Package ‘OpenRepGrid’

May 16, 2023

License GPL (>= 2)

Title Tools to Analyze Repertory Grid Data

LazyData yes

Type Package

LazyLoad yes

Description Analyze repertory grids, a qualitative-quantitative data collection technique devised by George A. Kelly in the 1950s. Today, grids are used across various domains ranging from clinical psychology to marketing. The package contains functions to quantitatively analyze and visualize repertory grid data.

Version 0.1.14

Date 2023-05-16

Encoding UTF-8

URL https://github.com/markheckmann/OpenRepGrid

Imports methods, graphics, grid, utils, stats, grDevices, crayon, plyr, stringr, abind, rgl, colorspace, psych, XML, pvclust, openxlsx, tidyr, dplyr, scales, igraph

Collate 'bertin.r' 'calc.r' 'data-openrepgrid.r' 'dev-functions.r' 'distance.R' 'double-entry.R' 'export.r' 'globals.R' 'gmMain.r' 'gridlist.R' 'import.r' 'measures.r' 'onair.r' 'openrepgrid.r' 'perturbate.R' 'repgrid.r' 'repgrid-basicops.r' 'repgrid-constructs.r' 'repgrid-elements.r' 'repgrid-output.r' 'repgrid-plots.r' 'repgrid-ratings.r' 'resampling.R' 'rgl-3d.r' 'settings.r' 'utils-import.r' 'utils.r' 'zzz.r'

RoxygenNote 7.2.3

NeedsCompilation no

Suggests testthat (>= 2.1.0), covr

Author Mark Heckmann [aut, cre, cph] (<https://orcid.org/0000-0002-0736-7417>),
Alejandro García Gutiérrez [ctb],
Diego Vitali [ctb]

Maintainer Mark Heckmann <heckmann.mark@gmail.com>

Repository CRAN

Date/Publication 2023-05-16 15:00:11 UTC

R topics documented:

+,repgrid,repgrid-method . . . 4
addAvgElement . . . 5
addConstruct . . . 5
addElement . . . 7
alignByIdeal . . . 8
alignByLoadings . . . 9
bertin . . . 11
bertinCluster . . . 13
bindConstructs . . . 15
biplot2d . . . 17
biplot3d . . . 22
biplotEsa2d . . . 25
biplotEsa3d . . . 26
biplotEsaPseudo3d . . . 27
biplotPseudo3d . . . 28
biplotSimple . . . 31
biplotSlater2d . . . 33
biplotSlater3d . . . 34
biplotSlaterPseudo3d . . . 35
center . . . 36
cluster . . . 37
clusterBoot . . . 39
constructCor . . . 41
constructD . . . 42
constructPca . . . 43
constructPcaLoadings . . . 45
constructRmsCor . . . 45
constructs . . . 46
data-bell2010 . . . 48
data-bellmcgorry1992 . . . 48
data-boeker . . . 49
data-fbb2003 . . . 49
data-feixas2004 . . . 49
data-leach2001 . . . 50
data-mackay1992 . . . 50
data-raeithel . . . 51
data-slater1977a . . . 51
data-slater1977b . . . 51
distance . . . 52
distanceHartmann . . . 53
distanceNormalized . . . 56
distanceSlater . . . 58
elementCor . . . 60
elementRmsCor . . . 61
elements . . . 62
gridlist . . . 63

grids_leave_n_out 64
home 65
importExcel 66
importGridcor 67
importGridstat 68
importGridsuite 70
importScivesco 71
importTxt 73
indexBias 75
indexBieri 76
indexConflict1 77
indexConflict2 78
indexConflict3 80
indexDilemma 82
indexDilemmatic 87
indexIntensity 88
indexPolarization 90
indexPvaff 91
indexSelfConstruction 92
indexVariability 93
is.repgrid 94
left 95
makeRepgrid 96
matches 97
midpoint 98
normalize 98
OpenRepGrid 99
OpenRepGrid-overview 100
permuteConstructs 104
perturbate 105
randomGrid 105
randomGrids 107
ratings 108
reorder.repgrid 109
reorder2d 110
saveAsExcel 111
saveAsTxt 112
setConstructAttr 113
setElementAttr 115
setScale 116
settings 117
settingsLoad 118
settingsSave 118
shift 118
show,repgrid-method 119
statsElements 120
swapConstructs 121
swapElements 122

swapPoles 123
[,repgrid-method 124
[<-,repgrid-method 124

Index **126**

`+,repgrid,repgrid-method`
Concatenate repgrid objects.

Description

Simple concatenation of repgrid objects or list containing repgrid objects using the '+' operator.

Usage

```
## S4 method for signature 'repgrid,repgrid'
e1 + e2

## S4 method for signature 'list,repgrid'
e1 + e2

## S4 method for signature 'repgrid,list'
e1 + e2
```

Arguments

`e1, e2`	A `repgrid` object.

Details

Methods for `"+"` function.

Examples

```
x <- bell2010
x + x
x + list(x,x)
list(x,x) + x
```

`addAvgElement` *Add a new average element*

Description

A new element is added to the grid. The scores are the arithmetic means across all selected elements.

Usage

```
addAvgElement(x, name = "avg", i, digits = Inf)
```

Arguments

`x`	A `repgrid` object.
`name`	Name of new element.
`i`	Indexes of elements to be averaged across. Negative indexes can be used to exclude elements from the complete set. Duplicate indexes are allowed but a warning is issued.
`digits`	Digits to round mean value to. By default no rounding is used (`digits = Inf`). Use `digits = 0` to round to nearest integer, i.e. only using original integer rating scores.

Value

A `repgrid` object with additional new element.

Examples

```
addAvgElement(feixas2004, "others", i = 2:12)
addAvgElement(feixas2004, "others", i = 2:12, digits = 0) # integers

# exluding elements via negative indexes
addAvgElement(feixas2004, "others", i = c(-1,-13))
```

`addConstruct` *Add a new construct to an existing grid object.*

Description

Add a new construct to an existing grid object.

Usage

```
addConstruct(
  x,
  l.name = NA,
  r.name = NA,
  scores = NA,
  l.preferred = NA,
  r.preferred = NA,
  l.emerged = NA,
  r.emerged = NA,
  position = NA,
  side = "pre"
)
```

Arguments

x	repgrid object.
l.name	Name of the left pole (character string).
r.name	Name of the right pole (character string).
scores	Numerical ratings for the new construct row (length must match number of elements in the grid).
l.preferred	Is the left one the preferred pole? (logical).
r.preferred	Is the right one the preferred pole? (logical).
l.emerged	Is the left one the emergent pole? (logical).
r.emerged	Is the right one the emergent pole? (logical).
position	An integer at which row the construct will be added. TODO. Does not work properly.
side	Not yet in use.

Value

repgrid object.

See Also

addElement

Examples

```
## Not run:

  # show grid
  bell2010
  addConstruct(bell2010, "left pole", "pole right", c(3,1,3,2,5,4,6,3,7,1))


## End(Not run)
```

`addElement` *Add an element to an existing grid.*

Description

Add an element to an existing grid.

Usage

```
addElement(
  x,
  name = NA,
  scores = NA,
  abbreviation = NA,
  status = NA,
  position = NA,
  side = "pre"
)
```

Arguments

`x`	`repgrid` object.
`name`	Name of the new element (character string).
`scores`	Numerical ratings for the new element column (length must match number of constructs in the grid).
`abbreviation`	Abbreviation for element name.
`status`	Element status (not yet in use).
`position`	An integer at which column the element will be added. TODO: Does not work properly yet.
`side`	Not yet in use.

Value

`repgrid` object

See Also

`addConstruct`

Examples

```
## Not run:

   bell2010
   addElement(bell2010, "new element", c(1,2,5,4,3,6,5,2,7))
```

```
## End(Not run)
```

---

alignByIdeal *Align constructs using the ideal element to gain pole preferences.*

Description

The direction of the constructs in a grid is arbitrary and a reflection of a scale does not affect the information contained in the grid. Nonetheless, the direction of a scale has an effect on inter-element correlations (Mackay, 1992) and on the spatial representation and clustering of the grid (Bell, 2010). Hence, it is desirable to follow a protocol to align constructs that will render unique results. A common approach is to align constructs by pole preference, i. e. aligning all positive and negative poles. This can e. g. be achieved using `swapPoles`. If an ideal element is present, this element can be used to identify the positive and negative pole. The function `alignByIdeal` will align the constructs accordingly. Note that this approach does not always yield definite results as sometimes ratings do not show a clear preference for one pole (Winter, Bell & Watson, 2010). If a preference cannot be determined definitely, the construct direction remains unchanged (a warning is issued in that case).

Usage

```
alignByIdeal(x, ideal, high = TRUE)
```

Arguments

`x`	`repgrid` object
`ideal`	Number of the element that is used for alignment (the ideal).
`high`	Logical. Whether to align the constructs so the ideal will have high ratings on the constructs (i.e. `TRUE`, default) or low ratings (`FALSE`). High scores will lead to the preference pole on the right side, low scores will align the preference pole on the left side.

Value

`repgrid` object with aligned constructs.

References

Bell, R. C. (2010). A note on aligning constructs. *Personal Construct Theory & Practice, 7*, 42-48.

Mackay, N. (1992). Identification, Reflection, and Correlation: Problems in the bases of repertory grid measures. *International Journal of Personal Construct Psychology, 5*(1), 57-75.

Winter, D. A., Bell, R. C., & Watson, S. (2010). Midpoint ratings on personal constructs: Constriction or the middle way? *Journal of Constructivist Psychology, 23*(4), 337-356.

See Also

alignByLoadings

Examples

```
## Not run:

  feixas2004                          # original grid
  alignByIdeal(feixas2004, 13)        # aligned with preference pole on the right

  raeithel                            # original grid
  alignByIdeal(raeithel, 3, high=FALSE)  # aligned with preference pole on the left


## End(Not run)
```

---

alignByLoadings   *Align constructs by loadings on first principal component.*

Description

In case a construct loads negatively on the first principal component, the function alignByLoadings will reverse it so that all constructs have positive loadings on the first principal component (see detail section for more).

Usage

```
alignByLoadings(x, trim = 20, index = TRUE)
```

Arguments

x	repgrid object.
trim	The number of characters a construct is trimmed to (default is 10). If NA no trimming is done. Trimming simply saves space when displaying the output.
index	Whether to print the number of the construct (e.g. for correlation matrices). The default is TRUE.

Details

The direction of the constructs in a grid is arbitrary and a reflection of a scale does not affect the information contained in the grid. Nonetheless, the direction of a scale has an effect on inter-element correlations (Mackay, 1992) and on the spatial representation and clustering of the grid (Bell, 2010). Hence, it is desirable to follow a protocol to align constructs that will render unique results. A common approach is to align constructs by pole preference, but this information is not always accessible. Bell (2010) proposed another solution for the problem of construct alignment. As a unique protocol he suggests to align constructs in a way so they all have positive loadings on the first component of a grid PCA.

