Package: FrF2 (via r-universe)

September 10, 2024

Title Fractional Factorial Designs with 2-Level Factors

Version 2.3-3

Depends R(>= 2.13.0), DoE.base(>= 0.25)

Imports sfsmisc(>= 1.0-26), utils, scatterplot3d, igraph(>= 0.7), methods

Suggests FrF2.catlg128, BsMD, DoE.wrapper

Date 2023-09-11

Author Ulrike Groemping

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

Description Regular and non-regular Fractional Factorial 2-level designs can be created. 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).

License GPL (>= 2)

LazyLoad yes

Encoding UTF-8

URL https://prof.bht-berlin.de/groemping/DoE/, https://prof.bht-berlin.de/groemping/

NeedsCompilation no

Date/Publication 2023-09-20 09:00:02 UTC

Repository https://ugroempi.r-universe.dev

RemoteUrl https://github.com/cran/FrF2

RemoteRef HEAD

RemoteSha 65d2de16bcb789bf76b58ccb2b6a0a2631d803c6

Contents

`FrF2-package` *Fractional Factorial designs with 2-level factors*

Description

creates regular and non-regular Fractional Factorial 2-level 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 still subject to development; most key functionality is now included. 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. For regular fractional factorials, function `FrF2` permits the specification of effects of interest, whose estimation is requested clear of aliasing with other effects. The function can furthermore generate regular fractional factorials as blocked or split-plot designs, and hard-to-change factors can be specified in order to keep the number of level changes low. Regular resolution V designs larger than

those obtainable from function FrF2 can be created by function FrF2Large (these are not guaranteed to be optimal). 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. The original above-mentioned functions are not available in R; similar functions have been implemented in package **DoE.base** together with other general functionality for experimental designs.

In terms of analysis, package FrF2 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 (less frequent) 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, to automatically distinguish between whole-plot and split-plot effects for split-plot designs, 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.

Author(s)

Ulrike Groemping

Maintainer: Ulrike Groemping <ulrike.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.

Groemping, U. (2014). R Package FrF2 for Creating and Analyzing Fractional Factorial 2-Level Designs. *Journal of Statistical Software*, **56**, Issue 1, 1-56. https://www.jstatsoft.org/v56/i01/.

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.

Ryan, K.J. and Bulutoglu, D.A. (2010). Minimum Aberration Fractional Factorial Designs With Large N. *Technometrics* **52**, 250-255.

Sanchez, S.M. and Sanchez, P.J. (2005). Very Large Fractional Factorial and Central Composite Designs. *ACM Transactions on Modeling and Computer Simulation* **15**, 362-377.

See Also

The key design generating functions: FrF2, pb, FrF2Large
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, package="BsMD")  #from BsMD
## iMdat <- BM93.e3.data[1:16,2:10]  #only original experiment
## re-create here
y=c(14, 16.8, 15, 15.4, 27.6, 24, 27.4, 22.6,
 22.3, 17.1, 21.5, 17.5, 15.9, 21.9, 16.7, 20.3)
iMdat <- FrF2(8,7,randomize=FALSE)
iMdat <- desnum(iMdat)
iMdat <- rbind(cbind(iMdat,H=1),cbind(-iMdat,H=-1))
iMdat <- cbind(as.data.frame(iMdat), y=y)

# 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, las=1)
# interaction plots
IAPlot(iM.lm, las=1)
# interaction plots with attention drawn to aliases
aus <- IAPlot(iM.lm, show.alias=TRUE)
# alias groups corresponding to interaction plots
aliases(iM.lm)$aliases[9:15]
```

```
# returned object
aus
# 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, ...)
```

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 randomdomized) 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 from package **DoE.base** provides a logical vector that is TRUE for cube points and FALSE for center points, which allows to 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). The element coding is also added to the design.info, in order to support steepest ascent/descent analysis from the center point.

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
  planc <- add.center(plan, 6, distribute=1)
    ## conduct additional runs for the center points
      y <- c(y, c(2+desnum(planc)[!iscube(planc),1:4]%*%c(2,3,0,0) +
```

```
          1.5*apply(desnum(planc)[!iscube(planc),][,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, ...)
## S3 method for class 'aliases'
print(x, ...)
```

Arguments

`fit`	a linear model object with only 2-level factors as explanatory 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`	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.
`condense`	if TRUE, reformats the alias information to be comparable to the version calculated by internal function alias3fi; does not work with models with higher than 3-way interactions; for up to 3-way interactions, the output may be more easily readible
`design`	a data frame of class `design` that should contain a fractional factorial 2-level design; the function does not print anything if the design is of different nature

x	an object of class aliases that should be the output from function aliases
...	further arguments to function print.default; the quote argument cannot be used

Value

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

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

legend	links the codes to variable names, if code=TRUE.
aliases	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 preferrable for looking at the alias structure of larger designs.

The output object of function aliases has class aliases, which is used for customized printing with the print method.

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, package="BsMD")  #from BsMD
## iMdat <- BM93.e3.data[1:16,2:10]  #only original experiment
## re-create here
y=c(14, 16.8, 15, 15.4, 27.6, 24, 27.4, 22.6,
 22.3, 17.1, 21.5, 17.5, 15.9, 21.9, 16.7, 20.3)
iMdat <- FrF2(8,7,randomize=FALSE)
iMdat <- desnum(iMdat)
iMdat <- rbind(cbind(iMdat,H=1),cbind(-iMdat,H=-1))
iMdat <- cbind(as.data.frame(iMdat), y=y)

# 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 accomodate 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`, `blockpick.big` (default before version 2 for large settings) or `colpick` (default since version 2 for large settings or settings with estimability requirements).

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 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))` .
Finally, it is also possible to specify choice of blocks using a vector of Yates column numbers, in order to be able to use catalogued blocking structures of this form, e.g. from Sitter, Chen and Feder (1997).
The block main effects are defined by the `k.block` specified effect and all interactions between them. The specified block effects are required to be independent from each other, which implies that they generate `2^k.block` 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, Sitter, Chen and Feder (1997), or Cheng and Wu 2002). Since version 2 of package **FrF2**, manual blocking is also checked for confounding of the block factor with main effects or two-factor interactions; this implies that some earlier code will now require the additional specification of argument `alias.block.2fis=TRUE` in order to avoid errors.

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 function `colpick` with subsequent `blockgencreate`, except where the Sun, Wu and Chen (1997) catalogue of blocked designs contains suitable block generators for a design without estimability requirements (implemented in function `blockpick`, which also calls `colpick`, if that catalogue does not offer a solution). Otherwise, depending on the situation, function `FrF2` uses function `blockpick` or function `colpick` with subsequent `blockgencreate`; function `blockpick` treats smaller problems (`choose(nruns-1-nfactors,k.block)` < `100000`) without estimability requirements and with `force.godolphin=FALSE` (the latter is per default set to TRUE whenever `alias.block.2fis=TRUE`), other problems are treated by function `colpick`.

Use of the earlier default function `blockpick.big` for large cases or the earlier behavior for full factorial designs can be requested with the argument `block.old=TRUE`; this should only be done for reproducing earlier results, as the new methodology is definitely superior.

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 (exception: with an estimability requirement, only a single design, prefiltered for the estimability requirement, is subjected to the blocking algorithm).
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, or blocking in combination with requiring some 2fis to be clear of aliasing, or per default for blocking with permitting 2fis to be aliased with blocks, function `colpick` creates a $q \times n$ $\mathbf{X}$ matrix for creating blocks of size 2^q based on the approach de-

scribed by Godolphin (2021); function `blockgencreate` creates block generators from this matrix. This approach can be used in combination with argument `estimable`, as long as `clear=TRUE`. The implementation of this approach is described in Groemping (2021). The argument `force.godolphin` of function `FrF2` can enforce the Godolphin approach instead of the default approach for small blocked designs without `alias.block.2fis=TRUE`, and the Godolphin approach can be switched off for `alias.block.2fis=TRUE` applications by explicitly requesting `force.godolphin=FALSE`. Note that the Godolphin approach solely focuses on clear 2fis of the blocked design and does not attempt to avoid confounding of the block factor with non-clear 2fis; it may thus confound 2fis with the block factor even if this were avoidable, maintaining the same number of clear 2fis.

For the larger problems, in versions before 2.0, which can be activated in current versions with `block.old=TRUE`, 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. 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) provided a catalogue of blocked designs with a few quality criteria, and they stated 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. Expert 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` or `colpick` (or, if desparate, `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.

Godolphin, J. (2021). Construction of Blocked Factorial Designs to Estimate Main Effects and Selected Two-Factor Interactions. *J. Royal Statistical Society* **B 83**, 5-29. doi: 10.1111/rssb.12397.

Groemping, U. (2019). An algorithm for blocking regular fractional factorial 2-level designs with clear two-factor interactions. *Reports in Mathematics, Physics and Chemistry*, Report 3/2019, Department II, Beuth University of Applied Sciences Berlin.

Sitter, R.R., Chen, J. and Feder, M. (1997). Fractional Resolution and Minimum Aberration in Blocked 2n-k Designs. *Technometrics* **39**, 382–390.

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"))

########## blocked design with estimable 2fis ###################
### all interactions of last two factors to be estimable clearly
### in 64 run design with blocks of size 4
### not possible with catalogue entry 9-3.1
FrF2(design="9-3.2", blocks=16, alias.block.2fis=TRUE,
    factor.names = list(C1="",C2="",C3="",C4="",C5="",C6="",C7="",
    N1=c("low","high"),N2=c("low","high")),
    default.levels = c("current","new"),
    estimable=compromise(9, 8:9)$requirement)
```

---

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 gen = 0 or gen=numeric(0) for no generating columns; but note that there is always just the one and only catalogued design returned for a full factorial. For function blockpick, gen refers to the generators of the base design only, and block columns are automatically added by blockpick. For function blockpick.big, gen refers to the generators for treatment factors and block generators. In fact, blockpick.big will always use the first k.block (base) factors for block generation. Hence, for example for generating a design in 64 runs and 7 factors with 32 blocks, gen 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 select.catlg
show	numeric integer indicating how many results are to be shown; the search for possible allocations stops, once show 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 show 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 and without estimability requirements. blockpick 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.

In **FrF2** versions before 2.0, function blockpick.big was used for large use cases; this can still be requested using argument block.old=TRUE. Since FrF2 version 2, the Godolphin (2021) based

approach is used instead, both for large cases and for cases where blocking is combined with estimability requirements (clear=TRUE only); the big advantage is the ability of combining blocking with estimability requirements, and a substantial speed gain if small blocks are needed.

All approaches 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 amoong treatment effects may be aliased with block main effects (option alias.block.2fis). (For the Godolphin approach to work, one will usually need to set alias.block.2fis to TRUE.)

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 ressources. 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 numbers (w.r.t. the Yates matrix) of the block generators for the current assignment (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 treatment 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^n and 2^{n-p} designs. *Technometrics* **39**, 298-307.

See Also

See Also FrF2

Examples

```
## look at possibilities for running a 32 run design with 6 factors in 8 blocks
## running this without alias.block.2fis=TRUE throws an error: not possible
## Not run: blockpick(k=5,design="6-1.1",k.block=3)
## the 8th to 10th design have more clear 2fis than the earlier ones
blockpick(k=5,design="6-1.1",k.block=3,alias.block.2fis=TRUE)
## function FrF2 can be used to manually accomodate this
des32.6fac.8blocks.MaxC2 <- FrF2(32,6,blocks=c(3,12,21),alias.block.2fis=TRUE)
summary(des32.6fac.8blocks.MaxC2)
## automatic block generation leads to more aliased 2fis
summary(FrF2(32,6,blocks=8,alias.block.2fis=TRUE))

