%MktDes Macro

The `%MktDes` autocall macro creates efficient experimental designs. Usually, we will not need to call the `%MktDes` macro directly. Instead, we will usually use the `%MktEx` autocall macro, which calls the `%MktDes` macro as one of its many tools. At the heart of the `%MktDes` macro are PROC PLAN, PROC FACTEX, and PROC OPTEX. PROC PLAN creates full-factorial designs. PROC FACTEX creates fractional-factorial designs. Both procedures can be used to create a candidate set for PROC OPTEX to search. We use a macro instead of calling these procedures directly because the macro has a simpler syntax. You specify the names of the factors and the number of levels for each factor. You also specify the number of runs you want in your final design. For example, you can create a design in 18 runs with 2 two-level factors (`x1` and `x2`) and 3 three-level factors (`x3`, `x4`, and `x5`) as follows:

```
   %mktdes(factors=x1-x2=2 x3-x5=3, n=18)
```

You can optionally specify interactions that you want to estimate. The macro creates a candidate design in which every effect you want to estimate is estimable, but the candidate design is bigger than you want. By default, the candidate set is stored in a SAS data set called CAND1. The macro then uses PROC OPTEX to search the candidate design for an efficient final design. By default, the final experimental design is stored in a SAS data set called DESIGN.

When the full-factorial design is small (by default less than 2189 runs, although sizes up to 5000 or 6000 runs are reasonably small), the experimental design problem is straightforward. First, the macro uses PROC PLAN to create a full-factorial candidate set. Next, PROC OPTEX searches the full-factorial candidate set. For very small problems (a few hundred candidates) PROC OPTEX will often find the optimal design, and for larger problems, it might not find *the* optimal design, but given sufficient iteration (for example, specify `maxiter=100` or more) it will find very good designs. Run time will typically be a few seconds or a few minutes, but it could be longer. The following shows a typical example of using the `%MktDes` macro to find an optimal nonorthogonal design when the full-factorial design is small (108 runs):

```
   *---2 two-level factors and 3 three-level factors in 18 runs---;
   %mktdes(factors=x1-x2=2 x3-x5=3, n=18, maxiter=500)
```

When the full-factorial design is larger, the macro uses PROC FACTEX to create a fractional-factorial candidate set. In those cases, the methods found in the `%MktEx` macro usually make better designs than those found with the `%MktDes` macro.

PROC FACTEX

The primary reason that the `%MktDes` macro exists is to provide a front end to PROC FACTEX, although it additionally provides a front end to PROC OPTEX and PROC PLAN. PROC FACTEX is powerful and general, and it does much more than we use it for here. However, that power leads to a somewhat long and detailed syntax. The `%MktDes` macro can help by writing that syntax for you. It should be pointed out that the syntax that the `%MktDes` macro writes for PROC FACTEX is often more verbose than it needs to be. It is simply easier for it to always write out a verbose style of syntax than try to be clever and take short cuts. Since PROC FACTEX is such an integral component of the `%MktDes` macro and hence the `%MktEx` macro, we will spend some time here reviewing what PROC FACTEX does and also the kinds of problems it was not designed to handle.

PROC FACTEX provides a powerful facility for constructing certain fractional-factorial designs. Specifically, PROC FACTEX is designed to make fractional-factorial designs based on prime numbers and powers of prime numbers. For example, since 2 is prime, you can use PROC FACTEX to make a design 2^{2^p-1} in 2^p runs. Examples include: 2^3 in $2^2 = 4$ runs, 2^7 in $2^3 = 8$ runs, 2^{15} in $2^4 = 16$ runs, 2^{31} in $2^5 = 32$ runs, 2^{63} in $2^6 = 64$ runs, and so on. Since 3 is prime, you can use PROC FACTEX to make a design $3^{(3^p-1)/2}$ in 3^p runs. Examples include: 3^4 in $3^2 = 9$ runs, 3^{13} in $3^3 = 27$ runs, 3^{40} in $3^4 = 81$ runs, and so on. Since 4 is a power of a prime, you can use PROC FACTEX to make a design $4^{(4^p-1)/3}$ in 4^p runs. Examples include: 4^5 in $4^2 = 16$ runs, 4^{21} in $4^3 = 64$ runs, and so on. Since 5 is prime, you can use PROC FACTEX to make a design $5^{(5^p-1)/4}$ in 5^p runs. Examples include: 5^6 in $5^2 = 25$ runs, 5^{31} in $5^3 = 125$ runs, and so on.

