Fachbereich II – Mathematik - Physik - Chemie

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

01/2016

Ulrike Grömping

R Package DoE.base for Factorial Experiments

R-Paket DoE.base für Faktorielle Versuche
(englischsprachig)

Reports in Mathematics, Physics and Chemistry
Berichte aus der Mathematik, Physik und Chemie

ISSN (print): 2190-3913

Reports in Mathematics, Physics and Chemistry

Berichte aus der Mathematik, Physik und Chemie

The reports are freely available via the Internet:
http://www1.beuth-hochschule.de/FB_II/reports/welcome.htm

01/2016, February 2016

R Package DoE.base for Factorial Experiments
R-Paket DoE.base für Faktorielle Versuche (englischsprachig)

Editorial notice / Impressum

Published by / Herausgeber:
Fachbereich II
Beuth Hochschule für Technik Berlin
Luxemburger Str. 10
D-13353 Berlin
Internet: http://public.beuth-hochschule.de/FB_II/
E-Mail: fbiireports@beuth-hochschule.de

Responsibility for the content rests with the author(s) of the reports.
Die inhaltliche Verantwortung liegt bei den Autor/inn/en der Berichte.

ISSN (print): 2190-3913

R Package DoE.base for Factorial Experiments

Ulrike Grömping
Beuth University of Applied Sciences Berlin

Abstract

The R package **DoE.base** can be used for creating full factorial designs and general factorial experiments based on orthogonal arrays. Besides design creation, some analysis functionality is also available, particularly (augmented) half-normal effects plots. In addition to this specific functionality, the package provides convenience features for analysing experimental designs and the infrastructure for a suite of further packages on designing and analyzing experiments. This infrastructure is available for use also by further design of experiments packages.

Keywords: Design of Experiments, DoE, factorial designs, **DoE.base**.

1. Introduction

Factorial experiments are very common in industrial experimentation. The most widely spread such experiments use 2-level factors only, but experiments with mixed level factors are also quite common, for example with the 18 run experimental plan proposed by Taguchi (NIST/SEMATECH 2012). The design and execution of such experiments is often done during everyday work without support from a statistical expert – thus it is important to have a software available that can be safely used by non-experts. At the same time, statisticians are often involved in the more important industrial experiments, and there are many facets to construction of such experiments for which a statistician very much appreciates support from a powerful software. The R package **DoE.base** (Grömping 2015b) targets both non-experts and statisticians. It is part of a larger package suite containing also the packages **FrF2**, **DoE.wrapper** and **RcmdrPlugin.DoE**, and a fifth supporting package **FrF2.catlg128** (Grömping 2014b,c, 2013b,d, 2011b, 2013c). All these packages and all packages on which **DoE.base** depends (Chasalow 2012; Venables 2013; Venables and Ripley 2002; Meyer, Zeileis, and Hornik 2013) are available from the Comprehensive R Archive Network CRAN, which also holds the software R itself (R Development Core Team 2015). The GUI package **Rcmdr-Plugin.DoE**, which will not be described in this article, provides access to some functionality from the package suite. Grömping (2011b) gives a detailed example-based tutorial for using it. Package **DoE.base** provides the infrastructure for the entire package suite, in particular the class `design`, functions for importing and exporting experimental designs, and simple analysis functions for printing, summarizing, plotting, and modeling design data.

Besides providing infrastructure, the main contribution of package **DoE.base** is to offer features for creating factorial designs: potentially blocked full factorials (function `fac.design`) and catalog-based general factorials (function `oa.design`) are available. Functions `fac.design` and `oa.design` have taken inspiration from the "white book" (Chambers and Hastie 1984), where these S functions are described that never made it into base R. The most advanced contributions of the package are the features around orthogonal arrays (function `oa.design`), which are subject to ongoing research. These rely heavily on a catalog of orthogonal arrays, most of which have been taken from Kuhfeld (2010). To the author's knowledge, the package is the only place in R where non-regular orthogonal arrays other than Plackett-Burman designs are provided for experimentation. Non-regularity of an array has been discussed to be beneficial for screening experiments because of their good projectivity properties (see, e.g., Box and Tyssedal 1996; Deng and Tang 1999; Tang and Deng 1999). This discussion has so far focused on 2-level designs, but should analogously apply to more general factorial designs.

There is another R package closely related to the design creation functionality of package **DoE.base**: the R package **planor** (Kobilinsky, Bouvier, and Monod 2015a) can create regular fractional factorial designs in a general sense (see also Kobilinsky, Monod, and Bailey 2015b). Package **DoE.base** is more general than package **planor** in that it also creates non-regular designs, can calculate various types of quality criteria, and does not require specification of a model but can optimize a design with respect to model robustness criteria. It is less general than **planor** in that it does not allow to specify a model and estimable effects, i.e., it treats all effects of the same order on an equal footing. Sections 4 and Section 5.3 will illustrate function `regular.design` of package **planor** as an alternative to functions from packages **DoE.base** and **FrF2**.

The remainder of this article is organized as follows: Section 2 briefly explains and exemplifies full factorial designs and orthogonal arrays and explains the basic principles of experimental design. Sections 3 presents the mathematical background and terminology for general orthogonal arrays and quality criteria from them. Section 4 discusses creation of full factorial designs, in particular also with the possibility of blocking them. Section 5 provides insights into usage and inspection of the orthogonal arrays implemented in package **DoE.base**. Section 6 discusses design creation and analysis tools, using the example of an experimental design in biotechnology (Vasilev, Schmitz, Grömping, Fischer, and Schillberg 2014). Section 7 describes in more detail the half-normal plotting functionality provided by package **DoE.base**. Finally, a brief overview of further developments is provided.

2. Basics

2.1. Full factorial designs and designs based on orthogonal arrays

A factorial design is an experimental plan in which k "factors" are systematically varied. The jth factor has l_j "levels", $j = 1...k$. If all factors have the same number of levels, i.e., $l_1 = ... = l_k$, the design is called a "fixed level" or "symmetric" design, otherwise it is called "mixed level" or "asymmetric". A "full factorial" design contains (a multiple of) *all* factor level combinations, i.e., a multiple of $l_1 * ... * l_k$ experimental runs. In a full factorial design, all coefficients for an adequately coded linear model with all main effects, 2-factor interactions, ..., up to k-factor interactions are estimable. The number of estimable effects remains the same, regardless of the choice of adequate coding. Section 7 will discuss how the coding affects correlation between coefficient estimates.

Full factorial designs are often not feasible in the real world, if the number of factors or the numbers of factor levels are not very small. For example, a full factorial experiment with one 2-level factor and six 3-level factors requires 1458 runs. There are several possibilities for designs with fewer runs: D-optimal designs require the specification of a model to be estimated; they can be created with R packages **AlgDesign** or **DoE.wrapper** (Wheeler 2014; Grömping 2013b), but are not the topic of this article. Here, we consider experimental designs based on orthogonal arrays: these do not require specification of a model but assume that (i) all effects of the same degree (main effect=degree 1, 2-factor interaction=degree 2, etc.) are equally important and (ii) that effects of lower degree are more important than those of higher degree. Orthogonal array designs are often used with the intention of estimating main effects only; they are particularly common for qualitative factors, although they can also be used for quantitative factors. For a design based on an orthogonal array, each factor has each level the same number of times, and each pair of factors has each pair of levels the same number of times. Genizi Taguchi provided various orthogonal arrays for engineering experimentation; one of the most-well-known ones is an 18-run array for up to one 2-level factor and up to seven 3-level factors. This array can for example be found in NIST/SEMATECH (2012), and is also contained in package **DoE.base**:

```
R>L18

   A B C D E F G H
1  1 1 1 1 1 1 1 1
2  1 1 2 2 2 2 2 2
3  1 1 3 3 3 3 3 3
4  1 2 1 1 2 2 3 3
5  1 2 2 2 3 3 1 1
6  1 2 3 3 1 1 2 2
7  1 3 1 2 1 3 2 3
8  1 3 2 3 2 1 3 1
9  1 3 3 1 3 2 1 2
10 2 1 1 3 3 2 2 1
11 2 1 2 1 1 3 3 2
12 2 1 3 2 2 1 1 3
13 2 2 1 2 3 1 3 2
14 2 2 2 3 1 2 1 3
15 2 2 3 1 2 3 2 1
16 2 3 1 3 2 3 1 2
17 2 3 2 1 3 1 2 3
18 2 3 3 2 1 2 3 1
attr(,"origin")
[1] "Taguchi"
attr(,"class")
[1] "oa"     "matrix"
```

This small array can already accommodate the above-mentioned experiment with one 2-level factor and six 3-level factors. Of course, as it is very much smaller than the 1458 runs for a full factorial, there is a substantial amount of confounding built into the array. If a small array like the L18 is to be used, two things are very important: picking the best possible columns for the design, and understanding the limitations of the resulting design. Package

DoE.base can help with both. However, except perhaps for very preliminary investigations, it will usually be preferable to use less severely confounded designs. Package **DoE.base** can also help with optimizing selection of an array and column selection within the array. This will be demonstrated in Sections 5 and 6.

Orthogonal arrays may be regular or non-regular: in the regular case, it is possible to describe the array by a few defining relations, similar to the well-known way of doing so for regular fractional factorial 2-level designs: Starting from a full factorial design in some "generating" or "base" factors, additional factors are accommodated by assigning them to interactions between the base factors, which are consequently completely confounded with the new factors' main effects. This is a little more complicated for factors with more than two levels, but the general principle remains the same. One complication arises from base level factors with non-prime numbers of levels; these can be decomposed into full factorials of factors with prime numbers of levels, so-called "pseudo factors". The aforementioned catalog of arrays contains quite a few regular arrays. Regular orthogonal designs can also be created using function `regular.design` of package **planor**.

Non-regular orthogonal arrays cannot be described by defining relations. Some of the cataloged arrays in package **DoE.base** are non-regular. Section 5.1 provides detail on the catalog and its usage. Note that the catalog is by no means complete; in particular, it is much more difficult to completely enumerate all orthogonal arrays than it is to enumerate all *regular* orthogonal arrays. Partially complete catalogs of orthogonal arrays are available, e.g., from the website by Eendebak and Schoen (2010) based on the algorithm described in Schoen, Eendebak, and Nguyen (2010). In many cases, the web site provides the best arrays only, or does not provide an array at all (in case of very large numbers of arrays). Where a number of arrays is shown, the complete set of arrays can in principle be obtained from Eric Schoen; however, with large numbers of arrays the complete catalogs are so large that it is not easily feasible to work with them.

2.2. Principles of experimental design

The most important principle of experimentation is replication: when comparing two different setups, one will usually not rely on a single instance of each setup, but will replicate each setup a specified number of times. This serves the purpose of making sure that differences are only interpreted if they are sufficiently larger than can be expected from experimental variation. Replication is quite different from repeating measurements only: for a proper replication, all experimental settings have to be redone for each replicate. Sometimes, with very variable measurement devices, it may make sense not to include replications but to repeat the measurement process only. This is called "repeated measurements" and has to be treated quite different from proper replication. Several design generation functions of packages **DoE.base** and **FrF2** offer the option `replications` for specifying the number of replications and `repeat.only` for indicating whether these are proper replications (default) or repeated measurements only.

One of the very useful aspects of factorial experiments is *implicit* replication: when experimenting with many factors, one can often expect higher order interactions to be irrelevant. If this is the case, the degrees of freedom that would have to be dedicated to higher order interactions can instead be used for estimating error variation (or for accommodating further experimental factors). Therefore, in factorial experimentation, one will encounter experiments without replicated runs.

A further important principle is blocking, which can be used to control for known influential factors that are not of interest in themselves, like batch-to-batch variation. For an orthogonal array design, one can simply include the block factor as an additional factor and thus has to find an array of the desired structure. A full factorial design can be blocked without increasing the number of runs, by allocating the degrees of freedom for the block factor to portions from interaction effects. This functionality is implemented in function `fac.design` (see Section 4).

Randomization means that the experimental runs are conducted in random order; it is a safeguard against bias from unknown influences. If the run order is completely randomized, all experimental runs can be treated as independent observations, and there is little risk of systematic bias from experimental order or unknown factors related to experimental order or time. In real life, there are sometimes so-called randomization restrictions; for example, experimental runs may be randomized within each block only. Function `fac.design` allows randomization within blocks, while designs created with function `oa.design` have to be rerandomized with function `rerandomize.design` for using one of the factors as a block factor.

