CBCS
HTTP://AGROBIOL.SGGW.WAW.PL/CBCS

COMMUNICATIONS IN BIOMETRY AND CROP SCIENCE
VOL. 10, NO. 2, 2015, PP. 96–114

INTERNATIONAL JOURNAL OF THE FACULTY OF AGRICULTURE AND BIOLOGY,
WARSAW UNIVERSITY OF LIFE SCIENCES – SGGW, POLAND

SOFTWARE TRICKS AND TIPS

Generating efficient designs for comparative experiments using the SAS procedure OPTEX

Hans-Peter Piepho

Biostatistics Unit, Institute of Crop Science, University of Hohenheim, Fruwirthstrasse 23, 70599 Stuttgart, Germany
E-mail: piepho@uni-hohenheim.de

CITATION: Piepho H.P. (2015). Generating efficient designs for comparative experiments using the SAS procedure OPTEX. *Communications in Biometry and Crop Science* 10, 96–114.

Received: 8 June 2015, Accepted: 8 September 2015, Published online: 27 September 2015

ABSTRACT

Good agricultural experiments require efficient experimental designs, so availability of convenient software for design generation is important. Here, we consider the SAS procedure OPTEX, which was developed as a general-purpose design generation tool focused mainly on response surface regression experiments. This procedure can also be used to generate good designs for comparative experiments as needed in crop science research. This is illustrated here for the most commonly used experimental designs.

Key Words: *A-optimality; alpha design; average efficiency factor; blocking; completely randomized design; D-optimality; Latin square design; lattice square design; non-resolvable; partially replicated design; randomized complete block design; resolvable; row-column design.*

INTRODUCTION

Much of the scientific advance in crop science relies on field experiments. To maximize the information that can be extracted from such experiments, the use of efficient experimental designs is crucial. While the underlying theory is well known and laid down in many standard textbooks (e.g. John and Williams 1995, Mead et al. 2012), the deployment in practice of good experimental designs suited for the targeted objectives is to a considerable extent governed by the available software. For example, the most commonly used procedure in SAS for the generation of field experimental designs is the PLAN procedure. This is very easy to use for generating some common designs, such as randomized complete block designs (RCBD) or split-plot designs. But I have found the code needed to generate some basic designs such as a completely randomized design (CRD) or a Latin square design (LSQD) unexpectedly complex. Also, it is difficult to use the PLAN procedure for generating row-column designs or designs with unequal block sizes or treatment replications.

In this paper I will introduce the reader to the procedure OPTEX (part of the QC module), which is a general-purpose tool for experiment design generation. It is mostly used for other design problems, including response surface designs and designs with many treatment factors (Atkinson et al. 2009), but it can also be used to design comparative experiments. The focus here will be on designs that are very commonly needed in field experiments. It will be demonstrated that all basic designs can be easily generated using this procedure and that it opens up the way to tackle more challenging design problems. Limitations of the OPTEX procedure will also be discussed.

SOME BACKGROUND ON OPTEX

The OPTEX procedure is a general-purpose design tool, which is mainly geared towards designs for quantitative treatment variables, such as response surface designs and designs for mixture experiments (Atkinson et al. 2009). The key idea behind the search strategies implemented in the procedure is to start with a number of candidate design points and a linear model for analysis. The search is then done by assembling a design making selections from the candidate set in such a way that efficient estimates of the relevant model terms are obtained. If the treatment factors are quantitative (x_1, x_2, …, say), as in response surface regression, a design point is a specific choice of values assigned to (x_1, x_2, …). Because of the continuity of quantitative treatment variables, potentially there is often a very large set of design points to choose from. In the context of comparative experiments, the treatment factor is usually qualitative, and the candidate set consists simply of the treatment labels. Thus, the first step in the code I provide for generating designs is to create a file "TreatmentLabels" that contains all treatment labels.

There are several design search algorithms implemented in OPTEX. All of them start with a set of design points, which is then modified successively to optimize its efficiency. A salient feature of all these search algorithms is that they use some kind of exchange of design points in the current design and points in the candidate set, where the benefit of the exchange is judged based on a specific optimality criterion computed from variances and covariances of the treatment effect estimators (D-efficiency; Atkinson et al. 2009).