## 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:
## (not run for saving check time on CRAN)
## Not run:
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)

## End(Not run)
```

BsProb.design *Bayesian posterior probabilities from Box and Meyer method*

Description

The function calculates Bayesian posterior probabilities according to Box and Meyer (1993) for screening experiments with 2-level factors. The function is modified from function BsProb in packge **BsMD** with the purpose of providing usage comfort for class design objects.

Usage

```
BsProb.design(design, mFac = NULL, response=NULL, select=NULL, mInt = 2, p = 0.25, g = 2,
    ng = 1, nMod = 10)
```

Arguments

design	an experimental design of class design with the type element of the design.info attribute containing "FrF2" or "pb" and at least one response variable
response	NULL or a character string that specifies response variable to be used, must be an element of response.names(obj); if NULL, the first response from response.names(obj) is used
mFac	integer. Maximum number of factors included in the models. The default is the number of factors in the design.
select	vector with position numbers of the factors to be included; default: all factors.
mInt	integer <= 3. Maximum order of interactions considered in the models. This can strongly impact the result.
p	numeric. Prior probability assigned to active factors. This can strongly impact the result.
g	numeric vector. Variance inflation factor(s) gamma associated to active and interaction factors; see "Details" section
ng	integer <=20. Number of different variance inflation factors (g) used in calculations.
nMod	integer <=100. Number of models to keep with the highest posterior probability.

Details

Factor and model posterior probabilities are computed by the Box and Meyer (1993) Bayesian procedure. The design factors - or a selection of these given by column numbers in select - are considered together with the specified response or the first response of the design. The function has been adapted from function BsProb in package **BsMD**, and a vignette in that package (../../BsMD/doc/BsMD.pdf) explains details of the usage regarding the parameters.

If g, the variance inflation factor (VIF) gamma, is a vector of length 1, the same VIF is used for factor main effects and interactions. If the length of g is 2 and ng is 1, g[1] is used for factor main effects and g[2] for the interaction effects. If ng greater than 1, then ng values of VIFs between g[1] and g[2] are used for calculations with the same gamma value for main effects and interactions. The function calls the FORTRAN subroutine bm and captures summary results. The complete output of the FORTRAN code is save in the BsPrint.out file in the working directory. The output is a list of class BsProb for which print, plot and summary methods are available from package **BsMD**.

Value

cf. documentation of function BsProb

Note

This method relies on the availability of package **BsMD**.

Author(s)

Daniel Meyer, ported to R by Ernesto Barrios, port adapted to designs by Ulrike Groemping.

References

Barrios, E. (2013). Using the BsMD Package for Bayesian Screening and Model Discrimination. Vignette. ../../BsMD/doc/BsMD.pdf.

Box, G. E. P and R. D. Meyer (1986). An Analysis for Unreplicated Fractional Factorials. *Technometrics* **28**, 11-18.

Box, G. E. P and R. D. Meyer (1993). Finding the Active Factors in Fractionated Screening Experiments. *Journal of Quality Technology* **25**, 94-105.

See Also

plot.BsProb, print.BsProb, summary.BsProb, BsMD

Examples

```
### there are several success stories and recommendations for this method
### in the simulated example here (not fabricated,
###         it was the first one that came to my mind),
### the method goes wrong, at least when using mInt=2 (the default, because
###         Daniel plots work quite well for pure main effects models):
### active factors are A to E (perhaps too many for the method to work),
### the method identifies F, J, and L with highest probability
### (but is quite undecided)
plan <- pb(12)
dn <- desnum(plan)
set.seed(8655)
y <- dn%*%c(2,2,2,2,3,0,0,0,0,0,0) + dn[,1]*dn[,3]*2 - dn[,5]*dn[,4] + rnorm(12)/10
plan.r <- add.response(plan, response=y)
if (requireNamespace("BsMD", quiet=TRUE)){
plot(bpmInt2 <- BsProb.design(plan.r), code=FALSE)
plot(bpmInt1 <- BsProb.design(plan.r, mInt=1), code=FALSE) ## much better!
summary(bpmInt2)
summary(bpmInt1)
}
### For comparison: A Daniel plot does not show any significant effects according
### to Lenths method, but makes the right effects stick out
DanielPlot(plan.r, half=TRUE, alpha=1)
```

CatalogueAccessors *Catalogue file and accessor functions*

Description

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

Usage

```
res(catlg)
## S3 method for class 'catlg'
res(catlg)
## S3 method for class 'character'
res(catlg)
nruns(catlg)
## S3 method for class 'catlg'
nruns(catlg)
## S3 method for class 'character'
nruns(catlg)
nfac(catlg)
## S3 method for class 'catlg'
nfac(catlg)
## S3 method for class 'character'
nfac(catlg)
WLP(catlg)
## S3 method for class 'catlg'
WLP(catlg)
## S3 method for class 'character'
WLP(catlg)
nclear.2fis(catlg)
## S3 method for class 'catlg'
nclear.2fis(catlg)
## S3 method for class 'character'
nclear.2fis(catlg)
clear.2fis(catlg)
## S3 method for class 'catlg'
clear.2fis(catlg)
## S3 method for class 'character'
clear.2fis(catlg)
all.2fis.clear.catlg(catlg)
dominating(catlg)
## S3 method for class 'catlg'
dominating(catlg)
## S3 method for class 'character'
dominating(catlg)
catlg
## S3 method for class 'catlg'
```

```
catlg[i]
## S3 method for class 'catlg'
print(x, name="all", nruns="all", nfactors="all",
                        res.min=3, MaxC2=FALSE, show=10,
                        gen.letters=FALSE, show.alias=FALSE, ...)
block.catlg
```

Arguments

catlg	Catalogue of designs of class catlg (cf. details section), or character vector with name(s) of catlg element(s) in case of accessor functions
i	vector of index positions or logical vector that can be used for indexing a catlg object
x	an object of class catlg
name	character vector of entry names from x; default "all" means: no selection made
nruns	numeric integer (vector), giving the run size(s) for entries of x to be shown; default "all" means: no selection made
nfactors	numeric integer (vector), giving the factor number(s) for entries of x to be shown; default "all" means: no selection made
res.min	numeric integer giving the minimum resolution for entries of x to be shown
MaxC2	logical indicating whether designs are ordered by minimum aberration (default, MaxC2=FALSE) or by maximum number of clear 2fis (MaxC2=TRUE)
show	integer number indicating maximum number of designs to be shown; default is 10
gen.letters	logical indicating whether the generators should be shown as column numbers (default, gen.letters=FALSE) or as generators with factor letters (e.g. E=ABCD, gen.letters=TRUE)
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 `2xnclear.2fis` matrix of clear 2-factor interactions (clear to be understood in the above sense); this matrix represents each designs clear interaction graph, which can be used in automated searches for designs that can accomodate (i.e. clearly) a certain requirement set of 2-factor interactions; cf. also `estimable.2fis`

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

dominating logical that indicates whether the current design adds a CIG structure that has not been seen for a design with less aberration (cf. Wu, Mee and Tang 2012 p.196 for dominating designs); TRUE, if so; FALSE, if current CIG is isomorphic to previous one or has no edges (IMPORTANT: the dominance assessment refers to the current catalogue; for designs with more than 64 runs, it is possible that a design marked dominating in catalogue `catlg` is not dominating when considering ALL non-isomorphic designs).
This element is helpful in omitting non-promising designs from a search for a clear design.
This element may be missing. In that case, all catalogue entries are assumed dominating.

Reference to factors in components `clear.2fis` and `all.2fis.clear` is via their position number (element of (1: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 is a living object, since it has to be updated with recent research results for larger designs. In particular, new MA designs may be found, or it may be proven that previous "good" designs are in fact of minimum aberration.

Currently, the catalogue 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.
- the MA designs in 128 runs:
  - for up to 24 factors obtained from Xu (2009),

  - for 25 to 64 factors taken from Block and Mee (2005, with corrigendum 2006),
  - for 65 to 127 factors (resolution III): 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.

- various further "good" resolution IV designs in 128 runs obtained by evaluating designs from the complete catalogue by Xu (2009, catalogue on his website) w.r.t. aberration and number of clear 2fis (including also all designs that yield minimum aberration clear compromise designs according to Groemping 2010); all designs with resolution at least IV for up to 11 factors have been added with version 2.2.
- the MA even designs in 128 runs, in support of blocking according to the Godolphin approach have been added with version 2.2.

  Note that additional non-isomorphic resolution IV designs in 128 runs are available in package (**FrF2.catlg128**); since the catalogues are quite large, they are not forced upon users of this package who do not need them. Since version 1.1 of that package, the catalogues are not complete but contain high resolution fractions and even/odd fractions only (status: version 1.2-x); re-inclusion of at least selected even fractions is intended, because these may yield improved support of blocking in connection with estimable 2fis.
- the best (MA) resolution IV or higher designs in
  256 runs for up to 36 factors (resolution V up to 17 factors),
  512 runs for up to 29 factors (resolution V for up to 23 factors).
  These have been taken from Xu (2009) with additions by Ryan and Bulutoglu (2010).
- Further "good" resolution IV designs with up to 80 factors in 256 runs and up to 160 factors in 512 runs have also been implemented from Xu (2009).
- the best (MA) resolution V or higher design for each number of factors or a "good" such design (if it is not known which one is best) in
  1024 runs (up to 33 factors, MA up to 28 factors, resolution VI up to 24 factors),
  2048 runs (up to 47 factors, MA up to 32 factors, resolution VI up to 34 factors),
  and 4096 runs (up to 65 factors, MA up to 26 factors, resolution VI up to 48 factors).

  Most of the large designs in `catlg` have been taken from Xu (2009), where complete catalogues of some scenarios are provided (cf. also his website) as well as "good" (not necessarily MA) designs for a larger set of situations. Some of the good designs by Xu (2009) have later been shown to be MA by Ryan and Bulutoglu (2010), who also found some additional larger MA designs, which are also included in `catlg`. Non-MA designs that were already available before Bulutoglu (2010) are still in the catalogue with their old name. (Note that designs that are not MA and cannot be placed in the ranking do not have a running number in the design name; for example, the MA 2048 runs design in 28 factors is named 28-17.1, the older previous design that was not MA is named 28-17 (without ".1" or another placement, because the designs position in the ranking of all designs is not known.))

There are also some non-regular 2-level fractional factorial designs of resolution V which may be interesting, as it is possible to increase the number of factors for which resolution V is possible (cf. Mee 2009, Chapter 8). These are part of package **DoE.base**, which is automatically loaded with this package. With versions higher than 0.9-14 of that package, the following arrays are available:
`L128.2.15.8.1`, which allows 4 additional factors and blocking into up to 8 blocks
`L256.2.19`, which allows just 2 additonal factors
`L2048.2.63`, which allows 16 additional factors. These non-regular arrays should be fine for most

purposes; the difference to the arrays generated by function FrF2 lies in the fact that there is partial aliasing, e.g. between 3-factor interactions and 2-factor interactions. This means that an affected 3-factor interaction is partially aliased with several different 2-factor interactions rather than being aliased either fully or not at all.

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, nclear.2fis and dominating 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.

Groemping, U. (2012). Creating clear designs: a graph-based algorithm and a catalog of clear compromise plans. *IIE Transactions* **44**, 988-1001. doi: 10.1080/0740817X.2012.654848. Early preprint at http://www1.bht-berlin.de/FB_II/reports/Report-2010-005.pdf.

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

Ryan, K.J. and Bulutoglu, D.A. (2010). Minimum Aberration Fractional Factorial Designs With Large N. *Technometrics* **52**, 250-255.

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, H., Mee, R. and Tang, B. (2012). Fractional Factorial Designs With Admissible Sets of Clear Two-Factor Interactions. *Technometrics* **54**, 191-197.

Xu, H. (2009) Algorithmic Construction of Efficient Fractional Factorial Designs With Large Run Sizes. *Technometrics* **51**, 262-277.

See Also

See also FrF2, oa.design

Examples

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