Value

An object of class `alignByLoadings` containing a list of calculations with the following entries:

`cor.before`	Construct correlation matrix before reversal
`loadings.before`	Loadings on PCs before reversal
`reversed`	Constructs that have been reversed
`cor.after`	Construct correlation matrix after reversal
`loadings.after`	Loadings on PCs after reversal

Note

Bell (2010) proposed a solution for the problem of construct alignment. As construct reversal has an effect on element correlation and thus on any measure that based on element correlation (Mackay, 1992), it is desirable to have a standard method for construct alignment independently from its semantics (preferred pole etc.). Bell (2010) proposes to align constructs in a way so they all have positive loadings on the first component of a grid PCA.

References

Bell, R. C. (2010). A note on aligning constructs. *Personal Construct Theory & Practice, 7*, 42-48.

Mackay, N. (1992). Identification, Reflection, and Correlation: Problems in the bases of repertory grid measures. *International Journal of Personal Construct Psychology, 5*(1), 57-75.

See Also

`alignByIdeal`

Examples

```
# reproduction of the example in the Bell (2010)
# constructs aligned by loadings on PC 1
bell2010
alignByLoadings(bell2010)

# save results
a <- alignByLoadings(bell2010)

# modify printing of resukts
print(a, digits=5)

# access results for further processing
names(a)
a$cor.before
a$loadings.before
a$reversed
a$cor.after
a$loadings.after
```

bertin *Make Bertin display of grid data.*

Description

One of the most popular ways of displaying grid data has been adopted from Bertin's (1974) graphical proposals, which have had an immense influence onto data visualization. One of the most appealing ideas presented by Bertin is the concept of the reorderable matrix. It is comprised of graphical displays for each cell, allowing to identify structures by eye-balling reordered versions of the data matrix (see Bertin, 1974). In the context of repertory grids, the display is made up of a simple colored rectangle where the color denotes the corresponding score. Bright values correspond to low, dark to high scores. For an example of how to analyze a Bertin display see e.g. Dick (2000) and Raeithel (1998).

Usage

```
bertin(
  x,
  colors = c("white", "black"),
  showvalues = TRUE,
  xlim = c(0.2, 0.8),
  ylim = c(0, 0.6),
  margins = c(0, 1, 1),
  cex.elements = 0.7,
  cex.constructs = 0.7,
  cex.text = 0.6,
  col.text = NA,
  border = "white",
  lheight = 0.75,
  id = c(T, T),
  cc = 0,
  cr = 0,
  cc.old = 0,
  cr.old = 0,
  col.mark.fill = "#FCF5A4",
  print = TRUE,
  ...
)
```

Arguments

x	repgrid object.
colors	Vector. Two or more colors defining the color ramp for the bertin (default c("white", "black")).
showvalues	Logical. Whether scores are shown in bertin
xlim	Vector. Left and right limits inner bertin (default c(.2, .8)).

ylim	Vector. Lower and upper limits of inner bertin default(c(.0, .6)).
margins	Vector of length three (default margins=c(0,1,1)). 1st element denotes the left, 2nd the upper and 3rd the right margin in npc coordinates (i.e. 0 to zero).
cex.elements	Numeric. Text size of element labels (default .7).
cex.constructs	Numeric. Text size of construct labels (default .7).
cex.text	Numeric. Text size of scores in bertin cells (default .7).
col.text	Color of scores in bertin (default NA). By default the color of the text is chosen according to the background color. If the background ist bright the text will be black and vice versa. When a color is specified the color is set independent of background.
border	Border color of the bertin cells (default white).
lheight	Line height for constructs.
id	Logical. Whether to print id number for constructs and elements respectively (default c(T,T)).
cc	Numeric. Current column to mark.
cr	Numeric. Current row to mark.
cc.old	Numeric. Column to unmark.
cr.old	Numeric. Row to unmark.
col.mark.fill	Color of marked row or column (default "#FCF5A4").
print	Print whole bertin. If FALSE only current and old row and column are printed.
...	Optional arguments to be passed on to bertinBase.

Value

NULL just for the side effects, i.e. printing.

References

Bertin, J. (1974). *Graphische Semiologie: Diagramme, Netze, Karten*. Berlin, New York: de Gruyter.

Dick, M. (2000). The Use of Narrative Grid Interviews in Psychological Mobility Research. *Forum Qualitative Sozialforschung / Forum: Qualitative Social Research, 1*(2).

Raeithel, A. (1998). Kooperative Modellproduktion von Professionellen und Klienten - erlauetert am Beispiel des Repertory Grid. *Selbstorganisation, Kooperation, Zeichenprozess: Arbeiten zu einer kulturwissenschaftlichen, anwendungsbezogenen Psychologie* (pp. 209-254). Opladen: Westdeutscher Verlag.

Examples

```
## Not run:

   bertin(feixas2004)
   bertin(feixas2004, c("white", "darkblue"))
   bertin(feixas2004, showvalues=F)
```

```
    bertin(feixas2004, border="grey")
    bertin(feixas2004, cex.text=.9)
    bertin(feixas2004, id=c(F, F))

    bertin(feixas2004, cc=3, cr=4)
    bertin(feixas2004, cc=3, cr=4, col.mark.fill="#e6e6e6")

## End(Not run)
```

`bertinCluster` *Bertin display with corresponding cluster analysis.*

Description

Element columns and constructs rows are ordered according to cluster criterion. Various distance measures as well as cluster methods are supported.

Usage

```
bertinCluster(
  x,
  dmethod = c("euclidean", "euclidean"),
  cmethod = c("ward", "ward"),
  p = c(2, 2),
  align = TRUE,
  trim = NA,
  type = c("triangle"),
  xsegs = c(0, 0.2, 0.7, 0.9, 1),
  ysegs = c(0, 0.1, 0.7, 1),
  x.off = 0.01,
  y.off = 0.01,
  cex.axis = 0.6,
  col.axis = grey(0.4),
  draw.axis = TRUE,
  ...
)
```

Arguments

`x` `repgrid` object.

`dmethod` The distance measure to be used. This must be one of `"euclidean"`, `"maximum"`, `"manhattan"`, `"canberra"`, `"binary"`, or `"minkowski"`. Default is `"euclidean"`. Any unambiguous substring can be given (e.g. `"euc"` for `"euclidean"`). A vector of length two can be passed if a different distance measure for constructs and elements is wanted (e.g.`c("euclidean", "manhattan")`). This will apply euclidean distance to the constructs and manhattan distance to the elements. For additional information on the different types see `?dist`.

cmethod	The agglomeration method to be used. This should be (an unambiguous abbreviation of) one of "ward", "single", "complete", "average", "mcquitty", "median" or "centroid". Default is "ward". A vector of length two can be passed if a different cluster method for constructs and elements is wanted (e.g.c("ward", "euclidean")). This will apply ward clustering to the constructs and single linkage clustering to the elements. If only one of either constructs or elements is to be clustered the value NA can be supplied. E.g. to cluster elements only use c(NA, "ward").
p	The power of the Minkowski distance, in case "minkowski" is used as argument for dmethod. p can be a vector of length two if different powers are wanted for constructs and elements respectively (e.g. c(2,1)).
align	Whether the constructs should be aligned before clustering (default is TRUE). If not, the grid matrix is clustered as is. See Details section in function cluster for more information.
trim	The number of characters a construct is trimmed to (default is 10). If NA no trimming is done. Trimming simply saves space when displaying the output.
type	Type of dendrogram. Either or "triangle" (default) or "rectangle" form.
xsegs	Numeric vector of normal device coordinates (ndc i.e. 0 to 1) to mark the widths of the regions for the left labels, for the bertin display, for the right labels and for the vertical dendrogram (i.e. for the constructs).
ysegs	Numeric vector of normal device coordinates (ndc i.e. 0 to 1) to mark the heights of the regions for the horizontal dendrogram (i.e. for the elements), for the bertin display and for the element names.
x.off	Horizontal offset between construct labels and construct dendrogram and (default is 0.01 in normal device coordinates).
y.off	Vertical offset between bertin display and element dendrogram and (default is 0.01 in normal device coordinates).
cex.axis	cex for axis labels, default is .6.
col.axis	Color for axis and axis labels, default is grey(.4).
draw.axis	Whether to draw axis showing the distance metric for the dendrograms (default is TRUE).
...	additional parameters to be passed to function bertin.

Value

A list of two hclust object, for elements and constructs respectively.

See Also

cluster

Examples

```
## Not run:

   # default is euclidean distance and ward clustering
```

```
bertinCluster(bell2010)

### applying different distance measures and cluster methods

# euclidean distance and single linkage clustering
bertinCluster(bell2010, cmethod="single")
# manhattan distance and single linkage clustering
bertinCluster(bell2010, dmethod="manhattan", cm="single")
# minkowksi distance with power of 2 = euclidean distance
bertinCluster(bell2010, dm="mink", p=2)

### using different methods for constructs and elements

# ward clustering for constructs, single linkage for elements
bertinCluster(bell2010, cmethod=c("ward", "single"))
# euclidean distance measure for constructs, manhatten
# distance for elements
bertinCluster(bell2010, dmethod=c("euclidean", "man"))
# minkowski metric with different powers for constructs and elements
bertinCluster(bell2010, dmethod="mink", p=c(2,1)))

### clustering either constructs or elements only
# euclidean distance and ward clustering for constructs no
# clustering for elements
bertinCluster(bell2010, cmethod=c("ward", NA))
# euclidean distance and single linkage clustering for elements
# no clustering for constructs
bertinCluster(bell2010, cm=c(NA, "single"))

### changing the appearance
# different dendrogram type
bertinCluster(bell2010, type="rectangle")
# no axis drawn for dendrogram
bertinCluster(bell2010, draw.axis=F)

### passing on arguments to bertin function via ...
 # grey cell borders in bertin display
bertinCluster(bell2010, border="grey")
# omit printing of grid scores, i.e. colors only
bertinCluster(bell2010, showvalues=FALSE)

### changing the layout
# making the vertical dendrogram bigger
bertinCluster(bell2010, xsegs=c(0, .2, .5, .7, 1))
# making the horizontal dendrogram bigger
bertinCluster(bell2010, ysegs=c(0, .3, .8, 1))

## End(Not run)
```

`bindConstructs` *Concatenate the constructs of two or more grids.*

Description

I.e. the constructs are combined to form one long grid. The girds must have the same set of elements and an identical scale range. The order of the elements may differ.

Usage

```
bindConstructs(..., index = FALSE)
```

Arguments

`...`	One or more repgrid objects or a list containing `repgrid` object.
`index`	TODO. Logical (default `TRUE`). Whether to add an index at the end of each construct name so it remains clear from which grid each construct came.