Another important feature of the procedure is that the model for the factors of interest, i.e., the treatment factors, is stated separately from a model pertaining to the blocking factors of the design. This separation of block and treatment model is generally very useful for setting up a suitable model for a randomized experiment (Piepho et al. 2003). This is a necessity for the efficient use of OPTEX because the optimality criterion based on which the design is judged should be evaluated based on the treatment model only (John and Williams 1995, Pereira and Tobias 2015) and treatments are allocated to blocks in a way that maximizes the efficiency. The general strategy implemented in OPTEX allows generating good designs for comparative experiments, but it also entails some limitations that will be mentioned both with the examples given below and in the concluding discussion. It is worth mentioning that the PLAN procedure is part of the CORE module of SAS, while OPTEX is available only with the QC module, which needs to be purchased separately. Full documentation of the procedure can be found on the web and via the help manual of the SAS package, so a full description will not be given here.

RANDOMIZED COMPLETE BLOCK DESIGN (RCBD)

Example (complete blocks): $v = 7$ treatments, $r = 4$ replicates.

To set the stage, it is shown here that OPTEX can also be used to generate such basic designs as the RCBD, though in practice it is simpler to use datastep programming or the PLAN procedure of SAS (see Appendix). To generate a RCBD using OPTEX, we generate a file "TreatmentLabels" that contains all treatment labels and a file "layout" that contains the

block and plot labels. This is done using do loops in the first two data steps shown at the top of Box 1.

```
data TreatmentLabels;
  do trt=1 to 7;
    output;
  end;
run;

data layout;
  do block=1 to 4;
    do plot=1 to 7;
      output;
    end;
  end;
run;

proc optex data=TreatmentLabels seed=98221534;
  class trt;
  model trt;
  blocks design=layout;
  class block plot;
  model block;
  output out=RCBD;
run;

proc print data=RCBD;
run;
```

Box 1: Code to generate a RCBD for seven treatments and four replications.

Obs	block	plot	trt
1	1	1	6
2	1	2	5
3	1	3	7
4	1	4	1
5	1	5	2
6	1	6	3
7	1	7	4
8	2	1	5
9	2	2	4
10	2	3	7
11	2	4	1
12	2	5	3
13	2	6	2
14	2	7	6
15	3	1	6
16	3	2	4
17	3	3	7
18	3	4	5
19	3	5	3
20	3	6	2
21	3	7	1
22	4	1	5
23	4	2	3
24	4	3	2
25	4	4	4
26	4	5	6
27	4	6	1
28	4	7	7

Output 1: Dataset generated by the code in Box 1.

The subsequent OPTEX code in Box 1 then generates a RCBD with seven treatments and four replications, requiring a total of 28 plots. A positive integer is used as seed for the random number generator so that the design generated can be reproduced perfectly. The seed value is specified using the SEED= option of the procedure call. It is recommended that users specify a new seed for each new trial in order to make sure that different randomizations are generated for different trials. The default is a negative seed so that the computer clock initializes the seed. A salient feature of the procedure, which is apparent here, is that the treatment model and the block model are stated separately. Design optimization focuses on the treatment model, but takes into account the block structure and assigns treatments to blocks in a way that maximizes efficiency. In this case, the optimal solution is to assign each treatment to each block exactly once. The file with the treatment labels is provided via the data= option in the call of the procedure (first line), followed by the statement of the treatment model in the second and third row. The layout file representing the physical field plan is provided via the blocks statement (third row), which is followed by a statement of the block model in the subsequent two rows. The design file generated by the code in Box 1 (output statement; second from last row) is shown here in Output 1. For the other examples I will refrain from showing these output files because the general appearance is the same for all of them.

LATIN SQUARE DESIGN (LSQD)

Example: v = 7 treatments, k = 7 rows and s = 7 columns.

The code in Box 2 generates a LSQD for seven treatments. The "layout" file now needs to have design variables for rows and columns of the field layout. The tabulate procedure is used to display the design as shown in Figure 1.

```
data TreatmentLabels;
  do trt=1 to 7;
    output;
  end;
run;

data layout;
  do row=1 to 7;
    do col=1 to 7;
      output;
    end;
  end;
run;

proc optex data=TreatmentLabels seed=3594362;
  class trt;
  model trt;
  blocks design=layout;
  class row col;
  model row col;
  output out=LSQD;
run;

proc tabulate DATA=LSQD format=best5.0;
  class row col;
  var trt;
  table row,  trt=' ' *col *sum=' ';
run;
```

Box 2: Code to generate a LSQD for seven treatments.

