1499	Alaska 1499	0.60000	1	0.77460
7	Alaska_1749	Alaska 1749	0.60000	1	0.77460
8	California_1499	California 1499	0.80000	1	0.89443
9	California_1749	California 1749	0.60000	1	0.77460
10	Hawaii_1499	Hawaii 1499	0.60000	1	0.77460
11	Hawaii_1749	Hawaii 1749	0.60000	1	0.77460
12	Maine_1499	Maine 1499	0.60150	1	0.77557
13	Maine_1749	Maine 1749	0.60150	1	0.77557
14	Mexico_1499	Mexico 1499	0.60150	1	0.77557
15	Mexico_1749	Mexico 1749	0.60150	1	0.77557
16	AlaskaBeach	Alaska Beach	0.60000	1	0.77460
17	AlaskaLake	Alaska Lake	0.60000	1	0.77460
18	CaliforniaBeach	California Beach	0.60000	1	0.77460
19	CaliforniaLake	California Lake	0.60000	1	0.77460



20	HawaiiBeach	Hawaii Beach	0.60000	1	0.77460
21	HawaiiLake	Hawaii Lake	0.60000	1	0.77460
22	MaineBeach	Maine Beach	0.60000	1	0.77460
23	MaineLake	Maine Lake	0.60000	1	0.77460
24	MexicoBeach	Mexico Beach	0.60000	1	0.77460
25	MexicoLake	Mexico Lake	0.60000	1	0.77460
26	AlaskaBed___Breakfast	Alaska Bed & Breakfast	0.60000	1	0.77460
27	AlaskaCabin	Alaska Cabin	0.60000	1	0.77460
28	CaliforniaBed___Breakfast	California Bed & Breakfast	0.60000	1	0.77460
29	CaliforniaCabin	California Cabin	0.60000	1	0.77460
30	HawaiiBed___Breakfast	Hawaii Bed & Breakfast	0.60000	1	0.77460
31	HawaiiCabin	Hawaii Cabin	0.60000	1	0.77460
32	MaineBed___Breakfast	Maine Bed & Breakfast	0.60000	1	0.77460
33	MaineCabin	Maine Cabin	0.60000	1	0.77460
34	MexicoBed___Breakfast	Mexico Bed & Breakfast	0.60000	1	0.77460
35	MexicoCabin	Mexico Cabin	0.60000	1	0.77460
				==	
				35	

---

The next step, now that the choice design has been constructed and evaluated with the %ChoiceEff macro, is making artificial data for some fictitious subjects. Before you do that, however, you should study the next three examples which explain some of the inner workings of the SAS DATA step. This background makes the data generation program clearer. The first program is a standard DATA step that creates a copy of the choice design. It is not a model for what you will do. Rather, it provides a standard and familiar DATA step program, which you can compare to what you will actually do. The following step creates a copy of a data set:

```

data Test;          /* Make one copy of the input SAS data set.          */
                   /* DATA step does automatic looping, it automatically  */
                   /* writes out the observation to a SAS data set, and it */
                   /* automatically stops when it hits the end of file on */
                   /* the input SAS data set. With 216 observations in   */
                   /* the input SAS data set, there are 216 passes through */
                   /* the DATA step, and since there is no OUTPUT      */
                   /* statement, each observation is automatically written */
                   /* to the output SAS data set by an implicit OUTPUT.  */
set sasuser.ChoiceDesign;
run;

```

In the preceding step, SAS does most of the work for you. You can use this type of DATA step when there is a one to one correspondence between the input and the output. The DATA step that makes the artificial data does not have this one to one correspondence.

The following step creates a copy of the design data set, but this time, you must program all of the looping yourself:

```

data Test;          /* Make one copy of the input SAS data set.  There is */
                   /* one pass through this DATA step, and the DO loop */
                   /* reads 216 observations from the input SAS data set. */

                   /* Do the looping yourself. */
do i = 1 to 216; /* 36 choice sets and 6 alternatives = 216 observations */

                   /* SET statement with POINT=i reads the ith */
                   /* observation, and the variable i is automatically */
                   /* dropped from the output SAS data set. */
set sasuser.ChoiceDesign point=i;

output;           /* Write out each observation to a SAS data set. */
                 /* Without this statement, an OUTPUT statement */
                 /* implicitly appears at the end of the DATA step, */
                 /* which would have written out only the last */
                 /* observation if the STOP statement hadn't stopped the */
                 /* DATA step first. */
end;

stop;             /* Must specify STOP since it never hits the end of the */
                 /* data file (it never attempts to read past the last */
                 /* record). Infinite loop without this statement. */

run;

```

This step uses one pass through the DATA step with 216 reads of the input data set. The previous step uses 216 passes through the DATA step to perform 216 reads of the input data set. You can use a variation on this looping approach to create two copies of the output data set. Now there is no longer a one-to-one correspondence between the input data set and the output data set. The following step makes two copies of the input data set:

```

data Test;          /* Make two copies of the input SAS data set. */
                   /* There is one pass through this DATA step, the outer */
                   /* DO loop creates two copies, and the inner DO loop */
                   /* reads 216 observations from the input SAS data set */
                   /* (twice due to the outer DO loop). */

do Subject = 1 to 2; /* Two copies. */

                   /* Do the looping yourself. */
do i = 1 to 216; /* 36 choice sets and 6 alternatives = 216 obs */

                   /* SET statement reads the ith observation, and the */
                   /* variable i is automatically dropped from the output */
                   /* SAS data set. */
set sasuser.ChoiceDesign point=i;

```

```

        output;      /* Write out each observation to a SAS data set.      */
                    /* Without this statement, an OUTPUT statement      */
                    /* implicitly appears at the end of the DATA step,   */
                    /* which would have written out only the last       */
                    /* observation if the STOP statement hadn't stopped the */
                    /* DATA step first.                                */
        end;
    end;

    stop;           /* Must specify STOP since it never hits the end of the */
                    /* data file (it never attempts to read past the last */
                    /* record). Infinite loop without this statement.    */

run;

```

When making artificial data, you will use an approach similar to this last approach. For each block, for each subject, for each choice set, and for each alternative, you will read the design. Since the design is used repeatedly (1/2 of the design is used for each subject), you will read the design multiple times and write out multiple data points. There is not a one-to-one correspondence between the input design and the output data.

The step that creates the artificial data illustrates handling design variables in three different ways. **Lodge** and **Scene** are numeric variables with values 1, 2, and 3. These will work nicely for indexing arrays of hypothesized part-worth utility values. **Price** is no different from **Lodge** and **Scene**, but since **Price** is quantitative, you can use its value directly in the utility function if you choose to do so. **Place**, is a character variable with levels: 'Hawaii', 'Alaska', 'Mexico', 'California', 'Maine', and '' (for stay at home). You will want to convert those character values to integers when you assign utilities for each of the levels. The following informat does this for you:

```

                    /* Make an informat that will convert the character      */
                    /* values of the Place variable into the integers     */
                    /* 1 - 5.                                           */
proc format;
    invalue plinf 'Hawaii' = 1 'Alaska' = 2 'Mexico' = 3
                  'California' = 4 'Maine' = 5;
run;

```



```

/* put /          - go to a new line                                */
/* blocknum 3.   - write the block number in 3 columns             */
/* +2            - skip two columns                                */
/* subject 3.    - write the subject number in 3 columns          */
/* +2            - skip two columns                                */
/* @@;           - hold the output line for the data yet to come  */
put / blocknum 3. +2 subject 3. +2 @@;

do SetNum = 1 to 18; /* Loop over the 18 sets in a block          */
  do Alt = 1 to 6; /* Loop over the 6 alts in a set              */

    i + 1;          /* same as i = i + 1; design index,          */
                   /* i = 1 to 216 (2 blocks x 18 sets x 6 alts) */

                   /* Read the ith observation of the choice          */
                   /* design. Note that you are reading the          */
                   /* choice design not the linear arrangement.    */
                   /* Just read in the variables that you need.  */
set sasuser.ChoiceDesign(keep=place--price) point=i;

if place ne ' ' /* process the destinations differently          */
  then do; /* from the constant 'stay at home'                  */
  p = input(place, plinf.); /* map place values to 1-5          */
  u[alt] = 1 + /* add 1 just for not at home          */
          dests[p] + /* util for destination          */
          scenes[scene] + /* util for scenery          */
          lodging[lodge] - /* util for lodging          */
          price; /* negative util for price          */

  if place = 'Hawaii' and /* add in Hawaii/Beach          */
    scene = 3 then /* interaction          */
    u[alt] = u[alt] + 2;

  if place = 'Maine' and /* add Maine on a lake in a          */
    scene = 2 and lodge = 1 /* cabin interaction          */
  then u[alt] = u[alt] + 1;

  end;

else u[alt] = 0; /* util for stay at home          */

u[alt] = u[alt] + 3 * normal(17); /* add error to utils          */
/* change '3' to change          */
/* the magnitude of error          */

/* Alternatively, you can create Type I Gumbel errors for          */
/* some scaling parameter b as follows:          */
/* u[alt] = u[alt] + b * log(-log(1 - uniform(104)));          */
end;

```

```

/* at this point you have gathered u1-u6 for all 6 alts      */
m = max(of u1-u6);          /* max util over alts      */

/* which one had the maximum util? do fuzzy comparison     */
if      abs(u1 - m) < 1e-4 then c = 1; /* alt 1 is chosen */
else if abs(u2 - m) < 1e-4 then c = 2; /* alt 2 is chosen */
else if abs(u3 - m) < 1e-4 then c = 3; /* alt 3 is chosen */
else if abs(u4 - m) < 1e-4 then c = 4; /* alt 4 is chosen */
else if abs(u5 - m) < 1e-4 then c = 5; /* alt 5 is chosen */
else          c = 6; /* alt 6 is chosen (home)*/

put +(-1) c @@; /* write number of chosen alt. Use '@@' to hold */
                /* the line for the rest of the data in this */
                /* block for this subject.                    */
                /* +(-1) skips forward -1 space, which is how */
                /* you move back one space                    */

end;
end;
end;
stop;          /* explicitly stop since no end of file is hit */
run;

```

You can copy and paste the results, which are displayed in the SAS log file, into the following DATA step<sup>†</sup> to input the results as you might do so when it comes time to analyze the data:

```

data results;          /* Read the input data. List input for block */
                      /* and subject. Formatted input (fields of */
                      /* size 1) for choices.                    */
input Block Subject (choose1-choose18) (1.);
datalines;
1 1 416434213415311535
2 2 115411541451441151
1 3 132455331434113144
2 4 313313244121311314
1 5 451143532541411214
2 6 311131311414411511
.
.
.
1 197 331335333114313112
2 198 111314251521211141
1 199 151351211114311131
2 200 311314311151143141
;

```

<sup>†</sup>Alternatively, you could use a FILE statement in the preceding DATA step to write the data to a file (for example, FILE 'mytestdata.dat'; after the ARRAY statements). Then you could use INFILE 'mytestdata.dat'; before the INPUT statement instead of DATALINES and in stream data after the INPUT statement. If you do this, you must ensure that you do not rerun this program after the data are collected and wipe out the data.

It is important to display the contents of the results data set to ensure that you read the input data correctly. The following step displays the data set:

```
proc print; run;                                /* make sure data match input */
```

Some of the results are as follows:

---

```

                                c c c c c c c c c
      S c c c c c c c c c h h h h h h h h h
      u h h h h h h h h h o o o o o o o o o
    B b o o o o o o o o o o o o o o o o o o
    l j o o o o o o o o o s s s s s s s s s
  0  o e s s s s s s s s s e e e e e e e e e
  b  c c e e e e e e e e e 1 1 1 1 1 1 1 1
  s  k t 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8

  1  1  1  4  1  6  4  3  4  2  1  3  4  1  5  3  1  1  5  3  5
  2  2  2  1  1  5  4  1  1  5  4  1  4  5  1  4  4  1  1  5  1
  3  1  3  1  3  2  4  5  5  3  3  1  4  3  4  1  1  3  1  4  4
  4  2  4  3  1  3  3  1  3  2  4  4  1  2  1  3  1  1  3  1  4
  5  1  5  4  5  1  1  4  3  5  3  2  5  4  1  4  1  1  2  1  4
  6  2  6  3  1  1  1  3  1  3  1  1  4  1  4  4  1  1  5  1  1
  .
  .
  .
199  1 199  1  5  1  3  5  1  2  1  1  1  1  4  3  1  1  1  3  1
200  2 200  3  1  1  3  1  4  3  1  1  1  5  1  1  4  3  1  4  1

```

---

You should ensure that there are no unexpected missing values and that the right number of observations (in this case, 2 blocks times 100 subjects) are present.

The following step merges the data and the choice design:

```
%mktmerge(design=sasuser.ChoiceDesign, /* merge the design data set */
          data=results,                /* with the results */
          out=res2,                    /* create output data set res2 */
          blocks=block,                /* name of blocking variable */
          nsets=18,                    /* number of choice sets per block */
          nalts=6,                     /* number of alternatives */
          setvars=choose1-choose18)    /* variables with the choices */
```

The following step displays part of the resulting data set:

```
proc print data=res2(obs=18);                /* display some data - sanity check */
  id block subject set;
  by block subject set;
run;
```

It is again important to look at the results before proceeding. The results are as follows:

---

Block	Subject	Set	Place	Lodge	Scene	Price	c
1	1	1	Hawaii	Cabin	Beach	1999	2
			Alaska	Hotel	Lake	1999	2
			Mexico	Cabin	Lake	1499	2
			California	Cabin	Lake	1749	1
			Maine	Hotel	Beach	1999	2
							2
1	1	2	Hawaii	Cabin	Mountains	1499	1
			Alaska	Bed & Breakfast	Mountains	1999	2
			Mexico	Bed & Breakfast	Mountains	1999	2
			California	Hotel	Lake	1749	2
			Maine	Hotel	Mountains	1749	2
							2
1	1	3	Hawaii	Hotel	Beach	1749	2
			Alaska	Bed & Breakfast	Beach	1749	2
			Mexico	Cabin	Mountains	1499	2
			California	Cabin	Beach	1499	2
			Maine	Hotel	Mountains	1499	2
							1

---

The following step codes the attributes for analysis:

```

proc transreg                      /* use proc transreg to code */
  data=res2                        /* name of data set to code */
  design=5000                      /* code big designs in chunks */
                                  /* code up to 5000 obs at a time */
  nozeroconstant                   /* don't zero constant variables */
  norestoremmissing;              /* zeros in coded vars, not missings */
model class(place /               /* code Place variable */
            zero=none             /* use all nonmissing levels */
            order=data)           /* don't sort levels */
  class(place * price             /* code other vars */
        place * scene
        place * lodge /
            zero=none             /* use all nonmissing levels */
                                  /* including 0 reference levels */
            order=formatted       /* do sort levels by formatted values*/
            separators=' ' ' ') / /* use blank separator in interacts */
  lprefix=0;                      /* make labels just from levels */
output out=coded                  /* output coded data set */
  (drop=_type_ _name_ intercept); /* drop vars that you don't need */
id subject set block c;          /* add extra vars that you do need */
run;
```



The following step customizes the output from PROC PHREG for choice modeling:

```
%phchoice( on )                /* customize output from PHREG for */
                                /* choice models                        */
```

The following step analyzes the artificial data:

```
proc phreg data=coded brief;    /* do analysis with a brief summary */
                                /* of strata (set, subject, block) */
  model c*c(2) = &_trgind / ties=breslow; /* standard choice model          */
  strata subject set block;      /* ID variables who taken together */
  run;                            /* identify each individual choice */
                                /* set                             */
```

Some of the results are as follows:

---

Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Pattern	Number of Choices	Number of Alternatives	Chosen Alternatives	Not Chosen
1	3600	6	1	5

---

It is important to note that there were 3600 choices (36 choice sets and 100 subjects per set). Each set consists of 6 alternatives—one was chosen and five were not. Each of the 3600 subject and set combinations jointly defines a separate stratum in the analysis. Any other pattern of results for this design and data set indicates an error in data collection, entry, or processing. More is said about this table at the end of this example.

The parameter estimates are as follows:

---

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Hawaii	1	2.29050	0.17219	176.9392	<.0001
Alaska	1	-1.58858	0.49315	10.3766	0.0013
Mexico	1	1.87521	0.18389	103.9896	<.0001
California	1	1.32370	0.21482	37.9705	<.0001
Maine	1	0.77912	0.23117	11.3595	0.0008
Alaska 1499	1	1.64559	0.42064	15.3050	<.0001
Alaska 1749	1	0.65152	0.44333	2.1597	0.1417
Alaska 1999	0	0	.	.	.
California 1499	1	0.97060	0.14844	42.7531	<.0001
California 1749	1	0.50921	0.13822	13.5731	0.0002
California 1999	0	0	.	.	.

Hawaii 1499	1	0.98873	0.09483	108.7181	<.0001
Hawaii 1749	1	0.49605	0.09585	26.7826	<.0001
Hawaii 1999	0	0	.	.	.
Maine 1499	1	1.19476	0.15276	61.1707	<.0001
Maine 1749	1	0.74813	0.16447	20.6919	<.0001
Maine 1999	0	0	.	.	.
Mexico 1499	1	0.91560	0.10707	73.1267	<.0001
Mexico 1749	1	0.51216	0.11346	20.3759	<.0001
Mexico 1999	0	0	.	.	.
Alaska Beach	1	1.54354	0.39272	15.4479	<.0001
Alaska Lake	1	1.27276	0.41947	9.2065	0.0024
Alaska Mountains	0	0	.	.	.
California Beach	1	1.08297	0.12645	73.3523	<.0001
California Lake	1	0.47855	0.13229	13.0850	0.0003
California Mountains	0	0	.	.	.
Hawaii Beach	1	1.80453	0.09781	340.3990	<.0001
Hawaii Lake	1	0.42488	0.09485	20.0664	<.0001
Hawaii Mountains	0	0	.	.	.
Maine Beach	1	0.99093	0.14340	47.7530	<.0001
Maine Lake	1	0.38796	0.16197	5.7372	0.0166
Maine Mountains	0	0	.	.	.
Mexico Beach	1	1.02413	0.10838	89.2899	<.0001
Mexico Lake	1	0.51537	0.11251	20.9839	<.0001
Mexico Mountains	0	0	.	.	.
Alaska Bed & Breakfast	1	0.40897	0.27771	2.1687	0.1408
Alaska Cabin	1	-2.43166	0.62578	15.0992	0.0001
Alaska Hotel	0	0	.	.	.
California Bed & Breakfast	1	0.54281	0.10989	24.4010	<.0001
California Cabin	1	-0.80653	0.14042	32.9922	<.0001
California Hotel	0	0	.	.	.
Hawaii Bed & Breakfast	1	0.50693	0.09488	28.5458	<.0001
Hawaii Cabin	1	-0.83312	0.09467	77.4467	<.0001
Hawaii Hotel	0	0	.	.	.
Maine Bed & Breakfast	1	0.44868	0.12833	12.2231	0.0005
Maine Cabin	1	-0.88742	0.17247	26.4743	<.0001
Maine Hotel	0	0	.	.	.
Mexico Bed & Breakfast	1	0.42361	0.09781	18.7575	<.0001
Mexico Cabin	1	-0.93363	0.11723	63.4222	<.0001
Mexico Hotel	0	0	.	.	.

---

The parameter estimates for the destinations, including the stay at home (reference level) which is not displayed in the table are displayed in the following table along with the parameters that were used in the DATA step that made the artificial data:

Destination	Estimate	Parameter
Hawaii	2.29050	5
Alaska	-1.58858	-1
Mexico	1.87521	4
California	1.32370	3
Maine	0.77912	2
Stay at home	0	0

You would *not* expect these values to match, but you would expect a consistent ordering of the two, at least if you have generated enough data.

The price, scenery, and lodging parameters and estimates are as follows:

	Alaska	California	Hawaii	Maine	Mexico	Parameter
1499	1.64559	0.97060	0.98873	1.19476	0.91560	-1
1749	0.65152	0.50921	0.49605	0.74813	0.51216	-2
1999	0	0	0	0	0	-3
Beach	1.54354	1.08297	1.80453	0.99093	1.02413	1
Lake	1.27276	0.47855	0.42488	0.38796	0.51537	0
Mountains	0	0	0	0	0	-1
Bed & Breakfast	0.40897	0.54281	0.50693	0.44868	0.42361	3
Cabin	-2.43166	-0.80653	-0.83312	-0.88742	-0.93363	0
Hotel	0	0	0	0	0	2

All orderings are consistent. Furthermore, the parameter estimate for Hawaii at the beach is higher than the other beach parameter estimates. The parameter estimate for Maine in a cabin is not higher than the others even though you added one to the utility for that combination. However, in fact you added one for a three way interaction that included scenery, and that was not modeled.

You fit an alternative-specific effects model. That is, you fit a model with separate parameters for each destination. However, the data were created mostly assuming a main effects model. You could make the artificial data creation much more elaborate than this. Examples of two interaction terms were added. You could add many more. You could in fact have a completely different set of parameters for each destination. You could also have different parameters for different groups of individuals and concatenate the group results to make the final data set. However, the point of this exercise is not to add detailed realism to the artificial data generation. The point is just to have some data that you can use to ensure that you know how to read, process, code, and analyze the data before you collect them. The variances, covariances, and standard errors from the `%ChoiceEff` macro show that all effects of interest are estimable and show other information about the design. Analyzing artificial data will not turn up anything beyond that. You could simply make random data to test your design.

The following step shows one way that you could create purely random data:

```
data _null_;
  do s = 1 to 100;
    do BlockNum = 1 to 2;
      Subject + 1;
      put / blocknum 3. +2 subject 3. +2 @@;
      do SetNum = 1 to 18;
        c = ceil(6 * uniform(17));
        put +(-1) c @@;
      end;
    end;
  end;
stop;
run;
```

The statement `c = ceil(6 * uniform(17))` generates random choices that are integers in the range 1 to 6. The expression `6 * uniform(17)` generates random values in the range 0 to 6, and the `ceil` function creates the ceiling, the integer greater than or equal to the value. This statement is the only statement that did not appear in the previous `DATA _NULL_` step.

However you make the artificial data, particularly when you are new to choice modeling, creating and analyzing artificial data is a useful step to help ensure that you know what you are doing before you go to the time and expense of collecting data.

The last part of this example returns to the “Summary of Subjects, Sets, and Chosen and Unchosen Alternatives” table. Recall that in this example, it is as follows:

---

Summary of Subjects, Sets, and Chosen and Unchosen Alternatives				
Pattern	Number of Choices	Number of Alternatives	Chosen Alternatives	Not Chosen
1	3600	6	1	5

---

The following PROC FREQ steps illustrate the logic that goes into making the first part of the table:

```
proc freq data=coded noprint;
  tables subject*set*block / list out=t1;
run;

proc freq; tables count; run;
```

The number of times each stratum variable occurs in the data set is counted, and then the number of different frequencies is counted and displayed. The results are as follows:

---

The FREQ Procedure

Frequency Count

COUNT	Frequency	Percent	Cumulative Frequency	Cumulative Percent
6	3600	100.00	3600	100.00

---

3600 choice sets of size six occur in the data. The number of stratification variables that you use depends on the way you organize the input data. In this data set, subject number varies across blocks (the subject number varies from 1 to 200 rather than from 1 to 100 in block 1 and again from 1 to 100 in block 2). Hence, in this data set, you could drop the blocking variable from the STRATA statement and get the same results. The following steps illustrate:

```
proc freq data=coded noprint;
  tables subject*set / list out=t1;
run;

proc freq; tables count; run;
```

The results match the PROC FREQ output from when the blocking variable was included.

# Vacation Example

## with Alternative-Specific Attributes

This example discusses choosing the number of choice sets, designing the choice experiment, ensuring that certain key interactions are estimable, examining the design, blocking an existing design, evaluating the design, generating the questionnaire, generating artificial data, reading, processing, and analyzing the data, binary coding, generic attributes, alternative-specific effects, aggregating the data, analysis, and interpretation of the results. In this example, a researcher is interested in studying choice of vacation destinations. This page and the next page contain two summaries of the design, one with factors grouped by attribute and one grouped by destination.

This example is a modification of the previous example. Now, all alternatives do not have the same factors, and all factors do not have the same numbers of levels. There are still five destinations of interest: Hawaii, Alaska, Mexico, California, and Maine. Each alternative is composed of three factors like before: package cost, scenery, and accommodations, only now they do not all have the same levels, and the Hawaii and Mexico alternatives are composed of one additional attribute. For Hawaii and Alaska, the costs are \$1,249, \$1,499, and \$1,749; for California, the prices are \$999, \$1,249, \$1,499, and \$1,749; and for Mexico and Maine, the prices are \$999, \$1,249, and \$1,499. Scenery (mountains, lake, beach) and accommodations (cabin, bed & breakfast, and hotel) are the same as before. The Mexico trip now has the option of a side trip to sites of archaeological significance, via bus, for an additional cost of \$100. The Hawaii trip has the option of a side trip to an active volcano, via helicopter, for an additional cost of \$200. This is typical of the problems that marketing researchers face. We have many factors and *asymmetry*—each alternative is not composed of the same factors, and the factors do not all have the same numbers of levels.

Factor	Destination	Attribute	Levels
X1	Hawaii	Accommodations	Cabin, Bed & Breakfast, Hotel
X2	Alaska	Accommodations	Cabin, Bed & Breakfast, Hotel
X3	Mexico	Accommodations	Cabin, Bed & Breakfast, Hotel
X4	California	Accommodations	Cabin, Bed & Breakfast, Hotel
X5	Maine	Accommodations	Cabin, Bed & Breakfast, Hotel
X6	Hawaii	Scenery	Mountains, Lake, Beach
X7	Alaska	Scenery	Mountains, Lake, Beach
X8	Mexico	Scenery	Mountains, Lake, Beach
X9	California	Scenery	Mountains, Lake, Beach
X10	Maine	Scenery	Mountains, Lake, Beach
X11	Hawaii	Price	\$1249, \$1499, \$1749
X12	Alaska	Price	\$1249, \$1499, \$1749
X13	Mexico	Price	\$999, \$1249, \$1499
X14	California	Price	\$999, \$1249, \$1499, \$1749
X15	Maine	Price	\$999, \$1249, \$1499
X16	Hawaii	Side Trip	Yes, No
X17	Mexico	Side Trip	Yes, No

Factor	Destination	Attribute	Levels
X1	Hawaii	Accommodations	Cabin, Bed & Breakfast, Hotel
X6		Scenery	Mountains, Lake, Beach
X11		Price	\$1249, \$1499, \$1749
X16		Side Trip	Yes, No
X2	Alaska	Accommodations	Cabin, Bed & Breakfast, Hotel
X7		Scenery	Mountains, Lake, Beach
X12		Price	\$1249, \$1499, \$1749
X3	Mexico	Accommodations	Cabin, Bed & Breakfast, Hotel
X8		Scenery	Mountains, Lake, Beach
X13		Price	\$999, \$1249, \$1499
X17		Side Trip	Yes, No
X4	California	Accommodations	Cabin, Bed & Breakfast, Hotel
X9		Scenery	Mountains, Lake, Beach
X14		Price	\$999, \$1249, \$1499, \$1749
X5	Maine	Accommodations	Cabin, Bed & Breakfast, Hotel
X10		Scenery	Mountains, Lake, Beach
X15		Price	\$999, \$1249, \$1499

### Choosing the Number of Choice Sets

We can use the `%MktRuns` autocall macro to suggest experimental design sizes. (All of the autocall macros used in this book are documented starting on page 803.) As before, we specify a list containing the number of levels of each factor as follows:

```
title 'Vacation Example with Asymmetry';
```

```
%mktruns(3 ** 14 4 2 2)
```

The results are as follows:

---

#### Vacation Example with Asymmetry

##### Design Summary

Number of Levels	Frequency
2	2
3	14
4	1

## Vacation Example with Asymmetry

Saturated = 34  
 Full Factorial = 76,527,504

Some Reasonable Design Sizes	Violations	Cannot Be Divided By
72 *	0	
144 *	0	
36	2	8
108	2	8
54	18	4 8 12
90	18	4 8 12
126	18	4 8 12
45	48	2 4 6 8 12
63	48	2 4 6 8 12
81	48	2 4 6 8 12
34 S	151	3 4 6 8 9 12

- \* - 100% Efficient design can be made with the MktEx macro.  
 S - Saturated Design - The smallest design that can be made.  
 Note that the saturated design is not one of the recommended designs for this problem. It is shown to provide some context for the recommended sizes.

## Vacation Example with Asymmetry

n	Design	Reference
72	2 ** 20 3 ** 24 4 ** 1	Orthogonal Array
72	2 ** 19 3 ** 20 4 ** 1 6 ** 1	Orthogonal Array
72	2 ** 18 3 ** 16 4 ** 1 6 ** 2	Orthogonal Array
72	2 ** 13 3 ** 25 4 ** 1	Orthogonal Array
72	2 ** 12 3 ** 21 4 ** 1 6 ** 1	Orthogonal Array
72	2 ** 11 3 ** 24 4 ** 1 6 ** 1	Orthogonal Array
72	2 ** 11 3 ** 17 4 ** 1 6 ** 2	Orthogonal Array
72	2 ** 10 3 ** 20 4 ** 1 6 ** 2	Orthogonal Array
72	2 ** 9 3 ** 16 4 ** 1 6 ** 3	Orthogonal Array
144	2 ** 92 3 ** 24 4 ** 1	Orthogonal Array
144	2 ** 91 3 ** 20 4 ** 1 6 ** 1	Orthogonal Array
144	2 ** 90 3 ** 16 4 ** 1 6 ** 2	Orthogonal Array
144	2 ** 85 3 ** 25 4 ** 1	Orthogonal Array
144	2 ** 84 3 ** 21 4 ** 1 6 ** 1	Orthogonal Array
144	2 ** 83 3 ** 24 4 ** 1 6 ** 1	Orthogonal Array
144	2 ** 83 3 ** 17 4 ** 1 6 ** 2	Orthogonal Array
144	2 ** 82 3 ** 20 4 ** 1 6 ** 2	Orthogonal Array
144	2 ** 81 3 ** 16 4 ** 1 6 ** 3	Orthogonal Array



·  
·  
·

---

The output tells us the size of the saturated design, which is the number of parameters in the linear arrangement, and suggests design sizes. We need at least 34 choice sets, as shown by (`Saturated=34`) in the listing. Any size that is a multiple of 72 is optimal. We would recommend 72 choice sets, four blocks of size 18. However, like the previous vacation example, we use fewer choice sets so that we can illustrate getting an efficient but nonorthogonal design. A design with 36 choice sets is pretty good. Thirty-six is not divisible by  $8 = 2 \times 4$ , so we cannot have equal frequencies in the California price and Mexico and Hawaii side trip combinations. This should not pose any problem. This leaves only 2 error *df* for the linear model, but in the choice model, we have adequate error *df*.

## Designing the Choice Experiment

This problem requires a design with 1 four-level factor for price and 4 three-level factors for price. There are 10 three-level factors for scenery and accommodations as before. Also, we need 2 two-level factors for the two side trips. Note that we do not need a factor for the price or mode of transportation of the side trips since they are constant within each trip. With the `%MktEx` macro, making an asymmetric design is no more difficult than making a symmetric design. The following step creates the design:\*

```
%mktex(3 ** 13 4 3 2 2, n=36, seed=205)
```

```
%mkteval(data=randomized)
```

The last part of the results are as follows:

---

### Vacation Example with Asymmetry

#### The OPTEX Procedure

#### Class Level Information

Class	Levels	Values
x1	3	1 2 3
x2	3	1 2 3
x3	3	1 2 3
x4	3	1 2 3
x5	3	1 2 3
x6	3	1 2 3
x7	3	1 2 3
x8	3	1 2 3

---

\*Due to machine, SAS release, and macro differences, you might not get exactly the same design used in this book, but the differences should be slight.

x9	3	1 2 3
x10	3	1 2 3
x11	3	1 2 3
x12	3	1 2 3
x13	3	1 2 3
x14	4	1 2 3 4
x15	3	1 2 3
x16	2	1 2
x17	2	1 2

Vacation Example with Asymmetry

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	98.8874	97.5943	97.4925	0.9718

Vacation Example with Asymmetry  
 Canonical Correlations Between the Factors  
 There are 2 Canonical Correlations Greater Than 0.316

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16	x17
x1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
x5	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
x6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
x7	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
x8	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
x9	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
x10	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
x11	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
x12	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
x13	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0.25	0	0
x14	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0.33	0.33
x15	0	0	0	0	0	0	0	0	0	0	0	0	0.25	0	1	0	0
x16	0	0	0	0	0	0	0	0	0	0	0	0	0	0.33	0	1	0
x17	0	0	0	0	0	0	0	0	0	0	0	0	0	0.33	0	0	1

---

The macro found a very nice, almost orthogonal and almost 99% *D*-efficient design in 40 seconds. However, we will not use this design. Instead, we will make a larger design with interactions.

## Ensuring that Certain Key Interactions are Estimable

Next, we ensure that certain key interactions are estimable. Specifically, it is good if in the aggregate, the interactions between price and accommodations are estimable for each destination. We would like the following interactions to be estimable:  $x_1 \times x_{11}$   $x_2 \times x_{12}$   $x_3 \times x_{13}$   $x_4 \times x_{15}$   $x_5 \times x_{15}$ . We use the `%MktEx` macro as follows:

```
%mktex(3 ** 13 4 3 2 2, n=36, seed=205
        interact=x1*x11 x2*x12 x3*x13 x4*x14 x5*x15)
```

The macro immediately displays the following errors:

```
ERROR: More parameters than runs.
       If you really want to do this, specify RIDGE=.
       There are 36 runs with 56 parameters.
ERROR: The MKTEX macro ended abnormally.
```

If we want interactions to be estimable, we need more choice sets. The number of parameters is 1 for the intercept,  $14 \times (3-1) + (4-1) + 2 \times (2-1) = 33$  for main effects, and  $4 \times (3-1) \times (3-1) + (4-1) \times (3-1) = 22$  for interactions for a total of  $1 + 33 + 22 = 56$  parameters. This means we need at least 56 choice sets, and ideally for this design with 2, 3, and 4 level factors, we would like the number of sets to be divisible by  $2 \times 2$ ,  $2 \times 3$ ,  $2 \times 4$ ,  $3 \times 3$ , and  $3 \times 4$ . Sixty is divisible by 2, 3, 4, 6, and 12 so is a reasonable design size. Sixty choice sets could be divided into three blocks of size 20, four blocks of size 15, or five blocks of size 12. Seventy-two choice sets is better, since unlike 60, 72 can be divided by 9. Unfortunately, 72 would require one more block.

We can use the `%MktRuns` macro to help us choose the number of choice sets. We also specified a keyword option `max=` to consider only the 45 design sizes from the minimum of 56 up to 100 as follows:

```
title 'Vacation Example with Asymmetry';
%mktruns(3 ** 13 4 3 2 2, interact=x1*x11 x2*x12 x3*x13 x4*x14 x5*x15, max=45)
```

The results are as follows:

---

### Vacation Example with Asymmetry

#### Design Summary

Number of Levels	Frequency
2	2
3	14
4	1

### Vacation Example with Asymmetry

```
Saturated      = 56
Full Factorial = 76,527,504
```

Some Reasonable Design Sizes	Violations	Cannot Be Divided By									
72	58	27	81	108							
81	79	2	4	6	8	12	18	24	36	108	
90	95	4	8	12	24	27	36	81	108		
63	133	2	4	6	8	12	18	24	27	36	81 108
99	133	2	4	6	8	12	18	24	27	36	81 108
96	174	9	18	27	36	81	108				
60	178	8	9	18	24	27	36	81	108		
84	178	8	9	18	24	27	36	81	108		
66	194	4	8	9	12	18	24	27	36	81	108
78	194	4	8	9	12	18	24	27	36	81	108
56 S	232	3	6	9	12	18	24	27	36	81	108

S - Saturated Design - The smallest design that can be made. Note that the saturated design is not one of the recommended designs for this problem. It is shown to provide some context for the recommended sizes.

We see that 72 cannot be divided by  $27 = 9 \times 3$  so, for example, the Maine accommodation/price combinations cannot occur with equal frequency with each of the three-level factors. We see that 72 cannot be divided by  $81 = 9 \times 9$  so, for example, the Mexico accommodation/price combinations cannot occur with equal frequency with each of the Hawaii accommodation/price combinations. We see that 72 cannot be divided by  $108 = 9 \times 12$  so, for example, the California accommodation/price combinations cannot occur with equal frequency with each of the Maine accommodation/price combinations. With interactions, there are many higher-order opportunities for nonorthogonality. However, usually we are not be overly concerned about potential unequal cell frequencies on combinations of attributes in different alternatives.

The smallest number of runs in the table is 60. While 72 is better in that it can be divided by more numbers, either 72 or 60 should work fine. We pick the larger number and run the %MktEx macro again with n=72 specified as follows:

```
%mktex(3 ** 13 4 3 2 2, n=72, seed=368,
interact=x1*x11 x2*x12 x3*x13 x4*x14 x5*x15)
```

The final *D*-efficiency table is as follows:

Vacation Example with Asymmetry				Average Prediction Standard Error
Design Number	D-Efficiency	A-Efficiency	G-Efficiency	
1	89.8309	79.7751	94.4393	0.8819

Sometimes, particularly in models with two-way interactions, we can do better by having %MktEx do pairwise exchanges in the coordinate-exchange algorithm instead of working on a single column at a time. You can always specify `exchange=2` and `order=sequential` to get all possible pairs, but this can be very time consuming. Alternatively, you can use the `order=matrix=SAS-data-set` option and tell %MktEx exactly which pairs of columns to work on. That approach is illustrated in the following steps:

```

data mat;
  do a = 1 to 17;
    b = .;
    output;
  end;
  do a = 1 to 5;
    b = 10 + a;
    output;
  end;
run;

proc print; run;

%mktx(3 ** 13 4 3 2 2,          /* all attrs of all alternatives */
      n=72,                    /* number of choice sets */
      seed=368,                /* random number seed */
      order=matrix=mat,        /* identifies pairs of cols to work on */
      interact=x1*x11 x2*x12 x3*x13 x4*x14 x5*x15) /* interactions */

```

The data set Mat is as follows:

---

Vacation Example with Asymmetry

Obs	a	b
1	1	.
2	2	.
3	3	.
4	4	.
5	5	.
6	6	.
7	7	.
8	8	.
9	9	.
10	10	.
11	11	.
12	12	.
13	13	.
14	14	.
15	15	.
16	16	.
17	17	.

18	1	11
19	2	12
20	3	13
21	4	14
22	5	15

---

It has two columns. The values in the data set indicate the pairs of columns that `%MktEx` should work on together. Missing values are replaced by a random column number for every row and for every pass through the design. This data set instructs `%MktEx` to sequentially go through every column each time paired with some other random column, then work through all of the interaction pairs, `x1*x11`, `x2*x12`, and so on. This performs 22 pairwise exchanges in every row, which is many fewer exchanges than the  $17 \times 16/2 = 136$  that are required with `exchange=2` and `order=sequential`. There are many other combinations that you might consider. A few examples are as follows:

One	Two	Three	Four	Five
1 .	1 11	1 1	1 11	1 . .
2 .	2 12	2 2	2 12	2 . .
3 .	3 13	3 3	3 13	3 . .
4 .	4 14	4 4	4 14	4 . .
5 .	5 15	5 5	5 15	5 . .
6 .	6 .	6 6	6 6	6 . .
7 .	7 .	7 7	7 7	7 . .
8 .	8 .	8 8	8 8	8 . .
9 .	9 .	9 9	9 9	9 . .
10 .	10 .	10 10	10 10	10 . .
11 .		11 11		11 . .
12 .		12 12		12 . .
13 .		13 13		13 . .
14 .		14 14		14 . .
15 .		15 15		15 . .
16 .		16 16		16 . .
17 .		17 17		17 . .
1 11		1 11		1 11 .
2 12		2 12		2 12 .
3 13		3 13		3 13 .
4 14		4 14		4 14 .
5 15		5 15		5 15 .

Set one is the set we just used. Each column is paired with a random column and every interaction pair is mentioned. Set two is like set one except it consists of only 10 pairs and the interaction columns are only paired with the other columns in its interaction term. Set three names each factor twice and then has the usual pairs for interactions. This requests 17 single-column exchanges followed by 5 pairwise exchanges. When a column is repeated, all but the first instance is ignored. `%MktEx` does not consider all pairs of a factor with itself. Set four is similar to set 3 but the interaction columns are only paired with the other columns in its interaction term. Set five is like set one except three-way exchanges are performed and a random column is added to each exchange. There are many other possibilities. It is impossible to know what will work best, but often, expending some effort to consider exchanges in pairs for two-way interactions or in triples for three-way interactions is rewarded with a small gain in

*D*-efficiency.

The macro displays the following notes:

NOTE: Generating the candidate set.

NOTE: Performing 20 searches of 243 candidates, full-factorial=76,527,504.

NOTE: Generating the orthogonal array design, n=72.

The candidate-set search is using a fractional-factorial candidate set with  $3^5 = 243$  candidates. The two-level factors in the candidate set are made from three-level factors by coding down. *Coding down* replaces an  $m$ -level factor with a factor with fewer than  $m$  levels, for example, a two-level factor could be created from a three-level factor:  $((1\ 2\ 3) \Rightarrow (1\ 2\ 1))$ . The four-level factor in the candidate set is made from 2 three-level factors and coding down.  $((1\ 2\ 3) \times (1\ 2\ 3) \Rightarrow (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9) \Rightarrow (1\ 2\ 3\ 4\ 1\ 2\ 3\ 4\ 1))$ . The orthogonal array used for the partial initialization in the coordinate-exchange steps has 72 runs. Some of the results are as follows:

---

Vacation Example with Asymmetry

Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
1	Start	84.8774	84.8774	Can
1	End	84.8774		
2	Start	41.1012		Tab
2	8 5	84.9292	84.9292	
2	16 2	84.9450	84.9450	
2	17 15	85.0008	85.0008	
2	18 3	85.0423	85.0423	
.				
.				
.				
2	42 14	87.3823	87.3823	
2	66 5	87.4076	87.4076	
2	2 13	87.4113	87.4113	
2	End	87.4113		
.				
.				
.				
11	Start	41.1012		Tab
11	End	87.2914		

12	Start	55.7719		Ran,Mut,Ann
12	53 16	87.4195	87.4195	
12	48 9	87.4355	87.4355	
12	49 6	87.4688	87.4688	
12	50 1	87.4688	87.4688	
.				
.				
.				
12	9 4	90.3157	90.3157	
12	End	90.3157		
.				
.				
.				
17	Start	58.3436		Ran,Mut,Ann
17	End	90.5685		

NOTE: Quitting the algorithm search step after 10.03 minutes and 17 designs.

Design Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
0	Initial	90.7877	90.7877	Ini
1	Start	59.2298		Ran,Mut,Ann
1	End	90.3158		
.				
.				
.				
14	Start	58.8515		Ran,Mut,Ann
14	End	90.3433		

NOTE: Quitting the design search step after 20.17 minutes and 14 designs.

Vacation Example with Asymmetry

Design Refinement History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
0	Initial	90.7877	90.7877	Ini
1	Start	88.9486		Pre,Mut,Ann
1	End	90.6106		



2	Start	87.6249		Pre,Mut,Ann
2	64 4	90.7886	90.7886	
2	End	90.7803		
.				
.				
.				
5	Start	89.3771		Pre,Mut,Ann
5	End	90.5049		

NOTE: Quitting the refinement step after 5.60 minutes and 5 designs.

#### Vacation Example with Asymmetry

##### The OPTEX Procedure

##### Class Level Information

Class	Levels	Values
x1	3	1 2 3
x2	3	1 2 3
x3	3	1 2 3
x4	3	1 2 3
x5	3	1 2 3
x6	3	1 2 3
x7	3	1 2 3
x8	3	1 2 3
x9	3	1 2 3
x10	3	1 2 3
x11	3	1 2 3
x12	3	1 2 3
x13	3	1 2 3
x14	4	1 2 3 4
x15	3	1 2 3
x16	2	1 2
x17	2	1 2

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	90.7886	81.5474	93.3553	0.8819

The macro ran in approximately 36 minutes. The algorithm search history shows that the candidate-set approach (Can) used in design 1 found a design that is 84.8774% *D*-efficient. The macro makes no attempt to improve this design, unless there are restrictions on the design, until the end in the design refinement step, and only if it is the best design found.

Designs 2 through 11 used the coordinate-exchange algorithm with an orthogonal array initialization (**Tab**). For this problem, the orthogonal array initialization initializes all 72 rows. For other problems, when the number of runs in the design is greater than the number of runs in the nearest orthogonal array, the remaining rows are randomly initialized. The orthogonal array initialization usually works very well when all but at most a very few rows and columns are left uninitialized and there are no interactions or restrictions. That is not the case in this problem, and when the algorithm switches to a fully-random initialization in design 12, it immediately does better.

The algorithm search phase picked the coordinate-exchange algorithm with a random initialization, random mutations, and simulated annealing as the algorithm to use in the next step, the design search step. The design search history is initialized with the best design ( $D$ -efficiency = 90.7877) found so far. The design search phase starts out with the initial design (**Ini**) found in the algorithm search phase. The macro finds a design with  $D$ -efficiency = 90.3433.

The final set of iterations tries to improve the best design found so far. Random mutations (**Ran**), simulated annealing (**Ann**), and level exchanges are used on the previous best (**Pre**) design. The random mutations are responsible for making the  $D$ -efficiency of the starting design worse than the previous best  $D$ -efficiency. In this case, the design refinement step found a very slight improvement on the best design found by the design search step.

Each stage ended before the maximum number of iterations and displayed a note. The macro displays the following notes:

NOTE: Quitting the algorithm search step after 10.03 minutes and 17 designs.

NOTE: Quitting the design search step after 20.17 minutes and 14 designs.

NOTE: Quitting the refinement step after 5.60 minutes and 5 designs.

The default values for `maxtime=10 20 5` constrain the three steps to run in an approximate maximum time of 10, 20, and 5 minutes. Fewer iterations are performed with `order=matrix` than with the default single-column exchanges because each pair of exchanges takes longer than a single exchange. For example, with two three-level factors, a pairwise exchange considers  $3 \times 3 = 9$  exchanges, whereas a single exchange considers 3 exchanges. However, a single design with a random initialization and annealing, is faster and better than the full `%MktEx` run with a single-column exchange. This is requested by specifying `options=quickr` as follows:

```
data mat;
  do a = 1 to 17;
    b = .;
    output;
  end;
  do a = 1 to 5;
    b = 10 + a;
    output;
  end;
run;

proc print; run;
```

```

%mktx(3 ** 13 4 3 2 2,          /* all attrs of all alternatives */
      n=72,                    /* number of choice sets */
      seed=368,                /* random number seed */
      order=matrix=mat,        /* identifies pairs of cols to work on */
      quickr                   /* very quick run with random init */
      interact=x1*x11 x2*x12 x3*x13 x4*x14 x5*x15) /* interactions */

```

These steps are not run, and we will use the design created with the previous steps.

## Examining the Design

We can use the %MktEval macro to evaluate the goodness of this design as follows:

```
%mkteval(data=design)
```

Some of the results are as follows:

---

Vacation Example with Asymmetry  
 Canonical Correlations Between the Factors  
 There are 0 Canonical Correlations Greater Than 0.316

	x1	x2	x3	x4	x5	x6	x7	x8	x9
x1	1	0.11	0.09	0.17	0.14	0.03	0.09	0.11	0.09
x2	0.11	1	0.18	0.13	0.09	0.14	0.09	0.07	0.15
x3	0.09	0.18	1	0.20	0.11	0.09	0.09	0.13	0.09
x4	0.17	0.13	0.20	1	0.13	0.11	0.07	0.09	0.15
x5	0.14	0.09	0.11	0.13	1	0.03	0.12	0.05	0.13
x6	0.03	0.14	0.09	0.11	0.03	1	0.10	0.04	0.11
x7	0.09	0.09	0.09	0.07	0.12	0.10	1	0.08	0.09
x8	0.11	0.07	0.13	0.09	0.05	0.04	0.08	1	0.07
x9	0.09	0.15	0.09	0.15	0.13	0.11	0.09	0.07	1
.									
.									
.									

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

Frequencies

*	x1	25 24 23
*	x2	24 25 23
	x3	24 24 24
*	x4	25 24 23





## Blocking an Existing Design

An existing design is blocked using the %MktBlock macro. The macro takes the observations in an existing design and optimally sorts them into blocks. Here, we are seeing how to block the linear version of the choice design, but the macro can also be used directly on the choice design. The following step blocks the design:

```
%mktblock(data=randomized, nblocks=4, out=sasuser.AsymVacLinDesBlckd, seed=114)
```

This step took 2 seconds. Some of the results including the one-way frequencies within blocks are as follows:

---

Vacation Example with Asymmetry  
Canonical Correlations Between the Factors  
There are 0 Canonical Correlations Greater Than 0.316

	Block	x1	x2	x3	x4	x5	x6	x7	x8
Block	1	0.08	0.06	0.10	0.06	0.08	0.08	0.08	0.06
x1	0.08	1	0.11	0.09	0.17	0.14	0.03	0.09	0.11
x2	0.06	0.11	1	0.18	0.13	0.09	0.14	0.09	0.07
x3	0.10	0.09	0.18	1	0.20	0.11	0.09	0.09	0.13
x4	0.06	0.17	0.13	0.20	1	0.13	0.11	0.07	0.09
x5	0.08	0.14	0.09	0.11	0.13	1	0.03	0.12	0.05
x6	0.08	0.03	0.14	0.09	0.11	0.03	1	0.10	0.04
x7	0.08	0.09	0.09	0.09	0.07	0.12	0.10	1	0.08
x8	0.06	0.11	0.07	0.13	0.09	0.05	0.04	0.08	1
.									
.									
.									

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

Frequencies

	Block	18	18	18	18
*	x1	25	23	24	
*	x2	25	24	23	
	x3	24	24	24	
*	x4	25	24	23	
*	x5	25	24	23	
*	x6	22	26	24	

```

*   x7           24 26 22
*   x8           26 23 23
*   x9           23 22 27
*   x10          24 26 22
*   x11          23 24 25
*   x12          24 23 25
*   x13          24 23 25
*   x14          17 19 19 17
*   x15          24 24 24
*   x16          36 36
*   x17          37 35
*   Block x1     6 6 6 6 5 7 6 6 6 7 6 5
*   Block x2     6 6 6 6 6 6 7 6 5 6 6 6
*   Block x3     6 6 6 6 6 6 6 7 5 6 5 7
*   Block x4     7 6 5 6 6 6 6 6 6 6 6 6
*   Block x5     6 6 6 7 5 6 6 7 5 6 6 6
*   Block x6     6 7 5 6 6 6 5 7 6 5 6 7
*   Block x7     6 7 5 5 7 6 7 6 5 6 6 6
*   Block x8     6 6 6 7 5 6 7 6 5 6 6 6
*   Block x9     7 5 6 5 5 8 6 6 6 5 6 7
*   Block x10    5 7 6 6 6 6 6 7 5 7 6 5
*   Block x11    4 7 7 7 5 6 5 6 7 7 6 5
*   Block x12    5 6 7 7 6 5 5 6 7 7 5 6
*   Block x13    6 6 6 6 5 7 6 6 6 6 6 6
*   Block x14    3 5 4 6 5 5 5 3 4 5 6 3 5 4 4 5
*   Block x15    6 6 6 6 5 7 5 7 6 7 6 5
*   Block x16    9 9 9 9 9 9 9 9
*   Block x17    9 9 9 9 10 8 9 9
.
.
.

N-Way          1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
               1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
               1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
               1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

```

---

They should be examined to ensure that each level is well represented in each block. The design is nearly balanced in most of the factors and blocks.

