

# Package ‘planor’

February 9, 2017

**Type** Package

**Title** Generation of Regular Factorial Designs

**Version** 1.3-1

**Date** 2017-02-09

**Author** Hervé Monod, Annie Bouvier, André Kobilinsky

**Maintainer** Annie Bouvier <Annie.Bouvier@inra.fr>

**Description** Automatic generation of regular factorial designs, including fractional designs, orthogonal block designs, row-column designs and split-plots.

**License** GPL (>= 2)

**Imports** stats, conf.design, utils, bit64, Rcpp (>= 0.12.7)

**LinkingTo** Rcpp, RcppArmadillo

**Depends** methods

**Collate** zzz.R generic.R designfactors.R keymatrix.R keyring.R  
listofkeyrings.R planor.R basep.R randomize.R designkey.R  
listofdesignkeys.R planordesign.R makedesignkey.R

**Encoding** UTF-8

**URL** <http://genome.jouy.inra.fr/logiciels/planor>

## R topics documented:

planor-package	2
alias-methods	3
as.data.frame.planordesign	5
bind-methods	5
designfactors-class	6
designkey-class	7
getDesign-methods	8
keymatrix-class	9
keyring-class	10
listofdesignkeys-class	11
listofkeyrings-class	12
makedesignkey	13
pick-methods	14
planor.design-methods	15
planor.designkey	16

planor.factors . . . . .	18
planor.harmonize . . . . .	19
planor.model . . . . .	20
planor.randomize . . . . .	22
planordesign-class . . . . .	23
regular.design . . . . .	24
show-methods . . . . .	25
summary-methods . . . . .	26

<b>Index</b>	<b>29</b>
--------------	-----------

---

planor-package	<i>Generation of Regular Factorial Designs</i>
----------------	--

---

## Description

A package dedicated to the automatic generation of regular factorial designs, including fractional designs, orthogonal block designs, row-column designs and split-plots.

## Details

The user describes the factors to be controlled in the experiment and the anova model to be used when the results will be analysed. He or she also specifies the size of the design, that is, the number of available experimental units. Then **planor** looks for a design satisfying these specifications and possibly randomizes it. The core of the algorithm is the search for the key matrix, an integer matrix which determines the aliasing in the resulting factorial design.

The user may use the function `regular.design` where all these steps are integrated, and transparent by default. Alternatively, the steps can be decomposed by using successively the functions `planor.factors`, `planor.model`, `planor.designkey` and `planor.design`. For the expert user, the function `planor.designkey` can give several key matrix solutions. Alias and summary methods allow to study and compare these solutions, in order to select the most appropriate one for the final design.

## Note

An R option named `planor.max.print` is set. It is equal to the number of printed rows and columns in the display of **planor** matrices. Default is 20. You can change its value by using the function `options()` (see `?options`).

## Author(s)

Monod, H. and Bouvier, A. and Kobilinsky, A.

## References

See `citation("planor")`.

## Examples

```
# DESIGN SPECIFICATIONS
# Treatments: four 3-level factors A, B, C, D
# Units: 27 in 3 blocks of size 9
# Non-negligible factorial terms:
#   block + A + B + C + D + A:B + A:C + A:D + B:C + B:D + C:D
# Factorial terms to estimate:
#   A + B + C + D
# 1. DIRECT GENERATION, USING regular.design
mydesign <- regular.design(factors=c("block", LETTERS[1:4]),
  nlevels=rep(3,5), model=~block+(A+B+C+D)^2, estimate=~A+B+C+D,
  nunits=3^3, randomize=~block/UNITS)
print(mydesign)
# DUMMY ANALYSIS
# Here we omit two-factor interactions from the model, so they are
# confounded with the residuals (but not with ABCD main effects)
set.seed(123)
mydesigndata <- mydesign@design
mydesigndata$Y <- runif(27)
mydesign.aov <- aov(Y ~ block + A + B + C + D, data=mydesigndata)
summary(mydesign.aov)
# 2. STEP-BY-STEP GENERATION, USING planor.designkey
F0 <- planor.factors(factors=c("block", LETTERS[1:4]), nlevels=rep(3,5),
  block=~block)
M0 <- planor.model(model=~block+(A+B+C+D)^2, estimate=~A+B+C+D)
K0 <- planor.designkey(factors=F0, model=M0, nunits=3^3, max.sol=2)
summary(K0)
mydesign.S4 <- planor.design(key=K0, select=2)
```

---

alias-methods

*Summarize the Design Properties*


---

## Description

Methods to summarize the design properties of an object containing key matrices. Display the design keys matrix(ces) and the factorial effects confounded with the mean.

## Usage

```
## S4 method for signature 'designkey'
alias(object, model, ...)

## S4 method for signature 'keymatrix'
alias(object, model, fact, block, ...)

## S4 method for signature 'listofdesignkeys'
alias(object, model, ...)

## S4 method for signature 'listofkeyrings'
alias(object, model, ...)

## S4 method for signature 'planordesign'
alias(object, model, fact, block, ...)
```

## Arguments

<code>object</code>	an object of the class.
<code>model</code>	an optional model formula (by default the first model in <code>object</code> ) or, when <code>object</code> is a <a href="#">keymatrix</a> , a matrix representing factorial model terms.
<code>fact</code>	a character or numeric vector of parent factor names for the columns of <code>object</code> .
<code>block</code>	a logical vector to identify the columns of <code>object</code> associated with a block factor.
<code>...</code>	ignored.

## Details

When `object` is a [keymatrix](#), “alias” displays the key matrix and the factorial effects confounded with the mean. It prints the unaliased treatment effects, then the groups of aliased treatment effects, then the treatments effects confounded with block effects and finally the unaliased block effects, when considering all the factorial terms that are represented in the `model` argument, which is set if missing to the identity matrix (main effects only).

## Value

- When `object` is a [keymatrix](#), a vector with (i) the number of unaliased treatment effects; (ii) the number of mutually aliased treatment effects; (iii) the number of treatment effects aliased with block effects.
- When `object` is a [designkey](#), an invisible NULL.
- When `object` is a [listofkeyrings](#), the factors, the model and the number of solutions for each prime in a list indexed by the primes `p` of the object. Each element is a 3-column matrix with one row per solution for prime `p`. The columns give (i) the number of unaliased treatment effects; (ii) the number of mutually aliased treatment effects; (iii) the number of treatment effects aliased with block effects.
- The method is NOT YET IMPLEMENTED on objects of class [listofdesignkeys](#).
- When `object` is a [planordesign](#), this function is the alias method applied on each of the [keymatrix](#) objects contained in its `designkey` slot.

