The %ChoicEff Macro

Introduction

The %ChoicEff autocall macro finds efficient experimental designs for choice experiments and evaluates choice designs. You supply a set of candidates, and the macro searches the candidates for an efficient experimental design. An efficient design is defined as a design for which the variances of the parameter estimates are minimized, given an assumed parameter vector $\boldsymbol{\beta}$.

You can use the %ChoicEff macro in the following four ways:

1. You create a candidate set of alternatives, and the %ChoicEff macro creates a design that consists of choice sets built from the alternatives that you supply. For each candidate alternative, you designate the design alternatives for which it is a candidate. For a generic design, you create one list of candidate alternatives, and each candidate can be used for every alternative in the design. For a branded study of *m* brands, you create a list of *m* types of candidate alternatives, one for each brand.

2. You create a candidate set of choice sets, and the %ChoicEff macro builds a design from those choice sets. This approach is designed to handle restrictions across alternatives (certain alternatives cannot appear with certain other alternatives) and in partial-profile designs. However, the candidate set of alternatives approach along with the RESTRICTIONS= argument is often better than this approach. This is because for all but the smallest designs, the candidate set of choice sets approach considers much smaller subsets of possible designs. Unless it is much easier for you to create a candidate set of restricted choice sets than to create a restrictions macro, you should use the RESTRICTIONS= argument and a candidate set of alternatives instead of a candidate set of choice sets.

3. You create a candidate set of alternatives and a macro that provides restrictions on how the alternatives can be used to make the design. The %ChoicEff macro creates a design that consists of choice sets built from the alternatives that you supply. For each candidate alternative, you designate the design alternatives for which it is a candidate. For a generic design, you create one list of candidate alternatives, and each candidate can be used for every alternative in the design. For a branded study of *m* brands, you create a list of *m* types of candidate alternatives, one for each brand. You can restrict the design in any way that you want (within alternatives, across alternatives and within choice sets, or across choice sets). For example, you can use the restrictions macro to prevent dominated alternatives, to force or prevent overlap in factor levels within choice sets, to prevent certain levels from occurring with other levels, to force constant attributes within choice sets, to control the number of constant attributes across choice sets, and so on.

4. You supply a choice design, and the %ChoicEff macro evaluates it. The choice design might have been created by a previous run of the %ChoicEff macro, by the %MktEx macro, or by other means.

The %ChoicEff macro uses a modified Fedorov candidate-set-search algorithm, just like the OPTEX procedure and parts of the %MktEx macro. You usually use candidate sets that consist of a full-factorial design, a fractional-factorial design, or an orthogonal array that is created by using the %MktEx macro.

The %ChoicEff macro starts by either constructing a random initial design from the candidates or using an initial design that you specify. The macro considers swapping out every design alternative and replacing it with each candidate alternative. Swaps that increase efficiency are performed. The process of evaluating and swapping continues until efficiency stabilizes at a local maximum. This process is repeated using different initial designs, and the best design is output for use.

The key differences between the %ChoicEff macro and the %MktEx macro are as follows:

- The %ChoicEff macro requires you to specify the true (or assumed true) parameters, and it optimizes the variance matrix for a multinomial logit discrete choice model, which is a nonlinear model.
- The %MktEx macro optimizes the variance matrix for a linear model, which does not depend on the parameters.

%ChoicEff Macro Syntax

%ChoicEff(MODEL=*model-specification*, **NSETS=***n* <, *optional arguments*>**)**

Required Arguments

MODEL=*model-specification*

specifies a PROC TRANSREG MODEL statement, which lists the attributes and describes how they are coded. There are many potential forms for the model specification and a number of options (see the PROC TRANSREG documentation in the *SAS/STAT User's Guide*). The PROC TRANSREG option STANDORTH, which requests standardized orthogonal contrast coding, is often useful in this macro. For some designs, if you use the STANDORTH option in the model specification along with the %ChoicEff macro argument OPTIONS=RELATIVE, you can get a relative *D*-efficiency in the range of 0 to 100. The following are examples of common model specifications.

The following MODEL= argument specifies generic effects:

```
model=class(x1-x3)
```

The following MODEL= argument specifies brand and alternative-specific effects:

```
model=class(b)
      class(b*x1 b*x2 b*x3 / effects zero=' ')
```

The following MODEL= argument specifies brand, alternative-specific, and cross effects:

```
model=class(b b*p / zero=' ')
      identity(x1-x5) * class(b / zero=none)
```

NSETS=*n*
specifies the number of choice sets that you want.

You must specify exactly one of the next two arguments. When the candidate set consists of individual alternatives to be swapped, specify the alternative flags by using the FLAGS= argument. When the candidate set consists of entire sets of alternatives to be swapped, specify the number of alternatives in each set by using the NALTS= argument.

