

Package ‘FrF2’

November 18, 2009

Title Fractional Factorial designs with 2-level factors

Version 1.0-1

Depends R(>= 2.8.0), BsMD, scatterplot3d, igraph, sfsmisc, DoE.base

Date 2009-11-17

Author Ulrike Groemping

Maintainer Ulrike Groemping <groemping@bht-berlin.de>

Description This package creates regular and non-regular Fractional Factorial designs. Furthermore, analysis tools for Fractional Factorial designs with 2-level factors are offered (main effects and interaction plots for all factors simultaneously, cube plot for looking at the simultaneous effects of three factors, full or half normal plot, alias structure in a more readable format than with the built-in function alias). The package is currently subject to intensive development. While much of the intended functionality is already available, some changes and improvements are still to be expected. Suggestions are welcome.

License GPL (>= 2)

LazyLoad yes

LazyData yes

Encoding latin1

URL <http://prof.tfh-berlin.de/groemping/DoE/>,
<http://prof.tfh-berlin.de/groemping/>

Repository CRAN

Date/Publication 2009-11-18 07:57:12

R topics documented:

FrF2-package	2
add.center	5
aliases	8
block	9
blockpick	12
CatalogueAccessors	15
cubePlot	18
DanielPlot	20
estimable.2fis	22
fold.design	24
FrF2	27
IAPlot	39
pb	42
splitplot	46
StructurePickers	49
Index	52

FrF2-package

Fractional Factorial designs with 2-level factors

Description

This package creates regular and non-regular Fractional Factorial designs. Furthermore, analysis tools for Fractional Factorial designs with 2-level factors are offered (main effects and interaction plots for all factors simultaneously, cube plot for looking at the simultaneous effects of three factors, full or half normal plot, alias structure in a more readable format than with the built-in function `alias`).

The package works together with packages `DoE.base` and `DoE.wrapper`.

Details

The package is currently subject to intensive development; most key functionality is now included. Some changes to input and output structures may still occur. Please contact me, if you have suggestions.

This package designs and analyses Fractional Factorial experiments with 2-level factors. Regular (function `FrF2`) and non-regular (function `pb`) 2-level fractional factorial designs can be generated. Regular fractional factorials can be generated as blocked or split-plot designs, and hard-to-change factors can be specified in order to keep the number of level changes low. Analysis facilities work for completely aliased designs only, i.e. e.g. not for analysing Plackett-Burman designs with interactions.

Functions `fac.design`, `fractionate` or `oa.design` from Chambers and Hastie (1993) have been used as role models e.g. for the option `factor.names` or for outputting a data frame with attributes. However, S compatibility has not been considered in devising this package.

In terms of analysis, the package works on linear models and enables convenient main effects and interaction plots (functions `MEPlot` and `IAPlot`) similar to those offered by Minitab software for all factors simultaneously, even though especially the interactions are often aliased, i.e. the model is typically singular. For the (rare) case of suspected three-factor-interactions, function `cubePlot` displays a cube with corners labeled with the (modeled) means of three factors simultaneously. Furthermore, the function `DanielPlot` from package **BsMD** has been modified to automatically label effects significant according to the Lenth-criterion, and to provide more usage comfort to the analyst. Finally, the function `aliases` determines the alias structure of a Fractional Factorial 2-level design in a format more suitable for human readers than the output from the built-in function `alias`.

CAUTION: Note that the analysis facilities currently do not make use of the design information attached to the design. For example, an effects plot function will treat a linear model based on a split-plot-design in the same way as if the design were a standard fractional factorial.

Author(s)

Ulrike Groemping

Maintainer: Ulrike Groemping <groemping@bht-berlin.de>

References

- Box G. E. P, Hunter, W. C. and Hunter, J. S. (2005) *Statistics for Experimenters, 2nd edition*. New York: Wiley.
- Chambers, J.M. and Hastie, T.J. (1993). *Statistical Models in S*, Chapman and Hall, London.
- Chen, J., Sun, D.X. and Wu, C.F.J. (1993) A catalogue of 2-level and 3-level orthogonal arrays. *International Statistical Review* **61**, 131-145.
- Daniel, C. (1959) Use of Half Normal Plots in Interpreting Two Level Experiments. *Technometrics*, **1**, 311-340.
- Hedayat, A.S., Sloane, N.J.A. and Stufken, J. (1999) *Orthogonal Arrays: Theory and Applications*, Springer, New York.
- Lenth, R.V. (1989) Quick and easy analysis of unreplicated factorials. *Technometrics*, **31**, 469-473.
- Mee, R. (2009). *A Comprehensive Guide to Factorial Two-Level Experimentation*. New York: Springer.
- Montgomery, D.C. (2001). *Design and Analysis of Experiments* (5th ed.). Wiley, New York.
- Plackett, R.L.; Burman, J.P. (1946) The design of optimum multifactorial experiments. *Biometrika* **33**, 305-325.

See Also

The key design generating functions: [FrF2](#), [pb](#)

S3 class [design](#)

Related packages: [DoE.base](#), [DoE.wrapper](#), [BsMD](#);

Graphical analysis functions: [MEPlot](#), [IAPlot](#), [cubePlot](#), [DanielPlot](#)

Analysis of alias structure for linear models of FrF2 designs: [aliases](#)

Examples

```

### for examples on design generation, cf. functions pb and FrF2

### Injection Molding Experiment. Box et al. 1978.
data(BM93.e3.data) #from BsMD
iMdat <- BM93.e3.data[1:16,2:10] #only original experiment
# make data more user-friendly
colnames(iMdat) <- c("MoldTemp","Moisture","HoldPress","CavityThick","BoostPress",
                    "CycleTime","GateSize","ScrewSpeed", "y")
# linear model with all main effects and 2-factor interactions
iM.lm <- lm(y ~ (. )^2, data = iMdat)
# determine aliases
aliases(iM.lm)
# coded version
aliases(iM.lm, code=TRUE)
# normal plot of effects, default is autolabel with alpha=0.05
DanielPlot(iM.lm)
DanielPlot(iM.lm,code=TRUE)
DanielPlot(iM.lm,code=TRUE,alpha=0.5)
# half normal plot of effects
DanielPlot(iM.lm,code=TRUE,alpha=0.5,half=TRUE)
# main effects plots
MEPlot(iM.lm)
# interaction plots
IAPlot(iM.lm)
# interaction plots with attention drawn to aliases
IAPlot(iM.lm, show.alias=TRUE)
# alias groups corresponding to interaction plots
aliases(iM.lm)$aliases[9:15]
# cube plot of three factors
# (not very useful for this model, for demonstration only)
## per default, modeled means are shown
## this does not make a difference here, since the main effect of
## ScrewSpeed is confounded with the MoldTemp:HoldPress:BoostPress
## interaction, so that the three-factor-interaction is indirectly included
## in the modeled means
cubePlot(iM.lm, "MoldTemp", "HoldPress", "BoostPress")
## modeled means without a three-factor interaction
cubePlot(lm(y ~ (MoldTemp+HoldPress+BoostPress)^2, data = iMdat),
         "MoldTemp", "HoldPress", "BoostPress")
## modeled=FALSE reverts to showing the apparent three-factor interaction
cubePlot(lm(y ~ (MoldTemp+HoldPress+BoostPress)^2, data = iMdat),
         "MoldTemp", "HoldPress", "BoostPress", modeled=FALSE)
## cubePlot also works on raw data
cubePlot(iMdat$y, iMdat$MoldTemp, iMdat$HoldPress, iMdat$BoostPress)
## plotting functions also work directly on designs,
## if these have been generated from functions FrF2 or pb:
plan <- FrF2(16, 7)
plan <- add.response(plan, rnorm(16))
MEPlot(plan)
IAPlot(plan)
DanielPlot(plan)

```

add.center

Function to add center points to a 2-level fractional factorial

Description

This function adds center points to a 2-level fractional factorial design. All factors must be quantitative!

Usage

```
add.center(design, ncenter, distribute=NULL, ...)
iscube(design, ...)
```

Arguments

design	a data frame of class design that contains a 2-level fractional factorial (regular or non-regular); design must neither be a split-plot nor a long version parameter design. For function add.center, the design must not contain center points yet, while it has to contain center points for function iscube. For function add.center, blocked and replicated (or repeated measurement) designs must be in the original run order (column run.no in run.order attribute in ascending order), as the algorithm relies on the related runs being grouped as expected. An error is thrown, if this condition is violated.
ncenter	the number of center points to be added to each block
distribute	the number of positions over which to distribute the center points within each block; note that the center points are not randomized but placed evenly throughout the (hopefully randomized) design (but see also the details section); if distribute is NULL, center points are all added at the end for non-randomized designs and are distributed as evenly as possible to beginning, middle and end of the experiment for randomized designs. distribute must neither be larger than ncenter nor than the number of runs of the design plus one.
...	currently not used

Details

Function add.center adds center points to 2-level fractional factorial designs. Instead of using this function directly, center points should usually be added directly with calls to functions [FrF2](#) or [pb](#). These make use of function add.center for this purpose.

Center points are added to designs for three main reasons: they provide a repeated benchmark run that can alert the experimenter to unplanned changes in experimental conditions, they provide an independent estimate of experimental error, and finally they provide a possibility for checking whether a first order model is sufficient. Especially for the first purpose, package FrF2 follows the

recommendation in Montgomery (2001, p.275). To distinguish them from the center points, the original fractional factorial runs are called “cube points”.

Addition of center points does not affect estimates for main effects and interactions. The difference between the averages of cube points and center points gives an indication whether quadratic terms might be needed in the model.

For blocked designs and properly replicated designs, `ncenter` center points are added to *each* (replication) block. In case of repeated measurements, center points are also measured repeatedly.

Center points are distributed as evenly as possible over the `distribute` selected positions throughout each block. `distribute=1` always adds all center points at the end of each block. If `distribute > 1`, (each block of) the design starts and ends with a (group of) center point(s), and the `distribute` positions for placing center points are as evenly placed throughout (each block of) the design as possible.

If `ncenter` is not a multiple of `distribute`, some center point groups have one more center point than others. If `ncenter%%distribute` is one or two only, the beginning and (for two) the end of (each block of) the design have one more center point, otherwise the `ncenter%%distribute` extra center points are randomized over the center point positions.

Function `iscube` provides a logical vector that is TRUE for cube points and FALSE for center points. Its purpose is to enable use of simple functions for “clean” 2-level fractional factorials like `MEPlot`.

Value

A data frame of class `design` with `ncenter` center point runs per block (or per replication block) added to the `design` (and its `desnum` and `run.order` attributes). The `run.no.in.std.order` column of `run.order` is “0” for the center points.

Existing response values for cube runs are preserved, and response values for the new center point runs are NA. Note, however, that center points should be added BEFORE running the experiment in order to benefit from all their useful properties; this should best be done within functions `pb` or `FrF2`.

The design is identifiable as a design with center points by the suffix `.center` to the `type` element of attribute `design.info`, and the elements `ncube` and `ncenter` are added (with the updated `nruns` being their sum).

Function `iscube` provides a logical vector that is TRUE for cube points and FALSE for center points.

Note

This function is still somewhat experimental.

Author(s)

Ulrike Groemping

References

Montgomery, D.C. (2001). *Design and Analysis of Experiments (5th ed.)*. Wiley, New York.

See Also

See also as [pb](#), [FrF2](#)

Examples

```
## purely technical example
plan <- FrF2(8,5, factor.names=c("one","two","three","four","five"))
add.center(plan, 6)
add.center(plan, 6, distribute=1)
add.center(plan, 6, distribute=6)
add.center(plan, 6, distribute=4)

## very artificial analysis example
plan <- FrF2(8,4, factor.names=list(one=c(0,10),two=c(1,3),three=c(25,32),four=c(3.7,4.8)))
## add some response data
y <- c(2+desnum(plan)%*%c(2,3,0,0) +
      1.5*apply(desnum(plan)[,c(1,2)],1,"prod") + rnorm(8))
## the "c()" makes y into a vector rather than a 1-column matrix
plan <- add.response(plan, y)
## analysing this design provides an impression
MEPlot(lm(y~(.)^2, plan))
IAPlot(lm(y~(.)^2, plan))
DanielPlot(lm(y~(.)^2,plan), half=TRUE, alpha=0.2)
## tentative conclusion: factors one and two do something
## wonder whether the model with one and two and their interaction is sufficient
## look at center points (!!! SHOULD HAVE BEEN INCLUDED FROM THE START,
## but maybe better now than not at all)
## use distribute=1, because all center points are run at the end
plan <- add.center(plan, 6, distribute=1)
## conduct additional runs for the center points
y <- c(y, c(2+desnum(plan)[!iscube(plan),1:4]%*%c(2,3,0,0) +
      1.5*apply(desnum(plan)[!iscube(plan)],[,c(1,2)],1,"prod") + rnorm(6)))
## add to the design
planc <- add.response(planc, y, replace=TRUE)
## sanity check: repeat previous analyses for comparison, with the help of function iscube
MEPlot(lm(y~(.)^2, planc, subset=iscube(planc)))
IAPlot(lm(y~(.)^2, planc, subset=iscube(planc)))
DanielPlot(lm(y~(.)^2, planc, subset=iscube(planc)), half=TRUE, alpha=0.2)
## quick check whether there a quadratic effect is needed: is the cube indicator significant
summary(lm(y~(.)^2+iscube(planc), planc))
## (in this unrealistic example, the quadratic effect is dominating everything else;
## with an effect that strong in practice, it is likely that
## one would either have expected a strong non-linearity before conducting the experiment
## OR that the effect is not real but the result of some stupid mistake
## alternatively, the check can be calculated per hand (cf. e.g. Montgomery, Chapter 11):
((mean(planc$y[iscube(planc)])-mean(planc$y[!iscube(planc)]))^2*8*6/(8+6)/var(y[!iscube(planc)]))
## must be compared to the F-quantile with 1 degree of freedom
## is the square of the t-value for the cube indicator in the linear model
```

 aliases

Alias structure for fractional factorial 2-level designs

Description

Functions to examine the alias structure of a fractional factorial 2-level design

Usage

```
aliases(fit, code = FALSE, condense=FALSE)
aliasprint(design, ...)
```

Arguments

<code>fit</code>	a linear model object with only 2-level factors as x-variables; the function will return an error, if the model contains partially aliased effects (like interactions in a Plackett-Burman design for most cases)
<code>code</code>	if TRUE, requests that aliasing is given in code letters (A, B, C etc.) instead of (potentially lengthy) variable names; in this case, a legend is included in the output object.
<code>condense</code>	if TRUE, reformats the alias information to be comparable to the version calculated by internal function <code>alias3fi</code> ; does not work with models with higher than 3-way interactions; for up to 3-way interactions, the output may be more easily readable
<code>design</code>	a data frame of class <code>design</code> that should contain a fractional factorial 2-level design; the function does not print anything if the design is of different nature
<code>...</code>	further arguments to function <code>print.default</code>

Value

Function `aliasprint` returns NULL and is called for its side effects only.