## Examples

```
### Creation of an object of class listofkeyrings
K0 <- planor.designkey(factors=c(LETTERS[1:4], "block"), nlevels=rep(3,5),
model=~block+(A+B+C+D)^2, estimate=~A+B+C+D,
nunits=3^3, base=~A+B+C, max.sol=2)
### alias on an object of class keymatrix
alias(K0[[1]][[1]])
### alias on an object of class designkey
alias(K0[1])
### alias on an object of class listofkeyrings
alias(K0)
```

---

as.data.frame.planordesign

*Data Frame Coercion*


---

**Description**

Method to extract the dataframe containing the final design from a `planordesign` object, i.e the slot design. The other slots are stored in attributes.

**Usage**

```
## S4 method for signature 'planordesign'
as.data.frame(x, ...)
```

**Arguments**

`x`                    an object of class `planordesign`.  
`...`                  ignored.

**Value**

A data frame with attributes `factors`, `model`, `designkey`, `nunits`, `recursive`.

**Examples**

```
### Creation of a planordesign object
K0 <- planor.designkey(factors=c("R", "C", "U", "A", "B1", "B2"),
  nlevels=c(3,2,2,3,2,2), model=~R*C + (A+B1+B2)^2, estimate=~A:B1+A:B2,
  nunits=12, base=~R+C+U, max.sol=2)
P0 <- planor.design(key=K0, select=1)
### Convert into a data frame
D0 <- as.data.frame(P0)
```

---

bind-methods

*Bind Two Objects*


---

**Description**

Methods to bind two objects of the same class.

**Usage**

```
## S4 method for signature 'designfactors,designfactors'
bind(x, y)
```

**Arguments**

`x`                    an object of the first class in the signature.  
`y`                    an object of the second class in the signature.

**Value**

An object of the same class as `x` and `y`, containing their joint content.

**Note**

Factors with same name are distinguished and advertised with a warning.

**See Also**

Class where this method applies: [designfactors](#)

**Examples**

```
F1 <- planor.factors(factors=c("block",LETTERS[1:4]), nlevels=c(6,6,4,2,6))
F2 <- planor.factors(factors=c("block",LETTERS[11:12]), nlevels=c(6,6,4))
### Method bind on designfactors objects
F3 <- bind(F1,F2)
names(F3)
```

---

designfactors-class

*Class designfactors*

---

**Description**

An S4 class to represent the design factors and store their characteristics.

**Objects from the Class**

Objects from this class can be created explicitly by calls to [planor.factors](#) or implicitly by functions such as [planor.designkey](#).

**Slots**

**fact.info** a dataframe with one row per factor and with columns progressively storing information on the factors, in particular their numbers of levels (`nlev`).

**pseudo.info** a dataframe with one row per pseudofactor and with columns progressively storing information on the pseudofactors.

**levels** a list of numeric or character vectors, with each vector containing the levels of one factor.

**Methods**

**[** extract a subset of factors and update all the slots.

**bind** bind two objects. See [bind](#) method.

**length** return the number of factors.

**names** return the names of the factors.

## Details

Depending on the context and on the construction stage, `fact.info` may contain logical columns that identify the block factors (`block`), the ordered factors (`ordered`), the basic factors (`basic`) and so on. It may also include columns that store the information on the hierarchy relationships between factors, if any.

In package **planor**, factors are systematically decomposed into pseudofactors which all have a prime number of levels and which play a key role in the design generation. The information on the pseudofactors is stored in the `pseudo.info` slot. In addition to the columns of `fact.info`, it contains a column (called `parent`) to give the factor that each pseudofactor decomposes.

## Author(s)

Monod, H. and Bouvier, A.

## See Also

Creator function: `planor.factors`

## Examples

```
F1 <- planor.factors(factors=c("block",LETTERS[1:4]), nlevels=c(6,6,4,2,6))
F2 <- planor.factors(factors=c("block",LETTERS[11:12]), nlevels=c(4,6,6))
### Method bind - a warning will be issued because two factors
### in F1 and F2 have the same name
F3 <- bind(F1,F2)
names(F3)
length(F3)
F3@levels
F3.trt <- F3[c(2:5,7,8)]
names(F3.trt)
```

---

designkey-class	<i>Class designkey</i>
-----------------	------------------------

---

## Description

An S4 class to represent a design-key solution.

## Objects from the Class

Objects can be created by extraction from an object of class `listofkeyrings` or class `listofdesignkeys`.

## Slots

**.Data** a single design-key solution, i.e a list with one `keymatrix` per prime.

**factors** an object of class `designfactors` which contains the factors specifications.

**model** a list which contains the model and estimate specifications.

**nunits** the number of units of the design.

**recursive** a logical equal to TRUE if the design has been constructed recursively.

**Extends**

Class `list`, from data part. Class `vector`, by class `list`, distance 2.

**Methods**

**alias** summarize the design properties. See `alias` method.

**planor.design** build the design from the design key matrix. See `planor.design` method.

**show** display the object. See `show` method.

**summary** summarize the design properties. See `summary` method.

**Author(s)**

Monod, H. and Bouvier, A.

**Examples**

```
### Creation of a designkey object
K0 <- planor.designkey(factors=c(LETTERS[1:4], "block"), nlevels=rep(3,5),
  model=~block+(A+B+C+D)^2, estimate=~A+B+C+D,
  nunits=3^3, base=~A+B+C, max.sol=2)
print(K0[1])
```

---

getDesign-methods    *Extract a Design*

---

**Description**

Methods to extract a design data frame from an object.

**Usage**

```
## S4 method for signature 'planordesign'
getDesign( object )
```

**Arguments**

`object`                      object of the class.

**Value**

A data frame which contains the design.

**Note**

Applied on a `planordesign` object, it is the same as function `as.data.frame`, without returning any attributes.

**See Also**

Class where this method applies: `planordesign`.



**Examples**

```
### Creation of a planordesign object
K0 <- planor.designkey(factors=c("R","C","U","A","B1","B2"),
  nlevels=c(3,2,2,3,2,2), model=~R*C + (A+B1+B2)^2, estimate=~A:B1+A:B2,
  nunits=12, base=~R+C+U, max.sol=2)
P0 <- planor.design(key=K0, select=1)
### Method getDesign on the planordesign object
show(getDesign(P0))
```

---

keymatrix-class	<i>Class keymatrix</i>
-----------------	------------------------

---

**Description**

An S4 class to represent an elementary key matrix.

**Objects from the Class**

Objects from this class are usually components of an object of class [keyring](#) or [designkey](#).

**Slots**

**.Data** a matrix of integers modulo  $p$ .

**p** a prime number.

**Extends**

Class [matrix](#), from data part. Class [array](#), by class matrix, distance 2. Class [structure](#), by class matrix, distance 3. Class [vector](#), by class matrix, distance 4, with explicit coerce.

**Methods**

**alias** give the aliasing relationships of the key matrix. See [alias](#) methods.

**show** display the object. See [show](#) methods.

**summary** summarize the design properties. See [summary](#) methods.

**Author(s)**

Monod, H. and Bouvier, A.

