A quick guide to planor, an R package for the automatic generation of regular factorial designs

Monod H., Bouvier A., Kobilinsky A.

INRA, UR 341, Unité MIA-Jouy
Mathématiques et Informatique Appliquées
F78352 Jouy en Josas Cedex
France

July 10, 2014

Contents

1 Introduction

The `planor` R package generates regular factorial designs for a wide and flexible range of user specifications. The main motivation for its creation was to generate full and fractional factorial designs, but `planor` can also be used to construct and randomize complete block designs, Latin squares, split-plot designs, *etc.* The main limitation is that `planor` generates orthogonal designs only, which excludes most incomplete block designs and diverges from an optimal design approach.

`planor` is based on algebraic methods of construction and more specifically on the key matrix method [1],[6],[15], described in detail in [7], [11], [12], and illustrated in [4]. This method produces so-called *regular* designs in which factorial effects are either estimable independently or completely confounded. The `planor` R package originates from the `planor` software which was written in the APL language by André Kobilinsky. The initial `planor` manual [8] has been adapted to the `planor` R package [10] and gives more details on the theory than this short guide.

To generate a design with `planor` , the user provides information on the design factors, on the anova model to be used when analysing the results, and on the design size. He or she then asks `planor` to search for one or more designs meeting the requirements. One `planor` function gives a design solution directly. The design can be randomized at once according to a block structure formula given by the user or it can be randomized later by a specific function. Alternatively, the solutions, if any, can be obtained as a list of design key matrices. Several specific functions then allow to investigate the solutions' properties and to print and store the resulting designs. `planor` can manage factors with different numbers of levels. It can take into account hierarchical relationships among factors. It is also possible to control the confounding of treatments effects with block effects, like in split-plot or criss-cross experiments.

This vignette presents the basic usage of `planor` . A more comprehensive presentation is under preparation, as well as additional package functions. More details are also available through the help functions of the `planor` package. For an introduction to the design of experiments, many textbooks are available in the statistics literature. For the `planor` user, we particularly recommend [2] and [7] (in French).

Please note that `planor` is still under construction. We advise to check that the designs obtained by `planor` behave as expected before using them for a real experiment, by inspecting them and conducting analysis on simulated data, for example.

2 Construction and randomization of orthogonal designs : one-step examples

In `planor` , the experiment requirements are specified in three parts : *(i)* the factors ; *(ii)* the model and (optionally) the subset of factorial effects to estimate ; *(iii)* the design size. All this information can be provided to the `regular.design` or to the `planor.designkey` functions. The function `regular.design` is simpler to use because it integrates all the steps included in `planor` for design construction. Indeed it gives the factorial design directly as a ready-to-use dataframe, if it finds a solution to the user's specifications. We illustrate this function by the construction of a few well known classes of orthogonal designs.

2.1 Full factorial design

Suppose we want to construct a full factorial design for three factors A, B, C at 2, 2, 3 levels respectively. Then the R code is:

```
> library("planor")
> ABCfull <- regular.design(factors=list(A=1:2,B=1:2,C=1:3),
+                           model=~A*B*C,
+                           nunits=2*2*3,
+                           randomize=~UNITS)

The search is closed: max.sol =  1 solution(s) found

> print(ABCfull)

An object of class "planordesign"
Slot "design":
   A B C
1  2 2 1
2  2 1 2
3  1 2 2
4  1 2 3
5  1 1 2
6  2 1 1
```

```
7  2 2 3
8  1 1 1
9  1 2 1
10 2 1 3
11 1 1 3
12 2 2 2

Slot "factors":
An object of class "designfactors"
Slot "fact.info":
  nlev block ordered model basic dummy
A    2 FALSE   FALSE  TRUE FALSE FALSE
B    2 FALSE   FALSE  TRUE FALSE FALSE
C    3 FALSE   FALSE  TRUE FALSE FALSE

Slot "pseudo.info":
  parent nlev block ordered model basic dummy
A      1    2 FALSE   FALSE  TRUE FALSE FALSE
B      2    2 FALSE   FALSE  TRUE FALSE FALSE
C      3    3 FALSE   FALSE  TRUE FALSE FALSE

Slot "levels":
$A
[1] 1 2

$B
[1] 1 2

$C
[1] 1 2 3



Slot "model":
list()

Slot "designkey":
[[1]]
An object of class keymatrix

********** Prime  2  design **********

    A B
*U* 1 0
*U* 0 1


[[2]]
An object of class keymatrix

********** Prime  3  design **********

    C
*U* 1
```

```
Slot "nunits":
[1] 12

Slot "recursive":
[1] FALSE
```

The first argument describes the factors. To get a full factorial design, the model formula includes all interactions and the size of the experiment is the product of the numbers of levels of all factors. We assume that the user wants a completely randomized design, so the randomization formula is limited to the `UNITS` level. The result is a dataframe. Based here on simulated data, the analysis gives:

```
> set.seed(123)
> dataABCfull= as.data.frame(ABCfull)
> dataABCfull$Y <- runif(2*2*3)
> ABCfull.aov <- aov(Y~A*B*C, data=dataABCfull)
> summary(ABCfull.aov)
```

```
            Df Sum Sq Mean Sq
A            1 0.3583  0.3583
B            1 0.0780  0.0780
C            2 0.1512  0.0756
A:B          1 0.0713  0.0713
A:C          2 0.1358  0.0679
B:C          2 0.1121  0.0560
A:B:C        2 0.0286  0.0143
```

2.2 Complete block design

From a combinatorial point of view, the complete block design is a special case of the full factorial design. From a statistical point of view, the main differences are that treatment and block effects are usually assumed to be additive (no interaction) and that the randomization takes into account the blocks.