FLAGS=*variable-list* | *number-of-alternatives*
specifies variables that flag the alternatives for which each candidate can be used. You must have one flag variable per alternative. If every candidate can be used in all alternatives, then the flags are constant. When the flags are all constant (in a purely generic design), you can request that the macro create the flag variables for you by specifying the number of alternatives rather than specifying a list of flag variables.

NALTS=*n*
specifies the number of alternatives in each choice set.

Optional Arguments

BESTCOV=*SAS-data-set*
specifies a name for the data set to contain the covariance matrix for the best design. By default, this data set is called BestCov.

BESTOUT=*SAS-data-set*
specifies a name for the data set to contain the best design. By default, this data set is called Best.

BETA=*list* | **ZERO**
specifies the values of the true parameters. Specify BETA=ZERO to set all the parameters equal to 0. You must specify BETA= to create a design. By default, when you do not specify BETA=, the macro reports only on coding.

CHUNKS=*n*
specifies the number of observations to process at one time by using the coding step and PROC TRANSREG. By default, the entire data set is processed at once. You can specify a value—such as one-half or one-third of the number of choice sets times the number of alternatives—to break up the coding into smaller chunks if you run out of memory. Ideally, you should make the value a multiple of the number of choice sets. Be sure not to leave one or a few extra observations in the last chunk, particularly if you are using one of the orthogonal codings, or the macro will generate an error. You usually do not need to specify this argument.

CONVERGE=*n*
specifies the *D*-efficiency convergence criterion. By default, CONVERGE=0.005.

COV=*SAS-data-set*
specifies a name for the data set to contain all the covariance matrices for all the designs. By default, this data set is called Cov.

DATA=*SAS-data-set*
specifies the data set to contain the choice candidate set. By default, the macro uses the last data set that is created.

DROP=*variable-list*
specifies a list of variables to drop from the model. If you specify a less-than-full-rank model in the MODEL= argument, you can use the DROP= argument to produce a full-rank coding. When you have redundant variables, the macro displays a list in the SAS log that you can use in the DROP= argument in a subsequent run.

FIXED=*variable-list*
specifies the variable that flags the fixed alternatives. You can use the FIXED= argument only when both the INIT= and INITVARS= arguments are specified. When you specify FIXED=*variable*, the INIT= data set must contain the named variable, which indicates which alternatives are fixed (cannot be swapped out) and which ones can be changed. Values of the FIXED= variable include the following:

- 1, which means that this alternative can never be swapped out
- 0, which means that this alternative is used in the initial design but can be swapped out
- ., which means that this alternative should be randomly initialized and can be swapped out

INIT=*SAS-data-set*
specifies an initial design data set. Null means a random start. One usage is to specify the BESTOUT= data set for an initial start. When you specify the FLAGS= argument, INIT= must contain the index variable; for example, INIT=BEST(KEEP=INDEX). When you specify the NALTS= argument, INIT= must contain the choice set variable; for example, INIT=BEST(KEEP=SET).

Alternatively, the INIT= data set can contain an arbitrary design, potentially created outside this macro. In that case, you must also specify INITVARS=*factors*, where *factors* are the factors in the design; for example, INITVARS=x1-x3. When alternatives are swapped, this data set must also contain the FLAGS= variables. When you specify INIT= together with INITVARS=, the data set can also contain a variable that is specified in the FIXED= argument, which indicates which alternatives are fixed and which ones can be swapped in and out.

INITITER=*n*
specifies the maximum number of internal iterations. Specify INTITER=0 to evaluate the efficiency of an existing design. By default, INTITER=10.

INITVARS=*variable-list*
specifies the factor variables in the INIT= data set that must match up with the variables in the DATA= data set. (See the INIT= argument.) All these variables must be of the same type.

MAXITER=*n*
ITER=*n*
specifies the maximum iterations (designs to create). By default, MAXITER=10.

MOREVARS=*variable-list*
specifies additional variables to add to the model. This argument enables you to specify a list of variables to copy as is, using the PROC TRANSREG coding, and then add them to the model.

N=*n*

specifies the number of observations to use in the variance matrix formula. By default, N=1.

OPTIONS=*options-list*

specifies binary arguments. By default, none of these arguments are specified. You can specify one or more of the following values.

CODED

displays the coded candidate set.

DETAIL

displays the details of the swaps and any restriction violations. This argument adds more information to the iteration history tables than is displayed by default. You can use OPTIONS=RESREP as an alias for OPTIONS=DETAIL. The former is the name of the argument in the %MktEx macro that provides a report on restriction violations and conformance. It is a good idea to specify this argument with restrictions until you are sure that your restrictions macro is correct.

NOBESTSTAR

specifies not to print an asterisk when a better design is found. By default, an asterisk is printed in the iteration history table whenever a design is found whose *D*-efficiency is greater than the previous best.