5	4	3	7	6	1	2
4	1	2	6	3	5	7
2	5	6	1	4	7	3
3	6	1	2	7	4	5
1	3	7	4	5	2	6
6	7	4	5	2	3	1
7	2	5	3	1	6	4

Figure 1. Tabulation of the LSQD generated by the code in Box 2.

NON-RESOLVABLE INCOMPLETE BLOCK DESIGN

Example: $v = 7$ treatments, $b = 7$ blocks, block size $k = 4$, $r = 4$ replications.

Now assume that seven treatments are to be tested in seven blocks of size four. The code in Box 3 will generate the appropriate design. Note that the code is essentially the same as for the RCBD; only the numbers of blocks and plots per block had to be adjusted.

```
data TreatmentLabels;
  do trt=1 to 7;
    output;
  end;
run;

data layout;
  do block=1 to 7;
    do plot=1 to 4;
      output;
    end;
  end;
run;

proc optex data=TreatmentLabels seed=73569547;
  class trt;
  model trt;
  blocks design=layout;
  class block plot;
  model block;
  output out=IBD;
run;
```

Box 3: Code to generate a non-resolvable incomplete block design for seven treatments in seven blocks of size four.

The design will have four replications per treatment. In this particular case, the design is also variance-balanced, i.e., the variance (or standard error) of a difference is the same for all pairs of treatments. This can be verified by a dummy analysis (Box 4). In this analysis, the dummy response variable is set to an arbitrary value (unity in this case), and an analysis is performed fixing the residual variance at an arbitrary value (unity in this case). The variance of a difference equals 0.5714 for all treatment pairs. This can be compared to the average pairwise variance of a design with the same number of treatments and replications laid out as an RCBD, which for a residual variance of unity equals $2/r = 0.5$, where $r = 4$ is the number of replications. The ratio of these two pairwise variances equals 0.5/0.5714 = 0.875, which is known as the average efficiency factor (a.e.f.). Generally, the a.e.f. is the average pairwise variance of the contemplated design, divided by the variance of a difference for an RCBD for the same number of treatments and replications, assuming that the error variances are identical (John and Williams 1995). There is always a loss incurred from incomplete blocking, so the a.e.f. for an incomplete block design is always smaller than unity. The closer to unity the a.e.f., the better is the design. The value a.e.f. = 0.875 for the design generated by the code in Box 3 is, in fact, the value for the optimal balanced incomplete block design for the same specification in terms of the number of treatments, number of replications, and block size (Mead 1988, p.152, Table 7.10; Hinkelmann and Kempthorne 1994, p.294).

```
data ibd;
  set ibd;
  dummy_response=1;
run;

ods output diffs=diffs_ibd;
proc mixed data=IBD;
  class trt block;
  model dummy_response=trt block;
  parms (1) / hold=1;
  lsmeans trt / pdiff;
run;

data diffs_ibd;
  set diffs_ibd;
  vd=Stderr**2;
run;

proc means data=diffs_ibd;
  var vd;
  output out=mean_vd mean=vd_mean;
run;

data aef;
  set mean_vd;
  r=4; /*number of replications*/
  aef=2/r/vd_mean;
run;

proc print data=aef;
  var aef;
run;
```

Box 4: Dummy analysis for design generated by code in Box 3 with code to compute the average pairwise variance and the a.e.f.

BLOCK DESIGNS WITH UNEQUAL BLOCK SIZES

Example: v = 7 treatments, b = 5 blocks of unequal size ranging from 5 to 10 units, r = 5 replications.

Mead (1988, p.139, Example 7.7) gives an example of a block design with unequal block sizes. Seven medical treatments need to be tested in piglets (Figure 1). The piglets come in litters, and litters are unequal in size (5, 6, 7, 7 and 10 piglets). Litters are a natural blocking unit. There are 35 piglets overall, so each treatment can be tested on five piglets. Based on the principle that the number of co-occurrences of pairs of treatments in a block should be as balanced as possible between pairs, Mead (1988) manually developed the design shown in Figure 2 based on basic principles. The two litters of size seven form complete blocks (blocks III and IV). The two smaller blocks lack treatments 6 and 7 (block I) and treatment 5 (block II). These same treatments have one extra replication in block V. The example nicely shows that one can usually generate a fairly efficient design manually based on some intuition even in non-standard situations.