Suppose there are 5 treatments and 4 blocks. The R code becomes:

```
> CBD <- regular.design(factors=list(
+                Block=1:4, Treatment=c("T1","T2","T3","T4", "T5")),
+                       model=~Block+Treatment,
+                       nunits=4*5,
+                       randomize=~Block/UNITS)
```

```
The search is closed: max.sol =  1 solution(s) found
```

```
> print(CBD)
```

```
An object of class "planordesign"
Slot "design":
   Block Treatment
1      1        T4
2      1        T5
3      1        T2
4      1        T3
5      1        T1
```

```
6      2        T3
7      2        T2
8      2        T1
9      2        T4
10     2        T5
11     3        T4
12     3        T3
13     3        T1
14     3        T2
15     3        T5
16     4        T1
17     4        T2
18     4        T3
19     4        T4
20     4        T5

Slot "factors":
An object of class "designfactors"
Slot "fact.info":
          nlev block ordered model basic dummy
Block        4 FALSE   FALSE  TRUE FALSE FALSE
Treatment    5 FALSE   FALSE  TRUE FALSE FALSE

Slot "pseudo.info":
          parent nlev block ordered model basic dummy
Block_1        1    2 FALSE   FALSE  TRUE FALSE FALSE
Block_2        1    2 FALSE   FALSE  TRUE FALSE FALSE
Treatment      2    5 FALSE   FALSE  TRUE FALSE FALSE

Slot "levels":
$Block
[1] 1 2 3 4

$Treatment
[1] "T1" "T2" "T3" "T4" "T5"



Slot "model":
list()

Slot "designkey":
[[1]]
An object of class keymatrix

********** Prime  2  design **********

    Block_1 Block_2
*U*       1       0
*U*       0       1


[[2]]
An object of class keymatrix
```

```
********** Prime  5  design **********

    Treatment
*U*         1



Slot "nunits":
[1] 20

Slot "recursive":
[1] FALSE
```

Based on simulated data again, the analysis gives:

```
> dataCBD= as.data.frame(CBD)
> dataCBD$Y <- runif(20)
> CBD.aov <- aov(Y~Block+Treatment, data=dataCBD)
> summary(CBD.aov)

            Df Sum Sq Mean Sq F value Pr(>F)
Block        3 0.0456 0.01521   0.325  0.807
Treatment    4 0.4384 0.10960   2.343  0.114
Residuals   12 0.5614 0.04678
```

2.3 Latin square

The Latin square involves three factors at n levels in n^2 units, with all three factors being pairwise orthogonal. From a combinatorial point of view, the Latin square is a fractional factorial design. From a statistical point of view, it is often used to study a treatment factor of interest and to control two block factors that are completely crossed.

Suppose that in a sensory experiment, there are 4 products to compare and 4 judges. Each judge tastes the four products during four consecutive periods. The design must ensure that the main effects of the product, judge and period factors are estimable and orthogonal, assuming the interactions are negligible. The appropriate randomization consists in permuting the judge and period labels at random, independently.

The R code is:

```
> LS <- regular.design(factors=list(
+   Judge=c("J1","J2","J3","J4"), Period=1:4, Product=c("P1","P2","P3","P4")),
+                          model= ~Judge + Period + Product,
+                          nunits=4*4,
+                          randomize=~Judge+Period)

The search is closed: max.sol =  1 solution(s) found

> print(LS)

An object of class "planordesign"
Slot "design":
   Judge Period Product
1     J1      1      P3
2     J1      2      P1
3     J1      3      P4
4     J1      4      P2
```

```
5     J2      1      P1
6     J2      2      P3
7     J2      3      P2
8     J2      4      P4
9     J3      1      P4
10    J3      2      P2
11    J3      3      P3
12    J3      4      P1
13    J4      1      P2
14    J4      2      P4
15    J4      3      P1
16    J4      4      P3

Slot "factors":
An object of class "designfactors"
Slot "fact.info":
        nlev block ordered model basic dummy
Judge      4 FALSE   FALSE  TRUE FALSE FALSE
Period     4 FALSE   FALSE  TRUE FALSE FALSE
Product    4 FALSE   FALSE  TRUE FALSE FALSE

Slot "pseudo.info":
          parent nlev block ordered model basic dummy
Judge_1        1    2 FALSE   FALSE  TRUE FALSE FALSE
Judge_2        1    2 FALSE   FALSE  TRUE FALSE FALSE
Period_1       2    2 FALSE   FALSE  TRUE FALSE FALSE
Period_2       2    2 FALSE   FALSE  TRUE FALSE FALSE
Product_1      3    2 FALSE   FALSE  TRUE FALSE FALSE
Product_2      3    2 FALSE   FALSE  TRUE FALSE FALSE

Slot "levels":
$Judge
[1] "J1" "J2" "J3" "J4"

$Period
[1] 1 2 3 4

$Product
[1] "P1" "P2" "P3" "P4"




Slot "model":
list()

Slot "designkey":
[[1]]
An object of class keymatrix

********** Prime  2  design **********

    Judge_1 Judge_2 Period_1 Period_2 Product_1 Product_2
*U*       1       0        0        0         1         0
*U*       0       1        0        0         0         1
```

```
*U*       0       0       1       0       1       0
*U*       0       0       0       1       0       1
```

```
Slot "nunits":
[1] 16

Slot "recursive":
[1] FALSE
```

The analysis gives:

```
> dataLS=as.data.frame(LS)
> dataLS$Y <- runif(16)
> LS.aov <- aov(Y~Judge + Period + Product, data=dataLS)
> summary(LS.aov)
```

```
            Df Sum Sq Mean Sq F value Pr(>F)
Judge        3 0.4454 0.14848   4.314 0.0607 .
Period       3 0.2271 0.07571   2.199 0.1890
Product      3 0.1525 0.05084   1.477 0.3125
Residuals    6 0.2065 0.03442
---
Signif. codes:  0 âĂŸ***âĂŹ 0.001 âĂŸ**âĂŹ 0.01 âĂŸ*âĂŹ 0.05 âĂŸ.âĂŹ 0.1 âĂŸ âĂŹ 1
```

2.4 Fractional factorial design of resolution 5

For the last example in this section, we consider the type of design for which `planor` was originally conceived: a highly fractionated factorial design. Such designs allow to cope with a large number of factors in (much) fewer units than required by a full factorial design. They have been used for real experiments for a long time [3] and, more recently, for computer experiments (*e.g.* [5], [14]).