NOCODE

skips the PROC TRANSREG coding stage, assuming that WORK.TMP_CAND was created by a previous step. This is most useful in set swapping when the candidate set can be large. It is important to note that when you specify OPTIONS=NOCODE, the effect of specifying the MOREVARS= and DROP= arguments in previous runs has already been incorporated, so do not specify MOREVARS= and DROP= unless you want to drop additional variables.

NODUPS

prevents the creation of duplicate choice sets. This argument does not affect the initialization, so the random initial design might have duplicates. This argument eliminates duplicates during the iterations, so do not set the INTITER= argument to a small value. It might take several iterations to eliminate all duplicates. It is possible that efficiency will decrease as duplicates are eliminated. With set swapping, this macro checks the candidate choice set numbers to avoid duplicates. With alternative swapping, this macro checks the candidate alternative index to avoid duplicates. The macro does not look at the actual factors. This makes the checks faster, but if the candidate set contains duplicate choice sets or alternatives, the macro might not succeed in eliminating all duplicates. You can use the %MktDups macro (which looks at the actual factors) on the design to ensure that all duplicates are eliminated. If you are using set swapping to make a generic design, you can use the %MktDups macro on the candidate set to eliminate duplicate choice sets in advance.

NOTESTS

suppresses the display of the diagonal of the covariance matrix and the display of hypothesis test results for this n and $\boldsymbol{\beta}$. When $\boldsymbol{\beta}$ is not 0, the results include a Wald test statistic (β divided by the standard error), which is normally distributed, and the probability of a larger squared Wald statistic.

ORTHCAN
orthogonalizes the candidate set.

OUTPUTALL
outputs all designs to the OUT= and COV= data sets. When the MAXITER= argument value is less than or equal to 100, this argument is the default. When the MAXITER= value is greater than 100, only designs that improve on the previous best design are output by default.

RELATIVE
displays the relative *D*-efficiency for the final design, which is 100 times the *D*-efficiency divided by the number of choice sets. In other words, this argument scales the *D*-efficiency relative to a (perhaps hypothetical) design whose *D*-efficiency is equal to the number of choice sets.

OUT=*SAS-data-set*
specifies a name for the output SAS data set that contains all the final designs. By default, OUT=Results.

RESTRICTIONS=*macro-name*
specifies the name of a restrictions macro, written in the SAS/IML matrix language, that quantifies the badness of the design in an IML scalar that must be called Bad. By default, there are no restrictions. When you specify a RESTRICTIONS= macro, you must also specify the RESVARS= argument.

RESVARS=*variable-list*
specifies the variables for restrictions. These variables must all be numeric. The RESVARS= argument variables are available for your restrictions macro in the following matrices:

- **Xmat**, which contains the current design
- **X**, which contains the choice set that is being considered for the SETNUM position (the current choice set number) in **Xmat**

When restrictions are written in terms of the entire design, the statements in the restrictions macro might have the following form:

```
do s = 1 to nsets;
   if s ^= setnum then z = xmat[((s - 1) * nalts + 1) : (s * nalts),];
   else z = x;
   ... evaluate choice set z and accumulate badness ...
   end;
```

The index vector `((s - 1) * nalts + 1) : (s * nalts)` (which is used as the row index in **Xmat**) extracts one choice set. For example, with three alternatives, the index vector is `1:3` when `s = 1`, and it extracts the first choice set from **Xmat**; the index vector is `4:6` when `s = 2`, and it extracts the second choice set from **Xmat**; the index vector is `7:9` when `s = 3`; and so on.

The submatrix `xmat[((setnum - 1) * nalts + 1) : (setnum * nalts),]` contains the Setnum choice set that is currently in the design, and **X** contains the candidate choice set that is being evaluated. The submatrix `xmat[((setnum - 1) * nalts + 1) : (setnum * nalts),]` is replaced by **X** when the replacement increases efficiency or decreases restriction violations.

The first RESVARS= variable is in the first column of **X** and **Xmat** (`x[,1]` and `xmat[,1]`), and the last RESVARS= variable is in the last column of **X** and **Xmat** (`x[,m]` and `xmat[,m]`), where `m =`

`ncol(xmat)`. The RESVARS= variables are usually the attributes, but they can also contain additional information. All restrictions must be written in terms of the values of **X** and **Xmat** along with the following:

- Nsets – number of choice sets in the design
- Nalts – number of alternatives in the design
- Setnum – number of current choice set
- Altnum – number of current alternative (available only with alternative swapping)

RSCALE=*r* | **GENERIC** | **ALT** | **PARTIAL=***p* **OF** *q*

specifies the scaling factor to use for relative *D*-efficiency computations. When you specify RSCALE=, the argument OPTIONS=RELATIVE is implied. By default, when you do not specify this argument, the number of choice sets is used when OPTIONS=RELATIVE is specified.

r

(where *r* is some number) indicates that relative *D*-efficiency equals 100 × *D*-efficiency / *r*. If you want relative *D*-efficiency and you know that the number of choice sets is not the right scaling factor (perhaps because you have a constant alternative), and if you know the *D*-efficiency of an optimal design, you can specify it to get relative *D*-efficiency. *r* must be a number and not an expression. However, you can use the %SYSEVALF function to evaluate an expression (for example, RSCALE=%SYSEVALF(16 * 4 / 8)).