Per default, Function `aliases` returns a list with two elements:

<code>legend</code>	links the codes to variable names, if <code>code=TRUE</code> .
<code>aliases</code>	is a list of vectors of aliased effects.

If option `condense` is TRUE, the function returns a list with elements `legend`, `main`, `fi2` and `fi3`; this may be preferable for looking at the alias structure of larger designs.

Author(s)

Ulrike Groemping

References

Box G. E. P, Hunter, W. C. and Hunter, J. S. (2005) *Statistics for Experimenters, 2nd edition*. New York: Wiley.

See Also

[FrF2-package](#) for information on the package, [alias](#) for the built-in R-function, [IAPlot](#) for effects plots

Examples

```
### Injection Molding Experiment. Box et al. 1978.
data(BM93.e3.data) #from BsMD
iMdat <- BM93.e3.data[1:16,2:10] #only original experiment
# make data more user-friendly
colnames(iMdat) <- c("MoldTemp", "Moisture", "HoldPress", "CavityThick",
  "BoostPress", "CycleTime", "GateSize", "ScrewSpeed", "y")
# determine aliases with all 2-factor-interactions
aliases(lm(y ~ (.)^2, data = iMdat))
# coded version
aliases(lm(y ~ (.)^2, data = iMdat), code=TRUE)
# determine aliases with all 3-factor-interactions
aliases(lm(y ~ (.)^3, data = iMdat), code=TRUE)
# show condensed form
aliases(lm(y ~ (.)^3, data = iMdat), code=TRUE, condense=TRUE)
# determine aliases for unaliased model
aliases(lm(y ~ ., data = iMdat))
```

 block

Statistical and algorithmic aspects of blocking in FrF2

Description

This help page documents the statistical and algorithmic details of blocking in FrF2

Details

Blocking is done with the purpose to balance the design with respect to a factor that is known or strongly suspected to have an influence but is not in itself of interest, and it is usually assumed that block factors do not interact with experimental factors. Examples are batches of material that are not large enough to accommodate the complete experiment so that e.g. half the experiment is done on the first batch and the other half on the second batch (two blocks). The block factor should be orthogonal to the experimental factors, at least to their main effects. Per default, it is also requested that the block factor is orthogonal to the 2-factor interactions. This can be changed by the user, if no such design can be found.

Blocking is currently implemented for regular fractional factorial designs only.

There are two principal ways to handle blocked designs, manual definition (i.e. the user specifies exactly which columns are to be used for which purpose) and automatic definition. Each situation has its specifics. These are detailed below. For users with not so much mathematical/statistical background, it will often be best to use the automatic way, specifying the treatment factors of interest via `nfactors` or `factor.names` and a single number for `blocks` or `WPs`. Users with more mathematical background may want to use the manual definitions, perhaps in conjunction with published catalogues of good block designs, or after inspecting possibilities with functions [blockpick](#) or [blockpick.big](#).

Manual definition of blocked designs for regular fractional factorials The user can start from a design with a number of factors and manually specify which factors or interactions are to be used as block generators. If this route is chosen, `blocks` can be a vector of factor names or factor letters, or of the same form as generators, except that not only base factors but all factors can be used and single factors are permitted (which would lead to resolution II designs if used in generators). For example,

```
block = Letters[c(2, 4, 5)]
```

or

```
block = list(2, 4, 5)
```

specify that the 2nd, 4th and 5th factor are to be used as block generators, while

```
block = c("Day", "Shift")
```

indicates that the named factors “Day” and “Shift” specified in `factor.names` are to be treated as blocking factors). In this case, the number of blocks or whole plots is calculated, and a new factor with the default name “Blocks” (in general the name chosen in option `block.name`) is generated, which would for example contain as levels the Day/Shift combinations. It is also possible to choose interaction effects rather than factors themselves as block generators, e.g.

```
block = c("ABCD", "EFGH")
```

or

```
block = list(c(1, 2, 3, 4), c(5, 6, 7, 8)) .
```

The chosen effects and all interactions between them generate the blocks. CAUTION: If the user manually generates a blocked design, it is his/her responsibility to ensure a good choice of design (e.g. by using a catalogued design from Bisgaard 1994, Sun, Wu and Chen 1997, or Cheng and Wu 2002).

Automatic definition of blocked designs for regular fractional factorials If the user only specifies the number of blocks required for the experiment, function `FrF2` automatically generates the blocks. For full factorial designs, function `FrF2` uses the Sun, Wu and Chen (1997) catalogue of blocked designs (implemented in function `blockpick`). Otherwise, depending on the size of the problem, function `FrF2` uses function `blockpick` or function `blockpick.big` for finding an appropriate allocation of block generator columns: Smaller problems

```
(choose(nruns-1-nfactors, k.block) < 100000)
```

are treated with `blockpick`.

The search for an appropriate blocked design starts with the overall best unblocked design (in terms of aberration or `MaxC2`, if requested). If this best design does not yield an adequate blocking possibility, the search continues with the next best design and so forth.

For the smaller problems, function `blockpick` looks for `k.block` independent subsets among the eligible columns of the design. (The eligible columns are all columns of the Yates matrix that are neither occupied by treatment main effects nor by 2fis among treatments (if `alias.block.2fis=FALSE`, which is the default), or all columns of the Yates matrix that are not occupied by treatment main effects (if `alias.block.2fis=TRUE`). Note that no effort is made to avoid aliasing with 2-factor interactions, if `alias.block.2fis=TRUE` is chosen.

For the larger problems, function `blockpick.big` permutes the `k`-base factors of candidate designs with `nfactors + k.block` factors in search of a design the first `k.block`-factors of which can be used for block construction. In the latter case, any specification of design (via options `design` or `generators`) is ignored. Note that function `blockpick.big` is not guaranteed to find an existing blocked design.

Sun, Wu and Chen (1997) provide a catalogue of blocked designs with a few quality criteria, and they state that there is no single best design, but that the choice depends on the situation. `FrF2` always comes up with one specific solution design. Comparisons to the catalogued designs in Sun, Wu and Chen (1997) have shown that the designs found in `FrF2` are often but not always isomorphic to the catalogued ones. Differences do occur, especially if the base designs are resolution III, or if `blockpick.big` has to be used. Users who want to be certain to use a “best” blocked design should manually implement a specific catalogued design or inspect several solutions from functions `blockpick` `blockpick.big`.

Please contact me with any suggestions for improvements.

Author(s)

Ulrike Groemping

References

- Bisgaard, S. (1994a). Blocking generators for small 2^{k-p} designs. *J. Quality Technology* **26**, 288-294.
- Chen, J., Sun, D.X. and Wu, C.F.J. (1993) A catalogue of 2-level and 3-level orthogonal arrays. *International Statistical Review* **61**, 131-145.
- Cheng, C.-S. and Tsai, P.-W. (2009). Optimal two-level regular fractional factorial block and split-plot designs. *Biometrika* **96**, 83-93.
- Cheng, S.W. and Wu, C.F.J. (2002). Choice of optimal blocking schemes in 2-level and 3-level designs. *Technometrics* **44**, 269-277.
- Sun, D.X., Wu, C.F.J. and Chen, Y.Y. (1997). Optimal blocking schemes for 2^p and 2^{n-p} designs. *Technometrics* **39**, 298-307.

See Also

See Also `FrF2` for regular fractional factorials, `catlg` for the Chen, Sun, Wu catalogue of designs and some accessor functions, and `splitplot` for the statistical aspects of split-plot designs.

Examples

```
##### automatic blocked designs #####
## from a full factorial ##
FrF2(8,3,blocks=2)
## with replication
run.order(FrF2(8,3,blocks=2,wbreps=2))
run.order(FrF2(8,3,blocks=2,wbreps=2,repeat.only=TRUE))
run.order(FrF2(8,3,blocks=2,bbreps=2))
run.order(FrF2(8,3,blocks=2,bbreps=2,wbreps=2))

## automatic blocked design with fractions
FrF2(16,7,blocks=4,alias.block.2fis=TRUE)
## isomorphic non-catalogued design as basis
FrF2(16,gen=c(7,11,14),blocks=4,alias.block.2fis=TRUE)
## FrF2 uses blockpick.big and ignores the generator
FrF2(64,gen=c(7,11,14),blocks=16,alias.block.2fis=TRUE)
```

```
##### manual blocked design #####
### example that shows why order of blocks is not randomized
### can of course be randomized by user, if appropriate
FrF2(32,9,blocks=c("Day","Shift"),alias.block.2fis=TRUE,
     factor.names=list(Day=c("Wednesday","Thursday"), Shift=c("Morning","Afternoon"),
                       F1="",F2="",F3="",F4="",F5="",F6="",F7=""), default.levels=c("current","new"))
```

blockpick

Function to show potential block assignments

Description

Functions to investigate potential assignments of blocks and show alias information of resulting designs, meant for expert users

Usage

```
blockpick(k, gen, k.block, design = NULL, show = 10,
          alias.block.2fis = FALSE, select.catlg = catlg)
blockpick.big(k, gen, k.block, design = NULL, show = 10,
             alias.block.2fis = FALSE, select.catlg = catlg)
```

Arguments

k	the number of base factors (designs have 2^k runs)
gen	vector of generating columns from Yates matrix; for a full factorial, choose <code>gen = 0</code> or <code>gen=numeric(0)</code> for no generating columns. For function <code>blockpick</code> , <code>gen</code> refers to the generators of the base design only, and block columns are automatically added by <code>blockpick</code> . For function <code>blockpick.big</code> , <code>gen</code> refers to the generators for treatment factors and block generators. In fact, <code>blockpick.big</code> will always use the first <code>k.block</code> (base) factors for block generation. Hence, for example for generating a design in 64 runs and 7 factors with 32 blocks, <code>gen</code> must have 6 entries in order to accomodate the 7 treatment factors together with the 5 block generators.
k.block	number of base factors needed for constructing blocks; there will be $2^{k.block}$ blocks in the design
design	design name (character string) of a specific design from the catalogue given in <code>select.catlg</code>
show	numeric integer indicating how many results are to be shown; the search for possible allocations stops, once <code>show</code> variants have been found. Note that the best designs may not be found early in the process, especially if a large number of eligible columns is available and many blocks are needed (e.g. full factorial in 64 runs with 16 blocks). In such cases, increasing <code>show</code> may lead to finding a better design (but may also increase calculation time from long to unbearable).

```
alias.block.2fis
    logical, indicates whether 2fis may be aliased with blocks
select.catlg design catalogue of class catlg
```

Details

Function `blockpick` is used per default by function `FrF2` for problems with `choose(nruns-1-nfactors, k.block) < 100000`, otherwise `blockpick.big` is used. `blockpick` is preferable for smaller problems, because it will find a design, if it exists. However, it may take a long time and/or much storage space in problems with large numbers of runs and blocks.

Both functions investigate the potential assignment of blocks such that main effects of treatment factors are not aliased with block main effects. It is left to the user whether or not 2fis among treatment effects may be aliased with block main effects (option `alias.block.2fis`).

Following Sun, Wu and Chen (1997), there is no single best block assignment. `blockpick` uses their catalogue for full factorials (implemented up to 256 runs). For fractional factorials, it develops designs according to a principle similar to that underlying the Sun Wu Chen catalogue that works also in uncatalogued situations.

Function `blockpick.big` uses a strategy similar to `splitpick` and `leftadjust` and often finds a solution quickly where `blockpick` does not work with the given resources. However, it is not guaranteed to find existing solutions or a best solution.

Value

The function `blockpick` outputs a list of entries with information on at most `show` suitable assignments. It ends with an error, if no suitable solution can be found.

```
gen          generator column numbers of the base design (w.r.t. the Yates matrix)
basics       named vector with number of runs (nruns), number of blocks (nblocks),
             number of treatment factors (ntreat) and resolution of base design (res.base);
             the vector is numeric or character, depending on whether resolution is known
             exactly or as "5+" only
blockcols    matrix with at most show rows; each row contains the k.block column num-
             bers (w.r.t. the Yates matrix) of the block generators for the current assign-
             ment (the 2^k.block-1 columns for block main effects can be obtained from
             these).
alias.2fis.block
             list of character vectors, which contain the 2fis aliased with block main effects
             for the respective rows of blockcols
nblock.2fis  vector with number of 2fis aliased with block main effects for the respective
             rows of blockcols
nclear.2fis  vector with number of 2fis clear (of aliasing with block main effects and treat-
             ment main effects or 2fis) for the respective rows of blockcols
clear.2fis   list of character vectors, which contain the 2fis that are counted in nclear.2fis
             for the respective rows of blockcols
```

Author(s)

Ulrike Groemping

References

Chen, J., Sun, D.X. and Wu, C.F.J. (1993) A catalogue of 2-level and 3-level orthogonal arrays. *International Statistical Review* **61**, 131-145.

Sun, D.X., Wu, C.F.J. and Chen, Y.Y. (1997). Optimal blocking schemes for 2^p and 2^{n-p} designs. *Technometrics* **39**, 298-307.