A fractional design of resolution 5 is generated below for 10 factors at 4 levels, assuming that $2^{10} = 1024$ units are available instead of $4^{10} = 1\ 048\ 576$ for a full factorial design. A fraction of resolution 5 guarantees that all terms can be estimated from a model with main effects and two-factor interactions. It can be generated as follows :

```
> FFD <- regular.design(factors=LETTERS[1:10], nlevels=4,
+                       resolution=5,
+                       nunits=2^10)
```

```
The search is closed: max.sol =  1 solution(s) found
```

```
> print(dim(FFD))
```

```
NULL
```

```
> print(FFD[1:5,])
```

```
An object of class "planordesign"
Slot "design":
  A B C D E F G H I J
1 1 1 1 1 1 1 1 1 1 1
2 1 1 1 1 1 2 2 2 2 2
3 1 1 1 1 1 3 3 3 3 3
4 1 1 1 1 1 4 4 4 4 4
```

```
5 1 1 1 2 2 1 1 2 2 4

Slot "factors":
An object of class "designfactors"
Slot "fact.info":
  nlev block ordered model basic dummy
A    4 FALSE   FALSE  TRUE FALSE FALSE
B    4 FALSE   FALSE  TRUE FALSE FALSE
C    4 FALSE   FALSE  TRUE FALSE FALSE
D    4 FALSE   FALSE  TRUE FALSE FALSE
E    4 FALSE   FALSE  TRUE FALSE FALSE
F    4 FALSE   FALSE  TRUE FALSE FALSE
G    4 FALSE   FALSE  TRUE FALSE FALSE
H    4 FALSE   FALSE  TRUE FALSE FALSE
I    4 FALSE   FALSE  TRUE FALSE FALSE
J    4 FALSE   FALSE  TRUE FALSE FALSE

Slot "pseudo.info":
    parent nlev block ordered model basic dummy
A_1      1    2 FALSE   FALSE  TRUE FALSE FALSE
A_2      1    2 FALSE   FALSE  TRUE FALSE FALSE
B_1      2    2 FALSE   FALSE  TRUE FALSE FALSE
B_2      2    2 FALSE   FALSE  TRUE FALSE FALSE
C_1      3    2 FALSE   FALSE  TRUE FALSE FALSE
C_2      3    2 FALSE   FALSE  TRUE FALSE FALSE
D_1      4    2 FALSE   FALSE  TRUE FALSE FALSE
D_2      4    2 FALSE   FALSE  TRUE FALSE FALSE
E_1      5    2 FALSE   FALSE  TRUE FALSE FALSE
E_2      5    2 FALSE   FALSE  TRUE FALSE FALSE
F_1      6    2 FALSE   FALSE  TRUE FALSE FALSE
F_2      6    2 FALSE   FALSE  TRUE FALSE FALSE
G_1      7    2 FALSE   FALSE  TRUE FALSE FALSE
G_2      7    2 FALSE   FALSE  TRUE FALSE FALSE
H_1      8    2 FALSE   FALSE  TRUE FALSE FALSE
H_2      8    2 FALSE   FALSE  TRUE FALSE FALSE
I_1      9    2 FALSE   FALSE  TRUE FALSE FALSE
I_2      9    2 FALSE   FALSE  TRUE FALSE FALSE
J_1     10    2 FALSE   FALSE  TRUE FALSE FALSE
J_2     10    2 FALSE   FALSE  TRUE FALSE FALSE

Slot "levels":
$A
[1] 1 2 3 4

$B
[1] 1 2 3 4

$C
[1] 1 2 3 4

$D
[1] 1 2 3 4

$E
```

```
[1] 1 2 3 4

$F
[1] 1 2 3 4

$G
[1] 1 2 3 4

$H
[1] 1 2 3 4

$I
[1] 1 2 3 4

$J
[1] 1 2 3 4



Slot "model":
list()

Slot "designkey":
[[1]]
An object of class keymatrix

********** Prime  2  design **********

    A_1 A_2 B_1 B_2 C_1 C_2 D_1 D_2 E_1 E_2 F_1 F_2 G_1 G_2 H_1 H_2 I_1 I_2 J_1
*U*   1   0   0   0   0   0   0   0   1   0   0   0   0   1   1   0   0   0   1
*U*   0   1   0   0   0   0   0   0   0   1   0   0   1   1   0   1   0   0   0
*U*   0   0   1   0   0   0   0   0   1   0   0   0   1   0   0   1   1   0   0
*U*   0   0   0   1   0   0   0   0   0   1   0   0   0   1   1   1   0   1   0
*U*   0   0   0   0   1   0   0   0   1   0   0   0   1   0   0   0   0   1   1
*U*   0   0   0   0   0   1   0   0   0   1   0   0   0   1   0   0   1   1   0
*U*   0   0   0   0   0   0   1   0   1   0   0   0   0   0   1   0   1   0   0
*U*   0   0   0   0   0   0   0   1   0   1   0   0   0   0   0   1   0   1   1
*U*   0   0   0   0   0   0   0   0   0   0   1   0   1   0   1   0   1   0   1
*U*   0   0   0   0   0   0   0   0   0   0   0   1   0   1   0   1   0   1   0
    J_2
*U*   0
*U*   1
*U*   0
*U*   0
*U*   0
*U*   1
*U*   1
*U*   1
*U*   0
*U*   1



Slot "nunits":
```

```
[1] 1024

Slot "recursive":
[1] FALSE
```

Note that in `planor` syntax, it is equivalent, but shorter, to specify `resolution = 5` rather than

`model = (A+B+C+D+E+F+G+H+I+J)^2.`

3 Construction of regular factorial designs through the search for a design key

We now adopt a more progressive way to construct the design. For this reason, we focus on the `planor.designkey` function rather than `regular.design`. In that case, design construction involves two main steps :

1. the search for key matrices (function `planor.designkey`);
2. then the design generation and randomization (function `planor.design`).