GENERIC

(with *n* choice sets) is equivalent to RSCALE=*n*.

ALT

(with n choice sets, m alternatives, and p parameters, and $r = (n^{m-1}(n(m-1)/m^2)^{p-m+1})^{1/p}$) is equivalent to RSCALE=*r*. If a design has brand (or alternative label) effects such that brand i always occurs in alternative i, and all other effects are alternative-specific, then there are $m-1$ parameters with a maximum determinant of n, the remaining $p-m+1$ have 1 of m alternatives that contribute information to each set, and $m-1$ of m alternatives contribute information so the maximum *D*-efficiency is $n(m-1)/m^2$ for the $p-m+1$ parameters in that part of the design. This formula does not provide a true maximum for more complicated designs, such as designs that have constant alternatives.

PARTIAL=*p* **OF** *q*

(where *p* and *q* are integers) is used with partial-profile designs (where *p* of *q* attributes vary) or with a generic choice design that has a constant alternative (where *p* of *q* alternatives vary). It sets RSCALE= to np/q. For example, for 16 choice sets and 4 of 8 attributes that vary, RSCALE=PARTIAL=4 of 16 is equivalent to RSCALE=%SYSEVALF(16 * 4 / 8) and RSCALE=8.

SEED=*n*

specifies the random number seed. By default, SEED=0, and clock time is used to make the random number seed.

SUBMAT=*number-list*

specifies a submatrix for which efficiency calculations are desired. Specify an index vector. For example, with three 3-level factors, a, b, and c, and the model **`CLASS(a b c a*b)`**, specify SUBMAT=1:6 to see the efficiency of just the 6×6 matrix of main effects. Specify SUBMAT=3:6 to see the efficiency of just the 4×4 matrix of b and c main effects.

TYPES=*integer-list*

specifies the number of sets of each type to put into the design. This argument is used when you have multiple types of choice sets and you want the design to consist of only certain numbers of each type. The argument is an integer list. When you specify TYPES=, you must also specify the TYPEVAR= argument. For example, suppose you are creating a design that contains 30 choice sets, and you want the first 10 sets to be sets whose TYPEVAR= variable in the candidate set is type 1, and you want the rest to be type 2. The correct specification is TYPES=10 20.

TYPEVAR=*variable*

specifies a variable in the candidate data set that contains choice set types. The types must be integers, starting with 1. When you specify TYPEVAR=, you must also specify the TYPES= argument.

WEIGHT=*weight-variable*

specifies an optional weight variable. You usually use this argument with an availability design where you give a weight of zero to unavailable alternatives and a weight of one to available alternatives. The number of alternatives must always be constant, so varying numbers of alternatives are handled by giving a weight of zero to unavailable or unseen alternatives.

Help Option

You can specify either of the following to display the option names and simple examples of the macro syntax:

```
%choiceff(help)
%choiceff(?)
```

%ChoicEff Macro Notes

This macro specifies **`options nonotes`** throughout most of its execution. If you want to see all the notes, submit the following statement before running the macro:

```
%let mktopts = notes;
```

To see the macro version, submit the following statement before running the macro:

```
%let mktopts = version;
```

Example 1: Constructing a Generic Choice Design from Candidate Alternatives

This example creates a design for a generic choice model that contains three 3-level factors. First, you use the %MktEx macro to create a set of candidate alternatives, where X1, X2, and X3 are the factors. The following

statements create and display the candidate set:

```
%mktex(3 ** 3, n=3**3, seed=238)
```

The %MktEx macro, by default, saves the candidate alternatives in a data set named Design. The following statements use the PRINT procedure to print the candidate alternatives:

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

Figure 1 displays the candidate alternatives.

Figure 1 Design Matrix

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

Next, you use the %ChoicEff macro to find an efficient design for the unbranded, purely generic choice model with the assumption that $\boldsymbol{\beta} = \mathbf{0}$ as follows:

```
%choiceff(data=design, model=class(x1-x3 / standorth), nsets=9, flags=3,
          seed=289, maxiter=60, options=relative, beta=zero)
```

The DATA= argument indicates that the design matrix is contained in the data set Design. The MODEL= argument specifies the MODEL statement in PROC TRANSREG for coding the design. **`CLASS(x1-x3 / STANDORTH)`** indicates that there are three main effects, and the STANDORTH option specifies a standardized

orthogonal coding. The NSETS= argument specifies that there are nine choice sets. OPTIONS=RELATIVE requests that the relative *D*-efficiency be displayed. The FLAGS= argument specifies that there are three alternatives in a purely generic design. BETA=ZERO specifies that all model parameters are 0. The SEED= argument specifies the random number seed. The MAXITER= argument requests that 60 designs be created.