Box 5 gives code to generate an efficient design for this example. The resulting design is shown in Figure 2. Note that the treatments are equally replicated, as in Figure 2. Using the same approach as in Box 4, the average pairwise variance is found to be 0.4086 for the design in Figure 3, compared to 0.4451 for the design in Figure 2, so using the computer to search for an efficient design leads to some improvement here.

Block				
I	II	III	IV	V
1	1	1	1	1
2	2	2	2	2
3	3	3	3	3
4	4	4	4	4
5	6	5	5	5
	7	6	6	6
		7	7	7
				5
				6
				7

Figure 2. Unrandomized design for five blocks of unequal size and seven treatments (1-7) (Mead 1988).

```
data TreatmentLabels;
  do trt=1 to 6;
    output;
  end;
run;

data layout;
  input block size;
  do plot=1 to size;
    output;
  end;
datalines;
1 5
2 6
3 7
4 7
5 10
;
proc optex data=TreatmentLabels seed=1644385;
  class trt;
  model trt;
  blocks design=layout;
  class block plot;
  model block;
  output out=Mead;
run;
```

Box 5: Code to generate a block design for seven treatments in five blocks of unequal size (5, 6, 7, 7 and 10).

Block				
I	II	III	IV	V
5	1	6	3	2
3	3	1	7	7
4	4	2	1	1
6	7	3	5	4
2	6	4	6	6
	2	7	4	1
		5	2	5
				7
				3
				5

Figure 3. Randomized design for five blocks of unequal size and seven treatments (1-7) generated using code in Box 5.

NON-RESOLVABLE ROW-COLUMN DESIGN

Example: v = 14 treatments, k = 7 rows, s = 4 columns, r = 2 replications.

Assume that we want to test 14 treatments in a layout with seven rows and four columns. Thus, each treatment is replicated twice. The code in Box 6 generates such a design. The code is basically the same as that for an LSQD in Box 2, but the numbers of treatments, rows and columns had to be adjusted.

```
data TreatmentLabels;
  do trt=1 to 14;
    output;
  end;
run;

data layout;
  do row=1 to 7;
    do col=1 to 4;
      output;
    end;
  end;
run;

proc optex data=TreatmentLabels seed=5482755;
  class trt;
  model trt;
  blocks design=layout;
  class row col;
  model row col;
  output out=RCD;
run;

proc tabulate data=RCD format=best5.0;
  class row col;
  var trt;
  table row,  trt=' ' *col *sum=' ';
run;
```

Box 6: Code to generate a non-resolvable row-column design for 14 treatments in seven rows and four columns.

PARTIALLY REPLICATED AND OTHER UNEQUALLY REPLICATED DESIGNS

Example (unequal replication): v = 21 treatments, k = 7 rows, s = 8 columns, 7 treatments replicated twice, 14 treatments replicated three times.

In all examples so far, each treatment was equally replicated. Sometimes, the number of experimental units available does not allow each treatment to be equally replicated. All code shown so far can also be used when the number of treatments is not a multiple of the number of experimental units (plots). In such cases, the number of replications per treatment will differ by one. For example, if we have 21 treatments and 56 plots laid out in seven rows and eight columns, then 14 of the treatments can be replicated three times, while seven will be replicated only twice. The code in Box 6 is easily modified to accommodate this setting (Box 7).

```
data TreatmentLabels;
  do trt=1 to 21;
    output;
  end;
run;

data layout;
  do row=1 to 7;
    do col=1 to 8;
      output;
    end;
  end;
run;

proc optex data=TreatmentLabels seed=5187532;
  class trt;
  model trt;
  blocks design=layout;
  class row col;
  model row col;
  output out=RCD2;
run;
```

Box 7: Code to generate a non-resolvable row-column design for 21 treatments in seven rows and eight columns.

Example (partial replication): v = 134 treatments, b = 40 blocks of size k = 5, 66 treatments replicated twice, 68 treatments replicated once.