**See Also**

[keyring](#), [designkey](#)

**Examples**

```
showClass("keymatrix")
### Creation of a listofkeyrings object
K0 <- planor.designkey(factors=c("block", LETTERS[1:4]), nlevels=rep(3,5),
  model=~block + (A+B+C+D)^2, estimate=~A+B+C+D,
  nunits=3^3, base=~A+B+C, max.sol=2)
### Show a keymatrix of K0
show(K0[[1]][[1]])
```

keyring-class

*Class keyring***Description**

An S4 class to represent a list of design-key matrices which are associated with the same prime and which represent alternative solutions to the same design specifications.

**Objects from the Class**

Each component of the structure returned by `planor.designkey` is a `keyring` object when the case is not recursive.

**Slots**

**.Data** a list of `keymatrix` objects.

**p** a prime number.

**LIB** a list containing a vector of row names and a vector of column names. The names are the same for all key matrices.

**pseudo.info** a dataframe containing information on the pseudofactors associated with the key matrices. See the description of the class `designfactors`.

**Extends**

Class `list`, from data part. Class `vector`, by class list, distance 2.

**Methods**

**show** display the object. See `show` method.

**summary** summarize the design properties. See `summary` method.

**Note**

Each key matrix in a `keyring` object is a possible solution to the same factors, model and estimate specifications, with respect to the same prime number. An object of class `listofkeyrings` is a list of `keyring` objects associated with the different primes involved in a given factorial design problem.

**Author(s)**

Monod, H. and Bouvier, A.

**See Also**

`planor.designkey`, methods `pick.listofkeyrings` and `summary.keymatrix`, the class `keyring`

**Examples**

```
showClass("keyring")
### Creation of a listofkeyrings object
K0 <- planor.designkey(factors=c("block", LETTERS[1:4]), nlevels=rep(3,5),
  model=~block+(A+B+C+D)^2, estimate=~A+B+C+D,
  nunits=3^3, base=~A+B+C, max.sol=2)
### Show a keyring component of K0
show(K0[[1]])
```

---

```
listofdesignkeys-class
```

```
Class listofdesignkeys
```

---

**Description**

An S4 class to represent a list of design key solutions.

**Objects from the Class**

Objects are created by `planor.designkey`, when the search is recursive.

**Slots**

**.Data** a list of objects of class `designkey`.

**factors** an object of class `designfactors` which contains the factors specifications.

**model** a list which contains the model and estimate specifications.

**nunits** the number of units in the design.

**Extends**

Class `list`, from data part. Class `vector`, by class list, distance 2.

**Methods**

**alias** NOT YET IMPLEMENTED.

**[** extract one design key in the list.

**pick** extract one design key in the list. See `pick` method.

**planor.design** build a design from one design key in the list. See `planor.design` method.

**show** display the object. See `show` method.

**summary** summarize the design properties. See `summary` method.

**Author(s)**

Monod, H. and Bouvier, A.

**See Also**

Creator function: `planor.designkey`

## Examples

```
showClass("listofdesignkeys")
### Creation of a listofdesignkeys object
K0 <- planor.designkey(factors=c("R", "C", "U", "A", "B1", "B2"),
  nlevels=c(3,2,2,3,2,2), model=~R*C + (A+B1+B2)^2, estimate=~A:B1+A:B2,
  nunits=12, base=~R+C+U, max.sol=2)
### Method show
show(K0)
### Method length
length(K0)
### Extract component. The two following two commands are equivalent
K <- K0[2]
K <- pick(K0,2)
```

---

```
listofkeyrings-class
```

```
Class listofkeyrings
```

---

## Description

An S4 class to store design key solutions when there is only one prime involved or when the solutions are independent between primes.

## Objects from the Class

Objects are created by `planor.designkey`, when the case is not recursive.

## Slots

**.Data** a list of objects of class `keyring` associated with different primes.  
**factors** an object of class `designfactors` which contains the factors specifications.  
**model** a list which contains the model and estimate specifications.  
**nunits** the number of units of the design.

## Extends

Class `list`, from data part. Class `vector`, by class list, distance 2.

## Methods

**alias** give the aliasing for each key-matrix. See `alias` method.  
**[** extract one design key by taking one key matrix per prime.  
**pick** extract one design key by taking one key matrix per prime. See `pick` method.  
**planor.design** build a design using one key matrix per prime. See `planor.design` method.  
**show** display the object. See `show` method.  
**summary** summarize the design properties from object. See `summary` method.

## Author(s)

Monod, H. and Bouvier, A.

**See Also**

Creator function: [planor.designkey](#)

**Examples**

```
showClass("listofkeyrings")
### Creation of a listofkeyrings object
K0 <- planor.designkey(factors=c(LETTERS[1:4], "block"), nlevels=rep(3,5),
  model=~block+(A+B+C+D)^2, estimate=~A+B+C+D,
  nunits=3^3, base=~A+B+C, max.sol=2)
show(K0)
```

---

makedesignkey	<i>Turn Integer Matrices into an Object of Class designkey</i>
---------------	--

---

**Description**

Create an object of class [designkey](#) from a list of integer matrices.

**Usage**

```
makedesignkey(keys, primes)
```

**Arguments**

keys	a list of n integer matrices with column names.
primes	a vector of n prime numbers.

**Details**

The names of the factors are extracted from the matrix column names.

**Value**

An object of class [designkey](#).

**Author(s)**

Monod, H.

**Examples**

```
mat1 <- cbind(diag(3),1)
colnames(mat1) <- c("A", "B", "C", "D")
mat2 <- cbind(diag(2),c(1,2))
colnames(mat2) <- c("E", "F", "G")
mat.dk <- makedesignkey(list(mat1,mat2), primes=c(2,3))
print(mat.dk)
summary(mat.dk)
alias(mat.dk)
mat.plan <- planor.design(mat.dk)
```

**Description**

Methods to extract a single `designkey` object (with one key matrix per prime) from a multi-components object.

**Usage**

```
## S4 method for signature 'listofdesignkeys'
pick(keys, selection)
## S4 method for signature 'listofkeyrings'
pick(keys, selection)
```

**Arguments**

<code>keys</code>	an object of the class.
<code>selection</code>	<ul style="list-style-type: none"> <li>• when <code>keys</code> is a <code>listofdesignkeys</code> object, an integer scalar equal to the position of the required solution.</li> <li>• when <code>keys</code> is a <code>listofkeyrings</code> object, the index vector to select the key matrix for each prime.</li> </ul>

**Value**

An object of class `designkey`, which contains the selected design.

**Note**

`pick(K0, 1)` can be simply written `K0[1]`

**See Also**

Classes where this method applies: `listofdesignkeys`, `listofkeyrings`.