Details

This function can be used in order to analyze multiple grids as one 'big grid' (eg. Slater, 1977, chap. 11).

Value

`repgrid` object with concatenated constructs.

References

Slater, P. (1977). *The measurement of intrapersonal space by grid technique*. London: Wiley.

Examples

```
a <- randomGrid()
b <- randomGrid()
b@elements <- rev(a@elements)   # reverse elements
bindConstructs(a, b)
bindConstructs(a, b, a)

# using lists of repgrid objects
bindConstructs(a, list(a, b))
```

biplot2d *Draw a two-dimensional biplot.*

Description

The biplot is the central way to create a joint plot of elements and constructs. Depending on the parameters chosen it contains information on the distances between elements and constructs. Also the relative values the elements have on a construct can be read off by projection the element onto the construct vector. A lot of parameters can be changed rendering different types of biplots (ESA, Slater's) and different looks (colors, text size). See the example section below to get started.

Usage

```
biplot2d(
  x,
  dim = c(1, 2),
  map.dim = 3,
  center = 1,
  normalize = 0,
  g = 0,
  h = 1 - g,
  col.active = NA,
  col.passive = NA,
  e.point.col = "black",
  e.point.cex = 0.9,
  e.label.col = "black",
  e.label.cex = 0.7,
  e.color.map = c(0.4, 1),
  c.point.col = "black",
  c.point.cex = 0,
  c.label.col = "black",
  c.label.cex = 0.7,
  c.color.map = c(0.4, 1),
  c.points.devangle = 91,
  c.labels.devangle = 91,
  c.points.show = TRUE,
  c.labels.show = TRUE,
  e.points.show = TRUE,
  e.labels.show = TRUE,
  inner.positioning = TRUE,
  outer.positioning = TRUE,
  c.labels.inside = FALSE,
  c.lines = TRUE,
  col.c.lines = grey(0.9),
  flipaxes = c(FALSE, FALSE),
  strokes.x = 0.1,
  strokes.y = 0.1,
```

```
    offsetting = TRUE,
    offset.labels = 0,
    offset.e = 1,
    axis.ext = 0.1,
    mai = c(0.2, 1.5, 0.2, 1.5),
    rect.margins = c(0.01, 0.01),
    srt = 45,
    cex.pos = 0.7,
    xpd = TRUE,
    unity = FALSE,
    unity3d = FALSE,
    scale.e = 0.9,
    zoom = 1,
    var.show = TRUE,
    var.cex = 0.7,
    var.col = grey(0.1),
    ...
  )
```

Arguments

x	repgrid object.
dim	Dimensions (i.e. principal components) to be used for biplot (default is c(1,2)).
map.dim	Third dimension (depth) used to map aesthetic attributes to (default is 3).
center	Numeric. The type of centering to be performed. 0= no centering, 1= row mean centering (construct), 2= column mean centering (elements), 3= double-centering (construct and element means), 4= midpoint centering of rows (constructs). The default is 1 (row centering).
normalize	A numeric value indicating along what direction (rows, columns) to normalize by standard deviations. 0 = none, 1= rows, 2 = columns (default is 0).
g	Power of the singular value matrix assigned to the left singular vectors, i.e. the constructs.
h	Power of the singular value matrix assigned to the right singular vectors, i.e. the elements.
col.active	Columns (elements) that are no supplementary points, i.e. they are used in the SVD to find principal components. default is to use all elements.
col.passive	Columns (elements) that are supplementary points, i.e. they are NOT used in the SVD but projected into the component space afterwards. They do not determine the solution. Default is NA, i.e. no elements are set supplementary.
e.point.col	Color of the element symbols. The default is "black". Two values can be entered that will create a color ramp. The values of map.dim are mapped onto the ramp. If only one color color value is supplied (e.g. "black") no mapping occurs and all elements will have the same color irrespective of their value on the map.dim dimension.
e.point.cex	Size of the element symbols. The default is .9. Two values can be entered that will create a size ramp. The values of map.dim are mapped onto the ramp.

If only one color size value is supplied (e.g. .8) no mapping occurs and all elements will have the same size irrespective of their value on the map.dim dimension.

e.label.col
: Color of the element label. The default is "black". Two values can be entered that will create a color ramp. The values of map.dim are mapped onto the ramp. If only one color color value is supplied (e.g. "black") no mapping occurs and all labels will have the same color irrespective of their value on the map.dim dimension.

e.label.cex
: Size of the element labels. The default is .7. Two values can be entered that will create a size ramp. The values of map.dim are mapped onto the ramp. If only one color size value is supplied (e.g. .7) no mapping occurs and all labels will have the same size irrespective of their value on the map.dim dimension.

e.color.map
: Value range to determine what range of the color ramp defined in e.color will be used for mapping the colors. Default is c(.4, ,1). Usually not important for the user.

c.point.col
: Color of the construct symbols. The default is "black". Two values can be entered that will create a color ramp. The values of map.dim are mapped onto the ramp. If only one color color value is supplied (e.g. "black") no mapping occurs and all construct will have the same color irrespective of their value on the map.dim dimension.

c.point.cex
: Size of the construct symbols. The default is .8. Two values can be entered that will create a size ramp. The values of map.dim are mapped onto the ramp. If only one color size value is supplied (e.g. .8) no mapping occurs and all construct will have the same size irrespective of their value on the map.dim dimension.

c.label.col
: Color of the construct label. The default is "black". Two values can be entered that will create a color ramp. The values of map.dim are mapped onto the ramp. If only one color color value is supplied (e.g. "black") no mapping occurs and all labels will have the same color irrespective of their value on the map.dim dimension.

c.label.cex
: Size of the construct labels. The default is .7. Two values can be entered that will create a size ramp. The values of map.dim are mapped onto the ramp. If only one color size value is supplied (e.g. .7) no mapping occurs and all labels will have the same size irrespective of their value on the map.dim dimension.

c.color.map
: Value range to determine what range of the color ramp defined in c.color will be used for mapping. Default is c(.4, ,1). Usually not important for the user.

c.points.devangle
: The deviation angle from the x-y plane in degrees. These can only be calculated if a third dimension map.dim is specified. Only the constructs that do not depart more than the specified degrees from the x-y plane will be printed. This facilitates the visual interpretation, as only vectors represented near the current plane are shown. Set the value to 91 (default) to show all vectors.

c.labels.devangle
: The deviation angle from the x-y plane in degrees. These can only be calculated if a third dimension map.dim is specified. Only the labels of constructs that do not depart more than the specified degrees from the x-y plane will be printed. Set the value to 91 (default) to show all construct labels.

c.points.show
: Whether the constructs are printed (default is TRUE). FALSE will suppress the printing of the constructs. To only print certain constructs a numeric vector can be provided (e.g. c(1:10)).

c.labels.show
: Whether the construct labels are printed (default is TRUE). FALSE will suppress the printing of the labels. To only print certain construct labels a numeric vector can be provided (e.g. c(1:10)).

e.points.show
: Whether the elements are printed (default is TRUE). FALSE will suppress the printing of the elements. To only print certain elements a numeric vector can be provided (e.g. c(1:10)).

e.labels.show
: Whether the element labels are printed (default is TRUE). FALSE will suppress the printing of the labels. To only print certain element labels a numeric vector can be provided (e.g. c(1:10)).

inner.positioning
: Logical. Whether to calculate positions to minimize overplotting of elements and construct labels (default isTRUE). Note that the positioning may slow down the plotting.

outer.positioning
: Logical. Whether to calculate positions to minimize overplotting of of construct labels on the outer borders (default isTRUE). Note that the positioning may slow down the plotting.

c.labels.inside
: Logical. Whether to print construct labels next to the points. Can be useful during inspection of the plot (default FALSE).

c.lines
: Logical. Whether construct lines from the center of the biplot to the surrounding box are drawn (default is FALSE).

col.c.lines
: The color of the construct lines from the center to the borders of the plot (default is gray(.9)).

flipaxes
: Logical vector of length two. Whether x and y axes are reversed (default is c(F,F)).

strokes.x
: Length of outer strokes in x direction in NDC.

strokes.y
: Length of outer strokes in y direction in NDC.

offsetting
: Do offsetting? (TODO)

offset.labels
: Offsetting parameter for labels (TODO).

offset.e
: offsetting parameter for elements (TODO).

axis.ext
: Axis extension factor (default is .1). A bigger value will zoom out the plot.

mai
: Margins available for plotting the labels in inch (default is c(.2, 1.5, .2, 1.5)).

rect.margins
: Vector of length two (default is c(.07, .07)). Two values specifying the additional horizontal and vertical margin around each label.

srt
: Angle to rotate construct label text. Only used in case offsetting=FALSE.

cex.pos
: Cex parameter used during positioning of labels if prompted. Does usually not have to be changed by user.

xpd	Logical (default is TRUE). Whether to extend text labels over figure region. Usually not needed by the user.
unity	Scale elements and constructs coordinates to unit scale in 2D (maximum of 1) so they are printed more neatly (default TRUE).
unity3d	Scale elements and constructs coordinates to unit scale in 3D (maximum of 1) so they are printed more neatly (default TRUE).
scale.e	Scaling factor for element vectors. Will cause element points to move a bit more to the center. (but only if unity or unity3d is TRUE). This argument is for visual appeal only.
zoom	Scaling factor for all vectors. Can be used to zoom the plot in and out (default 1).
var.show	Show explained sum-of-squares in biplot? (default TRUE).
var.cex	The cex value for the percentages shown in the plot.
var.col	The color value of the percentages shown in the plot.
...	parameters passed on to come.

Details

For the construction of a biplot the grid matrix is first centered and normalized according to the prompted options.
Next, the matrix is decomposed by singular value decomposition (SVD) into

$$X = UDV^T$$

The biplot is made up of two matrices

$$X = GH^T$$

These matrices are construed on the basis of the SVD results.

$$\hat{X} = UD^gD^hV^T$$

Note that the grid matrix values are only recovered and the projection property is only given if $g + h = 1$

See Also

Unsophisticated biplot: biplotSimple;
2D biplots: biplot2d, biplotEsa2d, biplotSlater2d;
Pseudo 3D biplots: biplotPseudo3d, biplotEsaPseudo3d, biplotSlaterPseudo3d;
Interactive 3D biplots: biplot3d, biplotEsa3d, biplotSlater3d;
Function to set view in 3D: home.