A particular case of unequal replication is partial replication, by which some treatments are replicated twice, while the other treatments are unreplicated. Such partially replicated (p-rep) designs are particularly useful in early generation plant breeding trials where the amount of seed available may be too limited to allow replication of all entries (Smith et al. 2006). For example, if we have 134 treatments and want to use a field layout with 40 blocks of size 5, we can adjust the code in Box 3 accordingly (Box 8).

```
data TreatmentLabels;
  do trt=1 to 134;
    output;
  end;
run;

data layout;
  do block=1 to 40;
    do plot=1 to 5;
      output;
    end;
  end;
run;

proc optex data=TreatmentLabels seed=27334109;
  class trt;
  model trt;
  blocks design=layout;
  class block plot;
  model block;
  output out=PREP;
run;
```

Box 8: Code to generate a p-rep design for 134 treatments laid out in 40 incomplete blocks of size five.

It is good practice to check if a generated p-rep design has a sufficient number of observations to be analysed. For this purpose, one may simulate dummy data and run a dummy analysis, this time without fixing the residual variance, just so that it can be checked that this variance is estimable. Alternatively, one can add up the model degrees of freedom and compare them to the number of observations. In the case at hand, there are 133 d.f. for treatment and 39 d.f. for blocks, so including the intercept (grand mean) the model d.f. equal 133+39+1=173, leaving 27 d.f. for error. Thus, the design can be analysed. Box 9 shows code for a dummy analysis of this example.

The term "partial replication" may suggest that replication is in any way optional. It is not! Without replication, there is no way to assess the experimental error variance and hence there is no basis for any useful statistical inference. The salient property of a p-rep design is that there is sufficient replication, based on a subset of the treatments, to allow a valid estimation of the error variance. So far the use of these designs has been restricted to early-generation testing in plant breeding where seed of new candidate lines is usually limited. It should be added that in early-generation testing, replication usually occurs at a higher level, because trials are replicated across locations, meaning that for the multi-location design there will be replication for all entries. The main challenge then is to balance out the partial replication across locations in such a way that each entry is replicated the same number of times and the overall design has good efficiency. The generation of such designs requires specialized design software such as CycDesigN (VSN-International, https://www.vsni.co.uk). CycDesigN is a stand-alone software package for the generation of efficient blocked experimental designs. It uses a combination of numerical search strategies and statistical theory, the foundations of which are described in John and Williams (1995); specifics pertaining to p-rep designs are given in Williams et al. (2011, 2014). Another package that also uses numerical search methods to generate efficient designs is DiGGer (available in R, cran.r-project.org).

```
data PREP;
  set PREP;
  dummy_response=normal(1);
run;

proc mixed data=PREP;
  class trt block;
  model dummy_response=trt block;
  lsmeans trt;
run;
```

Box 9: Code for dummy analysis of design generated using code of Box 9. This code checks whether there are enough observations to estimate the residual variance.

Example (unequal replication): v = 6 treatments, k = 6 rows, s = 8 columns, treatment replication numbers 9, 12, 9, 8, 6 and 4.

In some applications, the number of replications per treatment may be non-constant and fixed because of limited availability of material for some of the treatments. For example, Williams and Piepho (2015) consider an experiment for six treatments to be laid out in six-by-eight array. Replication numbers available for the treatments were 9, 12, 9, 8, 6 and 4. With the designs considered so far, we required designs that are either equally replicated or where the replication numbers differ by one, as was the case for the designs generated by the codes in Boxes 7 and 8. To fix the replication numbers at pre-specified values, we can provide an initial treatment design with the required treatment replications via the INITDESIGN option of the GENERATE statement. In the BLOCK statement we need to use the NOEXCHANGE option to prevent the procedure from exchanging treatments and thus

changing the initial treatment design. The code to generate a design for this problem is given in Box 10, where the initial treatment design with the desired replication numbers is in the file "TreatmentReps". The resulting design is shown in Figure 4.

```
data TreatmentLabels;
  do trt=1 to 6;
    output;
  end;
run;

data TreatmentReps;
  input trt replication;
  do copy=1 to replication;
    output;
  end;
datalines;
1 9
2 12
3 9
4 8
5 6
6 4
;
data layout;
  do row=1 to 6;
    do col=1 to 8;
      output;
    end;
  end;
run;

proc optex data=TreatmentLabels seed=9477245;
  class trt;
  model trt;
  blocks design=layout iter=10000 keep=10 noexchange;
  class row col;
  model row col;
  generate initdesign=TreatmentReps method=sequential;
  output out=RCD2;
run;

proc freq data=rcd2;
  table trt;
run;

proc tabulate DATA=RCD2 format=best5.0;
  class row col;
  var trt;
  table row,  trt=' ' *col *sum=' ';
run;
```

Box 10: Code to generate a row-column design for six treatments laid out in six rows and eight columns with treatment replication numbers equal to 9, 12, 9, 8, 6 and 4.