Whenever proper replication is used, package **DoE.base** separately randomizes each replication as though it were a block; however, it does not include a block factor for the replications. Users who want to include a block factor for replications in the analysis can obtain such a factor using the function `getblock`. Users who want to change the randomization, i.e., randomize all replications together instead of in separate blocks, can use the function `rerandomize.design`. Using the "[" method for the class `design`, users can also reorder a design according to a randomization scheme that has been worked out outside of R. Of course, whenever the randomization involves non-trivial restrictions like randomizing in meaningful blocks, the analysis has to be conducted accordingly.

3. General orthogonal arrays

This section provides the mathematical background for general orthogonal arrays, as far as it can be helpful for using the orthogonal arrays available in R package **DoE.base**.

3.1. Terminology for orthogonal arrays

An array in the sense of this article is a rectangular table of numbers with n rows and k columns, like the L18 shown on p. 5. The rows correspond to experimental runs, the columns to experimental factors. In the cataloged arrays in **DoE.base**, the levels of an l-level factor are denoted by the numbers $1...l$. An array becomes an experimental design by allocating numbers to factor levels. The array is orthogonal, i.e., an OA, if for each pair of columns each combination of levels occurs equally often. If this is the case, main effects of all factors can be estimated separately from each other (provided, no higher order effects are in the model).

An OA is said to be of strength s, if each combination of levels occurs equally often for each subset of s columns. Thus, each OA is at least of strength 2. Strength of an OA is directly related to resolution of an array: resolution, denoted by roman numerals, is always one higher than the strength, i.e., strength 2 arrays are of resolution III and so forth. For an array of resolution III, main effects are not aliased with main effects, but can be aliased with 2-factor interactions (three factors involved); for an array of resolution IV, main effects are not aliased with 2-factor interactions, but can be aliased with 3-factor interactions, while 2-factor interactions can be aliased with other 2-factor interactions (four factors involved). This notion is well-known for regular fractional factorial 2-level designs, and is completely analogous for non-regular designs and for designs with factors at more than 2 levels or in mixed level situations. Note that a full factorial in k factors has strength k.

3.2. Generalized word length pattern and refinements

Xu and Wu (2001) introduced the generalized word length pattern (GWLP) for general orthogonal arrays. It is an extension of the well-known word length pattern (WLP) for regular fractional factorial 2-level designs: in the latter, one starts out with a set of base factors and allocates additional factors to interactions among these (the generating contrasts). Coding all main effects model matrix columns with "-1" (one level) and "+1" (the other level), this way of design generation causes products of model matrix columns to be either half "-1" and half "+1", or constant columns. Factors, whose product of model matrix columns yields a constant column, form a "word" together. The word length pattern is a frequency table of word lengths. For regular fractional factorial 2-level designs, each group of c factors either does or does not form a word, i.e., contributes one or zero to the count for words of length c. This results in a word length pattern with only integer entries.

In general, partial aliasing is possible. Even if there are only 2-level factors, e.g., in a Plackett-Burman design (Plackett and Burman 1946), a set of factors can contribute a fraction of a word to the GWLP count for the respective word length. Consequently, GWLP entries need not be integers. For example, the GWLP of the L18 is

```
R>GWLP(L18)
```

```
   0    1    2    3    4    5    6    7    8
 1.0  0.0  0.0 28.0 52.5 52.5 70.0 33.0  6.0
```

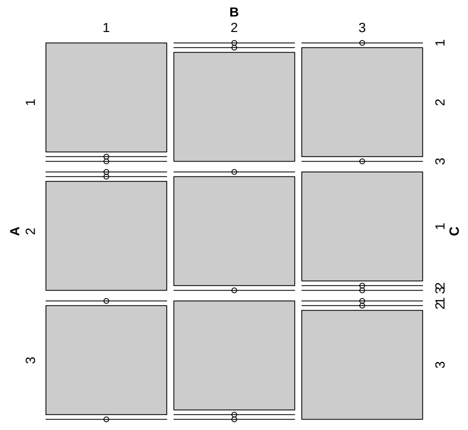
The GWLP is denoted as A_0, A_1, A_2, A_3, ..., with A_c the number of generalized words of length c. The entry "1" for A_0 is generic and does not indicate confounding. The GWLP for orthogonal arrays and designs based on them is usually presented starting with A_3, since orthogonality implies absence of words of lengths one or two. The GWLP coincides with the WLP for regular fractional factorial 2-level designs; for details, consult Xu and Wu (2001) themselves or Grömping (2011a) for a more accessible account. The concepts of strength and resolution directly relate to the (G)WLP: the shortest word length with a non-zero count is the resolution of the design, the longest word length with a zero count is the strength. Hence, the L18 has resolution III and strength 2. Note that it is not adequate to use the term "generalized resolution" here, because that term is already in use for a different concept that is also implemented in package **DoE.base** (see below).

The number of generalized words of length c is the sum over contributions from all sets of c factors. For a set of R factors with $l_1 ... l_R$ levels in a resolution R design (equivalent to $s+1$ factors in a strength s design), Grömping and Xu (2014) have shown an upper bound for the contribution to the count A_R to be $min((l_1 - 1), ..., (l_R - 1))$, i.e., the upper bound for the number of generalized words in a set of R factors depends on the pattern of levels in the set and is given by the main effect degrees of freedom for the factor with the fewest levels (the analogous result for symmetric designs was shown earlier by Xu, Cheng, and Wu 2004). Furthermore, Grömping and Xu (2014) provided a statistical rationale for the contributions of sets of R factors to A_R in resolution R designs, i.e., the building blocks for the number of shortest words: each R-set contribution can be seen as the sum of R^2-values from linear models explaining orthogonal contrast columns for any one of the R factors by a full model in the other $R-1$ factors (provided the factor to be explained has orthogonally coded model matrix columns; otherwise, R^2 values have to be replaced by squared canonical correlations). Thus, the number of shortest words measures the extent of worst-case confounding in a plausible way. It is therefore particularly instructive to study R factor sets in resolution R designs. Based on these, Figure 1 illustrates the meaning of words in an informal sense, using mosaic plots as proposed in Grömping (2014a). The most severe imbalance is shown in plot (a) of the figure: the factor level combination of any pair of factors completely determines the level of the third factor. This is a resolution III regular array, for which each main effect is confounded by the two-factor interaction of the other two factors. This is reflected in the number of words, which coincides with the aforementioned upper bound. The plot shows a triple from the array L36.2.16.3.4 from package **DoE.base**; an identical plot can be produced from columns 2, 4 and 5 of the well-known Taguchi L18 shown on p. 5.

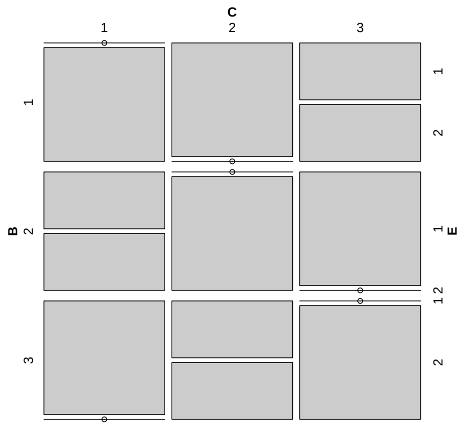
The factor set of Figure 1 (a) attains its upper bound for the number of generalized words. For the other factor sets shown in Figure 1, the upper bound is one because of one degree of freedom only for the 2-level factor in the set. However, with a 2-level and two 3-level factors in an *orthogonal* array, this upper bound cannot be attained. The upper bound can only be attained if the smallest number of levels is a divisor of all the other numbers of levels, which is for example the case in symmetric designs. In summary, the figure illustrates that more words are related to less balance.

The GWLP can be used for selecting a best design by comparing designs with respect to their so-called aberration: a design is better than another one, if it has higher resolution; in case of equal resolution, a design has smaller aberration, if its number of shortest words is smaller, in case of ties, successively considering longer words until a difference is encountered. If this principle is applied to a complete set of possible designs, the best design is said to have "generalized minimum aberration" (GMA).

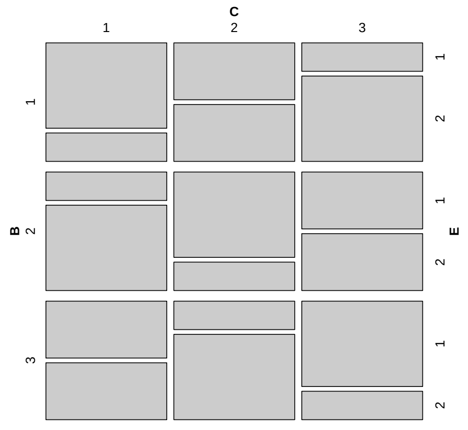
Over and above the GWLP, package DoE.base allows to look at individual R-factor set confounding through mosaic plots (Grömping 2014a), like the ones shown in Figure 1, and provides an overview of the confounding in R-factor projections through projection frequency tables (functions `P3.3` or `P4.4`), average R^2 frequency tables (ARFT) and squared canonical correlation frequency tables (SCFT); the latter are based on the results by Grömping and Xu (2014) and detailed in Grömping (2013a). Sometimes, several designs that have GMA can be distinguished further by the more detailed criteria. Grömping and Xu (2014) also introduced a generalization of the generalized resolution GR as proposed by Deng and Tang (1999): GR refines the resolution R by indicating the distance from complete confounding.



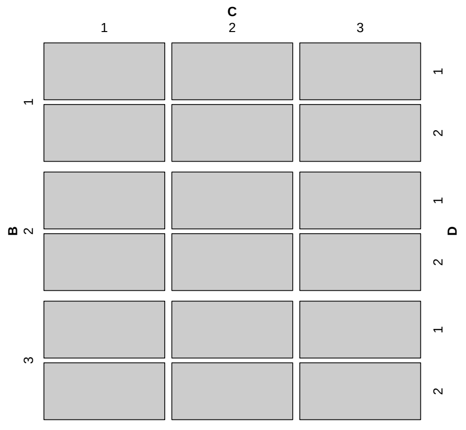
(a) Complete aliasing: 2 words of length 3
36 runs, 3,3,3 levels.



(b) 2/3 words of length 3
36 runs, 2,3,3 levels.



(c) 1/6 words of length 3
36 runs, 2,3,3 levels.



(d) Perfect balance: 0 words of length 3
36 runs, 2,3,3 levels.

Figure 1: Mosaic plots of different degrees of confounding for triples of factors in 36 run designs.

We have $R \leq GR < R+1$; the larger GR, the less severe the worst case confounding in the design; if $GR = R$, there is at least one instance of complete confounding in an R-factor set. Furthermore, GR_{ind} is a stricter version of GR which already becomes equal to R if there is a triple of factors for which there is a coding such that at least one degree of freedom of at least one factor is completely confounded. Besides the overall GR and GR_{ind}, individual factor versions $GR_{tot,i}$ and $GR_{ind,i}$ capture the corresponding worst cases for R factor sets involving the ith factor. The GWLP can be obtained with function `GWLP` (or with the older function `lengths`, which is usually slower but performs better for designs with many runs); GR can be obtained with the (old) function `GR` or – together with GR_{ind}, the individual $GR_{tot,i}$ and $GR_{ind,i}$, ARFT and SCFT – with function `GRind`.

4. Full factorial designs with function `fac.design`

Function `fac.design` creates full factorial designs. There is also a simple way in base R for creating all combinations of factor levels: function `expand.grid` with subsequent randomization of the run order will do the job. The benefits of using function `fac.design` lie in the inclusion into the general framework, and in the possibility of automatically blocking designs. The blocking method makes use of so-called pseudo-factors: whenever the number of levels of a factor is not prime, it can of course be factored into primes, e.g., 6 into 2 and 3; thus, a six-level factor can be obtained by the six different factor level combinations of a full factorial in a two- and a three-level factor; such component factors are called pseudo factors (e.g., F_1 and F_2 below).