When the number of runs is a power of 2, you can use PROC FACTEX to make designs that contain mixes of 2, 4, 8, and other power-of-two-level factors. Examples include: $2^4 4^1$ in $2^3 = 8$ runs, $2^{3(5-q)} 4^q$ in $2^4 = 16$ runs for $q < 5$, $2^{13} 4^{12} 8^2$ in $2^6 = 64$ runs, and so on. When the number of runs is a power of 3, you can use PROC FACTEX to make designs that contain mixes of 3, 9, 27, and other power-of-three-level factors, and so on.

The `%MktEx` macro uses the `%MktDes` macro and PROC FACTEX for all of the cases described so far. PROC FACTEX is used both to make larger orthogonal arrays and to make all fractional-factorial candidate sets. There are many times, however, when PROC FACTEX cannot be used to make orthogonal arrays. You *cannot* use PROC FACTEX to make a design $6^{(6^p-1)/5}$ in 6^p runs since 6 is not a prime number. Even for designs that exist, like 6^3 in $6^2 = 36$ runs, you cannot use PROC FACTEX to make this design, because again, 6 is not a prime number. Similarly, you cannot use PROC FACTEX to make 2^{11} in 12 runs (12 is not a power of 2), 2^{19} in 20 runs (20 is not a power of 2), 3^7 in 18 runs (18 is not a power of 3), and so on. You have to use the `%MktEx` macro to get designs like these. You can use PROC FACTEX to create the design 6^3 in 64 runs by creating 3 eight-level factors in 64 runs and coding down. However, such a design, while orthogonal, would be nonoptimal both compared to 6^3 in 36 runs (which is 100% D-efficient) and compared to an optimal but nonorthogonal array 6^3 in 64 runs.

The `%MktDes` and `%MktEx` macros rely on PROC FACTEX to make many orthogonal arrays that `%MktEx` cannot otherwise make, and the `%MktEx` macro can make many designs that PROC FACTEX cannot make. However, for $2^p < 128$, and for many of the other orthogonal arrays not based on powers of 2, both the `%MktEx` macro and PROC FACTEX can make the same designs, but by using totally different approaches. PROC FACTEX does a computerized search, whereas the `%MktEx` macro develops a difference scheme. The `%MktEx` macro will usually use its own methods to make these orthogonal arrays when an orthogonal array was directly requested, but use PROC FACTEX to make the same orthogonal array when it is needed for a candidate set.

We discussed making mixed orthogonal arrays when the numbers of levels are all powers of the same prime (with mixes of 2's and 4's being the most common example). You can also use PROC FACTEX to create other mixes in several ways. You can create a mix of two-level and three-level factors by first creating a mix of two-level and four-level (or eight-level) factors and then coding down the four- or eight-level factors into three-level factors. This preserves orthogonality but leads to imbalance. You can instead create a design with just the two-level factors (in 2^p runs) and a separate design with just the three-level factors (in 3^q runs) and cross them (pair each of the 2^p runs with each of the 3^q runs) creating a design in $2^p \times 3^q$ runs. This preserves orthogonality and balance, but the designs tend to get big quite quickly, and other approaches to mixed orthogonal array creation provide more factors. The smallest example of this approach is $2^3 3^4$ in $2^2 \times 3^2 = 36$ runs. In contrast, the `%MktEx` macro can directly construct the mixed orthogonal array $2^{11} 3^{12}$ in 36 runs. The `%MktEx` macro uses the `%MktDes` macro to construct candidate sets using this first method, but does not employ the second method.

In summary, PROC FACTEX provides powerful software for constructing fractional-factorial designs. The `%MktDes` macro provides a front end to this procedure that makes it easier to call. The `%MktEx` macro uses the `%MktDes` macro for the things it is best at, but not for other things.