**Examples**

```
### Creation of an object of class listofdesignkeys
K2 <- planor.designkey(factors=c("R", "C", "U", "A", "B1", "B2"),
  nlevels=c(3,2,2,3,2,2), model=~R*C + (A+B1+B2)^2, estimate=~A:B1+A:B2 , nunits=12,
  base=~R+C+U, max.sol=2)
### Method pick applied on the listofdesignkeys object
K2.1 <- pick(K2,1)
K2.1 <- K2[1] ## Another way of extracting ([ is synonym of pick)

### Creation of an object of class listofkeyrings
K0 <- planor.designkey(factors=c(LETTERS[1:4], "block"),
  nlevels=rep(3,5), model=~block+(A+B+C+D)^2, estimate=~A+B+C+D,
  nunits=3^3, base=~A+B+C, max.sol=2)
### Method pick applied on the listofkeyrings object
K0.1 <- pick(K0,1)
K0.1 <- K0[1] ## the same
```

---

planor.design-methods

*Build a Design from a Design Key Solution*


---

## Description

Methods to build a factorial design from an object containing key matrices.

## Usage

```
## S4 method for signature 'designkey'
planor.design(key, randomize=NULL, ...)

## S4 method for signature 'listofdesignkeys'
planor.design(key, randomize=NULL, selection=1, ...)

## S4 method for signature 'listofkeyrings'
planor.design(key, randomize=NULL, selection,...)

## S4 method for signature 'numeric'
planor.design(key, start=1)
```

## Arguments

key	an object of the first class in the signature, or a vector of integers.
randomize	an optional formula to specify the block structure for design randomization.
selection	when key is a <a href="#">listofdesignkeys</a> object, an integer scalar. when key is a <a href="#">listofkeyrings</a> object, should be an index vector to select the key matrix for each prime.
...	additional arguments, in particular those related to randomization (see <a href="#">planor.randomize</a> ).
start	an integer from where to start the series of symbols.

## Details

- When key is numeric, it should be a vector of integers of length  $s$ . Then, the function generates a full factorial  $n_1 x n_2 x \dots x n_s$  design with columns considered as factors. It returns an integer matrix with  $\text{prod}(n)$  rows and  $s$  columns giving all combinations along the rows, in lexicographic order.
- When key is a [listofdesignkeys](#) object, build one design from a selected solution.

## Value

An object of class [planordesign](#), which contains the design built from the input. This function is restricted to give a single design. When key is numeric, see Details.

## See Also

Classes where this method applies: [designkey](#), [listofdesignkeys](#), [listofkeyrings](#).

## Examples

```
### Creation of a listofdesignkeys object
K0 <- planor.designkey(factors=c("R","C","U","A","B1","B2"),
  nlevels=c(3,2,2,3,2,2), model=~R*C + (A+B1+B2)^2, estimate=~A:B1+A:B2,
  nunits=12, base=~R+C+U, max.sol=2)
### Method planor.design applied on the listofdesignkeys object
P0 <- planor.design(key=K0, select=1)
### Method planor.design applied on a designkey object
P0 <- planor.design(K0[1])

### Creation of a listofkeyrings object
K0 <- planor.designkey(factors=c(LETTERS[1:4], "block"), nlevels=rep(3,5),
  model=~block+(A+B+C+D)^2, estimate=~A+B+C+D,
  nunits=3^3, base=~A+B+C, max.sol=2, verbose=TRUE)
### Method planor.design applied on a designkey object
P0 <- planor.design(K0[1])
P0.R <- planor.design(K0[1], randomize=~A+B+C+D) # randomize the final design
```

---

planor.designkey      *Search for a Design Key or a Collection of Design Keys*

---

## Description

Search for a design key or a collection of design keys that satisfy the design properties specified by the arguments. This function implements the core algorithms of the **planor** package.

## Usage

```
planor.designkey(factors, nlevels, block, ordered, hierarchy, model,
  estimate, listofmodels, resolution, nunits, base, max.sol=1,
  randomsearch=FALSE, verbose=TRUE)
```

## Arguments

factors	an object of class <a href="#">designfactors</a> , typically an output from <a href="#">planor.factors</a> . Alternatively, you can use the arguments factors, nlevels, ordered, hierarchy as described in the syntax of <a href="#">planor.factors</a> .
nlevels	see <a href="#">planor.factors</a> . Ignored if factors is of class <a href="#">designfactors</a> .
block	see <a href="#">planor.factors</a> . Ignored if factors is of class <a href="#">designfactors</a> .
ordered	see <a href="#">planor.factors</a> . Ignored if factors is of class <a href="#">designfactors</a> .
hierarchy	see <a href="#">planor.factors</a> . Ignored if factors is of class <a href="#">designfactors</a> .
model	a list of model-estimate pairs of formulae, typically an output from <a href="#">planor.model</a> . Alternatively, you can use the arguments model, estimate, listofmodels and resolution, as described in the syntax of <a href="#">planor.model</a> .
estimate	see <a href="#">planor.model</a> . Ignored if model is a list.
listofmodels	see <a href="#">planor.model</a> . Ignored if model is a list.
resolution	see <a href="#">planor.model</a> . Ignored if model is a list. When set and there is no hierarchy, a faster algorithm is used which exploits the symmetries.



nunits	a scalar giving the total number of units in the design
base	an optional additive formula to specify the basic factors. See Note.
max.sol	maximum number of solutions before exit.
randomsearch	a logical. If TRUE, the searches for a key matrix are performed in a random order.
verbose	a logical to set to TRUE for verbose display.

## Details

The methods implemented in **planor** rely on a decomposition of the design search according to prime numbers. The prime numbers involved are those that decompose the numbers of levels of the factors. For example, if all factors have 2, 4, or 8 levels, then the number of units must be a power of 2 and the only prime number involved is 2. This is called the *symmetric* case. But if at least one factor has 6 levels, or if factor *A* has 2 levels and factor *B* has 3 levels, then the number of units must be the product of a power of 2 by a power of 3. In this case the search is automatically decomposed into one for prime 2 and one for prime 3. This is called the *asymmetric* case.

In the symmetric case with prime *p*, a regular factorial design requires a single key matrix of integers modulo *p*. In the asymmetric case, it requires one key matrix per prime. In **planor**, key matrices are stored in objects of class `keymatrix`. The lists made of one key matrix per prime are called design keys. They are stored in objects of class `designkey`.

The function `planor.designkey` essentially searches for design keys that satisfy the user specifications. For technical reasons, however, its output can take two different forms: either an object of class `listofkeyrings` or an object of class `listofdesignkeys`. The function `planor.designkey` detects automatically which case applies. In the first case (*independent case*), the key matrix solutions can be searched independently between primes and they are stored in objects of class `listofkeyrings`. The second case (*recursive case*) occurs exceptionnally. In that case the search cannot be independent between primes and so the different solutions are directly stored in a list of class `listofdesignkeys`.

## Value

An object of class `listofkeyrings` in most cases. Otherwise, i.e in recursive cases, an object of class `listofdesignkeys`.

## Note

The `nunits` argument is compulsory except if the `base` argument is used. When both arguments are missing, the program stops and it gives the size that would be required by a full factorial design. When only `nunits` is missing, the number of units is given by the product of the numbers of levels of the base factors.