Examples

```
## Not run:

   biplot2d(boeker)                # biplot of boeker data
   biplot2d(boeker, c.lines=T)     # add construct lines
   biplot2d(boeker, center=2)      # with column centering
```

```
  biplot2d(boeker, center=4)      # midpoint centering
  biplot2d(boeker, normalize=1)   # normalization of constructs

  biplot2d(boeker, dim=2:3)       # plot 2nd and 3rd dimension
  biplot2d(boeker, dim=c(1,4))    # plot 1st and 4th dimension

  biplot2d(boeker, g=1, h=1)            # assign singular values to con. & elem.
  biplot2d(boeker, g=1, h=1, center=1)  # row centering (Slater)
  biplot2d(boeker, g=1, h=1, center=4)  # midpoint centering (ESA)

  biplot2d(boeker, e.color="red", c.color="blue")   # change colors
  biplot2d(boeker, c.color=c("white", "darkred"))   # mapped onto color range

  biplot2d(boeker, unity=T)                 # scale con. & elem. to equal length
  biplot2d(boeker, unity=T, scale.e=.5)     # scaling factor for element vectors

  biplot2d(boeker, e.labels.show=F)         # do not show element labels
  biplot2d(boeker, e.labels.show=c(1,2,4))  # show labels for elements 1, 2 and 4
  biplot2d(boeker, e.points.show=c(1,2,4))  # only show elements 1, 2 and 4
  biplot2d(boeker, c.labels.show=c(1:4))    # show constructs labels 1 to 4
  biplot2d(boeker, c.labels.show=c(1:4))    # show constructs labels except 1 to 4

  biplot2d(boeker, e.cex.map=1)   # change size of texts for elements
  biplot2d(boeker, c.cex.map=1)   # change size of texts for constructs

  biplot2d(boeker, g=1, h=1, c.labels.inside=T)  # constructs inside the plot
  biplot2d(boeker, g=1, h=1, c.labels.inside=T,  # different margins and elem. color
           mai=c(0,0,0,0), e.color="red")

  biplot2d(boeker, strokes.x=.3, strokes.y=.05)  # change length of strokes

  biplot2d(boeker, flipaxes=c(T, F))      # flip x axis
  biplot2d(boeker, flipaxes=c(T, T))      # flip x and y axis

  biplot2d(boeker, outer.positioning=F)   # no positioning of con.-labels

  biplot2d(boeker, c.labels.devangle=20)  # only con. within 20 degree angle

## End(Not run)
```

`biplot3d` *Draw grid in rgl (3D device).*

Description

The 3D biplot opens an interactive 3D device that can be rotated and zoomed using the mouse. A 3D device facilitates the exploration of grid data as significant proportions of the sum-of-squares are often represented beyond the first two dimensions. Also, in a lot of cases it may be of interest to explore the grid space from a certain angle, e.g. to gain an optimal view onto the set of elements under investigation (e.g. Raeithel, 1998).

Usage

```
biplot3d(
  x,
  dim = 1:3,
  labels.e = TRUE,
  labels.c = TRUE,
  lines.c = TRUE,
  lef = 1.3,
  center = 1,
  normalize = 0,
  g = 0,
  h = 1,
  col.active = NA,
  col.passive = NA,
  c.sphere.col = grey(0.4),
  c.cex = 0.6,
  c.text.col = grey(0.4),
  e.sphere.col = grey(0),
  e.cex = 0.6,
  e.text.col = grey(0),
  alpha.sphere = 0.05,
  col.sphere = "black",
  unity = FALSE,
  unity3d = FALSE,
  scale.e = 0.9,
  zoom = 1,
  ...
)
```

Arguments

x	repgrid object.
dim	Dimensions to display.
labels.e	Logical. whether element labels are displayed.
labels.c	Logical. whether construct labels are displayed.
lines.c	Numeric. The way lines are drawn through the construct vectors. 0 = no lines, 1 = lines from constructs to outer frame, 2 = lines from the center to outer frame.
lef	Construct lines extension factor
center	Numeric. The type of centering to be performed. 0= no centering, 1= row mean centering (construct), 2= column mean centering (elements), 3= double-centering (construct and element means), 4= midpoint centering of rows (constructs). Default is 1 (row centering).
normalize	A numeric value indicating along what direction (rows, columns) to normalize by standard deviations. 0 = none, 1= rows, 2 = columns (default is 0).
g	Power of the singular value matrix assigned to the left singular vectors, i.e. the constructs.

h	Power of the singular value matrix assigned to the right singular vectors, i.e. the elements.
col.active	Columns (elements) that are no supplementary points, i.e. they are used in the SVD to find principal components. default is to use all elements.
col.passive	Columns (elements) that are supplementary points, i.e. they are NOT used in the SVD but projected into the component space afterwards. They do not determine the solution. Default is NA, i.e. no elements are set supplementary.
c.sphere.col	Color of construct spheres.
c.cex	Size of construct text.
c.text.col	Color for construct text.
e.sphere.col	Color of elements.
e.cex	Size of element labels.
e.text.col	Color of element labels.
alpha.sphere	Numeric. alpha blending of the surrounding sphere (default".05").
col.sphere	Color of surrounding sphere (default"black").
unity	Scale elements and constructs coordinates to unit scale (maximum of 1) so they are printed more neatly (default TRUE).
unity3d	To come.
scale.e	Scaling factor for element vectors. Will cause element points to move a bit more to the center (but only if unity or unity3d is TRUE). This argument is for visual appeal only.
zoom	Not yet used. Scaling factor for all vectors. Can be used to zoom the plot in and out (default 1).
...	Parameters to be passed on.

References

Raeithel, A. (1998). Kooperative Modellproduktion von Professionellen und Klienten - erlauetert am Beispiel des Repertory Grid. *Selbstorganisation, Kooperation, Zeichenprozess: Arbeiten zu einer kulturwissenschaftlichen, anwendungsbezogenen Psychologie* (pp. 209-254). Opladen: Westdeutscher Verlag.

See Also

Unsophisticated biplot: biplotSimple;
2D biplots: biplot2d, biplotEsa2d, biplotSlater2d;
Pseudo 3D biplots: biplotPseudo3d, biplotEsaPseudo3d, biplotSlaterPseudo3d;
Interactive 3D biplots: biplot3d, biplotEsa3d, biplotSlater3d;
Function to set view in 3D: home.

Examples

```
## Not run:

   biplot3d(boeker)
   biplot3d(boeker, unity3d=T)

   biplot3d(boeker, e.sphere.col="red",
            c.text.col="blue")
   biplot3d(boeker, e.cex=1)
   biplot3d(boeker, col.sphere="red")

   biplot3d(boeker, g=1, h=1)    # INGRID biplot
   biplot3d(boeker, g=1, h=1,    # ESA biplot
            center=4)

## End(Not run)
```

---

biplotEsa2d — *Plot an eigenstructure analysis (ESA) biplot in 2D.*

Description

The ESA is a special type of biplot suggested by Raeithel (e.g. 1998). It uses midpoint centering as a default. Note that the eigenstructure analysis is just a special case of a biplot that can also be produced using the `biplot2d` function with the arguments `center=4, g=1, h=1`. Here, only the arguments that are modified for the ESA biplot are described. To see all the parameters that can be changed see `biplot2d`.

Usage

```
biplotEsa2d(x, center = 4, g = 1, h = 1, ...)
```

Arguments

`x`	`repgrid` object.
`center`	Numeric. The type of centering to be performed. 0= no centering, 1= row mean centering (construct), 2= column mean centering (elements), 3= double-centering (construct and element means), 4= midpoint centering of rows (constructs). Eigenstructure analysis uses midpoint centering (4).
`g`	Power of the singular value matrix assigned to the left singular vectors, i.e. the constructs. Eigenstructure analysis uses g=1.
`h`	Power of the singular value matrix assigned to the right singular vectors, i.e. the elements. Eigenstructure analysis uses h=1.
`...`	Additional parameters for be passed to `biplot2d`.

References

Raeithel, A. (1998). Kooperative Modellproduktion von Professionellen und Klienten. Erlaeutert am Beispiel des Repertory Grid. In A. Raeithel (1998). Selbstorganisation, Kooperation, Zeichenprozess. Arbeiten zu einer kulturwissenschaftlichen, anwendungsbezogenen Psychologie (p. 209-254). Opladen: Westdeutscher Verlag.

See Also

Unsophisticated biplot: `biplotSimple`;
2D biplots: `biplot2d`, `biplotEsa2d`, `biplotSlater2d`;
Pseudo 3D biplots: `biplotPseudo3d`, `biplotEsaPseudo3d`, `biplotSlaterPseudo3d`;
Interactive 3D biplots: `biplot3d`, `biplotEsa3d`, `biplotSlater3d`;
Function to set view in 3D: `home`.

Examples

```
## Not run:
   # See examples in \code{\link{biplot2d}} as the same arguments
   # can used for this function.

## End(Not run)
```

`biplotEsa3d` *Draw the eigenstructure analysis (ESA) biplot in rgl (3D device).*

Description

The 3D biplot opens an interactive 3D device that can be rotated and zoomed using the mouse. A 3D device facilitates the exploration of grid data as significant proportions of the sum-of-squares are often represented beyond the first two dimensions. Also, in a lot of cases it may be of interest to explore the grid space from a certain angle, e.g. to gain an optimal view onto the set of elements under investigation (e.g. Raeithel, 1998). Note that the eigenstructure analysis just a special case of a biplot that can also be produced using the `biplot3d` function with the arguments `center=4, g=1, h=1`.

Usage

```
biplotEsa3d(x, center = 1, g = 1, h = 1, ...)
```

Arguments

`x`	`repgrid` object.
`center`	Numeric. The type of centering to be performed. 0= no centering, 1= row mean centering (construct), 2= column mean centering (elements), 3= double-centering (construct and element means), 4= midpoint centering of rows (constructs). Default is 4 (scale midpoint centering).

g	Power of the singular value matrix assigned to the left singular vectors, i.e. the constructs.
h	Power of the singular value matrix assigned to the right singular vectors, i.e. the elements.
...	Additional arguments to be passed to biplot3d.

See Also

Unsophisticated biplot: biplotSimple;
2D biplots: biplot2d, biplotEsa2d, biplotSlater2d;
Pseudo 3D biplots: biplotPseudo3d, biplotEsaPseudo3d, biplotSlaterPseudo3d;
Interactive 3D biplots: biplot3d, biplotEsa3d, biplotSlater3d;
Function to set view in 3D: home.

Examples

```
## Not run:

   biplotEsa3d(boeker)
   biplotEsa3d(boeker, unity3d=T)

   biplotEsa3d(boeker, e.sphere.col="red",
               c.text.col="blue")
   biplotEsa3d(boeker, e.cex=1)
   biplotEsa3d(boeker, col.sphere="red")


## End(Not run)
```

---

biplotEsaPseudo3d — *Plot an eigenstructure analysis (ESA) in 2D grid with 3D impression (pseudo 3D).*

Description

The ESA is a special type of biplot suggested by Raeithel (e.g. 1998). It uses midpoint centering as a default. Note that the eigenstructure analysis is just a special case of a biplot that can also be produced using the biplot2d function with the arguments center=4, g=1, h=1. Here, only the arguments that are modified for the ESA biplot are described. To see all the parameters that can be changed see biplot2d and biplotPseudo3d.