Function `fac.design` uses a method by Collings (1984, 1989) for creating the block factor; in case of automatic blocking, the blocking pattern for several 2- or 3-level factors is taken from optimal blocked catalogs (internal objects `block.catlg` and `block.catlg3`); if a factor contains several pseudo factors with the same prime, its use in block generators ensures that different pseudo factors are used for different block generators involving the factor (where possible). However, the procedure does not ensure overall optimality of the blocking strategy. Neither does function `regular.design` from package **planor**; however, that function may be worth a try if the result from function `fac.design` is not satisfactory, and it can be used for situations outside `fac.design`'s scope for automatic blocking.

The following two code examples exemplify situations for which automated blocking works without or with confounding of two-factor interactions of experimental factors (with a warning in the latter case).

For a full factorial design with six factors with 2, 3, 3, 2, 2 and 6 levels (hence 432 runs), running in six blocks is possible without confounding blocks with two-factor interactions in experimental factors:

```
R>full.factorial.blocked6 <- fac.design(nlevels = c(2, 3, 3, 2, 2, 6),
+   blocks = 6)
R>summary(full.factorial.blocked6)

Call:
fac.design(nlevels = c(2, 3, 3, 2, 2, 6), blocks = 6)
```

```
Experimental design of type  full factorial.blocked
432  runs
blocked design with  6  blocks of size  72

Confounding of  2 -level pseudo-factors with blocks
(each row gives one independent confounded effect):
A B C D E F F
1 0 0 1 1 1 0


Confounding of  3 -level pseudo-factors with blocks
(each row gives one independent confounded effect):
A B C D E F F
0 1 1 0 0 0 1


Factor settings (scale ends):
  A B C D E F
1 1 1 1 1 1 1
2 2 2 2 2 2 2
3   3 3     3
4           4
5           5
6           6
```

The summary indicates that the design confounds the block factor with the interactions $ADEF_1$ and BCF_2, where F_1 and F_2 denote two different pseudo factors that make up the six-level factor F. This means, in particular, that there is no confounding of the block factor with two-factor interactions of experimental factors.

Blocking this full factorial in four or nine blocks is also possible, but confounds the block factor with a two-factor interaction among experimental factors, which is signaled by a warning message and is also visible from the summary:

```
R>full.factorial.blocked4 <- fac.design(nlevels = c(2, 3, 3, 2, 2, 6),
+   blocks = 4)
R>summary(full.factorial.blocked4)

Call:
fac.design(nlevels = c(2, 3, 3, 2, 2, 6), blocks = 4)

Experimental design of type  full factorial.blocked
432  runs
blocked design with  4  blocks of size  108

Confounding of  2 -level pseudo-factors with blocks
(each row gives one independent confounded effect):
     A B C D E F F
[1,] 1 0 0 0 1 1 0
[2,] 0 0 0 1 1 1 0
[3,] 1 0 0 1 0 0 0

Factor settings (scale ends):
  A B C D E F
1 1 1 1 1 1 1
2 2 2 2 2 2 2
3   3 3     3
4           4
5           5
6           6
```

The function allows automatic blocking for the most frequent situations, where most prime factors are two and three, and only single prime (pseudo) factors are larger than three; there is also a limit on the number of 2- and 3-level factors (see the package manual).

We now consider an example, for which block generators have to be manually specified: for blocking a full factorial in one 2-level factor, three 5-level factors and one 10-level factor into 25 blocks, the prime 5 is needed twice for creating the block factor; thus, two block generators for the prime 5 need to be specified. These can be given as a matrix with two rows (one for each block generator) and a column for each prime factor (in the order of factors, and within each factor, in increasing order, i.e., 2, 5, 5, 5, 2, 5 for the present design). The following code yields the desired blocked full factorial for the above requirement.

```
R>BG <- rbind(c(0, 1, 1, 2, 0, 0),  c(0, 0, 1, 1, 0, 1))
R>full.factorial.blocked25 <- fac.design(nlevels = c(2, 5, 5, 5, 10),
+   blocks = 25, block.gen = BG)
R>summary(full.factorial.blocked25)

Call:
fac.design(nlevels = c(2, 5, 5, 5, 10), blocks = 25, block.gen = BG)

Experimental design of type  full factorial.blocked
2500  runs
blocked design with  25  blocks of size  100

Confounding of  5 -level pseudo-factors with blocks
(each row gives one independent confounded effect):
     A B C D E1 E2
[1,] 0 1 1 2  0  0
[2,] 0 0 1 1  0  1
[3,] 0 1 2 3  0  1
[4,] 0 1 3 4  0  2
[5,] 0 1 4 0  0  3
[6,] 0 1 0 1  0  4
```

```
Factor settings (scale ends):
   A B C D  E
1  1 1 1 1  1
2  2 2 2 2  2
3    3 3 3  3
4    4 4 4  4
5    5 5 5  5
6           6
7           7
8           8
9           9
10         10
```

The above design is reasonable and does not confound two-factor interactions of experimental factors with the block factor. Function `fac.design` would throw an error, if the chosen block generator confounded a block effect with a main effect contrast of experimental factors or did not provide an appropriate number of blocks. Apart from these gross issues, responsibility for an appropriate choice of `block.gen` is completely with the user.

If the user is not able to come up with a satisfactory block structure, function `regular.design` from package **planor** can be used to create a design with the required properties (see code below). If that function takes a very long time for a reasonably-sized problem, there is in many cases no solution for the requested situation; it can, however, also mean that an existing solution is difficult to find for regular designs with factors that have non-prime numbers of levels.

```
R>require("planor")
R>planor.blocked25 <- regular.design(
+   factors = c("Block", "A", "B", "C", "D", "E"),
+   nlevels = c(25, 2, 5, 5, 5, 10), block = ~Block, nunits = 2500,
+   model = ~ (A + B + C + D + E) ^ 2 + Block,
+   estimate = ~ (A + B + C + D + E) ^ 2)

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

For the design `planor.blocked25`, the `estimate` option guarantees that blocks are not confounded with 2-factor interactions. The block generator used by function `regular.design` is different from the one chosen in function `fac.design` above, but of the same quality with respect to the GMA criterion. This can be verified by applying function `GWLP` or function `lengths` to both designs (as mentioned before, `lengths` is faster for designs with many runs):

```
R>round(lengths(full.factorial.blocked25, with.blocks = TRUE), 2)

 2  3  4  5
 0  0 16  8
```

```
R>round(lengths(getDesign(planor.blocked25)), 2)

 2  3  4  5
 0  0 16  8
```

Neither function `fac.design` nor function `regular.design` from package **planor** guarantees optimality of the confounding structure, and neither is generally superior to the other.

5. Orthogonal arrays with package DoE.base

If more than two levels are required for some factors, but a full factorial cannot be afforded, a regular or non-regular orthogonal array can be used. For the creation of *regular* designs, also with mixed levels, function `regular.design` from package **planor** is very useful. As seen in the code of the previous section, the function can specify a model and separately a (sub) model to be estimable; in this way, it was, e.g., possible to treat the block factor different from the other factors. Note, however, that it is not possible to generate non-regular designs, and that no effort is made at a better design in terms of overall model robustness, whenever the requested estimability requirements are satisfied. Furthermore, creation of some designs takes a very long time, even for relatively small designs (e.g., 32 runs).

Package **DoE.base** pursues a different route: It contains the previously-mentioned catalog of orthogonal arrays; most of its arrays have been taken from Kuhfeld (2010). Similarly to most experimental design software, the default approach of function `oa.design` is to use the smallest array possible and to take columns with the requested numbers of levels from left to right until the design has been filled. If the user does not influence the chosen columns with the `columns` option, a warning is issued for making the user aware of potential improvements. The following sub sections discuss the catalog and ways to select arrays from it, the optimization of column selection from a selected array, and ways to inspect experimental plans regarding their suitability for the experiment at hand.

5.1. The data frame `oacat` and the function `show.oas`

The arrays available in package **DoE.base** are documented in the data frame `oacat`. Since version 0.27, this data frame contains additional columns with information regarding the array properties. `oacat` contains the following columns:

- `name` gives a structured array name, which indicates the number of runs and the frequency of factors with different numbers of levels; for example, the name `L18.2.1.3.7` indicates 18 runs with one 2-level factor and seven 3-level factors. `nruns` directly gives the number of runs. `n2` to `n72` give the number of factors with 2 to 72 levels. Thus, for `L18.2.1.3.7`, `nruns=18`, `n2=1`, `n3=7`, and all other `nx` entries have the value 0.

- `lineage` contains a string variable which indicates how the array was constructed from so-called parent arrays. An empty string indicates that the array itself is a parent array. Parent arrays are stored in the package and are objects of class `oa`, which are matrices that usually contain an `origin` attribute and sometimes also a `comment` attribute. Arrays with a lineage entry are constructed from the parent arrays.

- Logical columns indicate a full factorial array (`ff`), an array with only squared canonical correlations 0 and 1 (`regular`), and an array with only average R^2 values 0 and 1 (`regular.strict`; these can only be fixed level arrays). Note that the squared canonical correlations and average R^2 values currently refer to R factor projections, where R is the resolution of the entire array; an array that exhibits regular behavior for all such projections might still be non-regular when considering projections onto larger numbers of factors. Thus, the regularity indicator columns might change with future package versions.
- Column `SCones` contains the number of squared canonical correlations that are one.
- Columns `GR` and `GRind` contain the GR and GR_{ind} values, respectively.
- Columns `maxAR` and `maxSC` contain the maximum average R^2 or squared canonical correlation, respectively.
- Column `dfe` provides the number of error degrees of freedom, if the all columns of the array are used.
- Columns `A3` to `A8` provide the numbers of generalized words of lengths 3 to 8. (There are no words of shorter lengths, of course.)

It is possible to use the data frame `oacat` directly for inspecting which arrays of a certain nature are available, for example for finding 32 run arrays which are regular but not strictly regular:

```
R>oacat$name[oacat$nruns==32 & oacat$regular & !oacat$regular.strict]
```

```
 [1] "L32.2.28.4.1"     "L32.2.25.4.2"     "L32.2.24.8.1"
 [4] "L32.2.22.4.3"     "L32.2.21.4.1.8.1" "L32.2.19.4.4"
 [7] "L32.2.18.4.2.8.1" "L32.2.16.4.5"     "L32.2.16.16.1"
[10] "L32.2.15.4.3.8.1" "L32.2.13.4.6"     "L32.2.12.4.4.8.1"
[13] "L32.2.10.4.7"     "L32.2.9.4.5.8.1"  "L32.2.7.4.8"
[16] "L32.2.6.4.6.8.1"  "L32.2.4.4.9"      "L32.2.3.4.7.8.1"
[19] "L32.4.8.8.1"
```

The author prefers non-regular arrays for many situations, at least for the creation of screening designs. Looking at regular arrays in package **DoE.base** may nevertheless be of interest, if function `regular.design` of package **planor** runs for a long time without indicating failure for a run size that is in the scope of package `DoE.base`: if there is a regular array of the desired size in **DoE.base**, this array can be inspected and perhaps used after column optimization (see next section). Also, in principle, function `regular.design` can be expected to eventually succeed, unless estimability requirements make the existing array unsuitable. However, note again that the value `TRUE` in column `regular` or even `regular.strict` of `oacat` refers to projections onto R factors with R the array's resolution and does not guarantee regularity of the entire array. For example, the array `L24.2.12.12.1` has only regular three-factor projections (full factorial for triples of 2-level factors, confounded for triples with the 12-level factor), but non-regular four-factor projections (for quadruples of 2-level factors).

The function `show.oas` allows inspection of the available arrays in a more convenient way. Suppose, for example, that a design for three 2-level factors, two 3-level factors and one 6-level factor is to be created, and between 20 and 54 runs are affordable (a full factorial would have 432). The following statement allows to inspect the candidate arrays, displaying also the quality metrics:

```
R>show.oas(nruns = c(20, 54), nlevels = c(2, 3, 3, 2, 2, 6),
+   showmetrics = TRUE)
```

```
5  arrays found
                 name nruns lineage   GR GRind regular SCones    A3    A4   A5
78 L36.2.13.3.2.6.1    36         3.00  3.00   FALSE      6  45.3 158.4  426
81 L36.2.10.3.8.6.1    36         3.18  3.18   FALSE      0 130.3 737.2 3063
83  L36.2.9.3.4.6.2    36         3.00  3.00   FALSE     17  82.3 338.2 1025
87  L36.2.3.3.9.6.1    36         3.18  3.00   FALSE     12  73.8 300.2  912
88  L36.2.3.3.2.6.3    36         3.00  3.00   FALSE     37  35.6  73.1  120
      A6      A7      A8
78  1010  1753.2  2306.8
81 11096 31380.8 68828.1
83  2828  5507.5  7780.0
87  2404  4354.2  5793.4
88   125    63.5    13.9
```

Only arrays in 36 runs have been found; these are quite different in the available patterns of numbers of levels, and therefore the quality metrics (especially the A_k) are not directly comparable. Nevertheless, the values for GR and GR_{ind} suggest that one might try the array L36.2.10.3.8.6.1, which is the only one without any completely confounded degree of freedom.

5.2. Optimization methods for function `oa.design`

Function `oa.design` allows to select an array, and to specify columns from that array either manually or by an optimization approach. As was mentioned earlier, if no array is specified, the function picks the first (and thus smallest) array in the catalog that is able to accommodate the requested factors. If no column selection approach is specified, the function simply takes the first available columns (from left to right).

The code below compares three designs:

- an unoptimized default design (i.e., the left-most suitable columns of the first array encountered, which is the L36.2.13.3.2.6.1),
- an optimized default design with optimization option `columns="min34"` (i.e., an optimized choice of columns from the array L36.2.13.3.2.6.1, optimization being with respect to the number of generalized words of length 3, and subsequently length 4)),
- and an optimized design obtained by selecting columns from the array L36.2.10.3.8.6.1 selected in Section 5.1 because of its GR_{ind} value.