Collecting data is time consuming and expensive. Before collecting data, it is a good practice to convert your linear arrangement into a choice design and evaluate it in the context of a choice model. We start by creating some formats for the factor levels and the key to converting the linear arrangement into a choice design as follows:\*

```
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 has five attributes. **Place** is the alternative name. **Lodge**, **Scene**, **Price** and **Side** are created from the design using the indicated factors. See page 356 for more information about creating the design key. Notice that **Side** only applies to some of the alternatives and hence has missing values for the others. Processing the design and merging it with the data are similar to the examples on pages 356 and 371. One difference is now there are asymmetries in **Price**. For Hawaii's price, **x11**, we need to change 1, 2, and 3 to \$1249, \$1499, and \$1749. For Alaska's price, **x12**, we need to change 1, 2, and 3 to \$1249, \$1499, and \$1749. For Mexico's price, **x13**, we need to change 1, 2, and 3 to \$999, \$1249, and \$1499. For California's price, **x14**, we need to change 1, 2, 3, and 4 to \$999, \$1249, \$1499, and \$1749. For Maine's price, **x11**, we need to change 1, 2, and 3 to \$999, \$1249, and \$1499. We can simplify the problem by adding 1 to **x11** and **x12**, which are the factors that start at \$1249 instead of \$999. This lets us use a common format to set the price. See pages 499 and 884 for examples of handling more complicated asymmetries.

---

\*See page 67 for an explanation of the linear arrangement of a choice design versus the arrangement of a choice design that is more suitable for analysis.



The following steps process the design:

```

data temp;
  set sasuser.AsymVacLinDesBlckd(rename=(block=Form));
  x11 + 1;
  x12 + 1;
  run;

%mktroll(design=temp, key=key, alt=place, out=sasuser.AsymVacChDes,
         options=nowarn, keep=form)

data sasuser.AsymVacChDes;
  set sasuser.AsymVacChDes;
  format scene scene. lodge lodge. side side. price price.;
  run;

proc print data=sasuser.AsymVacChDes(obs=12);
  by form set; id form set;
  run;

```

The first two choice sets are as follows:

---

Vacation Example with Asymmetry							
Form	Set	Place	Lodge	Scene	Price	Side	
1	1	Hawaii	Bed & Breakfast	Lake	1499	No	
		Alaska	Bed & Breakfast	Mountains	1249		
		Mexico	Cabin	Lake	999		
		California	Cabin	Lake	1249		
		Maine	Hotel	Beach	1249		
1	2	Hawaii	Hotel	Lake	1749	Side Trip	
		Alaska	Hotel	Lake	1499		
		Mexico	Hotel	Beach	1249		No
		California	Cabin	Beach	999		
		Maine	Cabin	Lake	1249		

---

Notice that each has six alternatives, one of which is displaying in this format as all blank.

## Testing the Design Before Data Collection

Collecting data is time consuming and expensive. It is always good practice to make sure that the design works with the most complicated model that we anticipate fitting. The following step evaluates the choice design:

```

title2 'Evaluate the Choice Design';

%choicEff(data=sasuser.AsymVacChDes,/* candidate set of choice sets      */
init=sasuser.AsymVacChDes(keep=set), /* select these sets              */
intiter=0,                          /* evaluate without internal iterations */
                                      /* alternative-specific effects model  */
                                      /* zero=none - no ref levels for place */
                                      /* order=data - do not sort levels    */
model=class(place / zero=none order=data)
                                      /* zero=none - no ref levels any factor */
                                      /* order=formatted - sort levels       */
                                      /* use blank sep to build interact terms*/
class(place * price place * scene place * lodge /
zero=none order=formatted separators=' ' ')
                                      /* no ref level for place              */
                                      /* ref level for side is 'No'          */
                                      /* use blank sep to build interact terms*/
class(place * side / zero=' ' 'No' separators=' ' ' ') /
lprefix=0                            /* lpr=0 labels created from just levels*/
cprefix=0,                            /* cpr=0 names created from just levels */

nsets=72,                             /* number of choice sets                */
nalts=6,                               /* number of alternatives                */
beta=zero)                             /* assumed beta vector, Ho: b=0        */

```

We use the `%ChoiCEff` macro to evaluate our choice design. You can both use this macro to search a candidate set for an efficient choice design, and you can use it to evaluate a design created by other means. The way you check a design like this is to first name it in the `data=` option. This is the candidate set that contains all of the choice sets that we will consider. In addition, the same design is named in the `init=` option. The full specification is `init=sasuser.AsymVacChDes(keep=set)`. Just the variable `Set` is kept. It is used to bring in just the indicated choice sets from the `data=` design, which in this case is all of them. The option `nsets=72` specifies the number of choice sets, and `nalts=6` specifies the number of alternatives. The option `beta=zero` specifies that we are assuming for design evaluation purpose that all of the betas or part-worth utilities are zero. You can evaluate the design for other parameter vectors by specifying a list of numbers after `beta=`. This changes the variances and standard errors. We also specify `intiter=0` which specifies zero internal iterations. We use zero internal iterations when we want to evaluate an initial design, but not attempt to improve it. The last option specifies the model.

The model specification contains everything that appears in the TRANSREG procedure's `model` statement for coding the design. Many of these options should be familiar from previous examples. The specification `class(place / zero=none order=data)` names the `place` variable as a classification variable and asks for coded variables for every nonmissing level (`zero=none`). The order of the levels on output matches the order that the levels are first encountered in the input data set. This specification creates the alternative effects or alternative-specific intercepts.

The next specification, `class(place * price place * scene place * lodge / zero=none order=formatted separators=' ' ' ')`, requests alternative-specific effects for all of the attributes except the side trip. The alternative-specific effects are requested by interacting the alternative-specific intercepts, in this case the destination, with the attributes. The `zero=none` option creates binary variables for all categories. In contrast, by default, a variable is not created for the last category—the parameter for the last category is a structural zero. The `zero=none` option is used when you want to see the structural zeros in the results. The `separators=' ' ' '` option (`separators=` quote quote space quote space quote, which provides two strings (one null and the other blank), allows you to specify two label component separators for the main effect and interaction terms, respectively. By specifying a blank for the second value, we request labels for the side trip effects like 'Mexico Side Trip' instead of the default 'Mexico \* Side Trip'. This option is explained in more detail on page 449.

The last part of the model specification consists of `class(place * side / zero=' ' 'No' separators=' ' ' ')` and creates the alternative-specific side trip effects with all levels for `place` and 'No' as the reference level for the side trip attribute. See page 78 for more information about the `zero=` option. The last part of the model specification is followed by a slash and some options: `/ lprefix=0 cprefix=0`). The `cprefix=0` option specifies that when names are created for the binary variables, zero characters of the original variable name should be used as a prefix. This means that the names are created only from the level values. The `lprefix=0` option specifies that when labels are created for the binary variables, zero characters of the original variable name should be used as a prefix. This means that the labels are created only from the level values.

The last part of the output is as follows:

---

Vacation Example with Asymmetry  
Evaluate the Choice Design

Final Results

Design	1
Choice Sets	72
Alternatives	6
Parameters	38
Maximum Parameters	360
D-Efficiency	0
D-Error	.

Vacation Example with Asymmetry  
Evaluate the Choice Design

n Variable Name	Label	Variance	DF	Standard Error
1 Hawaii	Hawaii	0.91061	1	0.95426
2 Alaska	Alaska	0.70276	1	0.83831
3 Mexico	Mexico	0.79649	1	0.89246
4 California	California	0.89577	1	0.94645
5 Maine	Maine	0.82172	1	0.90649

6	Alaska_999	Alaska 999	.	0	.
7	Alaska_1249	Alaska 1249	0.59635	1	0.77223
8	Alaska_1499	Alaska 1499	0.60551	1	0.77814
9	Alaska_1749	Alaska 1749	.	0	.
10	California_999	California 999	0.85492	1	0.92462
11	California_1249	California 1249	0.81130	1	0.90072
12	California_1499	California 1499	0.82552	1	0.90858
13	California_1749	California 1749	.	0	.
14	Hawaii_999	Hawaii 999	.	0	.
15	Hawaii_1249	Hawaii 1249	0.60792	1	0.77969
16	Hawaii_1499	Hawaii 1499	0.59679	1	0.77252
17	Hawaii_1749	Hawaii 1749	.	0	.
18	Maine_999	Maine 999	0.60676	1	0.77894
19	Maine_1249	Maine 1249	0.61109	1	0.78172
20	Maine_1499	Maine 1499	.	0	.
21	Maine_1749	Maine 1749	.	0	.
22	Mexico_999	Mexico 999	0.59178	1	0.76927
23	Mexico_1249	Mexico 1249	0.60604	1	0.77849
24	Mexico_1499	Mexico 1499	.	0	.
25	Mexico_1749	Mexico 1749	.	0	.
26	AlaskaBeach	Alaska Beach	0.63778	1	0.79861
27	AlaskaLake	Alaska Lake	0.58330	1	0.76374
28	AlaskaMountains	Alaska Mountains	.	0	.
29	CaliforniaBeach	California Beach	0.59453	1	0.77106
30	CaliforniaLake	California Lake	0.67196	1	0.81973
31	CaliforniaMountains	California Mountains	.	0	.
32	HawaiiBeach	Hawaii Beach	0.63923	1	0.79952
33	HawaiiLake	Hawaii Lake	0.61115	1	0.78176
34	HawaiiMountains	Hawaii Mountains	.	0	.
35	MaineBeach	Maine Beach	0.63688	1	0.79805
36	MaineLake	Maine Lake	0.58479	1	0.76471
37	MaineMountains	Maine Mountains	.	0	.
38	MexicoBeach	Mexico Beach	0.59462	1	0.77111
39	MexicoLake	Mexico Lake	0.59710	1	0.77272
40	MexicoMountains	Mexico Mountains	.	0	.
41	AlaskaBed__Breakfast	Alaska Bed & Breakfast	0.62130	1	0.78823
42	AlaskaCabin	Alaska Cabin	0.61012	1	0.78110
43	AlaskaHotel	Alaska Hotel	.	0	.
44	CaliforniaBed__Breakfast	California Bed & Breakfast	0.62122	1	0.78817
45	CaliforniaCabin	California Cabin	0.60866	1	0.78016
46	CaliforniaHotel	California Hotel	.	0	.
47	HawaiiBed__Breakfast	Hawaii Bed & Breakfast	0.61876	1	0.78661
48	HawaiiCabin	Hawaii Cabin	0.59145	1	0.76906
49	HawaiiHotel	Hawaii Hotel	.	0	.
50	MaineBed__Breakfast	Maine Bed & Breakfast	0.61592	1	0.78480
51	MaineCabin	Maine Cabin	0.60681	1	0.77898
52	MaineHotel	Maine Hotel	.	0	.
53	MexicoBed__Breakfast	Mexico Bed & Breakfast	0.61050	1	0.78134
54	MexicoCabin	Mexico Cabin	0.61670	1	0.78530
55	MexicoHotel	Mexico Hotel	.	0	.

56	AlaskaSide_Trip	Alaska Side Trip	.	0	.
57	CaliforniaSide_Trip	California Side Trip	.	0	.
58	HawaiiSide_Trip	Hawaii Side Trip	0.40413	1	0.63572
59	MaineSide_Trip	Maine Side Trip	.	0	.
60	MexicoSide_Trip	Mexico Side Trip	0.40622	1	0.63735
				==	
				38	

It consists of a table with the name and label for each parameter along with its variance,  $df$ , and standard error. It needs to be carefully evaluated to see if the zeros and nonzeros are in all of the right places. We see one parameter for five of the six destinations, with the constant stay-at-home alternative in all cases excluded from the table. This is followed by four terms for the Alaska price effect. The Alaska at \$999 parameter is zero since \$999 does not apply to Alaska. The Alaska at \$1749 parameter is the reference level and hence is zero. The other two Alaska price parameters are nonzero. Similarly, each of the alternative-specific price effects have two or three parameters (the number of applicable prices minus one). For the scenery and accommodations attributes, each alternative has two nonzero parameters and a reference level. There are two nonzero parameters for the side trips for the two applicable destinations. The pattern of zeros and nonzeros looks perfect. There are 38 parameters in the alternative-specific model.

You should also note that the variances and standard errors. They are all approximately the same order of magnitude. Sometimes you might see wildly varying parameters. This is usually a sign of a problematic design, perhaps one with too few choice sets for the number of parameters. This design looks good. The number of parameters in our model is 38 and the maximum number we could estimate with this design is 360, so too few choice sets should not be a problem. Note one difference between these results and the results that we see in the previous example on page 361. Here, our standard errors are not constant within an attribute, although they are similar. This is because none of our factors are orthogonal, although they are close.

## Generating the Questionnaire

The following step produces the questionnaire:

```
%let m = 6; /* m alts including constant */
%let mm1 = %eval(&m - 1); /* m - 1 */
%let n = 18; /* number of choice sets */
%let blocks = 4; /* number of blocks */

title;
options ls=80 ps=60 nonumber nodate;

data _null_;
  array dests[&mm1] $ 10 _temporary_ ('Hawaii' 'Alaska' 'Mexico'
                                     'California' 'Maine');
  array scenes[3] $ 13 _temporary_
    ('the Mountains' 'a Lake' 'the Beach');
```

```

array lodging[3] $ 15 _temporary_
                    ('Cabin' 'Bed & Breakfast' 'Hotel');
array x[15];
array p[&mm1];
length price $ 6;
file print linesleft=11;

set sasuser.AsymVacLinDesBlckd;
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;

```

The first two choice sets for the first subject are as follows:

---

Form: 1 Subject: \_\_\_\_\_

- 1) Circle your choice of vacation destinations:
    - 1) Hawaii, staying in a Bed & Breakfast near a Lake, with a package cost of \$1,499.
    - 2) Alaska, staying in a Bed & Breakfast near the Mountains, with a package cost of \$1,249.
    - 3) Mexico, staying in a Cabin near a Lake, with a package cost of \$999.
    - 4) California, staying in a Cabin near a Lake, with a package cost of \$1,249.
    - 5) Maine, staying in a Hotel near the Beach, with a package cost of \$1,249.
    - 6) Stay at home this year.
  - 2) Circle your choice of vacation destinations:
    - 1) Hawaii, staying in a Hotel near a Lake, with a package cost of \$1,749.
    - 2) Alaska, staying in a Hotel near a Lake, with a package cost of \$1,499.
    - 3) Mexico, staying in a Hotel near the Beach, with a package cost of \$1,249, and an optional visit to archaeological sites for an additional \$100.
    - 4) California, staying in a Cabin near the Beach, with a package cost of \$999.
    - 5) Maine, staying in a Cabin near a Lake, with a package cost of \$1,249.
    - 6) Stay at home this year.
-

In practice, data collection is typically be much more elaborate than this. It might involve art work or photographs, and the choice sets might be presented and the data might be collected through personal interview or over the Web. However the choice sets are presented and the data are collected, the essential elements remain the same. Subjects are shown a set of alternatives and are asked to make a choice, then they go on to the next set.

## Generating Artificial Data

This section shows in a cursory way how to generate artificial data. Some researchers wisely like to use artificial data to test a design before spending lots of money on collecting data. See the section beginning on page 393 for a detailed discussion of generating artificial data.

The following step generates an artificial set of data:

```

data _null_;
  array dests[&mmm1] _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.AsymVacLinDesBlckd point=n;
        do dest = 1 to &mmm1;
          u[dest] = dests[dest] + lodging[x[dest]] +
            scenes[x[&mmm1 + dest]] -
            x[2 * &mmm1 + dest] +
            2 * normal(17);
        end;
        u[1] = u[1] + (x16 = 1);
        u[3] = u[3] + (x17 = 1);
        u[&m] = -3 + 3 * normal(17);
        m = max(of u1-u[&m]);
        if abs(u1 - m) < 1e-4 then c = 1;
        else if abs(u2 - m) < 1e-4 then c = 2;
        else if abs(u3 - m) < 1e-4 then c = 3;
        else if abs(u4 - m) < 1e-4 then c = 4;
        else if abs(u5 - m) < 1e-4 then c = 5;
        else c = 6;
        put +(-1) c @@;
      end;
    end;
  end;
stop;
run;

```



Collecting data is time consuming and expensive. Generating some artificial data before the data are collected to test your code and make sure the analysis will run is a good idea. It helps avoid the “How am I going to analyze this?” question from occurring after the data have already been collected. This step generates data for 400 subjects, 100 per block.

The `dests`, `scenes`, and `lodging` arrays are initialized with part-worth utilities for each level. The utilities for each of the destinations are computed and stored in the array `u` in the statement `u[dest] = ...`, which includes an error term `2 * normal(17)`. The utilities for the side trips are added in separately with `u[1] = u[1] + (x16 = 1)` and `u[3] = u[3] + (x17 = 1)`. The utility for the stay-at-home alternative is `-3 + 3 * normal(17)`. The maximum utility is computed, `m = max(of u1-u&m)` and the alternative with the maximum utility is chosen. The `put` statement writes out the results to the log.

## Reading, Processing, and Analyzing the Data

The results from the previous step are pasted into a `DATA` step and run to mimic reading real input data as follows:

```
title 'Vacation Example with Asymmetry';

data results;
  input Subj Form (choose1-choose&n) (1.) @@;
  datalines;
1 1 413414111315351335   2 2 115311141441134121   3 3 331451344433513341
4 4 113111143133311314   5 1 113413531545431313   6 2 145131111414331511
7 3 313413113111313331   8 4 415143311133541321   9 1 133314111133431113
.
.
.
;
```

The analysis proceeds in a fashion similar to the simpler vacation example on page 371. The following steps merge the data and display a subset of the results:

```
%mktmerge(design=sasuser.AsymVacChDes, data=results, out=res2, blocks=form,
           nsets=&n, nalts=&m, setvars=choose1-choose&n,
           stmts=%str(price = input(put(price, price.), 5.);
                    format scene scene. lodge lodge. side side.));

proc print data=res2(obs=18);
  id form subj set; by form subj set;
run;
```

The first three choice sets for the first subject are as follows:

---

Vacation Example with Asymmetry

Form	Subj	Set	Place	Lodge	Scene	Price	Side	c
1	1	1	Hawaii	Bed & Breakfast	Lake	1499	No	2
			Alaska	Bed & Breakfast	Mountains	1249		2
			Mexico	Cabin	Lake	999	No	2
			California	Cabin	Lake	1249		1
			Maine	Hotel	Beach	1249		2
1	1	2	Hawaii	Hotel	Lake	1749	Side Trip	1
			Alaska	Hotel	Lake	1499		2
			Mexico	Hotel	Beach	1249	No	2
			California	Cabin	Beach	999		2
			Maine	Cabin	Lake	1249		2
1	1	3	Hawaii	Hotel	Mountains	1749	Side Trip	2
			Alaska	Hotel	Mountains	1749		2
			Mexico	Bed & Breakfast	Beach	1249	Side Trip	1
			California	Cabin	Lake	1249		2
			Maine	Bed & Breakfast	Mountains	999		2

---

Indicator variables and labels are created using PROC TRANSREG like before as follows:

```
proc transreg design=5000 data=res2 nozeroconstant norestoremissing;
  model class(place / zero=none order=data)
    class(price scene lodge / zero=none order=formatted)
    class(place * side / zero=' ' 'No' separators=' ' ' ') /
    lprefix=0;
  output out=coded(drop=_type_ _name_ intercept);
  id subj set form c;
run;

proc print data=coded(obs=6) label;
run;
```

The `design=5000` option specifies that no model is fit; the procedure is just being used to code a design in blocks of 5000 observations at a time. The `nozeroconstant` option specifies that if the coding creates a constant variable, it should not be zeroed. The `norestoremissing` option specifies that missing values should not be restored when the `out=` data set is created. The `model` statement names the variables to code and provides information about how they should be coded. The specification `class(place / ...)` specifies that the variable `Place` is a classification variable and requests a binary coding. The `zero=none` option creates binary variables for all categories. The `order=data` option sorts the levels into the order they are encountered in the data set. Similarly, the variables `Price`, `Scene`, and `Lodge` are classification variables. The specification `class(place * side / ...)` creates alternative-specific side trip effects. The option `zero=' ' 'No'` specifies that indicator variables should be created for



Obs	Place	Price	Scene	Lodge	Side	Subj	Set	Form	c
1	Hawaii	1499	Lake	Bed & Breakfast	No	1	1	1	2
2	Alaska	1249	Mountains	Bed & Breakfast		1	1	1	2
3	Mexico	999	Lake	Cabin	No	1	1	1	2
4	California	1249	Lake	Cabin		1	1	1	1
5	Maine	1249	Beach	Hotel		1	1	1	2
6						1	1	1	2

The PROC PHREG specification is the same as we have used before. Recall that we used %phchoice(on) on page 287 to customize the output from PROC PHREG.) The following step analyzes the data:

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

The results are as follows:

#### Vacation Example with Asymmetry

##### The PHREG Procedure

##### Model Information

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

Number of Observations Read	43200
Number of Observations Used	43200

##### Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Pattern	Number of Choices	Number of Alternatives	Chosen Alternatives	Not Chosen
1	7200	6	1	5

##### Convergence Status

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

## Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	25801.336	12603.247
AIC	25801.336	12631.247
SBC	25801.336	12727.593

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	13198.0891	14	<.0001
Score	12223.0125	14	<.0001
Wald	5081.3295	14	<.0001

## Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Hawaii	1	3.61141	0.22224	264.0701	<.0001
Alaska	1	-0.94997	0.26364	12.9836	0.0003
Mexico	1	2.26877	0.22776	99.2247	<.0001
California	1	1.54548	0.22760	46.1102	<.0001
Maine	1	0.74153	0.23210	10.2074	0.0014
999	1	2.10214	0.07298	829.7619	<.0001
1249	1	1.44298	0.06078	563.6949	<.0001
1499	1	0.72311	0.05936	148.4188	<.0001
1749	0	0	.	.	.
Beach	1	1.42021	0.04635	938.8384	<.0001
Lake	1	0.72019	0.04472	259.3676	<.0001
Mountains	0	0	.	.	.
Bed & Breakfast	1	0.65045	0.04079	254.3369	<.0001
Cabin	1	-1.42317	0.04809	875.8795	<.0001
Hotel	0	0	.	.	.
Alaska Side Trip	0	0	.	.	.
California Side Trip	0	0	.	.	.
Hawaii Side Trip	1	0.71850	0.05753	155.9801	<.0001
Maine Side Trip	0	0	.	.	.
Mexico Side Trip	1	0.65550	0.06293	108.4863	<.0001

---

You would not expect the part-worth utilities to match those that are used to generate the data, but you would expect a similar ordering within each attribute, and in fact that does occur. These data can also be analyzed with quantitative price effects and destination by attribute interactions, as in the previous vacation example.

## Aggregating the Data

This data set is rather large with 43,200 observations. You can make the analysis run faster and with less memory by aggregating. Instead of stratifying on each choice set and subject combination, you can stratify just on choice set and specify the number of times each alternative is chosen and the number of times it is not chosen. First, use PROC SUMMARY to count the number of times each observation occurs. Specify all the analysis variables, and in this example, also specify `Form`. The variable `Form` is 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` is not needed.) PROC SUMMARY stores the number of times each unique observation appears in the variable `_freq_`. PROC PHREG is then run with a `freq` statement. Now, instead of analyzing a data set with 43,200 observations and 7200 strata, we analyze a data set with at most  $2 \times 6 \times 72 = 864$  observations and 72 strata. For each of the 6 alternatives and 72 choice sets, there are typically 2 observations in the aggregate data set: one that contains the number of times it is chosen and one that contains the number of times it is not chosen. When one of those counts is zero, there is one observation. In this case, the aggregate data set has 724 observations. The following steps aggregate and perform the analysis:

```
proc summary data=coded nway;
  class form set c &_trgind;
  output out=agg(drop=_type_);
run;

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

PROC SUMMARY ran in three seconds, and PROC PHREG ran in less than one second. The results are as follows:

---

### Vacation Example with Asymmetry

#### The PHREG Procedure

##### Model Information

Data Set	WORK.AGG
Dependent Variable	c
Censoring Variable	c
Censoring Value(s)	2
Frequency Variable	_FREQ_
Ties Handling	BRESLOW
Number of Observations Read	724
Number of Observations Used	724
Sum of Frequencies Read	43200
Sum of Frequencies Used	43200

## Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Stratum	Form	Set	Number of Alternatives	Chosen Alternatives	Not Chosen
1	1	1	600	100	500
2	1	2	600	100	500
3	1	3	600	100	500
4	1	4	600	100	500
.					
.					
71	4	71	600	100	500
72	4	72	600	100	500
-----					
Total			43200	7200	36000

---

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 are chosen in each stratum. The log likelihood statistics are different, but that does not matter since the Chi-Square statistics are the same. Page 466 provides more information about this.

## Brand Choice Example with Aggregate Data

In this next example, subjects are presented with brands of a product at different prices. There are 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 is chosen, are shown next.

Set	Brand 1	Brand 2	Brand 3	Brand 4	Other
1	\$3.99 4	\$5.99 29	\$3.99 16	\$5.99 42	\$4.99 9
2	\$5.99 12	\$5.99 19	\$5.99 22	\$5.99 33	\$4.99 14
3	\$5.99 34	\$5.99 26	\$3.99 8	\$3.99 27	\$4.99 5
4	\$5.99 13	\$3.99 37	\$5.99 15	\$3.99 27	\$4.99 8
5	\$5.99 49	\$3.99 1	\$3.99 9	\$5.99 37	\$4.99 4
6	\$3.99 31	\$5.99 12	\$5.99 6	\$3.99 18	\$4.99 33
7	\$3.99 37	\$3.99 10	\$5.99 5	\$5.99 35	\$4.99 13
8	\$3.99 16	\$3.99 14	\$3.99 5	\$3.99 51	\$4.99 14

The first choice set consists of Brand 1 at \$3.99, Brand 2 at \$5.99, Brand 3 at \$3.99, Brand 4 at \$5.99, and Other at \$4.99. From this choice set, Brand 1 is chosen 4 times, Brand 2 is chosen 29 times, Brand 3 is chosen 16 times, Brand 4 is chosen 42 times, and Other is chosen 9 times.

### Processing the Data

As in the previous examples, we 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 has 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 are 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 are 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. The following 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. The first 15 observations of the data set are as follows:

---

 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, are discussed in more detail on page 452. The following step displays the first five rows:

```
proc print data=price(obs=5); run;
```

The results are as follows:

---

## 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 as follows:

```
proc transreg design data=price nozeroconstant norestoremissing;
  model class(brand / zero=none) identity(price) / lprefix=0;
  output out=coded(drop=_type_ _name_ intercept);
  label price = 'Price';
  id subj set c;
run;
```

The `design` option specifies that no model is fit; the procedure is just being used to code a design. The `nozeroconstant` option specifies that if the coding creates a constant variable, it should not be zeroed. The `norestoremissing` option specifies that missing values should not be restored when the `out=` data set is created. The `model` statement names the variables to code and provides information about how they should be coded. The specification `class(brand / zero=none)` specifies that the variable `Brand` is a classification variable and requests a binary coding. The `zero=none` option creates binary variables for all categories. The specification `identity(price)` specifies that the variable `Price` is quantitative and hence should directly enter the model without coding. The `lprefix=0` option specifies that when labels are created for the binary variables, zero characters of the original variable name should be used as a prefix. This means that the labels are created only from the level values. An `output` statement names the output data set and drops variables that are not needed. Finally, the `id` statement names the additional variables that we want copied from the input to the output data set. The following step performs the analysis:

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

Recall that we used `%phchoice(on)` on page 287 to customize the output from PROC PHREG. The results are as follows:

---

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

The PHREG Procedure

Model Information

Data Set	WORK.CODED
Dependent Variable	c
Censoring Variable	c
Censoring Value(s)	2
Ties Handling	BRESLOW
Number of Observations Read	4000
Number of Observations Used	4000

## Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Pattern	Number of Choices	Number of Alternatives	Chosen Alternatives	Not Chosen
1	800	5	1	4

## Convergence Status

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

## Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	2575.101	2425.214
AIC	2575.101	2435.214
SBC	2575.101	2458.637

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	149.8868	5	<.0001
Score	153.2328	5	<.0001
Wald	142.9002	5	<.0001

## Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Brand 1	1	0.66727	0.12305	29.4065	<.0001
Brand 2	1	0.38503	0.12962	8.8235	0.0030
Brand 3	1	-0.15955	0.14725	1.1740	0.2786
Brand 4	1	0.98964	0.11720	71.2993	<.0001
Other	0	0	.	.	.
Price	1	0.14966	0.04406	11.5379	0.0007

---

## Alternative-Specific Price Effects

In the following step, the data are coded for fitting a multinomial logit model with brand by price effects:

```
proc transreg design data=price nozeroconstant norestoremissing;
  model class(brand / zero=none separators=' ' |
    identity(price) / lprefix=0;
  output out=coded(drop=_type_ _name_ intercept);
  label price = 'Price';
  id subj set c;
run;
```

The PROC TRANSREG `model` statement has a vertical bar, “|”, between the `class` specification and the `identity` specification. Since the `zero=none` option is specified with `class`, the vertical bar creates two sets of variables: five indicator variables for the brand effects and five more variables for the brand by price interactions. The `separators=` option allows you to specify two label component separators as quoted strings. The specification `separators=' ' ' '` (`separators=` quote quote space quote space quote) specifies a null string (quote quote) and a blank (quote space quote). The `separators=' ' ' '` option in the `class` specification specifies the separators that are used to construct the labels for the main effect and interaction terms, respectively. By default, the alternative-specific price effects—the brand by price interactions—have labels like `'Brand 1 * Price'` since the default second value for `separators=` is `' * '` (a quoted space asterisk space). Specifying `' '` (a quoted space) as the second value creates labels of the form `'Brand 1 Price'`. Since `lprefix=0`, the main-effects separator, which is the first `separators=` value, `' '` (quote quote), is ignored. Zero name or input variable label characters are used to construct the label. The label is simply the formatted value of the `class` variable. The following steps display the first two coded choice sets and perform the analysis:

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

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

title2;
```

The results are as follows:

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

Obs	Subj	Set	c	Brand	Price	Brand 1	Brand 2	Brand 3	Brand 4
1	1	1	1	Brand 1	3.99	1	0	0	0
2	1	1	2	Brand 2	5.99	0	1	0	0
3	1	1	2	Brand 3	3.99	0	0	1	0
4	1	1	2	Brand 4	5.99	0	0	0	1
5	1	1	2	Other	4.99	0	0	0	0
6	2	1	1	Brand 1	3.99	1	0	0	0
7	2	1	2	Brand 2	5.99	0	1	0	0
8	2	1	2	Brand 3	3.99	0	0	1	0
9	2	1	2	Brand 4	5.99	0	0	0	1
10	2	1	2	Other	4.99	0	0	0	0

Obs	Other	Brand 1 Price	Brand 2 Price	Brand 3 Price	Brand 4 Price	Other Price
1	0	3.99	0.00	0.00	0.00	0.00
2	0	0.00	5.99	0.00	0.00	0.00
3	0	0.00	0.00	3.99	0.00	0.00
4	0	0.00	0.00	0.00	5.99	0.00
5	1	0.00	0.00	0.00	0.00	4.99
6	0	3.99	0.00	0.00	0.00	0.00
7	0	0.00	5.99	0.00	0.00	0.00
8	0	0.00	0.00	3.99	0.00	0.00
9	0	0.00	0.00	0.00	5.99	0.00
10	1	0.00	0.00	0.00	0.00	4.99

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

The PHREG Procedure

Model Information

Data Set	WORK.CODED
Dependent Variable	c
Censoring Variable	c
Censoring Value(s)	2
Ties Handling	BRESLOW
Number of Observations Read	4000
Number of Observations Used	4000

## Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Pattern	Number of Choices	Number of Alternatives	Chosen Alternatives	Not Chosen
1	800	5	1	4

## Convergence Status

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

## Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	2575.101	2424.812
AIC	2575.101	2440.812
SBC	2575.101	2478.288

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	150.2891	8	<.0001
Score	154.2563	8	<.0001
Wald	143.1425	8	<.0001

## Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Brand 1	1	-0.00972	0.43555	0.0005	0.9822
Brand 2	1	-0.62230	0.48866	1.6217	0.2028
Brand 3	1	-0.81250	0.60318	1.8145	0.1780
Brand 4	1	0.31778	0.39549	0.6456	0.4217
Other	0	0	.	.	.
Brand 1 Price	1	0.13587	0.08259	2.7063	0.1000
Brand 2 Price	1	0.20074	0.09210	4.7512	0.0293
Brand 3 Price	1	0.13126	0.11487	1.3057	0.2532
Brand 4 Price	1	0.13478	0.07504	3.2255	0.0725
Other Price	0	0	.	.	.

---

The likelihood for this model is essentially the same as for the simpler, common-price-slope model fit previously,  $-2\log(\mathcal{L}_C) = 2425.214$  compared to 2424.812. You can test the null hypothesis that the two models are not significantly different by comparing their likelihoods. The difference between two  $-2\log(\mathcal{L}_C)$ 's (the number reported under 'With Covariates' in the output) has a chi-square distribution. We can get the  $df$  for the test by subtracting the two  $df$  for the two likelihoods. The difference  $2425.214 - 2424.812 = 0.402$  is distributed  $\chi^2$  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, the input data set for the first choice set is displayed by the following step:

```
proc print data=price(obs=5) label;
  run;
```

The results are as follows:

---

Brand Choice Example, Multinomial Logit Model										
Obs	p1	p2	p3	p4	p5	Set	Subj	Brand	Price	c
1	3.99	5.99	3.99	5.99	4.99	1	1	Brand 1	3.99	1
2	3.99	5.99	3.99	5.99	4.99	1	1	Brand 2	5.99	2
3	3.99	5.99	3.99	5.99	4.99	1	1	Brand 3	3.99	2
4	3.99	5.99	3.99	5.99	4.99	1	1	Brand 4	5.99	2
5	3.99	5.99	3.99	5.99	4.99	1	1	Other	4.99	2

---

The input consists of **Set**, **Subj**, **Brand**, **Price**, and a choice time variable **c**. In addition, it contains five variables **p1** through **p5**. The first observation of the **Price** variable shows us that the first alternative costs \$3.99; **p1** contains the cost of alternative 1, \$3.99, which is the same for all alternatives. It does not matter which alternative you are looking at, **p1** shows that alternative 1 costs \$3.99. Similarly, the second observation of the **Price** variable shows us that the second alternative costs \$5.99; **p2** contains the cost of alternative 2, \$5.99, which is the same for all alternatives. There is one price variable, **p1** through **p5**, for each of the five alternatives.

In all of the previous examples, we use models that are coded so that the utility of an alternative only depends on the attributes of that alternative. For example, the utility of Brand 1 only depends on the Brand 1 name and its price. In contrast, **p1**–**p5** contain information about each of the *other* alternatives' attributes. We 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 459, 468, 674, and 679 for other discussions of IIA.) The PROC TRANSREG step for coding the cross-effects model is as follows:



```

proc transreg design data=price nozeroconstant norestoremissing;
  model class(brand / zero=none separators=' ' ) | identity(price)
    identity(p1-p&m) *
      class(brand / zero=none lprefix=0 separators=' ' on ' ) /
    lprefix=0;
  output out=coded(drop=_type_ _name_ intercept);
  label price = 'Price'
    p1 = 'Brand 1' p2 = 'Brand 2' p3 = 'Brand 3'
    p4 = 'Brand 4' p5 = 'Other';
  id subj set c;
run;

```

The `class(brand / ...) | identity(price)` specification in the `model` statement is the same as the previous analysis. The additional terms, `identity(p1-p&m) * class(brand / ...)` create the cross-effects. The second value of the `separators=` option, `' on'` is used to create labels like `'Brand 1 on Brand 2'` instead of the default `'Brand 1 * Brand 2'`. It is important to note that you must specify the cross-effect by specifying `identity` with the price factors, followed by the asterisk, followed by `class` and the brand effect, *in that order*. The order of the specification determines the order in which brand names are added to the labels. Do not specify the brand variable first; doing so creates incorrect labels.

With  $m$  alternatives, there are  $m \times m$  cross-effects, but as we will see, many of them are zero. The first coded choice set is displayed with the following PROC PRINT steps:

```

title2 'Discrete Choice with Cross-Effects, Mother Logit';
proc format; value zer 0 = ' 0' 1 = ' 1'; run;
proc print data=coded(obs=5) label; var subj set c brand price; run;
proc print data=coded(obs=5) label; var Brand;;
  format brand: zer5.2 brand brand.; run;
proc print data=coded(obs=5) label; var p1B;; format p: zer5.2; id brand; run;
proc print data=coded(obs=5) label; var p2B;; format p: zer5.2; id brand; run;
proc print data=coded(obs=5) label; var p3B;; format p: zer5.2; id brand; run;
proc print data=coded(obs=5) label; var p4B;; format p: zer5.2; id brand; run;
proc print data=coded(obs=5) label; var p5B;; format p: zer5.2; id brand; run;

```

The coded data set contains the strata variable `Subj` and `Set`, choice time variable `c`, and `Brand` and `Price`. `Brand` and `Price` are used to create the coded independent variables but they are not used in the analysis with PROC PHREG.

The results are as follows:

---

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

Obs	Subj	Set	c	Brand	Price
1	1	1	1	Brand 1	3.99
2	1	1	2	Brand 2	5.99
3	1	1	2	Brand 3	3.99
4	1	1	2	Brand 4	5.99
5	1	1	2	Other	4.99

---

The effects 'Brand 1' through 'Other' in the next output are the binary brand effect variables. They indicate the brand for each alternative. The effects 'Brand 1 Price' through 'Other Price' are alternative-specific price effects. They indicate the price for each alternative. All ten of these variables are independent variables in the analysis, and their names are part of the `&_trgind` macro variable list, as are all of the cross-effects that are described next. The results are as follows:

---

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

Obs	Brand 1	Brand 2	Brand 3	Brand 4	Other	Brand 1 Price	Brand 2 Price	Brand 3 Price	Brand 4 Price	Other Price	Brand
1	1	0	0	0	0	3.99	0	0	0	0	Brand 1
2	0	1	0	0	0	0	5.99	0	0	0	Brand 2
3	0	0	1	0	0	0	0	3.99	0	0	Brand 3
4	0	0	0	1	0	0	0	0	5.99	0	Brand 4
5	0	0	0	0	1	0	0	0	0	4.99	Other

---

The effects 'Brand 1 on Brand 1' through 'Brand 1 on Other' in the next output are the first five cross-effects.

The results are as follows:

---

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

Brand	Brand 1 on Brand 1	Brand 1 on Brand 2	Brand 1 on Brand 3	Brand 1 on Brand 4	Brand 1 on Other
Brand 1	3.99	0	0	0	0
Brand 2	0	3.99	0	0	0
Brand 3	0	0	3.99	0	0
Brand 4	0	0	0	3.99	0
Other	0	0	0	0	3.99

---

They represent the effect of Brand 1 at its price on the utility of each alternative. The label 'Brand  $n$  on Brand  $m$ ' is read as 'the effect of Brand  $n$  at its price on the utility of Brand  $m$ .' For the first choice set, these first five cross-effects consist entirely of zeros and \$3.99's, where \$3.99 is the price of Brand 1 in this choice set. The nonzero value is constant across all of the alternatives in each choice set since Brand 1 has only one price in each choice set. Notice the 'Brand 1 on Brand 1' term, which is the effect of Brand 1 at its price on the utility of Brand 1. Also notice the 'Brand 1 Price' effect, which is shown in the previous output. The description "the effect of Brand 1 at its price on the utility of Brand 1" is just a convoluted way of describing the Brand 1 price effect. The 'Brand 1 on Brand 1' cross-effect is the same as the Brand 1 price effect, hence when we do the analysis, we see that the coefficient for the 'Brand 1 on Brand 1' cross-effect is zero.

The effects 'Brand 2 on Brand 1' through 'Brand 2 on Other' in the next output are the next five cross-effects. The results are as follows:

---

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

Brand	Brand 2 on Brand 1	Brand 2 on Brand 2	Brand 2 on Brand 3	Brand 2 on Brand 4	Brand 2 on Other
Brand 1	5.99	0	0	0	0
Brand 2	0	5.99	0	0	0
Brand 3	0	0	5.99	0	0
Brand 4	0	0	0	5.99	0
Other	0	0	0	0	5.99

---

They represent the effect of Brand 2 at its price on the utility of each alternative. For the first choice set, these five cross-effects consist entirely of zeros and \$5.99's, where \$5.99 is the price of Brand 2 in this choice set. The nonzero value is constant across all of the alternatives in each choice set since Brand 2 has only one price in each choice set. Notice the 'Brand 2 on Brand 2' term, which is the

effect of Brand 2 at its price on the utility of Brand 2. The description “the effect of Brand 2 at its price on the utility of Brand 2” is just a convoluted way of describing the Brand 2 price effect. The 'Brand 2 on Brand 2' cross-effect is the same as the Brand 2 price effect, hence when we do the analysis, we see that the coefficient for the 'Brand 2 on Brand 2' cross-effect is zero.

The effects 'Brand 3 on Brand 1' through 'Brand 3 on Other' in the next output are the next five cross-effects. The results are as follows:

---

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

Brand	Brand 3 on Brand 1	Brand 3 on Brand 2	Brand 3 on Brand 3	Brand 3 on Brand 4	Brand 3 on Other
Brand 1	3.99	0	0	0	0
Brand 2	0	3.99	0	0	0
Brand 3	0	0	3.99	0	0
Brand 4	0	0	0	3.99	0
Other	0	0	0	0	3.99

---

They represent the effect of Brand 3 at its price on the utility of each alternative. For the first choice set, these five cross-effects consist entirely of zeros and \$3.99's, where \$3.99 is the price of Brand 3 in this choice set. Notice that the 'Brand 3 on Brand 3' term is the same as the Brand 3 price effect, hence when we do the analysis, see that the coefficient for the 'Brand 3 on Brand 3' cross-effect is zero.

The remaining cross-effects follow the same pattern that was described for the previous cross-effects and are as follows:

---

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

Brand	Brand 4 on Brand 1	Brand 4 on Brand 2	Brand 4 on Brand 3	Brand 4 on Brand 4	Brand 4 on Other
Brand 1	5.99	0	0	0	0
Brand 2	0	5.99	0	0	0
Brand 3	0	0	5.99	0	0
Brand 4	0	0	0	5.99	0
Other	0	0	0	0	5.99

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

Brand	Other on Brand 1	Other on Brand 2	Other on Brand 3	Other on Brand 4	Other on Other
Brand 1	4.99	0	0	0	0
Brand 2	0	4.99	0	0	0
Brand 3	0	0	4.99	0	0
Brand 4	0	0	0	4.99	0
Other	0	0	0	0	4.99

We have been describing variables by their labels. While it is not necessary to look at it, the `&_trgind` macro variable name list that PROC TRANSREG creates for this problem is as follows:

```
%put &_trgind;

BrandBrand_1 BrandBrand_2 BrandBrand_3 BrandBrand_4 BrandOther
BrandBrand_1Price BrandBrand_2Price BrandBrand_3Price BrandBrand_4Price
BrandOtherPrice p1BrandBrand_1 p1BrandBrand_2 p1BrandBrand_3 p1BrandBrand_4
p1BrandOther p2BrandBrand_1 p2BrandBrand_2 p2BrandBrand_3 p2BrandBrand_4
p2BrandOther p3BrandBrand_1 p3BrandBrand_2 p3BrandBrand_3 p3BrandBrand_4
p3BrandOther p4BrandBrand_1 p4BrandBrand_2 p4BrandBrand_3 p4BrandBrand_4
p4BrandOther p5BrandBrand_1 p5BrandBrand_2 p5BrandBrand_3 p5BrandBrand_4
p5BrandOther
```

The analysis proceeds in exactly the same manner as before as follows:

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

The results are as follows:

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

The PHREG Procedure

Model Information

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

Number of Observations Read 4000  
 Number of Observations Used 4000

Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Pattern	Number of Choices	Number of Alternatives	Chosen Alternatives	Not Chosen
1	800	5	1	4

Convergence Status

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

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	2575.101	2349.325
AIC	2575.101	2389.325
SBC	2575.101	2483.018

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	225.7752	20	<.0001
Score	218.4500	20	<.0001
Wald	190.0257	20	<.0001

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Brand 1	1	1.24963	1.31259	0.9064	0.3411
Brand 2	1	-0.16269	1.38579	0.0138	0.9065
Brand 3	1	-3.90179	1.56511	6.2150	0.0127
Brand 4	1	2.49435	1.25537	3.9480	0.0469
Other	0	0	.	.	.
Brand 1 Price	1	0.51056	0.13178	15.0096	0.0001
Brand 2 Price	1	-0.04920	0.13411	0.1346	0.7137
Brand 3 Price	1	-0.27594	0.15517	3.1623	0.0754
Brand 4 Price	1	0.28951	0.12192	5.6389	0.0176
Other Price	0	0	.	.	.
Brand 1 on Brand 1	0	0	.	.	.
Brand 1 on Brand 2	1	0.51651	0.13675	14.2653	0.0002
Brand 1 on Brand 3	1	0.66122	0.15655	17.8397	<.0001
Brand 1 on Brand 4	1	0.32806	0.12664	6.7105	0.0096
Brand 1 on Other	0	0	.	.	.

Brand 2 on Brand 1	1	-0.39876	0.12832	9.6561	0.0019
Brand 2 on Brand 2	0	0	.	.	.
Brand 2 on Brand 3	1	-0.01755	0.15349	0.0131	0.9090
Brand 2 on Brand 4	1	-0.33802	0.12220	7.6512	0.0057
Brand 2 on Other	0	0	.	.	.
Brand 3 on Brand 1	1	-0.43868	0.13119	11.1823	0.0008
Brand 3 on Brand 2	1	-0.31541	0.13655	5.3356	0.0209
Brand 3 on Brand 3	0	0	.	.	.
Brand 3 on Brand 4	1	-0.54854	0.12528	19.1723	<.0001
Brand 3 on Other	0	0	.	.	.
Brand 4 on Brand 1	1	0.24398	0.12781	3.6443	0.0563
Brand 4 on Brand 2	1	-0.01214	0.13416	0.0082	0.9279
Brand 4 on Brand 3	1	0.40500	0.15285	7.0211	0.0081
Brand 4 on Brand 4	0	0	.	.	.
Brand 4 on Other	0	0	.	.	.
Other on Brand 1	0	0	.	.	.
Other on Brand 2	0	0	.	.	.
Other on Brand 3	0	0	.	.	.
Other on Brand 4	0	0	.	.	.
Other on Other	0	0	.	.	.

---

The results consist of:

- four nonzero brand effects and a zero for the constant alternative
- four nonzero alternative-specific price effects and a zero for the constant alternative
- $5 \times 5 = 25$  cross-effects, the number of alternatives squared, but only  $(5 - 1) \times (5 - 2) = 12$  of them are nonzero (four brands not counting Other affecting each of the remaining three brands).
  - There are three cross-effects for the effect of Brand 1 on Brands 2, 3, and 4.
  - There are three cross-effects for the effect of Brand 2 on Brands 1, 3, and 4.
  - There are three cross-effects for the effect of Brand 3 on Brands 1, 2, and 4.
  - There are three cross-effects for the effect of Brand 4 on Brands 1, 2, and 3.

All coefficients for the constant (other) alternative are zero as are the cross-effects of a brand on itself.

The mother logit model is used to test for violations of IIA (independence from irrelevant alternatives). IIA means the odds of choosing alternative  $c_i$  over  $c_j$  do not depend on the other alternatives in the choice set. Ideally, this more general model will not significantly explain more variation in choice than the restricted models. Also, if IIA is satisfied, few if any of the cross-effect terms should be significantly different from zero. (See pages 452, 468, 674, and 679 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 as follows:

```
proc transreg design data=price nozeroconstant noestoremissing;
  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 as follows:

```
proc transreg design data=price nozeroconstant noestoremissing;
  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) is 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. The following 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 is 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;

```

The results are as follows:

---

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`. The following steps code and analyze the data:

```
proc transreg design data=price2 nozeroconstant norestoremissing;
  model class(brand / zero=none) identity(price) / lprefix=0;
  output out=coded(drop=_type_ _name_ intercept);
  label price = 'Price';
  id freq set c;
run;

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

These steps produced the following results:

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

The PHREG Procedure

Model Information

Data Set	WORK.CODED
Dependent Variable	c
Censoring Variable	c
Censoring Value(s)	2
Frequency Variable	Freq
Ties Handling	BRESLOW
Number of Observations Read	80
Number of Observations Used	80
Sum of Frequencies Read	4000
Sum of Frequencies Used	4000

Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Stratum	Set	Number of Alternatives	Chosen Alternatives	Not Chosen
1	1	500	100	400
2	2	500	100	400
3	3	500	100	400
4	4	500	100	400
5	5	500	100	400
6	6	500	100	400
7	7	500	100	400
8	8	500	100	400
-----				
Total		4000	800	3200

Convergence Status

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

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	9943.373	9793.486
AIC	9943.373	9803.486
SBC	9943.373	9826.909

## Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	149.8868	5	<.0001
Score	153.2328	5	<.0001
Wald	142.9002	5	<.0001

## Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Brand 1	1	0.66727	0.12305	29.4065	<.0001
Brand 2	1	0.38503	0.12962	8.8235	0.0030
Brand 3	1	-0.15955	0.14725	1.1740	0.2786
Brand 4	1	0.98964	0.11720	71.2993	<.0001
Other	0	0	.	.	.
Price	1	0.14966	0.04406	11.5379	0.0007

The summary table is small with eight rows, one row per choice set. Each row represents 100 chosen alternatives and 400 unchosen. The 'Analysis of Maximum Likelihood Estimates' table exactly matches the one produced by the standard analysis. The  $-2$  LOG L statistics are different than before: 9793.486 now compared to 2425.214 previously. This is because the data are arrayed in this example so that the partial likelihood of the proportional hazards model fit by PROC PHREG with the `ties=breslow` option is now proportional to—not identical to—the likelihood for the choice model. However, the Model Chi-Square statistics,  $df$ , and  $p$ -values are the same as before. The two corresponding pairs of  $-2$  LOG L's differ by a constant  $9943.373 - 2575.101 = 9793.486 - 2425.214 = 7368.272 = 2 \times 800 \times \log(100)$ . Since the  $\chi^2$  is the  $-2$  LOG L without covariates minus  $-2$  LOG L with covariates, the constants cancel and the  $\chi^2$  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 as follows:

```
proc transreg design data=price2 nozeroconstant norestoremising;
  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:

---

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

The PHREG Procedure

Model Information

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

Number of Observations Read	80
Number of Observations Used	80
Sum of Frequencies Read	4000
Sum of Frequencies Used	4000

Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Stratum	Set	Number of Alternatives	Chosen Alternatives	Not Chosen
1	1	500	100	400
2	2	500	100	400
3	3	500	100	400
4	4	500	100	400
5	5	500	100	400
6	6	500	100	400
7	7	500	100	400
8	8	500	100	400
Total		4000	800	3200

Convergence Status

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

Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	9943.373	9793.084
AIC	9943.373	9809.084
SBC	9943.373	9846.561

## Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	150.2891	8	<.0001
Score	154.2562	8	<.0001
Wald	143.1425	8	<.0001

## Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Brand 1	1	-0.00972	0.43555	0.0005	0.9822
Brand 2	1	-0.62230	0.48866	1.6217	0.2028
Brand 3	1	-0.81250	0.60318	1.8145	0.1780
Brand 4	1	0.31778	0.39549	0.6456	0.4217
Other	0	0	.	.	.
Brand 1 Price	1	0.13587	0.08259	2.7063	0.1000
Brand 2 Price	1	0.20074	0.09210	4.7512	0.0293
Brand 3 Price	1	0.13126	0.11487	1.3057	0.2532
Brand 4 Price	1	0.13478	0.07504	3.2255	0.0725
Other Price	0	0	.	.	.

The only thing that changes from the analysis with one stratum for each subject and choice set combination is the likelihood.

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  $\chi^2$  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  $-2 \text{ LOG L}$  values differ by a constant with aggregate data versus individual data. It can 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 are 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 is 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  $k$ th choice set, is the same as the likelihood for the choice model, except for a multiplicative constant.

$$L_k^C = N^N L_k^B = 100^{100} L_k^B$$

Therefore, the Breslow likelihood for all eight choice sets is

$$\mathcal{L}_B = \prod_{k=1}^8 L_k^B = N^{-8N} \mathcal{L}_C = 100^{-800} \mathcal{L}_C$$

The two likelihoods are not exactly the same, because each choice set is designated as a separate stratum, instead of each choice set within each subject.

The log likelihood for the choice model is

$$\begin{aligned} \log(\mathcal{L}_C) &= 800 \times \log(100) + \log(\mathcal{L}_B), \\ \log(\mathcal{L}_C) &= 800 \times \log(100) + (-0.5) \times 9793.486, \\ \log(\mathcal{L}_C) &= -1212.607 \end{aligned}$$

and  $-2 \log(\mathcal{L}_C) = 2425.214$ , which matches the earlier output. However, it is usually not necessary to obtain this value.

## Food Product Example with Asymmetry and Availability Cross-Effects

This example is based on the choice example from page 255. This example discusses the multinomial logit model, number of parameters, choosing the number of choice sets, designing the choice experiment, long design searches, examining the design, examining the subdesigns, examining the aliasing structure, blocking the design, testing the design before data collection, generating artificial data, processing the data, coding, cross-effects, availability, multinomial logit model results, modeling subject attributes, results, and interpretation.

Consider the problem of using a discrete choice model to study the effect of introducing a retail food product. This might be useful, for instance, to refine a marketing plan or to optimize a product prior to test market. A typical brand team has 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 might also want to know what brand attributes have competitive clout and want to identify competitive attributes to which they are vulnerable.

To develop this further, assume our client wishes to introduce a line extension in the category of frozen entrées. The client has one nationally branded competitor, a regional competitor in each of three regions, and a profusion of private label products at the grocery chain level. The product can 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 might 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 draws attention to their product.

### The Multinomial Logit Model

This problem can be set up as a discrete choice model in which a respondent's choice among brands, given choice set  $C_a$  of available brands, corresponds 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:

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

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

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

One of the consequences of the MNL formulation is the property of independence from irrelevant alternatives (IIA). Under the assumption of IIA, all cross-effects are assumed to be equal, so that if a brand gains in utility, it draws share from all other brands in proportion to their current shares. Departures from IIA exist when certain subsets of brands are in more direct competition and tend to draw a disproportionate amount of share from each other than from other members in the category.



IIA is frequently described using a transportation example. Say you have three alternatives for getting to work: bicycle, car, or a blue bus. If a fourth alternative became available, a red bus, then according to IIA the red bus should draw riders from the other alternatives in proportion to their current usage. However, in this case, IIA is violated, and instead the red bus draws more riders from the blue bus than from car drivers and bicycle riders.

The mother logit formulation of McFadden (1974) can be used to capture departures from IIA. In a mother logit model, the utility for brand  $i$  is a function of both the attributes of brand  $i$  and the attributes of other brands. The effect of one brand's attributes on another is termed a cross-effect. In the case of designs in which only subsets  $C_a$  of the full shelf set  $C$  appear, the effect of the presence/absence of one brand on the utility of another is termed an *availability cross-effect*. (See pages 452, 459, 674, and 679 for other discussions of IIA.)

## Set Up

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

Factors and Levels

Alternative	Factor	Levels	Brand	Description
1	X1	4	Client	1.29, 1.69, 2.09 + absent
2	X2	4	Client Line Extension	1.39, 1.89, 2.39, + absent microwave/stove-top shelf talker yes/no
	X3	2		
	X4	2		
3	X5	3	Regional	1.99, 2.49 + absent
4	X6	3	Private Label	1.49, 2.29 absent microwave/stove-top
	X7	2		
5	X8	3	National	1.99 + 2.39 + absent

In addition to intercepts and main effects, we also require that all two-way interactions within alternatives be estimable:  $x_2 \times x_3$ ,  $x_2 \times x_4$ ,  $x_3 \times x_4$  for the line extension and  $x_6 \times x_7$  for private labels. This enables 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  $x_1$ – $x_8$ , 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  $k = 1, \dots, K_i$

$x_{ik}$  = 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  $l = 1, \dots, L_j$

$x_{jl}$  = level of attribute  $l$  for brand  $j$ .

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

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

The subdesign sizes are computed by taking the floor of the number of runs from the marginal times the expected proportion of runs in which the alternative appears. For example, for the client brand, which has three prices and not available and 22 runs,  $\text{floor}(22 \times 3/4) = 16$ ; for the competitor and 32 runs,  $\text{floor}(32 \times 2/3) = 21$ . The number of runs chosen is  $n=26$ . This number provides adequate degrees of freedom for the linear price model and also permits estimation of direct quadratic price effects. Estimating quadratic cross-effects for price requires 32 runs at the very least. Although the technique of using two-way interactions between nominal level variables usually guarantees that all direct and cross-effects are estimable, it is sometimes necessary and good practice to check the ranks of the subdesigns for more complex models (Lazari and Anderson 1994).

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

## Designing the Choice Experiment

This example originated with Kuhfeld, Tobias, and Garratt (1994), long before the `%MktRuns` macro existed. At least for now, we will skip the customary step of running the `%MktRuns` macro to suggest a design size and instead use the original size of 26 choice sets.

We use the `%MktEx` autocall macro to create the design. (All of the autocall macros used in this book are documented starting on page 803.) To recap, we want to make the design  $2^3 3^3 4^2$  in 26 runs, and we want the following interactions to be estimable: `x2*x3` `x2*x4` `x3*x4` `x6*x7`. Furthermore, there are restrictions on the design. Each of the price variables, `x1`, `x2`, `x5`, `x6`, and `x8`, has one level—the maximum level—that indicates the alternative is not available in the choice set. We use this to create choice sets with 2, 3, or 4 alternatives available. If `(x1 < 4)` then the first alternative is available, if `(x2 < 4)` then the second alternative is available, if `(x5 < 3)` then the third alternative is available, and so on. A Boolean term such as `(x1 < 4)` is one when true and zero otherwise. Hence,

$$((x1 < 4) + (x2 < 4) + (x5 < 3) + (x6 < 3) + (x8 < 3))$$

is the number of available alternatives. It is simply the sum of some 1's if available and 0's if not available.

We impose restrictions with the `%MktEx` macro by writing a macro, with IML statements, that quantifies the badness of each run (or in this case, each choice set). We do this so `bad = 0` is good and values larger than zero are increasingly worse. We write our restrictions using an IML row vector `x` that contains the levels (integers beginning with 1) of each of the factors in the *i*th choice set, the one the macro is currently seeking to improve. The *j*th factor is `x[j]`, or we can also use the factor names (for example, `x1`, `x2`). (See pages 604 and 1079 for other examples of restrictions.)

We must use IML logical operators, which do not have all of the same syntactical alternatives as DATA step operators:

Specify	For	Do Not Specify
=	equals	EQ
$\wedge =$ or $\neg =$	not equals	NE
<	less than	LT
<=	less than or equal to	LE
>	greater than	GT
>=	greater than or equal to	GE
&	and	AND
	or	OR
$\wedge$ or $\neg$	not	NOT
$a <= b$ & $b <= c$	range check	$a <= b <= c$

To restrict the design, we must specify `restrictions=macro-name`, in this case `restrictions=resmac`, that names the macro that quantifies badness. The first statement counts up the number of available alternatives. The next two set the actual badness value. Note that the `else bad = 0` statement is not necessary since `bad` is automatically initialized to zero by the `%MktEx` macro. If the number available is less than two or greater than 4, then `bad` gets set to the absolute difference between the number available and 3. Hence, zero available corresponds to `bad = 3`, one available corresponds to `bad = 2`, two through four available corresponds to `bad = 0`, and five available corresponds to `bad = 2`. Do not just set `bad` to zero when everything is fine and one otherwise, because the macro needs to know that when it switches from zero available to one available, it is going in the right direction. For simple restrictions like this, it does not matter very much. However, for complicated sets of restrictions, it is critical that the `bad` variable is set to a count of the number of current restriction violations. The following steps create the design:\*

```

title 'Consumer Food Product Example';

%macro resmac;
  navail = (x1 < 4) + (x2 < 4) + (x5 < 3) + (x6 < 3) + (x8 < 3);
  if (navail < 2) | (navail > 4) then bad = abs(navail - 3);
  else
      bad = 0;
%mend;

%mktx(4 4 2 2 3 3 2 3,          /* all attrs of all alternatives */
      n=26,                    /* number of choice sets */
      interact=x2*x3 x2*x4 x3*x4 x6*x7, /* interactions */
      restrictions=resmac,      /* name of restrictions macro */
      seed=377,                /* random number seed */
      outr=sasuser.EntreeLinDes1) /* randomized design stored permanently */

```

---

\*Due to machine, SAS release, and macro differences, you might not get exactly the same design used in this book, but the differences should be slight.