1	4	3	5	4	3	2	2
4	1	5	1	6	2	3	2
6	1	1	2	2	5	4	3
2	2	4	3	3	1	6	5
3	3	2	4	2	6	5	1
5	3	2	2	1	4	1	4

Figure 4. Two-dimensional layout for row-column design with six rows and eight columns for six treatments with replication numbers 9, 12, 9, 8, 6 and 4.

RESOLVABLE INCOMPLETE BLOCK DESIGN

Example: v = 30 treatments, block size k = 5, r = 3 replicates.

A blocked design is said to be resolvable if incomplete blocks can be grouped to form complete replicates. Typical examples of this class of designs are square and rectangular lattice designs and alpha designs. Generating such designs in OPTEX is a bit more complex than generating non-resolvable designs (Pereira and Tobias 2015). The reason for the complication is that now two nested blocking effects need to be considered, i.e. one for replicates and one for blocks nested within replicates. In searching a good design, the procedure exchanges current design points (treatments in our case) to optimize efficiency. With this approach, there is no restriction to guarantee that a design will be resolvable. It is therefore important to check that a generated design is resolvable.

Consider a resolvable design for 30 treatments in six blocks per replicate and three replicates. One might consider using the code in Box 11 for design generation. The call of the FREQ procedure produces a frequency table for the treatment-by-replicate classification. For a resolvable design, all entries in the table must be unity. But this is not the case for the design generated using the code in Box 11. The design is only optimized for the composition of incomplete blocks.

A solution to the problem, described in Pereira and Tobias (2015), is to model effects for replicates and blocks as random and assign prior values to their variances. If a large variance is assigned to an effect, it essentially behaves like a fixed effect (Piepho et al. 2013). The larger the variance, the more dominant it will be in the design search. Thus, if we assign a very large variance to replicates in comparison to blocks, then replicates will dominate the search, thus increasing the likelihood of finding a resolvable design. In OPTEX, one assigns a prior to the inverse of the variance, the so-called prior precision. Thus, we may assign a prior precision of zero to specify an infinite variance. The code in Box 12 does this, and the prior precision for blocks is set to 10. Note that the two effects in the block model now need to be separated by a comma. Generally, a comma separates groups of effects assigned the same prior. Also note that we have followed the suggestion of Pereira and Tobias (2015) to set the number of random starts for the design search to a large number (NITER=10000) because

this increases the likelihood of finding a good design, but computing time can be long (several minutes or more, depending on the size of the design). The resulting design is found to be resolvable. But is it the most efficient design?

```
data TreatmentLabels;
  do trt=1 to 30;
    output;
  end;
run;

data layout;
  do rep=1 to 3;
    do block=1 to 6;
      do plot=1 to 5;
         output;
      end;
    end;
  end;
run;

proc optex data=TreatmentLabels seed=58026475;
  class trt;
  model trt;
  blocks design=layout;
  class rep block plot;
  model rep block(rep);
  output out=RIBD;
run;

proc freq data=RIBD;
  table trt*rep / norow nocol nopct nocum;
run;
```

Box 11: Initial (but inappropriate) code to generate a resolvable incomplete block design for 30 treatments, three replicates and five blocks per replicate.

```
proc optex data=TreatmentLabels seed=58026475;
  class trt;
  model trt;
  blocks design=layout niter=10000 keep=10;
  class rep block plot;
  model rep, block(rep) / prior=0,10;
  output out=RIBD;
run;
```

Box 12: Code to generate a resolvable incomplete block design for 30 treatments, three replicates and six blocks per replicate.