```
R>oa.default <- oa.design(nlevels = c(2, 3, 3, 2, 2, 6))
R>GWLP(oa.default, digits = 2)

   0    1    2    3    4    5    6
1.00 0.00 0.00 5.78 2.11 2.44 0.67

R>oa.optimized <- oa.design(nlevels = c(2, 3, 3, 2, 2, 6), columns = "min34")
R>GWLP(oa.optimized, digits = 2)

   0    1    2    3    4    5    6
1.00 0.00 0.00 4.11 3.61 2.78 0.50

R>oa.manualoptimized <- oa.design(L36.2.10.3.8.6.1,
+   nlevels = c(2, 3, 3, 2, 2, 6), columns = "min34")
R>GWLP(oa.manualoptimized, digits = 2)

   0    1    2    3    4    5    6
1.00 0.00 0.00 2.44 6.44 1.78 0.33
```

Clearly, optimization improves the design from the default array, and the optimized preselected array is even better.

There are various methods for optimizing column allocation (see the manual for function `oa.design`); `columns = "min34"` is the most important one among these. Depending on the number of columns on offer and the number of columns to be selected, optimization can take a very long time; in the above example, $\binom{10}{3}\binom{8}{2} = 3360$ column choices have to be checked out, which is doable in reasonably short time. For larger designs, the resource implications of the numbers of available columns of the required lengths may also be considered in selecting an array to use.

5.3. Blocking general orthogonal arrays

Suppose a design with the above factors is to be run, and the 6-level factor is a blocking factor. In that case, the design should be randomized such that runs are randomized within blocks. Function `oa.design` does not directly allow to block randomization. However, the function `rerandomize.design` allows a post-hoc randomization within a single design factor declared as the block factor:

```
R>blockedoa1 <- rerandomize.design(oa.manualoptimized, seed = 24652,
+   block = "F")
R>blockedoa1

  run.no run.no.std.rp F A B C D E
1      1        22.5.4 5 2 1 1 1 2
2      2         5.5.2 5 1 2 3 2 2
3      3        15.5.3 5 1 3 3 1 1
4      4         1.5.1 5 1 1 2 1 1
```

```
5      5        35.5.6 5 2 2 1 2 1
6      6        33.5.5 5 2 3 2 2 2
   run.no run.no.std.rp F A B C D E
7       7         32.3.5 3 2 2 1 2 2
8       8         14.3.3 3 1 2 2 1 1
9       9         34.3.6 3 2 1 3 2 1
10     10          4.3.2 3 1 1 2 2 2
11     11          3.3.1 3 1 3 1 1 1
12     12         24.3.4 3 2 3 3 1 2
   run.no run.no.std.rp F A B C D E
13     13         18.4.3 4 1 3 3 2 2
14     14         26.4.5 4 2 2 2 2 1
15     15         30.4.6 4 2 3 1 1 1
16     16         11.4.2 4 1 2 1 1 2
17     17          7.4.1 4 1 1 3 2 1
18     18         19.4.4 4 2 1 2 1 2
   run.no run.no.std.rp F A B C D E
19     19         10.2.2 2 1 1 3 1 2
20     20         21.2.4 2 2 3 1 1 2
21     21         29.2.6 2 2 2 3 1 1
22     22         17.2.3 2 1 2 2 2 2
23     23          9.2.1 2 1 3 2 2 1
24     24         25.2.5 2 2 1 1 2 1
   run.no run.no.std.rp F A B C D E
25     25         20.6.4 6 2 2 3 1 2
26     26         16.6.3 6 1 1 1 2 2
27     27          8.6.1 6 1 2 1 2 1
28     28         12.6.2 6 1 3 2 1 2
29     29         28.6.6 6 2 1 2 1 1
30     30         27.6.5 6 2 3 3 2 1
   run.no run.no.std.rp F A B C D E
31     31         31.1.5 1 2 1 3 2 2
32     32         36.1.6 1 2 3 2 2 1
33     33         13.1.3 1 1 1 1 1 1
34     34          2.1.1 1 1 2 3 1 1
35     35         23.1.4 1 2 2 2 1 2
36     36          6.1.2 1 1 3 1 2 2
class=design, type= oa.blocked
NOTE: columns run.no and run.no.std.rp are annotation, not part of the
  data frame
```

Function GWLP per default ignores confounding with the block factor, but can be requested to include it:

```
R>GWLP(blockedoa1, digits = 2)

   0    1    2    3    4    5
1.00 0.00 0.00 0.28 0.33 0.39
```

```
R>GWLP(blockedoa1, with.block = TRUE, digits = 2)

   0    1    2    3    4    5
1.00 0.00 0.00 2.44 6.44 1.78
```

For designs that can also be created with function `FrF2` (package **FrF2**) and/or `regular.design` (package **planor**), using one of the latter two may be a better choice, since they allow a more direct control over design quality via minimum aberration or estimable effects: The code below (not run) shows creation and inspection of a 16 run design with eight 2-level factors in eight blocks of size 2 with all three methods; in all three cases, the design is resolution IV in terms of the experimental factors, which is systematically requested in both `FrF2` (by the minimum aberration approach of the function) and `regular.design` (by the `model` and `estimate` options). For function `oa.design`, the design quality is not as finely tunable, apart from the overall optimization that may precede usage of an array for which not all columns are used. For the designs below, however, the quality is the same for all three designs: The GWLPs, starting with A_3, are (28,14,56,0,28,1) including the block factor and (0,14,0,0,0,1) for the experimental factors alone (for class `design` objects, this is the default, obtained from calling `GWLP` without the `with.block = TRUE` option).

```
R>planFrF2 <- FrF2(16, 8, blocks = 8, alias.block.2fis = TRUE)
R>planDoEbase <- oa.design(L16.2.8.8.1,
+   nlevels = c(2, 2, 2, 2, 2, 2, 2, 2, 8),
+   factor.names = c(Letters[1:8], "Block"))
R>planDoEbase <- rerandomize.design(planDoEbase, seed = 31525,
+   block = "Block")
R>planplanor <- regular.design(factors = c(Letters[1:8], "Block"),
+   nlevels = c(rep(2, 8), 8),
+   model = ~ (A + B + C + D + E + F + G + H) ^ 2 + Block,
+   estimate = ~ A + B + C + D + E + F + G + H,
+   nunits = 16, randomize = ~ Block / UNITS)
R>GWLP(planFrF2, with.block = TRUE)
R>GWLP(planFrF2)
R>GWLP(planDoEbase, with.block = TRUE)
R>GWLP(planDoEbase)
R>GWLP(planplanor@design)
R>GWLP(planplanor@design[, 1:8])
```

As an aside, it is worth mentioning that any experimental plan should be carefully checked before using it for experimentation, since experimentation usually involves a lot of effort: adverse consequences from mistakes in design creation may be severe, but can often be prevented without much trouble, if attended to at the design creation stage. A simple check of the GWLP can serve as a first indication whether the design behaves as expected. For example, an earlier version of package package **planor** (version 0.2.0, when the package website still warned that the package was under test and should be used with caution) had a bug that could under certain circumstances create avoidably bad designs. The code below (results not shown) gives the above blocking example with a less wise design specification that yielded a non-orthogonal design with one word of length 2 (GWLP starting with A_2: 1,24,25,32,31,8,6):

```
R>planoldplanor.bug <- regular.design(factors = c(Letters[1:8], "Block"),
+   nlevels = c(rep(2, 8), 8),
+   model = ~ A + B + C + D + E + F + G + H + Block,
+   estimate = ~ A + B + C+ D + E + F + G + H,
+   nunits = 16, randomize = ~ Block / UNITS)
R>GWLP(planoldplanor.bug@design)
R>plot(planor2design(planoldplanor.bug), select = "all2")
```

The plot command in the code above revealed that factor C was aliased with the Block factor for the latter design, which explains the word of length 2. Such a design should of course not be used. Let me emphasize that this example is not meant to imply that package **planor** is less trustworthy than other design creation packages but rather that a design should always be checked before using it for experimentation!

5.4. Inspection methods for factorial designs

This is a good place to exemplify further possibilities for design inspection. The function names for obtaining quality criteria were already mentioned in Section 3.2.

The function `GRind` calculates the metrics introduced by Grömping and Xu (2014) with the additional detail proposed in Grömping (2013a). For example, the code below shows that the optimized design based on the manually selected array has a clearly better overall behavior regarding factor-specific worst case confounding (GR_{tot}, i and $GR_{ind,i}$, see Section 3.2) than the optimized design based on the automatically selected first array:

```
R>GRind1 <- GRind(oa.optimized)
R>GRind2 <- GRind(oa.manualoptimized)
R>print(cbind(rep(c("oa.opimized", "oa.optimized2"), each = 2),
+    rbind(GRind1$GR.i, GRind2$GR.i)), quote = FALSE)

                      A     B   C     D     E     F
GRtot.i oa.opimized   3.667 3   3     3.423 3.423 3.368
GRind.i oa.opimized   3.667 3   3     3.423 3.423 3
GRtot.i oa.optimized2 3.184 3.5 3.423 3.423 3.667 3.635
GRind.i oa.optimized2 3.184 3.5 3.423 3.423 3.667 3.423
```

The `plot` method for class `design` can be used to visualize the worst case confounding for a design. The optimized design based on the array that was automatically picked severely confounds the interaction of the two 3-level factors with the 6-level factor: of the 54 possible level combinations in this triple of factors, the design contains 18 only (instead of the possible 36). The optimized design after manually picking a better starting array shows a much less severe worst case confounding. A mosaic plot of that projection is created by the first line of the following code and is shown in Figure 2 (a). It shows that the interaction of the two 3-level factors restricts the 6-level factor to a third of its possibilities only, which is the most severe form of aliasing possible for a triple of factors with this level combination.