We start by a short technical subsection. It cannot go into details, but we hope it helps to make a link with other approaches to the construction of regular factorial designs.

3.1 A very short technical point

A key matrix of base p in `planor` is a matrix of integers modulo p, where p is a prime. It encodes the information required to construct a regular factorial design for factors at p levels.

Consider for example a design for 4 factors A, B, C, D at $p = 2$ levels in $2^3 = 8$ units, whereas a full factorial design would require $2^4 = 16$ units. It is possible to construct a design which allows to estimate the main effects of the factors assuming the three- and four-factor interactions are negligible. The solution is explained in many books on factorial designs (*e.g.* [3]) :

- assimilate the factors' levels to $0, 1$ mod 2;
- make a full factorial design on A, B, C;
- add the level of D on each unit by the equation $D = A + B + C$ mod 2, called the *defining relationship* of the design.

Then it can be shown that the interaction $A.B.C.D$ is confounded with the general mean, the main effect A is confounded with the interaction $B.C.D$, etc.

In `planor` , this construction is encoded in the following key matrix of base 2 :

$$K = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

The rows of K are associated with three factors U_1, U_2, U_3 which are called the units factors. The idea is that the set of units can be identified to the full factorial design on these units factors. The columns of K are associated with the treatment factors A, B, C, D. Here the first column of K means that in the design, we must have $A = U_1$ (modulo 2). The second, third and fourth columns mean $B = U_2$, $C = U_3$ and $D = U_1 + U_2 + U_3$, respectively. It follows that the defining relationship $D = A + B + C$ mod 2 will be satisfied.

The core algorithm in `planor` basically constructs K by searching for its columns successively, using a backtrack algorithm. However, there is also much pre-processing to turn the factors and model specifications into appropriate constraints on the columns of K. In particular, all factors are automatically decomposed into pseudofactors which all have a prime number of levels, and the whole problem is decomposed according to the different prime numbers involved.

A detailed presentation of the methodology implemented in `planor` is under preparation [9]. See also the references given in the introduction or [16] for the extension of regular factorial designs to the case when different primes are involved.

3.2 Fractional designs with 2-level factors

3.2.1 Search for a design key

Consider an experiment to study four treatment factors A, B, C, D at two levels, using two blocks of size four. A full factorial design on the treatment factors would require 16 units. Only eight are available so that a fractional design must be used. In addition, some treatment effects are necessarily confounded with the block effect.

At first, we may look for a design adapted to the model that includes the main effects of the block and treatment factors, as well as the interactions between pairs of treatment factors :

```
> ex1Key <- planor.designkey(factors=c("block","A","B","C","D"),nlevels=rep(2,5),
+                            model=~block+(A+B+C+D)^2,
+                            nunits=2^3)

Preliminary step 1 : processing the model specifications
Preliminary step 2 : performing prime decompositions on the factors
Main step for prime p = 2 : key-matrix search
  => search for columns 2 to 5
      first visit to column 2
      first visit to column 3
      first visit to column 4
      first visit to column 5
The search is closed:  0 solutions found
```

It turns out that `planor` fails to find a solution. There is indeed no solution to the problem.

For the second try, we keep the same model but relax the implicit constraint to estimate all factorial terms in the model. This is done by using the `estimate` argument of the `planor.designkey` function. This argument is optional : by default, it is considered that all terms in the `model` formula must be estimated. In contrast, we only require below that the main effects of the treatment factors be estimable. It follows that we now allow for designs in which two-factor interactions are mutually confounded.

```
> ex1Key <- planor.designkey(factors=c("block","A","B","C","D"),nlevels=rep(2,5),
+                            model=~block+(A+B+C+D)^2,
+                            estimate=~A+B+C+D,
+                            nunits=2^3)

Preliminary step 1 : processing the model specifications
Preliminary step 2 : performing prime decompositions on the factors
Main step for prime p = 2 : key-matrix search
  => search for columns 2 to 5
      first visit to column 2
      first visit to column 3
      first visit to column 4
      first visit to column 5
The search is closed: max.sol =  1 solution(s) found
```

During the search, the backtrack algorithm looks successively for new columns to add to the key matrix. Succinct information is given to check the algorithm progress (default argument verbose=TRUE). The search stops as soon as all columns of the key matrix have been found (default argument max.sol=1).

An alternative to using planor.designkey directly is to provide the information on the experiment step by step with the functions planor.factors and planor.model. The idea is to store the results of these functions in R objects and use them as arguments to planor.designkey. This may be convenient, for example, when one wants to explore several possible models and design sizes with the same set of factors.

```
> ex1Fac <- planor.factors(factors=c("block","A","B","C","D"), nlevels=rep(2,5),
+                           block=~block)
> ex1Mod <- planor.model( model=~block+(A+B+C+D)^2, estimate=~A+B+C+D )
> ex1Key <- planor.designkey(factors=ex1Fac, model=ex1Mod, nunits=2^3)

Preliminary step 1 : processing the model specifications
Preliminary step 2 : performing prime decompositions on the factors
Main step for prime p = 2 : key-matrix search
  => search for columns 2 to 5
      first visit to column 2
      first visit to column 3
      first visit to column 4
      first visit to column 5
The search is closed: max.sol =  1 solution(s) found
```

3.2.2 Design-key properties

In both cases, the key matrix solution is stored in the object ex1Key. Its detailed properties can be obtained by two different functions. The summary function prints the key matrix and the defining relationships associated with this key matrix. More detailed information on the aliasing between factorial effects is given by the function alias.