See AlsoSee Also [FrF2](#)**Examples**

```
## look at possibilities for running a 32 run full factorial in 8 blocks
## running this without alias.block.2fis=TRUE throws an error: not possible
## Not run: blockpick(5,0,3)
## the 10th design with block generators 3,12,21 has the most clear 2fis
blockpick(5,0,3,alias.block.2fis=TRUE)
## function FrF2 can be used to manually accomodate this design
des32.5fac.8blocks.MaxC2 <- FrF2(32,5,blocks=c(3,12,21))
design.info(des32.5fac.8blocks.MaxC2)

## look at possibilities for blocking design 7-3.1 from Chen, Sun, Wu catalogue
blockpick(4,design="7-3.1",k.block=2,alias.block.2fis=TRUE)

## big design
## running this throws an error on many machines because of too little memory
## Not run: blockpick(6,design="7-1.2",k.block=5,alias.block.2fis=TRUE)
## for obtaining a design for this scenario with blockpick.big,
## the number of factors must be increased to 7+k.block=12
## designs 12-6.1 and 12-6.2 dont do it, 12-6.3 does
bpb <- blockpick.big(6,design="12-6.3",k.block=5,alias.block.2fis=TRUE)
bpb
## based on the result of blockpick.big, a blocked design can be obtained as follows:
des64.7fac.32blocks <- FrF2(64,gen=bpb$gen[1,], blocks = as.list(1:5),
  alias.block.2fis=TRUE)
str(des64.7fac.32blocks)
## if the seven factors are to be named A,...,G:
des64.7fac.32blocks <- FrF2(64,gen=bpb$gen[1,], blocks = as.list(1:5),
  alias.block.2fis=TRUE, factor.names=c(paste("b",1:5,sep=""),Letters[1:7]))
str(des64.7fac.32blocks)
```

CatalogueAccessors *Catalogue file and accessor functions*

Description

Functions to select elements or extract information from design catalogues of class `catlg`

Usage

```
res.catlg(catlg)
nruns.catlg(catlg)
nfac.catlg(catlg)
WLP.catlg(catlg)
nclear.2fis.catlg(catlg)
clear.2fis.catlg(catlg)
all.2fis.clear.catlg(catlg)
catlg
## S3 method for class 'catlg':
catlg[i]
## S3 method for class 'catlg':
print(x, name="all", nruns="all", nfactores="all",
      res.min=3, MaxC2=FALSE, show=10,
      gen.letters=FALSE, show.alias=FALSE, ...)
block.catlg
```

Arguments

<code>catlg</code>	Catalogue of designs of class <code>catlg</code> (cf. details section)
<code>i</code>	vector of index positions or logical vector that can be used for indexing a <code>catlg</code> object
<code>x</code>	an object of class <code>catlg</code>
<code>name</code>	character vector of entry names from <code>x</code> ; default “all” means: no selection made
<code>nruns</code>	numeric integer (vector), giving the run size(s) for entries of <code>x</code> to be shown; default “all” means: no selection made
<code>nfactors</code>	numeric integer (vector), giving the factor number(s) for entries of <code>x</code> to be shown; default “all” means: no selection made
<code>res.min</code>	numeric integer giving the minimum resolution for entries of <code>x</code> to be shown
<code>MaxC2</code>	logical indicating whether designs are ordered by minimum aberration (default, <code>MaxC2=FALSE</code>) or by maximum number of clear 2fis (<code>MaxC2=TRUE</code>)
<code>show</code>	integer number indicating maximum number of designs to be shown; default is 10
<code>gen.letters</code>	logical indicating whether the generators should be shown as column numbers (default, <code>gen.letters=FALSE</code>) or as generators with factor letters (e.g. <code>E=ABCD</code> , <code>gen.letters=TRUE</code>)

`show.alias` logical indicating whether the alias structure (up to 2fis) is to be printed
`...` further arguments to function `print`
`block.catlg` data frame with block generators for full factorial designs up to 256~runs, taken from Sun, Wu and Chen (1997)

Details

The class `catlg` is a named list of design entries. Each design entry is again a list with the following items:

res resolution, numeric, i.e. 3 denotes resolution III and so forth

nfac number of factors

nruns number of runs

gen column numbers of additional factors in Yates order

WLP word length pattern (starting with words of length 1, i.e. the first two entries are 0 for all designs in `catlg`)

nclear.2fis number of clear 2-factor interactions (i.e. free of aliasing with main effects or other 2-factor interactions)

clear.2fis $2 \times \text{nclear.2fis}$ matrix of clear 2-factor interactions (clear to be understood in the above sense)

all.2fis.clear vector of factors with all 2-factor interactions clear in the above sense

Reference to factors in components `clear.2fis` and `all.2fis.clear` is via their position number (element of $(1:\text{nfac})$).

The `print` function for class `catlg` gives a concise overview of selected designs in any design catalogue of class `catlg`. It is possible to restrict attention to designs with certain run sizes, numbers of factors, and/or to request a minimum resolutions. Designs are ordered in decreasing quality, where the default is aberration order, but number of clear 2fis can be requested alternatively. The best 10 designs are displayed per default; this number can be changed by the `show` option. Options `gen.letters` and `show.alias` influence the style and amount of printed output.

The catalogue `catlg`, which is included with package `FrF2`, is of class `catlg` and currently contains

- the Chen, Sun and Wu (1993) 2-level designs (complete list of 2-level fractional factorials from 4 to 32~runs, complete list of resolution IV 2-level fractional factorials with 64~runs). Note that the Chen Sun Wu paper only shows a selection of the 64~run designs, the complete catalogue has been obtained from Don Sun and is numbered according to minimum aberration (lower number = better design); numbering in the paper is not everywhere in line with this numbering.
- minimum aberration (MA) resolution III designs for 33 to 63 factors in 64 runs. The first few of these have been obtained from Appendix G of Mee 2009, the designs for 38 and more factors have been constructed by combining a duplicated minimum aberration design in 32 runs and the required number of factors with columns 32 to 63 of the Yates matrix for 64 run designs. Using complementary design theory (cf. e.g. Chapter 6.2.2 in Mee 2009), it can be shown that the resulting designs are minimum aberration (because they are complementary to basically the same designs as the designs in 32 runs on which they are based). The author is grateful to Robert Mee for pointing this out.

- a few “good” resolution IV designs in 128 runs, for up to 24 factors obtained by evaluating designs from the complete catalogue by Xu (2009, catalogue on his website), for 25 to 64 factors taken from Block and Mee (2005, with corrigendum 2006). Furthermore, the MA resolution III designs for 65 to 127 factors in 128 runs are also included, with the designs up to 69 factors coming from Appendix G in Mee, whereas the designs for 70 or more factors have been constructed according to the same principle mentioned for the 64 run designs.
- the best (MA) resolution V design for each number of factors in 256 and 512 runs, the best resolution VI design for each number of factors in 1024 runs, the best resolution VII design for each number of factors in 2048 runs and the best resolution VIII design each number of factors in 4096 runs. These are taken from the website of Xu (2009), where complete catalogues of these designs are provided.

It is planned to extend the availability of large designs in catalogue `catlg`, e.g. by adding good resolution IV designs of 256 runs or other “good designs” provided by Mee (2009) and by Xu (2009).

Value

[selects a subset of designs based on `i`, which is again a list of class `catlg`, even if a single element is selected. `res`, `nruns`, `nfac` and `nclear.2fis` return a named vector, the `print` method does not return anything (i.e. it returns `NULL`), and the remaining functions return a list.

Author(s)

Ulrike Groemping

References

- Block, R. and Mee, R. (2005) Resolution IV Designs with 128 Runs *Journal of Quality Technology* **37**, 282-293.
- Block, R. and Mee, R. (2006) Corrigenda *Journal of Quality Technology* **38**, 196.
- Chen, J., Sun, D.X. and Wu, C.F.J. (1993) A catalogue of 2-level and 3-level orthogonal arrays. *Int. Statistical Review* **61**, 131-145.
- Mee, R. (2009). *A Comprehensive Guide to Factorial Two-Level Experimentation*. New York: Springer.
- Sun, D.X., Wu, C.F.J. and Chen, Y.Y. (1997). Optimal blocking schemes for 2^p and 2^{n-p} designs. *Technometrics* **39**, 298-307.
- Xu, H. (2009) Algorithmic Construction of Efficient Fractional Factorial Designs With Large Run Sizes. To appear in *Technometrics*.

See Also

See Also [FrF2](#)

Examples

```

c8 <- catlg[nruns.catlg(catlg)==8]
nclear.2fis.catlg(c8)
clear.2fis.catlg(c8)
all.2fis.clear.catlg(c8)

## usage of print function for inspecting catalogued designs
## the first 10 resolution V+ designs in catalogue catlg
print(catlg, res.min=5)
## the 10 resolution V+ designs in catalogue catlg with the most factors
## (for more than one possible value of nfactores, MaxC2 does usually not make sense)
print(catlg, res.min=5, MaxC2=TRUE)

## designs with 12 factors in 64 runs (minimum resolution IV because
## no resolution III designs of this size are in the catalogue)
## best 10 aberration designs
print(catlg, nfactores=12, nruns=64)
## best 10 clear 2fi designs
print(catlg, nfactores=12, nruns=64, MaxC2=TRUE)
## show alias structure
print(catlg, nfactores=12, nruns=64, MaxC2=TRUE, show.alias=TRUE)
## show best 20 designs
print(catlg, nfactores=12, nruns=64, MaxC2=TRUE, show=20)

## use vector-valued nruns
print(catlg, nfactores=7, nruns=c(16,32))
## all designs (as show=100 is larger than available number of designs)
## with 7 or 8 factors in 16 runs
print(catlg, nfactores=c(7,8), nruns=16, show=100)

```

cubePlot

Cube plot for three-factor-effects

Description

A cube plot for the combined effect of three factors is produced (function cubePlot). Functions cubedraw, cubecorners, cubelabel and myscatterplot3d are not intended for users.

Usage

```

cubePlot(obj, eff1, eff2, eff3, main=paste("Cube plot for", respnam),
         cex.title=1.5, cex.lab=par("cex.lab"), cex.ax=par("cex.axis"),
         cex.clab=1.2, size=0.3, round=NULL,
         abbrev=4, y.margin.add=-0.2, modeled=TRUE)

```

Arguments

obj a vector of response values to be analyzed
OR

	a linear model object with 2-level factors or numerical 2-level variables (CAUTION: numerical x-variable have to be coded as -1 and +1 only!); the structure must be such that effects are either fully aliased or orthogonal, like in a fractional factorial 2-level design
eff1	
eff2	
eff3	effects to be included in the cube plot (x-, y-, z-direction), EITHER vectors of equal length (two-level factors or numerical with the two values -1 and 1) OR variable names of main effects within the <code>obj</code> linear model object (character strings)
main	title for the plot, <code>respnam</code> is the name of the response variable as determined from the call
<code>cex.title</code>	multiplier for size of overall title (<code>cex.main</code> is multiplied with this factor)
<code>cex.ax</code>	size of axis tick marks, defaults to <code>cex.axis</code> -parameter
<code>cex.lab</code>	size of axis labels
<code>cex.clab</code>	size of corner labels
<code>size</code>	size of cube corners
<code>round</code>	optional rounding of corner labels (digits argument for function <code>round</code> , e.g. <code>round=0</code> for integers, <code>round=-1</code> for multiples of 10, <code>round=1</code> for 1 decimal place)
<code>abbrev</code>	number of characters shown for factor levels
<code>y.margin.add</code>	adjustment parameter for placement of y-axis labeling
<code>modeled</code>	TRUE (default: show modeled means; FALSE: show averages) NOTE: Even when showing modeled means, there also appears to be a three-factor-interaction, if the model contains an effect that is aliased with this interaction!

Details

`cubePlot` produces a cube plot of the modeled means or averages of all combinations for three factors. The other functions are internal and are called by `cubePlot`. `myscatterplot3d` is a modified version of `scatterplot3d`, made more suitable for this situation.

Value

`cubePlot` is used for its side effects only.

Author(s)

Ulrike Groemping

References

Box G. E. P, Hunter, W. C. and Hunter, J. S. (2005) *Statistics for Experimenters, 2nd edition*. New York: Wiley.

See Also

[FrF2-package](#) for examples

 DanielPlot

Normal or Half-Normal Effects Plots

Description

The function is modified from the same-name function in package **BsMD** with the purpose of providing more usage comfort (correct effect sizes in case of factors, automatic annotation, automatic labelling of the most significant factors only).

Usage

```
DanielPlot(fit, ...)
## S3 method for class 'design':
DanielPlot(fit, ...)
## Default S3 method:
DanielPlot(fit, code = FALSE, autolab = TRUE, alpha = 0.05, faclab = NULL,
           block = FALSE, datax = TRUE, half = FALSE, pch = "*",
           cex.fac = par("cex.lab"), cex.lab = par("cex.lab"),
           cex.pch = par("cex"), cex.legend = par("cex.lab"),
           main = NULL, ...)
```

Arguments

fit	an experimental design of class <code>design</code> with the <code>type</code> element of the <code>design.info</code> attribute containing “FrF2” or “pb” OR object of class <code>lm</code> . Fitted model from <code>lm</code> or <code>aov</code> .
code	logical. If <code>TRUE</code> labels “A”, “B”, etc. are used instead of the names of the coefficients (factors). A legend linking codes to names is provided.
autolab	If <code>TRUE</code> , only the significant factors according to the Lenth method (significance level given by <code>alpha</code>) are labelled.
alpha	significanc level for the Lenth method
faclab	<code>NULL</code> or list. If <code>NULL</code> , point labels are automatically determined according to the setting of <code>code</code> (i.e. A,B,C etc. for <code>code=TRUE</code> , natural effect names otherwise) and <code>autolab</code> (i.e. all effects are labelled if <code>autolab=FALSE</code> , only significant effects are labelled if <code>autolab=TRUE</code>). Otherwise, <code>faclab</code> can be used for manual labelling of certain effects and should be a list with <code>idx</code> (integer vector referring to position of effects to be labelled) and <code>lab</code> (character vector of labels) components.
block	logical. If <code>TRUE</code> , the first factor is labelled as “BK” (block).
datax	logical. If <code>TRUE</code> , the x-axis is used for the factor effects the the y-axis for the normal scores. The opposite otherwise.

<code>half</code>	logical. If TRUE, half-normal plot of effects is display.
<code>pch</code>	numeric or character. Points character.
<code>cex.fac</code>	numeric. Factor label character size.
<code>cex.lab</code>	numeric. Labels character size.
<code>cex.pch</code>	numeric. Points character size.
<code>cex.legend</code>	numeric. Legend size in case of codes.
<code>main</code>	NULL or character. Title of plot. If NULL, automatic title is generated.
<code>...</code>	further arguments to be passed to the default function, or graphical parameters to be passed to <code>plot</code> .

Details

The design underlying `fit` has to be a (regular or non-regular) fractional factorial 2-level design. Effects (except for the intercept) are displayed in a normal or half-normal plot with the effects in the x-axis by default.

If `fit` is a design with at least one response variable rather than a linear model fit, the `lm`-method for class `design` is applied to it with degree high enough that at least one effect is assigned to each column of the Yates matrix, and the default method for `DanielPlot` is afterwards applied to the resulting linear model.

Value

The function returns an invisible data frame with columns: `x`, `y`, `no` and `effect`, for the coordinates, the position numbers and the effect names for plotted points.

Author(s)

Ernesto Barrios, modified by Ulrike Groemping.

References

- Box G. E. P, Hunter, W. C. and Hunter, J. S. (2005) *Statistics for Experimenters, 2nd edition*. New York: Wiley.
- Daniel, C. (1959) Use of Half Normal Plots in Interpreting Two Level Experiments. *Technometrics* **1**, 311–340.
- Daniel, C. (1976) *Application of Statistics to Industrial Experimentation*. New York: Wiley.
- Lenth, R.V. (1989) Quick and easy analysis of unreplicated factorials. *Technometrics* **31**, 469–473.
- Lenth, R.V. (2006) Lenth's Method for the Analysis of Unreplicated Experiments. To appear in *Encyclopedia of Statistics in Quality and Reliability*, Wiley, New York. Downloadable at http://www.wiley.com/legacy/wileychi/eqr/docs/sample_1.pdf.

See Also

`qqnorm`, `LenthPlot`, `BsMD-package`

estimable.2fis	<i>Statistical and algorithmic aspects of requesting 2-factor interactions to be estimable in FrF2</i>
----------------	--

Description

This help page documents the statistical and algorithmic details of requesting 2-factor interactions to be estimable in FrF2

Details

The option `estimable` allows to specify 2-factor interactions (2fis) that have to be estimable in the model. Per default, it is assumed that a resolution IV model is intended, as it is normally not reasonable to allow main effects to be aliased with other 2-factor interactions in this situation. There are two types of estimability that are distinguished by the setting of option `clear` in function `link{FrF2}`.