```
R>plot(oa.optimized, select = "worst", selprop = 0.05)
R>plot(oa.manualoptimized, select = c(2, 3, 6))
R>plot(oa.manualoptimized, select = "worst", selprop = 0.05)
```

In comparison, a mosaic plot of the same projection for the optimized manually selected design shows a much better picture: now, the design has 36 distinct level combinations, the maximum possible (see Figure 2 (b)). The worst case confounding for that design is shown in the Figure 2 (c). It is distinctly less severe than that of Figure 2 (a); however, apart from the one worst case shown in Figure 2 (a), the optimized automatically-selected design is also quite reasonable.

6. An example from plant biotechnology

Vasilev *et al.* (2014) investigated cultivation factors for geraniol production by plant cells. There were four quantitative 2-level factors, two 3-level factors (one quantitative and one qualitative) and one qualitative 4-level factor. The data, including response values, have been published with the paper and are also included in package **DoE.base** as `VSGFS`.

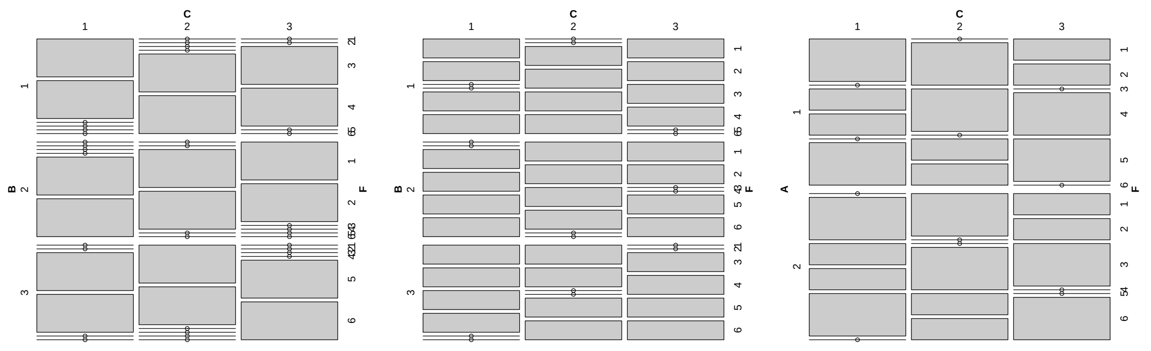
6.1. Creating and inspecting the design

For this experiment, a full factorial design would have had 576 runs. It would certainly have been necessary to conduct it in blocks, say in eight blocks of size 72 each. Such a design could have been created by function `fac.design` or by function `regular.design` of package **planor**, as shown in Section 4.

A full factorial appeared neither feasible nor appropriate for the screening situation of the experiment. Instead, the design was conducted using a 72 run orthogonal array, which was generated with function `oa.design`, using automatic optimization (option `columns = "min34"`) of the manually preselected array `L72.2.43.3.8.4.1.6.1`. The optimization takes quite a long time and results in the selection of columns 4,22,37,41 for the 2-level factors, 46 and 48 for the 3-level factors, and 52 for the 4-level factor. The design is available as `VSGFS`, and the code for reproducing it can be found in the manual.

When constructing the experimental plan for `VSGFS`, the array was preselected by intuition and trial and error, using the version of function `show.oas` available at the time, which only allowed to show all 72 run designs that can accommodate the requested factors. Using the recently added `showmetrics` option together with the `GRgt3="ind"` option (i.e., requesting that $GR_{ind} > 3$ which means absence of any instance of complete confounding), one can pick arrays that are particularly promising for screening purposes. The following command shows the available arrays without any complete confounding for the example situation:

```
R>show.oas(nlevels = c(2, 2, 2, 3, 2, 3, 4), GRgt3 = "ind",
+   showmetrics = TRUE)
```



(a) Autoselected array, worst case.

(b) Manually selected array, same case.

(c) Manually selected array, worst case.

Figure 2: Mosaic plots of triples of factors.

```
4  arrays found
                    name nruns                        lineage   GR GRind
366      L72.2.53.3.2.4.1    72                                3.18  3.18
380     L72.2.44.3.12.4.1    72                                3.18  3.18
382 L72.2.43.3.8.4.1.6.1    72                                3.18  3.18
431     L72.2.13.3.25.4.1    72 3~24;24~1;:(24~1!2~13;3~1;4~1;) 3.18  3.13
    regular SCones  A3    A4     A5      A6       A7       A8
366   FALSE      0 493  6952  73484  664600  5061660 3.29e+07
380   FALSE      0 823 13567 169200 1807970 16229326 1.24e+08
382   FALSE      0 687 10478 121425 1201691  9944686 6.99e+07
431   FALSE      0 653  9622 107523 1022817  8069007 5.36e+07
```

The array used in actual experimentation is the third array in this list; thus, the intuitive approach was successful in this case. It can be expected that all four arrays listed above are reasonably suited for screening the experimental factors. As the designs are far from saturated, optimal column allocation can substantially improve the worst case confounding: for the eventual design, the output below shows $GR = GR_{ind} = 3.\overline{6}$, implying that the largest squared correlation and the largest average R^2 are 1/9 only; these are attained in the triples 1,2,7 and 3,5,7. For illustrating the worst case degree of confounding, Figure 3 shows the triple 3,5,7 in quite reasonable balance.

```
R>GRind(VSGFS)

$GRs
   GR GRind
3.667 3.667

$GR.i
        Light ShakFreq InocSize FilledVol    CM Sugar   CDs
GRtot.i 3.667    3.667    3.667     3.864 3.667 3.864 3.808
GRind.i 3.667    3.667    3.667     3.808 3.667 3.808 3.667
```

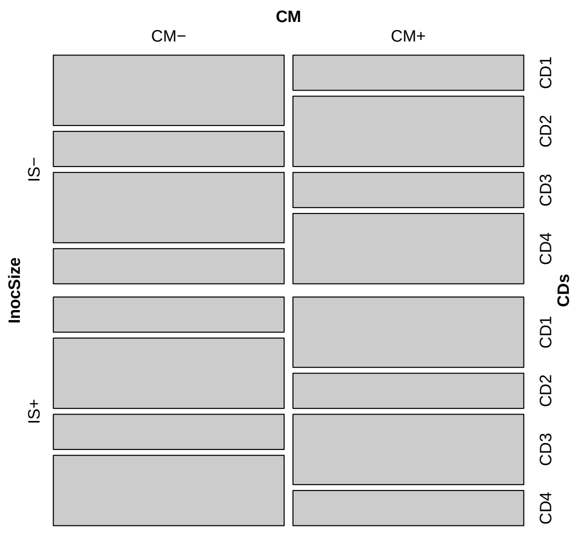


Figure 3: Mosaic plot for the worst case triple in the `VSFGS` example.

```
$ARFT
 aveR2 frequency
 0.000        69
 0.004         4
 0.009         1
 0.012        15
 0.019         6
 0.037         6
 0.111         4

$SCFT
    SC frequency
 0.000       129
 0.012        15
 0.019         3
 0.037        12
 0.111         6
```

The team was happy with the design and used it for collecting the data. Software-wise, data collection happened in Excel, exporting the randomized design with the `export.design` function and re-importing it after data entry using the exported rda file together with a csv file with response values added to the data rows using function `add.response`.

6.2. Analyzing experimental data

The data frame `VSFGS` contains the experiment with response data. Package **DoE.base** offers a few functions for analysis purposes:

- the `plot` method for class `design`, in case of data with responses, creates simple main effects plots, by invoking the plot method from package **graphics**
- the `lm` method for class `design` runs a linear model with a modifiable default degree. For orthogonal arrays created with function `oa.design`, the default degree is one, i.e., a model with main effects only. Linear models can of course also be run by the `lm` function from the core package **stats**, and analysis of variance functionality can also be used (see below).
- the `halfnormal` method for classes `lm` or `design` creates a half-normal effects plot (see Section 7)

Of course, other R functionality can also be used, for example interaction plots or functionality for mixed model analysis. The functionality for handling repeated measurement data or replicated data has been described for package **FrF2** in Grömping (2014c, Sections 5.3 and 5.9 of) and is analogous here.

Main effects plots can be obtained by the simple command `plot(VSGFS)`, which creates these plots for all three response variables. Figure 4 shows the plots with default labeling, arranged with three plots on one page and reduced margin sizes. Of course, one would usually adapt the annotation for final reports or publications. The plot shows that the sugar sucrose is very beneficial for the biomass, not very good for the content, but nevertheless, because of the strong effect on biomass, beneficial for the yield. The content apparently can be increased by choosing the sugar mannitol, level one of factor CD and the +-level of Light. Apart from the +-level of Light, the other settings for high content are not helpful for overall yield.

An assessment of significance can be obtained from a linear model analysis. This can be obtained separately for each response, for example for the content. Here, the analysis confirms the findings from the main effects plot:

```
R>summary(lm(VSGFS, response = "Content"))

Number of observations used: 72
Formula:
Content ~ Light + ShakFreq + InocSize + FilledVol + CM + Sugar +
    CDs

Call:
lm.default(formula = fo, data = model.frame(fo, data = formula))

Residuals:
    Min      1Q  Median      3Q     Max
-2.5472 -1.1746 -0.1052  0.7008  4.6994

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  18.67611    0.55106  33.891  < 2e-16 ***
Light1        0.63264    0.19483   3.247  0.00191 **
ShakFreq1    -0.15792    0.19483  -0.811  0.42083
```

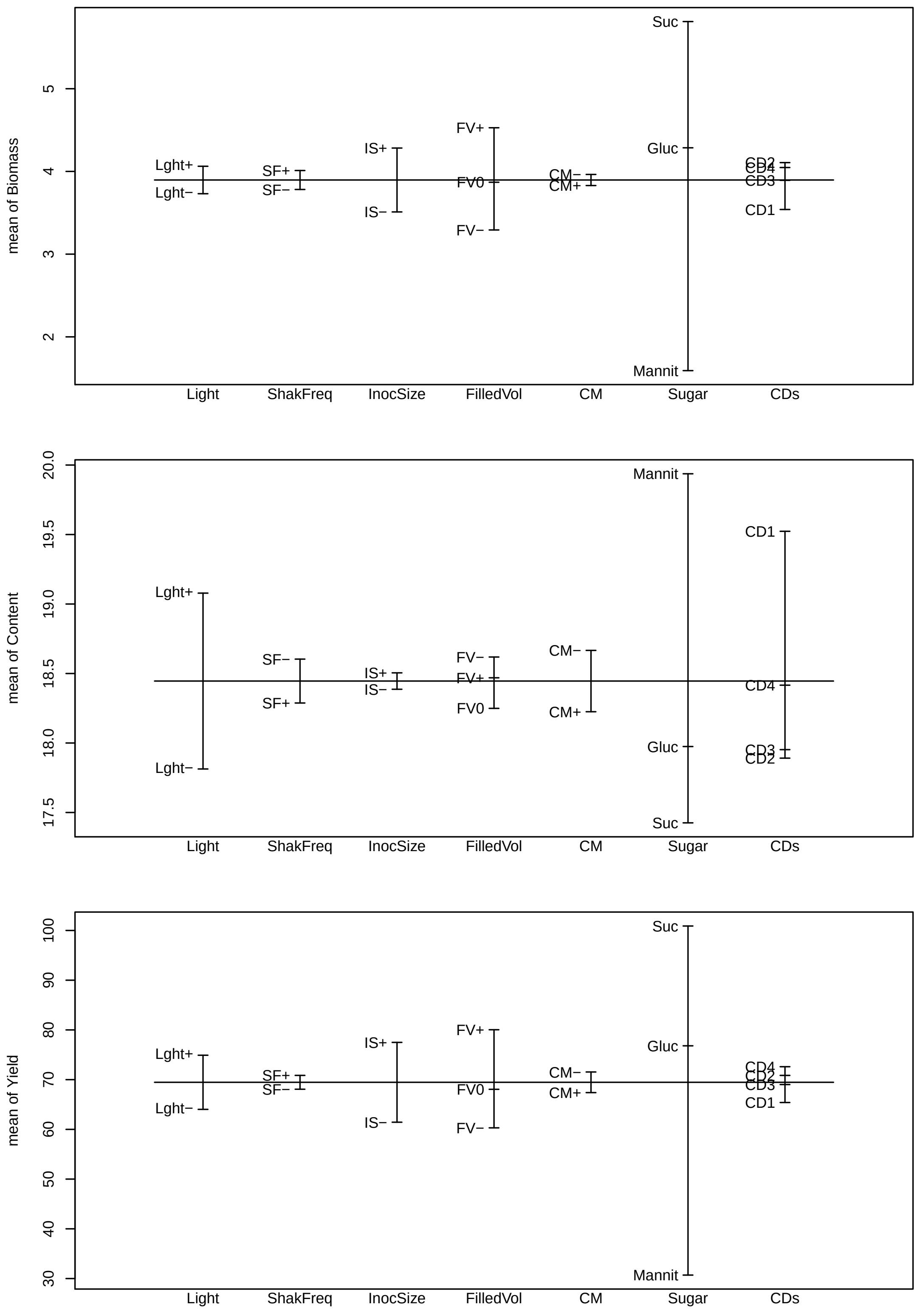


Figure 4: Main effects plots for all three responses.