Usage

```
biplotEsaPseudo3d(x, center = 4, g = 1, h = 1, ...)
```

Arguments

x	repgrid object.
center	Numeric. The type of centering to be performed. 0= no centering, 1= row mean centering (construct), 2= column mean centering (elements), 3= double-centering (construct and element means), 4= midpoint centering of rows (constructs). Eigenstructure analysis uses midpoint centering (4).
g	Power of the singular value matrix assigned to the left singular vectors, i.e. the constructs. Eigenstructure analysis uses g=1.
h	Power of the singular value matrix assigned to the right singular vectors, i.e. the elements. Eigenstructure analysis uses h=1.
...	Additional parameters for be passed to biplotPseudo3d.

See Also

Unsophisticated biplot: biplotSimple;
2D biplots: biplot2d, biplotEsa2d, biplotSlater2d;
Pseudo 3D biplots: biplotPseudo3d, biplotEsaPseudo3d, biplotSlaterPseudo3d;
Interactive 3D biplots: biplot3d, biplotEsa3d, biplotSlater3d;
Function to set view in 3D: home.

Examples

```
## Not run:
   # See examples in \code{\link{biplotPseudo3d}} as the same arguments
   # can used for this function.

## End(Not run)
```

---

biplotPseudo3d *See biplotPseudo3d for its use. Draws a biplot of the grid in 2D with depth impression (pseudo 3D).*

Description

This version is basically a 2D biplot. It only modifies color and size of the symbols in order to create a 3D impression of the data points. This function will call the standard biplot2d function with some modified arguments. For the whole set of arguments that can be used see biplot2d. Here only the arguments special to biplotPseudo3d are outlined.

Usage

```
biplotPseudo3d(
  x,
  dim = 1:2,
  map.dim = 3,
```

```
    e.point.col = c("white", "black"),
    e.point.cex = c(0.6, 1.2),
    e.label.col = c("white", "black"),
    e.label.cex = c(0.6, 0.8),
    e.color.map = c(0.4, 1),
    c.point.col = c("white", "darkred"),
    c.point.cex = c(0.6, 1.2),
    c.label.col = c("white", "darkred"),
    c.label.cex = c(0.6, 0.8),
    c.color.map = c(0.4, 1),
    ...
)
```

Arguments

x	`repgrid` object.
dim	Dimensions (i.e. principal components) to be used for biplot (default is `c(1,2)`).
map.dim	Third dimension (depth) used to map aesthetic attributes to (default is `3`).
e.point.col	Color(s) of the element symbols. Two values can be entered that will create a color ramp. The values of `map.dim` are mapped onto the ramp. The default is `c("white", "black")`. If only one color color value is supplied (e.g. `"black"`) no mapping occurs and all elements will have the same color irrespective of their value on the `map.dim` dimension.
e.point.cex	Size of the element symbols. Two values can be entered that will represents the lower and upper size of a range of cex the values of `map.dim` are mapped onto. The default is `c(.6, 1.2)`. If only one cex value is supplied (e.g. `.7`) no mapping occurs and all elements will have the same size irrespective of their value on the `map.dim` dimension.
e.label.col	Color(s) of the element labels. Two values can be entered that will create a color ramp. The values of `map.dim` are mapped onto the ramp. The default is `c("white", "black")`. If only one color color value is supplied (e.g. `"black"`) no mapping occurs and all element labels will have the same color irrespective of their value on the `map.dim` dimension.
e.label.cex	Size of the element labels. Two values can be entered that will represents the lower and upper size of a range of cex the values of `map.dim` are mapped onto. The default is `c(.6, .8)`. If only one cex value is supplied (e.g. `.7`) no mapping occurs and all element labels will have the same size irrespective of their value on the `map.dim` dimension.
e.color.map	Value range to determine what range of the color ramp defined in `e.color` will be used for mapping the colors. Default is `c(.4, ,1)`. Usually not important for the user.
c.point.col	Color(s) of the construct symbols. Two values can be entered that will create a color ramp. The values of `map.dim` are mapped onto the ramp. The default is `c("white", "darkred")`. If only one color color value is supplied (e.g. `"black"`) no mapping occurs and all elements will have the same color irrespective of their value on the `map.dim` dimension.

c.point.cex
: Size of the construct symbols. Two values can be entered that will represents the lower and upper size of a range of cex the values of map.dim are mapped onto. The default is c(.6, 1.2). If only one cex value is supplied (e.g. .7) no mapping occurs and all elements will have the same size irrespective of their value on the map.dim dimension.

c.label.col
: Color(s) of the construct labels. Two values can be entered that will create a color ramp. The values of map.dim are mapped onto the ramp. The default is c("white", "black"). If only one color color value is supplied (e.g. "black") no mapping occurs and all construct labels will have the same color irrespective of their value on the map.dim dimension.

c.label.cex
: Size of the construct labels. Two values can be entered that will represents the lower and upper size of a range of cex the values of map.dim are mapped onto. The default is c(.6, .9). If only one cex value is supplied (e.g. .7) no mapping occurs and all construct labels will have the same size irrespective of their value on the map.dim dimension.

c.color.map
: Value range to determine what range of the color ramp defined in c.color will be used for mapping. Default is c(.4, ,1). Usually not important for the user.

...
: Additional parameters passed to biplot2d.

See Also

Unsophisticated biplot: biplotSimple;
2D biplots: biplot2d, biplotEsa2d, biplotSlater2d;
Pseudo 3D biplots: biplotPseudo3d, biplotEsaPseudo3d, biplotSlaterPseudo3d;
Interactive 3D biplots: biplot3d, biplotEsa3d, biplotSlater3d;
Function to set view in 3D: home.

Examples

```
## Not run:
   # biplot with 3D impression
   biplotPseudo3d(boeker)
   # Slater's biplot with 3D impression
   biplotPseudo3d(boeker, g=1, h=1, center=1)

   # show 2nd and 3rd dim. and map 4th
   biplotPseudo3d(boeker, dim=2:3, map.dim=4)

   # change elem. colors
   biplotPseudo3d(boeker, e.color=c("white", "darkgreen"))
   # change con. colors
   biplotPseudo3d(boeker, c.color=c("white", "darkgreen"))
   # change color mapping range
   biplotPseudo3d(boeker, c.colors.map=c(0, 1))

   # set uniform con. text size
   biplotPseudo3d(boeker, c.cex=1)
   # change text size mapping range
   biplotPseudo3d(boeker, c.cex=c(.4, 1.2))
```

```
## End(Not run)
```

`biplotSimple` *A graphically unsophisticated version of a biplot.*

Description

It will draw elements and constructs vectors using similar arguments as biplot2d. It is a version for quick exploration used during development.

Usage

```
biplotSimple(
  x,
  dim = 1:2,
  center = 1,
  normalize = 0,
  g = 0,
  h = 1 - g,
  unity = T,
  col.active = NA,
  col.passive = NA,
  scale.e = 0.9,
  zoom = 1,
  e.point.col = "black",
  e.point.cex = 1,
  e.label.col = "black",
  e.label.cex = 0.7,
  c.point.col = grey(0.6),
  c.label.col = grey(0.6),
  c.label.cex = 0.6,
  ...
)
```

Arguments

`x`	`repgrid` object.
`dim`	Dimensions (i.e. principal components) to be used for biplot (default is `c(1,2)`).
`center`	Numeric. The type of centering to be performed. 0= no centering, 1= row mean centering (construct), 2= column mean centering (elements), 3= double-centering (construct and element means), 4= midpoint centering of rows (constructs). The default is `1` (row centering).
`normalize`	A numeric value indicating along what direction (rows, columns) to normalize by standard deviations. `0 = none, 1= rows, 2 = columns` (default is `0`).

g	Power of the singular value matrix assigned to the left singular vectors, i.e. the constructs.
h	Power of the singular value matrix assigned to the right singular vectors, i.e. the elements.
unity	Scale elements and constructs coordinates to unit scale in 2D (maximum of 1) so they are printed more neatly (default TRUE).
col.active	Columns (elements) that are no supplementary points, i.e. they are used in the SVD to find principal components. default is to use all elements.
col.passive	Columns (elements) that are supplementary points, i.e. they are NOT used in the SVD but projected into the component space afterwards. They do not determine the solution. Default is NA, i.e. no elements are set supplementary.
scale.e	Scaling factor for element vectors. Will cause element points to move a bit more to the center. This argument is for visual appeal only.
zoom	Scaling factor for all vectors. Can be used to zoom the plot in and out (default 1).
e.point.col	Color of the element symbols (default is "black".
e.point.cex	Size of the element symbol (default is 1.
e.label.col	Color of the element labels (default is "black".
e.label.cex	Size of the element labels (default is .7.
c.point.col	Color of the construct lines (default is grey(.6).
c.label.col	Color of the construct labels (default is grey(.6).
c.label.cex	Size of the construct labels (default is .6.
...	Parameters to be passed on to center() and normalize.

Value

repgrid object.

See Also

Unsophisticated biplot: biplotSimple;
2D biplots: biplot2d, biplotEsa2d, biplotSlater2d;
Pseudo 3D biplots: biplotPseudo3d, biplotEsaPseudo3d, biplotSlaterPseudo3d;
Interactive 3D biplots: biplot3d, biplotEsa3d, biplotSlater3d;
Function to set view in 3D: home.

Examples

```
## Not run:

   biplotSimple(boeker)
   biplotSimple(boeker, unity=F)

   biplotSimple(boeker, g=1, h=1)             # INGRID biplot
   biplotSimple(boeker, g=1, h=1, center=4)    # ESA biplot
```

```
    biplotSimple(boeker, zoom=.9)             # zooming out
    biplotSimple(boeker, scale.e=.6)          # scale element vectors

    biplotSimple(boeker, e.point.col="brown")   # change colors
    biplotSimple(boeker, e.point.col="brown",
                 c.label.col="darkblue")

## End(Not run)
```

---

biplotSlater2d *Draws Slater's INGRID biplot in 2D.*

Description

The default is to use row centering and no normalization. Note that Slater's biplot is just a special case of a biplot that can be produced using the `biplot2d` function with the arguments `center=1, g=1, h=1`. The arguments that can be used in this function are the same as in `biplot2d`. Here, only the arguments that are set for Slater's biplot are described. To see all the parameters that can be changed see `biplot2d`.