The output from the %ChoiceEff macro, as shown in Figure 2, includes a list of the model parameters' names, values, and labels; a series of iteration histories (not shown); a brief report on the most efficient design found; and a table that shows the model parameters' names, variances, degrees of freedom, and standard errors.

Figure 2 Selected Output from the %ChoicEff Macro

n	Name	Beta	Label
1	**x11**	0	x1 1
2	**x12**	0	x1 2
3	**x21**	0	x2 1
4	**x22**	0	x2 2
5	**x31**	0	x3 1
6	**x32**	0	x3 2

Final Results

Design	52
Choice Sets	9
Alternatives	3
Parameters	6
Maximum Parameters	18
D-Efficiency	9.0000
Relative D-Eff	100.0000
D-Error	0.1111
1 / Choice Sets	0.1111

n	Variable Name	Label	Variance	DF	Standard Error
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
				6	

By default, the %ChoiceEff macro saves the best design in a data set named Best and the covariance matrix of the best design in a data set named BestCov. The following statements use PROC PRINT to display the best design and its covariance matrix:

```
proc print data=best;
   var x1-x3;
   id set;
   by set;
run;
```

```
proc print data=bestcov label;
   id __label;
   label __label = '00'x;
   var x:;
run;
```

The output is shown in Figure 3 and Figure 4.

Figure 3 Choice Design Matrix

Set=1

Set	x1	x2	x3
1	3	1	3
	1	3	1
	2	2	2

Set	x1	x2	x3
2	2	2	3
	1	3	2
	3	1	1

Set	x1	x2	x3
3	3	2	2
	2	1	1
	1	3	3

Set	x1	x2	x3
4	2	1	3
	1	2	2
	3	3	1

Set	x1	x2	x3
5	1	2	1
	2	1	2
	3	3	3

Set	x1	x2	x3
6	3	1	2
	1	2	3
	2	3	1

Set	x1	x2	x3
7	3	2	1
	2	3	3
	1	1	2

Set	x1	x2	x3
8	1	1	3
	3	3	2
	2	2	1

Figure 3 *continued*

Set	x1	x2	x3
9	1	1	1
	3	2	3
	2	3	2

Figure 4 Covariance Matrix

	x1 1	x1 2	x2 1	x2 2	x3 1	x3 2
x1 1	0.11111	0.00000	-0.00000	0.00000	0.00000	0.00000
x1 2	0.00000	0.11111	0.00000	-0.00000	0.00000	-0.00000
x2 1	-0.00000	0.00000	0.11111	0.00000	0.00000	0.00000
x2 2	0.00000	-0.00000	0.00000	0.11111	0.00000	-0.00000
x3 1	0.00000	0.00000	0.00000	0.00000	0.11111	0.00000
x3 2	0.00000	-0.00000	0.00000	-0.00000	0.00000	0.11111

This design is optimal because it has 100% relative *D*-efficiency. Because a generic (main-effects-only) design is requested in which $\boldsymbol{\beta} = \mathbf{0}$, and the standardized orthogonal contrast coding is used, it is possible to get a relative *D*-efficiency on a 0-to-100 scale. This is not necessarily true for all models. Relative *D*-efficiency is computed as the raw *D*-efficiency divided by the number of choice sets, multiplied by 100. An optimal generic design such as this one exhibits all the following properties:

D-efficiency equals the number of choice sets.

D-error equals 1 over the number of choice sets.

All the variances equal 1 over the number of choice sets.

All the covariances are 0.

Relative *D*-efficiency equals 100.

NOTE: In practice, the values of the covariances are often not precisely 0 because the values are computed by using inexact floating-point arithmetic. This is why some values are displayed as –0.00000 in the output.

Example 2: Constructing a Generic Choice Design from Candidate Choice Sets

This example uses the %MktEx, %MktRoll, and %MktDups macros to create a candidate set of choice sets. The %ChoicEff macro is then used to search for an efficient design by using the candidate-set-swapping algorithm. The %ChoicEff macro can sometimes find a 100% *D*-efficient generic choice design by using this candidate set of choice sets approach. However, the optimal design is much harder to find when the macro is searching through a large candidate set of choice sets instead of searching through a small candidate set of alternatives. This is one reason why the candidate set of alternatives approach is usually preferred to creating a candidate set of choice sets.