The initial messages that the macro displays are as follows:

NOTE: Generating the fractional-factorial design, n=27.

NOTE: Generating the candidate set.

NOTE: Performing 60 searches of 2,776 candidates, full-factorial=3,456.

The orthogonal array initialization part of the coordinate-exchange algorithm iterations initializes the first 26 rows of a 27 run fractional-factorial design. This design has 13 three-level factors, ten of which are used to make  $2^33^34^2$ . The initial design is unbalanced and one row short of orthogonal, so we would expect other methods to be better for this problem. The macro also tells us that it is performing 60 PROC OPTEX searches of 2776 candidates, and that the full-factorial design has 3456 runs. The macro is searching the full-factorial design minus the excluded choice sets. Since the full-factorial design is not too large (less than 5000), and since there is no orthogonal array that is very good for this problem, this is the kind of problem where we expect the PROC OPTEX modified Fedorov algorithm (Fedorov 1972; Cook and Nachtsheim 1980) algorithm to work best. The macro chose 60 OPTEX iterations. In the fabric softener example, the macro did not try any OPTEX iterations, because it knew it could directly make a 100% *D*-efficient design. In the vacation examples, it ran the default minimum of 20 OPTEX iterations because the macro's heuristics concluded that OPTEX is probably not be the best approach for those problems. In this example, the macro's heuristics tried more iterations, since this is the kind of example where OPTEX works best.

Some of the output is as follows:

---

#### Consumer Food Product Example

##### Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
1	Start	84.3176		Can
1	2 1	84.3176	84.3176	Conforms
1	End	84.3176		
2	Start	27.8626		Tab,Unb,Ran
2	1 1	76.5332		Conforms
2	End	80.4628		
.				
.				
.				
11	Start	24.5507		Tab,Ran
11	26 1	78.6100		Conforms
11	End	81.8604		
12	Start	26.3898		Ran,Mut,Ann
12	1 1	67.0450		Conforms
12	End	83.0114		
.				
.				
.				

21	Start	45.9310		Ran,Mut,Ann
21	15 1	67.1046		Conforms
21	End	82.1657		

NOTE: Performing 600 searches of 2,776 candidates.

Consumer Food Product Example

Design Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
0	Initial	84.3176	84.3176	Ini
1	Start	84.7548		Can
1	2 1	84.7548	84.7548	Conforms
1	End	84.7548		

Consumer Food Product Example

Design Refinement History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
0	Initial	84.7548	84.7548	Ini
1	Start	84.7548		Pre,Mut,Ann
1	2 1	82.6737		Conforms
1	14 1	84.7548	84.7548	
1	End	82.6386		
.				
.				
.				
8	Start	84.7548		Pre,Mut,Ann
8	2 1	84.7548	84.7548	Conforms
8	14 1	84.7548	84.7548	
8	21 2	84.7548	84.7548	
8	12 3	84.7548	84.7548	
8	12 6	84.7548	84.7548	
8	18 1	84.7548	84.7548	
8	2 2	84.7548	84.7548	
8	End	84.7548		

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

## Consumer Food Product Example

## The OPTEX Procedure

## Class Level Information

Class	Levels	-Values-
x1	4	1 2 3 4
x2	4	1 2 3 4
x3	2	1 2
x4	2	1 2
x5	3	1 2 3
x6	3	1 2 3
x7	2	1 2

## Consumer Food Product Example

## The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	84.7548	71.1686	98.0583	0.9806

Design 1 (**Can**), which is created by the candidate-set search (using PROC OPTEX), has *D*-efficiency of 84.3176, and the macro confirms that the design conforms to our restrictions. The orthogonal array, unbalanced, and random initializations do not work as well. For each design, the macro iteration history states the *D*-efficiency for the initial design (27.8626 in design 2), the *D*-efficiency when the restrictions are met (76.5332, **Conforms**), and the *D*-efficiency for the final design (80.4628). The fully-random initialization tends to work a little better than the orthogonal array initialization for this problem, but not as well as PROC OPTEX. At the end of the algorithm search phase, the macro decides to use PROC OPTEX and performs 600 more searches, and it finds a design with 84.7548% *D*-efficiency. The design refinement step fails to improve on the best design. This step took 3.5 minutes.

## Restrictions Formulated Using Actual Attribute Names and Levels

In the previous section, we constructed a macro to create a design, maximizing efficiency while forcing the design to conform to a set of restrictions. We provide %MktEx with a macro that it used to evaluate how well it is doing in imposing the restrictions. Internally, %MktEx stores its results in matrices, vectors, and scalars, with fixed names: **x1**, **x2**, **x3**, and so on. Levels of the factors are always positive integers starting with one. In practice, before you actually use the design, you will usually want to change these names and levels to something else that is more meaningful to your particular problem. In some cases, it might be more convenient to express your restrictions in terms of these more meaningful names and levels as well. To do that, you need to convert the %MktEx names and levels into your names

and levels, for example, as follows:

```
%macro resmac;
  client    = {1.29 1.69 2.09 .}[x1];
  extension = {1.39 1.89 2.39 .}[x2];
  regional  = {1.99 2.49      .}[x5];
  private   = {1.49 2.29      .}[x6];
  national  = {1.99 2.39      .}[x8];

  navail    = (client ^= .) + (extension ^= .) + (regional ^= .) +
              (private ^= .) + (national ^= .);
  if (navail < 2) | (navail > 4) then bad = abs(navail - 3);
  else
                                bad = 0;
%mend;
```

The first line creates a scalar called `client`, that has one of four values: 1.29 when `x1` is 1, 1.69 when `x1` is 2, 2.09 when `x1` is 3, and `.` (missing) when `x1` is 4. Each of the first five assignment statements uses the `%MktEx` factor variables as an index into a constant vector of levels and stores the result in the attribute name so that restrictions can be posed in terms of the actual design problem. A literal vector consists of `{ list }` and you can provide a subscript variable to index any vector, including a literal vector by specifying `[ variable ]`. Hence, the first five statements just convert `x1`, `x2`, `x5`, `x6`, and `x8` into names and levels that are familiar to the problem. Character vectors are allowed as well. The following step shows an alternative but equivalent formulation of the macro, treating the first two factors as numeric and the rest as character:

```
%macro resmac;
  client    = {1.29 1.69 2.09 .}[x1];
  extension = {1.39 1.89 2.39 .}[x2];
  regional  = {"$1.99" "$2.49" "NA"}[x5];
  private   = {"$1.49" "$2.29" "NA"}[x6];
  national  = {"$1.99" "$2.39" "NA"}[x8];

  navail    = (client ^= .) + (extension ^= .) + (regional ^= "NA") +
              (private ^= "NA") + (national ^= "NA");
  if (navail < 2) | (navail > 4) then bad = abs(navail - 3);
  else
                                bad = 0;
%mend;
```

You cannot mix quoted character strings and unquoted numbers within a vector, however you can put any value in quotes. There is one extremely important caveat. Several names are used internally by `%MktEx`, and are available for you to look at when writing your restrictions. You must not change them. Do not use any of these names as names of anything you create when you write restrictions macros: `i`, `try`, `x`, `x1 x2 ...`, `j1 j2 ...`, `xmat`, `__pbad`, `__bbad`.

Macros like this are always be less efficient than macros based on the internal names since extra assignments are needed and the statements in the macro are evaluated **many** times. However, the slight loss of computer efficiency might be more than offset by the convenience of avoiding converting your restrictions to use internal names. It might not matter much for a problem such as this, but for complicated restrictions, it might be much easier to work with more informative names and levels. You can make the macro slightly more efficient, at a cost of a slightly more complicated looking restrictions



macro, by using `options=nox` to suppress the creation of `x1`, `x2`, and so on, and instead make the mnemonic names and levels directly based on `x`. The macro must be changed, substituting `x[1]` for `x1`, `x[2]` for `x2`, and so on, as follows:

```
%macro resmac;
  client    = {1.29 1.69 2.09 .}[x[1]];
  extension = {1.39 1.89 2.39 .}[x[2]];
  regional  = {"$1.99" "$2.49" "NA"}[x[5]];
  private   = {"$1.49" "$2.29" "NA"}[x[6]];
  national  = {"$1.99" "$2.39" "NA"}[x[8]];

  navail    = (client ^= .) + (extension ^= .) + (regional ^= "NA") +
              (private ^= "NA") + (national ^= "NA");
  if (navail < 2) | (navail > 4) then bad = abs(navail - 3);
  else
              bad = 0;
%mend;

%mktx(4 4 2 2 3 3 2 3,          /* all attrs of all alternatives */
      n=26,                    /* number of choice sets */
      interact=x2*x3 x2*x4 x3*x4 x6*x7, /* interactions */
      optiter=0,               /* no PROC OPTEX iterations */
      tabiter=0,               /* no OA initialization iterations */
      maxdesigns=10,           /* make at most 10 designs */
      options=nox,            /* suppress x1, x2, ... creation */
      restrictions=resmac,    /* name of restrictions macro */
      seed=377)                /* random number seed */
```

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

With a moderate sized candidate set such as this one (2000 to 6000 runs), we might be able to do better with more iterations. To test this, PROC OPTEX was run 10,000 times over the winter holiday vacation, from December 22 through January 2, creating a total of 200,000 designs, 20 designs on each try. (This was many years ago on computers that were much slower than the ones we have today.) The following table provides a summary of the results:

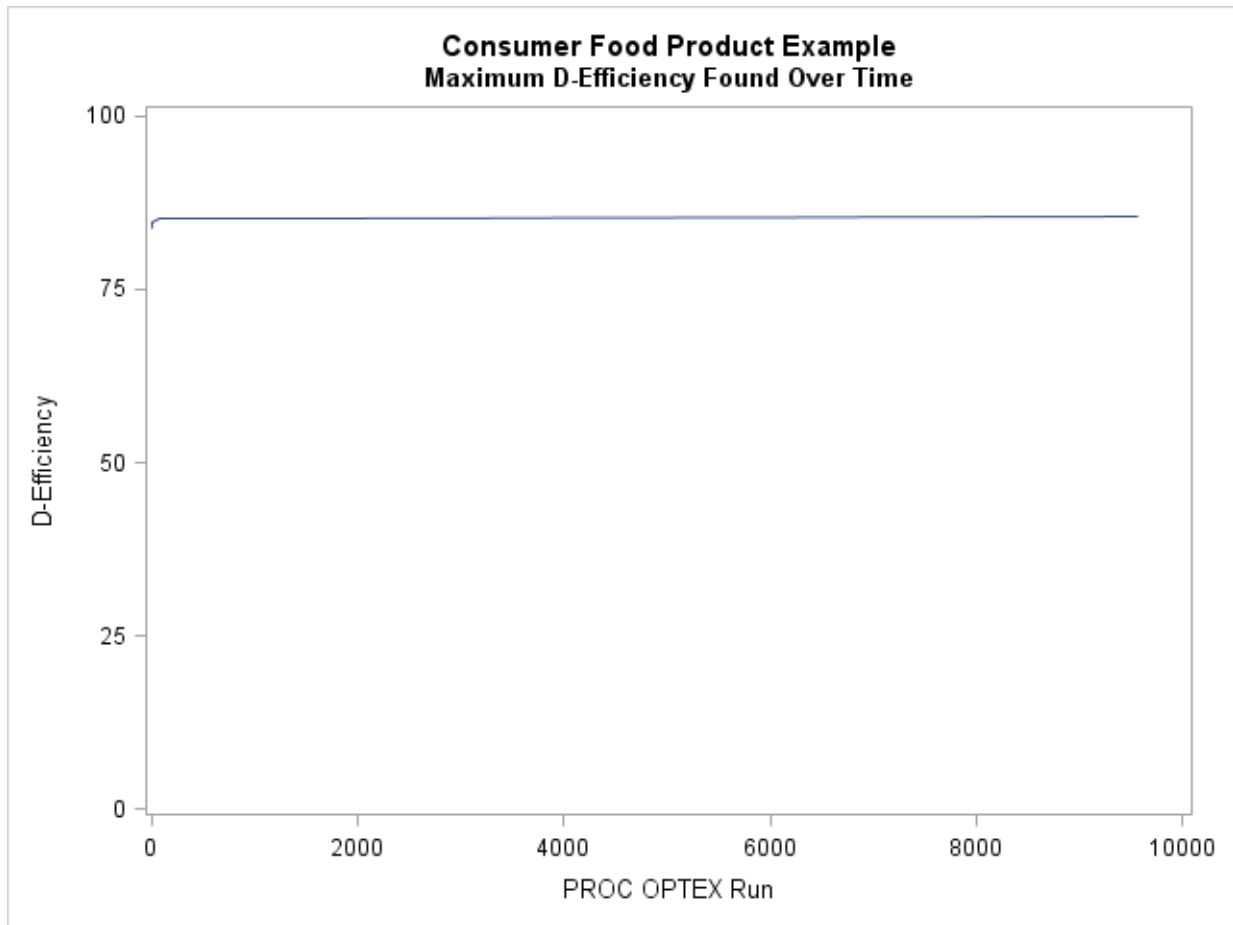
PROC OPTEX Run	<i>D</i> -Efficiency	Percent Improvement
1	83.8959	
2	83.9890	0.11%
3	84.3763	0.46%
6	84.7548	0.45%
84	85.1561	0.47%
1535	85.3298	0.20%
9576	85.3985	0.08%

This example is interesting, because it shows the diminishing value of increasing the number of iterations. Six minutes into the search, in the first six passes through PROC OPTEX ( $6 \times 20 = 120$  total iterations), we found a design with reasonably good  $D$ -efficiency=84.7548. Over an hour into the search, with  $(84 - 6) \times 20 = 1560$  more iterations, we get a small 0.47% increase in  $D$ -efficiency to 85.1561. About one day into the search, with  $(1535 - 84) \times 20 = 29,020$  more iterations, we get another small 0.20% increase in  $D$ -efficiency, 85.3298. Finally, almost a week into the search, with  $(9576 - 1535) \times 20 = 160,820$  more iterations, we get another small 0.08% increase in  $D$ -efficiency to 85.3985. Our overall improvement over the best design found in 120 iterations was 0.75952%, about three-quarters of a percent. These numbers will change with other problems and other seeds. However, as these results show, usually the first few iterations give you a good, efficient design, and usually, subsequent iterations give you slight improvements but with a cost of much greater run times. Next, we construct a plot of this table as follows:

```
data e;
  input n e;
  label n = 'PROC OPTEX Run' e = 'D-Efficiency';
  datalines;
    1 83.8959
    2 83.9890
    3 84.3763
    6 84.7548
   84 85.1561
 1535 85.3298
 9576 85.3985
;

proc sgplot data=e;
  title 'Consumer Food Product Example';
  title2 'Maximum D-Efficiency Found Over Time';
  series y=e x=n;
  yaxis values=(0 to 100 by 25);
run;
```

The plot of maximum  $D$ -efficiency as a function of PROC OPTEX run number clearly shows that the gain in  $D$ -efficiency that comes from a large number of iterations is very slight.



If you have a lot of time to search for a good design, you can specify some of the time and maximum number of iteration parameters. Sometimes you might get lucky and find a better design. In this next example, `maxtime=300 300 60` is specified. This gives the macro up to 300 minutes for the algorithm search step, 300 minutes for the design search step, and 60 minutes for the refinement step. The option `maxiter=` increases the number iterations to 10000 for each of the three steps (or the maximum time). With this specification, you expect the macro to run overnight. See the macro documentation (starting on page 1017) for more iteration options. Note that you must increase the number of iterations and the maximum amount of time if you want the macro to run longer. With this specification, the macro performs 1800 OPTEX iterations initially (compared to 60 by default). The following steps create the restrictions macro and search for a design:

```

title 'Consumer Food Product Example';

%macro resmac;
  navail = (x1 < 4) + (x2 < 4) + (x5 < 3) + (x6 < 3) + (x8 < 3);
  if (navail < 2) | (navail > 4) then bad = abs(navail - 3);
  else
    bad = 0;
  %mend;

%mktx(4 4 2 2 3 3 2 3,          /* all attrs of all alternatives */
      n=26,                    /* number of choice sets */
      interact=x2*x3 x2*x4 x3*x4 x6*x7, /* interactions */
      restrictions=resmac,      /* name of restrictions macro */
      seed=151,                /* random number seed */
      maxtime=300 300 60,      /* max time for each major search phase */
      maxiter=10000)           /* max iterations for each phase */

```

The results from this step are not shown.

## Examining the Design

We can use the %MktEval macro to start to evaluate the design as follows:

```
%mkteval(data=sasuser.EntreeLinDes1)
```

The results are as follows:

---

Consumer Food Product Example								
Canonical Correlations Between the Factors								
There are 4 Canonical Correlations Greater Than 0.316								
	x1	x2	x3	x4	x5	x6	x7	x8
x1	1	0.30	0.20	0.11	0.42	0.26	0.09	0.33
x2	0.30	1	0.10	0.10	0.13	0.17	0.51	0.18
x3	0.20	0.10	1	0.08	0.09	0.30	0	0.10
x4	0.11	0.10	0.08	1	0.09	0.10	0	0.10
x5	0.42	0.13	0.09	0.09	1	0.24	0.05	0.43
x6	0.26	0.17	0.30	0.10	0.24	1	0.14	0.13
x7	0.09	0.51	0	0	0.05	0.14	1	0.14
x8	0.33	0.18	0.10	0.10	0.43	0.13	0.14	1

Consumer Food Product Example  
 Canonical Correlations > 0.316 Between the Factors  
 There are 4 Canonical Correlations Greater Than 0.316

		r	r Square
x2	x7	0.51	0.26
x5	x8	0.43	0.18
x1	x5	0.42	0.18
x1	x8	0.33	0.11

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

Frequencies

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

*	x5 x6	2 4 3 2 2 4 3 4 2
*	x5 x7	4 5 4 4 4 5
*	x5 x8	1 2 6 4 2 2 2 5 2
*	x6 x7	3 4 4 6 5 4
*	x6 x8	2 2 3 2 4 4 3 3 3
*	x7 x8	4 4 4 3 5 6
	N-Way	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
		1 1 1 1 1 1 1

---

Some of the canonical correlations are bigger than we would like. They all involve attributes in different alternatives, so they should not pose huge problems. Still, they are large enough to make some researchers uncomfortable. The frequencies are pretty close to balanced. Perfect balance is not possible with 26 choice sets and this design. If we are willing to consider blocking the design, we might do better with more choice sets.

## Designing the Choice Experiment, More Choice Sets

Let's run the %MktRuns macro to see what design size looks good. For now, we ignore the interactions and just specify main effects as follows:

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

The results are as follows:

---

### Consumer Food Product Example

#### Design Summary

Number of Levels	Frequency
2	3
3	3
4	2

### Consumer Food Product Example

Saturated = 16  
Full Factorial = 3,456

Some Reasonable Design Sizes	Violations	Cannot Be Divided By
144 *	0	
72	1	16
48	3	9
96	3	9
192	3	9
24	4	9 16
120	4	9 16
168	4	9 16
36	7	8 16
108	7	8 16
16 S	21	3 6 9 12

\* - 100% Efficient design can be made with the MktEx macro.

S - Saturated Design - The smallest design that can be made.

Note that the saturated design is not one of the recommended designs for this problem. It is shown to provide some context for the recommended sizes.

n	Design	Reference
144	2 **118 3 ** 4 4 ** 2	Orthogonal Array
144	2 **115 3 ** 4 4 ** 3	Orthogonal Array
144	2 **113 3 ** 12 4 ** 2	Orthogonal Array
144	2 **112 3 ** 8 4 ** 2 6 ** 1	Orthogonal Array
144	2 **112 3 ** 4 4 ** 4	Orthogonal Array
144	2 **111 3 ** 4 4 ** 2 6 ** 2	Orthogonal Array
144	2 **110 3 ** 12 4 ** 3	Orthogonal Array
.		
.		
.		

The smallest suggestion larger than 26 is 36. With this mix of factor levels, we would need 144 runs to get an orthogonal design (ignoring interactions), so we definitely want to stick with a nonorthogonal design. Balance is possible in 36 runs, but 36 cannot be divided by  $2 \times 4 = 8$  and  $4 \times 4 = 16$ . With 36 runs, a blocking factor is required (2 blocks of 18 or 3 blocks of 12). We would like the shelf talker to appear in half of the choice sets within block, so with two blocks, we want the number of choice sets to be divisible by  $2 \times 2 = 4$ , and 36 can be divided by 4. Now let's specify the interactions as follows:

```
%mktruns(4 4 2 2 3 3 2 3, interact=x2*x3 x2*x4 x3*x4 x6*x7)
```

The results are as follows:

Consumer Food Product Example

Design Summary

Number of Levels	Frequency
2	3
3	3
4	2

Consumer Food Product Example

Saturated = 25  
Full Factorial = 3,456

Some Reasonable Design Sizes	Violations	Cannot Be Divided By
144	2	32
96	5	9 18
192	5	9 18
48	7	9 18 32
72	9	16 32 48
216	9	16 32 48
120	14	9 16 18 32 48
168	14	9 16 18 32 48
36	25	8 16 24 32 48
108	25	8 16 24 32 48
25 S	61	2 3 4 6 8 9 12 16 18 24 32 48

S - Saturated Design - The smallest design that can be made.

Note that the saturated design is not one of the recommended designs for this problem. It is shown to provide some context for the recommended sizes.

Thirty-six runs is still in our list of possibilities, but now we see that not only can it not be divided by 8 and 16, it also cannot be divided by 24, 32, 48. We will try making a design in 36 runs and see how it looks.

In the previous try in 26 runs, the PROC OPTEX modified Fedorov algorithm worked best. There are two reasons why this probably happened. First, the full-factorial design is small enough to use as a candidate set. After imposing restrictions, the candidate set has 2,776 runs, and any size under 5000 or 10,000 is very manageable. Second, the design has interactions. The coordinate exchange algorithm by default considers only a single factor at a time, which is just one part of an interaction term. Modified Fedorov in contrast, considers exchanges involving all of the factors. For this problem, Modified Fedorov is invariably superior to the default coordinate-exchange algorithm. However, we can



make coordinate exchange better, by having it perform multiple-column exchanges taking into account the interactions, just as we did in the vacation example on page 417. We use the `order=matrix=SAS-data-set` approach to looping over the columns of the design with the coordinate-exchange algorithm. In this case, coordinate exchange pairs columns 1, 5, and 8 with a randomly chosen column, it considers every possible triple in columns 2, 3, and 4, and it pairs columns 6 and 7 with a randomly chosen column. The following steps generate the order matrix and search for a design:

```

title 'Consumer Food Product Example';

%macro resmac;
  navail = (x1 < 4) + (x2 < 4) + (x5 < 3) + (x6 < 3) + (x8 < 3);
  if (navail < 2) | (navail > 4) then bad = abs(navail - 3);
  else
      bad = 0;
%mend;

data mat;
  input a b c;
  datalines;
1 1 .
2 3 4
5 5 .
6 7 .
8 8 .
;

%mktx(4 4 2 2 3 3 2 3,          /* all attrs of all alternatives      */
      n=36,                      /* number of choice sets             */
      order=matrix=mat,          /* pairs/trips of cols to work on    */
      interact=x2*x3 x2*x4 x3*x4 x6*x7, /* interactions                       */
      restrictions=resmac,       /* name of restrictions macro         */
      seed=377,                  /* random number seed                */
      outr=sasuser.EntreeLinDes2) /* randomized design stored permanently */

%mkteval(data=sasuser.EntreeLinDes2)

```

A small part of the output from the `%MktEx` macro is as follows:

---

Consumer Food Product Example

1

Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
1	Start	94.0517		Can
1	2 1	94.0517	94.0517	Conforms
1	End	94.0517		
.				
.				
.				

12	Start	71.6955		Ran,Mut,Ann
12	1 1	78.5418		Conforms
12	30 5	94.1433	94.1433	
12	33 5	94.1507	94.1507	
12	31 1	94.1532	94.1532	
12	23 6	94.1553	94.1553	
12	End	94.1553		

Design Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
0	Initial	94.1553	94.1553	Ini
.				
.				
.				
3	Start	68.5288		Ran,Mut,Ann
3	29 1	75.9029		Conforms
3	22 5	94.1682	94.1682	
3	34 5	94.1683	94.1683	
3	35 6	94.2926	94.2926	
3	16 8	94.3718	94.3718	
3	24 6	94.3718	94.3718	
3	9 1	94.4572	94.4572	
3	End	94.2846		
.				
.				
.				

Consumer Food Product Example

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	94.4571	88.7104	94.0740	0.8333

The `order=matrix=` option apparently helped. The coordinate exchange algorithm is in fact chosen over the modified Fedorov algorithm.

D-efficiency at 94.46% looks good. Part of the `%MktEval` results are as follows:

Consumer Food Product Example  
 Canonical Correlations Between the Factors  
 There is 1 Canonical Correlation Greater Than 0.316

	x1	x2	x3	x4	x5	x6	x7	x8
x1	1	0.13	0.10	0.11	0.11	0.17	0.10	0.12
x2	0.13	1	0.12	0.08	0.23	0.39	0.06	0.18
x3	0.10	0.12	1	0.06	0.10	0.04	0.00	0.10
x4	0.11	0.08	0.06	1	0.07	0.07	0.06	0.18
x5	0.11	0.23	0.10	0.07	1	0.13	0.04	0.15
x6	0.17	0.39	0.04	0.07	0.13	1	0.04	0.13
x7	0.10	0.06	0.00	0.06	0.04	0.04	1	0.04
x8	0.12	0.18	0.10	0.18	0.15	0.13	0.04	1

Consumer Food Product Example  
 Canonical Correlations > 0.316 Between the Factors  
 There is 1 Canonical Correlation Greater Than 0.316

	r	r Square
x2 x6	0.39	0.15

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

Frequencies

x1	9 9 9 9
* x2	8 9 10 9
* x3	19 17
x4	18 18
* x5	11 11 14
* x6	12 13 11
* x7	17 19
* x8	11 12 13

The correlations are better, although one is still not as good as we would like. The balance looks pretty good, however it would be nice if the balance, for example, in x5 were better. It is often the case that improving balance requires some sacrifice of *D*-efficiency. We can run the macro again, this time specifying `balance=2`, which forces better balance. The specification of 2 allows the maximum frequency for a level in a factor to be no more than two greater than the minimum frequency. You should always specify `mintry=` with `balance=`. This allows %MktEx to at first increase *D*-efficiency while ignoring the balance restrictions. Then, after `mintry=m` rows have been processed, the balance restrictions are considered. Typically, you specify an expression that is a function of the number of

rows for `mintry=`, for example, `mintry=5 * n`. The `balance=` option works best when its restrictions are imposed on a reasonably efficient design not an inefficient initial design.

This example also uses a somewhat more involved `order=matrix` data set. To understand why, you need to understand how the `balance=` option works. The `%MktEx` macro uses some of the following statements to impose balance:

```

__bbad = 1;
if try > &balancetry & j1 then do;
  acol = xmat[,j1];
  acol[i,] = x[,j1];
  acol = design(acol)[+,];
  __bbad = max(0, max(acol) - min(acol) - &balance);
end;

```

It checks the balance restrictions based on the first column index, `j1`. If we are doing multiple exchanges, the exchanges in the second or subsequent columns could degrade the balance without it registering as a violation in the code above. For example, in the `order=matrix=mat` data set used previously, the last line is: `8 8 ..`. The column index `j3` might change any of the columns and yet not register in the balance-checking code, because it is only looking at column 8. For this reason, we add eight more lines so the last thing the restrictions macro does in each row is check every column for the balance constraints. The following steps create and use the order matrix:

```

data mat;
  input a b c;
  datalines;
1 1 .
2 3 4
5 5 .
6 7 .
8 8 .
1 1 1
2 2 2
3 3 3
4 4 4
5 5 5
6 6 6
7 7 7
8 8 8
;

%mktex(4 4 2 2 3 3 2 3, /* all attrs of all alternatives */
n=36, /* number of choice sets */
order=matrix=mat, /* pairs/trips of cols to work on */
interact=x2*x3 x2*x4 x3*x4 x6*x7, /* interactions */
restrictions=resmac, /* name of restrictions macro */
seed=368, /* random number seed */
outr=sasuser.EntreeLinDes3, /* randomized design stored permanently */
balance=2, /* require near but not perfect balance */
mintry=5 * n) /* ignore balance= for first 5 passes */

```

The last part of the output from the %MktEx macro is as follows:

---

Consumer Food Product Example				
The OPTEX Procedure				
Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	93.9552	87.8357	92.9627	0.8333

---

The *D*-efficiency looks good. It is a little lower than before, but not much. Next, we look at the canonical correlations and frequencies using the %MktEval macro as follows:

```
%mkteval(data=sasuser.EntreeLinDes3)
```

The first part of the output from the %MktEval macro is as follows:

---

Consumer Food Product Example								
Canonical Correlations Between the Factors								
There are 0 Canonical Correlations Greater Than 0.316								
	x1	x2	x3	x4	x5	x6	x7	x8
x1	1	0.17	0.08	0.08	0.16	0.12	0.18	0.16
x2	0.17	1	0.08	0.08	0.16	0.31	0.27	0.16
x3	0.08	0.08	1	0.11	0.12	0.07	0	0.12
x4	0.08	0.08	0.11	1	0.12	0.07	0	0.07
x5	0.16	0.16	0.12	0.12	1	0.13	0.07	0.10
x6	0.12	0.31	0.07	0.07	0.13	1	0.07	0.19
x7	0.18	0.27	0	0	0.07	0.07	1	0.12
x8	0.16	0.16	0.12	0.07	0.10	0.19	0.12	1

---

The canonical correlations look good. The last part of the output from the %MktEval macro is as follows:



This design looks much better. It is possible to get designs with better balance by specifying `balance=1`, however, since this gives %MktEx much less freedom, the `balance=1` option might cause *D*-efficiency to go down. Because `balance=1` is a tough restriction, we try this without `order=matrix` as follows:

```
%mktex(4 4 2 2 3 3 2 3,          /* all attrs of all alternatives */
        n=36,                    /* number of choice sets */
        interact=x2*x3 x2*x4 x3*x4 x6*x7, /* interactions */
        restrictions=resmac,      /* name of restrictions macro */
        seed=472,                /* random number seed */
        outr=sasuser.EntreeLinDes4, /* randomized design stored permanently */
        balance=1,               /* require near perfect balance */
        mintry=5 * n)           /* ignore balance= for first 5 passes */

%mkteval(data=sasuser.EntreeLinDes4)
```

The *D*-efficiency, which is a lower than we saw previously, is as follows:

---

Consumer Food Product Example				
The OPTEX Procedure				
Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	90.4983	79.9621	87.0176	0.8333

---

More troubling is the fact that the balance restrictions have increased the correlations between factors. The correlations are as follows:

---

Consumer Food Product Example								
Canonical Correlations Between the Factors								
There are 2 Canonical Correlations Greater Than 0.316								
	x1	x2	x3	x4	x5	x6	x7	x8
x1	1	0.22	0.11	0.11	0.19	0.33	0.11	0.30
x2	0.22	1	0.11	0.11	0.44	0.29	0.11	0
x3	0.11	0.11	1	0	0.14	0	0	0.14
x4	0.11	0.11	0	1	0.14	0	0.11	0.14
x5	0.19	0.44	0.14	0.14	1	0.14	0	0.17
x6	0.33	0.29	0	0	0.14	1	0.14	0
x7	0.11	0.11	0	0.11	0	0.14	1	0.14
x8	0.30	0	0.14	0.14	0.17	0	0.14	1

Consumer Food Product Example  
 Canonical Correlations > 0.316 Between the Factors  
 There are 2 Canonical Correlations Greater Than 0.316

		r	r Square
x2	x5	0.44	0.20
x1	x6	0.33	0.11

---

The rest of the results are as follows:

---

Consumer Food Product Example  
 Summary of Frequencies  
 There are 2 Canonical Correlations Greater Than 0.316  
 \* - Indicates Unequal Frequencies

	Frequencies
x1	9 9 9 9
x2	9 9 9 9
x3	18 18
x4	18 18
x5	12 12 12
x6	12 12 12
x7	18 18
x8	12 12 12

---

The balance is perfect. Having balance in all of the factors is nice, but for this design, we only need to ensure that **x4**, the shelf talker factor is balanced, since we are dividing the design into two parts depending on whether the shelf talker is there or not. All things considered, it looks like the design that is created with **balance=2** is the best design for our situation. It is balanced in **x4**, it is either balanced or reasonably close to balanced in the other factors, and it has good *D*-efficiency and is reasonably close to orthogonal. If our design had not been balanced in **x4**, we could have tried again with a different seed, or we could have tried again with different values for **mintry=**. If the interactions had not been requested, we also could have switched it with another two-level factor, or added it after the fact by blocking (running the **%MktBlock** macro as if we are adding a blocking factor), or we could have used the **init=** option to constrain the factor to be balanced.

The **balance=** option in the **%MktEx** macro works by adding restrictions to the design. The approach it uses often works quite well, but sometimes it does not. Forcing balance gives the macro much less freedom in its search, and makes it easy for the macro to get stuck in suboptimal designs. If perfect balance is critical and there are no interactions or restrictions, you can also try the **%MktBal** macro.



## Examining the Subdesigns

As we mentioned previously, “it is sometimes necessary and good practice to check the ranks of the subdesigns for more complex models (Lazari and Anderson 1994).” The next step shows one way to do that with PROC OPTEX. This is the only usage of PROC OPTEX in this book that is too specialized to be run from one of the %Mkt macros (because not all variables are designated as `class` variables). For convenience, we call PROC OPTEX from an ad hoc macro, since it must be run five times, once per alternative, with only a change in the `where` statement. We need to evaluate the design when the client’s alternative is available (`x1 ne 4`), when the client line extension alternative is available (`x2 ne 4`), when the regional competitor is available (`x5 ne 3`), when the private label competitor is available (`x6 ne 3`), and when the national competitor is available (`x8 ne 3`). We need to use a `model` statement that lists all of the main effects and interactions. We do not designate all of the variables in the `class` statement because we only have enough runs to consider linear price effects within each availability group. The statement `generate method=sequential initdesign=desv` specifies that we are evaluating the initial design `desv`, using the sequential algorithm, which ensures no swaps between the candidate set and the initial design. The other option of note here appears in the `class` statement, and that is `param=orthref`. This specifies an orthogonal parameterization of the effects that gives us a nice 0 to 100 scale for the *D*-efficiencies. The design is evaluated as follows:

```
%macro evaleff(where);
data desv / view=desv; set sasuser.EntreeLinDes3(where=&where)); run;

proc optex data=desv;
  class x3 x4 x7 / param=orthref;
  model x1-x8 x2*x3 x2*x4 x3*x4 x6*x7;
  generate method=sequential initdesign=desv;
run; quit;

%mkteval(data=desv)
%mend;

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

Each step took just over two seconds. We hope to not see any efficiencies of zero, and we hope to not get the message `WARNING: Can't estimate model parameters in the final design`. Some of the results are as follows:

---

 Consumer Food Product Example

## The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	69.7007	61.6709	80.8872	0.7071

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	72.7841	64.9939	87.5576	0.6939

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	66.1876	50.8651	81.2554	0.7518

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	71.8655	59.8208	86.6281	0.7518

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	65.2313	50.1059	84.1610	0.7518

---

## Examining the Aliasing Structure

It is also good to look at the aliasing structure of the design. We use PROC GLM to do this, so we must create a dependent variable. We use a constant  $y=1$ . The following PROC GLM step just checks the model to make sure none of the specified effects are aliased with each other:

```
data temp;
  set sasuser.EntreeLinDes3;
  y = 1;
  run;

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

This step is not necessary since our  $D$ -efficiency value greater than zero already guarantees this. The results, ignoring the ANOVA and regression tables, which are not of interest, are as follows:

---

```
Intercept
x1
x2
x3
x4
x5
x6
x7
x8
x2*x3
x2*x4
x3*x4
x6*x7
```

---

Each of these lines is a linear combination that is estimable. It is simply a list of the effects. Contrast this with a specification that includes all simple effects and two-way and three-way interactions. We specify the model of interest first,  $x1-x8\ x2*x3\ x2*x4\ x3*x4\ x6*x7$ , so all of those terms are listed first, then we specify all main effects and two-way and three-way interactions using the notation  $x1\ | \ x2\ | \ x3\ | \ x4\ | \ x5\ | \ x6\ | \ x7\ | \ x8@3$ . It is not a problem that some of the terms are both explicitly specified and also generated by the  $x1\ | \ x2\ | \ x3\ | \ x4\ | \ x5\ | \ x6\ | \ x7\ | \ x8@3$  list since PROC GLM automatically eliminates duplicate terms. The following step displays this more complicated aliasing structure:

```
proc glm data=temp;
  model y = x1-x8 x2*x3 x2*x4 x3*x4 x6*x7
          x1|x2|x3|x4|x5|x6|x7|x8@3 / e aliasing;
  run; quit;
```

The results are as follows:

---


$$\begin{aligned}
& \text{Intercept} - 20.008*x4*x6 - 9.8483*x1*x4*x6 - 42.279*x2*x4*x6 - 9.0597*x3*x4*x6 + \\
& 57.417*x5*x6 + 151.23*x1*x5*x6 + 186.61*x2*x5*x6 + 80.158*x3*x5*x6 + \\
& 90.545*x4*x5*x6 - 50.89*x1*x7 + 4.2117*x2*x7 - 159.53*x1*x2*x7 + 12.566*x3*x7 - \\
& 52.475*x1*x3*x7 + 43.269*x2*x3*x7 + 0.3801*x4*x7 - 71.5*x1*x4*x7 + \\
& 36.725*x2*x4*x7 + 24.297*x3*x4*x7 + 21.563*x5*x7 - 27.16*x1*x5*x7 + \\
& 75.528*x2*x5*x7 + 62.984*x3*x5*x7 + 39.224*x4*x5*x7 - 85.333*x1*x6*x7 - \\
& 10.566*x2*x6*x7 + 15.818*x3*x6*x7 - 31.415*x4*x6*x7 + 123.51*x5*x6*x7 - \\
& 24.144*x1*x8 + 6.6197*x2*x8 - 12.153*x1*x2*x8 - 38.1*x3*x8 - 133.06*x1*x3*x8 - \\
& 135.02*x2*x3*x8 + 39.148*x4*x8 + 101.08*x1*x4*x8 + 149.27*x2*x4*x8 - \\
& 15.467*x3*x4*x8 - 30.981*x5*x8 - 157.71*x1*x5*x8 - 130.69*x2*x5*x8 - \\
& 107.69*x3*x5*x8 + 19.478*x4*x5*x8 - 40.116*x6*x8 - 116.84*x1*x6*x8 - \\
& 61.852*x2*x6*x8 - 97.721*x3*x6*x8 - 23.772*x4*x6*x8 + 44.985*x5*x6*x8 - \\
& 5.0186*x7*x8 - 171.5*x1*x7*x8 + 12.071*x2*x7*x8 - 2.9687*x3*x7*x8 + \\
& 44.468*x4*x7*x8 + 8.5765*x5*x7*x8 - 52.648*x6*x7*x8 \\
& x1 + 9.1371*x4*x6 + 7.8312*x1*x4*x6 + 17.618*x2*x4*x6 + 9.7563*x3*x4*x6 - \\
& 21.745*x5*x6 - 69.803*x1*x5*x6 - 73.705*x2*x5*x6 - 39.359*x3*x5*x6 - \\
& 25.304*x4*x5*x6 + 22.962*x1*x7 - 2.9296*x2*x7 + 71.792*x1*x2*x7 - 4.9586*x3*x7 + \\
& 26.888*x1*x3*x7 - 12.562*x2*x3*x7 - 7.8969*x4*x7 + 11.379*x1*x4*x7 - \\
& 35.377*x2*x4*x7 - 21.468*x3*x4*x7 - 12.723*x5*x7 + 10.604*x1*x5*x7 - \\
& 43.808*x2*x5*x7 - 32.655*x3*x5*x7 - 32.497*x4*x5*x7 + 31.754*x1*x6*x7 + \\
& 6.8554*x2*x6*x7 - 4.0467*x3*x6*x7 + 1.6149*x4*x6*x7 - 46.784*x5*x6*x7 - \\
& 1.133*x1*x8 + 7.3858*x2*x8 + 2.0538*x1*x2*x8 + 4.336*x3*x8 + 3.3233*x1*x3*x8 + \\
& 39.854*x2*x3*x8 - 5.3094*x4*x8 - 28.994*x1*x4*x8 - 5.5582*x2*x4*x8 + \\
& 7.6916*x3*x4*x8 + 6.3495*x5*x8 + 15.979*x1*x5*x8 + 58.815*x2*x5*x8 + \\
& 16.519*x3*x5*x8 + 11.175*x4*x5*x8 + 7.3054*x6*x8 + 13.278*x1*x6*x8 + \\
& 29.443*x2*x6*x8 + 14.09*x3*x6*x8 + 18.767*x4*x6*x8 - 34.202*x5*x6*x8 + \\
& 5.8152*x7*x8 + 65.231*x1*x7*x8 + 14.788*x2*x7*x8 - 3.885*x3*x7*x8 - \\
& 15.536*x4*x7*x8 - 6.816*x5*x7*x8 + 18.202*x6*x7*x8 \\
& \cdot \\
& \cdot \\
& \cdot
\end{aligned}$$


---

Again, we have a list of linear combinations that are estimable. This shows that the **Intercept** cannot be estimated independently of the  $x4*x6$  interaction and a bunch of others including four-way though eight-way interactions which are not specified and hence not shown. Similarly,  $x1$  is confounded with a bunch of interactions, and so on. This is why we want to be estimable the two-way interactions between factors that are combined to create an alternative. We did not want something like  $x2*x3$ , the client-line extension's price and microwave/stove top interaction to be confounded with say another brand's price.

## Blocking the Design

At 36 choice sets, this design is a bit large, so we can block it into two blocks of 18 choice sets. Within each block we want the shelf talker to be on half the time. The following step blocks the design:

```
%mktblock(data=sasuser.EntreeLinDes3, out=sasuser.EntreeLinDes,
           nblocks=2, seed=448)
```

The first attempt (not shown) produced a design where x4, the shelf talker did not occur equally often within each block. Changing the seed took care of the problem. The canonical correlations are as follows:

---

Consumer Food Product Example									
Canonical Correlations Between the Factors									
There are 0 Canonical Correlations Greater Than 0.316									
	Block	x1	x2	x3	x4	x5	x6	x7	x8
Block	1	0.08	0.08	0	0	0.07	0.07	0	0.07
x1	0.08	1	0.17	0.08	0.08	0.16	0.12	0.18	0.16
x2	0.08	0.17	1	0.08	0.08	0.16	0.31	0.27	0.16
x3	0	0.08	0.08	1	0.11	0.12	0.07	0	0.12
x4	0	0.08	0.08	0.11	1	0.12	0.07	0	0.07
x5	0.07	0.16	0.16	0.12	0.12	1	0.13	0.07	0.10
x6	0.07	0.12	0.31	0.07	0.07	0.13	1	0.07	0.19
x7	0	0.18	0.27	0	0	0.07	0.07	1	0.12
x8	0.07	0.16	0.16	0.12	0.07	0.10	0.19	0.12	1

---

The blocking variable is not highly correlated with any of the factors. Some of the frequencies are as follows:

---

Consumer Food Product Example		
Summary of Frequencies		
There are 0 Canonical Correlations Greater Than 0.316		
* - Indicates Unequal Frequencies		
Frequencies		
	Block	18 18
*	x1	9 8 9 10
*	x2	8 9 10 9
	x3	18 18
	x4	18 18
*	x5	12 11 13
*	x6	11 12 13
	x7	18 18
*	x8	11 12 13

```

*   Block x1   5 5 4 4 4 4 5 5
*   Block x2   4 5 4 5 4 4 5 5
*   Block x3   8 10 9 9
      Block x4   9 9 9 9
*   Block x1   5 4 4 5 4 4 5 5
*   Block x2   4 4 5 5 4 5 5 4
      Block x3   9 9 9 9
      Block x4   9 9 9 9
*   Block x5   6 6 6 6 5 7
*   Block x6   5 6 7 6 6 6
      Block x7   9 9 9 9
*   Block x8   5 6 7 6 6 6
      .
      .
      .
    
```

The blocking variable is perfectly balanced, as it is guaranteed to be if the number of blocks divides the number of runs. Balance within blocks, that is the cross-tabulations of the factors with the blocking variable, looks good. The macro also displays canonical correlations within blocking variables. These can sometimes be quite high, even 1.0, but that is *not* a problem.\* The design, as it is displayed by the %MktBlock macro, is as follows:

Consumer Food Product Example

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

\*Ideally, each subject only makes one choice, since the choice model is based on this assumption (which is almost always ignored). As the number of blocks increases, the correlations mostly go to one, and ultimately are undefined when there is only one choice set per block.

## Consumer Food Product Example

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

## The Final Design

The next steps create the final choice design, stored in `sasuser.EntreeChDes`, sorted by the blocking and shelf talker variable. We use the `%MktLab` macro to assign values, formats, and labels to the design. Previously, we have used the `%MktLab` macro to reassign factor names when we wanted something more descriptive than the default, `x1`, `x2`, and so on, and when we wanted to reassign the names of two  $m$ -level factors to minimize the problems associated with correlated factors. This time, we use the `%MktLab` macro primarily to deal with the asymmetry in the price factors. Recall our factor levels which are as follows:

## Factors and Levels

Alternative	Factor	Levels	Brand	Description
1	X1	4	Client	1.29, 1.69, 2.09, absent
2	X2	4	Client Line Extension	1.39, 1.89, 2.39, absent microwave/stove-top shelf talker yes/no
	X3	2		
	X4	2		
3	X5	3	Regional	1.99, 2.49, absent
4	X6	3	Private Label	1.49, 2.29, absent microwave/stove-top
	X7	2		
5	X8	3	National	1.99 + 2.39, absent

The choice design needs a quantitative price attribute, made from all five of the linear price factors, that contains the prices of each of the alternatives. At this point, our factor `x1` contains 1, 2, 3, 4, and not 1.29, 1.69, 2.09, and absent, which is different from `x2` and from all of the other factors. A 1 in `x1` needs to become a price of 1.29 in the choice design, a 1 in `x2` needs to become a price of 1.39 in the choice design, a 1 in `x3` needs to become a price of 1.99 in the choice design, and so on. Before we use the `%MktRoll` macro to turn the linear arrangement into a choice design, we need to use the `%MktLab` macro to assign the actual prices to the price factors.

The `%MktLab` macro is like the `%MktRoll` macro in the sense that it can use as input a `key=` data set that contains the rules for customizing a design for our particular usage. In the `%MktRoll` macro, the `key=` data set provides the rules for turning a linear arrangement into a choice design. In contrast, in the `%MktLab` macro, the `key=` data set contains the rules for turning a linear arrangement into another linear arrangement, changing one or more of the following: factor names, factor levels, factor types (numeric to character), level formats, and factor labels.

We could use the `%MktLab` macro to change the names of the variables and their types, but we will not do that for this example. Ultimately, we use the `%MktRoll` macro to assign all of the price factors to a variable called `Price` and similarly provide meaningful names for all of the factors in the choice design, just as we have in previous examples. We could also change a variable like `x3` with values of 1 and 2 to something like `Stove` with values 'Stove' and 'Micro'. We will not do that because we want to make a design with a simple list of numeric factors, with simple names like `x1-x8` that we can run through the `%MktRoll` macro to get the final choice design. We assign formats and labels, so we can display the design in a meaningful way, but ultimately, our only goal at this step is to handle the price asymmetries by assigning the actual price values to the factors.

The `key=` data set contains the rules for customizing our design. The data set has as many rows as the maximum number of levels, in this case four. Each variable is one of the factors in the design, and the values are the factor levels that we want in the final design. The first factor, `x1`, is the price factor for the client brand. Its levels are 1.29, 1.69, and 2.09. In addition, one level is 'not available', which is flagged by the SAS special missing value `.N`. In order to read special missing values in an input data set, you must use the `missing` statement and name the expected missing values. The factor `x2` has the same structure as `x1`, but with different levels. The factor `x3` has two levels, hence the `key=` data set has missing values in the third and fourth row. Since the design has only 1's and 2's for `x3`, the missing values are never used. Notice that we are keeping `x3` as a numeric variable with values 1 and 2 using a format to supply the character levels 'micro' and 'stove'. The other factors are created in a similar fashion. By default, ordinary missing values `.'` are not permitted as levels. By default, you can only use ordinary missing values as place holders for factors that have fewer levels than the maximum. If you want missing values in the levels, you must use one of the special missing values `.A`, `.B`, ..., `.Z`, and `..*` or the `cfill=` or `nfill=` options.

The `%MktLab` macro specification names the input SAS data set with the design and the key data set. By default, it creates an output SAS data set called `Final`. The data set is sorted by block and shelf talker and displayed in the following steps:

---

\*Note that the `.'` in `.'N'` is not typed in the data, nor is it typed in the `missing` statement. Furthermore, it does not appear in the displayed output. However, you need to type it if you ever refer to a special missing value in code: `if x1 eq .N then ....`



```

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

data key;
  missing N;
  input x1-x8;
  format x1 x2 x5 x6 x8 dollar5.2
         x4 yn. x3 x7 micro.;

  label x1 = 'Client Brand'
        x2 = 'Client Line Extension'
        x3 = 'Client Micro/Stove'
        x4 = 'Shelf Talker'

        x5 = 'Regional Brand'
        x6 = 'Private Label'
        x7 = 'Private Micro/Stove'
        x8 = 'National Competitor';

  datalines;
1.29 1.39 1 1 1.99 1.49 1 1.99
1.69 1.89 2 2 2.49 2.29 2 2.39
2.09 2.39 . . N      N      .      N
N      N      . . .      .      .      .
;

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

proc sort out=sasuser.EntreeLinDesLab(drop=run); by block x4; run;

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

```

The %MktLab macro displays the variable mapping that it uses, old names followed by new names, as follows:

```

Variable Mapping:
  x1 : x1
  x2 : x2
  x3 : x3
  x4 : x4
  x5 : x5
  x6 : x6
  x7 : x7
  x8 : x8

```

In this case, none of the names change, but it is good to make sure that you have the expected correspondence.

The design is as follows:

---

Consumer Food Product Example								
Block	Shelf Talker	Client Brand	Client Line Extension	Client Micro/Stove	Regional Brand	Private Label	Private Micro/Stove	National Competitor
1	No	\$1.29	\$2.39	Micro	\$1.99	N	Micro	\$1.99
		\$1.29	\$1.89	Stove	N	N	Stove	N
		N	\$2.39	Micro	N	\$2.29	Stove	N
		\$1.29	\$1.39	Stove	\$2.49	\$2.29	Stove	N
		\$2.09	\$1.39	Micro	\$1.99	\$1.49	Stove	N
		N	N	Stove	N	\$2.29	Micro	\$1.99
		\$1.29	N	Micro	\$2.49	\$1.49	Micro	\$2.39
		\$1.69	\$1.89	Micro	\$2.49	N	Micro	\$1.99
\$2.09	N	Stove	\$1.99	\$2.29	Stove	\$2.39		
1	Talker	\$2.09	\$1.39	Stove	\$1.99	N	Stove	\$1.99
		\$1.69	N	Stove	N	\$1.49	Micro	N
		N	\$2.39	Micro	\$2.49	\$2.29	Micro	\$1.99
		\$1.69	\$2.39	Micro	\$1.99	\$1.49	Micro	N
		N	\$1.89	Micro	\$1.99	N	Stove	N
		N	\$2.39	Stove	N	N	Micro	\$2.39
		\$1.29	N	Stove	\$2.49	\$1.49	Stove	\$2.39
		\$1.69	\$1.39	Micro	N	\$2.29	Stove	\$2.39
\$2.09	\$1.89	Stove	\$2.49	N	Micro	\$2.39		
2	No	N	\$1.89	Stove	\$1.99	\$1.49	Stove	\$2.39
		N	\$1.39	Micro	\$2.49	N	Stove	\$1.99
		\$1.69	N	Micro	N	\$1.49	Stove	N
		\$1.69	\$1.39	Stove	N	N	Micro	\$2.39
		\$2.09	\$1.89	Micro	N	\$2.29	Micro	\$2.39
		\$2.09	\$1.89	Stove	N	\$1.49	Micro	\$1.99
		N	\$2.39	Stove	\$1.99	\$1.49	Micro	\$2.39
		\$2.09	\$2.39	Stove	\$2.49	\$2.29	Micro	N
\$1.69	\$2.39	Stove	\$2.49	N	Stove	\$1.99		
2	Talker	\$1.29	\$2.39	Stove	N	\$2.29	Stove	\$1.99
		\$1.29	\$1.89	Micro	N	\$1.49	Stove	\$1.99
		\$2.09	N	Micro	\$2.49	N	Micro	N
		\$1.69	\$1.89	Stove	\$1.99	\$2.29	Micro	N
		\$1.29	N	Micro	\$1.99	\$2.29	Micro	\$2.39
		\$1.29	\$1.39	Micro	\$1.99	N	Micro	N
		N	N	Micro	\$1.99	\$2.29	Stove	\$1.99
		N	\$1.39	Stove	\$2.49	\$1.49	Stove	N
\$2.09	\$2.39	Micro	N	N	Stove	\$2.39		

---

In contrast, the actual values without formats and labels are displayed by the following step:

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

The results are as follows:

---

Consumer Food Product Example									
Obs	x1	x2	x3	x4	x5	x6	x7	x8	Block
1	1.29	2.39	1	1	1.99	N	1	1.99	1
2	1.29	1.89	2	1	N	N	2	N	1
3	N	2.39	1	1	N	2.29	2	N	1
4	1.29	1.39	2	1	2.49	2.29	2	N	1
5	2.09	1.39	1	1	1.99	1.49	2	N	1
6	N	N	2	1	N	2.29	1	1.99	1
7	1.29	N	1	1	2.49	1.49	1	2.39	1
8	1.69	1.89	1	1	2.49	N	1	1.99	1
9	2.09	N	2	1	1.99	2.29	2	2.39	1
10	2.09	1.39	2	2	1.99	N	2	1.99	1
11	1.69	N	2	2	N	1.49	1	N	1
12	N	2.39	1	2	2.49	2.29	1	1.99	1
13	1.69	2.39	1	2	1.99	1.49	1	N	1
14	N	1.89	1	2	1.99	N	2	N	1
15	N	2.39	2	2	N	N	1	2.39	1
16	1.29	N	2	2	2.49	1.49	2	2.39	1
17	1.69	1.39	1	2	N	2.29	2	2.39	1
18	2.09	1.89	2	2	2.49	N	1	2.39	1
19	N	1.89	2	1	1.99	1.49	2	2.39	2
20	N	1.39	1	1	2.49	N	2	1.99	2
21	1.69	N	1	1	N	1.49	2	N	2
22	1.69	1.39	2	1	N	N	1	2.39	2
23	2.09	1.89	1	1	N	2.29	1	2.39	2
24	2.09	1.89	2	1	N	1.49	1	1.99	2
25	N	2.39	2	1	1.99	1.49	1	2.39	2
26	2.09	2.39	2	1	2.49	2.29	1	N	2
27	1.69	2.39	2	1	2.49	N	2	1.99	2
28	1.29	2.39	2	2	N	2.29	2	1.99	2
29	1.29	1.89	1	2	N	1.49	2	1.99	2
30	2.09	N	1	2	2.49	N	1	N	2
31	1.69	1.89	2	2	1.99	2.29	1	N	2
32	1.29	N	1	2	1.99	2.29	1	2.39	2
33	1.29	1.39	1	2	1.99	N	1	N	2
34	N	N	1	2	1.99	2.29	2	1.99	2
35	N	1.39	2	2	2.49	1.49	2	N	2
36	2.09	2.39	1	2	N	N	2	2.39	2

---

One issue remains to be resolved regarding this design, and that concerns the role of the shelf talker when the client line extension is not available. The second part of each block of the design consists of choice sets in which the shelf talker is present and calls attention to the client line extension. However, in some of those choice sets, the client line extension is unavailable. This problem can be handled in several ways including the following:

- 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 makes inestimable the interaction between client-line-extension price and shelf talker. Furthermore, the shelf talker variable will almost certainly no longer be balanced.
- Move the choice sets with client line extension unavailable to the no-shelf talker block and rerandomize. The shelf talker is then on for all of the last nine choice sets.
- Let the shelf talker go on and off as needed.
- Let the shelf talker call attention to a brand that happens to be out of stock. It is easy to imagine this happening in a real store.