Let us first consider the situation of designs of at least resolution IV. With option `clear=TRUE`, `FrF2` searches for a model for which all main effects and all 2fis given in `estimable` are clear of aliasing with any other 2fis. This is a weaker requirement than resolution V, because 2fis outside those specified in `estimable` may be aliased with each other. But it is much stronger than what is done in case of `clear=FALSE`: For the latter, `FrF2` searches for a design that has a distinct column in the model matrix for each main effect and each interaction requested in `estimable`.

Users can explicitly permit that resolution III designs are included in the search of designs for which the specified 2fis are estimable (by the `res3=TRUE` option). In case of `clear=TRUE`, this leads to the somewhat strange situation that main effects can be aliased with 2fis from outside `estimable` while 2fis from inside `estimable` are not aliased with any main effects or 2fis.

With `clear=TRUE`, the algorithm compares the requirement set to catalogued sets of clear 2fis by a graph isomorphism algorithm from R-package **igraph**. The search is quite fast in this case.

With `clear=FALSE`, the algorithm loops through the eligible designs from `catlg.select` from good to worse (in terms of MA) and, for each design, loops through all eligible permutations of the experiment factors from `perms`. If `perms` is omitted, the permutations are looped through in lexicographic order starting from `1:nfac` or `perm.start`. Especially in this case, run times of the search algorithm can be very long. The `max.time` option allows to limit this run time. If the time limit is reached, the final situation (catalogued design and current permutation of experiment factors) is printed so that the user can decide to proceed later with this starting point (indicated by `catlg.select` for the catalogued design(s) to be used and `perm.start` for the current permutation of experiment factors). Note that - according to the structure of the catalogued designs and the lexicographic order of checking permutations - the initial order of the factors has a strong influence on the run time for larger or unlucky problems. For example, consider an experiment in 32~runs and 11~factors, for six of which the pairwise interactions are to be estimable (Example 1 in Wu and Chen 1992). `estimable` for this model can be specified as

```
formula ("~ (F+G+H+J+K+L) ^2")
```

OR

```
formula ("~ (A+B+C+D+E+F) ^2").
```

The former runs a lot faster than the latter (I have not yet seen the latter finish the first catalogued design, if `perms` is not specified). The reason is that the latter needs more permutations of the

experiment factors than the former, since the factors with high positions change place faster and more often than those with low positions.

For this particular design, it is very advisable to constrain the permutations of the experiment factors to the different subset selections of six factors from eleven, since permutations within the sets do not change the possibility of accomodating a design. The required permutations for the second version of this example can be obtained e.g. by the following code:

```
perms.6 <- combn(11,6)
perms.full <- matrix(NA,ncol(perms.6),11)
for (i in 1:ncol(perms.6))
perms.full[i,] <- c(perms.6[,i],setdiff(1:11,perms.6[,i]))
```

Handing perms.full to the procedure using the perms option makes the second version of the requested interaction terms fast as well, since up to almost 40 Mio permutations of experiment factors are reduced to at most 462. Thus, whenever possible, one should try to limit the permutations necessary in case of clear=FALSE.

Please contact me with any suggestions for improvements.

Author(s)

Ulrike Groemping

References

Chen, J., Sun, D.X. and Wu, C.F.J. (1993) A catalogue of 2-level and 3-level orthogonal arrays. *International Statistical Review* **61**, 131-145.

Wu, C.F.J. and Chen, Y. (1992) A graph-aided method for planning two-level experiments when certain interactions are important. *Technometrics* **34**, 162-175.

See Also

See Also [FrF2](#) for regular fractional factorials and [catlg](#) for the Chen, Sun, Wu catalogue of designs and some accessor functions

Examples

```
##### usage of estimable #####
## design with all 2fis of factor A estimable on distinct columns in 16 runs
FrF2(16, nfactors=6, estimable = rbind(rep(1,5),2:6), clear=FALSE)
FrF2(16, nfactors=6, estimable = c("AB", "AC", "AD", "AE", "AF"), clear=FALSE)
FrF2(16, nfactors=6, estimable = formula("~A+B+C+D+E+F+A: (B+C+D+E+F)"),
    clear=FALSE)
    ## formula would also accept self-defined factor names
    ## from factor.names instead of letters A, B, C, ...

## estimable does not need any other input
FrF2(estimable=formula("~(A+B+C)^2+D+E"))

## estimable with factor names
## resolution three must be permitted, as FrF2 first determines that 8 runs
```

```

##      would be sufficient degrees of freedom to estimate all effects
##      and then tries to accomodate the 2fis from the model clear of aliasing in 8 runs
FrF2(estimable=formula("~one+two+three+four+two:three+two:four"),
      factor.names=c("one","two","three","four"), res3=TRUE)
## clear=FALSE allows to allocate all effects on distinct columns in the
##      8 run MA resolution IV design
FrF2(estimable=formula("~one+two+three+four+two:three+two:four"),
      factor.names=c("one","two","three","four"), clear=FALSE)

## 7 factors instead of 6, but no requirements for factor G
FrF2(16, nfactors=7, estimable = formula("~A+B+C+D+E+F+A:(B+C+D+E+F)"),
      clear=FALSE)
## larger design for handling this with all required effects clear
FrF2(32, nfactors=7, estimable = formula("~A+B+C+D+E+F+A:(B+C+D+E+F)"),
      clear=TRUE)
## 16 run design for handling this with required 2fis clear, but main effects aliased
## (does not usually make sense)
FrF2(16, nfactors=7, estimable = formula("~A+B+C+D+E+F+A:(B+C+D+E+F)"),
      clear=TRUE, res3=TRUE)

## example for necessity of perms, and uses of select.catlg and perm.start
## based on Wu and Chen Example 1
## Not run:
## runs per default about max.time=60 seconds, before throwing error with
##      interim results
## results could be used in select.catlg and perm.start for restarting with
##      calculation of further possibilities
FrF2(32, nfactors=11, estimable = formula("~(A+B+C+D+E+F)^2"), clear=FALSE)
## would run for a long long time (I have not yet been patient enough)
FrF2(32, nfactors=11, estimable = formula("~(A+B+C+D+E+F)^2"), clear=FALSE,
      max.time=Inf)

## End(Not run)
## can be easily done with perms,
## as only different subsets of six factors are non-isomorphic
perms.6 <- combn(11,6)
perms.full <- matrix(NA,ncol(perms.6),11)
for (i in 1:ncol(perms.6))
  perms.full[i,] <- c(perms.6[,i],setdiff(1:11,perms.6[,i]))
FrF2(32, nfactors=11, estimable = formula("~(A+B+C+D+E+F)^2"), clear=FALSE,
      perms = perms.full )

```

fold.design

Function to create a foldover for 2-level fractional factorials

Description

This function creates a foldover design for a 2-level fractional factorial. The purpose is to dealias (some) effects. Per default, all factors are folded upon, which makes the resulting design at least resolution IV. Different foldover versions can be requested.

Usage

```
fold.design(design, columns = "full", ...)
```

Arguments

<code>design</code>	a data frame of class <code>design</code> that contains a 2-level fractional factorial; currently, <code>design</code> must neither be blocked nor a long version parameter <code>design</code>
<code>columns</code>	indicates which columns to fold on; the default “full” folds on all columns, i.e. swaps levels for all columns. A specific fold on certain columns can be requested giving a character vector of factor names or a numeric vector of factor positions. See the details section for some statistical comments.
<code>...</code>	currently not used

Details

Foldover is a method to dealias effects in relatively small 2-level fractional factorial designs. The folded design has twice the number of runs from the original design, and an additional column “fold” that distinguishes the original runs from the mirror runs. This column should be used in analyses, since it captures a block effect on time (often the mirror runs are conducted substantially later than the original experiment).

Like most other software, this function conducts a full foldover per default, i.e. the mirror portion reverses the levels of all factors. In terms of the convenient -1/1 notation for factor levels, this can be written as a multiplication with “-1” for the mirror portion of all factors. Thus, all confounding relations involving an odd number of factors (e.g. $A=BC$) are resolved, because the odd side of the equation involves a minus for the mirror runs, and the even side does not (since the minuses cancel each other). (These confounding relations are replaced by even ones for which the odd side of the equation is multiplied with minus the new mirror factor `fold`.)

There are many situations, for which the default full foldover is not the best possible foldover fraction, cf. e.g. Li and Mee (2002). It is therefore possible to choose an arbitrary foldover fraction. For example, folding on one particular factor alone dealias all confounding relations for that factor, folding on two particular factors dealias all confounding relations of these two with others but not of these two together with others and so on.

Folding Plackett-Burman designs also removes the (partial) aliasing with 2-factor interactions for all main effects that are mirrored.

Value

A data frame of class `design` with twice as many rows as `design` and the additional factor `fold` (added as the last factor for folded `pb` designs, as the first factor for `splitplot` designs, and as the last *base* factor for other folded regular fractional factorial designs).

Existing response values are of course preserved, and response values for the new mirror runs are `NA`.

The type in attribute `design.info` is suffixed with “.folded”, and `nruns` (and, if applicable, `nWPs`) is doubled, `nfactors` (and, if applicable, `nfac.WP`) is increased by one (for the factor `fold`, which is a block factor and can also be treated as such, but will currently be treated as a fixed

(whole plot) factor by any automated analysis routine). The creator element receives a list entry for the fold columns.

For regular fractional factorials (design type starting with `FrF2`), the generator element is adjusted (the generators for all generated fold factors now involve the folding factor), and an existing `catlg.entry` element is replaced by a new generators element. The `aliased` element is adapted to the new alias structure. Note that the fold factor enters as a new base factor and therefore is added to the factor matrix after the first $\log_2(\text{nruns})$ factors. This implies that all factor letters previously used for the generated factors are changed - for avoiding confusion it is always recommended to work with factor names that are meaningful in a subject-matter sense.

Furthermore, for the regular fractional factorial designs, the column `run.no.in.std.order` in attribute `run.order` for the mirror portion of the design is populated such that the base factors remain in the conventional order when ordered by `run.no.in.std.order` (regardless whether or not they are included in the fold; it is always possible to reorder runs such that the original base factors together with the folding factor form the new base in standard order).

Note

This function is still somewhat experimental.

Author(s)

Ulrike Groemping

References

Li, H. and Mee, R. (2002). Better foldover fractions for resolution III 2^{k-p} designs. *Technometrics* **44**, 278–283. New York: Springer.

Mee, R. (2009). *A Comprehensive Guide to Factorial Two-Level Experimentation*. New York: Springer.

Montgomery, D.C. (2001). *Design and Analysis of Experiments (5th ed.)*. Wiley, New York.

See Also

See also as [pb](#), [FrF2](#)

Examples

```
## create resolution III design
plan <- FrF2(8,5, factor.names=c("one", "two", "three", "four", "five"))
## add some response data
y <- c(2+desnum(plan)%*%c(2,3,0,0,0) +
      1.5*apply(desnum(plan)[,c(1,2)],1,"prod") + rnorm(8))
## the "c()" makes y into a vector rather than a 1-column matrix
plan <- add.response(plan, y)
DanielPlot(lm(y~(.)^2,plan), alpha=0.2, half=TRUE)
## alias information
design.info(plan)
## full foldover for dealiasing all main effects
plan <- fold.design(plan)
design.info(plan)
```

```

## further data, shifted by -2
y <- c(y, desnum(plan)[9:16,1:5]%*%c(2,3,0,0,0) +
      1.5*apply(desnum(plan)[9:16,c(1,2)],1,"prod") + rnorm(8))
plan <- add.response(plan, y, replace=TRUE)
linmod <- lm(y~(.)^2,plan)
DanielPlot(linmod, alpha=0.2, half=TRUE)
MEPlot(linmod)
IAPlot(linmod)

## fold on factor a only (also removes main effect aliasing here)
plan <- FrF2(8,5, factor.names=c("one","two","three","four","five"))
aliasprint(plan)
plan <- fold.design(plan, columns=1)
aliasprint(plan)

```

FrF2

*Function to provide regular Fractional Factorial 2-level designs***Description**

Regular fractional factorial 2-level designs are provided. Apart from obtaining the usual minimum aberration designs in a fixed number of runs, it is possible to request highest number of free 2-factor interactions instead of minimum aberration or to request the smallest design that fulfills certain requirements (e.g. resolution V with 8 factors).

Usage

```

FrF2(nruns = NULL, nfactors = NULL, factor.names = if (!is.null(nfactors)) {
  if (nfactors <= 50) Letters[1:nfactors] else
    paste("F", 1:nfactors, sep = "")} else NULL,
  default.levels = c(-1, 1), ncenter=0, center.distribute=NULL,
  generators = NULL, design = NULL,
  resolution = NULL, select.catlg=catlg,
  estimable = NULL, clear = TRUE, res3 = FALSE, max.time = 60,
  perm.start=NULL, perms = NULL,
  MaxC2 = FALSE, replications = 1, repeat.only = FALSE,
  randomize = TRUE, seed = NULL, alias.info = 2,
  blocks = 1, block.name = "Blocks", bbreps=replications, wbreps=1,
  alias.block.2fis = FALSE, hard = NULL, check.hard=10, WPs=1,nfac.WP=0,
  WPfacs=NULL, check.WPs = 10, ...)