```
## inspecting a specific catalogue element
clear.2fis("9-4.2")

## 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 nfactors, 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, nfactors=12, nruns=64)
## best 10 clear 2fi designs
print(catlg, nfactors=12, nruns=64, MaxC2=TRUE)
## show alias structure
print(catlg, nfactors=12, nruns=64, MaxC2=TRUE, show.alias=TRUE)
## show best 20 designs
print(catlg, nfactors=12, nruns=64, MaxC2=TRUE, show=20)

## use vector-valued nruns
print(catlg, nfactors=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, nfactors=c(7,8), nruns=16, show=100)

## the irregular resolution V arrays from package DoE.base (from version 0.9-17)
## designs can be created from them using function oa.design
## Not run:
## not run in case older version of DoE.base does not have these
length3(L128.2.15.8.1)
length4(L128.2.15.8.1)  ## aliasing of 2fis with block factor
length4(L128.2.15.8.1[,-16])

length3(L256.2.19)
length4(L256.2.19)

##length3(L2048.2.63)
##length4(L2048.2.63) do not work resource wise
## but the array is also resolution V (but irregular)

## End(Not run)
```

CIG *Clear interactions graph from catlg entry*

Description

Function CIG creates a clear interactions graph (CIG) from a catlg design (design name must be given). Function CIGstatic allows to create a static graph from a dynamically-adjusted one.

Usage

```
CIG(design, select.catlg = catlg, nfac = NULL, static = FALSE,
   layout = layout.auto, label = "num", plot = TRUE, ...)
CIGstatic(graph, id, label = "num", xlim = c(-1,1), ylim = c(1,-1), ...)
gen2CIG(nruns, gen)
```

Arguments

design	a character string that identifies a design in the catalogue specified by option select.catlg, OR a class catlg object with a single entry, OR a formula with all main effects and the requested clear 2-factor interactions, OR a character vector of length more than one with two-letter combinations of the clear 2-factor interactions, OR a numeric two-row matrix with factor numbers of the clear 2-factor interactions, OR a character two-row matrix with factor names of the clear 2-factor interactions. The first two are for graphing design CIGs, the other ones for requirement set CIGs.
select.catlg	name of catalogue (not in quotes); only relevant, if design is a character string
nfac	number of factors; this is not needed for a class catlg object, or if the graph is supposed to show only factors that are involved in at least one interaction
static	logical. If TRUE, a static graphic is produced, otherwise an interactive graphic is created that can be modified by moving aroung nodes; only relevant for plot=TRUE
layout	ignored for static=FALSE; possible values are two-column matrices with number of rows equal to the number of vertices of the graph, or layout parameters for function plot.igraph as described in plot.common and layout; default: layout.auto (changed with version 1.6, was layout.circle before
label	in effect for catlg object only (character design name or class catlg object); a character string that decides for numeric labels or character labels; any string other than the default will invoke the factor letters as labels
plot	a logical that decides whether a plot is requested (default: TRUE); plotting can be suppressed, if graph creation is desired for calculating graph characteristics with functions from package igraph (e.g. clique.number, largest.cliques, independence.number, degree)
...	further arguments to be passed to function tkplot, or graphical parameters to be passed to plot.

graph	a graph object from package igraph, or a list whose first element is such a graph object (like the output from function CIG)
id	identification number of the interactive graph to be reproduced in static form; this number can be found in the header line of the graphics window
xlim	horizontal axis limits
ylim	vertical axis limits (per default reversed in order to exactly reproduce the interactive graph)
nruns	number of runs of the design to be graphed
gen	generator (vector of Yates matrix column numbers)

Details

The design depicted in CIG has to be the name (character string) of a regular fractional factorial 2-level design present in select.catlg.

Clear 2fis are depicted as edges in the graph. In the interactive graph, users can change the layout manually or with the menus. For example, the Reingold-Tilford layout can be chosen, with a root vertex specified; this sometimes helps in identifying groups of vertices that are not connected with each other.

Previous versions of package igraph used to internally number the vertices from 0 to number of vertices -1, not from 1 to number of vertices. This has been changed in June 2012 (FrF2 adapted to this change with version 1.5).

Function CIGstatic serves the purpose to statically create the current interactively modified graph; the usual annotation possibilities for plots are available.

Function gen2CIG returns a graph object that can be plotted or otherwise investigated with graph-related functionality.

Value

For plot=FALSE or plot=TRUE with static=TRUE, function CIG visibly (plot=FALSE) or invisibly (plot=TRUE) returns a graph from package igraph.
For plot=TRUE with static=FALSE, the function returns a list with the first element graph the element coords with the coordinates of that graph.
Function CIGstatic works on the list produced by function CIG by plotting the graph statically using the positioning from the current interactive picture.

Function gen2CIG returns a clear interactions graph that can e.g. be plotted with functions plot (plot.igraph) or tkplot.

Author(s)

Ulrike Groemping

References

Groemping, U. (2012). Creating clear designs: a graph-based algorithm and a catalog of clear compromise plans. *IIE Transactions* **44**, 988-1001. doi: 10.1080/0740817X.2012.654848. Early preprint at http://www1.bht-berlin.de/FB_II/reports/Report-2010-005.pdf.

See Also

plot.igraph, tkplot, plot.common

Examples

```
## Not run:
ex.CIG <- CIG("9-4.2", vertex.color="white", vertex.label.color="darkred")
## play around with the dynamic graph until it looks right
## look up its id number in the title bar of the graph window and use it for id
par(xpd=TRUE)
CIGstatic(ex.CIG, id=1)

## End(Not run)

graph1 <- CIG("9-4.2", plot=FALSE)   ### create graph object from design name
### calculate graph properties
require(igraph)
degree(graph1)
clique.number(graph1)
independence.number(graph1)
largest.cliques(graph1)

graph2 <- gen2CIG(32, c(7,11,14,29))   ### create graph object from generator columns
### check isomorphism to graph1
graph.isomorphic(graph1, graph2)

## Not run:
## use a CIG for manual design search
## requirement set:
estim <- compromise(9, 8:9)$requirement ## all interactions of factors 8 and 9 (H, J)
## graph the requirement set CIG
CIG(estim, vertex.color="white", vertex.label.color="darkred")
## a human can easily see that columns 1, 8 and 9 are worth a try for factors P, Q and R
CIG("9-4.1", vertex.color="white", vertex.label.color="darkred")
    ## obviously, 9-4.1 cannot accommodate the requirement set
CIG("9-4.2", vertex.color="white", vertex.label.color="darkred")
    ## 9-4.2 can, by assigning factors H and J to columns 5 and 9
## function FrF2 automatically does such matchings

## End(Not run)
```

---

compromise *Function to support estimability requests for compromise designs*

Description

Addelman (1962) and Ke and Wu (2005) discuss compromise plans of different types. Their creation is supported by the function compromise.

Usage

```
compromise(nfactors, G1, class=3, msg=TRUE)
```

Arguments

nfactors	overall number of factors
G1	vector with indices of factors in group G1 (cf. details)
class	class of compromise designs that is to be generated; 1, 2, 3, or 4, cf. details below
msg	logical stating whether the minnruns.clear element of the result should be reported in a message

Details

For compromise plans, the factors are decomposed into a group G1 and a group G2. The different classes of compromise plans require estimability of different subsets of 2fis in addition to main effects:

Class 1: all 2fis within group G1 are estimable
Class 2: all 2fis within group G1 are estimable, as well as all 2fis within group G2
Class 3: all 2fis within group G1 are estimable, as well as all 2fis between groups G1 and G2
Class 4: all 2fis between groups G1 and G2 are estimable

The function returns a list of four components (cf. section "Value"). They can be used as input for the function FrF2, if compromise plans are to be created. Both distinct designs (Addelman 1962) and clear designs (Ke, Tang and Wu 2005) can be constructed, depending on the settings of option clear in function FrF2. More explanations on specifying estimability requirements for 2fis in general are provided under estimable.2fis.

Value

Value is a list of the four components perms.full, requirement, class, and minnrun.clear. The last two components are purely imformative, while the first two provide input parameters for function FrF2.
requirement can be used for specifying the required 2fis in the estimable option, both with clear=FALSE and clear=TRUE. For clear=FALSE, perms.full can be used in the perms option for speeding up the search into a hopefully realistic time frame.
minnrun.clear indicates the minimum number of runs needed for a clear design.

Note that the catalogue catlg contains all designs needed for accomodating existing clear compromise designs in up to 128 runs (even minimum aberration among all existing clear compromise designs; for a catalogue of these, cf. Gr\"omping 2010).

Author(s)

Ulrike Groemping

References

Addelman, S. (1962). Symmetrical and asymmetrical fractional factorial plans. *Technometrics* **4**, 47-58.

Groemping, U. (2012). Creating clear designs: a graph-based algorithm and a catalog of clear compromise plans. *IIE Transactions* **44**, 988-1001. doi: 10.1080/0740817X.2012.654848. Early preprint at http://www1.bht-berlin.de/FB_II/reports/Report-2010-005.pdf.

Ke, W., Tang, B. and Wu, H. (2005). Compromise plans with clear two-factor interactions. *Statistica Sinica* **15**, 709-715.

See Also

See Also FrF2 for creation of regular fractional factorial designs as well as estimable.2fis for statistical and algorithmic information on estimability of 2-factor interactions

Examples

```
## seven factors two of which are in group G1
C1 <- compromise(7, c(2,4), class=1)
C1$perms.full  ## the same for all classes
C1$requirement
C2 <- compromise(7, c(2,4), class=2)
C2$requirement
C3 <- compromise(7, c(2,4), class=3)
C3$requirement
C4 <- compromise(7, c(2,4), class=4)
C4$requirement

## Not run:
########## usage of estimable ############################
  ## design with with BD clear in 16 runs
  FrF2(16,7,estimable = C1$requirement)
  ## design with BD estimable on a distinct column in 16 runs (any design will do,
  ##    if resolution IV!!!
  FrF2(16,7,estimable = C1$requirement, clear=FALSE, perms=C1$perms.full)
  ## all four classes, mostly clear, for 32 runs
  FrF2(32,7,estimable = C1$requirement)
  FrF2(32,7,estimable = C2$requirement)   ## requires resolution V
         ## as clear class 2 compromise designs do not exist due to Ke et al. 2005
  FrF2(32,7,estimable = C2$requirement, clear=FALSE, perms=C2$perms.full)
  FrF2(32,7,estimable = C3$requirement)
  FrF2(32,7,estimable = C4$requirement)
  ## two additional factors H and J that do not show up in the requirement set
  FrF2(32,9,estimable = C3$requirement)
  ## two additional factors H and J that do not show up in the requirement set
  FrF2(32,9,estimable = C3$requirement, clear=FALSE)
  ## note that this is not possible for distinct designs in case perms is needed,
  ## because perms must have nfactors columns