%MktDes Macro Options

The following options can be used with the `%MktDes` macro:

Option	Description
`help`	(positional) "help" or "?" displays syntax summary
`big=`*n*	size of big candidate set
`cand=`*SAS-data-set*	candidate design
`classopts=`*options*	`class` statement options
`coding=`*name*	`coding=` option
`examine=`$<$ I $>$ $<$ V $>$	matrices that you want to examine
`facopts=`*options*	PROC FACTEX statement options
`factors=`*factor-list*	factors and levels for each factor
`generate=`*options*	`generate` statement options
`interact=`*interaction-list*	interaction terms
`iter=`*n*	number of designs
`keep=`*n*	number of designs to keep
`maxiter=`*n*	number of designs
`method=`*name*	search method
`n=`*n* | SATURATED	number of runs
`nlev=`*n*	number of levels for pseudo-factors
`options=allcode`	shows all code
`options=check`	checks the `cand=` design
`options=nocode`	suppress the procedure code display
`otherfac=`*variable-list*	other factors
`otherint=`*terms*	multi-step interaction terms
`out=`*SAS-data-set*	output experimental design
`procopts=`*options*	PROC OPTEX statement options
`run=`*procedure-list*	list of procedures that can be run
`seed=`*n*	random number seed
`size=`*n* | MIN	candidate-set size
`step=`*n*	step number
`where=`*where-clause*	`where` clause

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

```
%mktdes(help)
%mktdes(?)
```

big= *n*

specifies the size at which the candidate set is considered to be big. By default, `big=2188`. If the size of the full-factorial design is less than or equal to this size, and if PROC PLAN is in the `run=` list, the macro uses PROC PLAN instead of PROC FACTEX to create the candidate set. The default of 2188 is $\max(2^{11}, 3^7) + 1$. Specifying values as large as `big=6000` or even slightly larger is often reasonable. However, run time is slower as the size of the candidate set increases. The `%MktEx` macro coordinate-exchange algorithm will usually work better than a candidate-set search when the full-factorial design has more than several thousand runs.

cand= *SAS-data-set*

specifies the output data set with the candidate design (from PROC FACTEX or PROC PLAN). The default name is `Cand` followed by the step number, for example: `Cand1` for step 1, `Cand2` for step 2, and so on. You should only use this option when you are reading an external candidate set. When you specify `step=` values greater than 1, the macro assumes the default candidate set names, CAND1, CAND2, and so on, were used in previous steps. Specify just a data set name, no data set options.

classopts= *options*

specifies PROC OPTEX `class` statement options. The default, is `classopts=param=orthref`. You probably never want to change this option.

coding= *name*

specifies the PROC OPTEX `coding=` option. This option is usually not needed.

examine= < I > < V >

specifies the matrices that you want to examine. The option `examine=I` displays the information matrix, $\mathbf{X}'\mathbf{X}$; `examine=V` displays the variance matrix, $(\mathbf{X}'\mathbf{X})^{-1}$; and `examine=I V` displays both. By default, these matrices are not displayed.

facopts= *options*

specifies PROC FACTEX statement options.

factors= *factor-list*

specifies the factors and the number of levels for each factor. The `factors=` option must be specified. All other options are not required. The following shows a simple example of creating a design with 10 two-level factors:

```
%mktdes(factors=x1-x10=2)
```

First, a factor list, which is a valid SAS variable list, is specified. The factor list must be followed by an equal sign and an integer, which gives the number of levels. Multiple lists can be specified. For example, to create 5 two-level factors, 5 three-level factors, and 5 five-level factors, specify the following:

```
%mktdes(factors=x1-x5=2 x6-x10=3 x11-x15=5)
```

By default, this macro creates each factor in a fractional-factorial candidate set from a minimum number of pseudo-factors. Pseudo-factors are not output; they are used to create the factors of interest and then discarded. For example, with `nlev=2`, a three-level factor `x1` is created from 2 two-level pseudo-factors (`_1` and `_2`) and their interaction by coding down:

```
(_1=1, _2=1) -> x1=1
(_1=1, _2=2) -> x1=2
(_1=2, _2=1) -> x1=3
(_1=2, _2=2) -> x1=1
```