The `base` formula must be an additive formula involving a subset of factors, called the basic factors. Using the `base` argument ensures that the design solutions will include the full factorial design for the basic factors. This option can speed up the search because it restricts the domain to be explored by the search algorithm.

## Author(s)

Monod, H. and Bouvier, A.

**See Also**

`planor.factors`, `planor.model`, and the classes `designfactors`, `listofkeyrings`, `listofdesignkeys`

**Examples**

```
K0 <- planor.designkey(factors=c("block", LETTERS[1:4]),
  nlevels=rep(3,5), model=~block+(A+B+C+D)^2, estimate=~A+B+C+D,
  nunits=3^3, base=~A+B+C, max.sol=2)
### With automatic model generation
Km <- planor.designkey(factors=c("block", LETTERS[1:4]),
  nlevels=rep(2,5), resolution=3, nunits=2^4)
```

---

<code>planor.factors</code>	<i>Create an Object of Class <code>designfactors</code></i>
-----------------------------	---

---

**Description**

Create an object of class `designfactors`, from the factor names and their numbers of levels, or from a named list of factor levels. Both ways can be used in the same call. Additional information can be provided: they will be used during the design search or in the summary functions applied to the object.

**Usage**

```
planor.factors(factors = NULL, nlevels = NULL,
  block = NULL, ordered = NULL, hierarchy = NULL,
  dummy = FALSE)
```

**Arguments**

<code>factors</code>	a character vector of factor names, or possibly a scalar, a dataframe or a list (see DETAILS).
<code>nlevels</code>	a vector of level numbers for each factor name (see DETAILS).
<code>block</code>	an additive model formula to indicate the block factors.
<code>ordered</code>	an additive model formula to indicate the quantitative factors (not used at all in the present version).
<code>hierarchy</code>	a formula or a list of formulae to indicate hierarchy relationships between factors (see the <b>planor</b> vignette for details).
<code>dummy</code>	a logical to identify dummy factors created and deleted by <b>planor</b> functions for technical reasons.

**Value**

An object of class `designfactors`.

**Note**

The basic usage is to specify the names of the factors by a character vector of length  $n$  in argument `factors` and their numbers of levels by a numeric vector of length  $n$  in argument `nlevels`. Alternatively, the `factors` argument can be an integer  $n$ , in which case the first  $n$  capital letters of the alphabet are used as factor names. If `nlevels` is a scalar  $s$ , it is considered that all factors have  $s$  levels. There are two more possibilities which allow for alphanumeric factor levels. If `factors` is a dataframe, the factors in this dataframe are extracted together with their levels. Finally `factors` can be a named list of  $n$  vectors, with each vector containing the levels of one factor. Note that `nlevels` is ignored in these latter two cases. See the examples. The argument `block` allows to specify the block or nuisance factors. This information is used by the [alias](#) and [summary](#) functions but it has no effect on the design generation and randomization which depend on other arguments.

**Author(s)**

Monod, H. and Bouvier, A.

**See Also**

Class [designfactors](#)

**Examples**

```
planor.factors(c("A", "B", "C", "P"), c(2, 3, 6, 3))
planor.factors(LETTERS[1:12], 2)
planor.factors(12, 2)
planor.factors( c("A", "B", "Block"), 3, block=~Block)
zz <- planor.factors( c("A", "B", "Block"), c(2, 3, 5))
zz@levels$A <- c("plus", "moins")
planor.factors(factors=list(A=c("plus", "moins"), B=1:3, Block=1:5))
AB <- data.frame( A=c(rep(c("a", "b"), 3)), B=rep(c("z", "zz", "zzz"), rep(2, 3)), C=1:6)
planor.factors(factors=AB)
```

---

planor.harmonize     *Harmonize the Factors*

---

**Description**

Harmonize the factors originating from a list of factors, a list of models, and a list of basic factors (this function is essentially for internal use).

**Usage**

```
planor.harmonize(factors, nlevels, ordered, hierarchy, model, estimate,
  listofmodels, base)
```

**Arguments**

<code>factors</code>	an object of class <a href="#">designfactors</a> , typically an output from <a href="#">planor.factors</a> ). Otherwise the arguments <code>factors</code> , <code>nlevels</code> , <code>ordered</code> , <code>hierarchy</code> follow the syntax of <a href="#">planor.factors</a> .
<code>nlevels</code>	see <a href="#">planor.factors</a> . Ignored if <code>factors</code> is of class <a href="#">designfactors</a> .

ordered	see <code>planor.factors</code> . Ignored if <code>factors</code> is of class <code>designfactors</code> .
hierarchy	see <code>planor.factors</code> . Ignored if <code>factors</code> is an object of class <code>designfactors</code> .
model	a list of model-estimate pairs of formulae, typically an output from <code>planor.model</code> . Otherwise the arguments <code>model</code> , <code>estimate</code> and <code>listofmodels</code> follow the syntax of <code>planor.model</code> .
estimate	see <code>planor.model</code> . Ignored if <code>model</code> is a list.
listofmodels	see <code>planor.model</code> . Ignored if <code>model</code> is a list.
base	an optional formula to specify the basic factors. These factors must belong to the <code>factors</code> argument

### Value

An object of class `designfactors` very similar to `factors`, but with two additional logical columns in slots `fact.info` and `pseudo.info`:

- `model` (TRUE for the factors present in at least one model formula),
- `basic` (TRUE for the basic factors).

### Note

This function is called at the start of the design search. It is essentially a check that the factors in all three arguments are coherent, even though it performs some additional tasks. The function stops if it detects a model or basic factor that is absent from `factors`. This is because the number of levels of such a factor is unknown and so the design search cannot proceed. Besides, the function eliminates the factors that do appear neither in `model` nor in `base` and it reorders the factors by putting first the basic ones.

### Author(s)

Monod, H. and Bouvier, A.

### Examples

```
F2 <- planor.factors(factors=c("block", LETTERS[1:4]), nlevels=c(6,6,6,4,2))
M2 <- planor.model( model=~block+(A+B+C)^2, estimate=~A+B+C )
F2.h <- planor.harmonize(factors=F2, model=M2, base=~A+B)
names(F2)
names(F2.h)
```

---

planor.model

*Model and Estimate Specifications for a Design Search*

---

### Description

Declare the factorial terms that must be considered as non-negligible and the factorial terms that must be estimable when the experiment will be analysed.

### Usage

```
planor.model(model, estimate, listofmodels, resolution, factors)
```

**Arguments**

<code>model</code>	main model formula. It contains all the non-negligible factorial terms.
<code>estimate</code>	optional formula specifying the factorial terms to estimate. If missing, it is considered that all factorial terms in <code>model</code> have to be estimated.
<code>listofmodels</code>	list of <code>c(model, estimate)</code> pairs, where <code>model</code> and <code>estimate</code> are formulae; using several pairs allows more flexibility in the design constraints (see Kobilinsky, 2005, or the split-plot example in the vignette); <code>estimate</code> is optional.
<code>resolution</code>	an integer larger than or equal to 3, to specify the design resolution. When set, the <code>model</code> and <code>estimate</code> arguments are ignored. See Note.
<code>factors</code>	a <code>designfactors</code> object, typically an output from <code>planor.factors</code> . It must be set only when the <code>resolution</code> argument is used.