Note that we have used the optional block argument of planor.factors (also available in planor.designkey). It specifies the factors that should be considered as block (or nuisance) factors. In planor , the distinction between *treatment* and *block* factors is taken into account when studying confounding and aliasing properties.

```
> summary(ex1Key, show="dtb")

********** Prime  2  design **********

--- Solution  1  for prime  2  ---

DESIGN KEY MATRIX
    block A B C D
*U*     1 0 1 0 1
*U*     0 1 1 0 0
*U*     0 0 0 1 1

TREATMENT EFFECTS CONFOUNDED WITH THE MEAN
1 = A B C D

BLOCK-and-TREATMENT EFFECTS CONFOUNDED WITH THE MEAN
1 = block A B
1 = block C D

> alias(ex1Key)
```

```
********** Prime  2  design **********

--- Solution  1  for prime  2  ---

UNALIASED TREATMENT EFFECTS
A ; B ; C ; D

ALIASED TREATMENT EFFECTS
A:C = B:D
A:D = B:C

TREATMENT AND BLOCK EFFECTS CONFOUNDED WITH BLOCK EFFECTS
block = A:B = C:D

UNALIASED BLOCK EFFECTS
nil


--- Synthesis on the aliased treatment effects for prime  2  ---

     unaliased trt.aliased blc.aliased
[1,]         4           4           2
```

3.2.3 Design generation

Last but not least, a design can be generated by the function `planor.design`. The design itself is the object in slot `design` of the more complex object generated by `planor.design`. An option allows the design to be randomized, according to a block structure formula that the user must specify (option `randomize`).

```
> ex1Des <- planor.design(ex1Key)
> print(getDesign(ex1Des))

  block A B C D
1     1 1 1 1 1
2     1 1 1 2 2
3     1 2 2 1 1
4     1 2 2 2 2
5     2 1 2 1 2
6     2 1 2 2 1
7     2 2 1 1 2
8     2 2 1 2 1

> ex1Rand <- planor.design(ex1Key, randomize=~block/UNITS)
> print(getDesign(ex1Rand))

  block A B C D
1     1 1 2 1 2
2     1 1 2 2 1
3     1 2 1 1 2
4     1 2 1 2 1
5     2 1 1 1 1
6     2 2 2 1 1
7     2 2 2 2 2
8     2 1 1 2 2
```

3.3 Fractional designs with 3-level factors

We keep the same example but with 3-level factors and a few more options. The results are not shown for sake of brevity.

```
> # ***************** EXAMPLE 2 *****************
> # Four 3-level treatment factors and one 3-level block factor
> # Model: block+(A+B+C+D)^2  -   Estimate: A+B+C+D
> # N = 3^3 = 27 units
> #
> ex2Key <- planor.designkey(factors=c(LETTERS[1:4],"block"),
+                            nlevels=rep(3,5),
+                            block=~block,
+                            model=~block+(A+B+C+D)^2,
+                            estimate=~A+B+C+D,
+                            nunits=3^3, base=~A+B+C, max.sol=2)
> summary(ex2Key)
> summary(ex2Key)
> ex2Des <- planor.design(ex2Key[2])
```

Two optional arguments of `planor.designkey` have been used, first to specify that A, B and C should be used as basic factors, and second to ask for two solutions whereas the default is one. Both solutions are examined by `summary` and the second one, say, is chosen by the user to generate a factorial design. When *basic* factors are specified, they are identified to the units factors U_i [8]. As a consequence, all combinations of the basic factors are guaranteed to be included in the design. When relevant, using basic factors is recommended because it can speed up the search.

The following lines also work; they illustrate that the basic factors need not be part of the model but they must have been declared in `planor.factors`.

```
> ex2Fac <- planor.factors(factors=c(LETTERS[1:4], "block", "BASE"),
+                          nlevels=rep(3,6) )
> ex2Mod <- planor.model(model=~block+(A+B+C+D)^2,
+                        estimate=~A+B+C+D )
> ex2Key <- planor.designkey(factors=ex2Fac,
+                            model=ex2Mod,
+                            nunits=3^3,
+                            base=~A+B+BASE,
+                            max.sol=2)
```