The following three statements create a candidate set of choice sets:

```
%mktex(3 ** 9, n=2187, seed=368)

%mktroll(design=Design, key=3 3, out=Rolled)

%mktdups(generic, data=Rolled, factors=x1-x3, nalts=3, out=NoDups)
```

The %MktEx macro creates a design that has nine factors, three for each of the three alternatives. The %MktRoll macro turns a linear arrangement of a choice design into a true choice design by using the rules that are specified in the Key data set. In this example, the Key data set is automatically created to arrange one row with nine factors into three alternatives with three attributes. The %MktDups macro removes duplicate cadidate choice sets from the data set Rolled and saves the resulting candidate choice sets in the data set NoDups.

Figure 5 shows the first three candidate choice sets that are contained the the data set NoDups.

Figure 5 First Three Candidate Choice Sets

Set	_Alt_	x1	x2	x3
4	1	1	1	1
4	2	1	1	2
4	3	1	1	3
5	1	1	1	1
5	2	1	1	2
5	3	2	3	1
6	1	1	1	1
6	2	1	1	2
6	3	3	2	2

The following invocation of the %ChoicEff macro attempts to construct an efficient design from the candidate choice sets:

```
%choiceff(data=NoDups, model=class(x1-x3 / standorth), nsets=9, nalts=3,
          maxiter=20, seed=205, options=relative, beta=zero)
```

The DATA= argument specifies that the candidate choice sets are contained in the data set NoDups. The MODEL= argument specifies the MODEL statement in PROC TRANSREG. The NSETS= argument requests a design that contains nine choice sets. NALTS=3 specifies that there are three alternatives. The MAXITER= argument requests that 20 designs be created. The SEED argument specifies the seed for the pseudorandom number generator, which ensures reproducibility. OPTIONS=RELATIVE requests that the relative *D*-efficiency for the final design be displayed. BETA=ZERO specifies that all the model parameters be set equal to 0.

By default, the %ChoicEff macro saves the best design in a data set named Best, which you can display by using the PRINT procedure, as follows:

```
proc print data=Best;
  var set _alt_ x1-x3;
run;
```

The output is shown in Figure 6.

Figure 6 Best Generic Choice Design

Obs	Set	_Alt_	x1	x2	x3
1	1048	1	2	3	3
2	1048	2	1	1	2
3	1048	3	3	2	1
4	399	1	1	2	3
5	399	2	3	3	2
6	399	3	2	1	1
7	707	1	2	1	2
8	707	2	3	3	3
9	707	3	1	2	1
10	1234	1	3	1	3
11	1234	2	2	2	2
12	1234	3	1	3	1
13	1314	1	3	2	2
14	1314	2	1	1	3
15	1314	3	2	3	1
16	486	1	1	3	2
17	486	2	2	2	1
18	486	3	3	1	3
19	397	1	1	2	3
20	397	2	3	3	1
21	397	3	2	1	2
22	1047	1	2	3	3
23	1047	2	1	1	1
24	1047	3	3	2	2
25	489	1	1	3	2
26	489	2	2	2	3
27	489	3	3	1	1

Example 3: Imposing Restrictions on Choice Designs

Suppose you are designing a choice experiment in which there are six 4-level quantitative attributes. As the factor level value increases, the desirability of the feature increases. When one alternative contains levels that are all less than or equal to the levels for another alternative, the first alternative is dominated by the second. When one alternative is dominated by another, the choice task becomes easier for respondents. Eliminating dominated alternatives forces the respondents to consider all the attributes and all the alternatives in making a choice. This example shows how to use the %ChoiceEff macro to generate a design that avoids dominated alternatives by imposing restrictions on the design. You impose restrictions on a design by writing a restrictions macro that uses the SAS/IML matrix language and that specifies the restrictions in terms of a set of predefined IML matrices and scalars (see the documentation for the REVARS= argument for more details).

First you generate a candidate set of alternatives by using the %MktEx macro:

```
%mktex(4 ** 6, n=32, seed=104)
```

By default, the %MktEx macro saves the randomized experimental design in a data set named Randomized. This is your candidate set of alternatives.

Next, you write a macro that counts the number of dominated alternatives. You write this macro in the SAS/IML matrix language. The IML scalar Bad is increased by 1 every time a dominated alternative is found. The scalar Bad is automatically initialized to 0. In this example, the restrictions macro is evaluating each choice set at the same time that the %ChoicEff macro is constructing it. The current choice set that is being considered is stored in the matrix **X**. When every element in the *i*th row of **X** is greater than or equal to its corresponding element in the *k*th row of **X**, the *i*th row dominates the *k*th row and Bad is increased by 1. Similarly, when every element in the *k*th row of **X** is greater than or equal to its corresponding element in the *i*th row of **X**, the *k*th row dominates the *i*th row and Bad is increased by 1.

```
%macro res;
   do i = 1 to nalts;
      do k = i + 1 to nalts;
         if all(x[i,] >= x[k,]) then bad = bad + 1;
         if all(x[k,] >= x[i,]) then bad = bad + 1;
      end;
   end;
%mend;
```