```

Arguments

nruns Number of runs, must be a power of 2 (4 to 4096), if given.
The number of runs can also be omitted. In that case, if **resolution** is specified, the function looks for the smallest design of the requested resolution that

	<p>accommodates <code>nfactors</code> factors. If the smallest possible design is a full factorial or not catalogued, the function stops with an error.</p> <p>If <code>generators</code> is specified, <code>nruns</code> is required.</p> <p>If <code>estimable</code> is specified and <code>nruns</code> omitted, <code>nruns</code> becomes the size of the smallest design that MIGHT accommodate the effects requested in <code>estimable</code>. If this run size turns out to be too low, an error is thrown. In that case, explicitly choose <code>nruns</code> as twice the run size given in the error message and retry.</p>
<code>nfactors</code>	<p>is the number of 2-level factors to be investigated. It can be omitted, if it is obvious from <code>factor.names</code>, a specific catalogued design given in <code>design</code>, <code>nruns</code> together with <code>generators</code>, or <code>estimable</code>.</p> <p>If <code>estimable</code> is used for determining the number of factors, it is assumed that the largest main effect position number occurring in <code>estimable</code> coincides with <code>nfactors</code>.</p> <p>For blocked designs, block generator columns are not included in <code>nfactors</code>, except if the user explicitly specifies 2-level block generation factors as part of the fractional factorial design (e.g. a factor shift with levels morning and afternoon).</p> <p>For automatically-generated split-plot designs (cf. details section), <code>nfactors</code> simply is the number of all factors (whole plot and split plot together). If <code>nfac.WP < log2(WPs)</code>, the algorithm will add (an) artificial plot generation factor(s).</p> <p>For manually-specified split-plot designs (through options <code>generators</code> or <code>design</code> together with <code>WPfacs</code>), the user must specify at least <code>log2(WPs)</code> split plot factors, i.e. <code>nfac.WP >= log2(WPs)</code> is required (and must, if necessary, be achieved by adding <code>log2(WPs) - nfac.WP</code> extra independent generator columns for the whole plot structure, which have to be counted in <code>nfac.WP</code> and <code>nfactors</code>).</p>
<code>factor.names</code>	<p>a character vector of <code>nfactors</code> factor names or a list with <code>nfactors</code> elements;</p> <p>if the list is named, list names represent factor names, otherwise default factor names are used;</p> <p>the elements of the list are</p> <p>EITHER vectors of length 2 with factor levels for the respective factor</p> <p>OR empty strings. For each factor with an empty string in <code>factor.names</code>, the levels given in <code>default.levels</code> are used;</p> <p>Default factor names are the first elements of the character vector <code>Letters</code>, or the factors position numbers preceded by capital F in case of more than 50 factors.</p>
<code>default.levels</code>	<p>default levels (vector of length 2) for all factors for which no specific levels are given</p>
<code>ncenter</code>	<p>number of center points per block; <code>ncenter > 0</code> is permitted, if all factors are quantitative and the design is not a split-plot design</p>
<code>center.distribute</code>	<p>the number of positions over which the center points are to be distributed for each block; if NULL (default), center points are distributed over end, beginning, and middle (in that order, if there are fewer than three center points) for</p>

	<p>randomized designs, and appended to the end for non-randomized designs. for more detail, see function <code>add.center</code>, which does the work.</p>
<code>generators</code>	<p>There are <code>log2(nruns)</code> base factors the full factorial of which spans the design (e.g. 3 for 8 runs). The generators specify how the remaining factors are to be allocated to interactions of these.</p> <p><code>generators</code> can be</p> <ul style="list-style-type: none"> a list of vectors with position numbers of base factors (e.g. <code>c(1,3,4)</code> stands for the interaction between first, third and fourth base factor) a vector of character representations of these interactions, e.g. “ACD” stands for the same interaction as above a vector of columns numbers in Yates order (e.g. 13 stands for ACD). Note that the columns 1, 2, 4, 8, etc., i.e. all powers of 2, are reserved for the base factors and cannot be used for assigning additional factors, because the design would become a resolution II design. For looking up which column number stands for which interaction, type e.g. <code>names(Yates)[1:15]</code> for a 16 run design. <p>In all cases, preceding the respective entry with a minus sign (e.g. <code>-c(1,3,4)</code>, “-ACD”, <code>-13</code>) implies that the levels of the respective column are reversed.</p> <p>WARNING: Minus signs do not cause an error, but neither have an effect in case of automatic assignment of split-plot designs or hard-to-change columns.</p>
<code>design</code>	<p>is a character string specifying the name of a design listed in the catalogue specified as <code>select.catlg</code>, which is usually the catalogue <code>catlg</code></p>
<code>resolution</code>	<p>is the arabic numeral for the requested resolution of the design. FrF2 looks for a design with at least this resolution. Option <code>resolution</code> does not work, if <code>estimable</code>, <code>blocks</code> or <code>WPs</code> are specified, and neither if <code>nruns</code> is given.</p> <p>A design with resolution III (<code>resolution=3</code>) confounds main effects with 2-factor interactions, a design with resolution IV confounds main effects with three-factor interactions or 2-factor interactions with each other, and designs with resolution V or higher are usually regarded as very strong, because all 2-factor interactions are unconfounded with each other and with main effects.</p>
<code>select.catlg</code>	<p>specifies a catalogue of class <code>catlg</code> from which an adequate design is selected and adapted.</p> <p>The specified catalogue is used for design construction, unless <code>generators</code> explicitly constructs a non-catalogued design. The default <code>catlg</code> is adequate for most applications.</p> <p>If a specific different catalogue of designs is available, this can be specified here. Specification of a smaller subset of designs is useful, if <code>estimable</code> has been given and <code>clear=FALSE</code>, for restricting the search to promising designs.</p>
<code>estimable</code>	<p>indicates the 2-factor interactions (2fis) that are to be estimable in the design. Consult the specific help file (<code>estimable.2fis</code>) for details of two different approaches of requesting estimability, as indicated by the status of the <code>clear</code> option. <code>estimable</code> cannot be specified together with <code>block</code>, <code>splitplot</code>, <code>generators</code> or <code>design</code>.</p> <p><code>estimable</code> can be</p> <ul style="list-style-type: none"> a numeric matrix with two rows, each column of which indicates one interaction, e.g. column 1 3 for interaction of the first with the third factor <p>OR</p>

a character vector containing strings of length 2 with capital letters from `Letters` (cf. package **DoE.base**) for the first 25 factors and small letters for the last 25 (e.g. `c("\ AB\ ", "\ BE\ ")`)

OR

a formula that contains an adequate model formula, e.g.

```
formula ("\ ~A+B+C+D+E+ (F+G+H+J+K+L) ^2\ ")
```

for a model with (at least) eleven factors.

The names of the factors used in the formula can be the same letters usable in the character vector (cf. above, A the first factor, B the second etc.), or they can correspond to the factor names from `factor.names`.

<code>clear</code>	logical, indicating how estimable is to be used. See <code>estimable.2fis</code> .
<code>res3</code>	logical; if TRUE, <code>estimable</code> includes resolution III designs into the search for adequate designs; otherwise resolution IV and higher designs are included only.
<code>max.time</code>	maximum time for design search as requested by <code>estimable</code> , in seconds (default 60); used only if <code>clear=FALSE</code> , since the search can take a long time in complicated or unlucky situations; set <code>max.time</code> to <code>Inf</code> if you want to force a search over an extended period of time; however, be aware that it may still take longer than feasible (cf. also <code>estimable.2fis</code>)
<code>perm.start</code>	used only if <code>clear=FALSE</code> . Provides a start permutation for permuting experiment factors (numeric vector). This is useful for the case that a previous search was not (yet) successful because of a time limit, since the algorithm notifies the user about the permutation at which it had to stop.
<code>perms</code>	used only if <code>clear=FALSE</code> . Provides the matrix of permutations of experiment factors to be tried; each row is a permutation. For example, for an 11-factor design with the first six factors and their 2fis estimable, it is only relevant, which of the eleven factors are to be allocated to the first six experiment factors, and these as well as the other five factors can be in arbitrary order. This reduces the number of required permutations from about 40 Mio to 462. It is recommended to use <code>perms</code> whenever possible, if <code>clear=FALSE</code> , since this dramatically improves performance of the algorithm. It is planned to automatically generate <code>perms</code> for certain structures like compromise designs in the (not so near) future.
<code>MaxC2</code>	is a logical and defaults to FALSE. If TRUE, maximizing the number of free 2-factor interactions takes precedence over minimizing aberration. Resolution is always considered first. Most likely, features like this are going to change in the future.
<code>replications</code>	positive integer number. Default 1 (i.e. each row just once). If larger, each design run is executed replication times. If <code>repeat.only</code> , repeated measurements are carried out directly in sequence, i.e. no true replication takes place, and all the repeat runs are conducted together. It is likely that the error variation generated by such a procedure will be too small, so that average values should be analyzed for an unreplicated design. Otherwise (default), the full experiment is first carried out once, then for the second replication and so forth. In case of randomization, each such blocks is

randomized separately. In this case, replication variance is more likely suitable for usage as error variance (unless e.g. the same parts are used for replication runs although build variation is important).

<code>repeat.only</code>	logical, relevant only if <code>replications > 1</code> . If TRUE, replications of each run are grouped together (repeated measurement rather than true replication). The default is <code>repeat.only=FALSE</code> , i.e. the complete experiment is conducted in <code>replications</code> blocks, and each run occurs in each block.
<code>randomize</code>	logical. If TRUE, the design is randomized. This is the default. In case of replications, the nature of randomization depends on the setting of option <code>repeat.only</code> .
<code>seed</code>	optional seed for the randomization process
<code>alias.info</code>	can be 2 or 3, gives the order of interaction effects for which alias information is to be included in the <code>aliased</code> component of the <code>design.info</code> element of the output object.
<code>blocks</code>	is EITHER the number of blocks into which the experiment is subdivided OR a character vector of names of independent factors that are used as block constructors OR a vector or list of generators similar to <code>generators</code> . In the latter case, the differences to <code>generators</code> are

- that numbers/letters refer to the factors of the experiment and not to column numbers of the Yates matrix
- that numbers/letters can refer to `*all* nfact` factors rather than the `log2(nruns)` base factors only,
- that one single number is always interpreted as the number of blocks rather than a column reference,
- that individual numbers are allowed in a list (i.e. individual factors specified in the experiment can be used as block factors) and
- that no negative signs are allowed.

If `blocks` is a single number, it must be a power of 2. A blocked design can have at most `nruns-blocks` treatment factors, but should usually have fewer than that.

If the experiment is randomized, randomization happens within blocks.

For the statistical and algorithmic background of blocked designs, see [block](#).

<code>block.name</code>	name of the block factor, default “Blocks”
<code>bbreps</code>	between block replications; these are always taken as genuine replications, not repeat runs; default: equal to <code>replications</code> ; CAUTION: you should not modify <code>bbreps</code> if you do not work with blocks, because the program code uses it instead of <code>replications</code> in some places
<code>wbreps</code>	within block replications; whether or not these are taken as genuine replications depends on the setting of <code>repeat.only</code>

<code>alias.block.2fis</code>	<p>logical indicating whether blocks may be aliased with 2fis (default: FALSE); the option is effective only for automatic block assignment (cf. <code>block</code>); it will often be necessary to modify this option, because there is otherwise no solution.</p> <p>CAUTION: If <code>alias.block.2fis=TRUE</code>, no effort is made to avoid aliasing of 2fis with block main effects, even if complete de-aliasing were possible.</p>
<code>hard</code>	<p>gives the number of hard to change factors. These must be the first factors in <code>factor.names</code>. Implementation is via a non-randomized split-plot design with as few as possible whole plots (number of possible whole plots is determined via left-adjustment) and as few as possible non-hard factors within the whole plot structure (by applying split-plot after left-adjustment). If feasible, it is recommended to rather use a randomized split-plot experiment (cf. <code>splitplot</code> for a brief discussion of the difference) by explicitly treating the hard to change factors as whole-plot factors via <code>WPs</code> and <code>nfac.WP</code>.</p>
<code>check.hard</code>	<p>is the number of candidate designs from the catalogue specified in <code>select.catlg</code> that are checked for making hard-to-change factors change as little as possible. The default is 10 - if too many changes are needed, a larger choice might help find a design with fewer level changes (but will also take longer run time and will find a worse design in terms of resolution / aberration).</p> <p>If you want to use the best design and do not want to compromise the confounding structure for ease-of-change reasons, set <code>check.hard</code> to 1.</p>
<code>WPs</code>	<p>is the number of whole plots and must be a power of 2.</p> <p>If <code>WPs > 1</code>, at least one of <code>nfac.WP</code> or <code>WPfacs</code> must be given.</p> <p>If <code>WPs = 1</code>, all settings for split-plot related options are ignored.</p> <p>For statistical and algorithmic information on treatment of split-plot designs see the separate help file <code>splitplot</code>.</p>
<code>nfac.WP</code>	<p>is the number of whole plot factors and must be smaller than <code>WPs</code>.</p> <p>The <code>nfac.WP</code> whole plot factors are counted within <code>nfactors</code>. Per default, the first <code>nfac.WP</code> factors are the whole plot factors.</p> <p>If a design is provided and whole plot factors are manually provided (design or <code>generators</code> option together with <code>WPfacs</code>), <code>nfac.WP</code> can be omitted (i.e. remains 0). If given, it must coincide with the length of <code>WPfacs</code>.</p> <p>If <code>nfac.WP</code> is omitted (i.e. remains 0) and <code>WPs > 1</code>, an error is thrown, because the situation is a block rather than a split-plot situation (and either it was forgotten to specify the number of whole plot factors, or blocks should be specified).</p>
<code>WPfacs</code>	<p>is per default NULL. In this case, the first <code>nfac.WP</code> factors are considered whole plot factors (and are, if necessary, automatically supplemented by additional whole plot constructor factors).</p> <p>If <code>WPfacs</code> is given, it must specify at least $\log_2(WPs)$ whole plot factors. A custom design must be specified with options <code>design</code> or <code>generators</code>, and the requested factors in <code>WPfacs</code> must indeed create a split-plot structure with <code>WPs</code> whole plots. If the number of whole plots created by the whole plot factors differs from <code>WPs</code> or factors other than the specified factors from <code>WPfacs</code> would in fact become whole plot factors as well, an error is thrown.</p>

	<p><code>WPfacS</code> can be any of a vector or list of factor position numbers OR a character vector of factor position letters from <code>Letters</code> OR a character vector with entries “F” followed by factor position number OR a character vector of factor names (this takes precedence over factor letters, i.e. if the factor names were B, A, C, D, and E, factor letter entries in the <code>nfac.WP</code> are interpreted as factor names, not position letters). It is not possible to specify additional whole plot generators from interaction effects manually through <code>WPfacS</code>. Rather, all whole plot factors - even artificial ones needed only to increase the number of plots - need to be included in the design factors.</p>
<code>check.WPs</code>	is the number of potential split-plot designs that are compared by function <code>splitpick</code> w.r.t. resolution of the whole plot portion of the design. This option is effective, if <code>nfac.WP > k.WP</code> (i.e. bad resolution possible) and <code>nfac.WP</code> not larger than half the number of plots (i.e. resolution better than III is possible). The default is 10 - if not satisfied with the structure of the whole plot factors, a larger choice might help find a better design (but also take longer run time).
<code>...</code>	currently not used

Details

Per default, the function works on the basis of automatically-selected catalogued designs. The default catalogue used is `catlg` (a list object of class `catlg`).

Alternatively, the user can explicitly specify a design through accessing a specific catalogued design using the `design` option or specifying non-catalogued generators via the `generators` option.

Apart from generation of simple fractional factorial designs based on catalogued or non-catalogued generators, function `FrF2` allows specification of blocked designs and split-plot designs, as well as specification of a set of `2fis` that are required to be estimable. The implementation of these possibilities is explained in the separate help files `block`, `splitplot` and `estimable.2fis`. If you consider to use option `hard`, it may also be worth while to look at the `splitplot` help file.

Function `FrF2` is still under development, although most features are now included, and the principle structure of inputs and outputs should not change much any more. Please contact me with any suggestions for improvements.

Value

Value is a data frame of S3 class `design` and has attached attributes that can be accessed by functions `desnum`, `run.order` and `design.info`.

The data frame itself contains the design with levels coded as requested. If no center points have been requested, the design columns are factors with contrasts -1 and $+1$ (cf. also `contr.FrF2`); in case of center points, the design columns are numeric.