3.4 Asymmetric fractional factorial designs

A regular fractional factorial design is called mixed or asymmetric when the numbers of levels of the factors involve several different prime numbers. The asymmetric designs constructed in `planor` consist of the cross products of designs based on each prime. This does not allow for a great flexibility in terms of confounding, but it enlarges the scope of situations that can be addressed.

```
> # Four treatment factors at 6, 6, 4, 2 levels and one 6-level block factor
> # Model: block+(A+B+C+D)^2 ; Estimate: A+B+C+D\n")
> # N = 144 = 2^4 x 3^2 experimental units
> mixKey <- planor.designkey(factors=c( LETTERS[1:4], "block"),
+                            nlevels=c(6,6,4,2,6),
+                            block=~block,
+                            model=~block+(A+B+C+D)^2,
+                            estimate=~A+B+C+D,
+                             nunits=144,
+                         base=~A+B+D, max.sol=2)
```

```
Preliminary step 1 : processing the model specifications
Preliminary step 2 : performing prime decompositions on the factors
Main step for prime p = 2 : key-matrix search
  => search for columns 4 to 6
      first visit to column 4
      first visit to column 5
      first visit to column 6
The search is closed: max.sol =  2 solution(s) found
Main step for prime p = 3 : key-matrix search
  => search for column 3 .
      first visit to column 3
The search is closed: max.sol =  2 solution(s) found
```

```
> summary(mixKey)
```

```
********** Prime  2  design **********

--- Solution  1  for prime  2  ---

TREATMENT EFFECTS CONFOUNDED WITH THE MEAN
1 = A_1 B_1 D C_1

BLOCK-and-TREATMENT EFFECTS CONFOUNDED WITH THE MEAN
1 = A_1 B_1 block_1
1 = D C_1 block_1

WEIGHT PROFILES
Treatment effects confounded with the mean: 4^1
Treatment effects confounded with block effects: 2^2
Treatment pseudo-effects confounded with the mean: 4^1
Treatment pseudo-effects confounded with block effects: 2^2

--- Solution  2  for prime  2  ---

TREATMENT EFFECTS CONFOUNDED WITH THE MEAN
1 = A_1 B_1 D C_1

BLOCK-and-TREATMENT EFFECTS CONFOUNDED WITH THE MEAN
1 = A_1 D block_1
1 = B_1 C_1 block_1

WEIGHT PROFILES
Treatment effects confounded with the mean: 4^1
Treatment effects confounded with block effects: 2^2
Treatment pseudo-effects confounded with the mean: 4^1
Treatment pseudo-effects confounded with block effects: 2^2


********** Prime  3  design **********

--- Solution  1  for prime  3  ---
```

```
TREATMENT EFFECTS CONFOUNDED WITH THE MEAN
nil

BLOCK-and-TREATMENT EFFECTS CONFOUNDED WITH THE MEAN
1 = A_2^2  B_2^2  block_2

WEIGHT PROFILES
Treatment effects confounded with the mean: none
Treatment effects confounded with block effects: 2^1
Treatment pseudo-effects confounded with the mean: none
Treatment pseudo-effects confounded with block effects: 2^1

--- Solution  2  for prime  3  ---

TREATMENT EFFECTS CONFOUNDED WITH THE MEAN
nil

BLOCK-and-TREATMENT EFFECTS CONFOUNDED WITH THE MEAN
1 = A_2 B_2^2  block_2

WEIGHT PROFILES
Treatment effects confounded with the mean: none
Treatment effects confounded with block effects: 2^1
Treatment pseudo-effects confounded with the mean: none
Treatment pseudo-effects confounded with block effects: 2^1
```

```
> mixPlan <- planor.design(key=mixKey, select=c(1,1), randomize=~block/UNITS)
> print(getDesign(mixPlan)[1:25,])
```

```
    A B D C block
1   6 6 1 2     1
6   1 2 1 2     1
8   4 5 1 1     1
10  6 6 2 3     1
15  3 3 2 4     1
17  2 1 2 3     1
19  5 4 2 3     1
24  3 3 1 2     1
26  2 1 1 2     1
28  5 4 2 4     1
33  5 4 1 2     1
35  4 5 2 4     1
109 2 1 1 1     1
114 6 6 2 4     1
116 2 1 2 4     1
118 1 2 2 3     1
123 4 5 1 2     1
125 3 3 2 3     1
127 4 5 2 3     1
132 6 6 1 1     1
134 3 3 1 1     1
136 1 2 2 4     1
141 5 4 1 1     1
143 1 2 1 1     1
2   3 4 2 2     2
```

The algorithm starts by decomposing the factors into pseudofactors that all have a prime number of levels. Then it performs a similar decomposition of the `model` and `estimate` terms. After these initial steps, separate key-matrix searches are performed, one for each prime involved in the problem. The prime decompositions are automatic and transparent to the user. The recomposition when generating a design is transparent too. In contrast, most information on the search process and on the fraction properties are given according to the prime decompositions.

3.5 Split-plot designs

In a split-plot experiment, there are two treatment factors `variety` and `fert`, say, at m and n levels respectively. The block structure consists of r blocks each containing m sub-blocks of size n and the factor `variety` is constrained to be constant within sub-blocks.

In an orthogonal split-plot design, each variety occupies one sub-block of each block, and each sub-block contains the n distinct levels of factor `fert`. In `planor` , this design can be constructed by defining the block structure as a cross between a `block` and a `subblock` factor. The `hierarchy` argument is used to specify that `variety` must be constant within the combinations of `block` and `subblock`. Two model-estimate pairs are given to the `listofmodels` argument. First, the main effect of `fert` and the interaction between `fert` and `variety` must be estimable when blocks and sub-blocks are included in the model. Second, the main effect of `variety` must be estimable between sub-blocks, that is, when blocks but not sub-blocks are included in the model. The command below calculates the design key of a split-plot design with $r = 2$, $n = 2$, $m = 2$.

```
> splitKey <- planor.designkey(factors=list(block=1:2,
+                                 subblock=1:2,
+                                 variety=LETTERS[1:2],
+                                 fert=c("organic","mineral")),
+                               block=~block+subblock,
+                               hierarchy=list(~variety/(block*subblock)),
+                               listofmodels=
+                               list(c( ~block*subblock+variety*fert, ~fert+fert:variety),
+                                    c( ~block+variety,               ~variety)),
+                               nunits=2*2*2,
+                               base=~block+subblock)

Preliminary step 1 : processing the model specifications
Preliminary step 2 : performing prime decompositions on the factors
Main step for prime p = 2 : key-matrix search
  => search for columns 3 to 4
      first visit to column 3
      first visit to column 4
The search is closed: max.sol =  1 solution(s) found

> summary(splitKey)

********** Prime  2  design **********

--- Solution  1  for prime  2  ---

TREATMENT EFFECTS CONFOUNDED WITH THE MEAN
nil

BLOCK-and-TREATMENT EFFECTS CONFOUNDED WITH THE MEAN
1 = subblock variety

WEIGHT PROFILES
```

```
Treatment effects confounded with the mean: none
Treatment effects confounded with block effects: 1^1
Treatment pseudo-effects confounded with the mean: none
Treatment pseudo-effects confounded with block effects: 1^1
```

```
> alias(splitKey)
```

```
********** Prime  2  design **********

--- Solution  1  for prime  2  ---

UNALIASED TREATMENT EFFECTS
nil

ALIASED TREATMENT EFFECTS
nil

TREATMENT AND BLOCK EFFECTS CONFOUNDED WITH BLOCK EFFECTS
nil

UNALIASED BLOCK EFFECTS
block ; subblock


--- Synthesis on the aliased treatment effects for prime  2  ---

     unaliased trt.aliased blc.aliased
[1,]         0           0           0
```

```
> print(getDesign(planor.design(splitKey, randomize=~block/subblock/UNITS)))
```

```
  block subblock variety    fert
1     1        1       A organic
2     1        1       A mineral
3     1        2       B organic
4     1        2       B mineral
5     2        1       B organic
6     2        1       B mineral
7     2        2       A mineral
8     2        2       A organic
```