**Value**

A list of `c(model, estimate)` pairs, where `model` and `estimate` are formulae.

**Note**

The user can specify:

- 1/ either, `model` or `listofmodels` or both,
- 2/ or, `resolution` and `factors`, and possibly `listofmodels`.

When `model` and `resolution` are both set, `model` is ignored.

The second case, — when `resolution` and `factors` are set —, causes the automatic generation of the main `c(model, estimate)` pair. Assuming  $S$  denotes the additive formula including all factors,

- if `resolution` is odd, the model formula is  $\sim(S)^{(resolution-1)/2}$ ,
- if `resolution` is even, the model formula is  $\sim(S)^{(resolution/2)}$  and the estimate formula is  $\sim(S)^{(resolution/2)-1}$ .

**Author(s)**

Monod, H. and Bouvier, A.

**Examples**

```
### Basic example
planor.model(model=~block + (A+B+C)^2, estimate=~(A+B+C)^2)
### Resolution: both calls to planor.model below are equivalent
planor.model(model=~(A+B+C+D)^2, estimate=~A+B+C+D)
myfactors <- planor.factors(factors=c(LETTERS[1:4]), nlevels=rep(2,4))
planor.model(resolution=4, factors=myfactors)
### Complicated examples
planor.model(~A+B+C+D+A:B, ~A+B+C+D, listofmodels=list(c(~E+F,~E)))
planor.model(~A+B+C+D+A:B, ~A+B+C+D, listofmodels=
  list(c(~E+F,~E), ~G, ~H, c(~M+N,~N)))
```

---

planor.randomize     *Randomize a Factorial Design from an Orthogonal Block Structure*

---

## Description

Randomize a factorial design according to a specified block structure formula.

## Usage

```
planor.randomize(blockformula, data, out.order, keep.initial=FALSE)
```

## Arguments

`blockformula` the block structure formula.

`data` a data frame.

`out.order` a list of data factors that will be used to order the rows of the randomized design; if missing, the factors of the block formula are used.

`keep.initial` if TRUE, the initial row order of the design is stored in column `InitialUNITS` of the returned dataframe.

## Value

The input data frame after randomization.

## Note

Each name in `blockformula` must correspond to a factor of the dataframe `data`. The only exception is `UNITS`. If `UNITS` is used in `blockformula` but absent from `data`, a factor is added to `data`, with one level per row. See the examples below for the usage of `UNITS` in `blockformula`.

## Author(s)

Monod, H. and Bouvier, A.

## References

Bailey, R.A. (1983) Generalized wreath products of permutation groups. *Proc. London Math. Soc.*, **47**, 69–82.

Kobilinsky A. (1989) Randomization of a cartesian block structure. Technical Report. Laboratoire de Biométrie. INRA. Versailles.

## Examples

```
### Block design
Design <- data.frame(block=rep(1:4, rep(2, 4)),
  treatment=c("A1", "B1", "A2", "B2", "A3", "B3", "A4", "B4"))
### No within-block randomization
planor.randomize(~block, data=Design)
### Blocks and units within blocks randomization
planor.randomize(~block/UNITS, data=Design)
### Row-Column design
```

```
RowColDes <- data.frame(row=rep(1:3,rep(3,3)),col=rep(1:3,3),
  treatment=LETTERS[c(1:3,2,3,1,3,1,2)],
  oldRow=rep(1:3,rep(3,3)),oldCol=rep(1:3,3))
planor.randomize(~row*col, data=RowColDes)
```

---

planordesign-class *Class planordesign*

---

## Description

An S4 class to represent a final design.

## Objects from the Class

Objects can be created by calls to method `planor.design` applied on an object of class `designkey` or on an object of class `listofkeyrings`, and by calls to `regular.design` when argument `output` is equal to 'planordesign'.

## Slots

**design** a dataframe containing the final design.

**factors** an object of class `designfactors` which contains the factors specifications.

**model** a list containing the model and estimate specifications.

**designkey** a list of the `keymatrix` objects used to create the object.

**nunits** the number of units of the design.

**recursive** a logical equal to TRUE if the design has been constructed recursively.

## Methods

**getDesign** extract a design data frame. See `getDesign` method.

**as.data.frame** coerce into a data frame. See `as.data.frame` method.

## Author(s)

Monod, H. and Bouvier, A.

## Examples

```
showClass("planordesign")
### Creation of a listofdesignkeys object
K0 <- planor.designkey(factors=c("R","C","U","A","B1","B2"),
  nlevels=c(3,2,2,3,2,2), model=~R*C + (A+B1+B2)^2, estimate=~A:B1+A:B2,
  nunits=12, base=~R+C+U, max.sol=2)
### Creation of a planordesign object from K0
P0 <- planor.design(key=K0, select=1)
show(P0)
```

regular.design

*Construct and Randomize a Regular Factorial Design***Description**

Construct and randomize a regular factorial design.

**Usage**

```
regular.design(factors = NULL, nlevels = NULL, block = NULL,
  ordered = NULL, hierarchy = NULL, model = NULL, estimate = NULL,
  listofmodels = NULL, resolution = NULL, nunits = NULL,
  base = NULL, randomize = NULL, randomsearch = FALSE,
  output = "planordesign", verbose = FALSE, ...)
```

**Arguments**

factors	an object of class <a href="#">designfactors</a> , typically an output from <a href="#">planor.factors</a> ). Otherwise the arguments factors, nlevels, ordered, hierarchy follow the syntax of <a href="#">planor.factors</a> .
nlevels	see <a href="#">planor.factors</a> . Ignored if factors is of class <a href="#">designfactors</a> .
block	see <a href="#">planor.factors</a> . Ignored if factors is of class <a href="#">designfactors</a> .
ordered	see <a href="#">planor.factors</a> . Ignored if factors is of class <a href="#">designfactors</a> .
hierarchy	see <a href="#">planor.factors</a> . Ignored if factors is of class <a href="#">designfactors</a> .
model	a list of model-estimate pairs of formulae, typically an output from <a href="#">planor.model</a> . Otherwise the arguments model, estimate, listofmodels and resolution follow the syntax of <a href="#">planor.model</a> .
estimate	see <a href="#">planor.model</a> . Ignored if model is a list.
listofmodels	see <a href="#">planor.model</a> . Ignored if model is a list.
resolution	see <a href="#">planor.model</a> . Ignored if model is a list. When set and there is no hierarchy, a faster algorithm is used which exploits the symmetries.
nunits	see <a href="#">planor.designkey</a> .
base	see <a href="#">planor.designkey</a> .
randomize	an optional formula to randomize the design.
randomsearch	see <a href="#">planor.designkey</a> .
output	a string to specify the class of the output value: either a <code>data.frame</code> or a <a href="#">planordesign</a> object.
verbose	a logical to set to TRUE for verbose display.
...	additional arguments, in particular those related to randomization

**Value**

An object of class `data.frame` or [planordesign](#), depending on the output argument.