The following attributes are attached to it:

<code>desnum</code>	Design matrix in -1/1 coding
<code>run.order</code>	three column data frame, first column contains the run number in standard order, second column the run number as randomized, third column the run number with replication number as postfix (in blocked designs, there may be two postfixes for within and between block replications); useful for switching back and forth between actual and standard run number
<code>design.info</code>	list with the entries <ul style="list-style-type: none"> type character string “full factorial”, “FrF2”, “FrF2.estimable”, “FrF2.generators”, “FrF2.blocked” or “FrF2.splitplot” depending on the type of design nruns number of runs (replications are not counted) nfactors number of factors; since version 0.97, this is also true for designs of type <code>FrF2.blocked</code> (<code>nfactors</code> is now equal to <code>ntreat</code>) and for designs of type <code>FrF2.splitplot</code>, where <code>nfactors</code> is now the sum of <code>nfac.WP</code> and <code>nfac.SP</code>. ntreat for designs of type <code>FrF2.blocked</code> only; number of treatment factors nfac.WP for designs of type <code>FrF2.splitplot</code> only; number of whole plot factors (including extra factors that may have been added for whole plot construction); these are the first factors in the design data frame nfac.SP for designs of type <code>FrF2.splitplot</code> only; number of split-plot factors nlevels for designs of type <code>full factorial</code> only; vector with number of levels for each factor (of course, all the <code>nfactors</code> entries are “2” for <code>FrF2</code>) factor.names list named with (treatment) factor names and containing as entries vectors of length two each with coded factor levels nblocks for designs of type <code>FrF2.blocked</code> only; number of blocks block.gen vector of columns of the Yates matrix for generating block factors in case of automatic block generation OR list of (vectors of) factor numbers in case the <code>blocks</code> argument specifies certain factor combinations for blocking; ; for designs of type <code>FrF2.blocked</code> only blocksize for designs of type <code>FrF2.blocked</code> only; size of each block (without consideration of <code>wbreps</code>) nWPs for designs of type <code>FrF2.splitplot</code> only; number of whole plots plotsize for designs of type <code>FrF2.splitplot</code> only; size of each plot (without consideration of <code>repeat.only</code> replications if applicable) catlg.entry for designs of type <code>FrF2</code> only; list with one element, which is the entry of <code>catlg</code> on which the design is based gen.display for designs of type <code>FrF2.generators</code> only; character vector of generators in the form <code>D=ABC</code> etc.

- base.design** for designs of type `FrF2.blocked` or `FrF2.splitplot` only; gives a character string that contains the name of the base design in the catalogue or the column numbers of generating columns in Yates matrix; in case of automatic block generation, the exclusion or inclusion of `k.block` in the number of design factors / generators indicates whether the design was generated using function `blockpick` or `blockpick.big`.
- aliased.with.blocks** for designs of type `FrF2.blocked` only; treatment effects that are aliased with block main effects, up to 2fis or 3fis, depending on the choice of `alias.info`
- aliased** alias structure of main effects, 2fis and possibly 3fis, depending on the choice of `alias.info`; For non-blocked and non-split-plot designs, `aliased` is itself a list of the two or three components main, fi2, and optionally fi3, given in terms of factor letters from `Letters` (up to 50~factors) or F1, F2, and so forth (more than 50~factors). For blocked and split-plot designs, `aliased` is a single list with an entry for each column of the Yates matrix that accomodates aliased low-order effects, and entries are in terms of factor names.)
- replication** option setting in call to `FrF2`
- repeat.only** option setting in call to `FrF2`
- bbreps** for designs of type `FrF2.blocked` only; number of between block replications
- wbreps** for designs of type `FrF2.blocked` only; number of within block replications;
`repeat.only` indicates whether these are replications or repetitions only
- map** the mapping relation between factors in the base design and experimental factors, after using option `estimable`
- clear** option setting in call to `FrF2`, in case of `estimable`
- res3** option setting in call to `FrF2`, in case of `estimable`
- randomize** option setting in call to `FrF2`
- seed** option setting in call to `FrF2`
- creator** call to function `FrF2` (or stored menu settings, if the function has been called via the R commander plugin **RcmdrPlugin.DoE**)
- ncube** number of cube points per block, in case center points have been requested
- ncenter** number of center points per block, in case center points have been requested

Author(s)

Ulrike Groemping

References

- Bingham, D.R., Schoen, E.D. and Sitter, R.R. (2004). Designing Fractional Factorial Split-Plot Experiments with Few Whole-Plot Factors. *Applied Statistics* **53**, 325-339.
- Bingham, D. and Sitter, R.R. (2003). Fractional Factorial Split-Plot Designs for Robust Parameter Experiments. *Technometrics* **45**, 80-89.

- Bisgaard, S. (1994a). Blocking generators for small 2^{k-p} designs. *J. Quality Technology* **26**, 288-294.
- Chen, J., Sun, D.X. and Wu, C.F.J. (1993) A catalogue of 2-level and 3-level orthogonal arrays. *International Statistical Review* **61**, 131-145.
- Cheng, C.-S., Martin, R.J., and Tang, B. (1998). Two-level factorial designs with extreme numbers of level changes. *Annals of Statistics* **26**, 1522-1539.
- Cheng, C.-S. and Tsai, P.-W. (2009). Optimal two-level regular fractional factorial block and split-plot designs. *Biometrika* **96**, 83-93.
- Cheng, S.W. and Wu, C.F.J. (2002). Choice of optimal blocking schemes in 2-level and 3-level designs. *Technometrics* **44**, 269-277.
- Huang, P., Chen, D. and Voelkel, J.O. (1998). Minimum-Aberration Two-Level Split-Plot Designs. *Technometrics* **40**, 314-326.
- Mee, R. (2009). *A Comprehensive Guide to Factorial Two-Level Experimentation*. New York: Springer.
- Sun, D.X., Wu, C.F.J. and Chen, Y.Y. (1997). Optimal blocking schemes for 2^p and 2^{n-p} designs. *Technometrics* **39**, 298-307.
- Wu, C.F.J. and Chen, Y. (1992) A graph-aided method for planning two-level experiments when certain interactions are important. *Technometrics* **34**, 162-175.

See Also

See Also [pb](#) for non-regular fractional factorials according to Plackett-Burman, [catlg](#) for the Chen, Sun, Wu catalogue and some accessor functions, and [block](#), [splitplot](#) or [estimable.2fis](#) for statistical and algorithmic information on the respective topic.

Examples

```
## maximum resolution minimum aberration design with 4 factors in 8 runs
FrF2(8,4)
## the design with changed default level codes
FrF2(8,4, default.level=c("current","new"))
## the design with number of factors specified via factor names
## (standard level codes)
FrF2(8, factor.names=list(temp="",press="",material="",state=""))
## the design with changed factor names and factor-specific level codes
FrF2(8,4, factor.names=list(temp=c("min","max"),press=c("low","normal"),
  material=c("current","new"),state=c("new","aged")))
## a full factorial
FrF2(8,3, factor.names=list(temp=c("min","max"),press=c("low","normal"),
  material=c("current","new")))
## a replicated full factorial (implicit by low number of factors)
FrF2(16,3, factor.names=list(temp=c("min","max"),press=c("low","normal"),
  material=c("current","new")))
## three ways for custom specification of the same design
FrF2(8, generators = "ABC")
FrF2(8, generators = 7)
FrF2(8, generators = list(c(1,2,3)))
## more than one generator
```

```

FrF2(8, generators = c("ABC","BC"))
FrF2(8, generators = c(7,6))
FrF2(8, generators = list(c(1,2,3),c(2,3)))
## alias structure for three generators that differ only by sign
design.info(FrF2(16,generators=c(7,13,15),randomize=FALSE))$aliased
design.info(FrF2(16,generators=c(7,-13,15),randomize=FALSE))$aliased
design.info(FrF2(16,generators=c(-7,-13,-15),randomize=FALSE))$aliased
## finding smallest design with resolution 5 in 7 factors
FrF2(nfactors=7, resolution=5)
## same design, but with 12 center points in 6 positions
FrF2(nfactors=7, resolution=5, ncenter=12, center.distribute=6)

## maximum resolution minimum aberration design with 9 factors in 32 runs
## show design information instead of design itself
design.info(FrF2(32,9))
## maximum number of free 2-factor interactions instead of minimum aberration
## show design information instead of design itself
design.info(FrF2(32,9,MaxC2=TRUE))

## usage of replication
## shows run order instead of design itself
run.order(FrF2(8,4,replication=2,randomize=FALSE))
run.order(FrF2(8,4,replication=2,repeat.only=TRUE,randomize=FALSE))
run.order(FrF2(8,4,replication=2))
run.order(FrF2(8,4,replication=2,repeat.only=TRUE))

## Not run:
## examples below do work, but are repeated in the
## respective method's separate help file and are therefore prevented
## from running twice

##### automatic blocked designs #####
## from a full factorial ##
FrF2(8,3,blocks=2)
## with replication
run.order(FrF2(8,3,blocks=2,wbreps=2))
run.order(FrF2(8,3,blocks=2,wbreps=2,repeat.only=TRUE))
run.order(FrF2(8,3,blocks=2,bbreps=2))
run.order(FrF2(8,3,blocks=2,bbreps=2,wbreps=2))

## automatic blocked design with fractions
FrF2(16,7,blocks=4,alias.block.2fis=TRUE,factor.names=c("MotorSpeed", "FeedMode", "FeedSizing",
"Gain", "ScreenAngle", "ScreenVibLevel"))
## isomorphic non-catalogued design as basis
FrF2(16,gen=c(7,11,14),blocks=4,alias.block.2fis=TRUE)
## FrF2 uses blockpick.big and ignores the generator
FrF2(64,gen=c(7,11,14),blocks=16,alias.block.2fis=TRUE)

##### manual blocked design #####
### example that shows why order of blocks is not randomized
### can of course be randomized by user, if appropriate

```

```

FrF2(32,9,blocks=c("Day","Shift"),alias.block.2fis=TRUE,
    factor.names=list(Day=c("Wednesday","Thursday"), Shift=c("Morning","Afternoon"),
        F1="",F2="",F3="",F4="",F5="",F6="",F7=""), default.levels=c("current","new"))

##### hard to change factors #####
## example from Bingham and Sitter Technometrics 19999
## MotorSpeed, FeedMode,FeedSizing,MaterialType are hard to change
BS.ex <- FrF2(16,7,hard=4,
    factor.names=c("MotorSpeed", "FeedMode","FeedSizing","MaterialType",
        "Gain","ScreenAngle","ScreenVibLevel"),
    default.levels=c("-", "+"),randomize=FALSE)
design.info(BS.ex)
BS.ex
## NOTE: the design has 8 whole plots.
## If randomize=FALSE is used like here, the first hard-to-change factors
## do not always change between whole plots.
## A conscious and honest decision is required whether this is
## acceptable for the situation at hand!
## randomize=TRUE would cause more changes in the first four factors.

##### automatic generation for split plot #####
## 3 control factors, 5 noise factors, control factors are whole plot factors
## 8 plots desired in a total of 32 runs
## Bingham Sitter 2003
BS.ex2a <- FrF2(32, 8, WPs=8, nfac.WP=3,
    factor.names=c(paste("C",1:3,sep=""), paste("N",1:5,sep="")),randomize=TRUE)

## manual generation of this same design
BS.ex2m <- FrF2(32, 8, generators=c("ABD","ACD","BCDE"),WPs=8, WPfacs=c("C1","C2","C3"), nfac
    factor.names=c(paste("C",1:3,sep=""),paste("N",1:5,sep="")),randomize=TRUE)

## design with few whole plot factors
## 2 whole plot factors, 7 split plot factors
## 8 whole plots, i.e. one extra WP factor needed
BSS.cheese.exa <- FrF2(32, 9, WPs=8, nfac.WP=2,
    factor.names=c("A","B","p","q","r","s","t","u","v"))
design.info(BSS.cheese.exa)
## manual generation of the design used by Bingham, Schoen and Sitter
## note that the generators include a generator for the 10th spplitting factor
## s= ABq, t = Apq, u = ABpr and v = Aqr, splitting factor rho=Apqr
BSS.cheese.exm <- FrF2(32, gen=list(c(1,2,4),c(1,3,4),c(1,2,3,5),c(1,4,5),c(1,3,4,5)),
    WPs=8, nfac.WP=3, WPfacs=c(1,2,10),
    factor.names=c("A","B","p","q","r","s","t","u","v","rho"))
design.info(BSS.cheese.exm)

##### usage of estimable #####
## design with all 2fis of factor A estimable on distinct columns in 16 runs
FrF2(16, nfactors=6, estimable = rbind(rep(1,5),2:6), clear=FALSE)
FrF2(16, nfactors=6, estimable = c("AB","AC","AD","AE","AF"), clear=FALSE)
FrF2(16, nfactors=6, estimable = formula("~A+B+C+D+E+F+A:(B+C+D+E+F)"),
    clear=FALSE)
## formula would also accept self-defined factor names
## from factor.names instead of letters A, B, C, ...

```

```

## estimable does not need any other input
FrF2(estimable=formula("~(A+B+C)^2+D+E"))

## estimable with factor names
## resolution three must be permitted, as FrF2 first determines that 8 runs
## would be sufficient degrees of freedom to estimate all effects
## and then tries to accomodate the 2fis from the model clear of aliasing in 8 runs
FrF2(estimable=formula("~one+two+three+four+two:three+two:four"),
      factor.names=c("one","two","three","four"), res3=TRUE)
## clear=FALSE allows to allocate all effects on distinct columns in the
## 8 run MA resolution IV design
FrF2(estimable=formula("~one+two+three+four+two:three+two:four"),
      factor.names=c("one","two","three","four"), clear=FALSE)

## 7 factors instead of 6, but no requirements for factor G
FrF2(16, nfactors=7, estimable = formula("~A+B+C+D+E+F+A:(B+C+D+E+F)"),
      clear=FALSE)
## larger design for handling this with all required effects clear
FrF2(32, nfactors=7, estimable = formula("~A+B+C+D+E+F+A:(B+C+D+E+F)"),
      clear=TRUE)
## 16 run design for handling this with required 2fis clear, but main effects aliased
## (does not usually make sense)
FrF2(16, nfactors=7, estimable = formula("~A+B+C+D+E+F+A:(B+C+D+E+F)"),
      clear=TRUE, res3=TRUE)

## example for necessity of perms, and uses of select.catlg and perm.start
## based on Wu and Chen Example 1
\dontrun{
  ## runs per default about max.time=60 seconds, before throwing error with
  ## interim results
  ## results could be used in select.catlg and perm.start for restarting with
  ## calculation of further possibilities
  FrF2(32, nfactors=11, estimable = formula("~(A+B+C+D+E+F)^2"), clear=FALSE)
  ## would run for a long long time (I have not yet been patient enough)
  FrF2(32, nfactors=11, estimable = formula("~(A+B+C+D+E+F)^2"), clear=FALSE,
      max.time=Inf)
}
## can be easily done with perms,
## as only different subsets of six factors are non-isomorphic
perms.6 <- combn(11,6)
perms.full <- matrix(NA,ncol(perms.6),11)
for (i in 1:ncol(perms.6))
  perms.full[i,] <- c(perms.6[,i],setdiff(1:11,perms.6[,i]))
FrF2(32, nfactors=11, estimable = formula("~(A+B+C+D+E+F)^2"), clear=FALSE,
    perms = perms.full )

## End(Not run)

```

Description

Main effects plots and interaction plots are produced. The other documented functions are not intended for users.