An alternative command to get the split-plot is given below. The main difference is that the `subblock` factor now takes rm levels and is considered as nested in `block` rather than crossed with it.

```
> splitKey <- planor.designkey(factors=list(block=1:2,
+                                 subblock=1:4,
+                                 variety=LETTERS[1:2],
+                                 fert=c("organic","mineral")),
+                               block=~block+subblock,
+                               hierarchy=list( ~block/subblock, ~variety/subblock),
+                               listofmodels=
+                               list(c( ~subblock+variety*fert, ~fert+fert:variety),
+                                    c( ~block+variety,               ~variety)),
+                               nunits=2*2*2,
+                               base=~subblock)
```

```
Preliminary step 1 : processing the model specifications
Preliminary step 2 : performing prime decompositions on the factors
Main step for prime p = 2 : key-matrix search
  => search for columns 3 to 5
      first visit to column 3
      first visit to column 4
      first visit to column 5
The search is closed: max.sol =  1 solution(s) found
```

```
> print(getDesign(planor.design(splitKey, randomize=~block/subblock/UNITS)))
```

```
  subblock block variety    fert
1        1     1       B organic
2        1     1       B mineral
3        2     1       A mineral
4        2     1       A organic
5        3     2       B mineral
6        3     2       B organic
7        4     2       A organic
8        4     2       A mineral
```

3.6 Fractional designs with nested factors and a complex block structure

We now consider an experiment with concrete and more complex specifications. This example stems from an experiment to study the cleaning of surfaces by a robot, see [8], example 3 on page 3. There are five treatment factors at 2 levels. The block structure consists of four plates with 2 rows and 4 columns per plate, resulting in 32 experimental units. In addition, the design must cope with experimental constraints between treatment and block factors. The treatment factors concentration (`conc`) and temperature (`Tact`) must remain constant within a plate. The treatment factors denoted by `nsoil` and `qsoil` must remain constant within each column of each plate. Only treatment factor rugosity (`Rug`) can be modified freely between experimental units.

To begin with, we show how to specify user-defined factor levels, by providing a list to the `factors` argument of `planor.factors`. Then, experimental constraints are specified through the `hierarchy` argument of `planor.factors`.

```
> # ************ ROBOT1A EXAMPLE *************
> # Block structure: 4 plates / (2 rows x 4 columns)
> # Treatments: 4 2-level factors
> # Hierarchy 1: conc constant in plate
> # Hierarchy 2: Tact constant in plate
> # Hierarchy 3: nsoil constant in plate x column
> # Hierarchy 4: qsoil constant in plate x column
> # N = 32 units
> #
> robotFac <- planor.factors( factors=list(
+                              conc=c(1,3),
+                              Tact=c(15,30),
+                              nsoil=c("curd","Saint-Paulin"),
+                              qsoil=c("0.01g","0.10g"),
+                              Rug=c(0.25,0.73),
+                              plate=1:4,
+                              row=1:2,
+                              col=1:4),
+                             hierarchy=list(~conc/plate,
+                             ~Tact/plate,
+                             ~nsoil/(plate*col),
+                             ~qsoil/(plate*col)))
```

This example requires several model-estimate combinations. The main model-estimate pair contains all the treatment factorial effects but no block effect. It guarantees that all treatment combinations will be present in the design, since all treatment factorial effects are required to be estimable in the model with no block effect. The second model-estimate pair (`listofmodels` argument) ensures that the `Rug` factor is orthogonal to block factors.

```
> robotMod <- planor.model( model=~nsoil*qsoil*Rug*conc*Tact,
+                           listofmodels=list(c(~plate+row+col+Rug, ~Rug)) )
```