**Author(s)**

Monod, H. and Bouvier, A.



**See Also**

`planor.factors`, `planor.model`, and the classes `designfactors`, `listofkeyrings`, `listofdesignkeys`

**Examples**

```
mydesign <- regular.design(factors=c("block", LETTERS[1:4]),
  nlevels=rep(3,5), model=~block + (A+B+C+D)^2, estimate=~A+B+C+D,
  nunits=3^3, randomize=~block/UNITS)
print(mydesign)
```

---

show-methods

*Display Objects*


---

**Description**

Methods to display the design key matrices.

**Usage**

```
## S4 method for signature 'designkey'
show(object)

## S4 method for signature 'keymatrix'
show(object)

## S4 method for signature 'keyring'
show(object)

## S4 method for signature 'listofdesignkeys'
show(object)

## S4 method for signature 'listofkeyrings'
show(object)
```

**Arguments**

`object`                      `object` of the class

**Details**

The slot `pseudo.info` of the objects of class `keymatrix` is invisible.

**Value**

NULL

**Note**

- An R option named `planor.max.print` is set. It is equal to the number of printed rows and columns in the display of **planor** matrices. Default is 20. You can change its value by using the function `options()` (see `?options`).
- This method is automatically invoked when objects of the class are displayed (see examples).

**See Also**

Classes where this method applies: [designkey](#), [keymatrix](#), [keyring](#), [listofdesignkeys](#), [listofkeyrings](#)

**Examples**

```
### Creation of a listofdesignkeys object
K0 <- planor.designkey(factors=c("R", "C", "U", "A", "B1", "B2"),
  nlevels=c(3,2,2,3,2,2), model=~R*C + (A+B1+B2)^2, estimate=~A:B1+A:B2,
  nunits=12, base=~R+C+U, max.sol=2)
### Method show applied on a keymatrix object
show(K0[[1]][[1]])
### Method show applied on a designkey object
show(K0[1])
### Method show applied on the listofdesignkeys object
show(K0)
K0 # same

### Creation of a listofkeyrings object
K0 <- planor.designkey(factors=c(LETTERS[1:4], "block"), nlevels=rep(3,5),
  model=~block+(A+B+C+D)^2, estimate=~A+B+C+D,
  nunits=3^3, base=~A+B+C, max.sol=2)
### Method show applied on a keyring object
show(K0[[1]])
print(K0[[1]]) # same
K0[[1]] # same
### Method show applied on the listofkeyrings object
show(K0)
```

summary-methods

*Summarize the Design Properties***Description**

Methods to summarize the design properties of an object, by printing the summary of each key matrix.

**Usage**

```
## S4 method for signature 'designkey'
summary(object, show="dtbw", save="k", ...)

## S4 method for signature 'keymatrix'
summary(object, fact, block, show="dtbw", save="k", ...)

## S4 method for signature 'keyring'
summary(object, show="tbw", save="kw", ...)

## S4 method for signature 'listofdesignkeys'
summary(object, show="tbw", save="kw", ...)

## S4 method for signature 'listofkeyrings'
```

```
summary(object, show="tbw", save="kw", ...)

## S4 method for signature 'planordesign'
summary(object, fact, block, show="dtbw", save="k", ...)
```

### Arguments

<code>object</code>	an object of the class.
<code>fact</code>	a character or numeric vector of parent factor names for the columns of the object.
<code>block</code>	a logical vector to identify the columns of the object associated with a block factor
<code>show</code>	an optional string to identify the type of information to display. The recognized letters are: ‘d’ for the design keys matrices, ‘t’ for the treatment effects confounded with the mean, ‘b’ for the block-and-treatment effects confounded with the mean, ‘w’ for the weight profiles.
<code>save</code>	an optional string to identify the type of information to return. The recognized letters are: ‘k’ for the kernel matrices, ‘w’ for the weight profiles of the treatment effects confounded with the mean.
<code>...</code>	ignored.

### Details

The amount of display depends on the value of the argument `show`, and the type of returned information depends on the value of the argument `save`.

- When `object` is of class `keymatrix`, “summary” prints the key matrix, the factorial effects confounded with the mean, and the weight profiles of the effects confounded with the mean, according to the value of the argument `show`.  
The `keymatrix` argument being denoted by `key`,
  - The rows of `key` are associated with units factors (or pseudofactors) while its columns are associated with treatment or block factors (or pseudofactors).
  - The vectors in the arguments `fact` and `block` give information on the treatment and block factors, so that their length is expected to be equal to the number of columns of `key`.
  - If missing, `fact` attributes a distinct parent factor to each column of `key` and `block` is set to TRUE for all columns.
 “summary” returns a list with the components required by the argument `save`.
- When `object` is of class `designkey`, “summary” prints the summary of each of the key matrices. It returns a list with as many components as key matrices, each one with the components required by the argument `save`.
- When `object` is of class `listofdesignkeys`, “summary” prints the summary of each key matrix in each design key. It returns a list with as many components as design keys, each one is a list of the key matrices summaries.
- When `object` is of class `listofkeyrings`, “summary” prints the summary of each key matrix in each keyring. It returns a list with as many components as keyrings, each one is a list of the key matrices summaries.
- When `object` is of class `keyring`, “summary” prints the summary of each of its key matrices. It returns a list with as many components as key matrices.
- When `object` is a `planordesign`, this function is the summary method applied on each of the `keymatrix` objects contained in its `designkey` slot.

**Value**

A list. See Details

Information returned for each key matrix depends on the argument `save`.

- When `save` includes the character 'k', the returned list has a component named 'k'. It is a matrix, the columns of which are kernel generators of the key matrices.
- When `save` includes the character 'w', the returned list has a component named 'w', which contains the weight profiles of the effects confounded with the mean.

**Note**

An R option named `planor.max.print` is set. It is equal to the number of printed rows and columns in the display of **planor** matrices. Default is 20. You can change its value by using the function `options()` (see `?options`).