Finally, you invoke the %ChoicEff macro and provide the name of your restrictions macro in the RESTRICTIONS= argument. You must also provide the names of the variables that the design contains by using the RESVARS= argument. The %ChoicEff macro uses the %Res macro to evaluate the badness of each choice set when it considers swapping alternatives into the design.

```
%choiceff(data=randomized, model=class(x1-x6 / sta), nsets=8, flags=4,
          seed=104, options=relative, restrictions=res, resvars=x1-x6,
          maxiter=20, beta=zero)
```

Figure 7 shows that this construction method creates a design that is 91% efficient relative to the optimal design that has no restrictions. You can often find a slightly more efficient design by increasing the number of designs that are specified in the MAXITER= argument, but the gain in efficiency is subject to diminishing returns.

Figure 7 Selected Output from the %ChoicEff Macro

n	Name	Beta	Label
1	x11	0	x1 1
2	x12	0	x1 2
3	x13	0	x1 3
4	x21	0	x2 1
5	x22	0	x2 2
6	x23	0	x2 3
7	x31	0	x3 1
8	x32	0	x3 2
9	x33	0	x3 3
10	x41	0	x4 1
11	x42	0	x4 2
12	x43	0	x4 3
13	x51	0	x5 1
14	x52	0	x5 2
15	x53	0	x5 3
16	x61	0	x6 1
17	x62	0	x6 2
18	x63	0	x6 3

Final Results

Design	15
Choice Sets	8
Alternatives	4
Parameters	18
Maximum Parameters	24
D-Efficiency	7.2649
Relative D-Eff	90.8108
D-Error	0.1376
1 / Choice Sets	0.1250

Figure 7 *continued*

n	Variable Name	Label	Variance	DF	Standard Error
1	x11	x1 1	0.13140	1	0.36249
2	x12	x1 2	0.14255	1	0.37755
3	x13	x1 3	0.16634	1	0.40785
4	x21	x2 1	0.13183	1	0.36309
5	x22	x2 2	0.12435	1	0.35263
6	x23	x2 3	0.16369	1	0.40459
7	x31	x3 1	0.14167	1	0.37640
8	x32	x3 2	0.15253	1	0.39055
9	x33	x3 3	0.12718	1	0.35663
10	x41	x4 1	0.14989	1	0.38716
11	x42	x4 2	0.13142	1	0.36252
12	x43	x4 3	0.14708	1	0.38351
13	x51	x5 1	0.15714	1	0.39640
14	x52	x5 2	0.14356	1	0.37889
15	x53	x5 3	0.14336	1	0.37863
16	x61	x6 1	0.14827	1	0.38506
17	x62	x6 2	0.15225	1	0.39019
18	x63	x6 3	0.14319	1	0.37840
				18	

The following statements assign names and levels for the attributes and display the design:

```
proc format;
   value x1f 1='Bad'   2='Good'    3='Better'   4='Best';
   value x2f 1='Small' 2='Average' 3='Bigger'   4='Large';
   value x3f 1='Ugly'  2='OK'      3='Average'  4='Nice ';
   value x4f 1='Slow'  2='Fast'    3='Faster'   4='Fastest';
   value x5f 1='Rough' 2='Normal'  3='Smoother' 4='Smoothest';
   value x6f 1='$9.99' 2='$8.99'   3='$7.99'    4='$6.99';
run;
proc print data=best label;
   label x1 = 'Quality'
         x2 = 'Size'
         x3 = 'Appearance'
         x4 = 'Speed'
         x5 = 'Smoothness'
         x6 = 'Price';
   format x1 x1f. x2 x2f. x3 x3f. x4 x4f. x5 x5f. x6 x6f.;
   by set;
   id set;
   var x:;
run;
```

The attribute levels are assigned so that in terms of the original values (1, 2, 3, 4), larger values are always better than smaller values. In particular, notice that the largest price is assigned to the smallest level (1 becomes $9.99) and the smallest price is assigned to the largest level (4 becomes $6.99).

Figure 8 shows the final design.