This creates imbalance—the 1 level appears twice as often as 2 and 3. Somewhat better balance can be obtained by instead using three pseudo-factors. The number of pseudo-factors can be specified in parentheses after the number of levels, for example, as follows:

```
%mktdes(factors=x1-x5=2 x6-x10=3(3))
```

The levels 1 to 8 are coded down to 1 2 3 1 2 3 1 3, which is better balanced. The cost is candidate-set size might increase and efficiency might actually decrease. Some researchers are willing to sacrifice a little bit of efficiency in order to achieve better balance.

generate= *options*

specifies the PROC OPTEX `generate` statement options. By default, additional options are not added to the `generate` statement.

interact= *interaction-list*

specifies interactions that must be estimable. By default, no interactions are guaranteed to be estimable. Examples:

```
interact=x1*x2
interact=x1*x2 x3*x4*x5
interact=x1|x2|x3|x4|x5@2
interact=@2
```

The interaction syntax is like PROC GLM's and many of the other modeling procedures. It uses "`*`" for simple interactions (`x1*x2` is the interaction between `x1` and `x2`), "`|`" for main effects and interactions (`x1|x2|x3` is the same as `x1 x2 x1*x2 x3 x1*x3 x2*x3 x1*x2*x3`) and "`@`" to eliminate higher-order interactions (`x1|x2|x3@2` eliminates `x1*x2*x3` and is the same as `x1 x2 x1*x2 x3 x1*x3 x2*x3`). The specification "`@2`" creates main effects and two-way interactions. Only "`@`" values of `2` or `3` are permitted. If you specify "@2" by itself, a resolution V design is requested when PROC FACTEX is run.

iter= *n*
maxiter= *n*

specifies the PROC OPTEX `iter=` option which creates *n* designs. By default, `iter=10`.

keep= *n*

specifies the PROC OPTEX `keep=` option which keeps the *n* best designs. By default, `keep=5`.

nlev= *n*
specifies the number of levels from which factors are constructed through pseudo-factors and coding down. The value must be a prime or a power of a prime: 2, 3, 4, 5, 7, 8, 9, 11 .... This option is used with PROC FACTEX as follows:

```
   factors factors / nlev=&nlev;
```

By default, the macro uses the minimum prime or power of a prime from the `factors=` list or 2 if no suitable value is found.

method= *name*
specifies the PROC OPTEX `method=` search method option. The default is `method=m_Fedorov` (modified Fedorov).

n= *n* | `saturated`
specifies the PROC OPTEX `n=` option, which is the number of runs in the final design. The default is the PROC OPTEX default and depends on the problem. Typically, you will not want to use the default. Instead, you should pick a value using the information produced by the `%MktRuns` macro as guidance (see page 1159). The `n=saturated` option creates a design with the minimum number of runs.

options= *options-list*
specifies binary options. By default, none of these options are specified. Specify one or more of the following values after `options=`.

`check`
checks the efficiency of a given design, specified in `cand=`.

`nocode`
suppresses the display of the PROC PLAN, PROC FACTEX, and PROC OPTEX code.

`allcode`
shows all code, even code that will not be run.

otherfac= *variable-list*
specifies other terms to mention in the `factors` statement of PROC FACTEX. These terms are not guaranteed to be estimable. By default, there are no other factors.

otherint= *terms*
specifies interaction terms that will only be specified with PROC OPTEX for multi-step macro invocations. By default, no interactions are guaranteed to be estimable. Normally, interactions that are specified via the `interact=` option affect both the PROC FACTEX and the PROC OPTEX `model` statements. In multi-step problems, part of an interaction might not be in a particular PROC FACTEX step. In that case, the interaction term must only appear in the PROC OPTEX step. For example, if `x1` is created in one step and `x4` is created in another, and if the `x1*x4` interaction must be estimable, specify `otherint=x1*x4` on the final step, the one that runs PROC OPTEX. The following steps create

the design:

```
%mktdes(step=1, factors=x1-x3=2, n=30, run=factex)

%mktdes(step=2, factors=x4-x6=3, n=30, run=factex)

%mktdes(step=3, factors=x7-x9=5, n=30, run=factex optex,
        otherint=x1*x4)
```

out= *SAS-data-set*

specifies the output experimental design (from PROC OPTEX). By default, `out=design`. Often, you will want to specify a two-level name to create a permanent SAS data set so the design is available later for analysis.