Other options are available as well. No one approach is obviously superior to the alternatives. For this example, we take the latter approach and let the shelf talker be on even when the client line extension is not available. Note that if the shelf talker is turned off when the client line extension is not available then the design must be manually modified to reflect this fact.

## Testing the Design Before Data Collection

This is a complicated design that is used to fit a complicated model with alternative-specific effects, price cross-effects, and availability cross-effects. Collecting data is time consuming and expensive. It is always good practice, and particularly when there are cross-effects, to make sure that the design works with the most complicated model that we anticipate fitting. Before we collect any data, we convert the linear arrangement to a choice design\* and use the `%ChoiceEff` macro to evaluate its efficiency for a multinomial logit model with both price and availability cross-effects.

For analysis, the design has four factors, `Brand`, `Price`, `Micro`, `Shelf`. We use the `%MktRoll` macro and a `key=` data set (although not the same one as before) to make the choice design. `Brand` is the alternative name; its values are directly read from the `key=Key` in-stream data. `Price` is an attribute whose values are constructed from the factors `x1`, `x2`, `x5`, `x6`, and `x8` in `sasuser.EntreeLinDesLab` data set. `Micro`, the microwave attribute, is constructed from `x3` for the client line extension and `x7` for the private label. `Shelf`, the shelf talker attribute, is created from `x4` for the extension. The `keep=` option in the `%MktRoll` macro is used to keep the original price factors in the design, since we need them for the price cross-effects. Normally, they are dropped. The following steps process the design and display a subset of the results:

---

\*See page 67 for an explanation of the linear arrangement of a choice design versus the arrangement of a choice design that is more suitable for analysis.

```

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

%mktroll(design=sasuser.EntreeLinDesLab, key=key, alt=brand, out=rolled,
         keep=x1 x2 x5 x6 x8)

proc print data=sasuser.EntreeLinDesLab(obs=2); run;

proc print data=rolled(obs=12);
  format price dollar5.2 shelf yn. micro micro.;
  id set; by set;
run;

```

Consider the first two choice sets in the linear arrangement. They are as follows:

---

Consumer Food Product Example									
Obs	x1	x2	x3	x4	x5	x6	x7	x8	Block
1	\$1.29	\$2.39	Micro	No	\$1.99	N	Micro	\$1.99	1
2	\$1.29	\$1.89	Stove	No	N	N	Stove	N	1

---

When rolled out as a choice design, they are as follows:

---

Consumer Food Product Example									
Set	Brand	Price	Micro	Shelf	x1	x2	x5	x6	x8
1	Client	\$1.29	.	.	\$1.29	\$2.39	\$1.99	N	\$1.99
	Extension	\$2.39	Micro	No	\$1.29	\$2.39	\$1.99	N	\$1.99
	Regional	\$1.99	.	.	\$1.29	\$2.39	\$1.99	N	\$1.99
	Private	N	Micro	.	\$1.29	\$2.39	\$1.99	N	\$1.99
	National	\$1.99	.	.	\$1.29	\$2.39	\$1.99	N	\$1.99
	None	.	.	.	\$1.29	\$2.39	\$1.99	N	\$1.99

2	Client	\$1.29	.	.	\$1.29	\$1.89	N	N	N
	Extension	\$1.89	Stove	No	\$1.29	\$1.89	N	N	N
	Regional	N	.	.	\$1.29	\$1.89	N	N	N
	Private	N	Stove	.	\$1.29	\$1.89	N	N	N
	National	N	.	.	\$1.29	\$1.89	N	N	N
	None	.	.	.	\$1.29	\$1.89	N	N	N

## Set 1, Alternative 1

Brand	=	'Client'	the brand for this alternative
Price	=	x1 = \$1.29	the price of this alternative
Micro	=	.	does not apply to this brand
Shelf	=	.	does not apply to this brand
x1	=	\$1.29	the price of the client brand in this choice set
x2	=	\$2.39	the price of the extension in this choice set
x5	=	\$1.99	the price of the regional competitor in this choice set
x6	=	N	the private label unavailable in this choice set
x8	=	\$1.99	national competitor unavailable in this choice set

## Set 1, Alternative 2

Brand	=	'Extension'	the brand for this alternative
Price	=	x2 = \$2.39	the price of this alternative
Micro	=	Micro	Microwave version
Shelf	=	No	Shelf Talker, No
x1	=	\$1.29	the price of the client brand in this choice set
x2	=	\$2.39	the price of the extension in this choice set
x5	=	\$1.99	the price of the regional competitor in this choice set
x6	=	N	the private label unavailable in this choice set
x8	=	\$1.99	national competitor unavailable in this choice set

The factors x1 through x8 are used to make the price cross-effects. Notice that x1 through x8 are constant within each choice set. The variable x1 is the price of alternative one, which is the same no matter which alternative it is stored with. The factors x1 through x8 are also used to make five other factors that are used to make the availability cross-effects. The following table shows how the prices are recoded for those factors:

x1	→	a1	x2	→	a2	x5	→	a5	x6	→	a6	x8	→	a8
1.29		1	1.39		1	1.99		1	1.49		1	1.99		1
1.69		1	1.89		1	2.49		1	2.29		1	2.39		1
2.09		1	2.39		1	N		-2	N		-2	N		-2
N		-3	N		-3									

This is a contrast coding. Within each factor, the coding sums to zero. Each availability factor has a coding that contrasts unavailable with the remaining available prices. When an alternative is unavailable, the a variable is set to minus the number of available price points. The coding for available alternatives is 1. A -3 is used for the first two alternatives that have three prices, and a -2 is used for the remaining alternatives that have two prices.

We need to do a few more things to this design before we are ready to use it. We need to convert the missings for when `Micro` and `Shelf` do not apply to 2 for 'Stove' and 1 for 'No'. We need to do the contrast coding for making the availability cross-effects. More is said about this after the code is shown. Since we are treating all of the price factors as a quantitative (not as `class` variables), we need to convert the missing prices to zero. Eventually, we also need to output just the alternatives that are available (those with a nonzero price and also the none alternative). For now, we just make a variable `w` that flags the available alternatives (`w = 1`). We can do this using a weight or flag variable: `w = 1` means available and `w = 0` means not available. We also need to assign labels and formats. The following `DATA` step does the processing:

```
data sasuser.EntreeChDes(drop=i);
  set rolled;
  array x[6] price x1 -- x8;
  array a[5] a1 a2 a5 a6 a8;
  if nmiss(micro) then micro = 2; /* stove if not a factor in alt      */
  if nmiss(shelf) then shelf = 1; /* not talker if not a factor in alt */

  a1 = -3 * nmiss(x1) + n(x1);    /* alt1: -3 - not avail, 1 - avail */
  a2 = -3 * nmiss(x2) + n(x2);    /* alt2: -3 - not avail, 1 - avail */
  a5 = -2 * nmiss(x5) + n(x5);    /* alt3: -2 - not avail, 1 - avail */
  a6 = -2 * nmiss(x6) + n(x6);    /* alt4: -2 - not avail, 1 - avail */
  a8 = -2 * nmiss(x8) + n(x8);    /* alt5: -2 - not avail, 1 - avail */
  i = mod(_n_ - 1, 6) + 1;        /* alternative number              */
  if i le 5 then a[i] = 0;        /* 0 effect of an alt on itself   */

  do i = 1 to 6; if nmiss(x[i]) then x[i] = 0; end; /* missing price -> 0      */
  w = brand eq 'None' or price ne 0;                /* 1 - avail, 0 not avail*/
  format price dollar5.2 shelf yn. micro micro.;
  label x1 = 'CE, Client'      a1 = 'AE, Client'
        x2 = 'CE, Extension'   a2 = 'AE, Extension'
        x5 = 'CE, Regional'    a5 = 'AE, Regional'
        x6 = 'CE, Private'     a6 = 'AE, Private'
        x8 = 'CE, National'    a8 = 'AE, National';
run;

proc print data=sasuser.EntreeChDes(obs=18); by set; id set; run;
```

The statements in the middle of the `DATA` step, from `a1 = ...` through `if i ...` create the variables that are used to make the availability effects. When alternative 1 is unavailable (`x1` is missing), `a1` is set to `-3`, otherwise `a1` is set to `1`; when alternative 2 is unavailable (`x2` is missing), `a2` is set to `-3`, otherwise `a2` is set to `1`; when alternative 3 is unavailable (`x5` is missing), `a5` is set to `-3`, otherwise `a5` is set to `1`; and so on. Each of these statements could have been written in `if else` form. Here, for example, is the first assignment statement rewritten: `if nmiss(x1) then a1 = -3; else a1 = 1;`. The variables `x1`, `x2`, `x5`, `x6`, and `x8` are the five price factors, and the “a” factors use the same numbering scheme, although this is not a requirement. The `if i` and `i =` statements then set the variable to zero when the variable is used to construct the effect of an alternative on itself. For example, the first alternative is the client brand, so `a1` in the first alternative is set to zero. The first three choice sets are as follows:

## Consumer Food Product Example

Set	Brand	Price	Micro	Shelf	x1	x2	x5	x6	x8	a1	a2	a5	a6	a8	w
1	Client	\$1.29	Stove	No	\$1.29	\$2.39	\$1.99	\$0.00	\$1.99	0	1	1	-2	1	1
	Extension	\$2.39	Micro	No	\$1.29	\$2.39	\$1.99	\$0.00	\$1.99	1	0	1	-2	1	1
	Regional	\$1.99	Stove	No	\$1.29	\$2.39	\$1.99	\$0.00	\$1.99	1	1	0	-2	1	1
	Private	\$0.00	Micro	No	\$1.29	\$2.39	\$1.99	\$0.00	\$1.99	1	1	1	0	1	0
	National	\$1.99	Stove	No	\$1.29	\$2.39	\$1.99	\$0.00	\$1.99	1	1	1	-2	0	1
	None	\$0.00	Stove	No	\$1.29	\$2.39	\$1.99	\$0.00	\$1.99	1	1	1	-2	1	1
2	Client	\$1.29	Stove	No	\$1.29	\$1.89	\$0.00	\$0.00	\$0.00	0	1	-2	-2	-2	1
	Extension	\$1.89	Stove	No	\$1.29	\$1.89	\$0.00	\$0.00	\$0.00	1	0	-2	-2	-2	1
	Regional	\$0.00	Stove	No	\$1.29	\$1.89	\$0.00	\$0.00	\$0.00	1	1	0	-2	-2	0
	Private	\$0.00	Stove	No	\$1.29	\$1.89	\$0.00	\$0.00	\$0.00	1	1	-2	0	-2	0
	National	\$0.00	Stove	No	\$1.29	\$1.89	\$0.00	\$0.00	\$0.00	1	1	-2	-2	0	0
	None	\$0.00	Stove	No	\$1.29	\$1.89	\$0.00	\$0.00	\$0.00	1	1	-2	-2	-2	1
3	Client	\$0.00	Stove	No	\$0.00	\$2.39	\$0.00	\$2.29	\$0.00	0	1	-2	1	-2	0
	Extension	\$2.39	Micro	No	\$0.00	\$2.39	\$0.00	\$2.29	\$0.00	-3	0	-2	1	-2	1
	Regional	\$0.00	Stove	No	\$0.00	\$2.39	\$0.00	\$2.29	\$0.00	-3	1	0	1	-2	0
	Private	\$2.29	Stove	No	\$0.00	\$2.39	\$0.00	\$2.29	\$0.00	-3	1	-2	0	-2	1
	National	\$0.00	Stove	No	\$0.00	\$2.39	\$0.00	\$2.29	\$0.00	-3	1	-2	1	0	0
	None	\$0.00	Stove	No	\$0.00	\$2.39	\$0.00	\$2.29	\$0.00	-3	1	-2	1	-2	1

In the first choice set, for example, since alternative 1 is available,  $a_1$  is 1, for all alternatives except the first, where  $a_1$  is 0. Also in the first choice set, since alternative 4 is not available and there are two price levels,  $a_6$  is  $-2$  for all alternatives except the fourth, where  $a_6$  is 0. In the third choice set, since alternative 1 is not available and there are three price levels,  $a_1$  is  $-3$ , for all alternatives except the first, where  $a_2$  is 0. In general, the coding stores a zero in the  $i$ th effect for the  $i$ th alternative, otherwise a 1 if the alternative is available, otherwise minus the number of price levels if the alternative is unavailable.

Now our choice design is done except for the final coding for the analysis. We can now use the `%ChoiceEff` macro to evaluate our choice design. The complicated part of this is the model due to the alternative-specific price effects and the cross-effects. For now, let's concentrate on everything else. The following step provides some sample code, omitting for now the details of the model (indicated by `model= ...`):

```
%choiceff(data=sasuser.EntreeChDes, /* candidate set of choice sets      */
  init=sasuser.EntreeChDes(keep=set), /* select these sets              */
  intiter=0,                          /* evaluate without internal iterations */
  model= ...,                          /* model specification skipped for now */
  nsets=36,                            /* number of choice sets              */
  nalts=6,                             /* number of alternatives              */
  weight=w,                            /* weight to handle unavailable alts   */
  beta=zero)                          /* assumed beta vector, Ho: b=0       */
```



The way you check the efficiency of a design like this is to first name it in the `data=` option. This is the candidate set that contains all of the choice sets that we will consider. In addition, the same design is named in the `init=` option. The full specification is `init=sasuser.EntreeChDes(keep=set)`. Just the variable `Set` is kept. It is used to bring in just the indicated choice sets from the `data=` design, which in this case is all of them. The option `nsets=36` specifies the number of choice sets, and `nalts=6` specifies the number of alternatives. This macro requires a constant number of alternatives in each choice set for ease of data management. However, not all of the alternatives have to be used. In this case, we have an availability study. We need to keep the unavailable alternatives in the design for this step, but we do not want them to contribute to the analysis, so we specify a weight variable with `weight=w` and flag the available alternatives with `w=1` and the unavailable alternatives with `w=0`. The option `beta=zero` specifies that we are assuming for design evaluation purpose all zero betas. We can specify other values and get other results for the variances and standard errors. Finally, we specify `intiter=0` which specifies zero internal iterations. We use zero internal iterations when we want to evaluate an initial design, but not attempt to improve it. The actual specification we use, complete with the model specification, is shown in the following step:

```
%choiceff(data=sasuser.EntreeChDes, /* candidate set of choice sets      */
init=sasuser.EntreeChDes(keep=set), /* select these sets              */
intiter=0,                          /* evaluate without internal iterations */
/* cross-effects model              */
/* zero='None' - 'None' level is ref lev*/
model=class(brand / zero='None')
/* use blank sep to build interact terms*/
class(brand / zero='None' separators=' ' ' ') *
identity(price)
/* lpr=5 0 uses 5 var name chars in    */
/* label for shelf and 0 for micro     */
/* 'No' is ref level for shelf         */
/* 'Stove' is ref level for micro      */
class(shelf micro / lprefix=5 0 zero='No' 'Stove')
/* ' on ' is used to build interact term*/
identity(x1 x2 x5 x6 x8) *
class(brand / zero='None' separators=' ' ' ' on ')
identity(a1 a2 a5 a6 a8) *
class(brand / zero='None' separators=' ' ' ' on ') /
lprefix=0 /* lpr=0 labels are created from just */
/* levels unless overridden above */
order=data, /* order=data - do not sort levels */

nsets=36, /* number of choice sets */
nalts=6, /* number of alternatives */
weight=w, /* weight to handle unavailable alts */
beta=zero) /* assumed beta vector, Ho: b=0 */
```

The specification `class(brand / zero='None')` specifies the brand effects. This specification creates indicator variables for brand with the constant alternative being the reference brand. The option `zero='None'` ensures that the reference level is 'None' instead of the default last sorted level ('Regional'). Indicator variables are created for the brands Client, Extension, Regional, Private, and National, but not None. The `zero='None'` option, like `zero='Home'` and other `zero='literal-string'`

options we have used in previous examples, names the actual formatted value of the `class` variable that should be excluded from the coded variables because the coefficient is zero. Do not confuse `zero=none` and `zero='None'`. The `zero=none` option specifies that you want all indicator variables to be created, even including one for the last level. In contrast, the option `zero='None'` (or `zero=` any quoted string) names a specific formatted value, in this case `'None'`, for which indicator variables are not to be created. See page 78 for more information about the `zero=` option.

The specification `class(brand / ...) * identity(price)` creates the alternative-specific price effects. They are specified as an interaction between a categorical variable `Brand` and a quantitative attribute `Price`. The `separators=' ' ' '` option in the `class` specification specifies the separators that are used to construct the labels for the main effect and interaction terms. The main-effects separator, which is the first `separators=` value, `' '`, is ignored since `lprefix=0`. Specifying `' '` as the second value creates labels of the form *brand-blank-price* instead of the default *brand-blank-asterisk-blank-price*.

The specification `class(shelf micro / ...)` names the shelf talker and microwave variables as categorical variables and creates indicator variables for the `'Talker'` category, not the `'No'` category and the `'Micro'` category not the `'Stove'` category. In `zero='No' 'Stove'`, the `'No'` applies to the first variable, `Shelf` and the second value, `'Stove'`, applies to second variable, `Micro`.

The specification `identity(x1 x2 x5 x6 x8) * class(brand / ...)` creates the linear price cross-effects. The `separators=` option is specified with a second value of `' on '` to create cross-effect labels like `'Client on Extension'`. The specification `identity(a1 a2 a5 a6 a8) * class(brand / ...)` creates the availability cross-effects. Note that the order of the transformation specification is important. Make sure you specify `identity` followed by `class` in order to get the right labels. More is said about cross-effects when we look at the actual coded values in the next few pages.

PROC TRANSREG produces the following warning:

```
WARNING: This usage of * sets one group's slope to zero. Specify |
         to allow all slopes and intercepts to vary. Alternatively,
         specify CLASS(vars) * identity(vars) identity(vars) for
         separate within group functions and a common intercept.
         This is a change from Version 6.
```

This is because `class` is 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 displays 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 display one or more coded choice sets to check the coding as we will do later. The last part of the output from the `%ChoiceEff` macro is as follows:

---

### Consumer Food Product Example

#### Final Results

Design	1
Choice Sets	36
Alternatives	6
Parameters	52
Maximum Parameters	180
D-Efficiency	0
D-Error	.

## Consumer Food Product Example

n	Variable Name	Label	Variance	DF	Standard Error
1	BrandClient	Client	69.807	1	8.3551
2	BrandExtension	Extension	75.688	1	8.6999
3	BrandRegional	Regional	121.147	1	11.0067
4	BrandPrivate	Private	104.058	1	10.2009
5	BrandNational	National	110.456	1	10.5098
6	BrandClientPrice	Client Price	3.255	1	1.8042
7	BrandExtensionPrice	Extension Price	2.233	1	1.4942
8	BrandRegionalPrice	Regional Price	6.599	1	2.5688
9	BrandPrivatePrice	Private Price	2.604	1	1.6138
10	BrandNationalPrice	National Price	11.071	1	3.3273
11	ShelfTalker	Shelf Talker	0.928	1	0.9636
12	MicroMicro	Micro	0.562	1	0.7493
13	x1BrandClient	CE, Client on Client	.	0	.
14	x1BrandExtension	CE, Client on Extension	4.689	1	2.1655
15	x1BrandRegional	CE, Client on Regional	4.462	1	2.1124
16	x1BrandPrivate	CE, Client on Private	5.627	1	2.3720
17	x1BrandNational	CE, Client on National	5.374	1	2.3182
18	x2BrandClient	CE, Extension on Client	3.040	1	1.7435
19	x2BrandExtension	CE, Extension on Extension	.	0	.
20	x2BrandRegional	CE, Extension on Regional	3.038	1	1.7431
21	x2BrandPrivate	CE, Extension on Private	3.666	1	1.9146
22	x2BrandNational	CE, Extension on National	3.130	1	1.7691
23	x5BrandClient	CE, Regional on Client	8.961	1	2.9935
24	x5BrandExtension	CE, Regional on Extension	9.824	1	3.1343
25	x5BrandRegional	CE, Regional on Regional	.	0	.
26	x5BrandPrivate	CE, Regional on Private	10.496	1	3.2398
27	x5BrandNational	CE, Regional on National	10.360	1	3.2188
28	x6BrandClient	CE, Private on Client	3.965	1	1.9912
29	x6BrandExtension	CE, Private on Extension	4.195	1	2.0482
30	x6BrandRegional	CE, Private on Regional	4.429	1	2.1046
31	x6BrandPrivate	CE, Private on Private	.	0	.
32	x6BrandNational	CE, Private on National	4.453	1	2.1102
33	x8BrandClient	CE, National on Client	18.098	1	4.2541
34	x8BrandExtension	CE, National on Extension	16.311	1	4.0387
35	x8BrandRegional	CE, National on Regional	22.372	1	4.7299
36	x8BrandPrivate	CE, National on Private	18.271	1	4.2745
37	x8BrandNational	CE, National on National	.	0	.
38	a1BrandClient	AE, Client on Client	.	0	.
39	a1BrandExtension	AE, Client on Extension	0.981	1	0.9904
40	a1BrandRegional	AE, Client on Regional	0.892	1	0.9447
41	a1BrandPrivate	AE, Client on Private	1.071	1	1.0347
42	a1BrandNational	AE, Client on National	1.031	1	1.0155

43	a2BrandClient	AE, Extension on Client	0.766	1	0.8755
44	a2BrandExtension	AE, Extension on Extension	.	0	.
45	a2BrandRegional	AE, Extension on Regional	0.880	1	0.9381
46	a2BrandPrivate	AE, Extension on Private	0.990	1	0.9952
47	a2BrandNational	AE, Extension on National	0.999	1	0.9995
48	a5BrandClient	AE, Regional on Client	5.128	1	2.2644
49	a5BrandExtension	AE, Regional on Extension	5.530	1	2.3516
50	a5BrandRegional	AE, Regional on Regional	.	0	.
51	a5BrandPrivate	AE, Regional on Private	5.860	1	2.4208
52	a5BrandNational	AE, Regional on National	5.887	1	2.4263
53	a6BrandClient	AE, Private on Client	1.796	1	1.3402
54	a6BrandExtension	AE, Private on Extension	1.843	1	1.3577
55	a6BrandRegional	AE, Private on Regional	2.116	1	1.4547
56	a6BrandPrivate	AE, Private on Private	.	0	.
57	a6BrandNational	AE, Private on National	1.964	1	1.4015
58	a8BrandClient	AE, National on Client	10.135	1	3.1836
59	a8BrandExtension	AE, National on Extension	8.720	1	2.9529
60	a8BrandRegional	AE, National on Regional	12.021	1	3.4671
61	a8BrandPrivate	AE, National on Private	10.188	1	3.1919
62	a8BrandNational	AE, National on National	.	0	.
				==	
				52	

---

First we see estimable brand effects for each of the five brands, excluding the constant alternative 'None'. Next, we see quantitative alternative-specific price effects for each of the brands. The next two effects that are single *df* effects for the shelf talker and the microwave option. Then we see five sets of linear price cross-effects (those whose label begins with "CE"), each consisting of four effects of a brand on another brand, plus one more zero *df* cross-effect of a brand on itself. The zero *df* and missing variances and standard errors are correct since the cross-effect of an alternative on itself is perfectly aliased with its alternative-specific price effect. After that we see five sets of availability cross-effects (those whose label begins with "AE"), each consisting of four effects of a brand on another brand, plus one more zero *df* cross-effect of a brand on itself. The zero *df* and missing variances and standard errors are correct since the cross-effect of an alternative on itself is zero. These results look fine. Everything that should be estimable is estimable, and everything that should not be estimable is not.

Next, we run some further checks by looking at the coded design. Before we look at the coded design, recall that the design for the first five choice sets is as follows:

---

Consumer Food Product Example								
Block	Shelf Talker	Client Brand	Client Line Extension	Client Micro/Stove	Regional Brand	Private Label	Private Micro/Stove	National Competitor
1	No	\$1.29	\$2.39	Micro	\$1.99	N	Micro	\$1.99
		\$1.29	\$1.89	Stove	N	N	Stove	N
		N	\$2.39	Micro	N	\$2.29	Stove	N
		\$1.29	\$1.39	Stove	\$2.49	\$2.29	Stove	N
		\$2.09	\$1.39	Micro	\$1.99	\$1.49	Stove	N

---

The coded design that the %ChoiceEff macro creates is called TMP\_CAND. We can look at the coded data set in several ways. First, the Brand, Price, microwave and shelf talker factors, for just the available alternatives for the first five choice sets are displayed as follows:

```
proc print data=tmp_cand(obs=24) label;
  var Brand Price Shelf Micro;
  where w;
run;
```

The results are as follows:

---

Consumer Food Product Example					
Obs	Brand	Price	Shelf	Micro	
1	Client	\$1.29	No	Stove	
2	Extension	\$2.39	No	Micro	
3	Regional	\$1.99	No	Stove	
5	National	\$1.99	No	Stove	
6	None	\$0.00	No	Stove	
7	Client	\$1.29	No	Stove	
8	Extension	\$1.89	No	Stove	
12	None	\$0.00	No	Stove	
14	Extension	\$2.39	No	Micro	
16	Private	\$2.29	No	Stove	
18	None	\$0.00	No	Stove	
19	Client	\$1.29	No	Stove	
20	Extension	\$1.39	No	Stove	
21	Regional	\$2.49	No	Stove	
22	Private	\$2.29	No	Stove	
24	None	\$0.00	No	Stove	

25	Client	\$2.09	No	Stove
26	Extension	\$1.39	No	Micro
27	Regional	\$1.99	No	Stove
28	Private	\$1.49	No	Stove
30	None	\$0.00	No	Stove
34	Private	\$2.29	No	Micro
35	National	\$1.99	No	Stove
36	None	\$0.00	No	Stove

Unlike all previous examples, the number of alternatives is not the same in all of the choice sets due to differing subsets of brands being unavailable in each choice set.

The coded factors for the brand effects and alternative-specific price effects for the first choice set are displayed as follows:

```
proc print data=tmp_cand(obs=5) label;
  id Brand;
  var BrandClient -- BrandNational;
  where w;
  run;

proc format; value zer 0 = ' 0'; run;

proc print data=tmp_cand(obs=5) label;
  id Brand Price;
  var BrandClientPrice -- BrandNationalPrice;
  format BrandClientPrice -- BrandNationalPrice zer5.2;
  where w;
  run;
```

The results are as follows:

---

#### Consumer Food Product Example

Brand	Client	Extension	Regional	Private	National
Client	1	0	0	0	0
Extension	0	1	0	0	0
Regional	0	0	1	0	0
National	0	0	0	0	1
None	0	0	0	0	0

## Consumer Food Product Example

Brand	Price	Client Price	Extension Price	Regional Price	Private Price	National Price
Client	\$1.29	1.29	0	0	0	0
Extension	\$2.39	0	2.39	0	0	0
Regional	\$1.99	0	0	1.99	0	0
National	\$1.99	0	0	0	0	1.99
None	\$0.00	0	0	0	0	0

The brand effects and alternative-specific price effect codings are similar to those we have used previously. The difference is the presence of all zero columns for unavailable alternatives, in this case the private label and national brands. Note that **Brand Price** are just ID variables and do not enter into the analysis.

The shelf talker and microwave coded factors (along with the **Brand**, **Price**, **Shelf**, and **Micro** factors) are displayed by the following step:

```
proc print data=tmp_cand(obs=5) label;
  id Brand Price Shelf Micro;
  var shelftalker micromicro;
  where w;
run;
```

The results are as follows:

## Consumer Food Product Example

Brand	Price	Shelf	Micro	Shelf Talker	Micro
Client	\$1.29	No	Stove	0	0
Extension	\$2.39	No	Micro	0	1
Regional	\$1.99	No	Stove	0	0
National	\$1.99	No	Stove	0	0
None	\$0.00	No	Stove	0	0

The following steps display the price cross-effects along with **Brand** and **Price** for the first choice set:

```
proc print data=tmp_cand(obs=4) label;
  id Brand Price;
  var x1Brand;; format x1Brand: zer5.2;
  where w;
run;
```

```

proc print data=tmp_cand(obs=4) label;
  id Brand Price;
  var x2Brand;; format x2Brand: zer5.2;
  where w;
run;

proc print data=tmp_cand(obs=4) label;
  id Brand Price;
  var x5Brand;; format x5Brand: zer5.2;
  where w;
run;

proc print data=tmp_cand(obs=4) label;
  id Brand Price;
  var x6Brand;; format x6Brand: zer5.2;
  where w;
run;

proc print data=tmp_cand(obs=4) label;
  id Brand Price;
  var x8Brand;; format x8Brand: zer5.2;
  where w;
run;

```

The cross-effects are displayed in panels. This first panel shows the terms that capture the effect of the client brand on the utility of the other brands. The second panel shows the terms that capture the effect of the line extension on the other alternatives, and so on. An unavailable brand has no effect on any other brand's utility in that choice set. The results are as follows:

---

Consumer Food Product Example

Brand	Price	CE, Client on Client	CE, Client on Extension	CE, Client on Regional	CE, Client on Private	CE, Client on National
Client	\$1.29	1.29	0	0	0	0
Extension	\$2.39	0	1.29	0	0	0
Regional	\$1.99	0	0	1.29	0	0
National	\$1.99	0	0	0	0	1.29

Brand	Price	CE, Extension on Client	CE, Extension on Extension	CE, Extension on Regional	CE, Extension on Private	CE, Extension on National
Client	\$1.29	2.39	0	0	0	0
Extension	\$2.39	0	2.39	0	0	0
Regional	\$1.99	0	0	2.39	0	0
National	\$1.99	0	0	0	0	2.39



Brand	Price	CE, Regional on Client	CE, Regional on Extension	CE, Regional on Regional	CE, Regional on Private	CE, Regional on National
Client	\$1.29	1.99	0	0	0	0
Extension	\$2.39	0	1.99	0	0	0
Regional	\$1.99	0	0	1.99	0	0
National	\$1.99	0	0	0	0	1.99

Brand	Price	CE, Private on Client	CE, Private on Extension	CE, Private on Regional	CE, Private on Private	CE, Private on National
Client	\$1.29	0	0	0	0	0
Extension	\$2.39	0	0	0	0	0
Regional	\$1.99	0	0	0	0	0
National	\$1.99	0	0	0	0	0

Brand	Price	CE, National on Client	CE, National on Extension	CE, National on Regional	CE, National on Private	CE, National on National
Client	\$1.29	1.99	0	0	0	0
Extension	\$2.39	0	1.99	0	0	0
Regional	\$1.99	0	0	1.99	0	0
National	\$1.99	0	0	0	0	1.99

A column like 'CE, Client on Extension' in the first panel, for example, captures the effect of the client brand at \$1.29 on the utility of the extension. In the next panel, 'CE, Extension on Client' captures the effect of the extension at \$2.39 on the utility of the client brand.

The following steps displays the availability cross-effects along with Brand and Price for the first choice set:

```
proc print data=tmp_cand(obs=4) label;
  id Brand Price;
  var a1Brand;;
  where w;
run;
```

```
proc print data=tmp_cand(obs=4) label;
  id Brand Price;
  var a2Brand;;
  where w;
run;
```

```

proc print data=tmp_cand(obs=4) label;
  id Brand Price;
  var a5Brand;;
  where w;
  run;

proc print data=tmp_cand(obs=4) label;
  id Brand Price;
  var a6Brand;;
  where w;
  run;

proc print data=tmp_cand(obs=4) label;
  id Brand Price;
  var a8Brand;;
  where w;
  run;

```

The availability cross-effects are displayed in panels. The first panel shows the terms that capture the effect of the client brand which on the other available alternatives, and so on. Panels with 1's in them show the effects of the available brands and panels with negative numbers show the effects of the unavailable brands. The results are as follows:

---

Consumer Food Product Example

Brand	Price	AE, Client on Client	AE, Client on Extension	AE, Client on Regional	AE, Client on Private	AE, Client on National
Client	\$1.29	0	0	0	0	0
Extension	\$2.39	0	1	0	0	0
Regional	\$1.99	0	0	1	0	0
National	\$1.99	0	0	0	0	1

Brand	Price	AE, Extension on Client	AE, Extension on Extension	AE, Extension on Regional	AE, Extension on Private	AE, Extension on National
Client	\$1.29	1	0	0	0	0
Extension	\$2.39	0	0	0	0	0
Regional	\$1.99	0	0	1	0	0
National	\$1.99	0	0	0	0	1

Brand	Price	AE, Regional on Client	AE, Regional on Extension	AE, Regional on Regional	AE, Regional on Private	AE, Regional on National
Client	\$1.29	1	0	0	0	0
Extension	\$2.39	0	1	0	0	0
Regional	\$1.99	0	0	0	0	0
National	\$1.99	0	0	0	0	1

Brand	Price	AE, Private on Client	AE, Private on Extension	AE, Private on Regional	AE, Private on Private	AE, Private on National
Client	\$1.29	-2	0	0	0	0
Extension	\$2.39	0	-2	0	0	0
Regional	\$1.99	0	0	-2	0	0
National	\$1.99	0	0	0	0	-2

Brand	Price	AE, National on Client	AE, National on Extension	AE, National on Regional	AE, National on Private	AE, National on National
Client	\$1.29	1	0	0	0	0
Extension	\$2.39	0	1	0	0	0
Regional	\$1.99	0	0	1	0	0
National	\$1.99	0	0	0	0	0

---

The design looks good, it has reasonably good balance and correlations, it can be used to estimate all of the effects of interest, and we have shown that we know how to code all of the factors for a model with cross-effects and availability cross-effects. We are ready to collect data.

There is one more test that should be run before a design is used. The %MktDups macro checks the design to see if any choice sets are duplicates of any other choice sets as follows:

```
%mktdups(branded, data=sasuser.EntreeChDes,
          nalts=6, factors=brand price shelf micro)
```

The results are as follows:

---

```
Design:          Branded
Factors:         brand price shelf micro
                  Brand
                  Micro Price Shelf
Duplicate Sets:  0
```

---

The first line of the table tells us that this is a branded design as opposed to a generic design (bundles of attributes with no brands). The second line tells us the factors as specified in the `factors=` option. These are followed by the actual variable names for the factors. The last line reports the number of duplicates. In this case, there are no duplicate choice sets. If there are duplicate choice sets, then changing the random number seed might help. Changing other aspects of the design or the approach for making the design might help as well.

## Generating Artificial Data

We do not illustrate questionnaire generation for this example since we have done it several times before in previous examples. Instead we go straight to data processing and analysis. The following DATA step generates some artificial data:<sup>†</sup>

```
%let m    = 6;
%let mm1  = %eval(&m - 1);
%let n    = 36;

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

---

<sup>†</sup>This section shows in a cursory way how to generate artificial data. Some researchers wisely like to use artificial data to test a design before spending lots of money on collecting data. See the section beginning on page 393 for a detailed discussion of generating artificial data.

```

data _null_;
  array brands[&m] _temporary_ (5 7 1 2 3 -2);
  array u[&m];
  array x[&mm1] x1 x2 x5 x6 x8;
  do rep = 1 to 300;
    if mod(rep, 2) then put;
    put rep 3. +2 @@;
    do j = 1 to &n;
      set sasuser.EntreeLinDesLab point=j;
      do brand = 1 to &m; u[brand] = brands[brand] + 2 * normal(17); end;
      do brand = 1 to &mm1;
        if n(x[brand]) then u[brand] + -x[brand]; else u[brand] = .;
      end;
      if n(u2) and x4 = 2 then u2 + 1; /* shelf talker */
      if n(u2) and x3 = 1 then u2 + 1; /* microwave */
      if n(u4) and x7 = 1 then u4 + 1; /* microwave */
      * Choose the most preferred alternative.;
      m = max(of u1-u&m);
      do brand = 1 to &m;
        if n(u[brand]) then if abs(u[brand] - m) < 1e-4 then c = brand;
      end;
      put +(-1) c @@;
    end;
  end;
stop;
run;

```

Creating artificial data and trying the analysis before collecting real data is another way to test the design before going to the expense of data collection. The following DATA step reads the data:

```

data results;
  input Subj (choose1-choose&n) (1.) @@;
  datalines;
1 222224155212222522221221222221212522 2 222225421242222122221212222221211322
3 212224521545222122221222221121112522 4 212125121212222122221222421221212522
5 222225125212222122521212222221212422 6 222125523112222122224221212111242522
.
.
.
297 222225521212222422224222522121151622 298 222224121242122422221222512221212522
299 112225121212122121521222212211212622 300 122225123242122122221211222123212322
;

```

## Processing the Data

The analysis proceeds in a fashion similar to before. We have already made the choice design, so we just have to merge it with the data. The data and design are merged in the usual way using the %MktMerge macro. Notice at this point that the unavailable alternatives are still in the design. The

%MktMerge macro has an `nalts=` option and expects a constant number of alternatives in each choice set. The following steps perform the merge and display a subset of the results:

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

proc print data=res2(obs=12); id subj set; by subj set; run;
```

The data and design for the first two choice sets for the first subject, including the unavailable alternatives, are as follows:

---

Consumer Food Product Example

B	P	M	S												
S r	r	i	h												
u S a	i	c	e												
b e n	c	r	l	x	x	x	x	x	a	a	a	a	a	a	
j t d	e	o	f	1	2	5	6	8	1	2	5	6	8	w	c
1 1 Client	\$1.29	Stove	No	\$1.29	\$2.39	\$1.99	\$0.00	\$1.99	0	1	1	-2	1	1	2
Extension	\$2.39	Micro	No	\$1.29	\$2.39	\$1.99	\$0.00	\$1.99	1	0	1	-2	1	1	1
Regional	\$1.99	Stove	No	\$1.29	\$2.39	\$1.99	\$0.00	\$1.99	1	1	0	-2	1	1	2
Private	\$0.00	Micro	No	\$1.29	\$2.39	\$1.99	\$0.00	\$1.99	1	1	1	0	1	0	2
National	\$1.99	Stove	No	\$1.29	\$2.39	\$1.99	\$0.00	\$1.99	1	1	1	-2	0	1	2
None	\$0.00	Stove	No	\$1.29	\$2.39	\$1.99	\$0.00	\$1.99	1	1	1	-2	1	1	2
1 2 Client	\$1.29	Stove	No	\$1.29	\$1.89	\$0.00	\$0.00	\$0.00	0	1	-2	-2	-2	1	2
Extension	\$1.89	Stove	No	\$1.29	\$1.89	\$0.00	\$0.00	\$0.00	1	0	-2	-2	-2	1	1
Regional	\$0.00	Stove	No	\$1.29	\$1.89	\$0.00	\$0.00	\$0.00	1	1	0	-2	-2	0	2
Private	\$0.00	Stove	No	\$1.29	\$1.89	\$0.00	\$0.00	\$0.00	1	1	-2	0	-2	0	2
National	\$0.00	Stove	No	\$1.29	\$1.89	\$0.00	\$0.00	\$0.00	1	1	-2	-2	0	0	2
None	\$0.00	Stove	No	\$1.29	\$1.89	\$0.00	\$0.00	\$0.00	1	1	-2	-2	-2	1	2

---

These next steps aggregate the data and display a subset of the results:

```
proc summary data=res2 nway;
  class set brand price shelf micro x1 x2 x5 x6 x8 a1 a2 a5 a6 a8 c;
  output out=agg(drop=_type_);
  where w; /* exclude unavailable, w = 0 */
run;

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

The data set is fairly large at 64,800 observations, and aggregating greatly reduces its size, which makes both the TRANSREG and the PHREG steps run in just a few seconds. This step also excludes the unavailable alternatives. When `w` is 1 (true) the alternative is available and counted, otherwise when `w` is 0 (false) the alternative is unavailable and excluded by the `where` clause and not counted. There is nothing in subsequent steps that assumes a fixed number of alternatives.

All of the variables used in the analysis are named as `class` variables in PROC SUMMARY, which reduces the data set from 64,800 observations to 286. The aggregated data for the first choice set is as follows:

---

Consumer Food Product Example

B	P	S	M									-					
r	r	h	i									R					
0	S	a	i	e	c									E			
b	e	n	c	l	r	x	x	x	x	x	a	a	a	a	a	a	Q
s	t	d	e	f	o	1	2	5	6	8	1	2	5	6	8	c	-
1	1	Client	\$1.29	No	Stove	\$1.29	\$2.39	\$1.99	\$0.00	\$1.99	0	1	1	-2	1	1	74
2	1	Client	\$1.29	No	Stove	\$1.29	\$2.39	\$1.99	\$0.00	\$1.99	0	1	1	-2	1	2	226
3	1	Extension	\$2.39	No	Micro	\$1.29	\$2.39	\$1.99	\$0.00	\$1.99	1	0	1	-2	1	1	220
4	1	Extension	\$2.39	No	Micro	\$1.29	\$2.39	\$1.99	\$0.00	\$1.99	1	0	1	-2	1	2	80
5	1	National	\$1.99	No	Stove	\$1.29	\$2.39	\$1.99	\$0.00	\$1.99	1	1	1	-2	0	1	6
6	1	National	\$1.99	No	Stove	\$1.29	\$2.39	\$1.99	\$0.00	\$1.99	1	1	1	-2	0	2	294
7	1	None	\$0.00	No	Stove	\$1.29	\$2.39	\$1.99	\$0.00	\$1.99	1	1	1	-2	1	2	300
8	1	Regional	\$1.99	No	Stove	\$1.29	\$2.39	\$1.99	\$0.00	\$1.99	1	1	0	-2	1	2	300

---

In the first choice set, the client brand is chosen ( $c = 1$ ) a total of `_freq_` = 74 times and not chosen ( $c = 2$ ) a total of `_freq_` = 226 times. Each alternative is chosen and not chosen a total of 300 times, which is the number of subjects. These next steps code and run the analysis.

## Cross-Effects

This next step codes the design for analysis. This coding is discussed on page 509. 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). The following step does the coding:

```
proc transreg data=agg design=5000 nozeroconstant noestoremissing;
  model class(brand / zero='None')
    class(brand / zero='None' separators=' ' ') * identity(price)
    class(shelf micro / lprefix=5 0 zero='No' 'Stove')
    identity(x1 x2 x5 x6 x8) *
      class(brand / zero='None' separators=' ' ' on ')
    identity(a1 a2 a5 a6 a8) *
      class(brand / zero='None' separators=' ' ' on ') /
    lprefix=0;
  output out=coded(drop=_type_ _name_ intercept);
  id set c _freq_;
  label shelf = 'Shelf Talker'
    micro = 'Microwave';
run;
```

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

```
WARNING: This usage of * sets one group's slope to zero.  Specify |
         to allow all slopes and intercepts to vary.  Alternatively,
         specify CLASS(vars) * identity(vars) identity(vars) for
         separate within group functions and a common intercept.
         This is a change from Version 6.
```

This is because `class` is 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 displays a warning. If you get this warning while coding a choice model specifying `zero='constant-alternative-level'`, you can safely ignore it.

The analysis is the same as we have done previously with aggregate data. PROC PHREG is run to fit the mother logit model, complete with availability cross-effects as follows:

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

## Multinomial Logit Model Results

Recall that we used %phchoice(on) on page 287 to customize the output from PROC PHREG. These steps 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
Frequency Variable	_FREQ_
Ties Handling	BRESLOW
Number of Observations Read	284
Number of Observations Used	284
Sum of Frequencies Read	47400
Sum of Frequencies Used	47400



## Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Stratum	Set	Number of Alternatives	Chosen Alternatives	Not Chosen
1	1	1500	300	1200
2	2	900	300	600
3	3	900	300	600
4	4	1500	300	1200
5	5	1500	300	1200
6	6	900	300	600
7	7	1500	300	1200
8	8	1500	300	1200
9	9	1500	300	1200
10	10	1500	300	1200
11	11	900	300	600
12	12	1500	300	1200
13	13	1500	300	1200
14	14	900	300	600
15	15	900	300	600
16	16	1500	300	1200
17	17	1500	300	1200
18	18	1500	300	1200
19	19	1500	300	1200
20	20	1200	300	900
21	21	900	300	600
22	22	1200	300	900
23	23	1500	300	1200
24	24	1500	300	1200
25	25	1500	300	1200
26	26	1500	300	1200
27	27	1500	300	1200
28	28	1500	300	1200
29	29	1500	300	1200
30	30	900	300	600
31	31	1500	300	1200
32	32	1500	300	1200
33	33	1200	300	900
34	34	1200	300	900
35	35	1200	300	900
36	36	1200	300	900
-----				
Total		47400	10800	36600

## Convergence Status

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

## Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	154710.28	134305.63
AIC	154710.28	134409.63
SBC	154710.28	134788.57

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	20404.6461	52	<.0001
Score	22883.7078	52	<.0001
Wald	6444.0844	52	<.0001

## Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Client	1	8.16629	3.96395	4.2442	0.0394
Extension	1	10.30298	4.13379	6.2119	0.0127
National	1	5.41386	4.90468	1.2184	0.2697
Private	1	4.90773	4.06749	1.4558	0.2276
Regional	1	4.96459	5.93423	0.6999	0.4028
Client Price	1	-1.11653	0.77149	2.0945	0.1478
Extension Price	1	-0.99948	1.21987	0.6713	0.4126
National Price	1	1.25938	1.74132	0.5231	0.4695
Private Price	1	-1.33471	0.76283	3.0614	0.0802
Regional Price	1	-1.22852	1.48246	0.6867	0.4073
Shelf Talker	1	0.66941	0.07828	73.1204	<.0001
Micro	1	0.59645	0.06746	78.1706	<.0001
CE, Client on Client	0	0	.	.	.
CE, Client on Extension	1	-0.31640	0.78240	0.1635	0.6859
CE, Client on National	1	-0.50555	0.80031	0.3990	0.5276
CE, Client on Private	1	-0.25802	0.82061	0.0989	0.7532
CE, Client on Regional	1	1.15121	1.03011	1.2489	0.2638
CE, Extension on Client	1	-0.52993	1.22364	0.1876	0.6650
CE, Extension on Extension	0	0	.	.	.
CE, Extension on National	1	-0.55507	1.24852	0.1977	0.6566
CE, Extension on Private	1	0.20613	1.25789	0.0269	0.8698
CE, Extension on Regional	1	-0.54337	1.43547	0.1433	0.7050
CE, Regional on Client	1	-1.14955	1.07675	1.1398	0.2857
CE, Regional on Extension	1	-1.43726	1.08276	1.7620	0.1844
CE, Regional on National	1	-1.81230	1.13204	2.5629	0.1094
CE, Regional on Private	1	-1.20206	1.09592	1.2031	0.2727
CE, Regional on Regional	0	0	.	.	.

CE, Private on Client	1	-0.42457	0.75836	0.3134	0.5756
CE, Private on Extension	1	-0.35800	0.75937	0.2223	0.6373
CE, Private on National	1	-0.68966	0.79742	0.7480	0.3871
CE, Private on Private	0	0	.	.	.
CE, Private on Regional	1	-1.11543	1.08771	1.0516	0.3051
CE, National on Client	1	1.42556	1.75683	0.6584	0.4171
CE, National on Extension	1	1.03538	1.75043	0.3499	0.5542
CE, National on National	0	0	.	.	.
CE, National on Private	1	1.46740	1.78874	0.6730	0.4120
CE, National on Regional	1	-0.28269	2.28193	0.0153	0.9014
AE, Client on Client	0	0	.	.	.
AE, Client on Extension	1	0.12477	0.38019	0.1077	0.7428
AE, Client on National	1	0.10606	0.38579	0.0756	0.7834
AE, Client on Private	1	-0.04026	0.39633	0.0103	0.9191
AE, Client on Regional	1	-0.57219	0.48525	1.3904	0.2383
AE, Extension on Client	1	0.77428	0.65342	1.4041	0.2360
AE, Extension on Extension	0	0	.	.	.
AE, Extension on National	1	0.61514	0.67002	0.8429	0.3586
AE, Extension on Private	1	0.18324	0.67377	0.0740	0.7856
AE, Extension on Regional	1	0.38862	0.76269	0.2596	0.6104
AE, Regional on Client	1	0.87692	0.77389	1.2840	0.2572
AE, Regional on Extension	1	1.05490	0.77497	1.8529	0.1734
AE, Regional on National	1	1.29670	0.79530	2.6584	0.1030
AE, Regional on Private	1	0.98393	0.77581	1.6085	0.2047
AE, Regional on Regional	0	0	.	.	.
AE, Private on Client	1	0.29125	0.48172	0.3655	0.5454
AE, Private on Extension	1	0.26656	0.48436	0.3029	0.5821
AE, Private on National	1	0.49015	0.50341	0.9480	0.3302
AE, Private on Private	0	0	.	.	.
AE, Private on Regional	1	0.81907	0.74339	1.2140	0.2705
AE, National on Client	1	-1.15849	1.35844	0.7273	0.3938
AE, National on Extension	1	-0.88510	1.35042	0.4296	0.5122
AE, National on National	0	0	.	.	.
AE, National on Private	1	-1.32585	1.38251	0.9197	0.3376
AE, National on Regional	1	0.31543	1.74206	0.0328	0.8563

---

Since the number of alternatives is not constant within each choice set, the summary table has non-constant numbers of alternatives and numbers of alternatives not chosen. The number chosen, 300 (or one per subject per choice set), is constant, since each subject always chooses one alternative from each choice set regardless of the number of alternatives. The total number of alternatives ranges from 900 with three alternatives to 1500 with five alternatives.

The cross-effects are mostly nonsignificant. Since most of the cross-effects are nonsignificant, we can rerun the analysis with a simpler model as follows:

```
proc transreg data=agg design=5000 nozeroconstant norestoremissing;
  model class(brand / zero='None')
        class(brand / zero='None' separators=' ' ' ') * identity(price)
        class(shelf micro / lprefix=5 0 zero='No' 'Stove') /
        lprefix=0;
  output out=coded(drop=_type_ _name_ intercept);
  id set c _freq_;
  label shelf = 'Shelf Talker'
        micro = 'Microwave';
run;

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

The parameter estimates for the simpler model are as follows:

---

Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Client	1	5.65644	0.20231	781.6872	<.0001
Extension	1	6.42043	0.21086	927.1112	<.0001
National	1	0.77822	0.63713	1.4919	0.2219
Private	1	4.51101	0.30491	218.8745	<.0001
Regional	1	1.71388	0.99673	2.9567	0.0855
Client Price	1	-0.76985	0.09446	66.4224	<.0001
Extension Price	1	-0.50666	0.08350	36.8162	<.0001
National Price	1	0.74444	0.29078	6.5542	0.0105
Private Price	1	-1.33357	0.14151	88.8068	<.0001
Regional Price	1	-0.42010	0.46037	0.8327	0.3615
Shelf Talker	1	0.71984	0.06941	107.5588	<.0001
Micro	1	0.58407	0.05632	107.5642	<.0001

---

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

## Modeling Subject Attributes

This example uses the same design and data as we just saw, but this time we have some demographic information about our respondents that we wish to model. The following DATA step reads a subject number, the choices, and the respondent age and income (in thousands of dollars):

```
data results;
  input Subj (choose1-choose&n) (1.) age income;
  datalines;
1 222224155212222522221221222221212522 33 109
2 222225421242222122221212222221211322 56 117
3 212224521545222122221222221121112522 56 78
4 212125121212222122221222421221212522 57 107
.
.
.
299 112225121212122121521222212211212622 41 89
300 122225123242122122221211222123212322 38 95
;
```

Merging the data and design is no different from what we saw previously. To make this analysis simpler, we do not fit any cross-effects or availability cross-effects, although we certainly could. The following step performs the merge and displays a subset of the results:

```
%mktmerge(design=sasuser.EntreeChDes(drop=x1--x8 a1--a8), data=results,
  out=res2, nsets=&n, nalts=&m, setvars=choose1-choose&n)

proc print data=res2;
  by subj set; id subj set;
  where (subj = 1 and set = 1) or
        (subj = 2 and set = 2) or
        (subj = 3 and set = 3) or
        (subj = 300 and set = 36);
run;
```

A small sample of the data is as follows:

---

### Consumer Food Product Example

Subj	Set	Age	Income	Brand	Price	Micro	Shelf	w	c
1	1	33	109	Client	\$1.29	Stove	No	1	2
		33	109	Extension	\$2.39	Micro	No	1	1
		33	109	Regional	\$1.99	Stove	No	1	2
		33	109	Private	\$0.00	Micro	No	0	2
		33	109	National	\$1.99	Stove	No	1	2
		33	109	None	\$0.00	Stove	No	1	2

2	2	56	117	Client	\$1.29	Stove	No	1	2
		56	117	Extension	\$1.89	Stove	No	1	1
		56	117	Regional	\$0.00	Stove	No	0	2
		56	117	Private	\$0.00	Stove	No	0	2
		56	117	National	\$0.00	Stove	No	0	2
		56	117	None	\$0.00	Stove	No	1	2
3	3	56	78	Client	\$0.00	Stove	No	0	2
		56	78	Extension	\$2.39	Micro	No	1	1
		56	78	Regional	\$0.00	Stove	No	0	2
		56	78	Private	\$2.29	Stove	No	1	2
		56	78	National	\$0.00	Stove	No	0	2
		56	78	None	\$0.00	Stove	No	1	2
300	36	38	95	Client	\$2.09	Stove	No	1	2
		38	95	Extension	\$2.39	Micro	Talker	1	1
		38	95	Regional	\$0.00	Stove	No	0	2
		38	95	Private	\$0.00	Stove	No	0	2
		38	95	National	\$2.39	Stove	No	1	2
		38	95	None	\$0.00	Stove	No	1	2

Note that like before, the unavailable alternatives are required for the merge step. You can see that the demographic information matches the raw data and is constant within each subject. The rest of the data processing is virtually the same as well. Since we have demographic information, we do not aggregate. There would have to be ties in both the demographics and choice for aggregation to have any effect.

We use PROC TRANSREG to code, adding Age and Income to the analysis as follows:

```
proc transreg data=res2 design=5000 nozeroconstant noestoremissing;
  model class(brand / zero='None')
    identity(age income) * class(brand / zero='None' separators=' ' , ' )
    class(brand / zero='None' separators=' ' ' ') * identity(price)
    class(shelf micro / lprefix=5 0 zero='No' 'Stove') /
    lprefix=0 order=data;

  output out=code(drop=_type_ _name_ intercept);
  id subj set c w;
  label shelf = 'Shelf Talker'
    micro = 'Microwave';
  run;

data coded(drop=w); set code; where w; run; /* exclude unavailable */
```

The Age and Income variables are incorporated into the analysis by interacting them with Brand. Demographic variables must be interacted with product attributes to have any effect. If `identity(age income)` had been specified instead of `identity(age income) * class(brand / ...)` the coefficients for age and income would be zero. This is because age and income are constant within each choice set and subject combination, which means they are constant within each stratum. The second separator ' , ' is used to create names for the brand/demographic interaction terms like 'Age, Client'.