## End(Not run)
```

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	cf. eff3
eff2	cf. eff3
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 obj linear model object (character strings)
main	title for the plot, respnam is the name of the response variable as determined from the call
cex.title	multiplier for size of overall title (cex.main is multiplied with this factor)
cex.ax	size of axis tick marks, defaults to cex.axis-parameter
cex.lab	size of axis labels
cex.clab	size of corner labels
size	size of cube corners
round	optional rounding of corner labels (digits argument for function round, e.g. round=0 for integers, round=-1 for multiples of 10, round=1 for 1 decimal place
abbrev	number of characters shown for factor levels
y.margin.add	adjustment parameter for placement of y-axis labeling
modeled	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 packge **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, ..., response = NULL)
## 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, subtitle=NULL, ...)
```

Arguments

fit	an experimental design of class design with the type element of the design.info attribute containing "FrF2" or "pb" OR object of class lm. Fitted model from lm or aov.
...	further arguments to be passed to the default function, or graphical parameters to be passed to plot; note that one should not use pch for split-plot designs.
response	NULL or a character string that specifies response variable to be used, must be an element of response.names(obj); if NULL, the first response from response.names(obj) is used
code	logical. If TRUE labels "A","B", etc. are used instead of the names of the coefficients (factors). A legend linking codes to names is provided.
autolab	If TRUE, only the significant factors according to the Lenth method (significance level given by alpha) are labelled.
alpha	significanc level for the Lenth method
faclab	NULL or list. If NULL, point labels are automatically determined according to the setting of code (i.e. A,B,C etc. for code=TRUE, natural effect names otherwise) and autolab (i.e. all effects are labelled if autolab=FALSE, only significant effects are labelled if autolab=TRUE). Otherwise, faclab can be used for manual labelling of certain effects and should be a list with idx (integer vector referring to position of effects to be labelled) and lab (character vector of labels) components.
block	logical. If TRUE, the first factor is labelled as "BK" (block).
datax	logical. If TRUE, the x-axis is used for the factor effects the the y-axis for the normal scores. The opposite otherwise.
half	logical. If TRUE, half-normal plot of effects is display.
pch	numeric or character. Points character.
cex.fac	numeric. Factor label character size.
cex.lab	numeric. Labels character size.
cex.pch	numeric. Points character size.
cex.legend	numeric. Legend size in case of codes.
main	NULL or character. Title of plot. If NULL, automatic title is generated.
subtitle	NULL or character. Sub title of plot. Should not be used for split-plot designs, because automatic subtitle is generated for these.

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.

For split-plot designs, whole plot effects are shown as different plotting characters, because they are potentially subject to larger variability, and one should not be too impressed, if they look impressively large, as this may well be indication of plot-to-plot variability rather than a true effect.

Value

The function invisibly returns a data frame with columns: x, y, no, effect, coded (if coded plot was requested) and pchs, for the coordinates, the position numbers, the effect names, the coded effect names, and the plotting characters for plotted points.

The plotting characters are particularly useful for split-plot designs and can be used for subsequent separate plotting of whole-plot and split-plot effects, if necessary.

Note

If you load package **BsMD** after package **FrF2**, a mere call to function DanielPlot will use the function from package **BsMD** rather than the one from package **FrF2**. You can explicitly request usage of the **FrF2** function by FrF2::DanielPlot.

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, halfnormal, LenthPlot, BsMD-package

`estimable.2fis` *Statistical and algorithmic aspects of requesting 2-factor interactions to be estimable in FrF2*

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. Whenever a resolution V or higher design is available, this option is unnecessary, because all 2fis are estimable in the sense that they are not aliased with any main effect or any other 2fi. If resolution V or higher is not affordable, the option `estimable` can ensure that certain 2fis can nevertheless be estimated.

Per default, it is assumed that a resolution IV design is required, 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 `FrF2` (cf. Groemping 2010).

Let us first consider 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**. For details of this algorithm, cf. Groemping (2012). With the catalogue `catlg` available in this package, the best (minimum aberration) existing clear designs are guaranteed to be found for up to 64 runs and have a good chance to be found for 128 runs. For 128 runs, it is possible to load an additional large catalogue (package **FrF2.catlg128**) in order to also guarantee that the best clear design is found. For 256 and 512 runs, only one or two resolution IV designs of each size are catalogued so that option `estimable` can try to influence allocation of factors to columns, but may fail although an appropriate clear design would exist outside the catalogued designs.

The search for a clear design is often fast. If it isn't, option `sort` of function `FrF2` can help. For the occasional situation where this doesn't help either, a manual search may help, see `CIG` for an example of how to proceed.

Since version 2 of package **FrF2**, requesting 2fis to be clear is compatible with blocking a design. The algorithm behind that functionality is based on Godolphin (2021) and is described in Groemping (2021). The default implementation strives for a guartanteed and best possible result. Arguments `firsthit` and `useV` to function `FrF2` can be used for trying to obtain a possibly not best

result (firsthit) faster or to use a (sometimes) faster algorithm that is not guaranteed to deliver a result even though it might exist for resolution IV situations (useV=FALSE).

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).

With clear=TRUE, the algorithm loops through the eligible designs from catlg.select from good to worse (in terms of MA) and, for each design, uses a subgraph isomorphism check from package igraph. There are two such algorithms, VF2 (the default, Cordella et al. 2001) and LAD (introduced with version 1.7 of package **FrF2**, Solnon 2010), which can be chosen with the method option. Run times of the subgraph isomorphism search are often fast, but can also be very very slow in unlucky situations. Where the VF2 algorithm is particularly slow, the LAD algorithm is often fast (see Groemping 2014b). Especially for the VF2 algorithm, run times may strongly depend on the ordering of factors, which can be influenced by the option sort. As the slowness of the process is intrinsic to the subgraph isomorphism search problem (which is NP-complete), a max.time option analogous to the clear=FALSE situation would be of very limited use only and is therefore not available. Instead, it is possible to have a look at the number of the design that was in the process of being searched when the process was interrupted (with the command FrF2.currentlychecked()).

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`.

In order to support relatively comfortable creation of distinct designs of some frequently-used types of required interaction patterns, the function `compromise` has been divised: it supports creation of the so-called compromise plans of classes 1 to 4 (cf. e.g. Addelman 1962; Ke, Tang and Wu 2005; Groemping 2012). The list it returns also contains a component `perms.full` that can be used as input for the `perms` option.

Please contact me with any suggestions for improvements.

Author(s)

Ulrike Groemping

References

Addelman, S. (1962). Symmetrical and asymmetrical fractional factorial plans. *Technometrics* **4**, 47-58.

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.

Cordella, L.P., Foggia, P., Sansone, C. and Vento, M. (2001). An improved algorithm for matching large graphs. *Proc. of the 3rd IAPR TC-15 Workshop on Graphbased Representations in Pattern Recognition*, 149–159.

Godolphin, J. (2021). Construction of Blocked Factorial Designs to Estimate Main Effects and Selected Two-Factor Interactions. *J. Royal Statistical Society* **B 83**, 5-29. doi: 10.1111/rssb.12397.

Groemping, U. (2010). "Clear" and "Distinct": two approaches for regular fractional factorial designs with estimability requirements. *Reports in Mathematics, Physics and Chemistry*, report 02/2010, Department II, Beuth University of Applied Sciences Berlin. http://www1.bht-berlin.de/FB_II/reports/Report-2010-002.pdf.

Groemping, U. (2012). Creating clear designs: a graph-based algorithm and a catalogue of clear compromise plans. *IIE Transactions* **44**, 988–1001. Early preprint available at http://www1.bht-berlin.de/FB_II/reports/Report-2010-005.pdf.

Groemping, U. (2014a). R Package FrF2 for Creating and Analyzing Fractional Factorial 2-Level Designs. *Journal of Statistical Software*, **56**, Issue 1, 1-56. https://www.jstatsoft.org/v56/i01/.

Groemping, U. (2014b). A Note on Dominating Fractional Factorial Two-Level Designs With Clear Two-Factor Interactions. *Technometrics* **56**, 42–45.

Groemping, U. (2021). An algorithm for blocking regular fractional factorial 2-level designs with clear two-factor interactions. *Computational Statistics and Data Analysis* **153**, 1-18. doi: 10.1016/j.csda.2020.107059. Preprint at Report 3/2019.

Ke, W., Tang, B. and Wu, H. (2005). Compromise plans with clear two-factor interactions. *Statistica Sinica* **15**, 709-715.

Solnon, C. (2010). AllDifferent-based Filtering for Subgraph Isomorphism. *Artificial Intelligence* **174**, 850–864.

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, catlg for the Chen, Sun, Wu (1993) and larger catalogues of designs and some accessor functions, and function compromise for a convenience function to handle estimability requests for compromise plans

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 for the clear=FALSE case
## 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]))
  ## function compromise will calculate the necessary perms entries automatically
  compromise(11,1:6)$perms.full
  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

`design`	a data frame of class design that contains a 2-level fractional factorial; currently, `design` must neither be blocked nor a long version parameter design
`columns`	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.
`...`	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 dealiases all confounding relations for that factor, folding on two particular factors dealiases 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 log2(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 resonse 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)