procopts= *options*

specifies PROC OPTEX statement options. By default, no options are added to the PROC OPTEX statement.

run= *procedure-list*

specifies the list of procedures that the macro can run. Normally, the macro runs either PROC FACTEX or PROC PLAN and then PROC OPTEX. By default, `run=plan factex optex`. You can skip steps by omitting procedure names from this list. When both PLAN and FACTEX are in the list, the macro chooses between them based on the size of the full-factorial design and the value of `big=`. When PLAN is not in the list, the macro generates code for PROC FACTEX.

seed= *n*

specifies the random number seed. By default, `seed=0`, and clock time is used to make the random number seed. By specifying a random number seed, results should be reproducible within a SAS release for a particular operating system and for a particular version of the macro. However, due to machine and macro differences, some results might not be exactly reproducible everywhere, although you would expect the efficiency differences to be slight.

size= *n* | `min`

specifies the candidate-set size. Start with the default `size=min` and see how big that design is. If you want, subsequently you can specify larger values that are `nlev=`*n* multiples of the minimum size. This option is used with PROC FACTEX as follows:

```
size design=&size;
```

When `nlev=`*n*, increase the `size=` value by a factor of *n* each time. For example, when `nlev=2`, increase the `size=` value by a factor of two each time. If `size=min` implies `size=128`, then 256, 512, 1024, and 2048 are reasonable sizes to try. Integer expressions like `size=128*4` are permitted.

step= *n*

specifies the step number. By default, there is only one step. However, sometimes, a better design can be found using a multi-step approach. Do not specify the `cand=` option in any step of a multi-step run. Consider the problem of making a design with 3 two-level factors, 3 three-level factors, and 3 five-level factors. The simplest approach is to do something like the following—create a design from two-level factors using pseudo-factors and coding down:

```
%mktdes(factors=x1-x3=2 x4-x6=3 x7-x9=5, n=30)
```

However, for small problems like this, the following three-step approach is usually better:

```
%mktdes(step=1, factors=x1-x3=2, n=30, run=factex)
%mktdes(step=2, factors=x4-x6=3, n=30, run=factex)
%mktdes(step=3, factors=x7-x9=5, n=30, run=factex optex)
```

Note, however, that the following `%MktEx` macro step will usually be better still:

```
%mktex(2 2 2 3 3 3 5 5 5, n=30)
```

The first `%MktDes` macro step uses PROC FACTEX to create a fractional-factorial design for the two-level factors. The second step uses PROC FACTEX to create a fractional-factorial design for the three-level factors and cross it with the two-level factors. The third step uses PROC FACTEX to create a fractional-factorial design for the five-level factors and cross it with the design for the two and three-level factors and then run PROC OPTEX.

Each step globally stores two macro variables (`&class1` and `&inter1` for the first step, `&class2` and `&inter2` for the second step, ...) that are used to construct the PROC OPTEX `class` and `model` statements. When `step` $>$ 1, variables from the previous steps are used in the `class` and `model` statements. In this example, the following PROC OPTEX code is created by step 3:

```
proc optex data=Cand3;
   class
      x1-x3
      x4-x6
      x7-x9
      / param=orthref;
   model
      x1-x3
      x4-x6
      x7-x9
      ;
   generate n=30 iter=10 keep=5 method=m_fedorov;
   output out=Design;
   run; quit;
```

This step uses the previously stored macro variables `&class1=x1-x3` and `&class2=x4-x6`.

where= *where-clause*
specifies a SAS `where` clause for the candidate design, which is used to restrict the candidates. By default, the candidate design is not restricted.

%MktDes Macro Notes

This macro specifies `options nonotes` throughout much of its execution. If you want to see all of the notes, submit the statement `%let mktopts = notes;` before running the macro. To see the macro version, submit the statement `%let mktopts = version;` before running the macro.