Usage

```
biplotSlater2d(x, center = 1, g = 1, h = 1, ...)
```

Arguments

`x`	`repgrid` object.
`center`	Numeric. The type of centering to be performed. 0= no centering, 1= row mean centering (construct), 2= column mean centering (elements), 3= double-centering (construct and element means), 4= midpoint centering of rows (constructs). Slater's biplot uses 1 (row centering).
`g`	Power of the singular value matrix assigned to the left singular vectors, i.e. the constructs.
`h`	Power of the singular value matrix assigned to the right singular vectors, i.e. the elements.
`...`	Additional parameters for be passed to `biplot2d`.

See Also

Unsophisticated biplot: `biplotSimple`;
2D biplots: `biplot2d`, `biplotEsa2d`, `biplotSlater2d`;
Pseudo 3D biplots: `biplotPseudo3d`, `biplotEsaPseudo3d`, `biplotSlaterPseudo3d`;
Interactive 3D biplots: `biplot3d`, `biplotEsa3d`, `biplotSlater3d`;
Function to set view in 3D: `home`.

Examples

```
## Not run:
   # See examples in \code{\link{biplot2d}} as the same arguments
   # can used for this function.

## End(Not run)
```

---

biplotSlater3d *Draw the Slater's INGRID biplot in rgl (3D device).*

Description

The 3D biplot opens an interactive 3D device that can be rotated and zoomed using the mouse. A 3D device facilitates the exploration of grid data as significant proportions of the sum-of-squares are often represented beyond the first two dimensions. Also, in a lot of cases it may be of interest to explore the grid space from a certain angle, e.g. to gain an optimal view onto the set of elements under investigation (e.g. Raeithel, 1998). Note that Slater's biplot is just a special case of a biplot that can be produced using the `biplot3d` function with the arguments `center=1, g=1, h=1`.

Usage

```
biplotSlater3d(x, center = 1, g = 1, h = 1, ...)
```

Arguments

`x`	`repgrid` object.
`center`	Numeric. The type of centering to be performed. 0= no centering, 1= row mean centering (construct), 2= column mean centering (elements), 3= double-centering (construct and element means), 4= midpoint centering of rows (constructs). Default is 1 (row i.e. construct centering).
`g`	Power of the singular value matrix assigned to the left singular vectors, i.e. the constructs.
`h`	Power of the singular value matrix assigned to the right singular vectors, i.e. the elements.
`...`	Additional arguments to be passed to biplot3d.

See Also

Unsophisticated biplot: `biplotSimple`;
2D biplots: `biplot2d`, `biplotEsa2d`, `biplotSlater2d`;
Pseudo 3D biplots: `biplotPseudo3d`, `biplotEsaPseudo3d`, `biplotSlaterPseudo3d`;
Interactive 3D biplots: `biplot3d`, `biplotEsa3d`, `biplotSlater3d`;
Function to set view in 3D: `home`.

Examples

```
## Not run:

   biplotSlater3d(boeker)
   biplotSlater3d(boeker, unity3d=T)

   biplotSlater3d(boeker, e.sphere.col="red",
                  c.text.col="blue")
   biplotSlater3d(boeker, e.cex=1)
   biplotSlater3d(boeker, col.sphere="red")


## End(Not run)
```

`biplotSlaterPseudo3d` *Draws Slater's biplot in 2D with depth impression (pseudo 3D).*

Description

The default is to use row centering and no normalization. Note that Slater's biplot is just a special case of a biplot that can be produced using the `biplotPseudo3d` function with the arguments `center=1, g=1, h=1`. Here, only the arguments that are modified for Slater's biplot are described. To see all the parameters that can be changed see `biplot2d` and `biplotPseudo3d`.

Usage

```
biplotSlaterPseudo3d(x, center = 1, g = 1, h = 1, ...)
```

Arguments

`x`	`repgrid` object.
`center`	Numeric. The type of centering to be performed. 0= no centering, 1= row mean centering (construct), 2= column mean centering (elements), 3= double-centering (construct and element means), 4= midpoint centering of rows (constructs). Slater's biplot uses `1` (row centering).
`g`	Power of the singular value matrix assigned to the left singular vectors, i.e. the constructs.
`h`	Power of the singular value matrix assigned to the right singular vectors, i.e. the elements.
`...`	Additional parameters for be passed to `biplotPseudo3d`.

See Also

Unsophisticated biplot: `biplotSimple`;
2D biplots: `biplot2d`, `biplotEsa2d`, `biplotSlater2d`;
Pseudo 3D biplots: `biplotPseudo3d`, `biplotEsaPseudo3d`, `biplotSlaterPseudo3d`;
Interactive 3D biplots: `biplot3d`, `biplotEsa3d`, `biplotSlater3d`;
Function to set view in 3D: `home`.

Examples

```
## Not run:
   # See examples in \code{\link{biplotPseudo3d}} as the same arguments
   # can used for this function.

## End(Not run)
```

---

center — *Centering of rows (constructs) and/or columns (elements).*

Description

Centering of rows (constructs) and/or columns (elements).

Usage

```
center(x, center = 1, ...)
```

Arguments

`x`	`repgrid` object.
`center`	Numeric. The type of centering to be performed. 0= no centering, 1= row mean centering (construct), 2= column mean centering (elements), 3= double-centering (construct and element means), 4= midpoint centering of rows (constructs). of the scale(default `FALSE`). Default is `1` (row centering).
`...`	Not evaluated.

Value

`matrix` containing the transformed values.

Note

If scale midpoint centering is applied no row or column centering can be applied simultaneously. TODO: After centering the standard representation mode does not work any more as it remains unclear what color values to attach to the centered values.

Examples

```
## Not run:

 center(bell2010)       # no centering
 center(bell2010, rows=T)   # row centering of grid
 center(bell2010, cols=T)   # column centering of grid
 center(bell2010, rows=T, cols=T) # row and column centering

## End(Not run)
```

cluster *Cluster analysis (of constructs or elements).*

Description

`cluster` is a preliminary implementation of a cluster function. It supports various distance measures as well as cluster methods. More is to come.

align: Aligning will reverse constructs if necessary to yield a maximal similarity between constructs. In a first step the constructs are clustered including both directions. In a second step the direction of a construct that yields smaller distances to the adjacent constructs is preserved and used for the final clustering. As a result, every construct is included once but with an orientation that guarantees optimal clustering. This approach is akin to the procedure used in FOCUS (Jankowicz & Thomas, 1982).

Usage

```
cluster(
  x,
  along = 0,
  dmethod = "euclidean",
  cmethod = "ward",
  p = 2,
  align = TRUE,
  trim = NA,
  main = NULL,
  mar = c(4, 2, 3, 15),
  cex = 0,
  lab.cex = 0.8,
  cex.main = 0.9,
  print = TRUE,
  ...
)
```

Arguments

`x`	`repgrid` object.
`along`	Along which dimension to cluster. 1 = constructs only, 2= elements only, 0=both (default).
`dmethod`	The distance measure to be used. This must be one of "euclidean", "maximum", "manhattan", "canberra", "binary" or "minkowski". Any unambiguous substring can be given. For additional information on the different types type `?dist`.
`cmethod`	The agglomeration method to be used. This should be (an unambiguous abbreviation of) one of `"ward"`, `"single"`, `"complete"`, `"average"`, `"mcquitty"`, `"median"` or `"centroid"`.
`p`	The power of the Minkowski distance, in case `"minkowski"` is used as argument for `dmethod`.

`align`	Whether the constructs should be aligned before clustering (default is `TRUE`). If not, the grid matrix is clustered as is. See Details section for more information.
`trim`	the number of characters a construct is trimmed to (default is `10`). If `NA` no trimming is done. Trimming simply saves space when displaying the output.
`main`	Title of plot. The default is a name indicating the distance function and cluster method.
`mar`	Define the plot region (bottom, left, upper, right).
`cex`	Size parameter for the nodes. Usually not needed.
`lab.cex`	Size parameter for the constructs on the right side.
`cex.main`	Size parameter for the plot title (default is `.9`).
`print`	Logical. Whether to print the dendrogram (default is `TRUE`).
`...`	Additional parameters to be passed to plotting function from `as.dendrogram`. Type `?as.dendrogram` for further information. This option is usually not needed, except if special designs are needed.

Value

Reordered `repgrid` object.

References

Jankowicz, D., & Thomas, L. (1982). An Algorithm for the Cluster Analysis of Repertory Grids in Human Resource Development. *Personnel Review, 11*(4), 15-22. doi:10.1108/eb055464.

See Also

`bertinCluster`

Examples

```
## Not run:

  cluster(bell2010)
  cluster(bell2010, main="My cluster analysis")   # new title
  cluster(bell2010, type="t")                     # different drawing style
  cluster(bell2010, dmethod="manhattan")          # using manhattan metric
  cluster(bell2010, cmethod="single")             # do single linkage clustering
  cluster(bell2010, cex=1, lab.cex=1)             # change appearance
  cluster(bell2010, lab.cex=.7,                   # advanced appearance changes
          edgePar = list(lty=1:2, col=2:1))

## End(Not run)
```

clusterBoot *Multiscale bootstrap cluster analysis.*

Description

p-values are calculated for each branch of the cluster dendrogram to indicate the stability of a specific partition. `clusterBoot` will yield the same clusters as the `cluster` function (i.e. standard hierarchical clustering) with additional p-values. Two kinds of p-values are reported: bootstrap probabilities (BP) and approximately unbiased (AU) probabilities (see Details section for more information).

Usage

```
clusterBoot(
  x,
  along = 1,
  align = TRUE,
  dmethod = "euclidean",
  cmethod = "ward",
  p = 2,
  nboot = 1000,
  r = seq(0.8, 1.4, by = 0.1),
  seed = NULL,
  ...
)
```

Arguments

- `x` — `grid` object
- `along` — Along which dimension to cluster. 1 = constructs, 2= elements.
- `align` — Whether the constructs should be aligned before clustering (default is `TRUE`). If not, the grid matrix is clustered as is. See Details section for more information.
- `dmethod` — The distance measure to be used. This must be one of "euclidean", "maximum", "manhattan", "canberra", "binary" or "minkowski". Any unambiguous substring can be given. For additional information on the different types type `?dist`.
- `cmethod` — The agglomeration method to be used. This should be (an unambiguous abbreviation of) one of `"ward"`, `"single"`, `"complete"`, `"average"`, `"mcquitty"`, `"median"` or `"centroid"`.
- `p` — Power of the Minkowski metric. Not yet passed on to pvclust!
- `nboot` — the number of bootstrap replications. The default is `1000`.
- `r` — numeric vector which specifies the relative sample sizes of bootstrap replications. For original sample size n and bootstrap sample size n', this is defined as $r = n'/n$.
- `seed` — Random seed for bootstrapping. Can be set for reproducibility (see `set.seed`). Usually not needed.
- `...` — Arguments to pass on to `pvclust`.