## fold a Plackett-Burman design with 11 factors
plan <- pb(12)
fold.design(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, method="VF2", sort="natural",
        ignore.dom = !isTRUE(all.equal(blocks,1)),
        useV = TRUE, firsthit=FALSE, 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", block.old=FALSE,
        force.godolphin=alias.block.2fis,
        bbreps=replications, wbreps=1,
        alias.block.2fis = FALSE,
        hard = NULL, check.hard=10, WPs=1,nfac.WP=0,
        WPfacs=NULL, check.WPs = 10, ...)
FrF2.currentlychecked()
```

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 accomodates `nfactors` factors. If the smallest possible design is a full factorial or not catalogued, the function stops with an error. If `generators` is specified, `nruns` is required. If estimable is specified and `nruns` omitted, `nruns` becomes the size of the smallest design that MIGHT accomodate the effects requested in `estimable`. If this run size turns out to be too low, an error is thrown. In that case, explicitly choose `nruns` as twice the run size given in the error message and retry.
nfactors	is the number of 2-level factors to be investigated. It can be omitted, if it is obvious from `factor.names`, a specific catalogued design given in `design`, `nruns` together with `generators`, or `estimable`.

If estimable is used for determining the number of factors, it is assumed that the largest main effect position number occurring in estimable coincides with nfactors.

For blocked designs, block generator columns are not included in nfactors, 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).

For automatically-generated split-plot designs (cf. details section), nfactors simply is the number of all factors (whole plot and split plot together). If nfac.WP < log2(WPs), the algorithm will add (an) artificial plot generation factor(s).

For manually-specified split-plot designs (through options generators or design together with WPfacs), the user must specify at least log2(WPs) split plot factors, i.e. nfac.WP >= log2(WPs) is required (and must, if necessary, be achieved by adding log2(WPs) - nfac.WP extra independent generator columns for the whole plot structure, which have to be counted in nfac.WP and nfactors).

factor.names — a character vector of nfactors factor names or a list with nfactors 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 factor.names, the levels given in default.levels are used;
Default factor names are the first elements of the character vector Letters, or the factors position numbers preceded by capital F in case of more than 50 factors.

default.levels — default levels (vector of length 2) for all factors for which no specific levels are given

ncenter — number of center points per block; ncenter > 0 is permitted, if all factors are quantitative and the design is not a split-plot design

center.distribute

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 randomized designs, and appended to the end for non-randomized designs. for more detail, see function add.center, which does the work.

generators — There are log2(nruns) 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.

generators can be

a list of vectors with position numbers of base factors (e.g. c(1,3,4) 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. `names(Yates)[1:15]` for a 16 run design.

In all cases, preceding the respective entry with a minus sign (e.g. -c(1,3,4), "-ACD", -13) implies that the levels of the respective column are reversed.
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.

`design` is a character string specifying the name of a design listed in the catalogue specified as `select.catlg`, which is usually the catalogue `catlg`

`resolution` is the arabic numeral for the requested resolution of the design. `FrF2` looks for a design with at least this resolution. Option `resolution` does not work, if `estimable`, `blocks` or `WPs` are specified, and neither if `nruns` is given.
A design with resolution III (resolution=3) 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.

`select.catlg` specifies a catalogue of class `catlg` from which an adequate design is selected and adapted.
The specified catalogue is used for design construction, unless `generators` explicitly constructs a non-catalogued design. The default `catlg` is adequate for most applications.

If a specific different catalogue of designs is available, this can be specified here.
Specification of a smaller subset of designs is useful, if `estimable` has been given and `clear=FALSE`, for restricting the search to promising designs.

Names of catalogues from package **FrF2.catlg128** can be given here without prior loading of that package; loading of the package and the selected catalogue will then happen automatically, provided the package is installed (for version >=1.2 of package **FrF2.catlg128**; for earlier versions, the suitable catalogue has to be manually loaded using the `data()` command).

`estimable` indicates the 2-factor interactions (2fis) that are to be estimable in the design. Consult the specific help file (`estimable.2fis`) for details of two different approaches of requesting estimability, as indicated by the status of the `clear` option. `estimable` cannot be specified together with `splitplot`, `generators` or `design`.

`estimable` can be
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
OR
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`.

`clear` logical, indicating how estimable is to be used. See `estimable.2fis`.

`method` character string indicating which subgraph isomorphism search routine of package **igraph** is used (`graph.subisomorphic.vf2` or `graph.subisomorphic.lad`). The default `"VF2"` uses VF2 algorithm by Cordella et al. (2001), which was the only available algorithm before version 1.7 of package FrF2. The alternative `"LAD"` uses the LAD algorithm by Solnon (2010), which was reported to be substantially faster than VF2 especially for some notoriously difficult VF2 cases (see Gr?mping 2014b). This option is relevant for `estimable` with `clear=TRUE` only.
NOTE: The resulting design may be different for different settings of this option!

`sort` character string indicating how the estimability requirement and the candidate design clear 2fis are handed to the subgraph isomorphism search routine of package **igraph**. The default `"natural"` leaves them in unchanged order (like in **FrF2** versions up to 1.6). `sort="high"` and `sort="low"` sort both requirement set and candidate design graph according to vertex degrees (high first or low first).

This option is relevant for `estimable` with `clear=TRUE` only.
It has been added, because pre-sorting of vertices sometimes speeds up the search by several orders of magnitude especially for the `VF2` method.
NOTE: The resulting design may be different for different settings of this option!

`ignore.dom` logical, default FALSE for unblocked designs, TRUE for blocked designs; if TRUE, `estimable` ignores the dominating attribute of the catalogue entries; can be useful for searching a blocked design with estimable 2fis from a reduced catalogue

`useV` NULL or logical; relevant for designs with `blocks` and `estimable` only; if TRUE, function `link{colpick}` is used (default), otherwise function `link{colpickIV}`; if set to NULL, `colpick` is used for fractions with resolution at least V and function `link{colpickIV}` for resolution IV fractions;
`useV=FALSE` does the subgraph isomorphism check once only, at the expense of missing out opportunities; it may be worth trying `useV=NULL` or `useV=FALSE` for resolution IV situations, for which the search takes very long with the default

`firsthit` logical; relevant for designs with `blocks` and `estimable` only; if FALSE, the function tries to find a design with as many as possible clear 2fis, otherwise the function stops at the first possible blocking; in case of resource problems, setting `firsthit` to TRUE may be helpful

`res3` logical; if TRUE, `estimable` includes resolution III designs into the search for adequate designs; otherwise resolution IV and higher designs are included only.

`max.time` maximum time for design search as requested by `estimable`, in seconds (default 60); introduced for `clear=FALSE` situations because the search can take a long time in complicated or unlucky situations; set max.time to Inf 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 `estimable.2fis`)

`perm.start` used with `estimable` specified, and `clear=FALSE`.
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.

`perms` used with `estimable` specified, and `clear=FALSE`.
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 `perms` whenever possible, if `clear=FALSE`, since this dramatically improves performance of the algorithm.

It is planned to automatically generate perms for certain structures like compromise designs in the (not so near) future.

`MaxC2` is a logical and defaults to FALSE. If TRUE, maximizing the number of clear 2-factor interactions takes precedence over minimizing aberration. Resolution is always considered first. `MaxC2` is ignored when using the `estimable` option. `MaxC2=TRUE` is not a recommended choice.

`replications` positive integer number. Default 1 (i.e. each row just once). If larger, each design run is executed replication times. If `repeat.only`, 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).

`repeat.only` logical, relevant only if replications > 1. If TRUE, replications of each run are grouped together (repeated measurement rather than true replication). The default is `repeat.only=FALSE`, i.e. the complete experiment is conducted in `replications` blocks, and each run occurs in each block.

`randomize` 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 `repeat.only`.

`seed` optional seed for the randomization process
In R version 3.6.0 and later, the default behavior of function `sample` has changed. If you work in a new (i.e., >= 3.6.-0) R version and want to reproduce a randomized design from an earlier R version (before 3.6.0), you have to change the RNGkind setting by
`RNGkind(sample.kind="Rounding")`
before running function `FrF2`.
It is recommended to change the setting back to the new recommended way afterwards:

RNGkind(sample.kind="default")
For an example, see the documentation of the example data set VSGFS.

alias.info — can be 2 or 3, gives the order of interaction effects for which alias information is to be included in the aliased component of the design.info element of the output object.

blocks — 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 of Yates column numbers OR a list of generators similar to list entries for generators.
In the latter case, the differences to generators 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* nfactors 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-1 treatment factors, but should usually have fewer than that.

If the experiment is randomized, randomization happens within blocks. In case of many blocks, units should also be randomized to blocks wherever possible!

For the statistical and algorithmic background of blocked designs, see block.

block.name — name of the block factor, default "Blocks"

block.old — logical; if TRUE, blocking behavior of FrF2 version 1.7.2 is activated

force.godolphin — logical; if TRUE, blocking is forced to be done with the Godolphin method (using function colpick, see block), even if the default would have been to use function blockpick.
The Godolphin method was introduced with package version 2.0, and since package version 2.3, the default for alias.block.2fis=TRUE is to force the use of the Godolphin method.

bbreps — between block replications; these are always taken as genuine replications, not repeat runs; default: equal to replications; CAUTION: you should not modify bbreps if you do not work with blocks, because the program code uses it instead of replications in some places

wbreps — within block replications; whether or not these are taken as genuine replications depends on the setting of repeat.only

alias.block.2fis
: logical indicating whether blocks may be aliased with 2fis (default: FALSE);
it will often be necessary to modify this option, because there is otherwise no solution.

hard
: gives the number of hard to change factors. These must be the first factors in factor.names. 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).
Observations within whole plots are randomized, whole plots themselves are not randomized (contrary to split plot designs).
From the statistical point of view, randomisation of whole plots is strongly preferrable (cf. splitplot for a brief discussion of the difference). If this appears feasible, you may want to explicitly handle the situation by treating the hard to change factors as whole-plot factors via WPs and nfac.WP.

check.hard
: is the number of candidate designs from the catalogue specified in select.catlg 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).
If you want to use the best design and do not want to compromise the confounding structure for ease-of-change reasons, set check.hard to 1.

WPs
: is the number of whole plots and must be a power of 2.
If WPs > 1, at least one of nfac.WP or WPfacs must be given.
If WPs = 1, all settings for split-plot related options are ignored.

For statistical and algorithmic information on treatment of split-plot designs see the separate help file splitplot.

nfac.WP
: is the number of whole plot factors and must be smaller than WPs.
The nfac.WP whole plot factors are counted within nfactors. Per default, the first nfac.WP factors are the whole plot factors.
If a design is provided and whole plot factors are manually provided (design or generators option together with WPfacs), nfac.WP can be omitted (i.e. remains 0). If given, it must coincide with the length of WPfacs.
If nfac.WP is given without WPfacsonly, generators must not be given.
If nfac.WP is omitted (i.e. remains 0) and WPs > 1, 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).

WPfacs
: is per default NULL. In this case, the first nfac.WP factors are considered whole plot factors (and are, if necessary, automatically supplemented by additional whole plot constructor factors).
If WPfacs is given, it must specify at least log2(WPs) whole plot factors. A custom design must be specified with options design or generators, and the requested factors in WPfacs must indeed create a split-plot structure with WPs whole plots. If the number of whole plots created by the whole plot factors differs from WPs or factors other than the specified factors from WPfacs would in fact become whole plot factors as well, an error is thrown.

WPfacs can be any of
a vector or list of factor position numbers
OR
a character vector of factor position letters from Letters
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 nfac.WP are interpreted as factor names, not position letters).
It is not possible to specify additional whole plot generators from interaction effects manually through WPfacs. Rather, all whole plot factors - even artificial ones needed only to increase the number of plots - need to be included in the design factors.

check.WPs — is the number of potential split-plot designs that are compared by function splitpick w.r.t. resolution of the whole plot portion of the design. This option is effective, if nfac.WP>k.WP (i.e. bad resolution possible) and nfac.WP 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).

... — currently not used

Details

Per default, the function picks the best design from the default design catalogue 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.

Function FrF2.currentlychecked is meant as a diagnostic tool, when searching for designs with option estimable and clear=TRUE. If the search takes very long, it can be interrupted (CAUTION: in some igraph versions, interrupting the search may crash R). After a successful interruption, and FrF2.currentlychecked() returns a character string with the name of the design that was checked at the time of interruption.

Value

Function FrF2 returns a data frame of S3 class design that 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:

desnum — Design matrix in -1/1 coding

run.order — a three column data frame;
the first column (run.no.in.std.order) contains the run number in standard order, possibly including a block number and in that case also a number for the original position within the block,
the second column (run.no) contains the actual run number as randomized,
the third column contains (run.no.std.rp) contains the content of the first column accumulated with a replication identifier, if applicable.
A few remarks on the run number in standard order are needed here:
In blocked and split plot designs, the run number in standard order refers to a row ordering with the first base factor changing *slowest*, different from the usual order with the first base factor changing *fastest*. Also note that the run number in standard order may not refer to the base columns one would naturally expect for designs created with blocking, estimable 2fis or split plot designs; the elements map and/or orig.fac.order of the attribute design.info help identify which base factors drive the run number in standard order. (In case of option hard=TRUE, the remark on split plot designs applies, and the numbering refers to the special slow change matrix from Cheng et al. 1998.)
Before version 2 of package FrF2, blocking for large situations was internally done with function blockpick.big. This behavior can be reproduced using the argument block.old=TRUE; if blocking for a large design is not successful otherwise, block.old=TRUE might be worth a try. For blocked designs created internally with function blockpick.big, the run number in standard order is not easily related to the final design.

design.info — list with the entries

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 FrF2.blocked (nfactors is now equal to ntreat) and for designs of type FrF2.splitplot, where nfactors is now the sum of nfac.WP and nfac.SP.

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

nfac.WP for designs of type FrF2.splitplot 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 FrF2.splitplot only;
number of split-plot factors

nlevels for designs of type full factorial only;
vector with number of levels for each factor (of course, all the nfactors entries are "2" for FrF2)

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

FrF2.version version number of package FrF2, supporting correct usage of FrF2-specific functionality in functions summary and generators methods for class design

nblocks for designs of type FrF2.blocked 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 blocks argument specifies certain factor combinations for blocking; ;
for designs of type FrF2.blocked only

blocksize for designs of type FrF2.blocked only;
size of each block (without consideration of wbreps)

nWPs for designs of type FrF2.splitplot only;
number of whole plots

plotsize for designs of type FrF2.splitplot only;
size of each plot (without consideration of repeat.only replications if applicable)

orig.fac.order designs of type FrF2.splitplot only;
factor order of the design before reshuffling of factors in order to accomodate the split plot request; the run number in standard order can only be interpreted together with this information

catlg.entry for designs of type FrF2 only;
list with one element, which is the entry of catlg on which the design is based

generators for designs of type FrF2.generators only;
character vector of generators in the form D=ABC 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` and for split-plot designs

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

Warning

Since R version 3.6.0, the behavior of function `sample` has changed (correction of a biased previous behavior that should not be relevant for the randomization of designs). For reproducing a randomized design that was produced with an earlier R version, please follow the steps described with the argument `seed`.

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.

Cordella, L.P., Foggia, P., Sansone, C. and Vento, M. (2001). An improved algorithm for matching large graphs. *Proc. of the 3rd IAPR TC-15 Workshop on Graphbased Representations in Pattern Recognition*, 149–159.

Godolphin, J. (2021). Construction of Blocked Factorial Designs to Estimate Main Effects and Selected Two-Factor Interactions. *J. Royal Statistical Society* **B 83**, 5-29. doi: 10.1111/rssb.12397.

Groemping, U. (2012). Creating clear designs: a graph-based algorithm and a catalogue of clear compromise plans. *IIE Transactions* **44**, 988–1001. Early preprint available at http://www1.bht-berlin.de/FB_II/reports/Report-2010-005.pdf.

Groemping, U. (2014a). R Package FrF2 for Creating and Analyzing Fractional Factorial 2-Level Designs. *Journal of Statistical Software*, **56**, Issue 1, 1-56. https://www.jstatsoft.org/v56/i01/.

Groemping, U. (2014b). A Note on Dominating Fractional Factorial Two-Level Designs With Clear Two-Factor Interactions. *Technometrics* **56**, 42–45.

Groemping, U. (2021). An algorithm for blocking regular fractional factorial 2-level designs with clear two-factor interactions. *Computational Statistics and Data Analysis* **153**, 1-18. doi: 10.1016/j.csda.2020.107059. Preprint at Report 3/2019.

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.

Solnon, C. (2010). AllDifferent-based Filtering for Subgraph Isomorphism. *Artificial Intelligence* **174**, 850–864.

Sun, D.X., Wu, C.F.J. and Chen, Y.Y. (1997). Optimal blocking schemes for 2^n 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
FrF2Large for regular fractional factorial designs with more than 4096 runs (these are not supported by a design catalogue, except for a few resolution V designs which have not been checked for any optimality among the resolution V designs),
pb for non-regular fractional factorials according to Plackett-Burman,
catlg for the underlying design 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","MaterialType","Gain","ScreenAngle","ScreenVibLevel"))
## isomorphic non-catalogued design as basis, using Godolphin approach
FrF2(16,gen=c(7,11,14),blocks=4,alias.block.2fis=TRUE)
## isomorphic non-catalogued design as basis, not using Godolphin approach
## (different design of comparable quality in this case)
FrF2(16,gen=c(7,11,14),blocks=4,alias.block.2fis=TRUE, force.godolphin=FALSE)
## FrF2 uses blockpick.big and ignores the generator
FrF2(64,gen=c(7,11,14),blocks=16,alias.block.2fis=TRUE, block.old=TRUE)
## FrF2 uses Godolphin approach, regardless of force.godolphin argument
##      because the setting is large
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"))

########## blocked design with estimable 2fis ####################
### all interactions of last two factors to be estimable clearly
### in 64 run design with blocks of size 4
### not possible with catalogue entry 9-3.1
FrF2(64, 6, blocks=16, factor.names=Letters[15:20],
                estimable=compromise(6,3)$requirement,
                alias.block.2fis=TRUE, randomize=FALSE)
FrF2(design="9-3.2", blocks=16, alias.block.2fis=TRUE,
    factor.names = list(C1="",C2="",C3="",C4="",C5="",C6="",C7="",
    N1=c("low","high"),N2=c("low","high")),
    default.levels = c("current","new"),
    estimable=compromise(9, 8:9)$requirement)
FrF2(256, 13, blocks=64, alias.block.2fis=TRUE,
    factor.names = list(C1="",C2="",C3="",C4="",C5="",C6="",C7="",C8="",
    N1=c("low","high")),
    default.levels = c("current","new"),
    estimable=compromise(13, 1)$requirement)

########## 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.WP=3,
      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)

## End(Not run)
## example for the sort option added with version 1.6-1
## and for usage of a catalogue from package FrF2.catlg128 (simplified with version 1.6-5)
  ## Not run:
 estim <- compromise(17,15:17)$requirement  ## all interactions of factors 15 to 17 (P,Q,R)
  ## VF2 algorithm without pre-sorting of vertices
  ### CAUTION: in some igraph versions, the following may crash R ###
  FrF2(128, 17, estimable=estim, select.catlg=catlg128.17)
                        ## very slow, interrupt with ESC key after a short while
                        ## !!! save all important work before, in case R crashes
  FrF2.currentlychecked()   ## displays the design that was currently checked
                            ## should be 17-10.2407, if the interrupt was successful
  ## VF2 algorithm with pre-sorting of vertices
  FrF2(128, 17, estimable=estim, sort="high", select.catlg=catlg128.17)  ## very fast
  FrF2(128, 17, estimable=estim, sort="low", select.catlg=catlg128.17)  ## very fast
  ## LAD algorithm
  FrF2(128, 17, estimable=estim, method="LAD", select.catlg=catlg128.17)  ## very fast
  ## guaranteed to be MA clear design
  ## only works, if package FrF2.catlg128 is installed

## End(Not run)

## 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)
  ## 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)
```