Usage

```
MEPlot(obj, ...)
## S3 method for class 'design':
MEPlot(obj, ...)
## Default S3 method:
MEPlot(obj, main = paste("Main effects plot for", respnam),
       pch = 15, cex.xax = par("cex.axis"), cex.yax = cex.xax, mgp.ylab = 4,
       cex.title = 1.5, cex.main = par("cex.main"),
       lwd = par("lwd"), abbrev = 3, select = NULL, ...)

IAPlot(obj, ...)
## S3 method for class 'design':
IAPlot(obj, ...)
## Default S3 method:
IAPlot(obj, main = paste("Interaction plot matrix for", respnam),
       pch = c(15, 17), cex.lab = par("cex.lab"), cex = par("cex"),
       cex.xax = par("cex.axis"), cex.yax = cex.xax, cex.title = 1.5,
       lwd = par("lwd"), abbrev = 4, select = NULL, show.alias = FALSE, ...)

intfind(i, j, mat)

check(obj)

remodel(obj)
```

Arguments

<code>obj</code>	an experimental design of class <code>design</code> with the <code>type</code> element of the <code>design.info</code> attribute containing "FrF2" or "pb" OR a linear model object with 2-level factors or numerical 2-level variables; the structure must be such that effects are either fully aliased or orthogonal, like in a regular fractional factorial 2-level design; note that <code>IAPlot</code> currently requires the response in <code>obj</code> to be a pre-defined variable and not a calculated quantity
<code>...</code>	further arguments to be passed to the default function; ...in the default method are not used, they have been added because of formal requirements only
<code>main</code>	overall title for the plot assembly
<code>pch</code>	Plot symbol number <code>MEPlot</code> , or vector of two plot symbol numbers for the lower and higher level of the trace factor <code>iap</code>

<code>cex.xax</code>	size of x-axis annotation, defaults to <code>cex.axis</code> -parameter
<code>cex.yax</code>	size of y-axis annotation, defaults to <code>cex.xax</code>
<code>mgp.ylab</code>	horizontal placement of label of vertical axis in <code>MEPlot</code>
<code>cex.title</code>	multiplier for size of overall title (<code>cex.main</code> is multiplied with this factor)
<code>cex.main</code>	size of individual plot titles in <code>MEPlot</code>
<code>cex.lab</code>	Size of variable names in diagonal panels of interaction plots produced by <code>IAPlot</code> .
<code>cex</code>	size of plot symbols in interaction plots
<code>lwd</code>	line width for plot lines and axes
<code>abbrev</code>	number of characters shown for factor levels
<code>select</code>	vector with position numbers of the main effects to be displayed; default: all main effects; the default implies the full interaction plot matrix for <code>IAPlot</code> . For <code>IAPlot</code> , the full interaction plot matrix for the selected factors is displayed. Of course, at least two factors must be selected. Furthermore, the linear model <code>obj</code> must at least contain one interaction term among the selected variables. For interactions that do not occur in the linear model, not plot is shown. An interaction plot matrix of data means can be obtained by specifying the model with all possible 2-factor interactions (e.g. formula <code>y ~ (.)^2</code> for a regular 2-level fractional factorial, for which <code>y</code> is the only response and all other variables are 2-level factors).
<code>show.alias</code>	if TRUE, the interaction plot shows the number of the list entry from <code>aliases(obj)\\$aliases</code> (cf. <code>aliases</code>) in order to support immediate diagnosis of which depicted interaction may be due to other than the shown effect because of aliasing
<code>i</code>	integer, for internal use only
<code>j</code>	integer, for internal use only
<code>mat</code>	matrix, for internal use only

Details

For functions `MEPlot` or `IAPlot`, if `obj` is a design with at least one response variable rather than a linear model fit, the `lm`-method for class `design` is applied to it with the required degree (1 or 2), and the default method for the respective function is afterwards applied to the resulting linear model.

MEPlot produces plots of all main effects in the model, or selected ones if `select` is specified

IAPlot produces plots of all interaction effects in the model, or selected ones if `select` is specified

intfind is an internal function not directly useful for users

check is an internal function for checking whether the model complies with assumptions (fractional factorial of 2-level factors with full or no aliasing, not partial aliasing; this implies that Plackett-Burman designs with partial aliasing of 2-factor interactions give an OK (=TRUE) in `check` for pure main effects models only.)

remodel is an internal function that redoes factor values into -1 and 1 coding, regardless of the contrasts that have been used for the original factors; numerical data are transformed by subtracting the mean and dividing by half the range (max-min), which also transforms them to -1 and 1 coding in the 2-level case (and leads to an error otherwise)

Value

MEPlot and IAPlot are used for their side effects only.

The internal function `check` is used within other functions for checking whether the model is a fractional factorial with 2-level factors and no partial aliasing, as requested for the package to work. It is applied to remodeled objects only and returns a logical. If the returned value is `FALSE`, the calling function fails.

The internal function `intfind` returns an integer (length 1 or 0). It is not useful for users.

The internal function `remodel` is applied to a linear model object and returns a list of two components:

<code>model</code>	is the redone model with x-variables recoded to numeric -1 and 1 notation and aov objects made into “pure” lm objects
<code>labs</code>	is a list preserving the level information from original factors (levels are minus and plus for numerical variables)

Author(s)

Ulrike Groemping

References

Box G. E. P, Hunter, W. C. and Hunter, J. S. (2005) *Statistics for Experimenters, 2nd edition*. New York: Wiley.

See Also

[FrF2-package](#) for examples

pb	<i>Function to generate non-regular fractional factorial screening designs</i>
----	--

Description

The function generates Plackett-Burman designs and in some cases other screening designs in run numbers that are a multiple of 4. These designs are particularly suitable for screening a large number of factors, since interactions are not fully aliased with one main effect each but partially aliased. (The designs in 8 and 64 runs are exceptions from this rule.)

Usage

```
pb(nruns, nfactors = nruns - 1, factor.names = if (nfactors <= 50)
  Letters[1:nfactors] else paste("F", 1:nfactors, sep = ""),
  default.levels = c(-1, 1), ncenter=0, center.distribute=NULL,
  boxtysedal = TRUE, n12.taguchi = FALSE,
  replications = 1, repeat.only = FALSE,
```

```
randomize = TRUE, seed = NULL, ...)
```

```
pb.list
```

Arguments

<code>nruns</code>	number of runs, must be a multiple of 4
<code>nfactors</code>	number of factors, default is <code>nruns - 1</code> , and it is recommended to retain this default. It is possible to specify factor names for fewer factors, and the remaining columns will be named <code>e1</code> , <code>e2</code> , ... They are useful for representing error in effects plots.
<code>factor.names</code>	a character vector of factor names (length up to <code>nfactors</code>) or a list with <code>nfactors</code> elements; if the list is named, list names represent factor names, otherwise default factor names are used; the elements of the list are EITHER vectors of length 2 with factor levels for the respective factor OR empty strings. For each factor with an empty string in <code>factor.names</code> , the levels given in <code>default.levels</code> are used; Default factor names are the first elements of the character vector <code>Letters</code> , or the factors position numbers preceded by capital F in case of more than 50 factors.
<code>default.levels</code>	default levels (vector of length 2) for all factors for which no specific levels are given
<code>ncenter</code>	number of center points; <code>ncenter > 0</code> is permitted, if all factors are quantitative
<code>center.distribute</code>	the number of positions over which the center points are to be distributed ; if NULL (default), center points are distributed over end, beginning, and middle (in that order, if there are fewer than three center points) for randomized designs, and appended to the end for non-randomized designs. for more detail, see function <code>add.center</code> , which does the work.
<code>boxtyssedal</code>	logical, relevant only for <code>nruns=16</code> . If FALSE, the geometric (=standard) 16 run plan is used. If TRUE, the proposal by Box and Tyssedal is used instead, which has the advantage (for screening) of aliasing each interaction with several main effects, like the other Plackett-Burman designs.
<code>n12.taguchi</code>	logical, relevant only for <code>nruns=12</code> . If TRUE, the 12 run design is given in Taguchi order.
<code>replications</code>	positive integer number. Default 1 (i.e. each row just once). If larger, each design run is executed replication times. If <code>repeat.only</code> , repeated measurements are carried out directly in sequence, i.e. no true replication takes place, and all the repeat runs are conducted together. It is likely that the error variation generated by such a procedure will be too small, so that average values should be analyzed for an unreplicated design. Otherwise (default), the full experiment is first carried out once, then for the second replication and so forth. In case of randomization, each such blocks is

	randomized separately. In this case, replication variance is more likely suitable for usage as error variance (unless e.g. the same parts are used for replication runs although build variation is important).
<code>repeat.only</code>	logical, relevant only if replications > 1. If TRUE, replications of each run are grouped together (repeated measurement rather than true replication). The default is <code>repeat.only=FALSE</code> , i.e. the complete experiment is conducted in <code>replications</code> blocks, and each run occurs in each block.
<code>randomize</code>	logical. If TRUE, the design is randomized. This is the default.
<code>seed</code>	optional seed for the randomization process
<code>...</code>	currently not used

Details

`pb` stands for Plackett-Burman. Plackett-Burman designs (Plackett and Burman 1946) are generally used for screening many variables in relatively few runs, when interest is in main effects only, at least initially. Different from the regular fractional factorial designs created by function `FrF2`, they do not perfectly confound interaction terms with main effects but distribute interaction effects over several main effects. The designs with number of runs a power of 2 are an exception to this rule: they are just the resolution III regular fractional factorial designs and are as such not very suitable for screening because of a high risk of very biased estimates for the main effects of the factors. Where possible, these are therefore replaced by different designs (cf. below).

For most run numbers, function `pb` uses Plackett-Burman designs, and simply fills columns from left to right. The generating rows for these designs can be found in the list `pb.list` (a 0 entry indicates that the design is constructed by a different method, e.g. doubling).

For 12 runs, the isomorphic design by Taguchi can be requested. For 16 runs, the default is to use the designs suggested by Box and Tyssedal (2001), which up to 14 factors do not suffer from perfect aliasing. For 32 runs, a cyclic design with generating row given in Samset and Tyssedal (1999) is used. For 64 runs, the 32 run design is doubled. For 92 runs, a design is constructed according to the Williamson construction with matrices A, B, C and D from Hedayat and Stufken (1999), p. 160.

So far, designs up to 96-runs are covered. More and different ones may follow, since the package is currently under intensive development.

Usage of the 8 run design for more than 4 factors is discouraged, as it completely aliases main effects with individual two-factor interactions. It is recommended to use at least the 12 run design instead for screening more than 4 factors.

Value

Value is a data frame of S3 class `design` and has attached attributes that can be accessed by functions `desnum`, `run.order` and `design.info`.

The data frame itself contains the design with levels coded as requested. If no center points have been requested, the design columns are factors with contrasts -1 and $+1$ (cf. also `contr.FrF2`); in case of center points, the design columns are numeric.

The following attributes are attached to it:

<code>desnum</code>	Design matrix in $-1/1$ coding
---------------------	--------------------------------

`run.order` three column data frame, first column contains the run number in standard order, second column the run number as randomized, third column the run number with replication number as postfix; useful for switching back and forth between actual and standard run number

`design.info` list with entries

type character string “pb”, except for 8~runs with up to 4~factors, for which a type “FrF2” design is output

nruns number of runs (replications are not counted)

nfactors number of factors; not for designs of type `FrF2.blocked`, where `ntreat` takes this role

ntreat for designs of type `FrF2.blocked` only; number of treatment factors

factor.names list named with (treatment) factor names and containing as entries vectors of length two each with coded factor levels

replication option setting in call to `pb`

repeat.only option setting in call to `pb`

randomize option setting in call to `pb`

seed option setting in call to `pb`

creator call to function `pb` (or stored menu settings, if the function has been called via the R commander plugin **RcmdrPlugin.DoE**)

Author(s)

Ulrike Groemping

References

- Box, G.E.P. and Tyssedal, J. (2001) Sixteen Run Designs of High Projectivity for Factor Screening. *Communications in Statistics - Simulation and Computation* **30**, 217-228.
- Hedayat, A.S., Sloane, N.J.A. and Stufken, J. (1999) *Orthogonal Arrays: Theory and Applications*, Springer, New York.
- Mee, R. (2009). *A Comprehensive Guide to Factorial Two-Level Experimentation*. New York: Springer.
- Plackett, R.L.; Burman, J.P. (1946) The design of optimum multifactorial experiments. *Biometrika* **33**, 305-325.
- Samset, O.; Tyssedal, J. (1999) Two-level designs with good projection properties. *Technical Report 12, Department of Mathematical Sciences, The Norwegian University of Science and Technology, Norway*.
- Williamson, J. (1946) Determinants whose elements are 0 and 1. *American Mathematical Monthly* **53**, 427-434.

See Also

See Also [FrF2](#) for regular fractional factorial designs

Examples

```
pb(12, randomize=FALSE)
pb(12, randomize=FALSE, n12.taguchi=TRUE)
pb(20, seed=29869)
pb(16, factor.names=list(A="", B="", C="", D=c("min", "max"),
  E="", F="", G="", H="", J=c("new", "old")))
pb(8, default.levels=c("current", "new"))
test <- pb(40) ## design created by doubling the 20 run design
pb(12, ncenter=6) ## 6 center points with default placement
```

splitplot

Statistical and algorithmic aspects of split-plot designs in FrF2

Description

This help page documents the statistical and algorithmic details of split-plot designs in FrF2

Details

A split-plot design is similar to a **blocked** design, with the difference that there are also factors of interest that can be only changed on block level (so-called whole plot factors). The blocks are called “plots” in the context of split-plot designs. The factors that can (and should!) be varied within a plot are called split-plot factors. Note that the experiment provides more information on split-plot factors than on whole-plot factors.

Warning: In terms of analysis, split-plot designs would have to be treated by advanced random effects models, but often are not. At the very least, the user must be aware that all whole-plot effects (i.e. effects on columns that only change between plots) are (likely to be) more variable than split-plot effects so that e.g. it does not necessarily mean anything if they stick out in a normal or half-normal effects plot.