**See Also**

Classes where this method applies: [designkey](#), [keymatrix](#), [keyring](#), [listofdesignkeys](#), [listofkeyrings](#), [planor](#), [design](#)

**Examples**

```
### Creation of a listofdesignkeys object
K0 <- planor.designkey(factors=c("R", "C", "U", "A", "B1", "B2"),
  nlevels=c(3,2,2,3,2,2), model=~R*C + (A+B1+B2)^2, estimate=~A:B1+A:B2,
  nunits=12, base=~R+C+U, max.sol=2)
### Method summary applied on a keymatrix object
r <- summary(K0[[1]][[1]])
### Method summary applied on a designkey object
summary(K0[1], save=NULL)
### Method summary applied on the listofdesignkeys object
r <-summary(K0, show="dt")

### Creation of a listofkeyrings object
K0 <- planor.designkey(factors=c(LETTERS[1:4], "block"), nlevels=rep(3,5),
  model=~block+(A+B+C+D)^2, estimate=~A+B+C+D,
  nunits=3^3, base=~A+B+C, max.sol=2)
### Method summary applied on the keymatrix object
r <-summary(K0[[1]][[1]])
### Method summary applied on the keyring object
r <-summary(K0[[1]])
### Method summary applied on the listofkeyrings object
r <- summary(K0, show="dtb", save ="k")
print(r)
```

# Index

## \*Topic classes

- designfactors-class, 6
- designkey-class, 7
- keymatrix-class, 9
- keyring-class, 10
- listofdesignkeys-class, 11
- listofkeyrings-class, 12
- planordesign-class, 23

## \*Topic design

- alias-methods, 3
- bind-methods, 5
- designfactors-class, 6
- designkey-class, 7
- getDesign-methods, 8
- keymatrix-class, 9
- keyring-class, 10
- listofdesignkeys-class, 11
- listofkeyrings-class, 12
- pick-methods, 14
- planor-package, 2
- planor.design-methods, 15
- planor.designkey, 16
- planor.factors, 18
- planor.harmonize, 19
- planor.model, 20
- planor.randomize, 22
- planordesign-class, 23
- regular.design, 24
- show-methods, 25

## \*Topic methods

- alias-methods, 3
- as.data.frame.planordesign, 5
- bind-methods, 5
- getDesign-methods, 8
- pick-methods, 14
- planor.design-methods, 15
- show-methods, 25
- summary-methods, 26

## \*Topic package

- planor-package, 2
- [, designfactors, ANY, ANY, ANY-method  
(*designfactors-class*), 6
- [, designfactors-method

- (*pick-methods*), 14
- [, listofdesignkeys, ANY, ANY, ANY-method  
(*listofdesignkeys-class*),  
11
- [, listofdesignkeys-method  
(*pick-methods*), 14
- [, listofkeyrings, ANY, ANY, ANY-method  
(*listofkeyrings-class*), 12
- [, listofkeyrings-method  
(*pick-methods*), 14
- [, planordesign, ANY, ANY, ANY-method  
(*planordesign-class*), 23
- [, planordesign-method  
(*pick-methods*), 14

alias, 8, 9, 12, 19

alias, designkey-method  
(*alias-methods*), 3

alias, keymatrix-method  
(*alias-methods*), 3

alias, listofdesignkeys-method  
(*alias-methods*), 3

alias, listofkeyrings-method  
(*alias-methods*), 3

alias, planordesign-method  
(*alias-methods*), 3

alias-methods, 3

alias.designkey (*alias-methods*), 3

alias.keymatrix (*alias-methods*), 3

alias.listofdesignkeys  
(*alias-methods*), 3

alias.listofkeyrings  
(*alias-methods*), 3

alias.planordesign  
(*alias-methods*), 3

array, 9

as.data.frame, 8, 23

as.data.frame, planordesign-method  
(*as.data.frame.planordesign*),  
5

as.data.frame.planordesign, 5

bind, 6

bind (*bind-methods*), 5

- bind, designfactors, designfactors-method  
(*bind-methods*), 5
- bind-methods, 5
- designfactors, 6, 7, 10–12, 16, 18–21, 23–25
- designfactors-class, 6
- designkey, 4, 9, 11, 13–15, 17, 23, 26–28
- designkey-class, 7
- getDesign, 23
- getDesign (*getDesign-methods*), 8
- getDesign, planordesign-method  
(*getDesign-methods*), 8
- getDesign-method  
(*getDesign-methods*), 8
- getDesign-methods, 8
- keymatrix, 4, 7, 10, 17, 23, 25–28
- keymatrix-class, 9
- keyring, 9, 10, 12, 26–28
- keyring-class, 10
- length, designfactors-method  
(*designfactors-class*), 6
- list, 8, 10–12
- listofdesignkeys, 4, 7, 14, 15, 17, 18, 25–28
- listofdesignkeys-class, 11
- listofkeyrings, 4, 7, 10, 14, 15, 17, 18, 23, 25–28
- listofkeyrings-class, 12
- makedesignkey, 13
- matrix, 9
- names, designfactors-method  
(*designfactors-class*), 6
- pick, 11, 12
- pick (*pick-methods*), 14
- pick, listofdesignkeys-method  
(*pick-methods*), 14
- pick, listofkeyrings-method  
(*pick-methods*), 14
- pick-method (*pick-methods*), 14
- pick-methods, 14
- pick.listofkeyrings, 10
- planor-package, 2
- planor.design, 2, 8, 11, 12, 23
- planor.design  
(*planor.design-methods*), 15
- planor.design, designkey-method  
(*planor.design-methods*), 15
- planor.design, listofdesignkeys-method  
(*planor.design-methods*), 15
- planor.design, listofkeyrings-method  
(*planor.design-methods*), 15
- planor.design, numeric-method  
(*planor.design-methods*), 15
- planor.design-methods, 15
- planor.designkey, 2, 6, 10–13, 16, 17, 24
- planor.factors, 2, 6, 7, 16, 18, 18–21, 24, 25
- planor.harmonize, 19
- planor.model, 2, 16, 18, 20, 20, 24, 25
- planor.randomize, 15, 22
- planordesign, 4, 5, 8, 15, 24, 27, 28
- planordesign-class, 23
- regular.design, 2, 23, 24
- show, 8–12
- show, designkey-method  
(*show-methods*), 25
- show, keymatrix-method  
(*show-methods*), 25
- show, keyring-method  
(*show-methods*), 25
- show, listofdesignkeys-method  
(*show-methods*), 25
- show, listofkeyrings-method  
(*show-methods*), 25
- show-method (*show-methods*), 25
- show-methods, 25
- structure, 9
- summary, 8–12
- summary, designkey-method  
(*summary-methods*), 26
- summary, keymatrix-method  
(*summary-methods*), 26
- summary, keyring-method  
(*summary-methods*), 26
- summary, listofdesignkeys-method  
(*summary-methods*), 26
- summary, listofkeyrings-method  
(*summary-methods*), 26
- summary, planordesign-method  
(*summary-methods*), 26
- summary-methods, 26
- summary.designkey  
(*summary-methods*), 26
- summary.keymatrix, 10
- summary.keymatrix  
(*summary-methods*), 26

`summary.keyring`  
    (*summary-methods*), [26](#)  
`summary.listofdesignkeys`  
    (*summary-methods*), [26](#)  
`summary.listofkeyrings`  
    (*summary-methods*), [26](#)  
`summary.planordesign`  
    (*summary-methods*), [26](#)  
  
`vector`, [8–12](#)