To compare designs, one may compute the average efficiency factor (a.e.f.) (Box 13). For the design generated using the code in Box 12, we find a.e.f. = 0.78555, which is indeed the value for the optimal rectangular lattice for the same specification in terms of the number of treatments, replicates and blocks per replicate (John and Williams 1995, §4.3).

```
data RIBD;
  set RIBD;
  dummy_response=1;
run;

ods output diffs=vd_RIBD;
proc mixed data=RIBD;
  class trt rep block;
  model dummy_response=trt rep block(rep);
  parms (1) / hold=1;
  lsmeans trt / pdiff;
run;

data aef;
  set vd_RIBD;
  vd=StdErr**2;
run;

proc means data=aef noprint;
  var vd;
  output out=mean_vd mean=mean_vd;
run;

data aef;
  set mean_vd;
  r=3; /*number of replicates*/
  aef=2/r/mean_vd;
run;

proc print data=aef;
  var aef;
run;
```

Box 13: Code to compute the average efficiency factor a.e.f. for the design generated using the code in Box 12.

Example: v = 200 treatments, block size = 10, r = 2 replicates.

Consider the more challenging problem of finding a design for 200 treatments to be tested in two replicates with 20 blocks per replicate. Modifying the code in Box 12 accordingly, setting the seed to 392344 and increasing the prior precision to 100 (code not shown), we find a design with a.e.f.=0.82482. This is only marginally lower than that for an alpha design found with the design package CycDesigN (a.e.f.=0.82485). It should be noted that OPTEX uses D-optimality (based on the determinant of the treatment information matrix) whereas CycDesigN uses A-optimality (based on the trace of the treatment information matrix), which is directly related to the a.e.f., so the slight difference in a.e.f. is not unexpected. A- and D-optimality usually lead to very similar or equivalent designs (Atkinson et al. 2009, Williams and Piepho 2015). Running OPTEX with 1000 iterations took almost four hours of CPU time on a laptop (Intel ® Core ™2 Duo CPU, P9400 @ 2.40 GHz, 4.00 GB RAM, Windows XP, 32 bit).

RESOLVABLE ROW-COLUMN DESIGN

Example: v = 25 treatments, k = 5 rows, s = 5 columns, r = 2 replicates.

A design with two replicates, each arranged in five rows and columns, is generated by the code in Box 14. The prior for the row and column effects was set to 100 in order to obtain a

resolvable design. The average efficiency factor for the design found is a.e.f.=0.600, which is identical to the optimal value for a lattice square design (John and Williams 1995, §6.2, eq. 6.4).

```
data TreatmentLabels;
  do trt=1 to 25;
    output;
  end;
run;

data layout;
  do rep=1 to 2;
    do row=1 to 5;
      do col=1 to 5;
         output;
      end;
    end;
  end;
run;

proc optex data=TreatmentLabels seed=253385;
  class trt;
  model trt;
  blocks design=layout niter=10000 keep=10;
  class rep row col;
  model rep, row(rep) col(rep) / prior=0,100;
  output out=RRCD;
run;
```

Box 14: Code to generate a resolvable row-column design with two replicates for 25 treatments in five rows and five columns.

Example: v = 150 treatments, k = 10 rows, s = 15 columns, r = 2 replicates.

Next, a design for 150 treatments and two replicates with 15 rows and 10 columns is generated, using code analogous to that in Box 14 with 1000 iterations. It took about two hours of CPU time to compute on a laptop. The prior precision for row and column effects was set to 300 to achieve resolvability, and the seed was 14821263. This produces an a.e.f.=0.7532. For comparison, I generated a design with the same specification in CycDesigN and found a.e.f.=0.7542, which is slightly better. Again, the slight difference is not unexpected, because CycDesigN optimizes a.e.f. directly (A-optimality), whereas OPTEX uses D-optimality.