The following steps display the first coded choice set:

```
proc print data=coded(obs=4) label;
  id brand price;
  var BrandClient -- BrandPrivate Shelf Micro c;
run;

proc print data=coded(obs=4 drop=Age) label;
  id brand price;
  var Age;;
run;

proc print data=coded(obs=4 drop=Income) label;
  id brand price;
  var Income;;
run;

proc print data=coded(obs=4) label;
  id brand price;
  var BrandClientPrice -- BrandPrivatePrice;
  format BrandClientPrice -- BrandPrivatePrice best4.;
run;
```

The coded data set for the first subject and choice set is displayed next in a series of panels. The part that is new is the second and third panel, which are used to capture the brand by age and brand by income effects. The attributes and the brand effects are as follows:

---

Consumer Food Product Example

Brand	Price	Client	Extension	Regional	Private	Shelf		c
						Talker	Microwave	
Client	\$1.29	1	0	0	0	No	Stove	2
Extension	\$2.39	0	1	0	0	No	Micro	1
Regional	\$1.99	0	0	1	0	No	Stove	2
National	\$1.99	0	0	0	0	No	Stove	2

---

The age by brand effects are as follows:

---

Consumer Food Product Example						
Brand	Price	Age, Client	Age, Extension	Age, Regional	Age, Private	Age, National
Client	\$1.29	33	0	0	0	0
Extension	\$2.39	0	33	0	0	0
Regional	\$1.99	0	0	33	0	0
National	\$1.99	0	0	0	0	33

---

The income by brand effects are as follows:

---

Consumer Food Product Example						
Brand	Price	Income, Client	Income, Extension	Income, Regional	Income, Private	Income, National
Client	\$1.29	109	0	0	0	0
Extension	\$2.39	0	109	0	0	0
Regional	\$1.99	0	0	109	0	0
National	\$1.99	0	0	0	0	109

---

The alternative-specific price effects are as follows:

---

Consumer Food Product Example						
Brand	Price	Client Price	Extension Price	Regional Price	Private Price	
Client	\$1.29	1.29	0	0	0	
Extension	\$2.39	0	2.39	0	0	
Regional	\$1.99	0	0	1.99	0	
National	\$1.99	0	0	0	0	

---

The PROC PHREG specification is the same as we have used before with nonaggregated data. The following step performs the analysis:

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



This step took just about one minute and produced the following results:

---

### Consumer Food Product Example

#### The PHREG Procedure

#### Model Information

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

Number of Observations Read	47400
Number of Observations Used	47400

#### Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Pattern	Number of Choices	Number of Alternatives	Chosen Alternatives	Not Chosen
1	2400	3	1	2
2	1800	4	1	3
3	6600	5	1	4

#### Convergence Status

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

#### Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	31508.579	10939.139
AIC	31508.579	10983.139
SBC	31508.579	11143.460

#### Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	20569.4401	22	<.0001
Score	21088.4411	22	<.0001
Wald	6947.1766	22	<.0001

## Consumer Food Product Example

## The PHREG Procedure

## Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Client	1	2.04997	0.81287	6.3599	0.0117
Extension	1	0.47882	0.81714	0.3434	0.5579
Regional	1	-1.49923	1.51428	0.9802	0.3221
Private	1	3.33861	0.85485	15.2528	<.0001
National	1	-0.62955	1.06206	0.3514	0.5533
Age, Client	1	0.01142	0.00944	1.4655	0.2261
Age, Extension	1	0.00855	0.00949	0.8111	0.3678
Age, Regional	1	0.01114	0.01205	0.8548	0.3552
Age, Private	1	0.00826	0.00970	0.7247	0.3946
Age, National	1	0.00809	0.00964	0.7042	0.4014
Income, Client	1	0.03227	0.00771	17.4954	<.0001
Income, Extension	1	0.05717	0.00776	54.2991	<.0001
Income, Regional	1	0.02496	0.01014	6.0642	0.0138
Income, Private	1	0.00957	0.00794	1.4521	0.2282
Income, National	1	0.00929	0.00789	1.3858	0.2391
Client Price	1	-0.76510	0.09598	63.5410	<.0001
Extension Price	1	-0.51364	0.08465	36.8207	<.0001
Regional Price	1	-0.27824	0.46406	0.3595	0.5488
Private Price	1	-1.37957	0.14286	93.2538	<.0001
National Price	1	0.82684	0.29260	7.9852	0.0047
Shelf Talker	1	0.74026	0.07033	110.7751	<.0001
Micro	1	0.59312	0.05692	108.6006	<.0001

---

In other examples, when we use the `brief` option to produce a brief summary of the strata, the table has only one line. In this case, since our choice sets have 3, 4, or 5 alternatives, we have three rows, one for each choice set size. The coefficients for the age and income variables are generally not very significant in this analysis except an effect for income on the client brand and particularly on the extension.

# Allocation of Prescription Drugs

This example discusses an allocation study, which is a technique often used in the area of prescription drug marketing research. This example discusses designing the allocation experiment, processing the data, analyzing frequencies, analyzing proportions, coding, analysis, and results. The principles of designing an allocation study are the same as for designing a first-choice experiment, as is the coding and final analysis. However, processing the data before analysis is different.

The previous examples have all modeled simple choice. However, sometimes the response of interest is not simple first choice. For example, in prescription drug marketing, researchers often use allocation studies where multiple, not single choices are made. Physicians are asked questions like “For the next ten prescriptions you write for a particular condition, how many would you write for each of these drugs?” The response, for example, could be “5 for drug 1, none for drug 2, 3 for drug 3, and 2 for drug 4.”

## Designing the Allocation Experiment

In this study, physicians are asked to specify which of ten drugs they would prescribe to their next ten patients. In this study, ten drugs, Drug 1 — Drug 10, are available each at three different prices, \$50, \$75, and \$100. In real studies, real brand names are used and there would probably be more attributes. Since experimental design has been covered in some detail in other examples, we chose a simple design for this experiment so that we could concentrate on data processing. First, we use the `%MktRuns` autocall macro to suggest a design size. (All of the autocall macros used in this book are documented starting on page 803.) We specify `3 ** 10` for the 10 three-level factors as follows:

```
title 'Allocation of Prescription Drugs';
```

```
%mktruns(3 ** 10)
```

The results are as follows:

---

### Allocation of Prescription Drugs

#### Design Summary

Number of Levels	Frequency
3	10

#### Allocation of Prescription Drugs

Saturated = 21  
Full Factorial = 59,049

Some Reasonable Design Sizes	Violations	Cannot Be Divided By
27 *	0	
36 *	0	
45 *	0	
54 *	0	
21 S	45	9
24	45	9
30	45	9
33	45	9
39	45	9
42	45	9

\* - 100% Efficient design can be made with the MktEx macro.  
 S - Saturated Design - The smallest design that can be made.

#### Allocation of Prescription Drugs

n	Design	Reference
27	3 ** 13	Fractional-Factorial
36	2 ** 11 3 ** 12	Orthogonal Array
36	2 ** 4 3 ** 13	Orthogonal Array
36	2 ** 2 3 ** 12 6 ** 1	Orthogonal Array
36	3 ** 13 4 ** 1	Orthogonal Array
36	3 ** 12 12 ** 1	Orthogonal Array
45	3 ** 10 5 ** 1	Orthogonal Array
54	2 ** 1 3 ** 25	Orthogonal Array
54	2 ** 1 3 ** 21 9 ** 1	Orthogonal Array
54	3 ** 24 6 ** 1	Orthogonal Array
54	3 ** 20 6 ** 1 9 ** 1	Orthogonal Array
54	3 ** 18 18 ** 1	Orthogonal Array

We need at least 21 choice sets and we see the optimal sizes are all divisible by nine. We use 27 choice sets, which can give us up to 13 three-level factors.

Next, we use the %MktEx macro to create the design.\* In addition, one more factor is added to the design. This factor is used to block the design into three blocks of size 9. The following step generates the design:

```
%let nalts = 10;

%mktx(3 ** &nalts 3, n=27, seed=396)
```

\*Due to machine, SAS release, and macro differences, you might not get exactly the same design used in this book, but the differences should be slight.

The macro finds a 100% *D*-efficient design. The results are as follows:

Allocation of Prescription Drugs

Algorithm Search History

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

Allocation of Prescription Drugs

The OPTEX Procedure

Class Level Information

Class	Levels	-Values-
x1	3	1 2 3
x2	3	1 2 3
x3	3	1 2 3
x4	3	1 2 3
x5	3	1 2 3
x6	3	1 2 3
x7	3	1 2 3
x8	3	1 2 3
x9	3	1 2 3
x10	3	1 2 3
x11	3	1 2 3

Allocation of Prescription Drugs

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	100.0000	100.0000	100.0000	0.9230

The %MktEx macro always creates factor names of x1, x2, and so on with values of 1, 2, .... You can create a data set with the names and values you want and use it to rename the factors and reset the levels. This first step creates a data set with 11 variables, Block and Brand1 - Brand10. Block has values 1, 2, and 3, and the brand variables have values of 50, 75, and 100 with a dollar format. The %MktLab macro takes the data=Randomized design data set and uses the names, values, and formats in

the `key=Key` data set to make the `out=Final` data set. This data set is sorted by block and displayed. The `%MktEval` macro is called to check the results. The following steps process and evaluate the design:

```

data key(drop=i);
  input Block Brand1;
  array Brand[10];
  do i = 2 to 10; brand[i] = brand1; end;
  format brand: dollar4.;
  datalines;
1  50
2  75
3 100
;
proc print; run;

%mktlab(data=randomized, key=key)

proc sort out=sasuser.DrugAlloLinDes; by block; run;

proc print; id block; by block; run;

%mkteval(blocks=block)

```

The `key=` data set is as follows:

---

Allocation of Prescription Drugs

Obs	Block	Brand1	Brand2	Brand3	Brand4	Brand5	Brand6	Brand7	Brand8	Brand9	Brand10
1	1	\$50	\$50	\$50	\$50	\$50	\$50	\$50	\$50	\$50	\$50
2	2	\$75	\$75	\$75	\$75	\$75	\$75	\$75	\$75	\$75	\$75
3	3	\$100	\$100	\$100	\$100	\$100	\$100	\$100	\$100	\$100	\$100

---

The `%MktLab` macro displays the following mapping information:

## Variable Mapping:

x1 : Block  
 x2 : Brand1  
 x3 : Brand2  
 x4 : Brand3  
 x5 : Brand4  
 x6 : Brand5  
 x7 : Brand6  
 x8 : Brand7  
 x9 : Brand8  
 x10 : Brand9  
 x11 : Brand10

The design is as follows:

## Allocation of Prescription Drugs

Block	Brand1	Brand2	Brand3	Brand4	Brand5	Brand6	Brand7	Brand8	Brand9	Brand10
1	\$50	\$75	\$50	\$75	\$100	\$100	\$100	\$100	\$50	\$100
	\$100	\$50	\$100	\$75	\$75	\$75	\$100	\$50	\$100	\$75
	\$50	\$50	\$75	\$100	\$50	\$75	\$50	\$75	\$50	\$75
	\$75	\$50	\$50	\$50	\$100	\$75	\$75	\$100	\$75	\$75
	\$75	\$75	\$100	\$100	\$75	\$100	\$50	\$50	\$75	\$100
	\$50	\$100	\$100	\$50	\$75	\$50	\$75	\$50	\$50	\$50
	\$100	\$75	\$75	\$50	\$50	\$100	\$75	\$75	\$100	\$100
	\$100	\$100	\$50	\$100	\$100	\$50	\$50	\$100	\$100	\$50
	\$75	\$100	\$75	\$75	\$50	\$50	\$100	\$75	\$75	\$50
	2	\$100	\$75	\$50	\$100	\$75	\$50	\$100	\$75	\$75
\$100		\$100	\$100	\$75	\$50	\$75	\$75	\$100	\$75	\$100
\$50		\$75	\$100	\$50	\$50	\$50	\$50	\$100	\$100	\$75
\$75		\$50	\$100	\$100	\$50	\$100	\$100	\$100	\$50	\$50
\$50		\$100	\$75	\$100	\$100	\$75	\$100	\$50	\$100	\$100
\$100		\$50	\$75	\$50	\$100	\$100	\$50	\$50	\$75	\$50
\$50		\$50	\$50	\$75	\$75	\$100	\$75	\$75	\$100	\$50
\$75		\$75	\$75	\$75	\$100	\$50	\$75	\$50	\$50	\$75
\$75		\$100	\$50	\$50	\$75	\$75	\$50	\$75	\$50	\$100

3	\$100	\$75	\$100	\$75	\$100	\$75	\$50	\$75	\$50	\$50
	\$75	\$75	\$50	\$50	\$50	\$75	\$100	\$50	\$100	\$50
	\$50	\$75	\$75	\$100	\$75	\$75	\$75	\$100	\$75	\$50
	\$50	\$100	\$50	\$75	\$50	\$100	\$50	\$50	\$75	\$75
	\$50	\$50	\$100	\$50	\$100	\$50	\$100	\$75	\$75	\$100
	\$75	\$50	\$75	\$75	\$75	\$50	\$50	\$100	\$100	\$100
	\$75	\$100	\$100	\$100	\$100	\$100	\$75	\$75	\$100	\$75
	\$100	\$50	\$50	\$100	\$50	\$50	\$75	\$50	\$50	\$100
	\$100	\$100	\$75	\$50	\$75	\$100	\$100	\$100	\$50	\$75

Some of the evaluation results are as follows:

Allocation of Prescription Drugs  
 Canonical Correlations Between the Factors  
 There are 0 Canonical Correlations Greater Than 0.316

	Block	Brand1	Brand2	Brand3	Brand4	Brand5	Brand6	Brand7	Brand8	Brand9	Brand10
Block	1	0	0	0	0	0	0	0	0	0	0
Brand1	0	1	0	0	0	0	0	0	0	0	0
Brand2	0	0	1	0	0	0	0	0	0	0	0
Brand3	0	0	0	1	0	0	0	0	0	0	0
Brand4	0	0	0	0	1	0	0	0	0	0	0
Brand5	0	0	0	0	0	1	0	0	0	0	0
Brand6	0	0	0	0	0	0	1	0	0	0	0
Brand7	0	0	0	0	0	0	0	1	0	0	0
Brand8	0	0	0	0	0	0	0	0	1	0	0
Brand9	0	0	0	0	0	0	0	0	0	1	0
Brand10	0	0	0	0	0	0	0	0	0	0	1

Allocation of Prescription Drugs  
 Summary of Frequencies  
 There are 0 Canonical Correlations Greater Than 0.316

Frequencies

Block	9 9 9
Brand1	9 9 9
Brand2	9 9 9
Brand3	9 9 9
Brand4	9 9 9
Brand5	9 9 9
Brand6	9 9 9
Brand7	9 9 9
Brand8	9 9 9
Brand9	9 9 9
Brand10	9 9 9



```

Block Brand1      3 3 3 3 3 3 3 3 3
Block Brand2      3 3 3 3 3 3 3 3 3
Block Brand3      3 3 3 3 3 3 3 3 3
Block Brand4      3 3 3 3 3 3 3 3 3
Block Brand5      3 3 3 3 3 3 3 3 3
Block Brand6      3 3 3 3 3 3 3 3 3
Block Brand7      3 3 3 3 3 3 3 3 3
Block Brand8      3 3 3 3 3 3 3 3 3
Block Brand9      3 3 3 3 3 3 3 3 3
Block Brand10     3 3 3 3 3 3 3 3 3
.
.
.
N-Way             1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
                  1 1 1 1 1 1 1 1

```

---

We can also create a choice design from the linear arrangement and evaluate it using the %ChoiceEff macro. The design is rolled into a choice design as follows using the same technique that we used in previous examples:

```

%mkkey(Brand1-Brand10)

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.DrugAlloLinDes, key=key, alt=brand, out=rolled,
         options=nowarn)

```

At this point, the choice design can be stored in a permanent SAS data set stored for analysis time as we have done in previous examples. Alternatively, you can just store the linear arrangement and use the same (or similar) code to make the choice design at analysis time. We use the latter approach here. This is because a form that is convenient for analysis is not always convenient for design evaluation. This is explained in more detail once the evaluation results are displayed. For now, simply note that the constant alternative is denoted by a dash here, but we use a blank (or character missing value) for the analysis.

The choice design is evaluated as follows:

```
%choicereff(data=rolled,          /* candidate set of choice sets      */
            init=rolled(keep=set), /* select these sets                  */
            intiter=0,             /* evaluate without internal iterations */
                                     /* model with stdz orthogonal coding   */
                                     /* ref level for brand is '-'         */
            model=class(brand price / zero='- ' sta),
            options=relative,      /* display relative D-efficiency      */
            nsets=27,              /* number of choice sets              */
            nalts=11,              /* number of alternatives              */
            beta=zero)             /* assumed beta vector, Ho: b=0      */
```

Some of the results are as follows:

---

### Allocation of Prescription Drugs

#### Final Results

Design	1
Choice Sets	27
Alternatives	11
Parameters	12
Maximum Parameters	270
D-Efficiency	26.1557
Relative D-Eff	96.8729
D-Error	0.0382
1 / Choice Sets	0.0370

### Allocation of Prescription Drugs

n	Variable Name	Label	Variance	DF	Standard Error
1	BrandBrand__1	Brand Brand 1	0.037037	1	0.19245
2	BrandBrand__2	Brand Brand 2	0.037037	1	0.19245
3	BrandBrand__3	Brand Brand 3	0.037037	1	0.19245
4	BrandBrand__4	Brand Brand 4	0.037037	1	0.19245
5	BrandBrand__5	Brand Brand 5	0.037037	1	0.19245
6	BrandBrand__6	Brand Brand 6	0.037037	1	0.19245
7	BrandBrand__7	Brand Brand 7	0.037037	1	0.19245
8	BrandBrand__8	Brand Brand 8	0.037037	1	0.19245
9	BrandBrand__9	Brand Brand 9	0.037037	1	0.19245
10	BrandBrand_10	Brand Brand 10	0.037037	1	0.19245
11	Price50	Price 50	0.044815	1	0.21170
12	Price75	Price 75	0.044815	1	0.21170

==  
12

---

Note that there are 12 parameters, one of each of the 10 brands (10 brands minus plus one constant, minus 1) and two for price (three brands minus 1). The constant brand (not shown, but represented by '-') was explicitly designated as the reference level for brand. The standardized orthogonal contrast coding is used for both attributes. Note that each of the 11 brands (including the constant) appears in each of the 11 alternatives in each choice set, the variances for **Brand** are at the minimum value of one over the number of choice sets. The price attribute does not have a perfect relationship like this, 3 prices does not evenly divide 11 alternatives, and there is a constant alternative, so the variances for the price attribute are higher. Still, they look pretty good. Relative *D*-efficiency is 96.8729, which is pretty good. Note that like before, relative *D*-efficiency = 100 is not possible since there is a constant alternative and 3 does not divide 11. So again, this design looks pretty good.

## 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 is chosen. If all of the choice frequencies are zero, then the constant alternative is 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 0 10 0
1   1 4  1 0 0 1 3 3 0 0 2 0
1   1 5  2 0 8 0 0 0 0 0 0 0
1   1 6  0 1 3 1 0 0 0 0 1 4
1   1 7  0 1 3 1 1 2 0 0 2 0
1   1 8  0 0 3 0 0 2 1 0 0 4
1   1 9  0 2 5 0 0 0 0 0 3 0
2   210 1 1 0 2 0 3 0 1 1 1
2   211 1 0 3 1 0 1 1 0 2 1
.
.
.
;
```

In the first step, in creating an analysis data set for an allocation study, we reformat the data from one row per choice set per block per subject ( $9 \times 3 \times 100 = 2700$  observations) to one per alternative (including the constant) per choice set per block per subject ( $((10+1) \times 9 \times 3 \times 100 = 29,700$  observations).



## Allocation of Prescription Drugs

Block	Set	Brand	Count
1	1	Brand 1	0
		Brand 2	0
		Brand 3	8
		Brand 4	0
		Brand 5	2
		Brand 6	0
		Brand 7	0
		Brand 8	0
		Brand 9	0
		Brand 10	0
			0
1	2	Brand 1	0
		Brand 2	0
		Brand 3	8
		Brand 4	0
		Brand 5	0
		Brand 6	0
		Brand 7	2
		Brand 8	0
		Brand 9	0
		Brand 10	0
			0
1	3	Brand 1	0
		Brand 2	0
		Brand 3	0
		Brand 4	0
		Brand 5	0
		Brand 6	0
		Brand 7	0
		Brand 8	0
		Brand 9	10
		Brand 10	0
			0

---

The following step aggregates the data:

```
* 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;
```

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

These next steps prepare the design for analysis. We need to create a data set `Key` that describes how the factors in our design are used for analysis. It contains all of the factor names, `Brand1`, `Brand2`, ..., `Brand10`. We can run the `%MktKey` macro to get these names for cutting and pasting into the program without typing them as follows:

```
%mktkey(Brand1-Brand10)
```

The `%MktKey` macro produced the following line:

```
Brand1 Brand2 Brand3 Brand4 Brand5 Brand6 Brand7 Brand8 Brand9 Brand10
```

The next step rolls out the experimental design data set to match the choice allocations data set. The data set is transposed from one row per choice set to one row per alternative per choice set. This data set also has 297 observations. As we saw in previous examples, the `%MktRoll` macro can be used to process the design as follows:

```
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.DrugAlloLinDes, key=key, alt=brand, out=rolled,
options=nowarn)

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

The results are as follows:

---

Allocation of Prescription Drugs

Obs	Set	Brand	Price
1	1	Brand 1	\$50
2	1	Brand 2	\$75
3	1	Brand 3	\$50
4	1	Brand 4	\$75
5	1	Brand 5	\$100
6	1	Brand 6	\$100
7	1	Brand 7	\$100
8	1	Brand 8	\$100
9	1	Brand 9	\$50
10	1	Brand 10	\$100
11	1	.	.

---

Both data sets must be sorted the same way before they can be merged. The constant alternative, indicated by a missing brand, is last in the design choice set and hence is out of order. Missing must come before nonmissing for the merge. The order is correct in the `Allocs` data set since it is created by PROC SUMMARY with `Brand` as a class variable. The following step sorts the data:

```
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. The following steps merge the data and display a subset of the results:

```
data allocs2;
  merge allocs(in=flag1) rolled(in=flag2);
  by set brand;
  if flag1 ne flag2 then put 'ERROR: Merge is not 1 to 1.';
  format price dollar4.;
  run;

proc print data=allocs2(obs=22);
  var brand price count;
  sum count;
  by notsorted set;
  id set;
  run;
```

In the aggregate and combined data set, we see how often each alternative is chosen for each choice set. For example, in the first choice set, the constant alternative is chosen zero times, Brand 1 at \$100 was chosen 103 times, and so on. The 11 alternatives are chosen a total of 1000 times, 100 subjects times 10 choices each. The results are as follows:

## Allocation of Prescription Drugs

Set	Brand	Price	Count
1		.	0
	Brand 1	\$50	103
	Brand 2	\$75	58
	Brand 3	\$50	318
	Brand 4	\$75	99
	Brand 5	\$100	54
	Brand 6	\$100	83
	Brand 7	\$100	71
	Brand 8	\$100	58
	Brand 9	\$50	100
	Brand 10	\$100	56
---			-----
1			1000
2		.	10
	Brand 1	\$100	73
	Brand 2	\$50	76
	Brand 3	\$100	342
	Brand 4	\$75	55
	Brand 5	\$75	50
	Brand 6	\$75	77
	Brand 7	\$100	95
	Brand 8	\$50	71
	Brand 9	\$100	72
	Brand 10	\$75	79
---			-----
2			1000

At this point, the data set contains 297 observations (27 choice sets times 11 alternatives) showing the number of times each alternative is chosen. This data set must be augmented to also include the number of times each alternative is not chosen. For example, in the first choice set, brand 1 is chosen 103 times, which means it is 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 is 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.



The following step processes the data and displays a subset of the results:

```
%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 as follows:

---

Allocation of Prescription Drugs						
Obs	Set	Brand	Price	Count	c	
1	1		.	0	1	
2	1		.	1000	2	
3	1	Brand 1	\$50	103	1	
4	1	Brand 1	\$50	897	2	
5	1	Brand 2	\$75	58	1	
6	1	Brand 2	\$75	942	2	
7	1	Brand 3	\$50	318	1	
8	1	Brand 3	\$50	682	2	
9	1	Brand 4	\$75	99	1	
10	1	Brand 4	\$75	901	2	
11	1	Brand 5	\$100	54	1	
12	1	Brand 5	\$100	946	2	
13	1	Brand 6	\$100	83	1	
14	1	Brand 6	\$100	917	2	
15	1	Brand 7	\$100	71	1	
16	1	Brand 7	\$100	929	2	
17	1	Brand 8	\$100	58	1	
18	1	Brand 8	\$100	942	2	
19	1	Brand 9	\$50	100	1	
20	1	Brand 9	\$50	900	2	
21	1	Brand 10	\$100	56	1	
22	1	Brand 10	\$100	944	2	

---

In the first choice set, the constant alternative is chosen zero times and not chosen 1000 times, Brand 1 is chosen 103 times and not chosen  $1000 - 103 = 897$  times, Brand 2 is chosen 58 times and not chosen  $1000 - 58 = 942$  times, and so on. Note that allocation studies do not always have fixed sums, so it is important to use the %MktAllo macro or some other approach that actually counts the number of times each alternative is not chosen. It is not always sufficient to simply subtract from a fixed constant (in this case, 1000).

## Coding and Analysis

The next step codes the design for analysis. Indicator variables are created for **Brand** and **Price**. All of the PROC TRANSREG options have been discussed in other examples. The following step does the coding:

```
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. The following step performs the analysis:

```
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

Recall that we used `%phchoice(on)` on page 287 to customize the output from PROC PHREG. The results are as follows:

---

Allocation of Prescription Drugs	
The PHREG Procedure	
Model Information	
Data Set	WORK.CODED
Dependent Variable	c
Censoring Variable	c
Censoring Value(s)	2
Frequency Variable	Count
Ties Handling	BRESLOW
Number of Observations Read	583
Number of Observations Used	583
Sum of Frequencies Read	297000
Sum of Frequencies Used	297000

## Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Stratum	Set	Number of Alternatives	Chosen Alternatives	Not Chosen
1	1	11000	1000	10000
2	2	11000	1000	10000
3	3	11000	1000	10000
4	4	11000	1000	10000
5	5	11000	1000	10000
6	6	11000	1000	10000
7	7	11000	1000	10000
8	8	11000	1000	10000
9	9	11000	1000	10000
10	10	11000	1000	10000
11	11	11000	1000	10000
12	12	11000	1000	10000
13	13	11000	1000	10000
14	14	11000	1000	10000
15	15	11000	1000	10000
16	16	11000	1000	10000
17	17	11000	1000	10000
18	18	11000	1000	10000
19	19	11000	1000	10000
20	20	11000	1000	10000
21	21	11000	1000	10000
22	22	11000	1000	10000
23	23	11000	1000	10000
24	24	11000	1000	10000
25	25	11000	1000	10000
26	26	11000	1000	10000
27	27	11000	1000	10000
-----				
Total		297000	27000	270000

## Convergence Status

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

## Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	502505.13	489062.66
AIC	502505.13	489086.66
SBC	502505.13	489185.11

## Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	13442.4676	12	<.0001
Score	18340.8415	12	<.0001
Wald	14087.6778	12	<.0001

## Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Brand 1	1	2.09906	0.06766	962.5297	<.0001
Brand 2	1	2.09118	0.06769	954.5113	<.0001
Brand 3	1	3.54204	0.06484	2984.4698	<.0001
Brand 4	1	2.09710	0.06766	960.5277	<.0001
Brand 5	1	2.08523	0.06771	948.4791	<.0001
Brand 6	1	2.03530	0.06790	898.6218	<.0001
Brand 7	1	2.06920	0.06777	932.3154	<.0001
Brand 8	1	2.08573	0.06771	948.9824	<.0001
Brand 9	1	2.11705	0.06759	980.9640	<.0001
Brand 10	1	2.06363	0.06779	926.7331	<.0001
\$50	1	0.00529	0.01628	0.1058	0.7450
\$75	1	0.0005304	0.01629	0.0011	0.9740
\$100	0	0	.	.	.

---

The output shows that there are 27 strata, one per choice set, each consisting of 1000 chosen alternatives (10 choices by 100 subjects) and 10,000 unchosen alternatives. All of the brand coefficients are “significant,” with the Brand 3 effect being by far the strongest. (We will soon see that statistical significance should be ignored with allocation studies.) There is no price effect.

## Analyzing Proportions

Recall that we collected data by asking physicians to report which brands they would prescribe the next ten times they write prescriptions. Alternatively, we could ask them to report the *proportion* of time they would prescribe each brand. We can simulate having proportion data by dividing our count data by 10. This means our frequency variable no longer contains integers, so we need to specify the `nottruncate` option in the PROC PHREG `freq` statement to permit “noninteger frequencies.” This is illustrated in the following steps:

```
data coded2;
  set coded;
  count = count / 10;
run;
```

```

proc phreg data=coded2;
  where count > 0;
  model c*c(2) = &_trgind / ties=breslow;
  freq count / nottruncate;
  strata set;
run;

```

When we do this, we see the number of alternatives and the number chosen and not chosen decrease by a factor of 10 as do all of the Chi-Square tests. The coefficients are unchanged. This implies that market share calculations are invariant to the different scalings of the frequencies. However, the  $p$ -values are not invariant. The sample size is artificially inflated when counts are used so  $p$ -values are not interpretable in an allocation study. When proportions are used, each subject is contributing 1 to the number chosen instead of 10, just like a normal choice study, so  $p$ -values have meaning. The results are as follows:

---

### Allocation of Prescription Drugs

#### The PHREG Procedure

##### Model Information

Data Set	WORK.CODED2
Dependent Variable	c
Censoring Variable	c
Censoring Value(s)	2
Frequency Variable	Count
Ties Handling	BRESLOW
Number of Observations Read	583
Number of Observations Used	583
Sum of Frequencies Read	29700
Sum of Frequencies Used	29700

##### Summary of Subjects, Sets, and Chosen and Unchosen Alternatives

Stratum	Set	Number of Alternatives	Chosen Alternatives	Not Chosen
1	1	1100.0	100.0	1000.0
2	2	1100.0	100.0	1000.0
3	3	1100.0	100.0	1000.0
4	4	1100.0	100.0	1000.0
5	5	1100.0	100.0	1000.0
6	6	1100.0	100.0	1000.0
7	7	1100.0	100.0	1000.0
8	8	1100.0	100.0	1000.0
9	9	1100.0	100.0	1000.0
10	10	1100.0	100.0	1000.0

11	11	1100.0	100.0	1000.0
12	12	1100.0	100.0	1000.0
13	13	1100.0	100.0	1000.0
14	14	1100.0	100.0	1000.0
15	15	1100.0	100.0	1000.0
16	16	1100.0	100.0	1000.0
17	17	1100.0	100.0	1000.0
18	18	1100.0	100.0	1000.0
19	19	1100.0	100.0	1000.0
20	20	1100.0	100.0	1000.0
21	21	1100.0	100.0	1000.0
22	22	1100.0	100.0	1000.0
23	23	1100.0	100.0	1000.0
24	24	1100.0	100.0	1000.0
25	25	1100.0	100.0	1000.0
26	26	1100.0	100.0	1000.0
27	27	1100.0	100.0	1000.0
-----				
Total		29700.0	2700.0	27000.0

## Convergence Status

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

## Model Fit Statistics

Criterion	Without Covariates	With Covariates
-2 LOG L	37816.553	36472.307
AIC	37816.553	36496.307
SBC	37816.553	36567.119

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	1344.2468	12	<.0001
Score	1834.0841	12	<.0001
Wald	1408.7678	12	<.0001

## Multinomial Logit Parameter Estimates

	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
Brand 1	1	2.09906	0.21395	96.2530	<.0001
Brand 2	1	2.09118	0.21404	95.4511	<.0001
Brand 3	1	3.54204	0.20503	298.4470	<.0001
Brand 4	1	2.09710	0.21398	96.0528	<.0001
Brand 5	1	2.08523	0.21411	94.8479	<.0001
Brand 6	1	2.03530	0.21470	89.8622	<.0001
Brand 7	1	2.06920	0.21430	93.2315	<.0001
Brand 8	1	2.08573	0.21411	94.8982	<.0001
Brand 9	1	2.11705	0.21375	98.0964	<.0001
Brand 10	1	2.06363	0.21436	92.6733	<.0001
\$50	1	0.00529	0.05148	0.0106	0.9181
\$75	1	0.0005304	0.05152	0.0001	0.9918
\$100	0	0	.	.	.

---

## Chair Design with Generic Attributes

This study illustrates creating an experimental design for a purely generic choice model. This example discusses generic attributes, alternative swapping, choice set swapping, and constant alternatives. In a purely generic study, there are no brands, just bundles of attributes. Also see page 102 in the experimental design chapter for examples of how to combinatorially construct optimal generic choice designs for certain problems. You can find additional examples in the documentation for the `%ChoiceEff` macro beginning on page 806.

Say a manufacturer is interested in designing one or more new chairs. The manufacturer can vary the attributes of the chairs, present subjects with competing chair designs, and model the effects of the attributes on choice. The attributes of interest are as follows:

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

Since seeing descriptions of chairs is not the same as seeing and sitting in the actual chairs, the manufacturer is going to actually make sample chairs for people to try and choose from. Subjects are shown groups of three chairs at a time. If we were to make our design using the approach discussed in previous examples, we would use the `%MktEx` autocall macro to create a design with 15 factors, five for the first chair, five for the second chair, and five for the third chair. This design would need at least  $15 \times (3 - 1) + 1 = 31$  runs and 93 sample chairs. The following step shows how we could have made the design:\*

```
title 'Generic Chair Attributes';

* This design will not be used;
%mktex(3 ** 15, n=36, seed=238)

%mktkey(3 5)

%mktroll(design=randomized, key=key, out=cand)
```

The `%MktEx` approach to designing an experiment like this allows you to fit very general models including models with alternative-specific effects and even mother logit models. However, at analysis time for this purely generic model, we fit a model with 10 parameters, two for each of the five factors, `class(x1-x5)`. Creating a design with over  $31 \times 3 = 93$  chairs is way too expensive. In ordinary linear model designs, we need at least as many runs as parameters. In choice designs, we need to count the total number of alternatives across all choice sets, subtract the number of choice sets, and this number must be at least as large as the number of parameters. Equivalently, each choice set allows us to estimate  $m - 1$  parameters, where  $m$  is the number of alternatives in that choice set. In this case, we could fit our purely generic model with as few as  $10 / (3 - 1) = 5$  choice sets.

---

\*Due to machine, SAS release, and macro differences, you might not get exactly the same design used in this book, but the differences should be slight.



Since we only need a simple generic model for this example, and since our chair manufacturing for our research is expensive, we do not use the `%MktEx` approach for designing our choice experiment. Instead, we use a different approach that lets us get a smaller design that is adequate for our model and budget. Recall the discussion of linear model design efficiency, choice model design efficiency, and using linear model design efficiency as a surrogate for choice design goodness starting on page 62. Instead of using linear model design efficiency as a surrogate for choice design goodness, we can directly optimize choice design efficiency given an assumed model and parameter vector  $\beta$ . This approach uses the `%ChoiceEff` macro.

## Generic Attributes, Alternative Swapping, Large Candidate Set

This part of the example illustrates using the `%ChoiceEff` macro for efficient choice designs, using its algorithm that builds a design from candidate alternatives (as opposed to candidates consisting of entire choice sets). First, we use the `%MktRuns` macro to suggest a candidate-set size as follows:

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

Some of the results are as follows:

---

### Generic Chair Attributes

#### Design Summary

	Number of Levels	Frequency	
	3	5	
Saturated	= 11		
Full Factorial	= 243		
Some Reasonable Design Sizes	Violations	Cannot Be Divided By	
18 *	0		
27 *	0		
36 *	0		
12	10	9	
15	10	9	
21	10	9	
24	10	9	
30	10	9	
33	10	9	
11 S	15	3 9	

- \* - 100% Efficient design can be made with the MktEx macro.
- S - Saturated Design - The smallest design that can be made.  
 Note that the saturated design is not one of the recommended designs for this problem. It is shown to provide some context for the recommended sizes.

#### Generic Chair Attributes

n	Design	Reference
18	2 ** 1 3 ** 7	Orthogonal Array
18	3 ** 6 6 ** 1	Orthogonal Array
27	3 ** 13	Fractional-Factorial
27	3 ** 9 9 ** 1	Fractional-Factorial
36	2 ** 11 3 ** 12	Orthogonal Array
36	2 ** 10 3 ** 8 6 ** 1	Orthogonal Array
36	2 ** 4 3 ** 13	Orthogonal Array
36	2 ** 3 3 ** 9 6 ** 1	Orthogonal Array
36	2 ** 2 3 ** 12 6 ** 1	Orthogonal Array
36	2 ** 2 3 ** 5 6 ** 2	Orthogonal Array
36	2 ** 1 3 ** 8 6 ** 2	Orthogonal Array
36	3 ** 13 4 ** 1	Orthogonal Array
36	3 ** 12 12 ** 1	Orthogonal Array
36	3 ** 7 6 ** 3	Orthogonal Array

---

We could use candidate sets of size: 18, 27 or 36. Additionally, since this problem is small, we could try an 81-run fractional-factorial design or the 243-run full-factorial design. We choose the 243-run full-factorial design, since it is reasonably small and it gives the macro the most freedom to find a good design.\*

We use the %MktEx macro to create a candidate set. The candidate set consists of 5 three-level factors, one for each of the five generic attributes. The following steps create the candidates:

```
%mktex(3 ** 5, n=243)

proc print data=design(obs=27); run;
```

Part of the candidate set is as follows:

---

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

---

 Generic Chair Attributes

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

---

Next, we search that candidate set for an efficient design for the model specification `class(x1-x5)` and the assumption  $\beta = \mathbf{0}$ . We use the `%ChoiceEff` autocall macro to do this. (All of the autocall macros used in this book are documented starting on page 803.) 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  $\beta$ . The `%ChoiceEff` macro uses a modified Fedorov algorithm (Fedorov 1972; Cook and Nachtsheim 1980) to optimize the choice model variance matrix. We are using the largest possible candidate set for this problem, the full-factorial design, and we ask for more than the default number of iterations, so run time is slower than it could be. However, we are requesting a very small number of choice sets. Building the chairs is expensive, so we want to get a really good but small design. The following specification requests a generic design with six choice sets each consisting of three alternatives:

```

%choicereff(data=design,          /* candidate set of alternatives      */
             model=class(x1-x5 / sta), /* model with stdzd orthogonal coding */
             nsets=6,           /* number of choice sets            */
             maxiter=100,       /* maximum number of designs to make */
             seed=121,         /* random number seed                */
             flags=3,          /* 3 alternatives, generic candidates */
             options=relative,  /* display relative D-efficiency     */
             beta=zero)        /* assumed beta vector, Ho: b=0      */

```

The `data=final` option names the input data set of candidates. The `model=class(x1-x5 / sta)` option specifies the most general model that is considered at analysis time. This is a main-effects model for `x1-x5` with a standardized orthogonal contrast coding. You must specify this coding if you want to see relative  $D$ -efficiency on a 0 to 100 scale. The `nsets=6` option specifies the number of choice sets. Note that this is considerably smaller than the minimum of 31 that is required if we are just using the `%MktEx` linear-arrangement approach ( $6 \times 3 = 18$  chairs instead of  $31 \times 3 = 93$  chairs). The `maxiter=100` option requests 100 designs based on 100 random initial designs (by default, `maxiter=2`). The `seed=121` option specifies the random number seed. The `flags=3` option specifies that there are three alternatives in a purely generic design.<sup>†</sup> The `options=relative` option scales  $D$ -efficiency relative to the hypothetical maximum value so that we can get a 0 to 100 scale for  $D$ -efficiency. Relative  $D$ -efficiency is not displayed by default since it is only meaningful for some designs. The `beta=zero` option specifies the assumption  $\beta = \mathbf{0}$ . A vector of numbers like `beta=-1 0 -1 0 -1 0 -1 0 -1 0 -1 0` could be specified. (See page 878 for an example of this.) When you wish to assume all parameters are zero, you can specify `beta=zero` instead of typing a vector of the zeros. You can also omit the `beta=` option if you just want the macro to list the parameters. You can use this list to ensure that you specify the parameters in the right order.

The first part of the output from the macro is a list of all of the effects generated and the assumed values of  $\beta$ . The effects are as follows:

---

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

---

<sup>†</sup>The option name `flags=` comes from the fact that in the context of other types of designs (designs with brands or labeled alternatives), this option provides a set of “flag” variables that specify which candidates can be used for which alternatives.

It is very important that you check this list and make sure it is correct. In particular, when you are explicitly specifying the  $\beta$  vector, you need to make sure you specified all of the values in the right order. If you are not specifying a beta vector you can check the names and labels in the final table of variances and standard errors.

Next, the macro produces the iteration history, which is different from the iteration histories we are used to seeing in the %MktEx macro. The %ChoiceEff macro uses PROC IML and a modified Fedorov algorithm to iteratively improve the efficiency of the choice design given the specified candidates, model, and  $\beta$ . Note that these unscaled efficiencies are *not* on a 0 to 100 scale. This step took about 2 minutes. Some of the results are as follows:

---

Generic Chair Attributes			
Design	Iteration	D-Efficiency	D-Error
-----			
1	0	1.83063 *	0.54626
	1	4.91557 *	0.20344
	2	5.20220 *	0.19223
	3	5.40987 *	0.18485
	4	5.42657 *	0.18428
.			
.			
.			
Design	Iteration	D-Efficiency	D-Error
-----			
34	0	2.44100	0.40967
	1	4.77565	0.20940
	2	5.49875	0.18186
	3	6.00000 *	0.16667
	4	6.00000	0.16667
.			
.			
.			
Design	Iteration	D-Efficiency	D-Error
-----			
100	0	2.37105	0.42175
	1	5.22899	0.19124
	2	5.41804	0.18457
	3	5.41804	0.18457

---

An asterisk is used to indicate places where  $D$ -efficiency is greater than any previously reported value.

Next, the macro shows which design it chose and the final  $D$ -efficiency and  $D$ -error ( $D$ -efficiency =  $1 / D$ -error). The results are as follows:

---

 Generic Chair Attributes

## Final Results

Design	34
Choice Sets	6
Alternatives	3
Parameters	10
Maximum Parameters	12
D-Efficiency	6.0000
Relative D-Eff	100.0000
D-Error	0.1667
1 / Choice Sets	0.1667

---

Next, it shows the variance, standard error, and  $df$  for each effect as follows:

---

## Generic Chair Attributes

n	Variable		Variance	DF	Standard Error
	Name	Label			
1	x11	x1 1	0.16667	1	0.40825
2	x12	x1 2	0.16667	1	0.40825
3	x21	x2 1	0.16667	1	0.40825
4	x22	x2 2	0.16667	1	0.40825
5	x31	x3 1	0.16667	1	0.40825
6	x32	x3 2	0.16667	1	0.40825
7	x41	x4 1	0.16667	1	0.40825
8	x42	x4 2	0.16667	1	0.40825
9	x51	x5 1	0.16667	1	0.40825
10	x52	x5 2	0.16667	1	0.40825
				==	
				10	

---

It is important to ensure that each effect is estimable: ( $df = 1$ ). Usually, when all of the variances are constant, like we see in this table, it means that the macro has found the optimal design. In fact, in this case, we see that all of the variances in the last table equal both  $D$ -Error and one over the number of choice sets in the preceding table. This means that relative  $D$  efficiency, which is also shown in the preceding table must equal 100. With the standardized orthogonal contrast coding, and a situation like this where an optimal design is possible,  $D$ -efficiency ranges from 0 to the number of choice sets. In the optimal design,  $D$ -efficiency = 6, the number of sets, and relative  $D$ -efficiency is 100 times  $D$ -efficiency divided by the number of choice sets.

The data set `Best` contains the final, best design found. It is displayed as follows:

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

The data set contains: `Design` – the number of the design with the maximum  $D$ -efficiency,

`Efficiency` – the  $D$ -efficiency of this design,

`Index` – the candidate set observation number,

`Set` – the choice set number,

`Prob` – the probability that this alternative will be chosen given  $\beta$ ,

`n` – the observation number,

`x1-x5` – the design, and

`_f1-f3` – the automatically generated alternative flags.

The data set is as follows:

---

Generic Chair Attributes													
Set	Design	Efficiency	Index	Prob	n	x1	x2	x3	x4	x5	_f1	_f2	_f3
1	34	6	183	0.33333	595	3	1	3	1	3	1	1	1
	34	6	62	0.33333	596	1	3	1	3	2	1	1	1
	34	6	121	0.33333	597	2	2	2	2	1	1	1	1
2	34	6	217	0.33333	598	3	3	1	1	1	1	1	1
	34	6	45	0.33333	599	1	2	2	3	3	1	1	1
	34	6	104	0.33333	600	2	1	3	2	2	1	1	1
3	34	6	215	0.33333	601	3	2	3	3	2	1	1	1
	34	6	147	0.33333	602	2	3	2	1	3	1	1	1
	34	6	4	0.33333	603	1	1	1	2	1	1	1	1
4	34	6	78	0.33333	604	1	3	3	2	3	1	1	1
	34	6	178	0.33333	605	3	1	2	3	1	1	1	1
	34	6	110	0.33333	606	2	2	1	1	2	1	1	1
5	34	6	90	0.33333	607	2	1	1	3	3	1	1	1
	34	6	46	0.33333	608	1	2	3	1	1	1	1	1
	34	6	230	0.33333	609	3	3	2	2	2	1	1	1
6	34	6	195	0.33333	610	3	2	1	2	3	1	1	1
	34	6	11	0.33333	611	1	1	2	1	2	1	1	1
	34	6	160	0.33333	612	2	3	3	3	1	1	1	1

---

This design has 18 runs (6 choice sets  $\times$  3 alternatives). Notice that in this design, each level occurs exactly once in each factor and each choice set. To use this design for analysis, you only need the variables `Set` and `x1-x5`. Since it is already in choice design format, it does not need to be processed using the `%MktRoll` macro. Since data collection, processing, and analysis have already been covered in detail in other examples, this example concentrates solely on experimental design.

There is one more test that should be run before a design is used. The `%MktDups` macro checks the design to see if any choice sets are duplicates of any other choice sets. The following steps evaluate the design:

```
%mktdups(generic, data=best, nalts=3, factors=x1-x5)
```

The results are as follows:

---

```
Design:          Generic
Factors:         x1-x5
                 x1 x2 x3 x4 x5
Sets w Dup Alts: 0
Duplicate Sets:  0
```

---

The first line of the table tells us that this is a generic design. The second line tells us the factors as specified in the `factors=` option. These are followed by the actual variable names for the factors. The next line reports the number of choice sets that have duplicate alternatives—that is, the number of times an alternative appears in a set more than once. The last line reports the number of duplicate choice sets. In this case, there are no duplicates of either type. If there are duplicates, then changing the random number seed might help. Changing other aspects of the design or the approach for making the design might help as well.

## Generic Attributes, Alternative Swapping, Small Candidate Set

In this part of this example, we try to make an equivalent design to the one we just made, only this time using a smaller candidate set. The following steps create and evaluate the design:

```
%mktex(3 ** 5, n=18)

%choicetex(data=design,          /* candidate set of alternatives      */
            model=class(x1-x5 / sta), /* model with stdzd orthogonal coding */
            nsets=6,             /* number of choice sets          */
            maxiter=20,         /* maximum number of designs to make */
            seed=121,           /* random number seed              */
            flags=3,            /* 3 alternatives, generic candidates */
            options=relative,    /* display relative D-efficiency    */
            beta=zero)           /* assumed beta vector, Ho: b=0     */

proc print; by set; id set; run;

%mktdups(generic, data=best, nalts=3, factors=x1-x5)
```



This time, instead of creating a full-factorial candidate set, we asked for 5 three-level factors from the  $L_{18}$ , an orthogonal array in 18 runs. We also asked for fewer iterations in the %ChoiceEff macro. Since the candidate set is much smaller, the macro should be able to find the best design available in this candidate set fairly easily. Some of the results are as follows:

---

Generic Chair Attributes

n	Name	Beta	Label
1	x11	0	x1 1
2	x12	0	x1 2
3	x21	0	x2 1
4	x22	0	x2 2
5	x31	0	x3 1
6	x32	0	x3 2
7	x41	0	x4 1
8	x42	0	x4 2
9	x51	0	x5 1
10	x52	0	x5 2

Generic Chair Attributes

Design	Iteration	D-Efficiency	D-Error
-----			
1	0	0	.
	1	4.35495 *	0.22962
	2	4.35495	0.22962
	.		
	.		
	.		
-----			
20	0	1.89505	0.52769
	1	4.44909	0.22476
	2	5.05750	0.19773
	3	5.37575	0.18602
	4	6.00000	0.16667
	5	6.00000	0.16667

Generic Chair Attributes

Final Results

Design 4  
 Choice Sets 6  
 Alternatives 3  
 Parameters 10  
 Maximum Parameters 12  
 D-Efficiency 6.0000  
 Relative D-Eff 100.0000  
 D-Error 0.1667  
 1 / Choice Sets 0.1667

Generic Chair Attributes

n	Variable		Variance	DF	Standard Error
	Name	Label			
1	x11	x1 1	0.16667	1	0.40825
2	x12	x1 2	0.16667	1	0.40825
3	x21	x2 1	0.16667	1	0.40825
4	x22	x2 2	0.16667	1	0.40825
5	x31	x3 1	0.16667	1	0.40825
6	x32	x3 2	0.16667	1	0.40825
7	x41	x4 1	0.16667	1	0.40825
8	x42	x4 2	0.16667	1	0.40825
9	x51	x5 1	0.16667	1	0.40825
10	x52	x5 2	0.16667	1	0.40825
				==	
				10	

Generic Chair Attributes

Set	Design	Efficiency	Index	Prob	n	f1	f2	f3	x1	x2	x3	x4	x5
1	4	6	18	0.33333	55	1	1	1	3	3	3	3	3
	4	6	1	0.33333	56	1	1	1	1	1	1	1	1
	4	6	9	0.33333	57	1	1	1	2	2	2	2	2
2	4	6	3	0.33333	58	1	1	1	1	2	1	3	3
	4	6	12	0.33333	59	1	1	1	2	3	2	1	1
	4	6	14	0.33333	60	1	1	1	3	1	3	2	2
3	4	6	15	0.33333	61	1	1	1	3	2	1	2	1
	4	6	8	0.33333	62	1	1	1	2	1	3	1	3
	4	6	5	0.33333	63	1	1	1	1	3	2	3	2
4	4	6	16	0.33333	64	1	1	1	3	2	2	1	3
	4	6	6	0.33333	65	1	1	1	1	3	3	2	1
	4	6	7	0.33333	66	1	1	1	2	1	1	3	2

5	4	6	4	0.33333	67	1	1	1	1	2	3	1	2
	4	6	13	0.33333	68	1	1	1	3	1	2	3	1
	4	6	11	0.33333	69	1	1	1	2	3	1	2	3
6	4	6	17	0.33333	70	1	1	1	3	3	1	1	2
	4	6	2	0.33333	71	1	1	1	1	1	2	2	3
	4	6	10	0.33333	72	1	1	1	2	2	3	3	1

Design:           Generic  
 Factors:          x1-x5  
                   x1 x2 x3 x4 x5  
 Sets w Dup Alts: 0  
 Duplicate Sets:  0

Notice that we got the same  $D$ -efficiency and variances as before ( $D$ -efficiency = 6 and all variances are  $1/6 \approx 0.16667$ ). Also notice the `Index` variable in the design (which is the candidate set row number). Each candidate appears in the design exactly once, and the `%MktDups` macro confirms that there are no duplicates. As is shown in the experimental design chapter starting on page 102, for problems like this (all generic attributes, no brands, no constant alternative, total number of alternatives equal to the number of runs in an orthogonal design, all factors available in that orthogonal design, and an assumed  $\beta$  vector of zero) that the optimal design can be created by optimally sorting the rows of an orthogonal design into choice sets, and the `%ChoiceEff` macro can do this quite well. More directly, this design could be made from the orthogonal array  $3^6 6^1$  in 18 runs by using the six-level factor as the choice set number.

Six choice sets is a bit small. If you can afford a larger number, it would be good to try a larger design. In this case, nine choice sets are requested using a fractional-factorial candidate set in 27 runs. Notice that like before, the number of runs in the candidate set is chosen to be the product of the number of choice sets and the number of alternatives in each choice set. The following steps generate and evaluate the design:

```
%mktex(3 ** 5, n=27, seed=382)

%choiceff(data=design,          /* candidate set of alternatives      */
           model=class(x1-x5 / sta), /* model with stdzd orthogonal coding */
           nsets=9,             /* number of choice sets           */
           maxiter=20,          /* maximum number of designs to make */
           seed=121,            /* random number seed              */
           flags=3,             /* 3 alternatives, generic candidates */
           options=relative,    /* display relative D-efficiency    */
           beta=zero)           /* assumed beta vector, Ho: b=0     */

proc print; id set; by set; var index prob x;; run;

%mktdups(generic, data=best, nalts=3, factors=x1-x5)
```

The results summary, effects, and the variances are as follows:

Generic Chair Attributes

Final Results

Design	4
Choice Sets	9
Alternatives	3
Parameters	10
Maximum Parameters	18
D-Efficiency	9.0000
Relative D-Eff	100.0000
D-Error	0.1111
1 / Choice Sets	0.1111

Generic Chair Attributes

n	Variable		Variance	DF	Standard Error
	Name	Label			
1	x11	x1 1	0.11111	1	0.33333
2	x12	x1 2	0.11111	1	0.33333
3	x21	x2 1	0.11111	1	0.33333
4	x22	x2 2	0.11111	1	0.33333
5	x31	x3 1	0.11111	1	0.33333
6	x32	x3 2	0.11111	1	0.33333
7	x41	x4 1	0.11111	1	0.33333
8	x42	x4 2	0.11111	1	0.33333
9	x51	x5 1	0.11111	1	0.33333
10	x52	x5 2	0.11111	1	0.33333
				==	
				10	

Generic Chair Attributes

Set	Index	Prob	x1	x2	x3	x4	x5
1	22	0.33333	3	2	1	1	3
	8	0.33333	1	3	2	2	1
	12	0.33333	2	1	3	3	2
2	23	0.33333	3	2	1	2	1
	10	0.33333	2	1	3	1	3
	9	0.33333	1	3	2	3	2
3	6	0.33333	1	2	3	3	1
	20	0.33333	3	1	2	2	3
	16	0.33333	2	3	1	1	2

4	13	0.33333	2	2	2	1	1
	26	0.33333	3	3	3	2	2
	3	0.33333	1	1	1	3	3
5	24	0.33333	3	2	1	3	2
	7	0.33333	1	3	2	1	3
	11	0.33333	2	1	3	2	1
6	14	0.33333	2	2	2	2	2
	27	0.33333	3	3	3	3	3
	1	0.33333	1	1	1	1	1
7	15	0.33333	2	2	2	3	3
	25	0.33333	3	3	3	1	1
	2	0.33333	1	1	1	2	2
8	21	0.33333	3	1	2	3	1
	17	0.33333	2	3	1	2	3
	4	0.33333	1	2	3	1	2
9	5	0.33333	1	2	3	2	3
	18	0.33333	2	3	1	3	1
	19	0.33333	3	1	2	1	2

Design: Generic  
 Factors: x1-x5  
 x1 x2 x3 x4 x5  
 Sets w Dup Alts: 0  
 Duplicate Sets: 0

---

Notice that like before, the variances are constant, but in this case smaller at  $1/9$ , and each candidate appears once. This is an optimal design in 9 choice sets. More directly, this design could be made from the orthogonal array  $3^9 1$  in 27 runs by using the nine-level factor as the choice set number.