Designs for hard-to-change factors are also treated by the split-plot approach in function `FrF2`, although they are not quite split-plot designs: They are non-randomized split-plot designs arranged in an order such that the first whole-plot factors have as few as possible changes. This gives very poor information on these first whole-plot factors (which in the extreme are only changed once or twice), if there is variability involved with setting the factor levels.

If hard-to-change factors can be implemented as true whole-plot factors with randomization, this is by far preferable from a statistical point of view (but may nevertheless be rejected from a feasibility point of view, as the necessary changes may seem unaffordable).

For design generation, there are two principal ways to handle split-plot designs, manual definition (i.e. the user specifies exactly which columns are to be used for which purpose) and automatic definition. Each situation has its specifics. These are detailed below. For users with not so much mathematical/statistical background, it will often be best to use the automatic way, specifying the treatment factors of interest via `nfactors` or `factor.names` and a single number for WPs. Users with more mathematical background may want to use the manual definitions, perhaps in conjunction with published catalogues of good split-plot designs, or after inspecting possibilities with function `splitpick`.

Manual definition of split-plot designs The user can specify a design with the `design` or the `generators` option and specify manually with the `WPfacs` option, which factors are whole plot factors (i.e. factors that do not change within a plot). The other factors become split-plot factors (i.e. factors that do change within a plot). If the user chooses this route, `WPfacs` must be character vectors of factor names, factor letters, factor numbers preceded by capital F, or a vector or list of factor position numbers (NOT: Yates column numbers). Caution: It is the users responsibility to ensure a good choice of split-plot design (e.g. by using a catalogued design from Huang, Chen and Voelkel 1998, Bingham and Sitter 2003, or Bingham Schoen and Sitter 2004). In case of a user-mistake such that the resulting design is not a split-plot design with the alleged number of whole plots, an error is thrown.

Automatic definition of split-plot designs As mentioned above, split-plot designs differ from block designs by the fact that the block main effects are purely nuisance parameters which are assumed (based on prior knowledge) to be relevant but are not of interest, while the plots are structured by `nfac.WP` whole plot factors, which are of interest. The user has to decide on a number of whole plots (`WPs`) as well as the number of whole plot factors `nfac.WP`. If $\log_2(WPs) \leq nfac.WP \leq WPs-1$, it is obviously in principle possible to accommodate the desired number of whole plot factors in the desired number of whole plots. If $nfac.WP > WPs/2$, the base design for the split-plot structure has to be of resolution III. Sometimes, subject matter considerations limit whole plot sizes, and there are only few interesting whole plot factors, i.e. $nfac.WP < \log_2(WPs)$. In this case, it is of course nevertheless necessary to have a total of $\log_2(WPs)$ whole plot *construction* factors; the missing $\log_2(WPs) - nfac.WP$ factors are added to the design (names starting with `WP`), and `nfactors` is increased accordingly.

In all cases, the first `nfac.WPs` user-specified factors are treated as whole plot factors, the remaining factors as split-plot factors.

From there, function `FrF2` proceeds like in the blocked situation by starting with the best design and working its way down to worse designs, if the best design cannot accommodate the desired split-plot structure. For each design, function `FrF2` calls function `splitpick`, which permutes base factors until the requested whole plot / split-plot structure is achieved, or until impossibility for this design with these base factors has been ascertained. In the latter case, function `FrF2` proceeds to the next best design and so forth.

If several competing split-plot designs based on the same base design are found, the best possible resolution among the first `check.WPs` such designs is chosen. No further criteria are automatically implemented, and no more than `check.WPs` designs are checked. If not satisfied with the structure of the whole plot portion of the experiment, increasing `check.WPs` vs. the default 10 may help. Expert users may want to inspect possibilities, using function `splitpick` directly.

Note that the algorithm does not necessarily find an existing split-plot design. It has been checked out which catalogued designs it can find: designs for all catalogued situations from Bingham and Sitter (2003) have been found, as well as for most catalogued situations from Huang, Chen and Voelkel (1998). Occasionally, a better design than catalogued has been found, e.g. for 4 whole plot and 10 split plot factors in 32 runs with 16 whole plots, the design found by the algorithm is resolution IV, while Huang, Chen and Voelkel propose a resolution III design. The algorithm has the largest difficulties with extreme designs in the sense that a large number of whole plots with a small number of whole plot factors are to be accommodated; thus it does not find designs for the more extreme situations in Bingham, Schoen and Sitter (2004).

Please contact me with any suggestions for improvements.

Author(s)

Ulrike Groemping

References

- Bingham, D.R., Schoen, E.D. and Sitter, R.R. (2004). Designing Fractional Factorial Split-Plot Experiments with Few Whole-Plot Factors. *Applied Statistics* **53**, 325-339.
- Bingham, D. and Sitter, R.R. (2003). Fractional Factorial Split-Plot Designs for Robust Parameter Experiments. *Technometrics* **45**, 80-89.
- Chen, J., Sun, D.X. and Wu, C.F.J. (1993) A catalogue of 2-level and 3-level orthogonal arrays. *International Statistical Review* **61**, 131-145.
- Cheng, C.-S. and Tsai, P.-W. (2009). Optimal two-level regular fractional factorial block and split-plot designs. *Biometrika* **96**, 83-93.
- Huang, P., Chen, D. and Voelkel, J.O. (1998). Minimum-Aberration Two-Level Split-Plot Designs. *Technometrics* **40**, 314-326.

See Also

See Also [FrF2](#) for regular fractional factorials, [catlg](#) for the Chen, Sun, Wu catalogue of designs and some accessor functions, and [block](#) for the statistical aspects of blocked designs.

Examples

```
##### hard to change factors #####
## example from Bingham and Sitter Technometrics 19999
## MotorSpeed, FeedMode, FeedSizing, MaterialType are hard to change
BS.ex <- FrF2(16, 7, hard=4,
             factor.names=c("MotorSpeed", "FeedMode", "FeedSizing", "MaterialType",
                           "Gain", "ScreenAngle", "ScreenVibLevel"),
             default.levels=c("-", "+"), randomize=FALSE)
design.info(BS.ex)
BS.ex
## NOTE: the design has 8 whole plots.
## If randomize=FALSE is used like here, the first hard-to-change factors
## do not always change between whole plots.
## A conscious and honest decision is required whether this is
## acceptable for the situation at hand!
## randomize=TRUE would cause more changes in the first four factors.

##### automatic generation for split plot #####
## 3 control factors, 5 noise factors, control factors are whole plot factors
## 8 plots desired in a total of 32 runs
## Bingham Sitter 2003
BS.ex2a <- FrF2(32, 8, WPs=8, nfac.WP=3,
               factor.names=c(paste("C", 1:3, sep=""), paste("N", 1:5, sep="")), randomize=TRUE)

## manual generation of this same design
```

```

BS.ex2m <- FrF2(32, 8, generators=c("ABD", "ACD", "BCDE"), WPs=8, WPfacs=c("C1", "C2", "C3"), nfa
  factor.names=c(paste("C", 1:3, sep=""), paste("N", 1:5, sep="")), randomize=TRUE)

## design with few whole plot factors
## 2 whole plot factors, 7 split plot factors
## 8 whole plots, i.e. one extra WP factor needed
BSS.cheese.exa <- FrF2(32, 9, WPs=8, nfac.WP=2,
  factor.names=c("A", "B", "p", "q", "r", "s", "t", "u", "v"))
design.info(BSS.cheese.exa)
## manual generation of the design used by Bingham, Schoen and Sitter
## note that the generators include a generator for the 10th splitting factor
## s = ABq, t = Apq, u = ABpr and v = Aqr, splitting factor rho=Apqr
BSS.cheese.exm <- FrF2(32, gen=list(c(1,2,4), c(1,3,4), c(1,2,3,5), c(1,4,5), c(1,3,4,5)),
  WPs=8, nfac.WP=3, WPfacs=c(1,2,10),
  factor.names=c("A", "B", "p", "q", "r", "s", "t", "u", "v", "rho"))
design.info(BSS.cheese.exm)

```

StructurePickers *Functions to find split-plot or left-adjusted designs*

Description

Functions to restructure a fractional factorial by permuting the base factors such that the leftmost base factors have a suitable alias structure for the problem at hand; meant for expert users

Usage

```

splitpick(k, gen, k.WP, nfac.WP, show=10)
leftadjust(k, gen, early=NULL, show=10)

```

Arguments

k	the number of base factors (designs have 2^k runs)
gen	vector of generating columns from Yates matrix
k.WP	integer number of base factors used for whole plot generation; there will be $2^{k.WP}$ plots in the design
nfac.WP	integer number of whole plot factors, must not be smaller than k.WP; the other $k + \text{length}(\text{gen})$ factors are split-plot factors, i.e. they change within plots
show	numeric integer indicating how many results are to be shown; for function <code>splitpick</code> , this number also determines how many designs are investigated; the other two functions investigate all designs and show the best results only
early	number that indicates how many “leftmost” factors are needed in the design; used by <code>FrF2</code> for accomodating hard-to-change factors

Details

These functions exploit the fact that a factorial design can be arranged such that the $2^{k \cdot WP - 1}$ leftmost columns have exactly $2^{k \cdot WP}$ different patterns. They can thus accommodate whole plot effects if $2^{k \cdot WP}$ plots are available; also, with a specially rearranged version of the Yates matrix, the leftmost columns can have particularly few or particularly many level changes, cf. e.g. Cheng, Martin and Tang 1998.

By permuting the k base factors, the functions try to find $2^{k \cdot WP}$ ones that accommodate the current needs, if taken as the first base factors. They are used by function `FRF2`, if a user requests an automatically-generated split-plot design or a design with some factors declared hard-to-change.

There may be a possibility to better accommodate the experimenters needs within a given design by trying different sets of base factors. This is not done in these functions. Also, custom user needs may be better fulfilled, if an expert user directly uses one of these functions for inspecting the possibilities, rather than relying on the automatic selection routine in function `FRF2`.

Value

Both functions output a list of entries with information on at most `show` suitable permutations. `splitpick` ends with an error, if no suitable solution can be found.

<code>orig</code>	original generator column numbers
<code>basics</code>	named vector with the following entries: for function <code>splitpick</code> , entries are the number of runs, the number of plots, the number of whole plot factors and the number of split-plot factors for function <code>leftadjust</code> , entries are the number of runs, the number of factors, and - if <code>early</code> is not <code>NULL</code> - the entry for <code>early</code>
<code>perms</code>	matrix with rows containing permutations of base factors
<code>res.WP</code>	for <code>splitpick</code> only; vector of resolutions for the respective rows of <code>perms</code>
<code>maxpos</code>	for <code>leftadjust</code> only; vector of maximum positions for the <code>early</code> leftmost factors for the respective rows of <code>perms</code>
<code>k.early</code>	for <code>leftadjust</code> only; vector of numbers of base factors needed for spanning the <code>early</code> leftmost factors for the respective rows of <code>perms</code>
<code>gen</code>	matrix the rows of which contain the generator columns for the respective rows of <code>perms</code>

Author(s)

Ulrike Groemping

References

Cheng, C.-S., Martin, R.J., and Tang, B. (1998). Two-level factorial designs with extreme numbers of level changes. *Annals of Statistics* **26**, 1522-1539.

See Also

See Also [FrF2](#)

Examples

```
## leftadjusting MA design from table 6.22 in BHH2, 9 factors, 32 runs
## NOTE: nevertheless not as well left-adjusted as the isomorphic design 9-4.1 from catlg
leftadjust(5,c(30,29,27,23))
## with option early=4 (i.e. 4 columns as early as possible are requested)
leftadjust(5,c(30,29,27,23),early=4)
leftadjust(5,catlg$'9-4.1'$gen,early=4)

## look for a split plot design in 32 runs with 7 factors,
##      3 of which are whole plot factors,
##      and 8 plots
splitpick(5,catlg$'7-2.1'$gen,nfac.WP=3,k.WP=3)
```

Index

- *Topic **array**
 - add.center, 4
 - block, 9
 - blockpick, 11
 - CatalogueAccessors, 14
 - estimable.2fis, 21
 - fold.design, 24
 - FrF2, 27
 - FrF2-package, 2
 - pb, 42
 - splitplot, 46
 - StructurePickers, 49
- *Topic **design**
 - add.center, 4
 - aliases, 7
 - block, 9
 - blockpick, 11
 - CatalogueAccessors, 14
 - cubePlot, 18
 - DanielPlot, 19
 - estimable.2fis, 21
 - fold.design, 24
 - FrF2, 27
 - FrF2-package, 2
 - IAPlot, 39
 - pb, 42
 - splitplot, 46
 - StructurePickers, 49
 - [.catlg (CatalogueAccessors), 14

 - add.center, 4, 28, 43
 - alias, 8
 - aliases, 3, 7, 41
 - aliasprint (aliases), 7
 - all.2fis.clear.catlg (CatalogueAccessors), 14

 - block, 9, 31, 33, 36, 46, 48
 - block.catlg (CatalogueAccessors), 14

 - blockpick, 9, 10, 11, 34
 - blockpick.big, 9, 10, 34
 - BsMD, 3

 - CatalogueAccessors, 14
 - catlg, 11, 23, 29, 33, 36, 48
 - catlg (CatalogueAccessors), 14
 - check (IAPlot), 39
 - clear.2fis.catlg (CatalogueAccessors), 14
 - contr.FrF2, 33, 44
 - cubecorners (cubePlot), 18
 - cubedraw (cubePlot), 18
 - cubelabel (cubePlot), 18
 - cubePlot, 3, 18

 - DanielPlot, 3, 19
 - design, 3, 20, 33, 40, 44
 - design.info, 33, 44
 - desnum, 33, 44
 - DoE.base, 3
 - DoE.wrapper, 3

 - estimable.2fis, 21, 29, 30, 33, 36

 - fold.design, 24
 - FrF2, 3, 5, 6, 10–13, 17, 23, 26, 27, 44, 45, 47, 48, 50
 - FrF2-package, 8, 19, 42
 - FrF2-package, 2

 - IAPlot, 3, 8, 39
 - intfind (IAPlot), 39
 - iscube (add.center), 4

 - leftadjust, 13
 - leftadjust (StructurePickers), 49
 - Letters, 28, 43

 - MEPlot, 3
 - MEPlot (IAPlot), 39

`myscatterplot3d` (`cubePlot`), 18

`nclear.2fis.catlg`
(`CatalogueAccessors`), 14

`nfac.catlg` (`CatalogueAccessors`),
14

`nruns.catlg` (`CatalogueAccessors`),
14

`pb`, 3, 5, 6, 26, 36, 42

`print.catlg` (`CatalogueAccessors`),
14

`print.default`, 8

`qqnorm`, 21

`remodel` (`IAPlot`), 39

`res.catlg` (`CatalogueAccessors`), 14

`run.order`, 33, 44

`splitpick`, 13, 46, 47

`splitpick` (`StructurePickers`), 49

`splitplot`, 11, 32, 33, 36, 46

`StructurePickers`, 49

`WLP.catlg` (`CatalogueAccessors`), 14