Details

In standard (hierarchical) cluster analysis the question arises which of the identified structures are significant or just emerged by chance. Over the last decade several methods have been developed to test structures for robustness. One line of research in this area is based on resampling. The idea is to resample the rows or columns of the data matrix and to build the dendrogram for each bootstrap sample (Felsenstein, 1985). The p-values indicates the percentage of times a specific structure is identified across the bootstrap samples. It was shown that the p-value is biased (Hillis & Bull, 1993; Zharkikh & Li, 1995). In the literature several methods for bias correction have been proposed. In `clusterBoot` a method based on the *multiscale bootstrap* is used to derive corrected (approximately unbiased) p-values (Shimodaira, 2002, 2004). In conventional bootstrap analysis the size of the bootstrap sample is identical to the original sample size. Multiscale bootstrap varies the bootstrap sample size in order to infer a correction formula for the biased p-value on the basis of the variation of the results for the different sample sizes (Suzuki & Shimodaira, 2006).

align: Aligning will reverse constructs if necessary to yield a maximal similarity between constructs. In a first step the constructs are clustered including both directions. In a second step the direction of a construct that yields smaller distances to the adjacent constructs is preserved and used for the final clustering. As a result, every construct is included once but with an orientation that guarantees optimal clustering. This approach is akin to the procedure used in FOCUS (Jankowicz & Thomas, 1982).

Value

A pvclust object as returned by the function `pvclust`

References

Felsenstein, J. (1985). Confidence Limits on Phylogenies: An Approach Using the Bootstrap. *Evolution, 39*(4), 783. doi:10.2307/2408678

Hillis, D. M., & Bull, J. J. (1993). An Empirical Test of Bootstrapping as a Method for Assessing Confidence in Phylogenetic Analysis. *Systematic Biology, 42*(2), 182-192.

Jankowicz, D., & Thomas, L. (1982). An Algorithm for the Cluster Analysis of Repertory Grids in Human Resource Development. *Personnel Review, 11*(4), 15-22. doi:10.1108/eb055464.

Shimodaira, H. (2002) An approximately unbiased test of phylogenetic tree selection. *Syst, Biol., 51*, 492-508.

Shimodaira,H. (2004) Approximately unbiased tests of regions using multistep- multiscale bootstrap resampling. *Ann. Stat., 32*, 2616-2614.

Suzuki, R., & Shimodaira, H. (2006). Pvclust: an R package for assessing the uncertainty in hierarchical clustering. *Bioinformatics, 22*(12), 1540-1542. doi:10.1093/bioinformatics/btl117

Zharkikh, A., & Li, W.-H. (1995). Estimation of confidence in phylogeny: the complete-and-partial bootstrap technique. *Molecular Phylogenetic Evolution, 4*(1), 44-63.

Examples

```
## Not run:

 # pvclust must be loaded
 library(pvclust)
```

```
  # p-values for construct dendrogram
  s <- clusterBoot(boeker)
  plot(s)
  pvrect(s, max.only=FALSE)

  # p-values for element dendrogram
  s <- clusterBoot(boeker, along=2)
  plot(s)
  pvrect(s, max.only=FALSE)

## End(Not run)
```

`constructCor` *Calculate correlations between constructs.*

Description

Different types of correlations can be requested: PMC, Kendall tau rank correlation, Spearman rank correlation.

Usage

```
constructCor(
  x,
  method = c("pearson", "kendall", "spearman"),
  trim = 20,
  index = FALSE
)
```

Arguments

`x`	`repgrid` object.
`method`	A character string indicating which correlation coefficient is to be computed. One of `"pearson"` (default), `"kendall"` or `"spearman"`, can be abbreviated. The default is `"pearson"`.
`trim`	The number of characters a construct is trimmed to (default is `20`). If `NA` no trimming occurs. Trimming simply saves space when displaying correlation of constructs with long names.
`index`	Whether to print the number of the construct.

Value

Returns a matrix of construct correlations.

See Also

`elementCor`

Examples

```
# three different types of correlations
constructCor(mackay1992)
constructCor(mackay1992, method="kendall")
constructCor(mackay1992, method="spearman")

# format output
constructCor(mackay1992, trim=6)
constructCor(mackay1992, index=TRUE, trim=6)

# save correlation matrix for further processing
r <- constructCor(mackay1992)
r
print(r, digits=5)

# accessing the correlation matrix
r[1, 3]
```

---

constructD — *Calculate Somers' d for the constructs.*

Description

Somer's d is an asymmetric association measure as it depends on which variable is set as dependent and independent. The direction of dependency needs to be specified.

Usage

```
constructD(x, dependent = "columns", trim = 30, index = TRUE)
```

Arguments

`x`	`repgrid` object
`dependent`	A string denoting the direction of dependency in the output table (as d is asymmetrical). Possible values are `"columns"` (the default) for setting the columns as dependent, `"rows"` for setting the rows as the dependent variable and `"symmetric"` for the symmetrical Somers' d measure (the mean of the two directional values for code"columns" and `"rows"`).
`trim`	The number of characters a construct is trimmed to (default is `30`). If `NA` no trimming occurs. Trimming simply saves space when displaying correlation of constructs with long names.
`index`	Whether to print the number of the construct (default is `TRUE`).

Value

`matrix` of construct correlations.

Note

Thanks to Marc Schwartz for supplying the code to calculate Somers' d.

References

Somers, R. H. (1962). A New Asymmetric Measure of Association for Ordinal Variables. *American Sociological Review, 27*(6), 799-811.

Examples

```
## Not run:

   constructD(fbb2003)       # columns as dependent (default)
   constructD(fbb2003, "c")  # row as dependent
   constructD(fbb2003, "s")  # symmetrical index

   # suppress printing
   d <- constructD(fbb2003, out=0, trim=5)
   d

   # more digits
   constructD(fbb2003, dig=3)

   # add index column, no trimming
   constructD(fbb2003, col.index=TRUE, index=F, trim=NA)


## End(Not run)
```

`constructPca` *Principal component analysis (PCA) of inter-construct correlations.*

Description

Various methods for rotation and methods for the calculation of the correlations are available. Note that the number of factors has to be specified. For more information on the PCA function itself type `?principal`.

Usage

```
constructPca(
  x,
  nfactors = 3,
  rotate = "varimax",
  method = "pearson",
  trim = NA
)
```

Arguments

- `x` — `repgrid` object.
- `nfactors` — Number of components to extract (default is 3).
- `rotate` — `"none"`, `"varimax"`, `"promax"` and `"cluster"` are possible rotations (default is `none`).
- `method` — A character string indicating which correlation coefficient is to be computed. One of `"pearson"` (default), `"kendall"` or `"spearman"`, can be abbreviated. The default is `"pearson"`.
- `trim` — The number of characters a construct is trimmed to (default is 7). If `NA` no trimming occurs. Trimming simply saves space when displaying correlation of constructs with long names.

Value

Returns an object of class `constructPca`.

References

Fransella, F., Bell, R. & Bannister, D. (2003). *A Manual for Repertory Grid Technique* (2. Ed.). Chichester: John Wiley & Sons.

See Also

To extract the PCA loadings for further processing see `constructPcaLoadings`.

Examples

```
## Not run:

   constructPca(bell2010)

   # data from grid manual by Fransella et al. (2003, p. 87)
   # note that the construct order is different
   constructPca(fbb2003, nfactors=2)

   # no rotation
   constructPca(fbb2003, rotate="none")

   # use a different type of correlation (Spearman)
   constructPca(fbb2003, method="spearman")

   # save output to object
   m <- constructPca(fbb2003, nfactors=2)
   m

   # different printing options
   print(m, digits=5)
   print(m, cutoff=.3)
```

```
## End(Not run)
```

---

constructPcaLoadings   *Extract loadings from PCA of constructs.*

Description

Extract loadings from PCA of constructs.

Usage

```
constructPcaLoadings(x)
```

Arguments

x	`repgrid` object. This object is returned by the function `constructPca`.

Value

A matrix containing the factor loadings.

Examples

```
p <- constructPca(bell2010)
l <- constructPcaLoadings(p)
l[1, ]
l[, 1]
l[1,1]
```

---

constructRmsCor   *Root mean square (RMS) of inter-construct correlations.*

Description

The RMS is also known as 'quadratic mean' of the inter-construct correlations. The RMS serves as a simplification of the correlation table. It reflects the average relation of one construct to all other constructs. Note that as the correlations are squared during its calculation, the RMS is not affected by the sign of the correlation (cf. Fransella, Bell & Bannister, 2003, p. 86).

Usage

```
constructRmsCor(x, method = "pearson", trim = NA)
```

Arguments

`x`	`repgrid` object
`method`	A character string indicating which correlation coefficient is to be computed. One of `"pearson"` (default), `"kendall"` or `"spearman"`, can be abbreviated. The default is `"pearson"`.
`trim`	The number of characters a construct is trimmed to (default is `NA`). If `NA` no trimming occurs. Trimming simply saves space when displaying correlation of constructs with long names.

Value

`dataframe` of the RMS of inter-construct correlations

References

Fransella, F., Bell, R. C., & Bannister, D. (2003). A Manual for Repertory Grid Technique (2. Ed.). Chichester: John Wiley & Sons.

See Also

`elementRmsCor`, `constructCor`

Examples

```
# data from grid manual by Fransella, Bell and Bannister
constructRmsCor(fbb2003)
constructRmsCor(fbb2003, trim=20)

# modify output
r <- constructRmsCor(fbb2003)
print(r, digits=5)
# access calculation results
r[2, 1]
```

`constructs` *Get or replace construct poles*

Description

Allows to get and set construct poles. Replaces the older functions `getConstructNames`, `getConstructNames2`, and `eNames` which are deprecated.

Usage

```
constructs(x, collapse = FALSE, sep = " - ")

constructs(x, i, j) <- value

leftpoles(x)

leftpoles(x, position) <- value

rightpoles(x)

rightpoles(x, position) <- value
```

Arguments

`x`	A repgrid object.
`collapse`	Return vector with both poles instead.
`sep`	Separator if `collapse = TRUE`, default is `" - "`.
`i, j`	Row and column Index of repgrid matrix.
`value`	Character vector of construct poles names.
`position`	Index where to insert construct

Examples

```
# shorten object name
x <- boeker

## get construct poles
constructs(x)   # both left and right poles
leftpoles(x)    # left poles only
rightpoles(x)
constructs(x, collapse = TRUE)

## replace construct poles
constructs(x)[1,1] <- "left pole 1"
constructs(x)[1,"leftpole"] <- "left pole 1"  # alternative
constructs(x)[1:3,2] <- paste("right pole", 1:3)
constructs(x)[1:3,"rightpole"] <- paste("right pole", 1:3) # alternative
constructs(x)[4,1:2] <- c("left pole 4", "right pole 4")

l <- leftpoles(x)
leftpoles(x) <- sample(l)             # brind poles into random order
leftpoles(x)[1] <- "new left pole 1"  # replace name of first left pole

# replace left poles of constructs 1 and 3
leftpoles(x)[c(1,3)] <- c("new left pole 1", "new left pole 3")
```

`data-bell2010` *Grid data from Bell (2010).*

Description

Grid data originated (but is not shown in the paper) from a study by Haritos, Gindinis, Doan and Bell (2004) on element role titles. It was used to demonstrate the effects of construct alignment in Bell (2010, p. 46).