## Generic Attributes, a Constant Alternative, and Alternative Swapping

Now let's make a design for the same problem but this time with a constant alternative. We first use the %MktEx macro just like before to make a design for the nonconstant alternatives, then we use a DATA step to add the flags and a constant alternative as follows:

```

title 'Generic Chair Attributes';

%mktext(3 ** 5, n=243, seed=306)

data final(drop=i);
  set design end=eof;
  retain f1-f3 1 f4 0;
  output;
  if eof then do;
    array x[9] x1-x5 f1-f4;
    do i = 1 to 9; x[i] = i le 5 or i eq 9; end;
    output;
    end;
  run;

proc print data=final(where=(x1 eq x3 and x2 eq x4 and x3 eq x5 or f4)); run;

```

A sample of the observations in the candidate set is as follows:

---

Generic Chair Attributes									
Obs	x1	x2	x3	x4	x5	f1	f2	f3	f4
1	1	1	1	1	1	1	1	1	0
31	1	2	1	2	1	1	1	1	0
61	1	3	1	3	1	1	1	1	0
92	2	1	2	1	2	1	1	1	0
122	2	2	2	2	2	1	1	1	0
152	2	3	2	3	2	1	1	1	0
183	3	1	3	1	3	1	1	1	0
213	3	2	3	2	3	1	1	1	0
243	3	3	3	3	3	1	1	1	0
244	1	1	1	1	1	0	0	0	1

---

The first 243 observations can be used for any of the first three alternatives ( $f1 = f2 = f3 = 1$ ,  $f4 = 0$ ) and the 244<sup>th</sup> observation can only be used for fourth or constant alternative ( $f1 = f2 = f3 = 0$ ,  $f4 = 1$ ). In this example, the constant alternative is composed solely from the first level of each factor. Of course this could be changed depending on the situation. The %ChoiceEff macro invocation is the same as before, except now we have four flag variables. The following steps generate and evaluate the design:

```

%choiceff(data=final,          /* candidate set of alternatives */
           model=class(x1-x5 / sta), /* model with stdzd orthogonal coding */
           nsets=6,           /* number of choice sets */
           maxiter=100,       /* maximum number of designs to make */
           seed=121,         /* random number seed */
           flags=f1-f4,      /* flag which alt can go where, 4 alts */
           options=relative, /* display relative D-efficiency */
           beta=zero)        /* assumed beta vector, Ho: b=0 */

proc print; by set; id set; run;

%mktdups(generic, data=best, nalts=4, factors=x1-x5)

```

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. Some of the results are as follows:

---

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	2.20693 *	0.45312
	1	4.67998 *	0.21368
	2	4.87965 *	0.20493
	3	4.90282 *	0.20396
	.		
	.		
	.		

Design	Iteration	D-Efficiency	D-Error
13	0	2.56694	0.38957
	1	4.54049	0.22024
	2	4.75518	0.21030
	3	4.99035	0.20039
	4	5.19495 *	0.19249
	5	5.21381 *	0.19180
.			
.			
.			

Design	Iteration	D-Efficiency	D-Error
100	0	2.74564	0.36421
	1	4.59264	0.21774
	2	4.80304	0.20820
	3	4.88340	0.20478

Generic Chair Attributes

#### Final Results

Design	13
Choice Sets	6
Alternatives	4
Parameters	10
Maximum Parameters	18
D-Efficiency	5.2138
Relative D-Eff	86.8968
D-Error	0.1918
1 / Choice Sets	0.1667



## Generic Chair Attributes

n	Variable		Variance	DF	Standard Error
	Name	Label			
1	x11	x1 1	0.19116	1	0.43722
2	x12	x1 2	0.21187	1	0.46029
3	x21	x2 1	0.19116	1	0.43722
4	x22	x2 2	0.21187	1	0.46029
5	x31	x3 1	0.19965	1	0.44683
6	x32	x3 2	0.20072	1	0.44801
7	x41	x4 1	0.19965	1	0.44683
8	x42	x4 2	0.20072	1	0.44801
9	x51	x5 1	0.18850	1	0.43417
10	x52	x5 2	0.21187	1	0.46029
				==	
				10	

## Generic Chair Attributes

Set	Design	Efficiency	Index	Prob	n	x1	x2	x3	x4	x5	f1	f2	f3	f4
1	13	5.21381	152	0.25	289	2	3	2	3	2	1	1	1	0
	13	5.21381	213	0.25	290	3	2	3	2	3	1	1	1	0
	13	5.21381	15	0.25	291	1	1	2	2	3	1	1	1	0
	13	5.21381	244	0.25	292	1	1	1	1	1	0	0	0	1
2	13	5.21381	154	0.25	293	2	3	3	1	1	1	1	1	0
	13	5.21381	15	0.25	294	1	1	2	2	3	1	1	1	0
	13	5.21381	197	0.25	295	3	2	1	3	2	1	1	1	0
	13	5.21381	244	0.25	296	1	1	1	1	1	0	0	0	1
3	13	5.21381	108	0.25	297	2	1	3	3	3	1	1	1	0
	13	5.21381	220	0.25	298	3	3	1	2	1	1	1	1	0
	13	5.21381	38	0.25	299	1	2	2	1	2	1	1	1	0
	13	5.21381	244	0.25	300	1	1	1	1	1	0	0	0	1
4	13	5.21381	121	0.25	301	2	2	2	2	1	1	1	1	0
	13	5.21381	182	0.25	302	3	1	3	1	2	1	1	1	0
	13	5.21381	63	0.25	303	1	3	1	3	3	1	1	1	0
	13	5.21381	244	0.25	304	1	1	1	1	1	0	0	0	1
5	13	5.21381	111	0.25	305	2	2	1	1	3	1	1	1	0
	13	5.21381	77	0.25	306	1	3	3	2	2	1	1	1	0
	13	5.21381	178	0.25	307	3	1	2	3	1	1	1	1	0
	13	5.21381	244	0.25	308	1	1	1	1	1	0	0	0	1
6	13	5.21381	228	0.25	309	3	3	2	1	3	1	1	1	0
	13	5.21381	52	0.25	310	1	2	3	3	1	1	1	1	0
	13	5.21381	86	0.25	311	2	1	1	2	2	1	1	1	0
	13	5.21381	244	0.25	312	1	1	1	1	1	0	0	0	1

```

Design:          Generic
Factors:         x1-x5
                  x1 x2 x3 x4 x5
Sets w Dup Alts: 0
Duplicate Sets:  0

```

---

When there are three alternatives, each alternative has 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  $\beta = \mathbf{0}$ . With other assumptions about  $\beta$ , typically the probabilities will not all be equal. Relative  $D$ -efficiency is less than 100 since there is a constant alternative. It must also be less than 100 since we have all three-level factors but four alternatives. The optimal  $D$ -efficiency is not the number of choice sets in designs like this, so relative  $D$ -efficiency is less than 100.

To use this design for analysis, you only need the variables `Set` and `x1-x5`. Since it is already in choice design format (one row per alternative), it does not need to be processed using the `%MktRoll` macro. Note that when you make designs with the `%ChoiceEff` macro, the `model` statement in PROC TRANSREG should match or be no more complicated than the `model` specification that generated the design:

```
model class(x1-x5);
```

A model with fewer degrees of freedom is safe, although the design is suboptimal. For example, if `x1-x5` are quantitative attributes, the following model is safe:

```
model identity(x1-x5);
```

However, specifying interactions, or using this design in a branded study and specifying alternative-specific effects like this could lead to quite a few inestimable parameters. The following statements illustrate:

```

* Bad idea for this design!!;
model class(x1-x5 x1*x2 x4*x5);

* Another bad idea for this design!!;
model class(brand)
      class(brand * x1 brand * x2 brand * x3 brand * x4 brand * x5);

```

## Generic Attributes, a Constant Alternative, and Choice Set Swapping

The `%ChoiceEff` macro can be used in a very different way. Instead of providing a candidate set of alternatives to swap in and out of the design, you can provide a candidate set of entire choice sets. For this particular example, swapping alternatives is almost certainly better (see page 579). 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 start by using the `%MktEx` macro to make a candidate design, with one run per choice set and one factor for each attribute of each alternative (just like we did in the vacation, fabric softener, and food examples). We then process the candidates from one row per choice set to one row per alternative per choice set using the `%MktRoll`

macro. The following steps make the candidate set of choice sets:

```
%mktex(3 ** 15, n=81 * 81, seed=522)

%mktkey(3 5)

data key;
  input (x1-x5) ($);
  datalines;
x1      x2      x3      x4      x5
x6      x7      x8      x9      x10
x11     x12     x13     x14     x15
.       .       .       .       .
;

%mktroll(design=randomized, key=key, out=rolled)

* Code the constant alternative;
data final;
  set rolled;
  if _alt_ = '4' then do; x1 = 1; x2 = 1; x3 = 1; x4 = 1; x5 = 1; end;
run;

proc print; by set; id set; where set in (1, 100, 1000, 5000, 6561); run;
```

The %MktKey macro produced the following data set, which we copied, pasted, and augmented to make the Key data set:

---

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

---

A few of the candidate choice sets are as follows:

---

Generic Chair Attributes						
Set	_Alt_	x1	x2	x3	x4	x5
1	1	2	1	1	2	3
	2	3	3	1	1	1
	3	2	2	1	1	3
	4	1	1	1	1	1

100	1	1	2	1	2	1
	2	2	3	3	2	1
	3	2	2	3	3	2
	4	1	1	1	1	1
1000	1	2	1	2	1	3
	2	2	1	2	1	2
	3	1	3	2	2	2
	4	1	1	1	1	1
5000	1	3	1	3	2	3
	2	3	3	3	3	2
	3	1	2	1	2	3
	4	1	1	1	1	1
6561	1	1	3	1	2	2
	2	3	2	2	2	2
	3	1	3	3	1	3
	4	1	1	1	1	1

---

Next, we run the `%ChoiceEff` macro, only this time we 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. The option `nalts=` also specifies that the candidate set consists of candidate choice sets, whereas `flags=` specifies that the candidate set consists of candidate alternatives. We ask for fewer iterations since the candidate set is large. The following steps generate and evaluate the design:

```
%choicereff(data=final, /* candidate set of choice sets */
  model=class(x1-x5 / sta), /* model with stdzd orthogonal coding */
  nsets=6, /* number of choice sets */
  nalts=4, /* number of alternatives */
  maxiter=10, /* maximum number of designs to make */
  seed=109, /* random number seed */
  options=relative, /* display relative D-efficiency */
  beta=zero) /* assumed beta vector, Ho: b=0 */

%mktdups(generic, data=best, nalts=4, factors=x1-x5)
```

The results are as follows:

---

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	2.78600 *	0.35894
	1	4.40738 *	0.22689
	2	4.53259 *	0.22062
	3	4.53259	0.22062

.  
.  
.

Design	Iteration	D-Efficiency	D-Error
-----			
5	0	2.75184	0.36339
	1	4.34618	0.23009
	2	4.47415	0.22351
	3	4.67101 *	0.21409
	4	4.69946 *	0.21279
	5	4.69946	0.21279

.  
.  
.

Design	Iteration	D-Efficiency	D-Error
-----			
10	0	2.80475	0.35654
	1	4.26387	0.23453
	2	4.40048	0.22725
	3	4.51659	0.22141
	4	4.51659	0.22141

## Generic Chair Attributes

## Final Results

Design	5
Choice Sets	6
Alternatives	4
Parameters	10
Maximum Parameters	18
D-Efficiency	4.6995
Relative D-Eff	78.3243
D-Error	0.2128
1 / Choice Sets	0.1667

## Generic Chair Attributes

n	Variable		Variance	DF	Standard Error
	Name	Label			
1	x11	x1 1	0.19102	1	0.43705
2	x12	x1 2	0.37907	1	0.61568
3	x21	x2 1	0.24790	1	0.49790
4	x22	x2 2	0.27501	1	0.52441
5	x31	x3 1	0.19389	1	0.44033
6	x32	x3 2	0.23730	1	0.48713
7	x41	x4 1	0.22452	1	0.47384
8	x42	x4 2	0.21910	1	0.46809
9	x51	x5 1	0.21234	1	0.46081
10	x52	x5 2	0.24140	1	0.49133
				==	
				10	

Design: Generic  
 Factors: x1-x5  
           x1 x2 x3 x4 x5  
 Sets w Dup Alts: 0  
 Duplicate Sets: 0

---

This design is less  $D$ -efficient than we found using the alternative-swapping algorithm, so we will not use it.

## Design Algorithm Comparisons

It is instructive to compare the three approaches outlined in this chapter in the context of this problem. There are  $3^{3 \times 5} = 14,348,907$  choice sets for this problem (three-level factors and 3 alternatives times 5 factors per alternative). If we were to use the linear arrangement approach using the `%MktEx` macro, we could never begin to consider all possible candidate choice sets. Similarly, with the choice-set-swapping algorithm of the `%ChoiceEff` macro, we could never begin to consider all possible candidate choice sets. Furthermore, with the linear arrangement approach, we could not create a design with six choice sets since the minimum size is  $2 \times 15 + 1 = 31$ . Now consider the alternative-swapping algorithm. It uses at most a candidate set with only 244 observations ( $3^5 + 1$ ). From it, every possible choice set can potentially be constructed, although the macro only considers a tiny fraction of the possibilities. Hence, alternative swapping usually finds a better design, because the candidate set does not limit it.

Both uses of the `%ChoiceEff` macro have the advantage that they are explicitly minimizing the variances of the parameter estimates given a model and a  $\beta$  vector. They can be used to produce smaller, more specialized, and better designs. However, if the  $\beta$  vector or model is badly misspecified, the designs could be horrible. How badly do things have to be misspecified before you will have problems? Who knows. More research is needed. In contrast, the linear model `%MktEx` approach is very conservative and safe in that it should let you specify a very general model and still produce estimable parameters. The cost is you might 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 arrangement approach.

# Initial Designs

This section illustrates some design strategies that involve improving on or augmenting initial designs. We will not actually use any designs from this section.

## Improving an Existing Design

Sometimes, it is useful to try to improve an existing design. In this example, we use the `%MktEx` macro to create a design in 80 runs for 25 four-level factors. In the next step, we specify `init=`, and the macro goes straight into the design refinement history seeking to refine the input design. You might want to do this, for example, whenever you have a good, but not 100% *D*-efficient design, and you are willing to wait a few minutes to see if the macro can make it any better. The following steps generate the design:

```
title 'Try to Improve an Existing Design';

%mktx(4 ** 25, n=80, seed=368)

%mktx(4 ** 25,                /* 25 four-level factors          */
      n=80,                   /* number of runs                */
      init=design,            /* initial design                 */
      seed=306,              /* random number seed            */
      maxtime=20)           /* maximum time in minutes to work */
```

The *D*-efficiency of the final design from the first step is as follows:

---

Try to Improve an Existing Design				
The OPTEX Procedure				
Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
-----				
1	91.4106	83.9583	97.6073	0.9747

---

This is a large problem—one in which the `maxtime=` option might cause the macro to stop before it reaches the maximum number of iterations. Running a second refinement step might help improve the design by adding a few more iterations.



The results from the second step are as follows:

---

Design Refinement History				
Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
0	Initial	91.4106	91.4106	Ini
1	Start	90.0771		Pre,Mut,Ann
1	End	91.3476		
2	Start	88.8927		Pre,Mut,Ann
2	36 12	91.4181	91.4181	
2	56 6	91.4285	91.4285	
2	7 17	91.4372	91.4372	
2	13 10	91.4373	91.4373	
2	23 18	91.4404	91.4404	
2	17 16	91.4445	91.4445	
2	34 6	91.4572	91.4572	
2	56 19	91.4673	91.4673	
2	56 21	91.4768	91.4768	
2	77 1	91.4821	91.4821	
2	23 18	91.4827	91.4827	
2	48 3	91.4848	91.4848	
2	48 9	91.4863	91.4863	
2	40 18	91.4863	91.4863	
2	End	91.4863		
.				
.				
.				
6	Start	90.2194		Pre,Mut,Ann
6	63 19	91.5811	91.5811	
6	68 18	91.5835	91.5835	
6	End	91.5751		

7	Start	89.4607		Pre,Mut,Ann
7	25 4	91.5851	91.5851	
7	34 7	91.5902	91.5902	
7	47 2	91.5913	91.5913	
7	48 14	91.5930	91.5930	
7	56 4	91.5955	91.5955	
7	56 15	91.5999	91.5999	
7	60 6	91.6142	91.6142	
7	68 7	91.6172	91.6172	
7	78 5	91.6172	91.6172	
7	13 21	91.6249	91.6249	
7	18 19	91.6249	91.6249	
7	43 10	91.6249	91.6249	
7	48 14	91.6282	91.6282	
7	50 22	91.6408	91.6408	
7	61 4	91.6417	91.6417	
7	80 15	91.6430	91.6430	
7	46 12	91.6430	91.6430	
7	48 6	91.6430	91.6430	
7	End	91.6430		
.				
.				
.				
10	Start	89.8707		Pre,Mut,Ann
10	End	91.6629		
.				
.				
.				

Try to Improve an Existing Design

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	91.7082	83.9853	97.5951	0.9747

The macro skips the normal first steps, algorithm search and design search, and goes straight into the design refinement search. In this example a small improvement is found, although often, no improvement is found.

## When Some Choice Sets are Fixed in Advance

Sometimes certain runs or choice sets are fixed in advance and must be included in the design. The `%MktEx` macro can be used to efficiently augment a starting design with other choice sets. Suppose that you can make a choice design from the  $L_{36}$  ( $2^{11}3^{12}$ ). In addition, you want to optimally add four more choice sets to use as holdouts. First we look at how to do this using the `fixed=` option. This option can be used for fairly general design augmentation and refinement problems. On page 587, we see an easier way to handle this particular problem using the `holdouts=` option.

You can create the design in 36 runs as before. Next, a DATA step is used to add a flag variable `f` that has values of 1 for the original 36 runs. In addition, four more runs are added (just copies of the last run) but with a flag value of missing. When this variable is specified in the `fixed=f` option, it indicates that the first 36 runs of the `init=init` design are *fixed*—they must not change. The remaining 4 runs are to be randomly initialized and optimally refined to maximize the  $D$ -efficiency of the overall 40-run design. We specify `options=nosort` so that the additional runs stay at the end of the design. The following steps generate and display the design:

```

title 'Augment a Design';

%mktex(n=36, seed=292)

data init;
  set randomized end = eof;
  f = 1;
  output;
  if eof then do;
    f = .;
    do i = 1 to 4; output; end;
    drop i;
    end;
  run;

proc print; run;

%mktex(2 ** 11 3 ** 12,          /* 11 2-level and 12 3-level factors */
      n=40,                    /* 40 runs in final design          */
      init=init,               /* initial design                   */
      fixed=f,                 /* f var flags runs that cannot change */
      seed=513,                /* random number seed              */
      options=nosort)          /* do not sort output design        */

proc print; run;

```

The initial design is as follows:

Augment a Design

0	x																			f				
b	x	x	x	x	x	x	x	x	x	x	1	1	1	1	1	1	1	1	1	1	2	2	2	2
s	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	f
1	2	1	2	2	2	2	1	1	1	2	1	2	3	1	1	1	2	3	2	3	1	3	2	1
2	2	1	1	2	2	1	2	1	2	1	2	3	1	2	1	3	2	3	1	2	2	3	1	1
3	2	2	1	1	1	1	1	1	1	2	2	1	2	3	1	1	2	2	3	2	3	3	3	1
4	1	1	1	1	2	1	1	2	2	2	1	1	2	3	2	3	2	3	2	1	2	1	2	1
5	1	2	1	2	2	2	1	2	1	1	2	1	1	2	2	2	1	3	3	3	3	3	2	1
6	2	1	1	2	2	1	2	1	2	1	2	1	2	1	3	2	1	2	2	3	1	1	3	1
7	1	2	2	1	2	1	2	1	1	1	1	3	3	2	2	1	1	2	2	1	2	3	3	1
8	2	2	1	1	1	1	1	1	1	2	2	2	3	2	3	3	1	1	1	3	2	1	2	1
9	2	2	2	2	1	1	1	2	2	1	1	2	2	2	3	3	3	3	3	1	1	3	3	1
10	2	2	2	2	1	1	1	2	2	1	1	1	1	3	1	1	1	1	2	3	2	2	1	1
11	1	1	2	1	1	2	1	1	2	1	2	3	2	2	2	1	2	1	3	3	1	1	1	1
12	2	2	2	1	2	2	2	2	2	2	2	1	3	3	2	3	3	2	1	3	1	3	1	1
13	1	2	1	2	2	2	1	2	1	1	2	3	3	3	3	3	2	1	2	2	1	2	3	1
14	1	1	2	1	1	2	1	1	2	1	2	1	3	1	1	3	1	3	1	1	3	2	3	1
15	2	1	1	1	1	2	2	2	1	1	1	1	3	2	3	1	3	3	2	2	3	1	1	1
16	1	2	2	1	2	1	2	1	1	1	1	2	2	3	3	2	2	3	1	3	3	2	1	1
17	2	1	1	1	1	2	2	2	1	1	1	3	2	3	1	2	1	1	1	1	1	3	2	1
18	1	1	1	1	2	1	1	2	2	2	1	3	1	1	3	1	3	1	1	3	3	3	3	1
19	2	1	2	2	2	2	1	1	1	2	1	1	2	2	2	2	3	1	1	2	2	2	3	1
20	2	1	1	1	1	2	2	2	1	1	1	2	1	1	2	3	2	2	3	3	2	2	3	1
21	1	2	1	2	1	2	2	1	2	2	1	1	3	1	3	2	2	1	3	1	2	3	1	1
22	1	1	2	1	1	2	1	1	2	1	2	2	1	3	3	2	3	2	2	2	2	3	2	1
23	2	2	2	1	2	2	2	2	2	2	2	3	2	1	3	1	1	3	3	2	2	2	2	1
24	1	2	2	1	2	1	2	1	1	1	1	1	1	1	1	3	3	1	3	2	1	1	2	1
25	1	1	2	2	1	1	2	2	1	2	2	1	1	2	3	1	2	2	1	1	1	2	2	1
26	1	1	2	2	1	1	2	2	1	2	2	2	2	1	2	3	1	1	2	2	3	3	1	1
27	1	1	2	2	1	1	2	2	1	2	2	3	3	3	1	2	3	3	3	3	2	1	3	1
28	1	2	1	2	2	2	1	2	1	1	2	2	2	1	1	1	3	2	1	1	2	1	1	1
29	2	1	2	2	2	2	1	1	1	2	1	3	1	3	3	3	1	2	3	1	3	1	1	1
30	2	2	1	1	1	1	1	1	1	2	2	3	1	1	2	2	3	3	2	1	1	2	1	1
31	1	2	1	2	1	2	2	1	2	2	1	2	1	3	2	1	1	3	1	2	1	1	3	1
32	2	1	1	2	2	1	2	1	2	1	2	2	3	3	2	1	3	1	3	1	3	2	2	1
33	2	2	2	2	1	1	1	2	2	1	1	3	3	1	2	2	2	2	1	2	3	1	2	1
34	1	2	1	2	1	2	2	1	2	2	1	3	2	2	1	3	3	2	2	3	3	2	2	1
35	2	2	2	1	2	2	2	2	2	2	2	2	1	2	1	2	2	1	2	1	3	1	3	1
36	1	1	1	1	2	1	1	2	2	2	1	2	3	2	1	2	1	2	3	2	1	2	1	1
37	1	1	1	1	2	1	1	2	2	2	1	2	3	2	1	2	1	2	3	2	1	2	1	.
38	1	1	1	1	2	1	1	2	2	2	1	2	3	2	1	2	1	2	3	2	1	2	1	.
39	1	1	1	1	2	1	1	2	2	2	1	2	3	2	1	2	1	2	3	2	1	2	1	.
40	1	1	1	1	2	1	1	2	2	2	1	2	3	2	1	2	1	2	3	2	1	2	1	.

The iteration history for the augmentation is as follows:

---

Augment a Design				
Design Refinement History				
Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
0	Initial	97.0559	97.0559	Ini
1	Start	97.0788	97.0788	Pre,Mut,Ann
1	37 5	97.0805	97.0805	
1	37 8	97.0816	97.0816	
1	37 9	97.1049	97.1049	
1	37 11	97.1133	97.1133	
1	37 13	97.1177	97.1177	
1	38 1	97.1410	97.1410	
1	38 2	97.1605	97.1605	
1	38 3	97.1729	97.1729	
1	38 4	97.1822	97.1822	
1	38 6	97.1905	97.1905	
1	38 8	97.1944	97.1944	
1	39 2	97.1991	97.1991	
1	39 13	97.2007	97.2007	
1	39 19	97.2007	97.2007	
1	40 9	97.2007	97.2007	
1	40 10	97.2007	97.2007	
1	37 18	97.2023	97.2023	
1	37 3	97.2028	97.2028	
1	37 4	97.2028	97.2028	
1	37 23	97.2043	97.2043	
1	End	97.2043		
2	Start	97.2043	97.2043	Pre,Mut,Ann
2	40 21	97.2043	97.2043	
2	38 2	97.2043	97.2043	
2	40 9	97.2043	97.2043	
2	40 21	97.2043	97.2043	
2	End	97.2043		
3	Start	97.2043	97.2043	Pre,Mut,Ann
3	End	97.2043		
4	Start	97.2002		Pre,Mut,Ann
4	39 12	97.2043	97.2043	
4	39 23	97.2043	97.2043	
4	39 16	97.2043	97.2043	
4	End	97.2043		

5	Start	97.2043	97.2043	Pre,Mut,Ann
5	37 3	97.2043	97.2043	
5	37 15	97.2043	97.2043	
5	End	97.2043		
6	Start	97.2043	97.2043	Pre,Mut,Ann
6	40 1	97.2043	97.2043	
6	39 16	97.2043	97.2043	
6	38 16	97.2043	97.2043	
6	End	97.2043		

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

---

Notice that the macro goes straight into the design refinement stage. Also notice that in the iteration history, only rows 37 through 40 are changed. The design is as follows:

---

Augment a Design

0											x	x	x	x	x	x	x	x	x	x	x	x	x	x	
b	x	x	x	x	x	x	x	x	x	1	1	1	1	1	1	1	1	1	1	2	2	2	2		
s	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	f	
1	2	1	2	2	2	2	1	1	1	2	1	2	3	1	1	1	2	3	2	3	1	3	2	1	
2	2	1	1	2	2	1	2	1	2	1	2	3	1	2	1	3	2	3	1	2	2	3	1	1	
3	2	2	1	1	1	1	1	1	1	2	2	1	2	3	1	1	2	2	3	2	3	3	3	1	
4	1	1	1	1	2	1	1	2	2	2	1	1	2	3	2	3	2	3	2	1	2	1	2	1	
5	1	2	1	2	2	2	1	2	1	1	2	1	1	2	2	2	1	3	3	3	3	3	2	1	
6	2	1	1	2	2	1	2	1	2	1	2	1	2	1	3	2	1	2	2	3	1	1	3	1	
7	1	2	2	1	2	1	2	1	1	1	1	3	3	2	2	1	1	2	2	1	2	3	3	1	
8	2	2	1	1	1	1	1	1	1	2	2	2	3	2	3	3	1	1	1	3	2	1	2	1	
9	2	2	2	2	1	1	1	2	2	1	1	2	2	2	3	3	3	3	3	1	1	3	3	1	
10	2	2	2	2	1	1	1	2	2	1	1	1	1	3	1	1	1	1	2	3	2	2	1	1	
11	1	1	2	1	1	2	1	1	2	1	2	3	2	2	2	1	2	1	3	3	1	1	1	1	
12	2	2	2	1	2	2	2	2	2	2	2	1	3	3	2	3	3	2	1	3	1	3	1	1	
13	1	2	1	2	2	2	1	2	1	1	2	3	3	3	3	3	2	1	2	2	1	2	3	1	
14	1	1	2	1	1	2	1	1	2	1	2	1	3	1	1	3	1	3	1	1	3	2	3	1	
15	2	1	1	1	1	2	2	2	1	1	1	1	3	2	3	1	3	3	2	2	3	1	1	1	
16	1	2	2	1	2	1	2	1	1	1	1	2	2	3	3	2	2	3	1	3	3	2	1	1	
17	2	1	1	1	1	2	2	2	1	1	1	3	2	3	1	2	1	1	1	1	1	3	2	1	
18	1	1	1	1	2	1	1	2	2	2	1	3	1	1	3	1	3	1	1	3	3	3	3	1	
19	2	1	2	2	2	2	1	1	1	2	1	1	2	2	2	2	3	1	1	2	2	2	3	1	
20	2	1	1	1	1	2	2	2	1	1	1	2	1	1	2	3	2	2	3	3	2	2	3	1	

```

21 1 2 1 2 1 2 2 1 2 2 1 1 3 1 3 2 2 1 3 1 2 3 1 1
22 1 1 2 1 1 2 1 1 2 1 2 2 1 3 3 2 3 2 2 2 2 3 2 1
23 2 2 2 1 2 2 2 2 2 2 2 3 2 1 3 1 1 3 3 2 2 2 2 1
24 1 2 2 1 2 1 2 1 1 1 1 1 1 1 1 3 3 1 3 2 1 1 2 1
25 1 1 2 2 1 1 2 2 1 2 2 1 1 2 3 1 2 2 1 1 1 2 2 1
26 1 1 2 2 1 1 2 2 1 2 2 2 2 1 2 3 1 1 2 2 3 3 1 1
27 1 1 2 2 1 1 2 2 1 2 2 3 3 3 1 2 3 3 3 3 2 1 3 1
28 1 2 1 2 2 2 1 2 1 1 2 2 2 1 1 1 3 2 1 1 2 1 1 1
29 2 1 2 2 2 2 1 1 1 2 1 3 1 3 3 3 1 2 3 1 3 1 1 1
30 2 2 1 1 1 1 1 1 1 2 2 3 1 1 2 2 3 3 2 1 1 2 1 1
31 1 2 1 2 1 2 2 1 2 2 1 2 1 3 2 1 1 3 1 2 1 1 3 1
32 2 1 1 2 2 1 2 1 2 1 2 2 3 3 2 1 3 1 3 1 3 2 2 1
33 2 2 2 2 1 1 1 2 2 1 1 3 3 1 2 2 2 2 1 2 3 1 2 1
34 1 2 1 2 1 2 2 1 2 2 1 3 2 2 1 3 3 2 2 3 3 2 2 1
35 2 2 2 1 2 2 2 2 2 2 2 2 1 2 1 2 2 1 2 1 3 1 3 1
36 1 1 1 1 2 1 1 2 2 2 1 2 3 2 1 2 1 2 3 2 1 2 1 1
37 1 2 2 2 2 1 1 2 1 1 1 3 2 3 3 2 1 2 1 3 1 3 1 .
38 2 2 2 1 2 1 2 2 1 1 1 2 2 1 2 3 3 3 1 1 3 2 3 .
39 1 2 2 2 2 2 2 1 2 2 2 3 3 3 1 2 1 3 2 1 3 2 3 .
40 2 1 1 1 1 1 1 2 2 2 1 2 1 2 3 2 1 3 2 1 1 2 1 .

```

---

The last four rows are the holdouts.

The following steps do the same thing only using the `holdouts=4` option instead:

```

title 'Augment a Design';

%mktx(n=36, seed=292)

%mktx(2 ** 11 3 ** 12,          /* 11 2-level and 12 3-level factors */
      n=40,                    /* 40 runs in final design          */
      init=randomized,        /* initial design with 36 runs     */
      holdouts=4,             /* add four holdouts               */
      seed=513,               /* random number seed              */
      options=nosort)        /* do not sort output design       */

proc print data=design(firstobs=37); run;

```

The holdout observations, which are the same as we saw previously, are as follows:

---

 Augment a Design

```

0          x x x x x x x x x x x x x x x x x x
b  x  x  x  x  x  x  x  x  x  x  1  1  1  1  1  1  1  1  1  1  2  2  2  2
s  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5  6  7  8  9  0  1  2  3  w

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

```

---

The `%MktEx` macro provides another way to use initial designs. The initial design can indicate that part of the design is fixed and cannot change and a different part should be randomly initialized and can change. The initial design can have three types of values:

- positive integers are fixed and constant and do not change throughout the course of the iterations.
- zero and missing values are replaced by random values at the start of each new design search and can change throughout the course of the iterations.
- negative values are replaced by their absolute value at the start of each new design attempt and can change throughout the course of the iterations.

Returning to the example of making the design  $4^{25}$  in 80 runs, we could do it in two steps. The maximum number of four-level factors in 80 runs is 11. If it is important that some factors be orthogonal, we could first make an orthogonal array with 11 four-level factors and then append 14 more nonorthogonal-factors. The following steps create the design:

```

title 'Differential Design Initialization';

%mktext(4 ** 11, n=80)

data init;
  set design;
  retain x12-x25 .;
  run;

%mktext(4 ** 25,          /* 25 four-level factors          */
        n=80,           /* 80 runs                    */
        init=init,      /* initial design              */
        seed=472)       /* random number seed         */

%mkteval;

```

The initial design consists of the orthogonal array with 11 columns followed by 14 more columns that are all missing. When `%MktEx` sees missing values in the initial design, it holds all the nonmissing values fixed. Then it randomly replaces the missing values and uses the coordinate-exchange algorithm to refine the last columns. The final design is slightly less  $D$ -efficient than we saw previously, but the



first 11 columns are orthogonal. The remaining columns are all slightly correlated with themselves and with the first columns. The final efficiency table is as follows:

---

Differential Design Initialization				
Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	89.4216	78.9807	97.7767	0.9747

---

The canonical correlations are as follows:

---

Differential Design Initialization													
Canonical Correlations Between the Factors													
There are 0 Canonical Correlations Greater Than 0.316													
	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13
x1	1	0	0	0	0	0	0	0	0	0	0	0.15	0.17
x2	0	1	0	0	0	0	0	0	0	0	0	0.17	0.11
x3	0	0	1	0	0	0	0	0	0	0	0	0.20	0.13
x4	0	0	0	1	0	0	0	0	0	0	0	0.16	0.12
x5	0	0	0	0	1	0	0	0	0	0	0	0.16	0.18
x6	0	0	0	0	0	1	0	0	0	0	0	0.12	0.12
x7	0	0	0	0	0	0	1	0	0	0	0	0.18	0.18
x8	0	0	0	0	0	0	0	1	0	0	0	0.16	0.12
x9	0	0	0	0	0	0	0	0	1	0	0	0.15	0.14
x10	0	0	0	0	0	0	0	0	0	1	0	0.12	0.18
x11	0	0	0	0	0	0	0	0	0	0	1	0.16	0.12
x12	0.15	0.17	0.20	0.16	0.16	0.12	0.18	0.16	0.15	0.12	0.16	1	0.10
x13	0.17	0.11	0.13	0.12	0.18	0.12	0.18	0.12	0.14	0.18	0.12	0.10	1
x14	0.18	0.14	0.10	0.14	0.19	0.20	0.18	0.19	0.14	0.15	0.11	0.16	0.15
x15	0.16	0.25	0.15	0.18	0.17	0.17	0.15	0.12	0.24	0.13	0.16	0.14	0.13
x16	0.10	0.23	0.14	0.15	0.17	0.16	0.15	0.13	0.10	0.16	0.21	0.12	0.11
x17	0.20	0.20	0.15	0.07	0.12	0.17	0.16	0.12	0.13	0.20	0.14	0.12	0.11
x18	0.09	0.18	0.10	0.16	0.15	0.13	0.13	0.17	0.10	0.15	0.14	0.15	0.09
x19	0.17	0.18	0.17	0.16	0.13	0.17	0.19	0.21	0.15	0.15	0.16	0.14	0.15
x20	0.23	0.09	0.11	0.14	0.17	0.21	0.25	0.27	0.06	0.17	0.15	0.14	0.11
x21	0.12	0.09	0.15	0.17	0.17	0.15	0.20	0.14	0.20	0.17	0.16	0.18	0.12
x22	0.17	0.14	0.07	0.18	0.15	0.12	0.11	0.16	0.17	0.09	0.17	0.12	0.16
x23	0.19	0.15	0.15	0.18	0.09	0.10	0.16	0.16	0.12	0.16	0.16	0.13	0.12
x24	0.18	0.16	0.14	0.20	0.09	0.22	0.13	0.13	0.18	0.13	0.18	0.14	0.11
x25	0.14	0.18	0.09	0.13	0.19	0.18	0.17	0.19	0.16	0.20	0.14	0.15	0.12

	x14	x15	x16	x17	x18	x19	x20	x21	x22	x23	x24	x25
x1	0.18	0.16	0.10	0.20	0.09	0.17	0.23	0.12	0.17	0.19	0.18	0.14
x2	0.14	0.25	0.23	0.20	0.18	0.18	0.09	0.09	0.14	0.15	0.16	0.18
x3	0.10	0.15	0.14	0.15	0.10	0.17	0.11	0.15	0.07	0.15	0.14	0.09
x4	0.14	0.18	0.15	0.07	0.16	0.16	0.14	0.17	0.18	0.18	0.20	0.13
x5	0.19	0.17	0.17	0.12	0.15	0.13	0.17	0.17	0.15	0.09	0.09	0.19
x6	0.20	0.17	0.16	0.17	0.13	0.17	0.21	0.15	0.12	0.10	0.22	0.18
x7	0.18	0.15	0.15	0.16	0.13	0.19	0.25	0.20	0.11	0.16	0.13	0.17
x8	0.19	0.12	0.13	0.12	0.17	0.21	0.27	0.14	0.16	0.16	0.13	0.19
x9	0.14	0.24	0.10	0.13	0.10	0.15	0.06	0.20	0.17	0.12	0.18	0.16
x10	0.15	0.13	0.16	0.20	0.15	0.15	0.17	0.17	0.09	0.16	0.13	0.20
x11	0.11	0.16	0.21	0.14	0.14	0.16	0.15	0.16	0.17	0.16	0.18	0.14
x12	0.16	0.14	0.12	0.12	0.15	0.14	0.14	0.18	0.12	0.13	0.14	0.15
x13	0.15	0.13	0.11	0.11	0.09	0.15	0.11	0.12	0.16	0.12	0.11	0.12
x14	1	0.22	0.14	0.12	0.12	0.21	0.13	0.13	0.13	0.20	0.18	0.12
x15	0.22	1	0.12	0.09	0.19	0.09	0.09	0.13	0.10	0.15	0.16	0.13
x16	0.14	0.12	1	0.15	0.14	0.14	0.18	0.13	0.12	0.10	0.17	0.18
x17	0.12	0.09	0.15	1	0.09	0.16	0.18	0.09	0.15	0.12	0.18	0.12
x18	0.12	0.19	0.14	0.09	1	0.14	0.13	0.20	0.15	0.13	0.14	0.17
x19	0.21	0.09	0.14	0.16	0.14	1	0.13	0.11	0.10	0.11	0.16	0.12
x20	0.13	0.09	0.18	0.18	0.13	0.13	1	0.10	0.11	0.11	0.18	0.11
x21	0.13	0.13	0.13	0.09	0.20	0.11	0.10	1	0.21	0.15	0.23	0.09
x22	0.13	0.10	0.12	0.15	0.15	0.10	0.11	0.21	1	0.19	0.18	0.22
x23	0.20	0.15	0.10	0.12	0.13	0.11	0.11	0.15	0.19	1	0.13	0.13
x24	0.18	0.16	0.17	0.18	0.14	0.16	0.18	0.23	0.18	0.13	1	0.12
x25	0.12	0.13	0.18	0.12	0.17	0.12	0.11	0.09	0.22	0.13	0.12	1

---

The one-way frequencies are as follows:

---

Differential Design Initialization

Summary of Frequencies

There are 0 Canonical Correlations Greater Than 0.316

\* - Indicates Unequal Frequencies

Frequencies

x1	20	20	20	20
x2	20	20	20	20
x3	20	20	20	20
x4	20	20	20	20
x5	20	20	20	20
x6	20	20	20	20
x7	20	20	20	20
x8	20	20	20	20
x9	20	20	20	20
x10	20	20	20	20
x11	20	20	20	20

```

*   x12      19 22 19 20
*   x13      21 19 20 20
*   x14      18 20 21 21
*   x15      18 21 22 19
*   x16      19 18 22 21
*   x17      21 19 18 22
*   x18      21 20 20 19
*   x19      21 18 18 23
*   x20      20 20 21 19
*   x21      18 20 23 19
*   x22      20 18 22 20
*   x23      19 20 21 20
*   x24      23 17 20 20
*   x25      21 19 19 21

```

---

We could run one more refinement on this design and force  $x_{12}$ - $x_{25}$  to be balanced. We need to make a new initial design using our current design. This time, we make  $x_{12}$ - $x_{25}$  negative. Then  $x_1$ - $x_{11}$  do not change, the absolute values of  $x_{12}$ - $x_{25}$  are used as initial values, but  $x_{12}$ - $x_{25}$  are still be allowed to change. Then we can use the `balance=1` option with `%MktEx` to make a design that is better balanced. Usually, when you are creating a design, you should specify `mintry=` with `balance=`, however, since we are refining not creating a design it is not necessary. The following steps create the design:

```

data init(drop=j);
  set design;
  array x[25];
  do j = 12 to 25; x[j] = -x[j]; end;
run;

%mktx(4 ** 25,          /* 25 four-level factors      */
      n=80,            /* 80 runs                      */
      init=init,       /* initial design                */
      seed=472,        /* random number seed           */
      balance=1)       /* require near perfect balance */

%mkteval;

```

The final  $D$ -efficiency, which again, is a bit lower than we saw previously, is as follows:

---

#### Differential Design Initialization

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	86.1917	71.1301	97.9379	0.9747

---

The canonical correlations are as follows:

---

Differential Design Initialization  
 Canonical Correlations Between the Factors  
 There are 0 Canonical Correlations Greater Than 0.316

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13
x1	1	0	0	0	0	0	0	0	0	0	0	0.20	0.15
x2	0	1	0	0	0	0	0	0	0	0	0	0.15	0.13
x3	0	0	1	0	0	0	0	0	0	0	0	0.15	0.12
x4	0	0	0	1	0	0	0	0	0	0	0	0.20	0.15
x5	0	0	0	0	1	0	0	0	0	0	0	0.20	0.24
x6	0	0	0	0	0	1	0	0	0	0	0	0.13	0.15
x7	0	0	0	0	0	0	1	0	0	0	0	0.17	0.19
x8	0	0	0	0	0	0	0	1	0	0	0	0.13	0.15
x9	0	0	0	0	0	0	0	0	1	0	0	0.12	0.16
x10	0	0	0	0	0	0	0	0	0	1	0	0.13	0.17
x11	0	0	0	0	0	0	0	0	0	0	1	0.20	0.13
x12	0.20	0.15	0.15	0.20	0.20	0.13	0.17	0.13	0.12	0.13	0.20	1	0.14
x13	0.15	0.13	0.12	0.15	0.24	0.15	0.19	0.15	0.16	0.17	0.13	0.14	1
x14	0.20	0.10	0.09	0.13	0.18	0.22	0.17	0.16	0.13	0.16	0.09	0.20	0.15
x15	0.21	0.23	0.22	0.25	0.20	0.24	0.17	0.15	0.22	0.13	0.22	0.18	0.16
x16	0.09	0.20	0.17	0.18	0.12	0.18	0.17	0.17	0.10	0.16	0.20	0.09	0.14
x17	0.20	0.23	0.15	0.15	0.09	0.22	0.20	0.10	0.15	0.17	0.15	0.15	0.15
x18	0.10	0.23	0.10	0.16	0.15	0.15	0.18	0.17	0.10	0.15	0.15	0.15	0.12
x19	0.17	0.20	0.18	0.14	0.17	0.15	0.23	0.16	0.12	0.19	0.17	0.16	0.25
x20	0.26	0.10	0.10	0.17	0.17	0.17	0.24	0.23	0.09	0.17	0.19	0.12	0.10
x21	0.20	0.13	0.20	0.19	0.19	0.19	0.20	0.20	0.15	0.14	0.17	0.14	0.14
x22	0.17	0.16	0.10	0.23	0.14	0.16	0.10	0.16	0.15	0.10	0.14	0.15	0.15
x23	0.24	0.18	0.14	0.18	0.09	0.10	0.15	0.20	0.13	0.22	0.19	0.19	0.12
x24	0.24	0.20	0.27	0.18	0.15	0.29	0.12	0.12	0.18	0.20	0.17	0.20	0.14
x25	0.20	0.16	0.09	0.15	0.23	0.19	0.17	0.18	0.17	0.22	0.15	0.16	0.12

	x14	x15	x16	x17	x18	x19	x20	x21	x22	x23	x24	x25
x1	0.20	0.21	0.09	0.20	0.10	0.17	0.26	0.20	0.17	0.24	0.24	0.20
x2	0.10	0.23	0.20	0.23	0.23	0.20	0.10	0.13	0.16	0.18	0.20	0.16
x3	0.09	0.22	0.17	0.15	0.10	0.18	0.10	0.20	0.10	0.14	0.27	0.09
x4	0.13	0.25	0.18	0.15	0.16	0.14	0.17	0.19	0.23	0.18	0.18	0.15
x5	0.18	0.20	0.12	0.09	0.15	0.17	0.17	0.19	0.14	0.09	0.15	0.23
x6	0.22	0.24	0.18	0.22	0.15	0.15	0.17	0.19	0.16	0.10	0.29	0.19
x7	0.17	0.17	0.17	0.20	0.18	0.23	0.24	0.20	0.10	0.15	0.12	0.17
x8	0.16	0.15	0.17	0.10	0.17	0.16	0.23	0.20	0.16	0.20	0.12	0.18
x9	0.13	0.22	0.10	0.15	0.10	0.12	0.09	0.15	0.15	0.13	0.18	0.17
x10	0.16	0.13	0.16	0.17	0.15	0.19	0.17	0.14	0.10	0.22	0.20	0.22
x11	0.09	0.22	0.20	0.15	0.15	0.17	0.19	0.17	0.14	0.19	0.17	0.15
x12	0.20	0.18	0.09	0.15	0.15	0.16	0.12	0.14	0.15	0.19	0.20	0.16
x13	0.15	0.16	0.14	0.15	0.12	0.25	0.10	0.14	0.15	0.12	0.14	0.12
x14	1	0.20	0.20	0.12	0.18	0.24	0.10	0.25	0.14	0.18	0.17	0.13
x15	0.20	1	0.15	0.15	0.15	0.12	0.09	0.15	0.14	0.12	0.21	0.18
x16	0.20	0.15	1	0.15	0.15	0.12	0.14	0.13	0.09	0.15	0.23	0.17
x17	0.12	0.15	0.15	1	0.20	0.15	0.19	0.24	0.17	0.14	0.26	0.19
x18	0.18	0.15	0.15	0.20	1	0.17	0.17	0.22	0.15	0.09	0.17	0.22
x19	0.24	0.12	0.12	0.15	0.17	1	0.09	0.15	0.17	0.14	0.15	0.13
x20	0.10	0.09	0.14	0.19	0.17	0.09	1	0.12	0.20	0.18	0.10	0.09
x21	0.25	0.15	0.13	0.24	0.22	0.15	0.12	1	0.15	0.12	0.28	0.12
x22	0.14	0.14	0.09	0.17	0.15	0.17	0.20	0.15	1	0.22	0.17	0.27
x23	0.18	0.12	0.15	0.14	0.09	0.14	0.18	0.12	0.22	1	0.21	0.18
x24	0.17	0.21	0.23	0.26	0.17	0.15	0.10	0.28	0.17	0.21	1	0.14
x25	0.13	0.18	0.17	0.19	0.22	0.13	0.09	0.12	0.27	0.18	0.14	1

---

The one-way frequencies, which are now all perfect, are as follows:

---

 Differential Design Initialization

## Summary of Frequencies

There are 0 Canonical Correlations Greater Than 0.316

\* - Indicates Unequal Frequencies

	Frequencies			
x1	20	20	20	20
x2	20	20	20	20
x3	20	20	20	20
x4	20	20	20	20
x5	20	20	20	20
x6	20	20	20	20
x7	20	20	20	20
x8	20	20	20	20
x9	20	20	20	20
x10	20	20	20	20
x11	20	20	20	20
x12	20	20	20	20
x13	20	20	20	20
x14	20	20	20	20
x15	20	20	20	20
x16	20	20	20	20
x17	20	20	20	20
x18	20	20	20	20
x19	20	20	20	20
x20	20	20	20	20
x21	20	20	20	20
x22	20	20	20	20
x23	20	20	20	20
x24	20	20	20	20
x25	20	20	20	20

---

You can initialize any part of the design to positive integers (fixed), any other part to zero or missing (randomly initialize and change), and any other part to negative (do not reinitialize but change is allowed). This capability gives you very flexible control over the components of your design.

## Partial Profiles and Restrictions

Partial-profile designs (Chrzan and Elrod 1995) are used when there are many attributes but no more than a few of them are allowed to vary at any one time. Chrzan and Elrod show an example where respondents must choose between vacuum cleaners that vary along 20 different attributes: Brand, Price, Warranty, Horsepower, and so on. It is difficult for respondents to simultaneously evaluate that many attributes, so it is better if they are only exposed to a few at a time. Partial-profile designs have become very popular among some researchers.

This section serves two purposes. It shows ways in which partial-profile designs can be made. It also illustrates some of the important techniques available for restricting designs with the `%MktEx` macro. Partial-profile designs provide a nice context for illustrating the creation of highly-restricted designs.

### Pairwise Partial-Profile Choice Design

For example, a partial-profile design for 20 two-level factors, with 5 varying at a time, with the factors that are not shown displayed with an ordinary missing value is as follows:

---

```

. . . 2 . . . 1 1 1 . . . . 1 . . . . .
2 . . . 2 . . . . . 1 . . . . . 1 . . 1
2 . 1 . . . . 1 . . . . . 1 . . . . . 2
. . . . . . . 2 . . . . . 2 2 . . 2 2 .
. . . . 2 . . . . . 2 . 2 . . . 2 . 2 . .
. . . . . . . . 2 . . . . . 1 . . . 2 1 1
. . . . . . . . 1 . 2 . 2 . . . 2 . . 2
1 . . . . . 1 . . . . . . . 2 . 2 . . . 1
. . . . . . . 2 . . . . . 2 1 . 2 . . . . 1
. . 1 . 1 . . . . 2 . . . . . 1 . . . 1
1 . 2 . . . 2 . . . . . . . . 1 2 . .
. 1 . . . . . . . . 2 . 1 . 1 1 . . . .
2 . . . . . . . . 2 . . 2 . . . 1 2 . . .
. . 1 . . . . . . . . 2 2 . 1 . . . 1 .
. . . . 2 . 2 . . . . . . 2 . 1 . . 1 .
. 2 . 1 . . . 1 . . . . . . . . 2 . 2 .
2 2 . . . . 1 . 1 . . . 1 . . . . . .
. 2 . . . . . . . . 1 1 . . . . . 1 . 2
. 1 . 1 . 2 . 2 . . . . . . . . 1 . .
. . . 1 1 . . . . . 2 2 . 1 . . . . .
1 . . . 2 2 1 . . . . . . . 2 . . . .
. . . . . 1 1 2 . 1 . . . . . . 1 . . .
. . . 2 . . . . . . . . 1 2 1 . 1 . . .
. . . 1 . . . . 2 . . 1 . . 1 . . . . 2
. . 2 . . 1 . . 1 . 1 . . . 2 . . . .
. . 2 . 1 . . 2 . . . . . 1 . . 2 . . .
. . 2 . . . 1 1 . . 2 . . . . . . 1 . .
. 1 . . 1 . . 1 2 . . . 2 . . . . . .
1 . . . . . . . 1 . . . . 1 . . . . 1 2 .
2 . . 2 1 1 . . . . . . . . . . 2 . .

```

```

. . . . . 2 2 1 1 1 .
. 1 2 2 . . . . . 2 2
. . 1 1 . . 1 . . . . 2 . . . .
. . . . . 2 . . . . 2 . . 1 2 . . 1 . . .
. . . . . 2 . . 2 . . . 1 1 . 2 . . .
1 . . . . . 2 . 2 2 . . . . . 1 .
2 . 2 1 . . . . . 1 . . 2 . . . . .
. 1 1 . . . . . 1 . 1 . . . . 2 . . .
. 2 1 2 2 . . . 2 . . . . . . . .
. . . . . 2 2 . . . 2 . . . . 2 . . 2 .

```

A design like this could be used to make a binary choice experiment. For example, the second last run has factors 2, 3, 4, 5, and 9 varying. Assume they are all yes-no factors (1 yes, 2 no). Subjects could be offered a choice between these two profiles:

```

x2 = no,   x3 = yes,   x4 = no,   x5 = no,   x9 = no
x2 = yes,  x3 = no,   x4 = yes,  x5 = yes,  x9 = yes

```

The first profile came directly from the design and the second came from shifting the design: yes → no, and no → yes.

The following steps generated and displayed the partial-profile design:\*

```

title 'Partial Profiles';

%mktx(3 ** 20,           /* 20 three-level factors      */
      n=41,             /* 41 runs                      */
      partial=5,        /* partial profile, 5 attrs vary */
      seed=292,         /* random number seed           */
      maxdesigns=1)     /* make only one design         */

%mktxlab(data=randomized, values=. 1 2, nfill=99)

data _null_; set final(firstobs=2); put (x1-x20) (2.); run;

```

A  $3^{20}$  design is requested in 41 runs. The three levels are yes, no, and not shown. Forty-one runs gives us 40 partial profiles and one more run with all attributes not shown (all ones in the original design before reassigning levels). When we ask for partial profiles, in this case `partial=5`, we are imposing a constraint that the number of 2's and 3's in each run equals 5 and the number of 1's equals 15. This makes the sum of the coded variables constant in each run and hence introduces a linear dependency (the sum of the coded variables is proportional to the intercept). The way we avoid having the linear dependency is by adding this additional row where all attributes are set to the not-shown level. The sum of the coded variables for this row are different than the constant sum for the other rows and hence eliminate the linear dependency we would otherwise have.

\*Due to machine, SAS release, and macro differences, you might not get exactly the same design used in this book, but the differences should be slight.



The %MktLab macro reassigns the levels (1, 2, 3) to (., 1, 2) where “.” means not shown. Normally, the %MktLab macro complains about using missing values for levels, because missing values are used in the key= data set as fillers when some factors have more levels than others. Encountering missing levels normally indicates an error. We can permit missing levels by specifying nfill=99. Then the macro considers levels of 99 to be invalid, not missing. A DATA step displays the design excluding the constant (all not shown) first row.

The next steps take this design and turn it into a partial-profile choice design. It reads each profile in the design, and outputs it. If the level is not missing, the code changes 1 to 2 and 2 to 1 and outputs the new profile. The next step uses the %ChoiEff macro to evaluate the design. We specify zero=none for now to see exactly which parameters we can estimate and which ones we cannot. This usage of the %ChoiEff macro is similar to what we saw in the food product example on page 509. Our choice design is specified on the data= option and the same data set, with just the Set variable kept, is specified on the init= option. The number of choice sets, 40 (we drop the constant choice set), number of alternatives, 2, and assumed betas, a vector of zeros, are also specified. Zero internal iterations are requested since we want a design evaluation, not an attempt to improve the design. The following steps generate and evaluate the design:

```

data des(drop=i);
  Set = _n_;
  set final(firstobs=2);
  array x[20];
  output;
  do i = 1 to 20;
    if n(x[i]) then do; if x[i] = 1 then x[i] = 2; else x[i] = 1; end;
    end;
  output;
run;

%choicEff(data=des,                /* candidate set of choice sets      */
          init=des(keep=set),      /* select these sets from candidates */
          intiter=0,               /* evaluate without internal iterations */
          model=class(x1-x20 / zero=none), /* main effects, no ref levels      */
          nsets=40,                /* number of choice sets             */
          nalts=2,                 /* number of alternatives             */
          beta=zero)               /* assumed beta vector, Ho: b=0      */

%mktdups(generic, data=best, nalts=2, factors=x1-x20)

```

The last part of the output is as follows:

## Partial Profiles

n	Variable Name	Label	Variance	DF	Standard Error
1	x11	x1 1	0.46962	1	0.68529
2	x12	x1 2	.	0	.
3	x21	x2 1	0.52778	1	0.72648
4	x22	x2 2	.	0	.
5	x31	x3 1	0.42989	1	0.65566
6	x32	x3 2	.	0	.
7	x41	x4 1	0.46230	1	0.67993
8	x42	x4 2	.	0	.
9	x51	x5 1	0.54615	1	0.73902
10	x52	x5 2	.	0	.
11	x61	x6 1	0.81069	1	0.90038
12	x62	x6 2	.	0	.
13	x71	x7 1	0.50135	1	0.70806
14	x72	x7 2	.	0	.
15	x81	x8 1	0.49753	1	0.70536
16	x82	x8 2	.	0	.
17	x91	x9 1	0.48632	1	0.69737
18	x92	x9 2	.	0	.
19	x101	x10 1	0.54529	1	0.73844
20	x102	x10 2	.	0	.
21	x111	x11 1	0.56975	1	0.75482
22	x112	x11 2	.	0	.
23	x121	x12 1	0.54158	1	0.73592
24	x122	x12 2	.	0	.
25	x131	x13 1	0.54817	1	0.74039
26	x132	x13 2	.	0	.
27	x141	x14 1	0.55059	1	0.74201
28	x142	x14 2	.	0	.
29	x151	x15 1	0.52638	1	0.72552
30	x152	x15 2	.	0	.
31	x161	x16 1	0.44403	1	0.66636
32	x162	x16 2	.	0	.
33	x171	x17 1	0.57751	1	0.75994
34	x172	x17 2	.	0	.
35	x181	x18 1	0.56915	1	0.75442
36	x182	x18 2	.	0	.
37	x191	x19 1	0.58340	1	0.76381
38	x192	x19 2	.	0	.
39	x201	x20 1	0.54343	1	0.73718
40	x202	x20 2	.	0	.