The `base` option of the `planor.designkey` function is used here to impose that experimental units be associated with the combinations of the block factors.

```
> robotKey <- planor.designkey(factors=robotFac, model=robotMod,
+                               nunits=32, base=~plate+row+col)

Preliminary step 1 : processing the model specifications
Preliminary step 2 : performing prime decompositions on the factors
Main step for prime p = 2 : key-matrix search
  => search for columns 6 to 10
      first visit to column 6
      first visit to column 7
      first visit to column 8
      first visit to column 9
      first visit to column 10
The search is closed: max.sol =  1 solution(s) found

> summary(robotKey[1])

********** Prime  2  design **********

DESIGN KEY MATRIX
        plate_1 plate_2 row col_1 col_2 conc Tact nsoil qsoil Rug
plate_1       1       0   0     0     0    1    0     0     0   1
plate_2       0       1   0     0     0    0    1     0     0   0
row           0       0   1     0     0    0    0     0     0   1
col_1         0       0   0     1     0    0    0     1     0   0
col_2         0       0   0     0     1    0    0     0     1   0

TREATMENT EFFECTS CONFOUNDED WITH THE MEAN
1 = plate_1 conc
1 = plate_2 Tact
1 = col_1 nsoil
1 = col_2 qsoil
1 = plate_1 plate_2 conc Tact
1 = col_1 col_2 nsoil qsoil
1 = plate_1 row Rug
1 = row conc Rug
1 = plate_1 col_1 conc nsoil
1 = plate_2 col_1 Tact nsoil
1 = plate_1 col_2 conc qsoil
1 = plate_2 col_2 Tact qsoil
1 = plate_1 plate_2 row Tact Rug
1 = plate_1 plate_2 col_1 conc Tact nsoil
```

```
1 = plate_1 plate_2 col_2 conc Tact qsoil
1 = plate_1 col_1 col_2 conc nsoil qsoil
1 = plate_2 col_1 col_2 Tact nsoil qsoil
1 = plate_2 row conc Tact Rug
1 = plate_1 row col_1 nsoil Rug
1 = row col_1 conc nsoil Rug

BLOCK-and-TREATMENT EFFECTS CONFOUNDED WITH THE MEAN
nil

WEIGHT PROFILES
Treatment effects confounded with the mean: 2^4 3^4 4^5 5^9 6^5 7^3 8^1
Treatment effects confounded with block effects: none
Treatment pseudo-effects confounded with the mean: 2^4 4^6 3^2 5^6 6^4 8^1 7^6 9^2
Treatment pseudo-effects confounded with block effects: none
```

```
> robotDes <- planor.design(robotKey[1], randomize=~plate/(row*col))
> print(getDesign(robotDes))
```

```
   plate row col conc Tact        nsoil qsoil  Rug
1      1   1   1    1   15         curd 0.01g 0.25
2      1   1   2    1   15         curd 0.10g 0.25
3      1   1   3    1   15 Saint-Paulin 0.01g 0.25
4      1   1   4    1   15 Saint-Paulin 0.10g 0.25
5      1   2   1    1   15         curd 0.01g 0.73
6      1   2   2    1   15         curd 0.10g 0.73
7      1   2   3    1   15 Saint-Paulin 0.01g 0.73
8      1   2   4    1   15 Saint-Paulin 0.10g 0.73
9      2   1   1    3   15 Saint-Paulin 0.10g 0.73
10     2   1   2    3   15         curd 0.10g 0.73
11     2   1   3    3   15         curd 0.01g 0.73
12     2   1   4    3   15 Saint-Paulin 0.01g 0.73
13     2   2   1    3   15 Saint-Paulin 0.10g 0.25
14     2   2   2    3   15         curd 0.10g 0.25
15     2   2   3    3   15         curd 0.01g 0.25
16     2   2   4    3   15 Saint-Paulin 0.01g 0.25
17     3   1   1    1   30         curd 0.01g 0.25
18     3   1   2    1   30 Saint-Paulin 0.10g 0.25
19     3   1   3    1   30         curd 0.10g 0.25
20     3   1   4    1   30 Saint-Paulin 0.01g 0.25
21     3   2   1    1   30         curd 0.01g 0.73
22     3   2   2    1   30 Saint-Paulin 0.10g 0.73
23     3   2   3    1   30         curd 0.10g 0.73
24     3   2   4    1   30 Saint-Paulin 0.01g 0.73
25     4   1   1    3   30 Saint-Paulin 0.10g 0.73
26     4   1   2    3   30         curd 0.01g 0.73
27     4   1   3    3   30 Saint-Paulin 0.01g 0.73
28     4   1   4    3   30         curd 0.10g 0.73
29     4   2   1    3   30 Saint-Paulin 0.10g 0.25
30     4   2   2    3   30         curd 0.01g 0.25
31     4   2   3    3   30 Saint-Paulin 0.01g 0.25
32     4   2   4    3   30         curd 0.10g 0.25
```

Acknowledgements

This vignette was typed using the Sweave package (Leisch, 2002[13]).

References

[1] R. Bailey – "Factorial design and abelian groups", *Linear Algebra Appl.* **70** (1985), no. 349-368.

[2] — , *Design of comparative experiments*, Cambridge Series in Statistical and Probabilistic Mathematics, Cambridge University Press, 2008.

[3] G. Box, W. Hunter & J. Hunter – *Statistics for experimenters*, Wiley, 1978.

[4] S. Cliquet, C. Durier & A. Kobilinsky – "Principle of a fractional factorial design for qualitative and quantitative factors: application to the production of *Bradyrhizobium japonicum* in culture media", *Agronomie* **14** (1994), p. 569–587.

[5] A. Courcoul, H. Monod, M. Nielen, D. Klinkenberg, L. Hogerwerf, F. Beaudeau & E. Vergu – "Modelling the effect of heterogeneity of shedding on the within herd *Coxiella burnetii* spread and identification of key parameters by sensitivity analysis", *Journal of Theoretical Biology* **284** (2011), p. 130–141.

[6] M. Franklin – "Selecting defining contrasts and confounded effects in p^{n-m} factorial experiments", *Technometrics* **27** (1985), p. 165–172.

[7] A. Kobilinsky – "Les plans factoriels", in *Plans d'expériences: applications à l'entreprise* (J. Droesbeke, J. Fine & G. Saporta, éds.), Technip, Paris, 1997, p. 69–209 (Chapter 3).

[8] — , "PLANOR : program for the automatic generation of regular experimental designs. version 2.2 for Windows", Tech. report, MIA Unit, INRA Jouy en Josas, 2005.

[9] A. Kobilinsky, R. Bailey & H. Monod – "Automatic generation of generalised regular factorial designs", *In prep.* **x** (2012), p. x.

[10] A. Kobilinsky, A. Bouvier & H. Monod – "PLANOR : an R package for the automatic generation of regular fractional factorial designs. Version 1.0", Tech. report, MIA Unit, INRA Jouy en Josas, 2011.

[11] A. Kobilinsky & H. Monod – "Experimental design generated by group morphisms: an introduction", *Scand. J. Statist.* **18** (1991), p. 119–134.

[12] — , "Juxtaposition of regular factorial designs and the complex linear model", *Scand. J. Statist* **22** (1995), p. 223–254.

[13] F. Leisch – "Sweave: Dynamic generation of statistical reports using literate data analysis", in *Compstat 2002 — Proceedings in Computational Statistics* (W. Härdle & B. Rönz, éds.), Physica Verlag, Heidelberg, 2002, ISBN 3-7908-1517-9, p. 575–580.

[14] A. Lurette, S. Touzeau, M. Lamboni & H. Monod – "Sensitivity analysis to identify key parameters influencing *Salmonella* infection dynamics in a pig batch", *Journal of Theoretical Biology* **258** (2009), no. 1, p. 43–52.

[15] H. Patterson & R. Bailey – "Design keys for factorial experiments", *Appl. Statist.* **27** (1978), p. 335–343.

[16] G. Pistone & M.-P. Rogantin – "Indicator function and complex coding for mixed fractional factorial designs", *Journal of Statistical Planning and Inference* **138** (2008), no. 3, p. 787 – 802.