Figure 8 Final Design

Set=1

Set	Quality	Size	Appearance	Speed	Smoothness	Price
1	Bad	Average	Ugly	Slow	Smoothest	$7.99
	Good	Large	Average	Fastest	Rough	$9.99
	Better	Small	Nice	Faster	Rough	$9.99
	Good	Large	OK	Fast	Normal	$6.99

Set	Quality	Size	Appearance	Speed	Smoothness	Price
2	Bad	Average	Nice	Faster	Smoother	$8.99
	Best	Bigger	OK	Fast	Smoothest	$7.99
	Better	Large	Average	Slow	Smoothest	$8.99
	Good	Small	Ugly	Fastest	Smoother	$7.99

Set	Quality	Size	Appearance	Speed	Smoothness	Price
3	Good	Bigger	Ugly	Faster	Normal	$8.99
	Bad	Small	OK	Fastest	Smoothest	$9.99
	Best	Large	Nice	Slow	Smoother	$6.99
	Better	Average	Average	Fast	Rough	$7.99

Set	Quality	Size	Appearance	Speed	Smoothness	Price
4	Better	Large	OK	Faster	Smoother	$7.99
	Best	Average	Ugly	Fastest	Rough	$6.99
	Good	Small	Nice	Fast	Smoothest	$8.99
	Bad	Bigger	Average	Slow	Normal	$9.99

Set	Quality	Size	Appearance	Speed	Smoothness	Price
5	Better	Bigger	Ugly	Fast	Smoother	$9.99
	Best	Small	OK	Slow	Rough	$8.99
	Bad	Large	Nice	Fastest	Normal	$7.99
	Good	Average	Average	Faster	Smoothest	$6.99

Set	Quality	Size	Appearance	Speed	Smoothness	Price
6	Good	Bigger	Nice	Slow	Rough	$7.99
	Better	Average	OK	Fastest	Normal	$8.99
	Best	Large	Ugly	Faster	Smoothest	$9.99
	Bad	Small	Average	Fast	Smoother	$6.99

Set	Quality	Size	Appearance	Speed	Smoothness	Price
7	Good	Average	OK	Slow	Smoother	$9.99
	Good	Small	Nice	Fast	Smoothest	$8.99
	Best	Small	Average	Faster	Normal	$7.99
	Bad	Bigger	OK	Faster	Rough	$6.99

Set	Quality	Size	Appearance	Speed	Smoothness	Price
8	Best	Bigger	Average	Fastest	Smoother	$8.99
	Bad	Large	Ugly	Fast	Rough	$8.99
	Best	Average	Nice	Fast	Normal	$9.99
	Better	Small	Ugly	Slow	Normal	$6.99

Example 4: Evaluating a Generic Choice Design

This example shows how to use the %ChoicEff macro to evaluate an existing design. The design might have been created by a previous invocation of the %ChoicEff macro, by the %MktEx macro, or by other means. The following DATA step reads a data set that contains a generic choice design with three 3-level factors. There are three choice sets, indexed by the variable Set. The three factors are Size, Color, and Price.

```
data Design;
   input Set Size Color Price ;
   datalines;
1 1 1 1
1 2 3 2
1 3 2 3
2 1 3 3
2 2 2 1
2 3 1 2
3 1 2 2
3 2 1 3
3 3 3 1
;
```

The following invocation of the %ChoicEff macro evaluates the design:

```
%choiceff(data=Design, init=Design(keep=Set), intiter=0,
          model=class(Size Color Price / sta), nsets=3,
          nalts=3, options=relative, beta=zero)
```

When you evaluate a design, you provide the name of the data set that contains the design in the DATA= argument. INIT=DESIGN(KEEP=SET) instructs the %ChoicEff macro to select all the choice sets in the data set Design that are indexed by the variable Set and use them as the initial design. INTITER=0 instructs the macro to evaluate the efficiency of the initial design. The MODEL= argument specifies the MODEL statement in PROC TRANSREG. The NSETS= argument specifies that the design contains three choice sets. The NALTS= argument specifies that each choice set has three alternatives. OPTIONS=RELATIVE requests that the design's relative *D*-efficiency be displayed. BETA=ZERO specifies that the values of the true parameters are equal to 0.

Figure 9 displays the %ChoicEff macro's output.

Figure 9 Output from the %ChoicEff Macro

n	Name	Beta	Label
1	Size1	0	Size 1
2	Size2	0	Size 2
3	Color1	0	Color 1
4	Color2	0	Color 2
5	Price1	0	Price 1
6	Price2	0	Price 2

Design	Iteration	D-Efficiency	D-Error
1	0	3.00000 *	0.33333

Figure 9 *continued*

Final Results

Design	1
Choice Sets	3
Alternatives	3
Parameters	6
Maximum Parameters	6
D-Efficiency	3.0000
Relative D-Eff	100.0000
D-Error	0.3333
1 / Choice Sets	0.3333

n	Variable Name	Label	Variance	DF	Standard Error
1	Size1	Size 1	0.33333	1	0.57735
2	Size2	Size 2	0.33333	1	0.57735
3	Color1	Color 1	0.33333	1	0.57735
4	Color2	Color 2	0.33333	1	0.57735
5	Price1	Price 1	0.33333	1	0.57735
6	Price2	Price 2	0.33333	1	0.57735
				6	

This design is optimal. The relative *D*-efficiency is 100%, and all the variances of the parameter estimates are equal to 1 over the number of choice sets.