==

20

```
Design:          Generic
Factors:         x1-x20
```

```
x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17 x18 x19 x20
Sets w Dup Alts: 0
Duplicate Sets:  0
```

---

We see that one parameter is estimable for each factor and that is the parameter for the 1 or yes level. We also see that there are no duplicates. The %ChoiceEff macro displays the following list of all redundant variables:

Redundant Variables:

```
x12 x22 x32 x42 x52 x62 x72 x82 x92 x102 x112 x122 x132 x142 x152 x162 x172 x182
x192 x202
```

We can copy and paste this list into our program and drop those terms as follows:

```
%choiceff(data=des,          /* candidate set of choice sets      */
           init=des(keep=set), /* select these sets from candidates */
           intiter=0,         /* evaluate without internal iterations */
           model=class(x1-x20 / zero=none), /* main effects, no ref levels */
           nsets=40,         /* number of choice sets */
           nalts=2,          /* number of alternatives */
           /* extra model terms to drop from model */
           drop=x12 x22 x32 x42 x52 x62 x72 x82 x92 x102
           x112 x122 x132 x142 x152 x162 x172 x182 x192 x202,
           beta=zero)         /* assumed beta vector, Ho: b=0      */
```

The last part of the output is as follows:

---

### Partial Profiles

#### Final Results

Design	1
Choice Sets	40
Alternatives	2
Parameters	20
Maximum Parameters	40
D-Efficiency	2.1648
D-Error	0.4619

## Partial Profiles

n	Variable		Variance	DF	Standard Error
	Name	Label			
1	x11	x1 1	0.46962	1	0.68529
2	x21	x2 1	0.52778	1	0.72648
3	x31	x3 1	0.42989	1	0.65566
4	x41	x4 1	0.46230	1	0.67993
5	x51	x5 1	0.54615	1	0.73902
6	x61	x6 1	0.81069	1	0.90038
7	x71	x7 1	0.50135	1	0.70806
8	x81	x8 1	0.49753	1	0.70536
9	x91	x9 1	0.48632	1	0.69737
10	x101	x10 1	0.54529	1	0.73844
11	x111	x11 1	0.56975	1	0.75482
12	x121	x12 1	0.54158	1	0.73592
13	x131	x13 1	0.54817	1	0.74039
14	x141	x14 1	0.55059	1	0.74201
15	x151	x15 1	0.52638	1	0.72552
16	x161	x16 1	0.44403	1	0.66636
17	x171	x17 1	0.57751	1	0.75994
18	x181	x18 1	0.56915	1	0.75442
19	x191	x19 1	0.58340	1	0.76381
20	x201	x20 1	0.54343	1	0.73718
				==	
				20	

---

We could also run this using the standardized orthogonal contrast coding as follows:

```
%choicemf(data=des,          /* candidate set of choice sets      */
  init=des(keep=set),       /* select these sets from candidates  */
  intiter=0,                /* evaluate without internal iterations*/
  model=class(x1-x20 / sta), /* model with stdzd orthogonal coding */
  nsets=40,                 /* number of choice sets             */
  nalts=2,                  /* number of alternatives             */
  /* extra model terms to drop from model */
  options=relative,        /* display relative D-efficiency     */
  beta=zero)                /* assumed beta vector, Ho: b=0     */
```

The results are as follows:

---

 Partial Profiles

## Final Results

Design	1
Choice Sets	40
Alternatives	2
Parameters	20
Maximum Parameters	40
D-Efficiency	8.6590
Relative D-Eff	21.6476
D-Error	0.1155
1 / Choice Sets	0.0250

## Partial Profiles

n	Variable		Variance	DF	Standard Error
	Name	Label			
1	x11	x1 1	0.11741	1	0.34264
2	x21	x2 1	0.13194	1	0.36324
3	x31	x3 1	0.10747	1	0.32783
4	x41	x4 1	0.11558	1	0.33996
5	x51	x5 1	0.13654	1	0.36951
6	x61	x6 1	0.20267	1	0.45019
7	x71	x7 1	0.12534	1	0.35403
8	x81	x8 1	0.12438	1	0.35268
9	x91	x9 1	0.12158	1	0.34868
10	x101	x10 1	0.13632	1	0.36922
11	x111	x11 1	0.14244	1	0.37741
12	x121	x12 1	0.13539	1	0.36796
13	x131	x13 1	0.13704	1	0.37019
14	x141	x14 1	0.13765	1	0.37101
15	x151	x15 1	0.13159	1	0.36276
16	x161	x16 1	0.11101	1	0.33318
17	x171	x17 1	0.14438	1	0.37997
18	x181	x18 1	0.14229	1	0.37721
19	x191	x19 1	0.14585	1	0.38190
20	x201	x20 1	0.13586	1	0.36859
				==	
				20	

---

The output has the same model terms as previously, but the efficiencies and variances have changed. Relative  $D$ -efficiency is 21.6476. The largest it could be under optimal circumstances is 25 since only 5 of 20 attributes (25%) can vary.

## Linear Partial-Profile Design

This section provides another example. Say that you want to make a design in 36 runs with 12 three-level factors, but you want only four factors to be considered at a time. You need to create four-level factors with one of the levels meaning not shown. You also need to ask for a design in 37 runs, because with partial profiles, one run must be all-constant. The following steps create a partial-profile design with the %MktEx macro using the `partial=` option:

```

title 'Partial Profiles';

%mktx(4 ** 12,          /* 12 four-level factor          */
      n=37,            /* 37 runs                        */
      partial=4,       /* partial profile, 4 attrs vary  */
      seed=462,        /* random number seed            */
      maxdesigns=1)    /* make only one design          */

%mktxlab(data=randomized, values=. 1 2 3, nfill=99)

proc print; run;

```

The iteration history proceeds like before, so we will not discuss it. The final  $D$ -efficiency is as follows:

---

Partial Profiles				
The OPTEX Procedure				
Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
-----				
1	49.4048	22.4346	100.0000	1.0000

---

With partial-profile designs,  $D$ -efficiency is typically much less than we are accustomed to seeing with other types of designs. The design is as follows:

---

Partial Profiles

Obs	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12
1	.	.	.	.	.	.	.	.	.	.	.	.
2	.	.	1	1	.	.	.	.	.	1	.	1
3	.	.	1	.	.	.	2	.	1	.	2	.
4	.	2	2	.	.	.	.	.	.	3	.	3
5	.	.	3	3	3	.	3	.	.	.	.	.
6	.	.	.	.	2	.	1	.	.	.	3	3
7	1	.	.	.	2	.	.	.	.	3	1	.
8	.	1	.	.	1	.	.	.	1	2	.	.
9	2	.	.	.	.	.	.	3	.	.	1	3
10	.	2	2	.	.	.	.	.	3	.	1	.
11	2	2	.	2	.	.	1	.	.	.	.	.
12	.	.	.	.	3	3	.	.	.	2	2	.
13	1	.	2	.	.	2	1	.	.	.	.	.
14	3	3	.	1	3	.	.	.	.	.	.	.
15	3	.	3	.	.	.	.	1	.	.	.	2
16	.	1	1	3	.	3	.	.	.	.	.	.
17	2	.	.	.	.	2	.	.	.	3	3	.
18	.	.	.	.	.	.	1	3	2	3	.	.
19	.	.	.	.	3	1	.	1	.	.	.	1
20	1	2	.	.	.	.	.	.	2	.	3	.
21	.	.	.	3	1	.	.	1	.	.	2	.
22	.	.	.	.	1	3	2	.	.	.	.	1
23	3	.	.	.	.	1	3	.	.	1	.	.
24	.	.	1	.	1	.	.	2	.	.	.	2
25	.	1	.	.	.	.	2	2	.	1	.	.
26	.	3	.	.	.	.	3	1	.	2	.	.
27	.	.	.	1	.	1	2	.	.	2	.	.
28	.	1	.	.	.	.	3	.	.	.	2	2
29	2	.	2	.	2	.	.	.	2	.	.	.
30	.	.	.	3	.	.	.	.	1	1	.	2
31	.	.	.	2	.	2	.	.	2	.	1	.
32	3	.	.	3	.	.	.	2	.	.	.	1
33	.	2	.	.	2	2	.	3	.	.	.	.
34	.	3	3	.	.	1	.	.	.	.	2	.
35	.	.	.	1	.	3	.	2	1	.	.	.
36	1	.	.	2	.	.	.	.	3	.	.	3
37	.	.	2	2	.	.	.	3	.	.	3	.

---

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

## Choice from Triples; Partial Profiles Constructed Using Restrictions

The approach we just saw, constructing partial profiles using the `partial=` option, is fine for a full-profile conjoint study or a pairwise choice study with level shifts. However, it is not good for a more general choice experiment with more alternatives. For a choice experiment, you need partial-profile restrictions on each alternative, and you must have the same attributes varying in every alternative within each choice set. There is currently no automatic way to request this in the `%MktEx` macro, so you have to program the restrictions yourself. To specify restrictions for choice designs, you need to take into consideration the number of attributes that can vary within each alternative, which ones, and which attributes go with which alternatives. Fortunately, that is not too difficult. See page 471 for another example of restrictions.

In this section, we construct a partial-profile design for a purely generic (unbranded) study, with ten attributes and three alternatives. Each attribute has three levels, and each alternative is a bundle of attributes. Partial-profile designs have the advantage that subjects do not have to consider all attributes at once. However, this is also a bit of a disadvantage as well in the sense that the subjects must constantly shift from considering one set of attributes to considering a different set. For this reason, it can be helpful to get more information out of each choice, and having more than two alternatives per choice set accomplishes this.

This example has several parts. As we mentioned in the chair study, we usually do not directly use the `%MktEx` macro to generate designs for generic studies. Instead, we use the `%MktEx` macro to generate a candidate set of partial-profile choice sets. Next, the design is checked and turned into a candidate set of choice sets. Next, the `%MktDups` macro is called to ensure there are no duplicate choice sets. Finally, the `%ChoiceEff` macro is used to create an efficient generic partial-profile choice design.

Before we go into any more detail on making this design, let's skip ahead and look at a couple of potential choice sets so it is clear what we are trying to accomplish and why. Two potential choice sets, still in linear arrangement, are as follows:

---

```

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

```

---

The same two potential choice sets, but now arrayed in choice design format, are as follows:

---

Partial Profiles										
Set	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
1	2	2	1	3	1	2	1	2	2	2
	2	1	3	2	1	1	2	1	2	2
	2	3	2	1	1	3	3	3	2	2
2	2	2	1	3	2	2	1	3	2	3
	3	2	2	3	1	2	1	1	1	1
	1	2	3	3	3	2	1	2	3	2

---



Each choice set has 10 three-level factors and three alternatives. Four attributes are constant in each choice set:  $x_1$ ,  $x_5$ ,  $x_9$ , and  $x_{10}$  in the first choice set, and  $x_2$ ,  $x_4$ ,  $x_6$ , and  $x_7$  in the second choice set. We do not need an all-constant choice set like we saw in our earlier partial-profile designs, nor do we need an extra level for not varying. In this approach, we simply construct choice sets for four constant attributes (they can be constant at 1, 2, or 3) and six varying attributes (with levels: 1, 2, and 3). Respondents are given a choice task along the lines of “Given a set of products that differ on these attributes but are identical in all other respects, which one would you choose?”. They are then shown a list of differences.

The following steps create the candidate design:

```

title 'Partial Profiles';

%macro partprof;
  sum = 0;
  do k = 1 to 10;
    sum = sum + (x[k] = x[k+10]    &   x[k] = x[k+20]);
  end;
  bad = abs(sum - 4);
%mend;

%mktx(3 ** 30,                /* 30 three-level factors          */
      n=198,                  /* 198 runs                        */
      options=quickcr        /* very quick run with random init */
          nox,                /* suppress x1, x2, ... creation   */
      order=random,          /* loop over columns in a random order */
      out=sasuser.cand,      /* output design stored permanently */
      restrictions=partprof, /* name of restrictions macro       */
      seed=382)              /* random number seed              */

```

We requested a design in 198 runs with 30 three-level factors. The 198 is chosen arbitrarily as a number divisible by  $3 \times 3 = 9$  that gives us approximately 200 candidate sets. The first ten factors,  $x_1$ - $x_{10}$ , make the first alternative, the next ten,  $x_{11}$ - $x_{20}$ , make the second alternative, and the last ten,  $x_{21}$ - $x_{30}$ , make the third alternative. We want six attributes to be nonconstant at a time. The PartProf macro counts the number of constant attributes:  $x_1 = x_{11} = x_{21}$ ,  $x_2 = x_{12} = x_{22}$ , ..., and  $x_{10} = x_{20} = x_{30}$ . If the number of constant attributes is four, our choice set conforms. If it is more or less than four, our choice set is in violation of the restrictions. The badness is the absolute difference between four and the number of constant attributes.

We specify `order=random` so the columns are looped over in a random order in the coordinate-exchange algorithm. When `partial=` is specified, as it is in the previous partial-profile examples, `order=random` is the default. Whenever you are imposing partial-profile restrictions without using the `partial=` option, you should specify `order=random`. Without `order=random`, %MktEx tends to put the nonconstant attributes close together in each row.

Our goal in this step is to make a candidate set of potential partial-profile choice sets, not to make a final experimental design. Ideally, it is nice to have more than random candidates—it is nice if our candidate generation code at least makes some attempt to ensure that our attributes are approximately orthogonal and balanced across attributes both between and within alternatives. This is a big problem (30 factors and 198 runs) with restrictions, so the %MktEx macro runs slowly by default. It is not critical that we permit the macro to spend a great deal of time optimizing linear model  $D$ -efficiency.

For this reason, we limit the number of iterations by specifying `options=quickr`. This is equivalent to specifying `optiter=0`, which specifies no OPTEX iterations, since with large partial-profile studies, we will never have a good candidate set for PROC OPTEX to search. It also specifies `tabiter=0` since an orthogonal array initial design is horrible for this problem. We also specified `options=nox` to increase efficiency by suppressing the creation of `x1`, `x2`, and so on, since they are not needed, and our restrictions are just based on `x`. Some of the results are as follows:

---

Partial Profiles

Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
1	Start	85.1531		Ran,Mut,Ann
1	169 1	93.3863	93.3863	Conforms
1	169 8	93.3892	93.3892	
1	169 22	93.3892	93.3892	
1	169 12	93.3911	93.3911	
1	170 20	93.3954	93.3954	
.				
.				
1	123 1	96.5805	96.5805	
1	38 21	96.5806	96.5806	
1	47 1	96.5811	96.5811	
1	End	96.5811		
.				
.				
.				

Partial Profiles

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	96.5811	93.5359	95.3976	0.5551

---

The macro finds a design that conforms to the restrictions (shown by the `Conforms` note). This step took approximately 3 minutes.

The rest of the code for making the partial-profile choice design is as follows:

```
%mktkey(3 10)

%mktroll(design=sasuser.cand, key=key, out=rolled)

%mktdups(generic, data=rolled, out=nodups, factors=x1-x10, nalts=3)

proc print data=nodups(obs=9); id set; by set; run;

%choiceff(data=nodups,          /* candidate set of choice sets      */
           model=class(x1-x10 / sta), /* model with stdzd orthogonal coding */
           seed=495,             /* random number seed              */
           maxiter=10,          /* maximum iterations for each phase */
           nsets=27,           /* number of choice sets           */
           nalts=3,            /* number of alternatives           */
           options=nodups      /* no duplicate choice sets        */
           relative,          /* display relative D-efficiency    */
           beta=zero)         /* assumed beta vector, Ho: b=0    */

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

The %MktKey macro is run to generate a Key data set with 3 rows, 10 columns and the variable names x1 - x30. The Key data set is as follows:

---

Partial Profiles									
x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
x11	x12	x13	x14	x15	x16	x17	x18	x19	x20
x21	x22	x23	x24	x25	x26	x27	x28	x29	x30

---

The %MktRoll macro creates a choice design from the linear candidate design.

The next step uses the %MktDups macro to check a design to see if there are any duplicate runs and output just the unique ones. For a generic study like this, it can also check to make sure there are duplicate choice sets taking into account the fact that two choice sets can be duplicates even if the alternatives are not in the same order. The %MktDups step names in a positional parameter the type of design as a **generic** choice design. It names the input data set and the output data set that contain the design with any duplicates removed. It names the factors in the choice design **x1-x10** and the number of alternatives. The result is a data set called **NoDups**. The first 3 candidate choice sets are as follows:

---

Partial Profiles											
Set	_Alt_	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
1	1	1	1	1	1	1	3	1	1	2	1
	2	2	3	2	3	3	2	1	1	2	1
	3	2	2	3	2	2	1	1	1	2	1
2	1	1	1	1	1	2	3	3	2	2	2
	2	2	3	1	1	2	2	1	3	3	2
	3	3	2	1	1	2	3	2	1	1	2
3	1	1	1	1	1	2	3	3	3	1	3
	2	3	1	1	1	2	2	3	1	3	2
	3	2	1	2	1	2	3	3	2	2	1

---

The `%ChoiEff` macro is called to search for an efficient choice design. The model specification `class(x1-x10 / sta)` specifies a generic model with 10 attributes and the standardized orthogonal contrast coding. The option `maxiter=10` specifies more than the default number of iterations (the default is 2 designs). We ask for a design with 27 sets and 3 alternatives. Furthermore, we ask for no duplicate choice sets, relative *D*-efficiency, and specify an assumed beta vector of zero. Some of the results from the `%ChoiEff` macro are as follows:

---

Partial Profiles			
Design	Iteration	D-Efficiency	D-Error
-----			
1	0	12.57808 *	0.07950
	1	14.91821 *	0.06703
	2	14.92197 *	0.06702
	.		
	.		
	.		
-----			
Design	Iteration	D-Efficiency	D-Error
-----			
10	0	12.55237	0.07967
	1	14.95788	0.06685
	2	15.05607	0.06642
	3	15.06856	0.06636

## Partial Profiles

## Final Results

Design	9
Choice Sets	27
Alternatives	3
Parameters	20
Maximum Parameters	54
D-Efficiency	15.3187
Relative D-Eff	56.7360
D-Error	0.0653
1 / Choice Sets	0.0370

## Partial Profiles

n	Variable		Variance	DF	Standard Error
	Name	Label			
1	x11	x1 1	0.069368	1	0.26338
2	x12	x1 2	0.061410	1	0.24781
3	x21	x2 1	0.069033	1	0.26274
4	x22	x2 2	0.065628	1	0.25618
5	x31	x3 1	0.072568	1	0.26938
6	x32	x3 2	0.082373	1	0.28701
7	x41	x4 1	0.071943	1	0.26822
8	x42	x4 2	0.074494	1	0.27294
9	x51	x5 1	0.068258	1	0.26126
10	x52	x5 2	0.070409	1	0.26535
11	x61	x6 1	0.063116	1	0.25123
12	x62	x6 2	0.065635	1	0.25619
13	x71	x7 1	0.062414	1	0.24983
14	x72	x7 2	0.069006	1	0.26269
15	x81	x8 1	0.071232	1	0.26689
16	x82	x8 2	0.080080	1	0.28298
17	x91	x9 1	0.062817	1	0.25063
18	x92	x9 2	0.068655	1	0.26202
19	x101	x10 1	0.063551	1	0.25209
20	x102	x10 2	0.069304	1	0.26326
				==	
				20	

---

Part of the design is as follows:

---

Partial Profiles										
Set	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
197	3	3	3	3	1	3	3	3	2	3
	3	1	3	2	1	1	2	2	3	3
	3	2	3	1	1	2	1	1	1	3
191	3	3	3	1	3	1	2	1	3	2
	3	1	1	2	3	1	3	1	1	1
	3	2	2	3	3	1	1	1	2	3
.										
.										
.										
110	2	3	1	1	1	3	1	3	1	2
	3	3	3	3	1	3	3	1	1	1
	1	3	2	2	1	3	3	2	1	3

---

The design has 27 choice sets. The choice set numbers shown in this output correspond to the *original* set numbers in the candidate design not the choice set numbers in the final design. This design has a relative  $D$ -efficiency of 56.7360. It can be no larger than 60 since 6 of 10 attributes vary.

### Six Alternatives; Partial Profiles Constructed Using Restrictions

In this next example, we construct a partial-profile design with 20 binary attributes and six alternatives with 15 attributes fixed at the base-line level of 1 for each alternative. Our partial-profile restriction macro is an obvious modification of the one used in the previous example. Our linear arrangement needs  $6 \times 20 = 120$  factors. The first attribute is made from  $x_1$ ,  $x_{21}$ ,  $x_{41}$ ,  $x_{61}$ ,  $x_{81}$ , and  $x_{101}$ ; the second attribute is made from  $x_2$ ,  $x_{22}$ ,  $x_{42}$ ,  $x_{62}$ ,  $x_{82}$ , and  $x_{102}$ ; and so on. The do loop counts the number of times all of the linear factors within an attribute are equal to one and our badness function increases as the number of constant attributes deviates from 15. The following step creates the macro:

```
%macro partprof;
  sum = 0;
  do k = 1 to 20;
    sum = sum + (x[k] = 1 & x[k+20] = 1 & x[k+40] = 1 &
                x[k+60] = 1 & x[k+80] = 1 & x[k+100] = 1);
  end;
  bad = abs(sum - 15);
%mend;
```

The %MktEx macro is run as follows:

```
%mktex(2 ** 120,          /* 120 two-level factors      */
        n=300,            /* 300 runs                    */
        order=random,     /* loop over columns in a random order */
        out=cand,         /* output design                */
        restrictions=partprof, /* name of restrictions macro    */
        seed=424,         /* random number seed           */
        maxtime=0,        /* stop once design conforms     */
        options=largedesign /* make large design more quickly */
        nosort            /* do not sort output design     */
        resrep            /* detailed report on restrictions */
        quickr            /* very quick run with random init */
        nox)              /* suppress x1, x2, ... creation */
```

This step requests 120 binary factors and 300 runs. We specify `order=random` so that the columns are looped over in a random order in the coordinate-exchange algorithm. You should always specify `order=random` with partial-profile designs. With a sequential order you tend to get nonvarying attributes paired with only nearby attributes. Several options are specified to make this step run quickly. With `maxtime=0`, only one design is created. With `options=quickr`, that one design is created with the coordinate exchange algorithm and a random initialization. (The “r” in `quickr` stands for random.) The `largedesign` option allows %MktEx to stop as soon as it has imposed all restrictions. The `resrep` option reports on the progress of imposing restrictions.

The first part of the output is as follows:

---

#### Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
1	Start	79.5699		Ran,Mut,Ann
1	1	79.8384		14 Violations
1	2	80.0541		12 Violations
1	3	80.3047		14 Violations
1	4	80.5649		15 Violations
1	5	80.7840		13 Violations
1	6	81.0383		12 Violations
.				
.				
.				

---

At the end of the first pass through the design we see the following warning:

```
WARNING: It may be impossible to meet all restrictions.
```

The macro never gets anywhere in imposing restrictions. Every row of the design has many restriction violations as shown by the output from the `resrep` (restrictions report) option. Always specify `options=resrep` with complicated restrictions so you can look for things like this. When the macro is

never succeeding in imposing restrictions, there is something wrong with the restrictions macro. The macro we used is as follows:

```
%macro partprof;
  sum = 0;
  do k = 1 to 20;
    sum = sum + (x[k]      = 1 & x[k+20] = 1 & x[k+40] = 1 &
                x[k+60] = 1 & x[k+80] = 1 & x[k+100] = 1);
  end;
  bad = abs(sum - 15);
%mend;
```

It correctly evaluates the Boolean expression to determine whether an attribute is all one, and it correctly counts the number of such attributes. It also correctly sets `bad` to the absolute difference between the sum and 15, the desired number of constant attributes. Sometimes you get results like we just saw when you made a logical mistake in programming the restrictions. For example, you might have written a set of restrictions that are impossible to satisfy. That is not the problem in this case. The problem in this case is the quantification of badness is not fine enough. You need to tell the macro whenever it does something that moves it closer to an acceptable solution. Consider a potential choice set that almost conforms to the restrictions such as the following:

---

Set	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	x18	x19	x20
1	2	1	1	1	2	2	1	1	1	1	1	1	2	1	1	1	1	1	1	1
	2	1	2	1	2	2	1	1	1	1	1	1	2	1	1	1	1	1	1	1
	1	1	2	1	1	2	2	1	1	1	1	1	2	1	1	1	1	1	1	1
	1	1	2	1	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2	1	1	1	1	1	2	1	1	1	1	1	1	1	1	1	1	1	1	1
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

---

The problem with this choice set is it has six nonconstant attributes instead of five. So `bad = abs(14 - 15) = 1`. Now consider what happens when the macro changes the first attribute of the first alternative from 2 to 1. This is a change in the right direction because it moves the choice set closer to having one more constant attribute. However, this change has no effect on the badness criterion. It is still 1, because we still have six nonconstant attributes. We are not giving `%MktEx` enough information about when it is heading in the right direction. The `%MktEx` macro, without your restrictions macro to otherwise guide it, is guided by the goal of maximizing *D*-efficiency. It does not particularly want to make partial-profile designs, because imposing all of those ties decreases statistical *D*-efficiency. Using the hill climbing analogy, `%MktEx` wants to climb Mount Everest; your restrictions macro needs to tell it to find the top of an island in the middle of a river valley. This is not where `%MktEx` would normally look. If your restrictions macro is going to overcome the `%MktEx` macro's normal goal, you have to train it and more explicitly tell it where to look.

Training the `%MktEx` macro to find highly restricted designs is like training a dog. You have to be persistent and consistent, and you need to watch it every second. You have to reward it whenever it does the right thing and you have to punish it for even thinking about doing the wrong thing. When it tears up your favorite slippers, you need to whack it over the head with a rolled up newspaper.\* It

---

\*No actual dogs were harmed in the process of developing any of this software.



is eager to please, but it is easily tempted, and it is not smart enough to figure out what to do unless you very explicitly tell it. It will run wherever it wants unless you keep it on a short leash. With that in mind, the revised partial-profile restrictions macro is as follows:

```
%macro partprof;
  sum = 0;
  do k = 1 to 20;
    sum = sum + (x[k] = 1 & x[k+20] = 1 & x[k+40] = 1 &
                x[k+60] = 1 & x[k+80] = 1 & x[k+100] = 1);
  end;
  bad = abs(sum - 15);
  if sum < 15 & x[j1] = 2 then do;
    k = mod(j1 - 1, 20) + 1;
    c = (x[k] = 1) + (x[k+20] = 1) + (x[k+40] = 1) +
        (x[k+60] = 1) + (x[k+80] = 1) + (x[k+100] = 1);
    if c >= 3 then bad = bad + (6 - c) / 6;
  end;
%mend;
```

It starts the same way as the old one, but it has some additional fine tuning. Like before, this macro counts the number of times that all six alternatives equal 1 and stores the result in `sum`. When `sum` is less than 15, we need more constant attributes. When the current level of the current factor, `x[j1]`, is 2, the macro next considers whether it should change the 2 to a 1. First, we compute `k`, the attribute number and evaluate the number of ones in that attribute, and store that in `c`. (For example, when `j1`, the linear factor number, equals 2, 22, 42, 62, 82, or 102, we are working with attribute `k = 2`.) If it looks like this factor is going to be constant (three or more ones), we add the proportion of twos to the badness function (more twos is worse).

Consider again the first row of the sample choice set shown previously. Badness starts out as 1 since `sum` is 14. Since badness is nonzero and since we are looking at a 2 in the first column, badness is increased by 1/2, which is the proportion of twos. Now consider changing that first two to a one. Now badness is  $1 + 1/3$ , which is smaller than  $1 + 1/2$  so changing 2 to 1 moves badness in the right direction.

There are many other ways that you could write a restrictions macro for this problem. However you do it, you need to provide a quantification of badness that guides the macro toward acceptable designs. Most of the time, for complicated restrictions, you will not be able to sit down and write a good restrictions macro off of the top of your head.\* It requires some trial and error to get something that works. For this reason, at least at first, you should specify `maxtime=0`, `options=largedesign resrep quickr` so you can see if the macro is succeeding in imposing restrictions, and so the macro stops quickly when it has succeeded. Then you need to carefully check your design. It is not unusual for the macro to succeed splendidly only to find you gave it the wrong set of restrictions. We can run the `%MktEx` macro exactly as before to test our new macro:

---

\*This one might look simple once you see it, but it took me several tries to get it right. I had previous versions that worked quite well in spite of some logical flaws that caused them to quantify badness in not quite the way that I intended.

```

%mktx(2 ** 120,          /* 120 two-level factors          */
      n=300,             /* 300 runs                          */
      order=random,     /* loop over columns in a random order */
      out=cand,         /* output design                      */
      restrictions=partprof, /* name of restrictions macro        */
      seed=424,         /* random number seed                */
      maxtime=0,        /* stop once design conforms          */
      options=largedesign /* make large design more quickly    */
        nosort          /* do not sort output design         */
        resrep          /* detailed report on restrictions    */
        quickr          /* very quick run with random init   */
        nox)            /* suppress x1, x2, ... creation     */

```

Some of the output from the first pass through the design is as follows:

---

#### Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
-----				
1	Start	79.5699		Ran,Mut,Ann
1	1	79.5426		0 Violations
1	2	79.5904		1 Violations
1	3	79.6679		4 Violations
1	4	79.6601		2 Violations
1	5	79.6136		0 Violations
.				
.				
.				
1	100	70.7815		0 Violations
1	101	70.6383		3.5 Violations
1	102	70.4911		0 Violations
.				
.				
.				
1	197	54.9906		2.5 Violations
1	198	54.8216		2 Violations
1	199	54.6956		3 Violations
1	200	54.5982		4 Violations
1	201	54.4009		3 Violations
1	202	54.2690		5 Violations
1	203	54.0316		1 Violations
.				
.				
.				
1	297	37.1307		4 Violations
1	298	36.9190		0 Violations
1	299	36.7720		4 Violations
1	300	36.5326		2 Violations

---

This iteration history looks better. At least some choice sets conform. Most do not however. Notice the fractional number of violations in some rows. Throughout most of this iteration history,  $D$ -efficiency is steadily going down as more and more restrictions are imposed. Part of the iteration history for the second pass through the design is as follows:

---

Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
1	1	36.5626		0 Violations
1	2	36.6017		0 Violations
1	3	36.5544		0 Violations
1	4	36.5408		0 Violations
1	5	36.5872		0 Violations
.				
.				
.				
1	71	35.3957		0 Violations
1	72	35.2990		1 Violations
1	72	35.3149		0 Violations
.				
.				
.				
1	200	32.7482		0 Violations
1	201	32.7255		0 Violations
1	202	32.6120		0 Violations
.				
.				
.				
1	298	30.8923		0 Violations
1	299	30.8077		0 Violations
1	300	30.7819		0 Violations

---

This part of the iteration history looks much better. The table contains a number of rows like we see in row 72. The %MktEx macro attempts to impose restrictions and it does not quite succeed, so it immediately tries again, one or more times, until it succeeds or gives up. In this case, it always succeeds.  $D$ -efficiency is still mostly decreasing. Note that the “0 Violations” that we see in this table tells us that there are no violations remaining in the indicated row. There might very well still be violations in other rows. However by the end of this pass, the restrictions are almost certainly completely imposed, so we should see  $D$ -efficiency start back up. Some of the output from the next pass through the design is as follows:

---

 Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
1	1	30.7883		0 Violations
1	2	30.8000		0 Violations
1	3	30.8232		0 Violations
.				
.				
1	100	32.2841		0 Violations
1	101	32.3019		0 Violations
1	102	32.3076		0 Violations
.				
.				
1	200	33.1653		0 Violations
1	201	33.1732		0 Violations
1	202	33.1784		0 Violations
.				
.				
1	298	33.8588		0 Violations
1	299	33.8877		0 Violations
1	300	33.8971		0 Violations

---

Now *D*-efficiency is increasing. The rest of the iteration history is as follows:

---

## Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
1	1 1	33.8971	33.8971	Conforms
1	1 53	33.8988	33.8988	
1	1 60	33.9013	33.9013	
1	1 6	33.9016	33.9016	
1	End	33.9016		

---

It is followed by the following messages:

NOTE: Stopping early, possibly before convergence, with a large design.

Due to `maxtime=0`, `options=largedesign quickr`, the macro stops after it has completed one pass through the design without encountering any restriction violations. Our final  $D$ -efficiency = 33.9016 is not very high, but this is just a candidate set. Furthermore, this is a *highly* restricted design, so it is not surprising that  $D$ -efficiency is not very high. We might want to iterate more and see if we can do better, but for now, let's test the rest of our code. The `%MktEx` step took about 2.5 minutes. When the macro completes a full pass through the design without detecting any violations, it displays `Conforms` and switches from the `options=resrep` style to the normal iteration history style. The following steps turn the linear arrangement into a choice design and eliminate duplicate choice sets:

```
%mktkey(6 20)

%mktroll(design=cand, key=key, out=rolled)

proc print; by set; id set; var x:; where set le 5; run;

%mktdups(generic, data=rolled, out=nodups, factors=x1-x20, nalts=6)

proc print data=nodups(obs=18); id set; by set; run;
```

The `%MktKey` macro is run to generate a Key data set with 6 rows, 20 columns and the variable names `x1 - x120`. The Key data set, displayed in two panels, is as follows:

---

x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
x21	x22	x23	x24	x25	x26	x27	x28	x29	x30
x41	x42	x43	x44	x45	x46	x47	x48	x49	x50
x61	x62	x63	x64	x65	x66	x67	x68	x69	x70
x81	x82	x83	x84	x85	x86	x87	x88	x89	x90
x101	x102	x103	x104	x105	x106	x107	x108	x109	x110
x11	x12	x13	x14	x15	x16	x17	x18	x19	x20
x11	x12	x13	x14	x15	x16	x17	x18	x19	x20
x31	x32	x33	x34	x35	x36	x37	x38	x39	x40
x51	x52	x53	x54	x55	x56	x57	x58	x59	x60
x71	x72	x73	x74	x75	x76	x77	x78	x79	x80
x91	x92	x93	x94	x95	x96	x97	x98	x99	x100
x111	x112	x113	x114	x115	x116	x117	x118	x119	x120

---

The `%MktDups` macro eliminated 70 choice sets with duplicate alternatives resulting in a candidate set with 230 choice sets.

The results of the last PROC PRINT, with the first three candidate choice sets, are as follows:

---

Set	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	x18	x19	x20
1	1	1	1	1	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2
	1	1	1	1	1	1	1	1	2	1	1	1	2	1	1	1	1	1	1	1
	1	1	1	1	2	1	1	1	2	1	1	1	1	1	1	1	1	1	1	1
	1	1	1	1	1	2	1	1	1	1	1	1	2	1	1	1	1	1	1	2
	1	1	1	1	2	2	1	1	2	1	1	1	1	1	1	1	1	1	1	2
	1	1	1	1	1	1	1	1	1	1	1	1	2	1	1	1	1	1	1	1
2	1	2	1	1	1	1	1	1	1	1	1	1	2	1	1	2	2	1	2	1
	1	2	1	1	1	1	1	1	1	1	1	1	1	1	1	2	2	1	2	1
	1	2	1	1	1	1	1	1	1	1	1	1	1	1	1	2	2	1	2	1
	1	1	1	1	1	1	1	1	1	1	1	1	2	1	1	2	1	1	1	1
	1	1	1	1	1	1	1	1	1	1	1	1	2	1	1	1	1	1	2	1
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3	2	1	1	1	1	1	1	2	2	1	1	1	1	2	1	1	1	1	1	1
	1	1	1	1	1	1	1	2	1	1	1	1	1	1	1	2	1	1	1	1
	2	1	1	1	1	1	1	1	2	1	1	1	1	2	1	2	1	1	1	1
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	1	1	1	1
	1	1	1	1	1	1	1	2	2	1	1	1	1	1	1	2	1	1	1	1
	2	1	1	1	1	1	1	2	2	1	1	1	1	2	1	1	1	1	1	1

---

Each choice set has the correct number of nonconstant attributes. The following step runs the %ChoiceEff macro to find a choice design:

```
%choiceff(data=nodups,          /* candidate set of choice sets      */
           model=class(x1-x20 / zero=first), /* main effects, first ref level*/
           seed=495,             /* random number seed              */
           maxiter=10,          /* maximum iterations for each phase */
           nsets=18,            /* number of choice sets           */
           nalts=6,             /* number of alternatives           */
           options=nodups,      /* no duplicate choice sets        */
           beta=zero)           /* assumed beta vector, Ho: b=0    */
```

```
proc print data=best; id set; by notsorted set; var x1-x20; run;
```

We set the reference level to the first level of each factor, which is the base-line level of 1. Some of the output is as follows:

---

	n	Name	Beta	Label
	1	x12	0	x1 2
	2	x22	0	x2 2
	3	x32	0	x3 2
	4	x42	0	x4 2
	5	x52	0	x5 2
	6	x62	0	x6 2
	7	x72	0	x7 2
	8	x82	0	x8 2
	9	x92	0	x9 2
	10	x102	0	x10 2
	11	x112	0	x11 2
	12	x122	0	x12 2
	13	x132	0	x13 2
	14	x142	0	x14 2
	15	x152	0	x15 2
	16	x162	0	x16 2
	17	x172	0	x17 2
	18	x182	0	x18 2
	19	x192	0	x19 2
	20	x202	0	x20 2

Design	Iteration	D-Efficiency	D-Error
-----			
1	0	0	.
	1	1.02522 *	0.97540
	2	1.03841 *	0.96301
	3	1.03875 *	0.96269

Design	Iteration	D-Efficiency	D-Error
-----			
2	0	0.82624	1.21030
	1	1.02442	0.97616
	2	1.03920 *	0.96228
	3	1.04403 *	0.95783

.

.

.

Design	Iteration	D-Efficiency	D-Error
-----			
10	0	0.84581	1.18229
	1	1.01501	0.98521
	2	1.03206	0.96893
	3	1.04232	0.95940
	4	1.04232	0.95940

Final Results

Design	2
Choice Sets	18
Alternatives	6
Parameters	20
Maximum Parameters	90
D-Efficiency	1.0501
D-Error	0.9495

n	Variable		Variance	DF	Standard Error
	Name	Label			
1	x12	x1 2	0.97668	1	0.98827
2	x22	x2 2	1.10960	1	1.05338
3	x32	x3 2	1.10506	1	1.05122
4	x42	x4 2	0.87763	1	0.93682
5	x52	x5 2	0.92710	1	0.96286
6	x62	x6 2	0.75023	1	0.86616
7	x72	x7 2	0.92874	1	0.96371
8	x82	x8 2	1.10497	1	1.05118
9	x92	x9 2	1.06918	1	1.03401
10	x102	x10 2	1.08282	1	1.04058
11	x112	x11 2	1.14818	1	1.07153
12	x122	x12 2	0.85384	1	0.92404
13	x132	x13 2	0.84918	1	0.92151
14	x142	x14 2	1.10046	1	1.04903
15	x152	x15 2	1.09171	1	1.04485
16	x162	x16 2	0.87602	1	0.93596
17	x172	x17 2	1.05549	1	1.02737
18	x182	x18 2	0.86694	1	0.93109
19	x192	x19 2	1.06911	1	1.03398
20	x202	x20 2	1.08380	1	1.04106
				==	
				20	

Set	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	x18	x19	x20
227	1	1	1	1	1	1	1	1	1	1	2	1	1	1	1	1	1	1	1	1
	1	1	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2
	1	1	2	1	1	1	1	1	1	1	2	1	1	1	1	2	1	1	1	1
	1	1	1	1	1	1	1	1	1	1	1	1	2	1	1	2	1	1	1	2
	1	1	1	1	1	1	1	1	1	1	1	1	2	1	1	1	1	1	1	1
	1	1	2	1	1	1	1	1	1	1	1	1	2	1	1	2	1	1	1	2



```

187  1  1  1  1  1  1  1  1  1  2  1  2  1  1  1  1  1  2  1  1
      1  1  1  2  1  1  1  1  1  1  1  2  1  1  2  1  1  1  1  1
      1  1  1  2  1  1  1  1  1  2  1  1  1  1  2  1  1  2  1  1
      1  1  1  2  1  1  1  1  1  2  1  2  1  1  1  1  1  1  1  1
      1  1  1  2  1  1  1  1  1  1  1  1  1  1  2  1  1  2  1  1
      1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
      .
      .
      .
191  1  1  2  1  2  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
      1  1  2  1  1  1  1  1  1  1  1  1  1  1  1  1  1  2  1  1
      1  1  2  1  1  1  1  2  1  1  1  1  1  2  1  1  1  1  1  1
      1  1  1  1  1  1  1  2  1  1  1  1  1  2  1  1  1  1  1  1
      1  1  1  1  2  1  1  2  1  1  1  1  1  1  1  1  1  2  1  1
      1  1  1  1  2  1  1  1  1  1  1  1  1  2  1  1  1  2  1  1

```

Since the `%ChoiceEff` macro did not have any problems, and the results look reasonable, it appears that we did everything right. Now we could try again, perhaps with more choice sets (say 400), and we could iterate longer (say one hour), then we could make our final partial-profile choice design. This is illustrated in the following step:

```

%mktx(2 ** 120,          /* 120 two-level factors          */
      n=400,             /* 400 runs                          */
      optiter=0,         /* no PROC OPTEX iterations          */
      tabiter=0,         /* no OA initialization iterations    */
      maxtime=60,       /* at most 60 minutes per phase     */
      order=random,     /* loop over columns in a random order */
      out=cand,         /* output design                      */
      restrictions=partprof, /* name of restrictions macro        */
      seed=424,         /* random number seed                */
      maxstages=1,     /* maximum number of algorithm stages */
      options=largedesign /* make large design more quickly    */
      nosort)           /* do not sort output design         */

```

The results of this step are not shown.

To recap, the first restrictions macro correctly differentiated between acceptable and unacceptable choice sets, but it provided `%MktEx` with no guidance or direction on how to find acceptable choice sets. Hence, the first macro did not work. The second macro corrected this problem by “nudging” `%MktEx` in the right direction. The restrictions macro looked for attributes that appear to be heading toward constant and created a penalty function that encouraged `%MktEx` to make those attributes constant. Next, we will look at another way of writing a restrictions macro for this problem. There is nothing subtle about this next approach. This next macro uses the whack-it-over-the-head-with-a-rolled-up-newspaper approach. Sometimes you need to tell `%MktEx` that a restriction is really important by strongly eliminating that source of badness. In this case, our macro strongly eliminates twos from the design until the restriction violations go away. This is illustrated in the following steps:

```

%macro partprof;
  sum = 0;
  do k = 1 to 20;
    sum = sum + (x[k] = 1 & x[k+20] = 1 & x[k+40] = 1 &
                x[k+60] = 1 & x[k+80] = 1 & x[k+100] = 1);
  end;
  bad = abs(sum - 15);
  if sum < 15 & x[j1] = 2 then bad = bad + 1000;
%mend;

%mktx(2 ** 120, /* 120 two-level factors */
      n=300, /* 300 runs */
      order=random, /* loop over columns in a random order */
      out=cand, /* output design */
      restrictions=partprof, /* name of restrictions macro */
      seed=424, /* random number seed */
      maxtime=0, /* stop once design conforms */
      options=largedesign /* make large design more quickly */
          nosort /* do not sort output design */
          resrep /* detailed report on restrictions */
          quickr /* very quick run with random init */
          nox) /* suppress x1, x2, ... creation */

```

Recall that we specified `order=random` so within each choice set, the columns are traversed in a different random order. As long as there are violations, within each choice set, this macro turns random twos into ones until there are so few twos left that the violations go away. Then once all of the violations go away, it allows twos to be changed back to ones to increase  $D$ -efficiency.

Part of the iteration history is as follows:

---

#### Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
1	Start	79.5699		Ran,Mut,Ann
1	1	79.5701		0 Violations
1	2	79.5231		0 Violations
1	3	79.4445		0 Violations
1	4	79.3049		0 Violations
1	5	79.1400		0 Violations
.	.	.	.	.
1	100	57.7096		0 Violations
1	101	57.5138		0 Violations
1	102	57.2500		0 Violations

.			
.			
.			
1	200	34.9873	0 Violations
1	201	34.7893	0 Violations
1	202	34.5983	0 Violations
.			
.			
.			
1	298	18.0811	0 Violations
1	299	17.8622	0 Violations
1	300	17.5956	0 Violations
1	1	17.7230	0 Violations
1	2	17.8323	0 Violations
1	3	17.9371	0 Violations
.			
.			
.			
1	100	23.7155	0 Violations
1	101	23.7609	0 Violations
1	102	23.8067	0 Violations
.			
.			
.			
1	200	28.3416	0 Violations
1	201	28.3683	0 Violations
1	202	28.3953	0 Violations
.			
.			
.			
1	298	32.0063	0 Violations
1	299	32.0372	0 Violations
1	300	32.0804	0 Violations

1	1	1	32.0804	32.0804	Conforms
1	1	13	32.0873	32.0873	
1	1	27	32.0950	32.0950	
1	1	105	32.0981	32.0981	
1	1	103	32.1020	32.1020	
1	1	113	32.1032	32.1032	
1	1	65	32.1096	32.1096	
1	1	106	32.1116	32.1116	
1	1	86	32.1159	32.1159	
1	1	43	32.1163	32.1163	
1	1	47	32.1193	32.1193	
1	1	45	32.1232	32.1232	
1	1	3	32.1294	32.1294	
1	1	53	32.1295	32.1295	
1	1	107	32.1344	32.1344	
1	1	6	32.1366	32.1366	
1	1	73	32.1372	32.1372	
1		End	32.1372		

With this approach, all violations in each row are eliminated in the first pass through the design. The macro quits after the end of the second pass, when it has completed an entire pass without encountering any restriction violations.

Call the first approach the “nudge” approach and the second approach the “whack” approach. The nudge approach starts with  $D$ -efficiency on the order of 80% for the random design. After one complete pass, imposing many but not all restrictions,  $D$ -efficiency is down around 37%. After another pass and imposing all restrictions, it is down to around 31%. Then it creeps back up to 34%.  $D$ -efficiency for the choice design is approximately 1.05. The whack approach starts with the same random design (due to the same random number seed) and with the same  $D$ -efficiency on the order of 80%. Then after one pass of severe restriction imposition,  $D$ -efficiency drops to 18%. The nudge approach asks “are you a good two or a bad two?” and then acts accordingly. In contrast, when the whack approach sees a two, it whacks it—no questions asked. There is no subtle nudging in the whack approach, and initially, it over corrects to impose restrictions. Hence, the design it makes at the end of the first pass is not very good, but once all of the restrictions are in place,  $D$ -efficiency quickly recovers to 32%. This is not quite as high as the nudge approach, but the nudge approach had one more complete pass through the design.

It is natural to ask, which approach is better? There are many ways to get %MktEx to impose the restrictions. It is not clear that one is better than the other. Our goal here is to use %MktEx to create a set of candidates for the %ChoiceEff macro to search. Any restrictions macro that accomplishes this should be fine. Writing a macro that uses the whack approach is probably a bit easier. However, there is always some worry that the initial over correction might not be a good thing. In contrast, subtle nudging takes longer to get to the point of all restrictions being met, and you have to be a bit creative sometimes to write nudges that actually work. Sometimes you need to combine both approaches—whack it to take care of one set of restrictions then nudge it in the right direction for a secondary set of restrictions. This is all part of the art of sophisticated experimental design. Note that it is not the fact that we used a large penalty of 1000 that makes this approach an example of the whack approach. It is the whack approach because we strongly over-corrected every violation.

The differential weighting of the components of badness is another very important strategy to keep in mind. When there is more than one aspect to the set of restrictions, they sometimes trade off against each other. Every time %MktEx fixes one thing, something else gets worse. Differentially weighting the components, as we did in the partprof macro, can greatly help avoid this. The first component, `bad = abs(sum - 15)`; sets badness to values in the range 0 to 15. The next component `if sum < 15 & x[j1] = 2 then bad = bad + 1000`; adds contributions in the thousands. %MktEx will never increase the second component to decrease the first. This gives %MktEx something of a two-stage functionality. At first, it worries about eliminating twos and also having the right number, but mostly it eliminates twos. Once it eliminates the most-penalized source of badness then it can concentrate on the remaining source, as long as it does not bring back any of the badness it previously eliminated. Often times, it does not matter which component you weight the most, nor does it matter how much you weight them, as long as each component has a differential weight.

We could run the %ChoiceEff macro again to see what the final *D*-efficiency is like before as follows:

```
%choiceff(data=nodups,          /* candidate set of choice sets      */
           model=class(x1-x20 / zero=first), /* main effects, first ref level*/
           seed=495,            /* random number seed              */
           maxiter=10,         /* maximum iterations for each phase */
           nsets=18,           /* number of choice sets           */
           nalts=6,            /* number of alternatives           */
           options=nodups,     /* no duplicate choice sets        */
           beta=zero)          /* assumed beta vector, Ho: b=0    */
```

The summary table is as follows:

---

#### Final Results

Design	3
Choice Sets	18
Alternatives	6
Parameters	20
Maximum Parameters	90
D-Efficiency	1.0531
D-Error	0.9495

---

These results are similar to those from before. We could also specify the standardized orthogonal contrast coding as follows:

```
%choiceff(data=nodups,          /* candidate set of choice sets      */
           model=class(x1-x20 / sta), /* model with stdzd orthogonal coding */
           seed=495,            /* random number seed              */
           maxiter=10,         /* maximum iterations for each phase */
           nsets=18,           /* number of choice sets           */
           nalts=6,            /* number of alternatives           */
           options=nodups     /* no duplicate choice sets        */
           relative,          /* display relative D-efficiency    */
           beta=zero)          /* assumed beta vector, Ho: b=0    */
```

The summary table is as follows:

---

Final Results	
Design	3
Choice Sets	18
Alternatives	6
Parameters	20
Maximum Parameters	90
D-Efficiency	4.2126
Relative D-Eff	23.4032
D-Error	0.2374
1 / Choice Sets	0.0556

---

D-efficiency is 23.4032. The largest it could possibly be is 25 since 5 of 20 attributes (25%) vary.

### Five-Level Factors; Partial Profiles Constructed Using Restrictions

This next example extends what we discussed in the previous example and constructs a somewhat different style of partial-profile design. This design has five alternatives and 15 five-level factors, five of which vary in each choice set. Unlike the previous example, however, the constant factors are not all at the base-line level. The constant factors can have any of the levels, and we use the first factor within each attribute when we check the restrictions. The partial-profile restrictions macro along with %MktEx code, which uses the same basic option set that we used in the previous example, is as follows:

```
%macro partprof;
  sum = 0;
  do k = 1 to 15;
    sum = sum + (x[k+15] = x[k] & x[k+30] = x[k] &
                x[k+45] = x[k] & x[k+60] = x[k]);
  end;
  bad = abs(sum - 10);
  if sum < 10 & x[j1] ^= x[mod(j1 - 1, 15) + 1] then bad = bad + 1000;
%mend;

%mktext(5 ** 75,          /* 75 five-level factors          */
        n=400,           /* 400 runs                        */
        order=random,    /* loop over columns in a random order */
        out=cand,        /* output design                    */
        restrictions=partprof, /* name of restrictions macro      */
        seed=472,        /* random number seed              */
        maxtime=0,       /* stop once design conforms        */
        options=largedesign /* make large design more quickly  */
        nosort           /* do not sort output design        */
        resrep           /* detailed report on restrictions  */
        quickr           /* very quick run with random init  */
        nox)             /* suppress x1, x2, ... creation    */
```

The macro counts the number of times all of the linear factors in a choice set attribute are constant within choice set, that is they equal the level of the first linear factor in the attribute, then it computes badness in the customary way. Also like before, when not all restrictions are met, any level that is not equal to the first factor within its attribute is heavily penalized. The macro uses the “whack” approach to impose constant attributes. Some of the iteration history is as follows:

---

Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
-----				
1	1	58.6862		0 Violations
1	2	58.7150		1 Violations
1	3	58.6810		2 Violations
.				
.				
.				
1	100	56.0416		5 Violations
1	101	55.9496		0 Violations
1	102	55.8855		4 Violations
.				
.				
.				
1	200	48.0539		3 Violations
1	201	47.9776		3 Violations
1	202	47.9279		4 Violations
.				
.				
.				
1	300	38.9711		0 Violations
1	301	38.8695		4 Violations
1	302	38.7805		3 Violations
.				
.				
.				
1	398	30.4723		2 Violations
1	399	30.4075		2 Violations
1	400	30.3169		2 Violations
1	1	30.4110		0 Violations
1	2	30.4491		0 Violations
1	3	30.5001		0 Violations
.				
.				
.				

1	73	32.8050		2 Violations
1	73	32.8708		0 Violations
1	74	32.8840		0 Violations
1	75	32.8518		1 Violations
1	75	32.8845		0 Violations
.				
.				
.				
1	398	34.9958		0 Violations
1	399	35.0104		0 Violations
1	400	34.9901		0 Violations
1	1	35.0015		0 Violations
1	2	35.0133		0 Violations
1	3	35.0337		0 Violations
.				
.				
.				
1	398	43.0865		0 Violations
1	399	43.1054		0 Violations
1	400	43.1304		0 Violations
1	1 1	43.1304	43.1304	Conforms
1	1 42	43.1311	43.1311	
1	1 17	43.1338	43.1338	
1	End	43.1305		

---

This iteration history has a pattern very similar to what we saw in the previous example with the nudge approach. In the first pass through the design, not all restrictions are met. Even the whack approach does not guarantee that all restrictions get met right away. In the second pass, some rows are processed more than once until all restrictions are met. We can see that in choice sets 73 and 75. By the end of the second pass, all restrictions are met (efficiency starts back up), `%MktEx` realizes all restrictions are met at the end of the third pass, and then it stops. The `%MktEx` macro iterates longer if we do not specify `options=largedesign`, but for now when you are testing a new restrictions macro, it is good to check the results before `%MktEx` spends a long time iterating. The following steps create a choice design from this linear candidate set in the familiar way:



```

%mkkey(5 15)

%mktroll(design=cand, key=key, out=rolled)

%mktdups(generic, data=rolled, out=nodups, factors=x1-x15, nalts=5)