FrF2Large *Function to provide large (at least 8192 runs) regular Fractional Factorial designs that are not necessarily optimal, especially large resolution V designs.*

Description

Large regular fractional factorial 2-level designs in 8192 or more runs are provided: Resolution V designs in 8096 to 32768 runs with up to 120 factors according to the suggestion by Sanchez and Sanchez 2005 are automatically created (these are not necessarily optimal). Furthermore, manual generation of large regular fractional factorial designs via specification of generators is possible.

Usage

```
FrF2Large(nruns, 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,
    replications = 1, repeat.only = FALSE,
    randomize = TRUE, seed = NULL, alias.info = 2, ...)
nrunsV(nfactors)
```

Arguments

nruns	Number of runs, must be a power of 2 (8192 to 32768). The number of runs must match the number of factors. Function nrunsV can be used for determining the number of runs needed for a resolution V design, and for advice on the function to be used. For more detail on specification of the number of runs, see the Details section.
nfactors	is the number of 2-level factors to be investigated. It can be omitted, if it is obvious from options factor.names or generators. The number of factors must match the length of generators.
factor.names	a character vector of nfactors factor names or a list with nfactors 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 factor.names, the levels given in default.levels are used; Default factor names are the first elements of the character vector Letters, or the factors position numbers preceded by capital F in case of more than 50 factors.

`default.levels` default levels (vector of length 2) for all factors for which no specific levels are given

`ncenter` number of center points per block; `ncenter > 0` is permitted, if all factors are quantitative and the design is not a split-plot design

`center.distribute`
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 randomized designs, and appended to the end for non-randomized designs. for more detail, see function `add.center`, which does the work.

`generators` There are `log2(nruns)` base factors the full factorial of which spans the design (e.g. 10 for 1024 runs). The generators specify how the remaining factors are to be allocated to interactions of these.
WARNING: Of course, with manual specification of generators, the structure of the design is in the users responsibility; the function only prevents confounding of two main effects with each other.

`generators` can be

a list of vectors with position numbers of base factors (e.g. c(1,3,4) 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. `names(Yates)[1:15]` for a 16 run design.

WARNING: Contrary to function `FrF2`, it is not possible to precede generator entries with a minus sign for reversing column levels; instead, the levels must be swapped.

`replications` positive integer number. Default 1 (i.e. each row just once). If larger, each design run is executed replication times. If `repeat.only`, 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).

`repeat.only` logical, relevant only if replications > 1. If TRUE, replications of each run are grouped together (repeated measurement rather than true replication). The default is `repeat.only=FALSE`, i.e. the complete experiment is conducted in `replications` blocks, and each run occurs in each block.

`randomize` 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 `repeat.only`.

seed	optional seed for the randomization process In R version 3.6.0 and later, the default behavior of function sample has changed. If you work in a new (i.e., >= 3.6.-0) R version and want to reproduce a randomized design from an earlier R version (before 3.6.0), you have to change the RNGkind setting by RNGkind(sample.kind="Rounding") before running function FrF2Large. It is recommended to change the setting back to the new recommended way afterwards: RNGkind(sample.kind="default") For an example, see the documentation of the example data set VSGFS.
alias.info	can be 2 or 3, gives the order of interaction effects for which alias information is to be included in the aliased component of the design.info element of the output object.
...	currently not used

Details

If generators are not explicitly specified, function FrF2Large creates a resolution V design according to the rules by Sanchez and Sanchez (2005) for the specified number of factors in the specified number of runs. The Sanchez and Sanchez article offers designs with
at least 1024 runs for 25 to 29 factors (1024 up to 33 factors with FrF2),
at least 2048 runs for 30 to 38 factors (2048 up to 47 factors with FrF2),
at least 4096 runs for 39 to 52 factors (4096 up to 65 factors with FrF2),
at least 8192 runs for 53 to 69 factors (up to 65 factors in half the run size with FrF2),
at least 16384 runs for 70 to 92 factors, (
at least 32768 runs for 93 to 120 factors.
For designs with up to 4096 runs, function FrF2 creates better automatic designs. Therefore, function FrF2Large is restricted to usage for larger designs.

Users can explicitly specify a design through specifying generators via the generators option. For up to 4096 runs, this is also possible with function FrF2, even with more flexibility. Therefore, manual design generation with function FrF2Large is also restricted to designs of at least 8192 runs.

Manual generation of large designs with the option generators is limited by computer memory only. nruns must be at least large enough to accomodate the rightmost generator column; for example, if generators contains an element ABEP, P is the 15th base factor (15th letter in Letters), i.e. nruns must be at least 2^15=32768; if the largest generator column number in Yates column notation is 4201, nruns must be at least 2^ceiling(log2(4201))=8192.