DISCUSSION

It has been demonstrated using a few pertinent examples that the OPTEX procedure can be used to generate good designs for comparative experiments as needed in crop science research. It should be kept in mind, however, that the procedure was not tailor-made for these types of design. In my experience, the procedure works very well for small experiments, particularly when the design is non-resolvable. For resolvable designs, the exchange algorithms used entail an extra computational challenge because these algorithms are not tailored to ensure resolvability. Using the PRIOR option, the replicate effect can be made the dominant effect in the block model. This increases the likelihood that the final design will be resolvable, but there is no guarantee for this, and it is crucial to check any design for resolvability. Also, some tuning of the prior precision assigned to the incomplete block effects is required, and there is currently only limited experience as to the optimal prior values (Pereira and Tobias 2015). During iterations, many non-resolvable designs are

probably generated and discarded because of low efficiency. It is also crucial to use a large number of start configurations, which can be specified using the NITER option to the BLOCKS statement (Pereira and Tobias 2015). A more targeted search strategy would restrict the design space to resolvable designs. For example, one could start with a design that is resolvable and then swap treatments within replicates to improve the average efficiency factor. This type of design strategy is implemented in more specialized design packages such as CycDesigN and Gendex. Also, the cyclic generation of resolvable designs from alpha arrays is particularly efficient, and the use of theoretical upper bounds for the average efficiency factor helps in judging how far from the best possible design we are in the search (John and Williams 1995). None of these specialized strategies are implemented in the general-purpose tool OPTEX, meaning that for large designs computing time may be quite long (up to several hours) and that there is no guarantee that a resolvable design can be found. Thus, for large resolvable designs it is probably better to use a more specialized design package. For smaller designs, using OPTEX is usually a very convenient and efficient option for SAS users. The main advantage of OPTEX is its great versatility, including facilities to include pre-existing covariates or to deal with unbalanced block structures, as described in the advanced examples given in the online documentation.

The focus here has been on a particular procedure in the statistical package SAS. Of course there are many alternative good design packages, including CycDesigN and Gendex, which have already been mentioned. This is not the place for an extensive review. Noteworthy, additional options are the package "blockdesign" for R (also see http://www.expdesigns.co.uk/) and the DiGGer package, also available in R. Good general design facilities specifically targeted at agricultural experiments are available in the package GenStat.

ACKNOWLEDGEMENT

Thanks go to Randy Tobias (SAS Institute) for helpful comments and in particular for suggesting the code to fix the treatment replication numbers (Box 10).

REFERENCES

Atkinson A.C., Donev A.N., Tobias R.D. (2009). *Optimum experimental designs with SAS*. Oxford University Press, Oxford.

Hinkelmann K., Kempthorne O. (1994). *Design and analysis of experiments. Volume 1: Introduction to experimental design*. Wiley, New York.

John J.A., Williams E.R. (1995). *Cyclic and computer generated designs*. Chapman & Hall, London.

Mead R. (1988). *The design of experiments. Statistical principles for practical application*. Cambridge University Press, Cambridge.

Pereira C., Tobias R. (2015). Catering to your tastes: Using PROC OPTEX to design custom experiments, with applications in food science and field trials. In: *SAS Global Forum 2015*, paper 3148–2015. http://support.sas.com/resources/papers/proceedings15/3148–2015.pdf

Piepho H.P., Büchse A., Emrich, K. (2003). A hitchhiker's guide to the mixed model analysis of randomized experiments. *Journal of Agronomy and Crop Science* 189, 310–322.

Piepho H.P., Williams E.R., Ogutu J.O. (2013). A two-stage approach to recovery of inter-block information and shrinkage of block effect estimates. *Communications in Biometry and Crop Science* 8, 10–22.

Smith A.B., Lim P., Cullis B. R. (2006). The design and analysis of multi-phase plant breeding experiments. *Journal of Agricultural Science* 144, 393–409.

Williams, E.R., John, J.A., Whitaker, D. (2014). Construction of more flexible and efficient p-rep designs. *Australian and New Zealand Journal of Statistics* 56, 89–96.

Williams E.R., Piepho H.P. (2015). Optimality and contrasts in block designs with unequal treatment replication. *Australian and New Zealand Journal of Statistics* 57, 203–209.

Williams, E.R., Piepho, H.P., Whitaker, D. (2011). Augmented p-rep designs. *Biometrical Journal* 53, 19–27.

APPENDIX

An RCBD for seven treatments and four replications can be generated using a basic datastep programming as follows:

```
data RCBD;
  do trt=1 to 7;
    do rep=1 to 4;
      r=ranuni(8364331);
      output;
    end;
  end;
run;

proc sort data=RCBD;
  by rep r;
run;

proc print data=RCBD noobs;
  var rep trt;
run;
```

For the same specification, the PLAN procedure statements are:

```
proc plan seed=982261;
  factors block=4 ordered trt=7;
run;
```