%choicEff(data=nodups,          /* candidate set of choice sets      */
           model=class(x1-x15 / sta), /* model with stdz orthogonal coding */
           seed=513,             /* random number seed              */
           maxiter=10,          /* maximum iterations for each phase */
           nsets=15,           /* number of choice sets            */
           nalts=5,            /* number of alternatives            */
           options=nodups      /* no duplicate choice sets          */
           relative,          /* display relative D-efficiency     */
           beta=zero)         /* assumed beta vector, Ho: b=0      */

proc print data=best(obs=15); id set; by notsorted set; var x1-x15; run;

```

The key with 5 rows, 15 columns and the variable names x1 - x75 is as follows:

---

x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15
x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15
x16	x17	x18	x19	x20	x21	x22	x23	x24	x25	x26	x27	x28	x29	x30
x31	x32	x33	x34	x35	x36	x37	x38	x39	x40	x41	x42	x43	x44	x45
x46	x47	x48	x49	x50	x51	x52	x53	x54	x55	x56	x57	x58	x59	x60
x61	x62	x63	x64	x65	x66	x67	x68	x69	x70	x71	x72	x73	x74	x75

---

Skipping the %ChoiEff macro output for a moment, part of the choice design is as follows:

---

Set	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15
33	5	5	4	5	2	1	4	2	3	5	5	2	5	3	1
	5	1	3	1	2	1	4	4	3	5	1	2	5	3	1
	5	5	3	4	2	1	4	5	3	5	2	2	5	3	1
	5	1	1	2	2	1	4	1	3	5	5	2	5	3	1
	5	2	5	3	2	1	4	3	3	5	3	2	5	3	1
291	3	2	3	1	2	4	5	4	2	4	3	1	1	2	4
	3	2	3	1	2	4	5	3	3	4	3	1	5	4	4
	3	2	3	1	2	4	4	2	5	4	3	1	3	5	4
	3	2	3	1	2	4	5	5	2	4	3	1	4	3	4
	3	2	3	1	2	4	4	1	4	4	3	1	1	1	4

186	3	4	3	3	5	2	1	1	4	1	3	1	1	3	5
	3	4	5	3	5	2	5	2	4	2	3	1	1	3	4
	3	4	2	3	5	2	3	1	4	4	3	1	1	3	2
	3	4	1	3	5	2	4	3	4	5	3	1	1	3	5
	3	4	4	3	5	2	4	5	4	3	3	1	1	3	3

The pattern of constant and nonconstant attributes looks correct: 10 constant and 5 nonconstant attributes per choice set. Furthermore, the constant attributes have the full range of levels. The last part of the output from the %ChoiceEff macro is as follows:

#### Final Results

Design	9
Choice Sets	15
Alternatives	5
Parameters	60
Maximum Parameters	60
D-Efficiency	2.3286
Relative D-Eff	15.5243
D-Error	0.4294
1 / Choice Sets	0.0667

n	Variable Name	Label	Variance	DF	Standard Error
1	x11	x1 1	1.24427	1	1.11547
2	x12	x1 2	1.37689	1	1.17341
3	x13	x1 3	1.62664	1	1.27540
4	x14	x1 4	1.25242	1	1.11912
5	x21	x2 1	1.06398	1	1.03149
6	x22	x2 2	2.50082	1	1.58140
7	x23	x2 3	1.54702	1	1.24379
8	x24	x2 4	1.00564	1	1.00282
9	x31	x3 1	1.68441	1	1.29785
10	x32	x3 2	0.72847	1	0.85351
11	x33	x3 3	1.58669	1	1.25964
12	x34	x3 4	1.36204	1	1.16707
13	x41	x4 1	1.57492	1	1.25496
14	x42	x4 2	2.40504	1	1.55082
15	x43	x4 3	2.07707	1	1.44120
16	x44	x4 4	1.37807	1	1.17391
17	x51	x5 1	1.84721	1	1.35912
18	x52	x5 2	3.63100	1	1.90552
19	x53	x5 3	2.87391	1	1.69526
20	x54	x5 4	0.87844	1	0.93725

21	x61	x6 1	2.42661	1	1.55776
22	x62	x6 2	3.33545	1	1.82632
23	x63	x6 3	2.00194	1	1.41490
24	x64	x6 4	2.13206	1	1.46016
25	x71	x7 1	0.82550	1	0.90857
26	x72	x7 2	1.15680	1	1.07555
27	x73	x7 3	0.80379	1	0.89655
28	x74	x7 4	0.76404	1	0.87409
29	x81	x8 1	2.98038	1	1.72638
30	x82	x8 2	1.37852	1	1.17410
31	x83	x8 3	2.82780	1	1.68161
32	x84	x8 4	2.81404	1	1.67751
33	x91	x9 1	1.00980	1	1.00489
34	x92	x9 2	0.98026	1	0.99008
35	x93	x9 3	1.30321	1	1.14158
36	x94	x9 4	2.07571	1	1.44073
37	x101	x10 1	1.66894	1	1.29187
38	x102	x10 2	0.69715	1	0.83496
39	x103	x10 3	0.83388	1	0.91317
40	x104	x10 4	1.44343	1	1.20143
41	x111	x11 1	0.77932	1	0.88279
42	x112	x11 2	2.22471	1	1.49155
43	x113	x11 3	0.45386	1	0.67369
44	x114	x11 4	1.50188	1	1.22551
45	x121	x12 1	1.69913	1	1.30351
46	x122	x12 2	1.18246	1	1.08741
47	x123	x12 3	0.83460	1	0.91357
48	x124	x12 4	0.99096	1	0.99547
49	x131	x13 1	4.57118	1	2.13803
50	x132	x13 2	1.51438	1	1.23060
51	x133	x13 3	3.21429	1	1.79284
52	x134	x13 4	8.78785	1	2.96443
53	x141	x14 1	1.22443	1	1.10654
54	x142	x14 2	1.85878	1	1.36337
55	x143	x14 3	1.39724	1	1.18205
56	x144	x14 4	0.91801	1	0.95813
57	x151	x15 1	1.27954	1	1.13117
58	x152	x15 2	3.05334	1	1.74738
59	x153	x15 3	3.27077	1	1.80853
60	x154	x15 4	2.76477	1	1.66276

==  
60

---

The relative  $D$ -efficiency is 15.5243. The largest it could possibly be is 33 since 5 of 15 attributes (33%) vary. Our value is on the order of half that. Another way to assess the quality of the design is to look at the parameter variances. In this table they seem quite variable. This is usually not a good sign. This run of the %ChoiceEff macro requested a design with only 15 choice sets, which is not a lot. In fact, with 15 choice sets and 4 alternatives, we can estimate at most 60 parameters, which is exactly the number we are trying to estimate here. This is usually not a good idea. Let's try again, this time

with 30 choice sets. The following step finds the design:

```
%choicereff(data=nodups,          /* candidate set of choice sets      */
             model=class(x1-x15 / sta), /* model with stdz orthogonal coding */
             seed=513,             /* random number seed              */
             maxiter=10,           /* maximum iterations for each phase */
             nsets=30,             /* number of choice sets           */
             nalts=5,              /* number of alternatives           */
             options=nodups        /* no duplicate choice sets        */
             relative,             /* display relative D-efficiency    */
             beta=zero)            /* assumed beta vector, Ho: b=0    */
```

The new results and variance tables are as follows:

---

### Partial Profiles

#### Final Results

Design	1
Choice Sets	30
Alternatives	5
Parameters	60
Maximum Parameters	120
D-Efficiency	7.7251
Relative D-Eff	25.7503
D-Error	0.1294
1 / Choice Sets	0.0333

#### Partial Profiles

n	Variable		Variance	DF	Standard Error
	Name	Label			
1	x11	x1 1	0.16200	1	0.40250
2	x12	x1 2	0.15231	1	0.39027
3	x13	x1 3	0.15801	1	0.39751
4	x14	x1 4	0.18802	1	0.43361
5	x21	x2 1	0.20554	1	0.45337
6	x22	x2 2	0.15363	1	0.39196
7	x23	x2 3	0.16931	1	0.41147
8	x24	x2 4	0.17124	1	0.41381
9	x31	x3 1	0.14382	1	0.37924
10	x32	x3 2	0.13760	1	0.37094
11	x33	x3 3	0.19570	1	0.44238
12	x34	x3 4	0.15172	1	0.38952
13	x41	x4 1	0.13177	1	0.36300
14	x42	x4 2	0.13093	1	0.36185
15	x43	x4 3	0.13788	1	0.37133

16	x44	x4 4	0.15115	1	0.38878
17	x51	x5 1	0.14697	1	0.38336
18	x52	x5 2	0.12922	1	0.35947
19	x53	x5 3	0.12505	1	0.35362
20	x54	x5 4	0.17476	1	0.41804
21	x61	x6 1	0.15004	1	0.38735
22	x62	x6 2	0.17357	1	0.41661
23	x63	x6 3	0.12204	1	0.34934
24	x64	x6 4	0.13766	1	0.37102
25	x71	x7 1	0.18417	1	0.42915
26	x72	x7 2	0.16833	1	0.41027
27	x73	x7 3	0.13836	1	0.37197
28	x74	x7 4	0.14091	1	0.37538
29	x81	x8 1	0.16784	1	0.40969
30	x82	x8 2	0.16454	1	0.40564
31	x83	x8 3	0.14048	1	0.37481
32	x84	x8 4	0.17935	1	0.42349
33	x91	x9 1	0.17887	1	0.42293
34	x92	x9 2	0.17178	1	0.41447
35	x93	x9 3	0.18717	1	0.43263
36	x94	x9 4	0.16287	1	0.40357
37	x101	x10 1	0.17880	1	0.42284
38	x102	x10 2	0.15523	1	0.39400
39	x103	x10 3	0.15494	1	0.39362
40	x104	x10 4	0.15128	1	0.38894
41	x111	x11 1	0.15144	1	0.38916
42	x112	x11 2	0.19196	1	0.43813
43	x113	x11 3	0.18085	1	0.42526
44	x114	x11 4	0.17372	1	0.41680
45	x121	x12 1	0.16954	1	0.41175
46	x122	x12 2	0.19912	1	0.44623
47	x123	x12 3	0.16908	1	0.41120
48	x124	x12 4	0.15868	1	0.39834
49	x131	x13 1	0.17289	1	0.41580
50	x132	x13 2	0.19481	1	0.44138
51	x133	x13 3	0.21682	1	0.46564
52	x134	x13 4	0.17501	1	0.41834
53	x141	x14 1	0.15621	1	0.39524
54	x142	x14 2	0.13804	1	0.37154
55	x143	x14 3	0.13537	1	0.36793
56	x144	x14 4	0.16000	1	0.40000
57	x151	x15 1	0.14634	1	0.38254
58	x152	x15 2	0.16575	1	0.40712
59	x153	x15 3	0.14842	1	0.38525
60	x154	x15 4	0.17341	1	0.41643

==  
60

This looks much better. The variances are smaller and more uniform and relative  $D$ -efficiency is larger. Note that it is good to run the %MktEx macro again without `options=largedesign` and let it iterate more before making the final choice design.

Next, we will investigate another thing you can try. Typically, the %MktEx macro is run so that it loops over all of the columns in a row, and then it goes on to the next row. Alternatively, it can work with *pairs* of columns at one time using the `exchange=2` option. Working with pairs of columns instead of single columns is always much slower, but sometimes it can make better designs. The following steps create the design:

```
%macro partprof;
  sum = 0;
  do k = 1 to 15;
    sum = sum + (x[k+15] = x[k] & x[k+30] = x[k] &
                x[k+45] = x[k] & x[k+60] = x[k]);
  end;
  bad = abs(sum - 10);
  if sum < 10 then do;
    if x[j1] ^= x[mod(j1 - 1, 15) + 1] then bad = bad + 1000;
    if x[j2] ^= x[mod(j2 - 1, 15) + 1] then bad = bad + 1000;
  end;
%mend;

%mktx(5 ** 75,                /* 75 five-level factors          */
      n=400,                  /* 400 runs                        */
      optiter=0,              /* no PROC OPTEX iterations       */
      tabiter=0,              /* no OA initialization iterations */
      maxtime=720,           /* at most 720 minutes per phase  */
      order=random=15,       /* pairwise exchanges within      */
                                /* nonconstant attributes         */
      out=sasuser.cand,      /* output design stored permanently */
      restrictions=partprof, /* name of restrictions macro     */
      seed=472,              /* random number seed             */
      exchange=2,           /* pairwise exchanges             */
      maxstages=1,          /* maximum number of algorithm stages */
      options=largedesign   /* make large design more quickly  */
      nosort                 /* do not sort output design      */
      resrep                 /* detailed report on restrictions */
      nox)                   /* suppress x1, x2, ... creation  */
```

The initial quantification of badness is the same. Like before, nonconforming levels are whacked. This time however, they are whacked in two ways—when `j1`, the primary column index points to a nonconstant level, and when `j2`, the secondary column index for the pairwise exchange indexes a nonconstant level. In the %MktEx invocation, we now see `maxtime=720` so that %MktEx can run over night for 12 hours (or 720 minutes). We also see `exchange=2` for pairwise exchanges. The output data set is stored as a permanent SAS data set in the `sasuser` library. If we search for 12 hours for a design, we want to make sure it is there for us if we accidentally trip over the power cord in the morning before we have our coffee. See page 309 for more information about permanent SAS data sets.

There is one more option in this example that we have not used previously, `order=random=15`. This is a special variation on `order=random` for pairwise exchanges in partial-profile designs. Before this option is explained, here is a bit of background. Sequential pairwise exchanges for  $m$  factors works like this: %MktEx sets  $j1 = 1, 2, 3, \dots, m$  and  $j2 = j1 + 1, j1 + 2, \dots, m$ . Together, the variables  $j1$  and  $j2$  loop over all pairs of columns. Random pairwise exchanges work like this: this: %MktEx sets  $j1 = \text{random\_permutation}(1, 2, 3, \dots, m)$  and  $j2 = \text{random\_permutation}(j1 + 1, j1 + 2, \dots, m)$ . Together, the variables  $j1$  and  $j2$  loop over all pairs of columns but in a random order. For partial profiles, pairwise exchanges are appealing, because sometimes there is a lot to be gained by having two values change at once. However, it does not make sense to consider simultaneously changing the level of a nonconstant attribute and the level of a constant attribute, nor does it make sense to consider pairwise exchanges within constant attributes. Random exchange with a value of 15 specified works like this: %MktEx sets  $j1 = \text{random\_permutation}(1, 2, 3, \dots, m)$  and  $j2$  is set to a sequential list of the other factors in the same attribute as  $j1$ . For example, when  $j1 = 18$ , which means  $j1$  is indexing the second alternative (18 is in the second block of 15 factors) for the third attribute (18 is 3 beyond the fifteenth factor, which is the end of the first block), then  $j2 = 3, 18, 33, 48, \text{ and } 63$  for a nonconstant third attribute (which index the 5 factors that make up the third attribute) and  $j2 = j1$  for constant attributes. This does pairwise exchanges but only within nonconstant attributes. This eliminates a lot of unproductive pairs from consideration.

A small part of the iteration history is as follows:

---

Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
-----				
1	Start	58.6611		Ran,Mut,Ann
1	1	58.6962		0 Violations
1	2	58.7140		0 Violations
1	3	58.7171		1 Violations
1	4	58.7076		1 Violations
1	5	58.7269		2 Violations
1	6	58.6943		2 Violations
.				
.				
.				
1	398	31.1040		0 Violations
1	399	31.0346		3 Violations
1	400	30.9260		1 Violations
1	1	31.0114		0 Violations
1	2	31.0566		0 Violations
1	3	31.1412		0 Violations
.				
.				
.				

1	48		32.6265		0 Violations
1	49		32.6150		3 Violations
1	49		32.5890		1 Violations
1	49		32.6547		0 Violations
1	50		32.7205		0 Violations
.					
.					
.					
1	398		34.7853		0 Violations
1	399		34.7463		0 Violations
1	400		34.7763		0 Violations
1	1		34.8061		0 Violations
1	2		34.8381		0 Violations
1	3		34.8520		0 Violations
.					
.					
.					
1	398		43.2881		0 Violations
1	399		43.2774		0 Violations
1	400		43.3051		0 Violations
1	1	1	43.3051	43.3051	Conforms
1	1	72	43.3085	43.3085	
1	1	26	43.3105	43.3105	
1	1	41	43.3113	43.3113	
.					
.					
.					
1	154	7	50.0432	50.0432	
1	176	1	50.0432	50.0432	
1	217	17	50.0432	50.0432	
1		End	50.0432		

---

It is followed by the following messages:

NOTE: Stopping early, possibly before convergence, with a large design.

NOTE: Quitting the algorithm search step after 720.01 minutes and 22 designs.

The following steps make and evaluate the choice design:



```

%mkkey(5 15)

%mkroll(design=sasuser.cand, key=key, out=rolled)

%mktdups(generic, data=rolled, out=nodups, factors=x1-x15, nalts=5)

%choiceff(data=nodups,           /* candidate set of choice sets      */
           model=class(x1-x15 / sta), /* model with stdz orthogonal coding */
           seed=513,              /* random number seed                */
           maxiter=10,           /* maximum iterations for each phase */
           nsets=30,            /* number of choice sets             */
           nalts=5,             /* number of alternatives            */
           options=nodups       /* no duplicate choice sets          */
           relative,           /* display relative D-efficiency     */
           beta=zero)          /* assumed beta vector, Ho: b=0     */

proc print data=best(obs=15); id set; by notsorted set; var x1-x15; run;

```

The new results and variance tables are as follows:

---

#### Partial Profiles

##### Final Results

Design	3
Choice Sets	30
Alternatives	5
Parameters	60
Maximum Parameters	120
D-Efficiency	8.1495
Relative D-Eff	27.1650
D-Error	0.1227
1 / Choice Sets	0.0333

##### Partial Profiles

n	Variable		Variance	DF	Standard Error
	Name	Label			
1	x11	x1 1	0.15201	1	0.38989
2	x12	x1 2	0.11803	1	0.34356
3	x13	x1 3	0.13012	1	0.36072
4	x14	x1 4	0.11936	1	0.34548
5	x21	x2 1	0.15472	1	0.39334
6	x22	x2 2	0.14263	1	0.37766
7	x23	x2 3	0.17875	1	0.42279
8	x24	x2 4	0.17528	1	0.41866
9	x31	x3 1	0.13811	1	0.37163
10	x32	x3 2	0.15821	1	0.39775

11	x33	x3 3	0.13531	1	0.36784
12	x34	x3 4	0.13226	1	0.36367
13	x41	x4 1	0.14763	1	0.38422
14	x42	x4 2	0.14864	1	0.38554
15	x43	x4 3	0.13789	1	0.37134
16	x44	x4 4	0.14818	1	0.38494
17	x51	x5 1	0.15596	1	0.39491
18	x52	x5 2	0.15623	1	0.39526
19	x53	x5 3	0.15559	1	0.39444
20	x54	x5 4	0.13505	1	0.36750
21	x61	x6 1	0.14271	1	0.37777
22	x62	x6 2	0.14806	1	0.38479
23	x63	x6 3	0.14098	1	0.37548
24	x64	x6 4	0.14159	1	0.37629
25	x71	x7 1	0.14727	1	0.38376
26	x72	x7 2	0.11428	1	0.33806
27	x73	x7 3	0.13768	1	0.37105
28	x74	x7 4	0.14633	1	0.38253
29	x81	x8 1	0.12268	1	0.35026
30	x82	x8 2	0.13987	1	0.37399
31	x83	x8 3	0.13153	1	0.36267
32	x84	x8 4	0.12463	1	0.35303
33	x91	x9 1	0.21901	1	0.46798
34	x92	x9 2	0.16280	1	0.40349
35	x93	x9 3	0.14938	1	0.38650
36	x94	x9 4	0.14706	1	0.38348
37	x101	x10 1	0.15695	1	0.39617
38	x102	x10 2	0.17411	1	0.41726
39	x103	x10 3	0.14308	1	0.37826
40	x104	x10 4	0.13558	1	0.36822
41	x111	x11 1	0.14586	1	0.38192
42	x112	x11 2	0.15817	1	0.39771
43	x113	x11 3	0.15272	1	0.39080
44	x114	x11 4	0.19799	1	0.44496
45	x121	x12 1	0.14785	1	0.38451
46	x122	x12 2	0.21303	1	0.46155
47	x123	x12 3	0.14543	1	0.38136
48	x124	x12 4	0.12144	1	0.34848
49	x131	x13 1	0.19242	1	0.43866
50	x132	x13 2	0.20030	1	0.44755
51	x133	x13 3	0.17448	1	0.41770
52	x134	x13 4	0.17275	1	0.41563
53	x141	x14 1	0.15431	1	0.39282
54	x142	x14 2	0.13919	1	0.37309
55	x143	x14 3	0.15637	1	0.39543

56	x144	x14 4	0.13149	1	0.36261
57	x151	x15 1	0.15171	1	0.38950
58	x152	x15 2	0.14736	1	0.38387
59	x153	x15 3	0.14358	1	0.37892
60	x154	x15 4	0.15097	1	0.38855
				==	
				60	

---

Now, relative  $D$ -Efficiency is 27.1650 compared to 25.7503 previously. This is typical in the sense that it is not unusual to get a small gain for a large increase in computation. The first few choice sets are as follows:

---

Set	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15
128	1	3	5	3	4	1	5	1	5	2	3	3	3	2	4
	4	3	3	5	4	1	2	1	5	3	3	3	3	2	4
	5	3	1	1	4	1	3	1	5	4	3	3	3	2	4
	3	3	2	4	4	1	4	1	5	1	3	3	3	2	4
	2	3	4	2	4	1	1	1	5	5	3	3	3	2	4
100	1	4	3	4	3	4	5	3	2	2	4	4	4	1	5
	5	4	5	4	4	5	5	1	2	2	4	4	4	1	5
	4	4	2	4	5	2	5	2	2	2	4	4	4	1	5
	2	4	1	4	1	3	5	4	2	2	4	4	4	1	5
	3	4	4	4	2	1	5	5	2	2	4	4	4	1	5
11	3	1	3	4	1	1	4	4	2	4	1	3	4	1	5
	3	1	4	4	1	1	2	2	5	4	4	3	4	1	5
	3	1	5	4	1	1	3	5	1	4	5	3	4	1	5
	3	1	2	4	1	1	1	1	4	4	2	3	4	1	5
	3	1	1	4	1	1	5	3	3	4	3	3	4	1	5

---

The example starting on page 613 and the choice design shown on page 618 creates a partial-profile design with all constant attributes equal to one. In contrast, this design uses the full range of values for the constant attributes. In terms of fitting the choice model, it does not matter. An attribute that is constant within a choice set does not contribute to the likelihood function for that choice set, and this is true no matter what the constant value is. It typically does not matter for data collection either, since data collection is typically phrased in terms like “everything else being equal” without any specifics about what the equal levels are. About the only difference is in the `%MktEx` macro.  $D$ -efficiency should be higher with the varying-constant approach than with the all-one approach.

# Partial Profiles from Block Designs and Orthogonal Arrays

These next examples make optimal partial-profile designs from balanced incomplete block designs (BIBDs) and small orthogonal arrays (OAs). Before we get into partial profiles, let's discuss BIBDs and review OAs.

## *Balanced Incomplete Block Designs*

The following design is a BIBD:

---

Balanced Incomplete Block Design

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

---

It consists of  $b = 20$  rows or *blocks*. The design has  $k = 4$  columns, so each block is of size four. The design consists of the integers 1 to 16, so over the entire design, there are  $t = 16$  attributes. The design has a total of  $bk = 20 \times 4 = 80$  elements. This design is balanced in the sense that each of the  $t = 16$  attributes occurs exactly  $r = 5$  times ( $rt = 5 \times 16 = bk = 20 \times 4 = 80$ ). Furthermore, the  $t(t-1)/2 = 16(16-1)/2 = bk(k-1)/2 = 20 \times 4(4-1)/2 = 120$  pairs of attributes occur together in the same block exactly once. This is what makes this design a BIBD. The attribute by attribute frequencies are as follows:

## Attribute by Attribute Frequencies

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

In the partial-profile context, this BIBD instructs us to create 20 blocks of choice sets. This BIBD can be used for the situation where there are 16 attributes and 4 are varied at a time. In the first block, attributes 10, 4, 16, and 1 vary while the remaining attributes are held constant; in the second block, attributes 16, 13, 14, and 11 vary while the remaining attributes are held constant; and so on. Every attribute appears in a block of choice sets with every other attribute and each attribute occurs exactly as often as every other attribute. A BIBD is not an orthogonal array or ordinary factorial design like we get out of the %MktEx macro, so we use a different tool, the %MktBIBD macro to find BIBDs. For more information about BIBDs and how they relate to factorial designs see page 965. BIBDs only exist for specific and limited combinations of  $b$ ,  $k$ , and  $t$ . We can run the %MktBSize macro to find sizes that meet necessary but not sufficient criteria for the existence of a BIBD as follows:

```
%mktbsize(b=20, nattrs=16, setsize=4)
```

This step reports the following, which shows that a BIBD might exist for this specification, but of course we already knew that:

t	k	b	r	Lambda	n
Number of Attributes	Set Size	Number of Sets	Attribute Frequency	Pairwise Frequencies	Total Sample Size
16	4	20	5	1	80

More generally, we can run the %MktBSize macro with lists of parameters and get a list of those combinations of parameters that meet the necessary but not sufficient conditions for the existence of a BIBD. This list includes the design we just saw. The following step generates the list:

```
%mktbsize(nattrs=15 to 20, setsize=3 to t / 2, b=t to 30)
```

The results are as follows:

---

t	k	b	r	Lambda	n
Number of Attributes	Set Size	Number of Sets	Attribute Frequency	Pairwise Frequencies	Total Sample Size
15	5	21	7	2	105
15	7	15	7	3	105
16	4	20	5	1	80
16	6	16	6	2	96
16	8	30	15	7	240
19	9	19	9	4	171

---

There is no guarantee that %MktBIBD will find a BIBD for any specification, even when one is in fact known to exist. However, it usually does quite well in finding smaller BIBDS and in finding designs that are close the rest of the time. When it cannot find a BIBD, it finds a design where each attribute occurs as often as every other attribute or almost as often and each pair of attributes occurs together almost equally often. Furthermore, the %MktBIBD macro can find designs like this even when the parameters do not meet the necessary criteria. For the purposes of making a partial-profile design, a block design with almost equal frequencies is usually good enough. The following step shows how we found the BIBD shown in the beginning of this section:

```
%mktbibd(b=20, nattrs=16, setsize=4, seed=17)
```

The entire set of output from this macro is as follows:

---

Block Design Efficiency Criterion	100.0000
Number of Attributes, t	16
Set Size, k	4
Number of Sets, b	20
Attribute Frequency	5
Pairwise Frequency	1
Total Sample Size	80
Positional Frequencies Optimized?	Yes

## Attribute by Attribute Frequencies

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

## Attribute by Position Frequencies

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

## Balanced Incomplete Block Design

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

---

The first part of the output consists of a “factoid” of miscellaneous statistics, parameters, and information. The efficiency criterion is 100, so a BIBD was found. Next, the parameters of the BIBD,  $t = 16$ ,  $k = 4$ , and  $b = 20$  are reported. The attribute frequency of 5 shows that each of the  $t = 16$  attributes occurs five times in the array. Each pair of attributes occurs once. The total sample size refers to the  $bk = 20 \times 4 = 80$  elements in the BIBD. In the final BIBD, the positional frequencies are optimized. This is shown in the attribute by position frequencies, which are discussed soon.

The factoid is followed by the attribute by attribute frequency matrix. We want to see a constant diagonal and a constant upper triangular matrix above the diagonal, and that is what we get since we have a BIBD. The next matrix contains the attribute by position frequencies. We see that each attribute occurs in each position either one or two times. It is impossible for the results to be constant in this BIBD. However, they are as close to constant as they possibly can be, and there are no zeros, threes, or greater values. Note though, that the attribute by position frequencies are not important to us in the partial-profile context. We could specify `positer=` to turn off the iterations that optimize the positional frequencies to make the macro run faster for this problem. In this case, we get the same BIBD, but with the values in a different order within row. The last matrix is the BIBD.



*Orthogonal Arrays*

An orthogonal array (OA) is a factorial design in which all estimable effects are uncorrelated. The `%MktEx` macro has an extensive catalog of orthogonal arrays. A sample of a small OA is as follows:

---

x1	x2	x3	x4	x5
1	1	1	1	1
1	2	2	2	2
2	1	2	1	2
2	2	1	2	1
3	1	1	2	2
3	2	2	1	1
4	1	2	2	1
4	2	1	1	2

---

This design is balanced—each of the four levels of `x1` occurs exactly twice, and each of the two levels of `x2-x4` occurs exactly four times. This design is orthogonal—each of the eight pairs of levels of `x1` with `x2-x5` occurs exactly once, and each of the four pairs of levels for each of the pairs of factors in `x2-x5` occurs exactly twice. Hence this design is an OA. It can be created as follows:

```
%mktex(4 2**4, n=8)
```

```
proc print noobs; run;
```

*80 Choice Sets, 16 Binary Attributes, Four Varying*

In this example, we create a partial-profile design with  $t = 16$  binary attributes and  $k = 4$  varying at one time. We create 80 choice sets of two alternatives each. The resulting design is optimal under the assumption  $\beta = \mathbf{0}$  (Anderson 2003). The following steps create and evaluate the design:

```
%mktbibd(b=20, nattrs=16, setsize=4, seed=17)
```

```
%mktex(4 2 ** 4, n=8, seed=306)
```

```
proc sort data=randomized out=randes(drop=x1);
```

```
  by x2 x1;
```

```
  run;
```

```
proc print noobs data=randes; run;
```

```
%mktppro(design=randes, ibd=bibd)
```

```

%choicetf(data=chdes,          /* candidate set of choice sets      */
  init=chdes,                  /* initial design                      */
  initvars=x1-x6,             /* factors in the initial design       */
  model=class(x1-x16 / sta), /* model with stdz orthogonal coding   */
  nsets=80,                   /* number of choice sets              */
  nalts=2,                    /* number of alternatives              */
  rscale=                      /* relative D-efficiency scale factor  */
  %sysevalf(80 * 4 / 16), /* 4 of 16 attrs in 80 sets vary     */
  beta=zero)                  /* assumed beta vector, Ho: b=0       */

proc print data=chdes(obs=12); id set; by set; run;

%mktdups(generic, data=best, factors=x1-x16, nalts=2)

```

The first step makes the BIBD as we saw in the previous step. It tells us which attributes to vary in each choice set. The following design is a BIBD:

---

Balanced Incomplete Block Design

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

---

Next we need to know how those attributes are to vary. We need an OA for that. The OA must be a  $p^k$  subset of an array  $p^k s^1$  in  $p \times s$  runs with  $k \leq s$ . Here we have  $p = 2$  alternatives and we want to vary  $k = 4$  at a time. The smallest array that works is  $2^4 4^1$  in 8 runs. The `%MktEx` step generates this array, and the PROC PRINT step displays it. The results are as follows:

---

x2	x3	x4	x5
1	1	2	1
1	2	2	2
1	1	1	2
1	2	1	1
2	2	1	2
2	1	1	1
2	2	2	1
2	1	2	2

---

This array has two blocks of observations. The first block shows the attribute levels for the first alternative, and the second block shows the attribute levels for the second alternative.

There are several ways to create an OA that is sorted in the right way. The preceding `%MktEx` step requests the  $s$ -level factor first, then it sorts the randomized design by the first  $p$ -level factor followed by the  $s$ -level factor. Finally, it discards the  $s$ -level factor. Alternatively, you could first request one of the  $p$ -level factors, then request the  $s$ -level factor, then request the remaining  $(k - 1)$   $p$ -level factors. Then in the `out=design` data set, *after* the array is created, discard `x2`, the  $s$ -level factor. Note that you cannot drop the second factor in the `%MktEx` step by specifying `out=design(drop=x2)`, because `%MktEx` will drop the variable and then sort, and your rows will be in the wrong order. The former approach has the advantage of being less likely to produce alternatives that are constant (that is, in one alternative, all attributes are absent and in other alternatives, all attributes are present).

The `%MktPPro` step combines the BIBD and the OA. The data set with the partial-profile choice design is called `chdes`. We can use the `%ChoiEff` macro to evaluate the design. The last part of the output is as follows:

---

#### Final Results

Design	1
Choice Sets	80
Alternatives	2
Parameters	16
Maximum Parameters	80
D-Efficiency	20.0000
Relative D-Eff	100.0000
1 / Choice Sets	0.0125

n	Variable		Variance	DF	Standard Error
	Name	Label			
1	x11	x1 1	0.05	1	0.22361
2	x21	x2 1	0.05	1	0.22361
3	x31	x3 1	0.05	1	0.22361
4	x41	x4 1	0.05	1	0.22361
5	x51	x5 1	0.05	1	0.22361
6	x61	x6 1	0.05	1	0.22361
7	x71	x7 1	0.05	1	0.22361
8	x81	x8 1	0.05	1	0.22361
9	x91	x9 1	0.05	1	0.22361
10	x101	x10 1	0.05	1	0.22361
11	x111	x11 1	0.05	1	0.22361
12	x121	x12 1	0.05	1	0.22361
13	x131	x13 1	0.05	1	0.22361
14	x141	x14 1	0.05	1	0.22361
15	x151	x15 1	0.05	1	0.22361
16	x161	x16 1	0.05	1	0.22361
				==	
				16	

---

The variances are constant, and the relative  $D$ -efficiency is 100.  $D$ -efficiency is 20, which is  $4 / 16$  of the number of choice sets, 80. Since 25% of the attributes vary,  $D$ -efficiency is 25% of what we would expect in a full-profile generic choice design. With the `rscale=20` option, relative  $D$ -efficiency is based on the true maximum. The macro function `%sysevalf` is used to evaluate the expression  $80 * 4 / 16$  to get the 20. This function evaluates expressions that contain or produce floating point numbers and then returns the result. This particular expression could be evaluated with `%eval`, which does integer arithmetic, but that is not always the case. In summary, this design is optimal for partial profiles, and it is 25% efficient relative to an optimal generic design where all attributes can vary.

The final PROC PRINT step displays the first six choice sets as follows:

---

Set	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2
	2	1	1	2	1	1	1	1	1	2	1	1	1	1	1	1
2	1	1	1	2	1	1	1	1	1	2	1	1	1	1	1	2
	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1	1	2	1	1	1	1	1	1
	2	1	1	2	1	1	1	1	1	1	1	1	1	1	1	2
4	1	1	1	2	1	1	1	1	1	1	1	1	1	1	1	1
	2	1	1	1	1	1	1	1	1	2	1	1	1	1	1	2
5	1	1	1	1	1	1	1	1	1	1	1	1	1	2	1	1
	1	1	1	1	1	1	1	1	1	1	2	1	2	1	1	2
6	1	1	1	1	1	1	1	1	1	1	2	1	2	2	1	1
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2

---

There are a total of 80 choice sets,  $b = 20$  blocks of size  $s = 4$ . The first four choice sets show the 8-run OA embedded in the partial-profile design in columns 1, 4, 10, and 16, as directed by the first row of the BIBD. The next four choice sets (only two are shown) show the same 8-run OA embedded in the partial-profile design in columns 11, 13, 14, and 16, as directed by the second row of the BIBD.

We have already seen that the choice design is statistically optimal. However, it might not be optimal from a practical usage point of view. In every choice set, there is a pair of alternatives, one with three attributes present and one with only one attribute present. This might not meet your needs (or maybe it does). You always need to inspect your designs to see how well they work for your purposes. Statistical efficiency and optimality are just one part of the picture. With the OA  $4^1 2^4$  in 8 runs, there is one other kind of OA that you can get by specifying other random number seeds, that always produces a choice set (one in each block of choice sets) where all four attributes are present in one alternative and none are present in the other alternative. This might be worse. This kind of problem is less likely to be an issue when you vary more than four attributes.

The output from the %MktDups macro is as follows:

---

```

Design:          Generic
Factors:         x1-x16
                x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16
Sets w Dup Alts: 0
Duplicate Sets:  0

```

---

There are no duplicates.

*80 Choice Sets, 16 Binary Attributes, Four Varying, Part 2*

This example is identical to the previous example except that now  $b = 12$  blocks are requested instead of 20, resulting in 48 choice sets instead of 80. The following steps create the design:

```
%mktbibd(b=12, nattrs=16, setsize=4, seed=17)

%mktx(4 2 ** 4, n=8, seed=306)

proc sort data=randomized out=randes(drop=x1);
  by x2 x1;
run;

%mktppro(ibd=bibd, design=randes)

%choicelf(data=chdes,          /* candidate set of choice sets      */
  init=chdes,                 /* initial design                    */
  initvars=x1-x6,            /* factors in the initial design     */
  model=class(x1-x16 / sta), /* model with stdz orthogonal coding */
  nsets=48,                  /* number of choice sets             */
  nalts=2,                   /* number of alternatives             */
  rscale=                     /* relative D-efficiency scale factor */
  %sysevalf(48 * 4 / 16), /* 4 of 16 attrs in 80 sets vary    */
  beta=zero)                 /* assumed beta vector, Ho: b=0     */

%mktdups(generic, data=best, factors=x1-x16, nalts=2)
```

The first part of the output from the %MktBIBD macro is as follows:

---

Block Design Efficiency Criterion	98.0066
Number of Attributes, t	16
Set Size, k	4
Number of Sets, b	12
Average Attribute Frequency	3
Average Pairwise Frequency	0.6
Total Sample Size	48
Positional Frequencies Optimized?	Yes

---

We can see that our results, an unbalanced block design, is not a BIBD since the efficiency is not 100. The attribute by attribute frequencies are as follows:

## Attribute by Attribute Frequencies

---

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	3	1	1	0	0	0	1	1	0	1	1	1	1	0	0	1
2		3	0	0	1	1	0	0	0	1	0	1	1	1	1	1
3			3	1	1	1	1	1	0	0	0	1	0	0	1	1
4				3	1	0	1	0	1	1	0	1	1	1	0	1
5					3	1	0	1	0	1	1	1	0	1	0	0
6						3	1	1	1	1	0	0	1	0	1	0
7							3	0	0	1	1	0	1	1	1	0
8								3	1	0	1	0	1	1	0	1
9									3	1	1	1	1	0	1	1
10										3	1	0	0	0	0	1
11											3	1	0	1	1	0
12												3	1	0	1	0
13													3	1	0	0
14														3	1	1
15															3	1
16																3

---

Each attribute occurs exactly 3 times, but each pair of attributes does not occur equally often. With 20 blocks, each pair occurred exactly once, so with fewer blocks, the best we can do is some zeros and some ones. The same is true for the attribute by position frequencies, which are as follows:

## Attribute by Position Frequencies

---

	1	2	3	4
1	1	1	1	1
2	1	0	1	1
3	1	1	1	0
4	1	0	1	1
5	1	1	1	0
6	0	1	1	1
7	1	1	0	1
8	1	1	0	1
9	1	0	1	1
10	1	0	1	1
11	1	1	1	0
12	0	1	1	1
13	0	1	1	1
14	0	1	1	1
15	1	1	0	1

---

The unbalanced block design is as follows:

---

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

---

The last part of the output from the %ChoiceEff macro's evaluation of the design is as follows:

---

Final Results	
Design	1
Choice Sets	48
Alternatives	2
Parameters	16
Maximum Parameters	48
D-Efficiency	12.0000
Relative D-Eff	100.0000
D-Error	0.0833
1 / Choice Sets	0.0208



n	Variable		Variance	DF	Standard Error
	Name	Label			
1	x11	x1 1	0.083333	1	0.28868
2	x21	x2 1	0.083333	1	0.28868
3	x31	x3 1	0.083333	1	0.28868
4	x41	x4 1	0.083333	1	0.28868
5	x51	x5 1	0.083333	1	0.28868
6	x61	x6 1	0.083333	1	0.28868
7	x71	x7 1	0.083333	1	0.28868
8	x81	x8 1	0.083333	1	0.28868
9	x91	x9 1	0.083333	1	0.28868
10	x101	x10 1	0.083333	1	0.28868
11	x111	x11 1	0.083333	1	0.28868
12	x121	x12 1	0.083333	1	0.28868
13	x131	x13 1	0.083333	1	0.28868
14	x141	x14 1	0.083333	1	0.28868
15	x151	x15 1	0.083333	1	0.28868
16	x161	x16 1	0.083333	1	0.28868
				==	
				16	

---

The relative  $D$ -efficiency is 100% corresponding to 4 / 16 of 48 choice sets. This design is optimal! It is optimal even though our block design is not balanced. We might not have used a real BIBD, but we did use a real OA, and that is what is critical to achieve optimality. It is optimal for a partial-profile design in 48 choices sets even though it provides less information than a design optimized for a larger number of choice sets such as 80.

The output from the %MktDups macro is as follows:

---

```

Design:          Generic
Factors:         x1-x16
                 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16
Sets w Dup Alts: 0
Duplicate Sets:  0

```

---

There are no duplicates.

Note again, however, that while this design is statistically optimal, it might not be optimal from a practical usage point of view. Besides the problems with the imbalance in the number of attributes shown in each alternative, now every attribute is no longer paired with every other attribute. These are issues you need to think about when you evaluate your designs.

*80 Choice Sets, 16 Binary Attributes, Four Varying, Part 3*

This example is identical to the previous example except that now  $b = 8$  blocks are requested instead of 12 or 20, resulting in 32 choice sets instead of 48 or 80. The following steps create and evaluate the design:

```

%mktribd(b=8, nattrs=16, setsize=4, seed=17)

%mkrtex(4 2 ** 4, n=8, seed=306)

proc sort data=randomized out=randes(drop=x1);
  by x2 x1;
run;

%mkrtpro(ibd=tribd, design=randes)

%choiceff(data=chdes,          /* candidate set of choice sets      */
  init=chdes,                 /* initial design                      */
  initvars=x1-x6,             /* factors in the initial design       */
  model=class(x1-x16 / sta), /* model with stdz orthogonal coding   */
  nsets=32,                   /* number of choice sets               */
  nalts=2,                    /* number of alternatives               */
  rscale=                      /* relative D-efficiency scale factor  */
  %sysevalf(32 * 4 / 16), /* 4 of 16 attrs in 32 sets vary      */
  beta=zero)                  /* assumed beta vector, Ho: b=0       */

%mktdups(generic, data=best, factors=x1-x16, nalts=2)

```

The attribute by attribute frequencies are as follows:

## Attribute by Attribute Frequencies

---

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	2	0	1	0	0	0	0	1	1	0	1	1	1	0	0	0
2		2	1	0	0	0	1	0	0	1	0	0	1	0	1	1
3			2	0	0	0	1	1	0	1	0	1	0	0	0	0
4				2	0	1	1	0	1	0	0	1	0	1	0	1
5					2	1	0	1	0	1	1	0	0	1	1	0
6						2	0	0	0	1	1	1	0	0	0	1
7							2	0	1	1	0	0	0	1	0	0
8								2	0	0	0	1	0	1	1	0
9									2	0	1	0	1	1	0	0
10										2	1	0	0	0	0	0
11											2	0	1	0	0	0
12												2	0	0	0	1
13													2	0	1	1
14														2	1	0
15															2	1
16																2

---

Again, we have a constant on the diagonal and ones and zeros off diagonal. Again, this design is optimal. The results are as follows:

---

## Final Results

Design	1
Choice Sets	32
Alternatives	2
Parameters	16
Maximum Parameters	32
D-Efficiency	8.0000
Relative D-Eff	100.0000
D-Error	0.1250
1 / Choice Sets	0.0313

n	Variable		Variance	DF	Standard Error
	Name	Label			
1	x11	x1 1	0.125	1	0.35355
2	x21	x2 1	0.125	1	0.35355
3	x31	x3 1	0.125	1	0.35355
4	x41	x4 1	0.125	1	0.35355
5	x51	x5 1	0.125	1	0.35355
6	x61	x6 1	0.125	1	0.35355
7	x71	x7 1	0.125	1	0.35355
8	x81	x8 1	0.125	1	0.35355
9	x91	x9 1	0.125	1	0.35355
10	x101	x10 1	0.125	1	0.35355
11	x111	x11 1	0.125	1	0.35355
12	x121	x12 1	0.125	1	0.35355
13	x131	x13 1	0.125	1	0.35355
14	x141	x14 1	0.125	1	0.35355
15	x151	x15 1	0.125	1	0.35355
16	x161	x16 1	0.125	1	0.35355
				==	
				16	

---

It is optimal for a partial-profile design in 32 choices sets even though it provides less information than designs optimized for a larger number of choice sets such as 48 or 80.

The output from the %MktDups macro is as follows:

---

```

Design:          Generic
Factors:         x1-x16
                  x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16
Sets w Dup Alts: 0
Duplicate Sets:  0

```

---

There are no duplicates.

#### *80 Choice Sets, 16 Binary Attributes, Four Varying, Part 4*

This example is identical to the previous example except that now  $b = 7$  blocks are requested instead of a multiple of 4. The following steps create the design:

```

%mktribd(b=7, nattrs=16, setsize=4, seed=17)

%mkrtex(4 2 ** 4, n=8, seed=306)

proc sort data=randomized out=randes(drop=x1);
  by x2 x1;
run;

```

```

%mktppro(ibd=bibd, design=randes)

%choicceff(data=chdes,          /* candidate set of choice sets      */
            init=chdes,         /* initial design                */
            initvars=x1-x6,     /* factors in the initial design  */
            model=class(x1-x16 / sta), /* model with stdz orthogonal coding */
            nsets=28,          /* number of choice sets         */
            nalts=2,           /* number of alternatives         */
            rscale=            /* relative D-efficiency scale factor */
            %sysevalf(28 * 4 / 16), /* 4 of 16 attrs in 28 sets vary */
            beta=zero)         /* assumed beta vector, Ho: b=0  */

%mktdups(generic, data=best, factors=x1-x16, nalts=2)

```

The attribute by attribute frequencies are as follows:

---

Attribute by Attribute Frequencies

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	2	0	0	1	0	0	0	0	1	0	1	1	1	0	1	0
2		2	0	0	0	0	1	1	0	1	0	0	1	0	1	1
3			2	0	0	1	1	0	1	0	0	1	0	1	0	1
4				1	0	0	0	0	0	0	0	1	0	0	1	0
5					1	1	0	0	0	1	1	0	0	0	0	0
6						2	0	0	0	1	1	1	0	0	0	1
7							2	0	1	1	0	0	0	1	1	0
8								1	0	0	0	0	1	0	0	1
9									2	0	1	0	1	1	0	0
10										2	1	0	0	0	1	0
11											2	0	1	0	0	0
12												2	0	0	1	1
13													2	0	0	1
14														1	0	0
15															2	0
16																2

---

The 16 attributes cannot be equally distributed across the  $7 \times 4$  entries in the incomplete block design, so the diagonal has nonconstant frequencies. The evaluation is as follows:

---

 Final Results

Design	1
Choice Sets	28
Alternatives	2
Parameters	16
Maximum Parameters	28
D-Efficiency	6.7272
Relative D-Eff	96.1024
D-Error	0.1487
1 / Choice Sets	0.0357

n	Variable Name	Label	Variance	DF	Standard Error
1	x11	x1 1	0.125	1	0.35355
2	x21	x2 1	0.125	1	0.35355
3	x31	x3 1	0.125	1	0.35355
4	x41	x4 1	0.250	1	0.50000
5	x51	x5 1	0.250	1	0.50000
6	x61	x6 1	0.125	1	0.35355
7	x71	x7 1	0.125	1	0.35355
8	x81	x8 1	0.250	1	0.50000
9	x91	x9 1	0.125	1	0.35355
10	x101	x10 1	0.125	1	0.35355
11	x111	x11 1	0.125	1	0.35355
12	x121	x12 1	0.125	1	0.35355
13	x131	x13 1	0.125	1	0.35355
14	x141	x14 1	0.250	1	0.50000
15	x151	x15 1	0.125	1	0.35355
16	x161	x16 1	0.125	1	0.35355
				==	
				16	

---

The design does not have a relative  $D$ -efficiency of 100%, so it is not optimal relative to a hypothetical, optimal partial-profile design. With seven blocks, or any  $b$  not a multiple of 4, the kind of combinatorial optimality we saw previously is not possible.

The output from the %MktDups macro is as follows:

---

Design:	Generic
Factors:	x1-x16
	x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16
Sets w Dup Alts:	0
Duplicate Sets:	0

---

There are no duplicates.

### *Partial Profiles, the General Combinatorial Approach*

In the previous sections, we saw several examples of using the %MktBIBD macro to make BIBDs, the %MktEx macro to make OAs, the %MktPPro macro to combine them into a partial-profile design, and the %ChoiceEff macro to evaluate the design. Next an ad hoc wrapper macro that combines them into a single macro for making partial-profile designs based on this combinatorial structure is shown. It works for *attr* attributes and *lev*-level factors, with *vary* varying, and  $maxv \times lev$  rows in each of the *b* blocks of choice sets. The macro is as follows:

```
%macro pp(                                /*-----*/
    lev=,                                  /* levels                                */
    nattrs=,                               /* total attributes                      */
    setsize=,                              /* number of attributes that vary       */
    maxv=,                                  /* max number of attrs that can vary   */
    b=,                                     /* number of blocks                     */
    seed=);                                /* random number seed                   */
                                        /*-----*/

%let sets = %sysevalf(&b * &maxv);
%put NOTE: nattrs=&nattrs, setsize=&setsize, sets=&sets, b=&b..;

%mktbibd(b=&b, nattrs=&nattrs, setsize=&setsize, seed=&seed, positer=0)

%mktex(&maxv &lev ** &setsize, n=&lev * &maxv)

proc sort data=randomized out=randes(drop=x1);
    by x2 x1;
run;

%mktppro(ibd=bibd, design=randes)

%choiceff(data=chdes,                    /* candidate set of choice sets        */
    init=chdes,                          /* initial design                       */
    initvars=x1-x&nattrs,                /* factors in the initial design        */
    intiter=0,                            /* evaluate without internal iterations */
    model=class(x1-x&nattrs / sta),/* model with stdz orthog coding        */
    nsets=&sets,                          /* number of choice sets                */
    nalts=&lev,                           /* number of alternatives                */
    rscale=                               /* relative D-efficiency scale factor   */
    %sysevalf(&sets * &setsize / &nattrs),
    beta=zero)                            /* assumed beta vector, Ho: b=0        */

proc print data=chdes; by set; id set; var x;; run;

%mktdups(generic, data=best, factors=x1-x&nattrs, nalts=&lev)
%mend;
```

The next steps create 25 examples of partial-profile designs. Thousands more could be made. You can use any positive integer  $< 2^{31} - 1$  for a seed.\* Note that some are subject to problems with duplicates. The following steps create the designs:

```
%pp(lev = 2, nattrs = 16, setsize = 4, maxv = 4, b = 20, seed = 17)
%pp(lev = 4, nattrs = 16, setsize = 4, maxv = 4, b = 20, seed = 93)
%pp(lev = 2, nattrs = 16, setsize = 8, maxv = 8, b = 30, seed = 109)
%pp(lev = 3, nattrs = 13, setsize = 6, maxv = 6, b = 26, seed = 114)
%pp(lev = 2, nattrs = 23, setsize = 12, maxv = 12, b = 23, seed = 121)
%pp(lev = 5, nattrs = 11, setsize = 5, maxv = 5, b = 11, seed = 145)
%pp(lev = 3, nattrs = 19, setsize = 9, maxv = 9, b = 19, seed = 151)
%pp(lev = 2, nattrs = 22, setsize = 7, maxv = 16, b = 22, seed = 205)
%pp(lev = 4, nattrs = 13, setsize = 3, maxv = 8, b = 26, seed = 238)
%pp(lev = 3, nattrs = 21, setsize = 5, maxv = 12, b = 21, seed = 289)
%pp(lev = 2, nattrs = 22, setsize = 8, maxv = 20, b = 11, seed = 292)
%pp(lev = 3, nattrs = 25, setsize = 9, maxv = 15, b = 25, seed = 306)
%pp(lev = 4, nattrs = 21, setsize = 5, maxv = 12, b = 21, seed = 350)
%pp(lev = 7, nattrs = 15, setsize = 7, maxv = 7, b = 15, seed = 368)
%pp(lev = 5, nattrs = 16, setsize = 10, maxv = 10, b = 16, seed = 377)
%pp(lev = 4, nattrs = 16, setsize = 8, maxv = 16, b = 10, seed = 382)
%pp(lev = 8, nattrs = 15, setsize = 8, maxv = 8, b = 15, seed = 396)
%pp(lev = 2, nattrs = 8, setsize = 2, maxv = 28, b = 28, seed = 420)
%pp(lev = 5, nattrs = 7, setsize = 6, maxv = 15, b = 7, seed = 424)
%pp(lev = 4, nattrs = 11, setsize = 6, maxv = 20, b = 11, seed = 448)
%pp(lev = 3, nattrs = 45, setsize = 12, maxv = 27, b = 45, seed = 462)
%pp(lev = 9, nattrs = 16, setsize = 4, maxv = 9, b = 20, seed = 472)
%pp(lev = 3, nattrs = 15, setsize = 5, maxv = 30, b = 21, seed = 495)
%pp(lev = 7, nattrs = 9, setsize = 3, maxv = 14, b = 12, seed = 513)
%pp(lev = 5, nattrs = 13, setsize = 4, maxv = 20, b = 13, seed = 522)
```

It is instructive to see how these parameters are set, particularly `maxv=` and `rep` parameters. In the first example, there are 16 two-level attributes, and four vary at a time. This corresponds to a BIBD with  $t = 14$  and  $k = 4$ . We can run the `%MktBSize` macro with these specifications as follows:

```
%mktbsize(nattrs=16, setsize=4)
```

We see that a BIBD is available with each attribute appearing  $r = 5$  times. This is the `rep=` parameter in the ad hoc macro. Remember though, that we do not need a BIBD in order to make good partial-profile designs. We can use other specifications as well, as long as there is an OA that works with our block design. We can find all OAs  $p^k s^1$  in  $p \times s$  runs with  $k \leq s$  as follows:

---

\*These are selected from some particularly interesting OA sections.



```

%mktoth(options=parent, maxlev=144)

data x(keep=n design);
  set mktdeslev;
  array x[144];
  c = 0; one = 0; k = 0;
  do i = 1 to 144;
    c + (x[i] > 0); /* how many differing numbers of levels */
    if x[i] > 1 then do; p = i; k = x[i]; end; /* p^k */
    if x[i] = 1 then do; one + 1; s = i; end; /* s^1 */
  end;
  if c = 1 then do; c = 2; one = 1; s = p; k = p - 1; end;
  if c = 2 and one = 1 and k > 2 and s * p = n;
  design = compbl(left(design));
run;

proc print noobs; by n; id n; run;

```

A few of the smaller ones that work are as follows:

---

n	Design
8	2 ** 4 4 ** 1
16	2 ** 8 8 ** 1 4 ** 5
18	3 ** 6 6 ** 1
24	2 ** 12 12 ** 1
25	5 ** 6
27	3 ** 9 9 ** 1
32	2 ** 16 16 ** 1 4 ** 8 8 ** 1
36	3 ** 12 12 ** 1
40	2 ** 20 20 ** 1
45	3 ** 9 15 ** 1
48	2 ** 24 24 ** 1 4 ** 12 12 ** 1
49	7 ** 8
50	5 ** 10 10 ** 1

---

The first example used  $2^4 4^1$  in 8 runs ( $2^k s^1$  or  $2^{\text{setsize} \text{maxv}^1}$ ). Note that you can always select a subset of the available columns to vary ( $\text{vary} < k$ ), and many of the examples do precisely that.

### *Efficient but Nonoptimal Partial Profiles*

We can use the same methods to make efficient but nonoptimal partial-profile designs. With 12 attributes, six varying, and six-level factors, an OA is not possible in 36 runs. Furthermore, with 12 blocks of choice sets and 12 attributes and six varying, a BIBD is not possible either. Still, we can use an efficient factorial design and an unbalanced block design to make a partial-profile design as follows:

```
%mktbibd(b=12, nattrs=12, setsize=6, seed=93, positer=)

%mktx(6 ** 7, n=36, seed=238)

proc sort data=randomized out=randes(drop=x1);
  by x2 x1;
run;

%mktppro(ibd=bibd, design=randes)

%choicelf(data=chdes,          /* candidate set of choice sets      */
  init=chdes,                 /* initial design                    */
  initvars=x1-x12,           /* factors in the initial design     */
  intiter=0,                  /* evaluate without internal iterations */
  model=class(x1-x12 / sta), /* model with stdz orthogonal coding  */
  nsets=72,                   /* number of choice sets             */
  nalts=6,                    /* number of alternatives             */
  rscale=                      /* relative D-efficiency scale factor */
  %sysevalf(72 * 6 / 12), /* 6 of 12 attrs in 72 sets vary    */
  beta=zero)                  /* assumed beta vector, Ho: b=0     */

proc print; by set; id set; var x1-x12; run;

%mktdups(generic, data=best, factors=x1-x12, nalts=6)
```

# Conclusions

This very long chapter has several purposes. It provides an introduction to the multinomial logit model used in choice modeling, it discusses linear model designs created by the `%MktEx` macro and their use in choice modeling, and it discusses using the `%ChoiceEff` macro to find and evaluate designs. These two macros, along with all of the other macros, provide powerful tools for designing choice experiments in a variety of ways. Numerous other macros and techniques are discussed as well.