Value

Function nrunsV invisibly returns the number of runs requested and prints a message with the number of runs and the appropriate function.

Function FrF2Large returns 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:

`desnum` 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 the entries

type character string "FrF2.large"

nruns number of runs (replications are not counted)

nfactors number of factors

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

generators for designs of type `FrF2.generators` only; character vector of generators in the form D=ABC etc.

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.)

replications option setting in call to `FrF2`

repeat.only option setting in call to `FrF2`

randomize option setting in call to `FrF2`

seed option setting in call to `FrF2`

creator call to function FrF2Large; (in future, may also contain stored menu settings from R commander plugin **RcmdrPlugin.DoE**, once the function has been implemented in that package)

FrF2.version version number of package FrF2, supporting correct usage of `FrF2`-specific functionality in functions `summary` and `generators` methods for class `design`

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

Warning

Since R version 3.6.0, the behavior of function `sample` has changed (correction of a biased previous behavior that might be relevant for the randomization of very large designs). For reproducing a randomized design that was produced with an earlier R version, please follow the steps described with the argument `seed`.

Author(s)

Ulrike Groemping

References

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

Sanchez, S.M. and Sanchez, P.J. (2005). Very Large Fractional Factorial and Central Composite Designs. *ACM Transactions on Modeling and Computer Simulation* **15**, 362-377.

See Also

See also FrF2 for smaller regular fractional factorials and oacat for two non-regular resolution V fractional factorials (reported e.g. by Mee 2009) for up to 19 factors in 256 runs or up to 63 factors in 2048 runs

Examples

```
## numbers of runs needed for resolution V designs in different numbers of factors
nrunsV(8)
nrunsV(18)
needed <- nrunsV(27)
needed
nrunsV(65)
nrunsV(71)

## Not run:
plan <- FrF2Large(nrunsV(75),75)
summary(plan)

## End(Not run)
```

---

godolphin *Functions in support of Godolphin's approach for blocking designs*

Description

Function colpick handles the creation of X matrices for blocking, function FF_from_X blocks a full factorial, function X_from_profile creates an X matrix from a profile, function phimax calculations the maximum number of clear 2fis from Godolphin's approach. Further helper functions support the use of the method. The functions are meant for expert users only.

Usage

```
colpick(design, q, all = FALSE, select.catlg = catlg,
      estimable = NULL, method = "VF2", sort = "natural",
      res3 = FALSE, all0 = FALSE, quiet = FALSE,
      firsthit = is.numeric(design))
FF_from_X(X, randomize = TRUE, seed = NULL, alias.info=2)
X_from_profile(n, q, profile = NULL)
clear2fis_from_profile(n, q, profile = NULL)
X_from_parts(n, q, parts)
```

```
phimax(n, q, profile = NULL)
blockgencreate(X, p = 0)
Xcalc(XI, gen)
blockgengroup(X, p = 0, num = FALSE)
colpickIV(design, q, all = FALSE, select.catlg = catlg,
      estimable = NULL, method = "VF2", sort = "natural",
      res3 = FALSE, all0 = FALSE, quiet = FALSE,
      firsthit = is.numeric(design))
```

Arguments

design	a character string that identifies a design in the cataloge specified by option select.catlg, OR a class catlg object with a single entry (of longer, only the first one is used), OR an integer number of factors for which a full factorial is assumed.
q	the requested block size is 2^q
all	if TRUE (default FALSE), all possible **X** matrices are returned; otherwise, colpick returns the first successful one (if estimable is not NULL) or the best one (otherwise)
select.catlg	name of catalogue (not in quotes); only relevant, if design is a character string
estimable	a specification of 2fis to be kept clear in the blocked design, either as a character vector of pairs of factor letters (using the first elements of 'Letters') or as a two-row matrix of pairs of factor numbers occurring in 2fis)
method	character string identifying a subgraph isomorphism method (VF2 or LAD), see FrF2
sort	character string specifying a presort strategy for subgraph isomorphism search, see FrF2
res3	relevant only if estimable is not NULL; per default (res3=FALSE), design will yield no result, if it has resolution III only; set res3 to TRUE for allowing a design of resolution III (almost never useful)
all0	per default (all0=FALSE), **X** matrices are requested to be free of all-zero columns. Set all0 to TRUE for permitting aliasing between blocks and factor main effects, i.e. for finding a suitable split-plot constructor.
quiet	if TRUE, the message about failure is suppressed (for using the function inside other functions, like FrF2)
firsthit	if TRUE, the function does not attempt to optimize the number of clear 2fis but accepts the first acceptable blocking (relevant for non-null estimable only); per default, optimizing is suppressed for full factorials only (where it is not very reasonable to use function colpick instead of simply hand-crafting an **X** matrix; function FrF2 sets it to FALSE for creation of full factorials with estimability requirements); for large applications with estimability requirements, specifying firsthit as TRUE may allow to inspect more candidates and to then possibly deepdive some of them
X	a $q \times n$ **X** matrix with only 0/1 elements for block construction

`randomize` logical. If TRUE, the design is randomized. This is the default. Randomization is implemented using function `rerandomize.design` of package **DoE.base** as the last step in design creation.

`seed` optional seed for the randomization process
In R version 3.6.0 and later, the default behavior of function `sample` has changed. If you work in a new (i.e., >= 3.6.-0) R version and want to reproduce a randomized design from an earlier R version (before 3.6.0), you have to change the RNGkind setting by
`RNGkind(sample.kind="Rounding")`
before running function `FrF2`.
It is recommended to change the setting back to the new recommended way afterwards:
`RNGkind(sample.kind="default")`
For an example, see the documentation of the example data set `VSGFS`.

`alias.info` degree of effects aliased with blocks to be included in the `design.info` attribute

`profile` profile to use for calculation (NULL or integer vector of up to $2^q - 1$ elements that sum to n); if NULL, the maximally balanced profile is used (which yields the overall maximum number of clear 2fis when blocking a full factorial)

`parts` list that provides factor partitions; list entries must either be all integers from 1 to `n` or the elements of `Letters[1:n]`

`n` number of factors

`p` the number of generated factors (among the `n` factors); 2^{n-p} is the number of runs in the experiment)

`XI` a $q \times k$ **X**_I matrix with only 0/1 elements, to be extended into a $q \times n$ **X** matrix for block construction, given the generators in `gen` ($k = n - p$)

`gen` generators for extending `XI`: `gen` can be
a class `catlg` object (e.g. `catlg["7-2.1"]` or `catlg[nruns(catlg)==32 & nfac(catlg)==7]`); each `nfac` entry must be the sum of k and the length of the `gen` element)
a vector of Yates column numbers (e.g. `c(7, 27)`)
a vector of defining contrasts (e.g. `c("ABC","ABDE")`) a list of vectors of base column numbers

`num` if TRUE (default FALSE), Yates column numbers are returned instead of their character representations

Details

These are the functions for the Godolphin (2021) approach to blocking; most of them are user-visible. This approach and its implementation are described in Groemping (2021). Direct use of this functions is intended for expert use only.

Function `colpick` is the main workhorse function for blocking larger situations in function `FrF2` (since version 2 of the package, it replaces the earlier approach with function `blockpick.big`); it makes use of function `blockgencreate`, and of the internal function `blockgengroup`.

Function `FF_from_X` creates a class `design` object. Design size is limited by computer memory and run time. The function can use an **X** matrix that was produced by function `colpick`; but note that it

is quite easy to hand-craft an **X** matrix for a full factorial, even with estimability requirements. The light-weight function does not have arguments for customization; it can be post-processed, however, e.g. using function `factor.names<-`.

Function `X_from_profile` creates an X matrix that corresponds to the specified profile.

Function `phimax` returns the maximum number of 2fis that can be kept clear when blocking a full factorial design in n factors into blocks of size 2^q, given the specified profile.

Function `blockgencreate` creates block generators from an **X** matrix for blocking a design in $2^{n-p} = 2^k$ runs into blocks of size 2^q, where n and q are derived from `X` as the number of columns and rows, respectively. The generators are returned as a character vector that consists of strings of base factor letters.

Function `Xcalc` extends a $q \times k$ matrix **X**_I by $p = n - k$ columns (**X**_II in Godolphin notatation) based on the generators provided in `gen`.

Function `blockgengroup` is internal only, as are functions `colpickIV` and `clear2fis_from_profile`.

Value

Function `colpick` returns a list of at least two elements:
if `all` is FALSE, the list consists of the matrix `X`, the character vector `clear.2fis` and possibly the integer vector `map`,
otherwise of list-valued elements `X_matrices`, `clearlist` and `profiles` and `maplist`.

Function `FF_from_X` returns a class design object of type `FrF2.blocked`.

Function `phimax` returns a real number.

Function `blockgencreate` returns a character vector of generators in terms of `Letters` combinations of the first $n-p$ factors.

Function `Xcalc` returns a $q \times n$ matrix (in case of a single generator) or a list of such matrices (if `gen` is a class `catlg` object with more than one element).

The internal function `blockgengroup` returns a character vector of all effects (denoted as base column letter combinations) aliased with the block main effect, or corresponding Yates column numbers.

The internal function `colpickIV` returns almost the same type of results as `colpick`. The difference:
if `all` is TRUE, there is an integer vector `map` instead of the `maplist` element, because the `map` does not depend on the choice of X-matrix (separate subgraph isomorphism checking is skipped with this function).

Author(s)

Ulrike Groemping

References

Groemping, U. (2012). Creating clear designs: a graph-based algorithm and a catalogue of clear compromise plans. *IIE Transactions* **44**, 988–1001. Early preprint available at `http://www1.bht-berlin.de/FB_II/reports/Report-2010-005.pdf`.

Godolphin, J. (2021). Construction of Blocked Factorial Designs to Estimate Main Effects and Selected Two-Factor Interactions. *J. Royal Statistical Society* **B 83**, 5-29. doi: 10.1111/rssb.12397.

Groemping, U. (2021). An algorithm for blocking regular fractional factorial 2-level designs with clear two-factor interactions. *Computational Statistics and Data Analysis* **153**, 1-18. doi: 10.1016/j.csda.2020.107059. Preprint at Report 3/2019.

See Also

plot.igraph, tkplot, plot.common

Examples

```
phimax(7, 2)  ## 16 2fis can be clear, if 128 run full factorial is blocked
              ## into 32 blocks of size 2^2=4

## X matrices for blocking full factorials
## do not care about which factors have which role
X_from_profile(7, 2, c(3,2,2))
   # X_from_profile(7, 2, c(2,2,3)) returns same matrix
## ensure specific partition, i.e. specific requirement CIG to be accommodated
X <- X_from_parts(7, 2, parts=list(c("A","D","F"), c("B","G"), c("C","E")))

## blocked full factorial
summary(FF_from_X(X))

## using colpick
## estimable in standard letters
requ <- c("BA", "BC", "BD", "BE", "BF", "BG", "BH", "BJ")
## estimability requirement in factor names
fn <- Letters[15:23] ## P to X
requfn <- requ
requfn <- sapply(1:8, function(obj) gsub(Letters[obj], fn[obj], requfn[obj]))

## obtain X matrix for accommodating estimability requirement in 9-4.2
(aus <- colpick("9-4.2", 2, estimable=requ))
## obtain the same matrix manually with Xcalc
XI <- aus$X[,1:5]
## obtain the same matrix manually with Xcalc
all(Xcalc(XI, catlg["9-4.2"])==aus$X)
## inspect X matrices generated from XI
Xcalc(XI, catlg[nruns(catlg)==32 & nfac(catlg)==9 & res(catlg)>=4])

## factor permutation needed
aus$map
## calculate block generators
blockgencreate(aus$X, p=4)
## automatic creation from the design 9-4.2 uses these block generators
summary(FrF2(32, 9, blocks=8, estimable=requ, factor.names=fn,
    alias.block.2fis = TRUE, select.catlg = catlg["9-4.2"]),
    brief=TRUE)
## can also be reproduced manually (internal function invperm does the permuting)
summary(FrF2(design="9-4.2", blocks=blockgencreate(aus$X, p=4),
    factor.names=fn[FrF2:::invperm(aus$map)],
    alias.block.2fis = TRUE),
    brief=TRUE)
```

IAPlot *Main Effects and Interaction Plots*

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, ..., response = NULL)
## 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"), las=par("las"), abbrev = 3, select = NULL, ...)