```
InocSize1      0.05903    0.19483   0.303  0.76296
FilledVolFV0 -0.36958    0.47723  -0.774  0.44172
FilledVolFV+ -0.14958    0.47723  -0.313  0.75503
CM1          -0.22069    0.19483  -1.133  0.26182
SugarGluc     0.54917    0.47723   1.151  0.25441
SugarMannit   2.51167    0.47723   5.263    2e-06 ***
CDsCD2       -1.63222    0.55106  -2.962  0.00438 **
CDsCD3       -1.57111    0.55106  -2.851  0.00597 **
CDsCD4       -1.10722    0.55106  -2.009  0.04901 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.653 on 60 degrees of freedom
Multiple R-squared:  0.4787,	Adjusted R-squared:  0.3831
F-statistic: 5.008 on 11 and 60 DF,  p-value: 1.75e-05
```

The model explains less than 50% of the response variability, which is far from perfect. The strong heredity principle – interactions are only active if both their component factors are active – suggests to look at a model with the three active main effect factors and their interactions, which leaves a somewhat confusing picture (not shown).

For the data at hand, there are enough degrees of freedom to run an Anova analysis with the full degree 2 model. Of course, while main effects are orthogonal to each other, 2-factor interactions can be slightly confounded with main effects and severely confounded with other 2-factor interactions. However, at least, the theory tells us that the estimable effects can be estimated without bias (unless effects of order higher than two bias them). As Anova analyses sums of squares, the analysis is invariant with respect to factor coding. In order to obtain an order-invariant assessment of significance, the function `Anova` from package **car** (Fox and Weisberg 2011) can be used; contrary to function `anova` from package **stats**, `Anova` avoids order-dependence by using type II sums of squares, which condition on all other effects except for ones that contain the effect under investigation. The results point to the main effects that were also identified before, and a liberal look at p values additionally indicates some marginal 2-factor interactions (Sugar with each of `InocSize`, `FilledVol`, `CDs`, `Light`, and `ShakFreq:CM`, which is completely unrelated to the active main effects).

```
R>require("car")
R>Anova(lm(VSGFS, response = "Content", degree = 2))

Anova Table (Type II tests)

Response: Content
                   Sum Sq Df F value    Pr(>F)
Light              16.345  1  8.2606 0.0165515 *
ShakFreq            0.483  1  0.2441 0.6319195
InocSize            1.604  1  0.8105 0.3891387
FilledVol           0.889  2  0.2247 0.8026463
CM                  0.805  1  0.4069 0.5378565
```

```
Sugar               67.692  2 17.1048 0.0005921 ***
CDs                 29.468  3  4.9642 0.0230814 *
Light:ShakFreq       1.672  1  0.8450 0.3795988
Light:InocSize       0.320  1  0.1616 0.6961803
Light:FilledVol      3.296  2  0.8330 0.4628103
Light:CM             2.874  1  1.4524 0.2558948
Light:Sugar         11.387  2  2.8774 0.1030201
Light:CDs            6.073  3  1.0231 0.4231004
ShakFreq:InocSize    1.162  1  0.5871 0.4612301
ShakFreq:FilledVol   0.420  2  0.1061 0.9003686
ShakFreq:CM          6.303  1  3.1856 0.1046054
ShakFreq:Sugar       3.074  2  0.7767 0.4857813
ShakFreq:CDs        11.717  3  1.9738 0.1819413
InocSize:FilledVol   1.626  2  0.4109 0.6737704
InocSize:CM          0.752  1  0.3802 0.5512885
InocSize:Sugar      20.475  2  5.1739 0.0286697 *
InocSize:CDs         7.710  3  1.2989 0.3280819
FilledVol:CM         2.976  2  0.7521 0.4962887
FilledVol:Sugar     24.172  4  3.0539 0.0692876 .
FilledVol:CDs        6.333  6  0.5334 0.7719232
CM:Sugar             3.387  2  0.8559 0.4538120
CM:CDs              10.566  3  1.7799 0.2144014
Sugar:CDs           33.075  6  2.7858 0.0734918 .
Residuals           19.787 10
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Even though this design does not require analysis with half-normal effects plots, the next section will illustrate application of half-normal effects plots for this example.

7. Half-normal effects plots

In (almost) saturated designs, conventional analysis of variance methods are not very successful, because there are too few degrees of freedom for error. If one assumes a screening design, for which most effects are inactive, the inactive effects actually represent experimental error; however, it is not known a-priori, which are the active effects. Daniel (1959) proposed to use half-normal effects plots for diagnosing which effects are active, and Lenth (1989) proposed a numerical activity check for these, known as "Lenth's method". The critical values proposed by Lenth were later found to be conservative, and it was proposed to use simulated ones instead. Half-normal effects plots with Lenth's method and simulated critical values are implemented in function `halfnormal` of package **DoE.base**.

7.1. The principle

The standard use for such plots is with 2-level factors which are conventionally coded in -1/+1 coding (see, e.g., Grömping 2014c). Function `halfnormal` from package **DoE.base** covers not only these standard situations, but also offers half-normal plots for

- 2-level designs with a few error degrees of freedom. For these, it automatically augments the estimated effects with error effects, distinguishing these into lack-of-fit and pure error. Significance assessment can be done with Lenth's method on the augmented set of estimates, or with other methods proposed in the literature (Larntz and Whitcomb 1998; Edwards and Mee 2008). Note that error effects are not necessarily uniquely determined.

- 2-level designs with partially confounded effects. For these, it projects out all preceding effects from the remaining ones (thus, the plotting points depend on the model order for such situations).

- mixed level designs, for which there is no unique coding and the plotting points are coding dependent. Mixed level designs can also have error degrees of freedom or partially confounded effects.

The strategy chosen in function `halfnormal` seems to be similar to that applied in the JMP software screening platform (see Chapter 8 of SAS Institute, Inc. 2012), both regarding the treatment of error points and the single degree of freedom representations for factors with more than two levels. On the contrary, Design-Expert (Stat-Ease Inc. 2012) does not plot individual degrees of freedom, but scaled Chi-squared values for effects with more than one degree of freedom. This avoids the coding dependence, but has the adverse effect that the number of plotting points is small so that effect sparsity is not easily achieved.

The steps for augmenting the estimated effects with error degrees of freedom are described, e.g., in Grömping (2015a) and are very similar also to the suggestions by Langsrud (2001) in a different context. A coarse overview works as follows:

- Make sure the model matrix $\mathbf{X}$ has orthogonal columns all of which have the same Euclidean length; if this is not the case, $\mathbf{X}$ has to be pre-treated (see below).

- For N observations, a trivial saturated model matrix is the N-dimensional identity matrix $\mathbf{I}_N$. If a distinction between lack-of-fit and pure error is sought, one can replace this matrix by a matrix $\mathbf{S}$ of dummy variables for *distinct runs*, and additionally include appropriately scaled orthogonal contrast matrices for replicated runs. In the following, for simplicity, $\mathbf{I}_N$ is used.

- Residualize the matrix $\mathbf{I}_N$ by projecting out the model matrix $\mathbf{X}$, i.e., calculate the residual matrix $\mathbf{R} = \mathbf{I}_N - \mathbf{X}(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top$.

- Create the half-normal effects plot for the augmented model matrix $(\mathbf{X}|\mathbf{R})$, which has been created such that it has orthogonal columns of all the same length.

The pre-treatment mentioned in the first bullet is as follows: If $\mathbf{X}$ has orthogonal columns of varying Euclidean lengths, one simply has to normalize all columns to a common length. The case of non-orthogonal columns is more demanding and will be discussed using the model matrix $\mathbf{X}$ for a full model in an unreplicated full factorial for one 2-level and one 3-level factor, both in dummy coding, as given in Equation 1:

$$\mathbf{X} = \begin{pmatrix} Int & A_2 & B_2 & B_3 & A_2B_2 & A_2B_3 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \tag{1}$$

. The columns of this matrix are not orthogonal, which can be easily verified from looking at $\mathbf{X}^\top\mathbf{X}$. As the intercept estimates the mean for the combination A_1B_1 and the coefficients for the main effects columns estimate deviations from that level, there is an obvious dependency. With the interaction coefficients also measuring deviations of particular cells from additivity, there is another clear dependence. It is well-known which effects are estimable in factorial models: those are the overall means and the contrasts. A model matrix formulated to directly estimate these has orthogonal columns.

Xu and Wu (2001) showed that it is particularly useful to code all main effects columns using an orthogonal coding normalized to squared length N, where N is the number of runs: such a coding yields orthogonal columns of the model matrix for all effects up to degree s for any projection of s factors from a strength s design, if the interaction columns are created in the usual way as products of the normalized main effects contrast columns; the interaction columns will then also have squared Euclidean length N and will be orthogonal to each other and to the main effects columns. Such coding will be called Xu Wu coding in the sequel. It is available in two versions in package **DoE.base**: there are `contr.XuWu` and `contr.XuWuPoly` contrasts. The model matrix below is obtained by coding both factors A and B with `contr.XuWu` contrasts.

$$\mathbf{X} = \begin{pmatrix} Int & A_2 & B_2 & B_3 & A_2B_2 & A_2B_3 \\ 1 & -1 & -\sqrt{1.5} & -\sqrt{0.5} & \sqrt{1.5} & \sqrt{0.5} \\ 1 & 1 & -\sqrt{1.5} & -\sqrt{0.5} & -\sqrt{1.5} & -\sqrt{0.5} \\ 1 & -1 & \sqrt{1.5} & -\sqrt{0.5} & -\sqrt{1.5} & \sqrt{0.5} \\ 1 & 1 & \sqrt{1.5} & -\sqrt{0.5} & \sqrt{1.5} & -\sqrt{0.5} \\ 1 & -1 & 0 & \sqrt{2} & 0 & -\sqrt{2} \\ 1 & 1 & 0 & \sqrt{2} & 0 & \sqrt{2} \end{pmatrix} \tag{2}$$

If the design has strength s, a model matrix in a Xu Wu coding for main effects with up to s-factor interactions can also be obtained post-hoc from a model matrix $\mathbf{X}$ based on non-orthogonal coding by sequential orthogonalization and normalization: Project out the first column (intercept column) from the second, which is just subtraction of the column mean. Project out the first two columns from the third and so on. In addition, one also has to normalize all columns to squared length N (or any other common Euclidean length) in order to satisfy the requirements for a half-normal effects plot.

If the model has truly confounded effects that cannot be orthogonalized by simply applying Xu Wu coding, the same orthogonalization strategy can be applied, but the consequence is not only a recoding of in principle the same information, but an order-dependent removal of earlier confounded effects from later confounded effects.

7.2. Example application

The example design is an orthogonal array, i.e., main effect *contrasts* can be estimated independent of each other. However, depending on the coding, the actual *estimated coefficients* may be correlated, as was discussed above. For example, for 2-level factors, if -1/+1 coding is used, the estimates are uncorrelated, with 0/1 (dummy) coding, however, they are correlated. It is advisible to explicitly choose an orthogonal factor coding, ideally the Xu Wu variant discussed above. The code below creates an orthogonal main effects model matrix with squared Euclidean length $N = 72$ (output not shown).

```
R>VSGFS.XuWuPoly <- change.contr(VSGFS, "contr.XuWuPoly")
R>round(crossprod(model.matrix(lm(VSGFS.XuWuPoly))), 2)
```

Per default, function `halfnormal` refuses to work in case of correlated main effects (except in case of the perfect confounding of regular designs). The option `ME.partial=TRUE` can be used to change that; if the partial aliasing among main effects estimates is due to non-orthogonal coding in an orthogonal array, use of `ME.partial=TRUE` is acceptable in the light of the previous considerations, although it seems preferable to decide on an orthogonal coding explicitly.