IAPlot(obj, ...)
## S3 method for class 'design'
IAPlot(obj, ..., response = NULL)
## 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"), las=par("las"), abbrev = 4, select = NULL, show.alias = FALSE, ...)

intfind(i, j, mat)

check(obj)

remodel(obj)
```

Arguments

obj	an experimental design of class design with the type element of the design.info 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 IAPlot currently requires the response in obj to be a pre-defined variable and not a calculated quantity

...	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
response	character string that specifies response variable to be used, must be an element of response.names(obj); if NULL, the first response from response.names(obj) is used
main	overall title for the plot assembly
pch	Plot symbol number MEPlot, or vector of two plot symbol numbers for the lower and higher level of the trace factor iap
cex.xax	size of x-axis annotation, defaults to cex.axis-parameter
cex.yax	size of y-axis annotation, defaults to cex.xax
mgp.ylab	horizontal placement of label of vertical axis in MEPlot
cex.title	multiplier for size of overall title (cex.main is multiplied with this factor)
cex.main	size of individual plot titles in MEPlot
cex.lab	Size of variable names in diagonal panels of interaction plots produced by IAPlot.
cex	size of plot symbols in interaction plots
lwd	line width for plot lines and axes
las	orientation for tick mark labels (las=1 is recommended)
abbrev	number of characters shown for factor levels
select	vector with position numbers of the main effects to be displayed; default: all main effects; the default implies the full interaction plot matrix for IAPlot. For IAPlot, the full interaction plot matrix for the selected factors is displayed. Of course, at least two factors must be selected. Furthermore, the linear model obj 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 y~(.)^2 for a regular 2-level fractional factorial, for which y is the only response and all other variables are 2-level factors).
show.alias	if TRUE, the interaction plot shows the number of the list entry from aliases(obj) (cf. aliases) in order to support immediate diagnosis of which depicted interaction may be due to other than the shown effect because of aliasing; CAUTION: if the select option is used, the model is reduced to the selected factors, i.e. aliases with unselected factors are not shown!
i	integer, for internal use only
j	integer, for internal use only
mat	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.
If the design contains a block factor, the plot functions show non-block effects only.

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

IAPlot produces plots of all treatment 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` invisibly return the plotted effects (two-row matrix or four-row matrix, respectively). If `show.alias=TRUE`, the matrix returned by IAPlot has as the attribute `aliasgroups`, which contains all alias groups (list element number corresponds to number in the graphics tableau).

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:

`model`	is the redone model with x-variables recoded to numeric -1 and 1 notation and aov objects made into "pure" lm objects
`labs`	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

makecatlg *Function for creating a class catlg catalogue from a vector of generators*

Description

creates a class catlg catalogue with a single element for use in functions colpick or FrF2

Usage

```
makecatlg(k, gen)
```

Arguments

k	number of base factors spanning a full factorial with the desired number of runs
gen	generators as a numeric vector of Yates column numbers

Details

If generators are available in a different format, they must be transformed to Yates column numbers.

For a character vector genc with elements like ABC, ADE, etc., a code for obtaining Yates columns with order preserved is `sapply(1:length(genc), function(obj) which(names(Yates)==genc[obj]))` (a solution with `which` applied to the entire vector at once does not preserve the order).

Yet different formats like 123, 145, etc., can e.g. be preprocessed by picking the suitable elements from `Letters`, e.g. `paste(Letters[as.numeric(unlist(strsplit("123","")))],collapse="")`.

Value

The function returns a list of class `catlg` with a single element.

Note

This package is still under development, but does already provide useful and well-tested results.

Author(s)

Ulrike Groemping

See Also

See also `FrF2`

Examples

```
## Xu's fraction 13-5.2
genXu <- c(127, 143, 179, 85, 150)
catXu <- makecatlg(k=8, genXu)
colpick(catXu, q=2) ## Godolphin blocking into blocks of size 4 yields 56 clear 2fis
FrF2(256, 13, blocks=64, alias.block.2fis=TRUE, select.catlg=catXu)
```

pb *Function to generate non-regular fractional factorial screening designs*

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 design in 8 runs is an exception 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,
     boxtyssedal = TRUE, n12.taguchi = FALSE,
     replications = 1, repeat.only = FALSE,
     randomize = TRUE, seed = NULL, oldver = FALSE, ...)

pb.list
```

Arguments

nruns	number of runs, must be a multiple of 4
nfactors	number of factors, default is nruns - 1, 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 e1, e2, ... They are useful for representing error in effects plots (so-called dummy factors).
factor.names	a character vector of factor names (length up to nfactors) or a list with nfactors 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 factor.names, the levels given in default.levels are used; Default factor names are the first elements of the character vector Letters, or the factors position numbers preceded by capital F in case of more than 50 factors.
default.levels	default levels (vector of length 2) for all factors for which no specific levels are given
ncenter	number of center points; ncenter > 0 is permitted, if all factors are quantitative

`center.distribute`
: 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 `add.center`, which does the work.

`boxtyssedal`
: logical, relevant only for nruns=16. 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.

`n12.taguchi`
: logical, relevant only for nruns=12. If TRUE, the 12 run design is given in Taguchi order.

`replications`
: positive integer number. Default 1 (i.e. each row just once). If larger, each design run is executed replication times. If `repeat.only`, 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).

`repeat.only`
: logical, relevant only if replications > 1. If TRUE, replications of each run are grouped together (repeated measurement rather than true replication). The default is `repeat.only=FALSE`, i.e. the complete experiment is conducted in `replications` blocks, and each run occurs in each block.

`randomize`
: logical. If TRUE, the design is randomized. This is the default.

`seed`
: optional seed for the randomization process
  In R version 3.6.0 and later, the default behavior of function `sample` has changed. If you work in a new (i.e., >= 3.6.-0) R version and want to reproduce a randomized design from an earlier R version (before 3.6.0), you have to change the RNGkind setting by
  `RNGkind(sample.kind="Rounding")`
  before running function pb.
  It is recommended to change the setting back to the new recommended way afterwards:
  `RNGkind(sample.kind="default")`
  For an example, see the documentation of the example data set `VSGFS`.

`oldver`
: logical. If `TRUE`, the column ordering from package versions 1.0-5 to 1.2.10 is used. This affects designs in 40, 52, 56, 64, 76, 92, 96 and 100 runs. Usually, option `oldver` should not be set to is useful for reproducing an old design, or for making a design in 40, 56, 64, 88 or 96 runs with exactly half the number of factors resolution IV.

`...`
: 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.

Designs up to 100~runs are covered.

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:

`desnum` 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
- **factor.names** list named with (treatment) factor names and containing as entries vectors of length two each with coded factor levels
- **ndummies** number of dummy factors for error
- **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**)

Warning

With version 1.0-5 of package **FrF2**, design generation for the designs based on doubling has changed (internal function `double.des`). This affected designs for 40,56,64,88,96 runs. With version 1.3 of package **FrF2**, this and further behaviors (52, 76) has changed again, in the interest of improving generalized resolution of desigs produced by function pb.

For the affected run sizes, package versions from 1.0-5 onwards cannot exactly reproduce pb designs that have been created with a version before 1.0-5. Package versions from 1.3 onwards reproduce the behavior of versions 1.0-5 to 1.2-10 through option `oldver`.

Warning

Since R version 3.6.0, the behavior of function `sample` has changed (correction of a biased previous behavior that should not be relevant for the randomization of designs). For reproducing a randomized design that was produced with an earlier R version, please follow the steps described with the argument `seed`.

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.

Groemping, U. (2014). R Package FrF2 for Creating and Analyzing Fractional Factorial 2-Level Designs. *Journal of Statistical Software*, **56**, Issue 1, 1-56. https://www.jstatsoft.org/v56/i01/.

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, generalized.word.length for functions length3 and length4 used in examples

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

## Not run:
## note: designs in 40, 56, 64, 88, and 96 runs are resolution IV,
## if the number of factors is up to nruns/2 - 1, e.g.:
plan1 <- pb(40, 19)
length3(plan1)  ## 0 generalized words of length 3
length4(plan1)  ## 228 generalized words of length 4
## they can be made resolution IV by oldver=TRUE for
## nfactors=nruns/2, e.g.:
plan2 <- pb(40, 20)
plan3 <- pb(40, 20, oldver=TRUE)
length3(plan2)  ## 9 generalized words of length 3
length3(plan3)  ## 0 generalized words of length 3
length4(plan3)  ## 285 generalized words of length 4

## note: designs in 52, 76, and 100 runs are almost resolution IV,
## if the number of factors is up to nruns/2 - 1, e.g.:
plan4 <- pb(52, 25)
GR(plan4)       ## generalized resolution 3.92

## note: versions >1.3 avoid complete and heavy aliasing of triples of factors
## for up to nruns-2 factors for 40, 52, 56, 64, 76, 88, 92 and 96 runs
## (the same for 100 runs, which were not implemented before version 1.3)
plan5 <- pb(40, 38)
plan6 <- pb(40, 38, oldver=TRUE)
GR(plan5)       ## generalized resolution 3.4
GR(plan6)       ## generalized resolution 3
plan7 <- pb(52, 50)
plan8 <- pb(52, 50, oldver=TRUE)
GR(plan7)       ## generalized resolution 3.62
GR(plan8)       ## generalized resolution 3.15

## End(Not run)
```

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: The 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 preferrable 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 treatement 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 `log2(WPs) <= nfac.WP <= WPs-1`, it is obviously in principle possible to accomodate 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 < log2(WPs)`. In this case, it is of course nevertheless necessary to have a total of `log2(WPs)` whole plot *construction* factors; the missing `log2(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 accomodate 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 accomodated; 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("-","+"))
design.info(BS.ex)
BS.ex
## NOTE: the design has 8 whole plots.
## The first hard-to-change factors have very few changes only
## between whole plots.
## A conscious and honest decision is required whether it is
##    acceptable for the situation at hand not to reset them!
## A proper split-plot design with resetting all whole plot factors
## for each whole plot would be strongly preferred from a
## statistical point of view.

########## 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.WP=3,
      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)
```

`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 + length(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 `splitpick`, 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 `FrF2` for accomodating hard-to-change factors

Details

These functions exploit the fact that a factorial design can be arranged such that the `2^k.WP-1` leftmost columns have exactly `2^k.WP` different patterns. They can thus accomodate whole plot effects if `2^k.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.WP` ones that accomodate 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 accomodate 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.

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

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