For the example design, a main effects analysis is quite well-protected against bias from two-factor interactions. However, two-factor interactions may be quite heavily confounded with each other. Nevertheless, we will now consider an almost saturated array in order to demonstrate the most general usage of the function `halfnormal`.

```
R>hnAuto <- halfnormal(lm(VSGFS, response = "Content", degree = 2),
+   ME.partial = TRUE, cex.text = 0.9, cex = 0.9, xlim = c(0, 1.1),
+   ylim = c(0, 2.8), main = "Half-normal effects plot for Content")
R>hnXuWuPoly <- halfnormal(lm(VSGFS.XuWuPoly, response = "Content",
+   degree = 2), cex.text = 0.9, cex = 0.9, xlim = c(0, 1.1),
+   ylim = c(0, 2.8), main = "Half-normal effects plot for Content")
R>hnXuWuPolyreordered <- halfnormal(
+   lm(Content ~ (CDs + Sugar + CM + FilledVol + InocSize + ShakFreq +
+   Light) ^ 2, VSGFS.XuWuPoly), cex.text = 0.9, cex = 0.9, xlim = c(0, 1.1),
+   ylim = c(0, 2.8), main = "Half-normal effects plot for Content")
```

Figure 5 shows half-normal effects plots from a model with all main effects and two-factor interactions for the response variable "Content". In order to demonstrate the coding dependence of half-normal effects plots in case of factors with more than two levels, plot (a) shows automatical coding, plot (b) `contr.XuWuPoly` coding. Plot (c) shows the analysis for `contr.XuWuPoly` coding with the effects in different order. The codings between (a) and (b) differ in the level ordering for the 3-level factors; for the 4-level factor, the coding differences are more complicated. While there are clear visual differences between the differently-coded plots, the message is more or less the same: As already seen in the linear main effects model, light, sugar and CD are active factors. Furthermore, the Sugar by CD interaction and the Inoculum Size by Sugar interaction seem to have active components. However, the interaction effects are order dependent. The plot with different interaction orders brings up three further interactions into the possibly active range and drops the Sugar by CD interaction;

thus, clearly, one has to be cautious with statements on these effects from half-normal effects plots. However, all effects that show up in the half-normal effects plots have also been at least marginal in the Anova analysis of the previous section.

The calculation results have been stored in the list objects

`hnAuto`, `hnXuWuPoly`, and `hnXuWuPolyreordered`,

respectively. The list element `res` simply indicates which effects have been projected out from which other effects; furthermore, the results contain the model matrix after orthogonalization, and details about the orthogonalization itself. Printed output of orthogonalization steps has been suppressed here. The reader is encouraged to check the printed output of the simple command

```
R>hnAuto <- halfnormal(VSGFS, ME.partial = TRUE, plot = FALSE)
```

which explains how columns of the main effects model matrix stored in

```
R>mm <- hnAuto$mm[, 2:11]
```

are obtained from the original model matrix stored (without intercept column) in

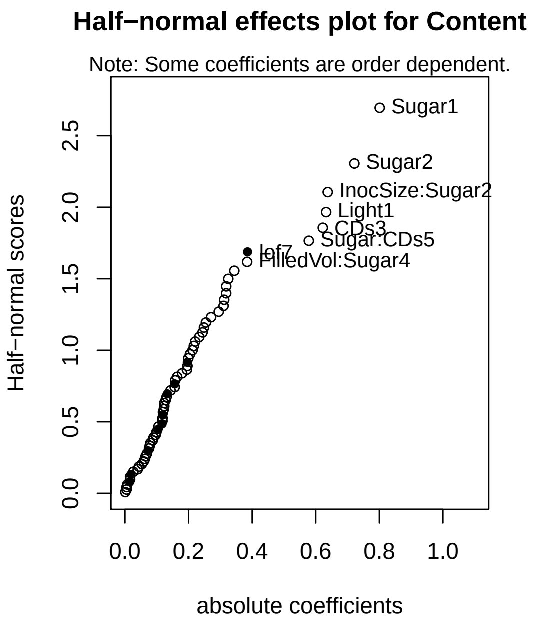
```
R>desnum(VSGFS)[, 1:10]
```

8. Further developments

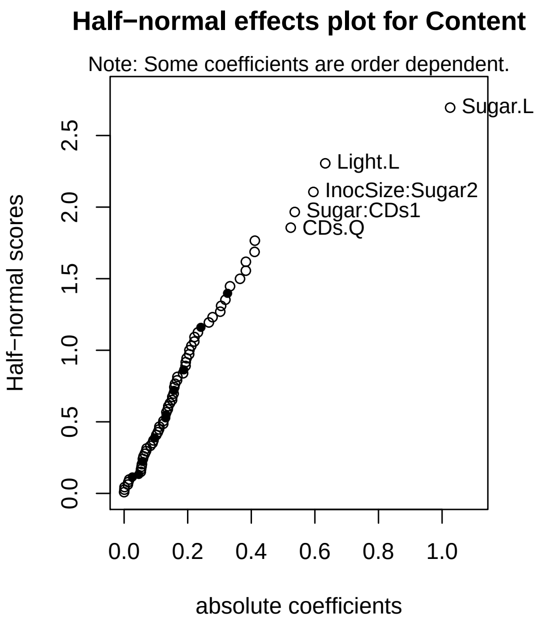
Package **DoE.base** tries a balancing act of offering tools both for practitioners with a relatively weak statistical background and for statistical experts. In order to save the former from avoidable grave mistakes, the package takes a cautious strategy issuing many warnings, where designs might be improvable or analyses might be inadequate. In various cases, it would be desirable to avoid unnecessary warnings; for example, there are arrays for which it is known from theoretical work (Butler 2005) that all choices for certain columns are GMA. For these arrays, warnings for array optimization should be eliminated, which requires some slightly tedious work.

The current design catalog covers many situations, but is also limited especially with respect to availability of non-regular designs. Augmenting the catalog with further useful non-regular designs is a difficult task and should not be addressed before studying in more detail the relation between recently developed design quality criteria like GR, GR_{ind}, ARFT and SCFT and a design's usefulness for experimentation.

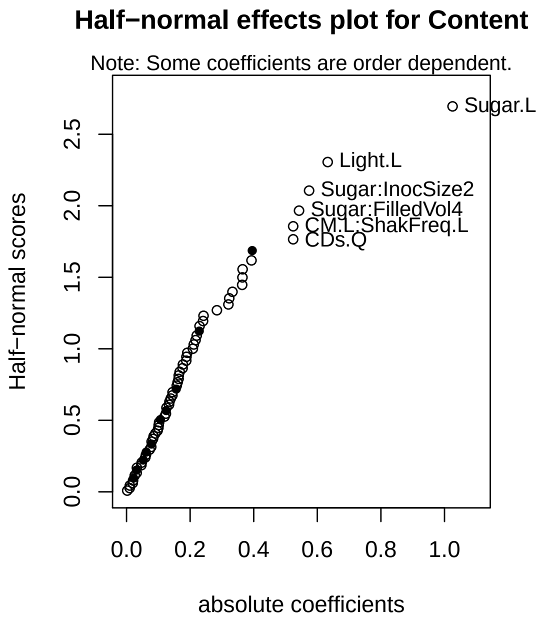
So far, the design catalog is used for selecting designs and optimizing column choices from them. Kuhfeld (2010) makes much more extensive use of the catalog within a SAS software



(a) Automatic coding, default order.



(b) `contr.XuWuPoly` coding, default order.



(c) `contr.XuWuPoly` coding, reversed order.

Figure 5: Half-normal effects plots for a model with all 2-factor interactions, labeling from Lenth's method with $\alpha = 0.05$.

macro suite for creating experimental designs from the arrays in the catalog, specifically created with marketing applications in mind (Kuhfeld 2010). Such use can certainly also be combined with the catalog in package **DoE.base**. The dependent package **support.CEs** (Aizaki 2012) implements choice experiments in R. However, implementing a functionality as flexible as the one offered in the (Kuhfeld 2010) SAS macros would require substantial additional effort.

Package **DoE.base** also serves as a tool for supporting further theoretical investigations into aspects like design regularity and new design quality criteria.

Last but not least, the class `design` and infrastructure for it are also available for use by other packages. Authors of other S3-based packages are invited to use it; the methods within package **DoE.base** for its class `design` can be adjusted to accommodate further types of design; please inform the package maintainer, if you would like a new type of design included into the methods offered.

Acknowledgments

Boyko Amarov and Hongquan Xu contributed code to the package. Creation of generalized resolution functionality was partly supported by grant GR 3843/1-1 of Deutsche Forschungsgemeinschaft.

References

Aizaki H (2012). “Basic Functions for Supporting an Implementation of Choice Experiments in R.” *Journal of Statistical Software, Code Snippets*, **50**(2), 1–24. ISSN 1548-7660. URL `http://www.jstatsoft.org/v50/c02`.

Box G, Tyssedal J (1996). “Projective Properties of Certain Orthogonal Arrays.” *Biometrika*, **83**, 950–955.

Butler N (2005). “Generalised Minimum Aberration Construction Results for Symmetrical Orthogonal Arrays.” *Biometrika*, **92**, 485–491.

Chambers J, Hastie T (1984). *Statistical Models in S.* Wadsworth & Brooks/Cole, Pacific Grove, CA.

Chasalow S (2012). ***combinat**: Combinatorics Utilities.* R package version 0.0-8, URL `http://CRAN.R-project.org/package=combinat`.

Collings B (1984). “Generating the Intrablock and Interblock Subgroups for Confounding in General Factorial Experiments.” *The Annals of Statistics*, **12**, 1500–1509.

Collings B (1989). “Quick Confounding.” *Technometrics*, **31**, 107–110.

Daniel C (1959). “Use of Half Normal Plots in Interpreting Two Level Experiments.” *Technometrics*, **1**, 311–340.

Deng LY, Tang B (1999). “Generalized Resolution and Minimum Aberration Criteria for Plackett-Burman and Other Nonregular Factorial Designs.” *Statistica Sinica*, **9**, 1072–1082.

Edwards D, Mee R (2008). "Empirically Determined p-Values for Lenth t-Statistics." *Journal of Quality Technology*, **40**, 368–380.

Eendebak P, Schoen E (2010). "Complete Series of Non-Isomorphic Orthogonal Arrays." URL `http://pietereendebak.nl/oapage/`.

Fox J, Weisberg S (2011). *An R Companion to Applied Regression*. 2nd edition. Sage, Thousand Oaks CA. URL `http://socserv.socsci.mcmaster.ca/jfox/Books/Companion`.

Grömping U (2011a). "Relative Projection Frequency Tables for Orthogonal Arrays." *Reports in Mathematics, Physics and Chemistry 1*, Beuth University of Applied Sciences Berlin, Germany.

Grömping U (2011b). "Tutorial for Designing Experiments Using the R Package RcmdrPlugin.DoE." *Reports in Mathematics, Physics and Chemistry 4*, Beuth University of Applied Sciences Berlin, Germany.

Grömping U (2013a). "Frequency Tables for the Coding Invariant Ranking of Orthogonal Arrays." *Reports in Mathematics, Physics and Chemistry 2*, Beuth University of Applied Sciences Berlin, Germany.

Grömping U (2013b). ***DoE.wrapper**: Wrapper Package for Design of Experiments Functionality.* R package version 0.8-9, URL `http://CRAN.R-project.org/package=DoE.wrapper`.

Grömping U (2013c). ***FrF2.catlg128**: Complete Catalogues of Resolution IV 128 Run 2-Level Fractional Factorials up to 24 Factors.* R package version 1.2-1, URL `http://CRAN.R-project.org/package=FrF2`.

Grömping U (2013d). ***RcmdrPlugin.DoE**: R Commander Plugin for (Industrial) Design of Experiments.* R package version 0.12-2, URL `http://CRAN.R-project.org/package=RcmdrPlugin.DoE`.

Grömping U (2014a). "Mosaic Plots Are Useful for Visualizing Low Order Projections of Factorial Designs." *The American Statistician*, **68**(2), 108–116.

Grömping U (2014b). ***FrF2**: Fractional Factorial Designs With 2-Level Factors.* R package version 1.7-1, URL `http://CRAN.R-project.org/package=FrF2`.

Grömping U (2014c). "R Package FrF2 for Creating and Analyzing Fractional Factorial 2-Level Designs." *Journal of Statistical Software*, **56**(1), 1–56. ISSN 1548-7660. URL `http://www.jstatsoft.org/v56/i01`.

Grömping U (2015a). "Augmented Half Normal Effects Plots in the Presence of a Few Error Degrees of Freedom." *Quality and Reliability International*, **31**(7), 1185–1196.

Grömping U (2015b). ***DoE.base**: Full Factorials, Orthogonal Arrays and Base Utilities for DoE Packages.* R package version 0.27-1, URL `http://CRAN.R-project.org/package=DoE.base`.

Grömping U, Xu H (2014). "Generalized Resolution for Orthogonal Arrays." *The Annals of Statistics*, **42**(3), 918–939.

Kobilinsky A, Bouvier A, Monod H (2015a). ***planor****: Generation of Regular Factorial Designs.* R package version 0.2-3, URL http://CRAN.R-project.org/package=planor.

Kobilinsky A, Monod H, Bailey R (2015b). "Automatic Generation of Generalised Regular Factorial Designs." *Preprint Series 7*, Issac Newton Institue for Mathematical Sciences, London, UK.

Kuhfeld W (2010). "Orthogonal Arrays." URL http://support.sas.com/techsup/technote/ts723.html.

Langsrud O (2001). "Identifying Significant Effects in Fractional Factorial Multiresponse Experiments." *Technometrics*, **43**, 415–424.

Larntz K, Whitcomb P (1998). "Use of Replication in Almost Unreplicated Factorials." Manuscript of a presentation given at the 42nd ASQ Fall Technical conference in Corning, New York, downloaded April 26, 2013., URL http://www.statease.com/pubs/use-of-rep.pdf.

Lenth R (1989). "Quick and Easy Analysis of Unreplicated Factorials." *Technometrics*, **31**, 469–473.

Meyer D, Zeileis A, Hornik K (2013). ***vcd****: Visualizing Categorical Data.* R package version 1.3-1, URL http://CRAN.R-project.org/package=vcd.

NIST/SEMATECH (2012). "NIST/SEMATECH e-Handbook of Statistical Methods, Section 3." Accessed 17 Feb 2015, URL http://www.itl.nist.gov/div898/handbook/pri/section3/pri33a.htm.

Plackett R, Burman J (1946). "The Design of Optimum Multifactorial Experiments." *Biometrika*, **33**(4), 305–325. doi:10.1093/biomet/33.4.305.

R Development Core Team (2015). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org.

SAS Institute, Inc (2012). *JMP Modeling and Multivariate Methods.* Cary, NC. Chapter 8, URL https://www.jmp.com/support/downloads/pdf/jmp1002/Modeling-and-Multivariate-Methods.pdf.

Schoen E, Eendebak P, Nguyen M (2010). "Complete Enumeration of Pure-Level and Mixed-Level Orthogonal Arrays." *Journal of Combinatorial Designs*, **18**(2), 123–140.

Stat-Ease Inc (2012). *Design-Expert, Version 8.*

Tang B, Deng LY (1999). "Minimum G2-Aberration for Nonregular Fractional Factorial Designs." *The Annals of Statistics*, **27**, 1914–1926.

Vasilev N, Schmitz C, Grömping U, Fischer R, Schillberg S (2014). "Assessment of Cultivation Factors that Affect Biomass and Geraniol Production in Transgenic Tobacco Cell Suspension Cultures." *PLOS one*, **9**(8), 1–7. doi:10.1371/journal.pone.0104620.

Venables B (2013). ***conf.design****: Construction of Factorial Designs.* R package version 2.0.0, URL http://CRAN.R-project.org/package=conf.design.

Venables WN, Ripley BD (2002). *Modern Applied Statistics with S.* Fourth edition. Springer, New York. ISBN 0-387-95457-0, URL http://www.stats.ox.ac.uk/pub/MASS4.

Wheeler B (2014). *pkgAlgDesign: Algorithmic Experimental Design.* R package version 1.1-7.2, URL http://CRAN.R-project.org/package=AlgDesign.

Xu H, Cheng SW, Wu C (2004). "Algorithmic Construction of Efficient Fractional Factorial Designs With Large Run Sizes." *Technometrics*, **46**, 280–292.

Xu H, Wu C (2001). "Generalized Minimum Aberration for Asymmetrical Fractional Factorial Designs." *The Annals of Statistics*, **29**(4), 1066–1077. doi:10.1214/aos/1013699993.

Appendix A. Class `design` and functionality for it

Generally, all design generating functions from packages **DoE.base** and **FrF2** (exceptions highlighted in the documentation) create an output design that is of S3 class `design` and thus follows a certain structure (cf. Section Appendix A) and allows application of certain inspection, modification and analysis methods and functions. The first sub section describes the class `design` itself, the second sub section functionality applicable to class `design` objects.

Appendix A.1. Class `design`

An object of S3 class `design` is a a data frame with the three attributes `desnum`, `run.order` and `design.info`.

- Attribute `desnum` can be used for a numeric version of the data frame, which may be useful for users who want to do manual matrix calculations. The package functionality itself makes little use of that attribute.
- Attribute `run.order` has the main purpose of always being able to switch back and forth between a standard order and the randomized run order.
- Attribute `design.info` is a list that contains all the important information on the design and is heavily used by the methods and functions discussed in the following sub section. This attribute will be described in some detail in the rest of this sub section.

The `design.info` attribute has some mandatory elements that have to be present for all class `design` objects and many elements that are needed for some types of designs only. The author's website contains a large table that details which types of designs need which elements of the `design.info` attribute. Table 1 lists the elements and their meaning/context.

Element	Data type	Role
Mandatory elements		
type	character string	identifies the type of design
nruns	number	number of runs (replications not counted)
nfactors	number	number of factors

Continued on next page

Table 1: Elements of the `design.info` attribute of class `design`.

Table 1 – continued from previous page

Element	Data type	Role
factor.names	named list	factor names and factor levels (or scale ends for quantitative factors)
replications	number	number of replications or repeated measurements per run
repeat.only	logical	if TRUE, the number given in the replications element refers to repeated measurements only
randomize	logical	if TRUE, run order has been randomized
seed	number	the seed used for randomization
creator	call or list of menu settings in GUI	the creation history of the object
Optional general elements		
response.names	vector of character strings	names of the response columns (column names from the data frame)
Elements for blocked designs (from functions `fac.design`, `rerandomize.design`, or `FrF2`)		
block.name	character string	name of block variable
nblocks	number	number of blocks
block.gen	number	Yates matrix column number(s) of factor(s) used for blocking (`FrF2`) or block generator matrix (`fac.design`)
blocksize	number	run size of each block (without replications)
bbreps	number	number of between block replications (identical to replications)
wbreps	number	number of within block replications (these can be proper replications or repeated measurements only)
Further element for full factorial designs and orthogonal array designs (functions `fac.design` or `oa.design`)		
nlevels	numeric vector with `nfactors` elements	number of levels for each factor
Further elements for general orthogonal array based designs (function `oa.design`)		
generating.oa	character string	name of the oa used (from catalog of orthogonal arrays or user-provided)
selected.columns	numeric vector with `nfactors` elements	column number from `generating.oa` for each factor
origin	character string	origin specification from `generating.oa` (empty, if there is none)
comment	character string	comment from `generating.oa` (empty, if there is none)
residual.df	number	residual degrees of freedom for main effects only analysis

Continued on next page

Table 1: Elements of the `design.info` attribute of class `design`.

Table 1 – continued from previous page

Element	Data type	Role
Further element for designs created by function `pb`		
ndummies	number	number of columns not assigned to experimental factors
Further elements for designs created by function `FrF2`		
aliased	list with character elements	information on the alias structure of the design up to degree 2
FrF2.version	character string	version number of package **FrF2**, when design was created
generators	character vector or design key matrix	design generators in the format "D=ABC" etc. for `FrF2` designs or design key matrix for designs created with package **planor**
catlg.name	character string	name of the catalog used for design creation
catlg.entry	list of length 1 of class `catlg`	the catalog entry used for the design
ntreat	number	identical to nfactors, present for blocked designs for historical reasons
aliased.with. blocks	character vector	lists 2fis that are aliased with the block main effect
base.design	character string	element of design catalog used for creating a blocked or split plot design from
nfac.WP	number	number of whole plot factors
nfac.SP	number	number of split plot factors
nWPs	number	number of whole plots
plotsize	number	run size of each plot (without replications)
res.WP	number	resolution of the whole plot portion of the design
map	numeric vector with k elements	mapping of base factors so that estimability or randomization restriction requirement is fulfilled
orig.fac.order	numeric vector with `nfactors` elements	order of original factors from function call for split plot designs
clear	logical	if TRUE, the design is clear (for estimability requirement)
res3	logical	if TRUE, resolution III has been permitted for estimability request
quantitative	logical vector with `nfactors` elements	`TRUE` elements indicate quantitative factors
ncube	number	number of cube points in a design with center points
ncenter	number	number of center points
coding	list of formulae	coding of quantitative factors (for use with package **rsm**)

Continued on next page

Table 1: Elements of the `design.info` attribute of class `design`.

Table 1 – continued from previous page

Element	Data type	Role
Further elements for designs created by function `Dopt.design`		
The function also outputs the split plot related elements and element `quantitative` described under specific elements for `FrF2` above.		
plot.name	character string	name of whole plot factor in split plot request
digits	number	number of digits to which quantitative factors are to be rounded
formula	formula	model formula for which D-optimality is to be achieved
constraint	logical expression	constraint applied to the experimental factors
optimality Criteria	named numeric vector	performance of design on several optimality criteria
Further elements for designs created by function `lhs.design`		
subtype	character string	type of latin hypercube design
The function also outputs `digits` and `optimalityCriteria` elements described under specific elements for `Dopt.design` above.		

Table 1: Elements of the `design.info` attribute of class `design`.

Apart from the elements listed in the table, there are further elements for class `design` objects created by design combination functions. These are not discussed here. For such designs, the entry types given in the table can also be replaced by lists of several such entries.

Appendix A.2. Functionality for class `design` objects

Package **DoE.base** offers inspection methods (`print`, `summary`, `plot`) and analysis methods (`plot`, `lm`, `halfnormal`) for this class, as well as a subsetting method by `"["`, which is useful for reordering experimental runs (switch between standard order and randomized order, re-randomize).

Besides these major functions, there are various further functions tailored to the need of class `design` objects, either as methods for generic functions or just as functions:

- Functions `design.info`, `run.order`, and `desnum` extract or set the respective attributes of a class `design` object.
- Functions `factor.names` and `response.names` get or set the respective element from the `design.info` attribute.
- Function `generators` extracts generating information for designs of types `FrF2` or `planor`.
- Function `getblock` extracts block information for replicated designs (see also Section 2).

- Function `rerandomize.design` allows to re-randomize a class `design` object without response data; this also allows explicit blocking of designs created with functions `oa.design` and `pb` (see also Section 5.3).
- Functions for designs that can be both in wide or in long format, i.e., parameter designs and designs with repeated measurements, can change between long and wide format or aggregate wide format designs: function `rep2wide` brings repeated measurement designs into wide format, `reptolong` does the opposite; function `paramtowide` brings a parameter design to wide format (irreversible, they are created in long format); function `aggregate.design` (method for the generic `aggregate`) aggregates designs in wide format into designs with a single response.
- Functions `add.response` and `col.remove` add responses or remove columns; function `response.names` can also be used to remove response columns.
- Function `qua.design` influences, which design columns are quantitative,
  function `change.contr` changes the contrasts of design columns.
- Functions `undesign` and `redesign` remove the class `design` properties from an object or reinstate them.
- Functions `cross.design` and `param.design` combine class `design` objects into crossed designs or parameter designs, respectively.
- Function `fix.design` is a method for the generic `fix` and has been adapted from package **utils**; it allows to edit `design` objects; however, its use is not recommended.
- Function `export.design` can export a design in html or csv format, together with an R workspace for the design. After entering response data in a spreadsheet program, responses can be added to the design itself using the `add.response` function.

Affiliation:

Ulrike Grömping
Department II – Mathematics, Physics, Chemistry
Beuth University of Applied Sciences Berlin
D-13353 Berlin
E-mail: `groemping@bht-berlin.de`
URL: `http://prof.beuth-hochschule.de/groemping/`