References

Bell, R. C. (2010). A note on aligning constructs. *Personal Construct Theory and Practice*, 7, 43-48.

Haritos, A., Gindidis, A., Doan, C., & Bell, R. C. (2004). The effect of element role titles on construct structure and content. *Journal of constructivist psychology, 17*(3), 221-236.

`data-bellmcgorry1992` *Grid data from Bell and McGorry (1992).*

Description

The grid data set is used in Bell's technical report "Using SPSS to Analyse Repertory Grid Data" (1997, p. 6). Originally, the data comes from a study by Bell and McGorry (1992).

References

Bell, R. C. (1977). *Using SPSS to Analyse Repertory Grid Data*. Technical Report, University of Melbourne.

Bell, R. C., & McGorry, P. (1992). The analysis of repertory grids used to monitor the perceptions of recovering psychotic patients. In A. Thomson & P. Cummins (Eds.), *European Perspectives in Personal Construct Psychology* (p. 137-150). Lincoln, UK: European Personal Construct Association.

`data-boeker` *Grid data from Boeker (1996).*

Description

Grid data from a schizophrenic patient undergoing psychoanalytically oriented psychotherapy. The data was taken during the last stage of therapy (Boeker, 1996, p. 163).

References

Boeker, H. (1996). The reconstruction of the self in the psychotherapy of chronic schizophrenia: a case study with the Repertory Grid Technique. In: Scheer, J. W., Catina, A. (Eds.): *Empirical Constructivism in Europe - The Personal Construct Approach* (p. 160-167). Giessen: Psychosozial-Verlag.

`data-fbb2003` *Grid data from Fransella, Bell and Bannister (2003).*

Description

A dataset used throughout the book "A Manual for Repertory Grid Technique" (Fransella, Bell and Bannister, 2003, p. 60).

References

Fransella, F., Bell, R. & Bannister, D. (2003). A Manual for Repertory Grid Technique (2. Ed.). Chichester: John Wiley & Sons.

`data-feixas2004` *Grid data from Feixas and Saul (2004).*

Description

A description by the authors: "When Teresa, 22 years old, was seen by the second author (LAS) at the psychological services of the University of Salamanca, she was in the final year of her studies in chemical sciences. Although Teresa proves to be an excellent student, she reveals serious doubts about her self worth. She cries frequently, and has great difficulty in meeting others, even though she has a boyfriend who is extremely supportive. Teresa is anxiously hesitant about accepting a new job which would involve moving to another city 600 Km away from home." (Feixas & Saul, 2004, p. 77).

References

Feixas, G., & Saul, L. A. (2004). The Multi-Center Dilemma Project: an investigation on the role of cognitive conflicts in health. *The Spanish Journal of Psychology, 7*(1), 69-78.

`data-leach2001` *Pre- and post therapy dataset from Leach et al. (2001).*

Description

Case as described by the authors: "Sarah, aged 32, was referred with problems of depression and sexual difficulties relating to childhood sexual abuse. She had three children and was living with her male partner. From the age of 9, her brother, an adult, had sexually abused Sarah. She attended a group for survivors of child sexual abuse and completed repertory grids prior to the group, immediately after the group and at 3- and 6-month follow-up." (Leach et al. 2001, p. 230).

Details

`leach2001a` is the pre-therapy, `leach2001b` is the post-therapy therapy dataset. The construct and elements are identical.

References

Leach, C., Freshwater, K., Aldridge, J., & Sunderland, J. (2001). Analysis of repertory grids in clinical practice. *The British Journal of Clinical Psychology, 40*, 225-248.

`data-mackay1992` *Grid data from Mackay (1992). Data set 'Grid C'-*

Description

used in Mackay's paper on inter-element correlation (1992, p. 65).

References

Mackay, N. (1992). Identification, reflection, and correlation: Problems in the bases of repertory grid measures. *International Journal of Personal Construct Psychology, 5*(1), 57-75.

data-raeithel — *Grid data from Raeithel (1998).*

Description

Grid data to demonstrate the use of Bertin diagrams (Raeithel, 1998, p. 223). The context of its administration is unknown.

References

Raeithel, A. (1998). Kooperative Modellproduktion von Professionellen und Klienten. Erlaeutert am Beispiel des Repertory Grid. In A. Raeithel (1998). Selbstorganisation, Kooperation, Zeichenprozess. Arbeiten zu einer kulturwissenschaftlichen, anwendungsbezogenen Psychologie (p. 209-254). Opladen: Westdeutscher Verlag.

data-slater1977a — *Drug addict's grid data set from Slater (1977, p. 32).*

Description

Drug addict's grid data set from Slater (1977, p. 32).

References

Slater, P. (1977). *The measurement of intrapersonal space by grid technique*. London: Wiley.

data-slater1977b — *Grid data from Slater (1977).*

Description

Grid data (ranked) from a seventeen year old female psychiatric patient (Slater, 1977, p. 110). She was depressed, anxious and took to cutting herself. The data was originally reported by Watson (1970).

References

Slater, P. (1977). *The measurement of intrapersonal space by grid technique*. London: Wiley.

Watson, J. P. (1970). The relationship between a self-mutilating patient and her doctor. *Psychotherapy and Psychosomatics, 18*(1), 67-73.

`distance` *Distance measures (between constructs or elements).*

Description

Various distance measures between elements or constructs are calculated.

Usage

```
distance(
  x,
  along = 1,
  dmethod = "euclidean",
  p = 2,
  normalize = FALSE,
  trim = 20,
  index = TRUE,
  ...
)
```

Arguments

x	`repgrid` object.
along	Whether to calculate distance for 1 = constructs (default) or for 2= elements.
dmethod	The distance measure to be used. This must be one of "euclidean", "maximum", "manhattan", "canberra", "binary" or "minkowski". Any unambiguous substring can be given. For additional information on the different types type `?dist`.
p	The power of the Minkowski distance, in case `"minkowski"` is used as argument for `dmethod`.
normalize	Use normalized distances. The distances are divided by the highest possible value given the rating scale fo the grid, so all distances are in the interval [0,1].
trim	The number of characters a construct or element is trimmed to (default is `20`). If `NA` no trimming occurs. Trimming simply saves space when displaying correlation of constructs with long names.
index	Whether to print the number of the construct or element in front of the name (default is `TRUE`). This is useful to avoid identical row names, which may cause an error.
...	Additional parameters to be passed to function `dist`. Type `dist` for further information.

Value

`matrix` object.

Examples

```
## Not run:

   # between constructs
   distance(bell2010, along = 1)
   distance(bell2010, along = 1, normalize = TRUE)

   # between elements
   distance(bell2010, along = 2)

   # several distance methods
   distance(bell2010, dm = "man")           # manhattan distance
   distance(bell2010, dm = "mink", p = 3)   # minkowski metric to the power of 3

   # to save the results without printing to the console
   d <- distance(bell2010, trim = 7)
   d

   # some more options when printing the distance matrix
   print(d, digits = 5)
   print(d, col.index = FALSE)
   print(d, upper = FALSE)

   # accessing entries from the matrix
   d[1,3]


## End(Not run)
```

---

distanceHartmann *'Hartmann distance' (standardized Slater distances).*

Description

Calculate Hartmann distance

Usage

```
distanceHartmann(
  x,
  method = "paper",
  reps = 10000,
  prob = NULL,
  progress = TRUE,
  distributions = FALSE
)
```

Arguments

`x`	`repgrid` object.
`method`	The method used for distance calculation, on of `"paper"`, `"simulate"`, `"new"`. `"paper"` uses the parameters as given in Hartmann (1992) for calculation. `"simulate"` (default) simulates a Slater distribution for the calculation. In a future version the time consuming simulation will be replaced by more accurate parameters for Hartmann distances than used in Hartmann (1992).
`reps`	Number of random grids to generate sample distribution for Slater distances (default is `10000`). Note that a lot of samples may take a while to calculate.
`prob`	The probability of each rating value to occur. If `NULL` (default) the distribution is uniform. The number of values must match the length of the rating scale.
`progress`	Whether to show a progress bar during simulation (default is `TRUE`) (for `method="simulate"`). May be useful when the distribution is estimated on the basis of many quasis.
`distributions`	Whether to additionally return the values of the simulated distributions (Slater etc.) The default is `FALSE` as it will quickly boost the object size.

Details

Hartmann (1992) showed in a simulation study that Slater distances (see `distanceSlater`) based on random grids, for which Slater coined the expression quasis, have a skewed distribution, a mean and a standard deviation depending on the number of constructs elicited. He suggested a linear transformation (z-transformation) which takes into account the estimated (or expected) mean and the standard deviation of the derived distribution to standardize Slater distance scores across different grid sizes. 'Hartmann distances' represent a more accurate version of 'Slater distances'. Note that Hartmann distances are multiplied by `-1`. Hence, negative Hartmann values represent dissimilarity, i.e. a big Slater distance.

There are two ways to use this function. Hartmann distances can either be calculated based on the reference values (i.e. means and standard deviations of Slater distance distributions) as given by Hartmann in his paper. The second option is to conduct an instant simulation for the supplied grid size for each calculation. The second option will be more accurate when a big number of quasis is used in the simulation.

It is also possible to return the quantiles of the sample distribution and only the element distances considered 'significant' according to the quantiles defined.

Value

A matrix containing Hartmann distances.
In the attributes several additional parameters can be found:

`"arguments"`	A list of several parameters including `mean` and `sd` of Slater distribution.
`"quantiles"`	Quantiles for Slater and Hartmann distance distribution.
`"distributions"`	List with values of the simulated distributions.

Calculation

The 'Hartmann distance' is calculated as follows (Hartmann 1992, p. 49).

$$D = -1(\frac{D_{slater} - M_c}{sd_c})$$

Where D_{slater} denotes the Slater distances of the grid, M_c the sample distribution's mean value and sd_c the sample distribution's standard deviation.

References

Hartmann, A. (1992). Element comparisons in repertory grid technique: Results and consequences of a Monte Carlo study. *International Journal of Personal Construct Psychology, 5*(1), 41-56.

See Also

distanceSlater

Examples

```
## Not run:

   ### basics  ###

   distanceHartmann(bell2010)
   distanceHartmann(bell2010, method="simulate")
   h <- distanceHartmann(bell2010, method="simulate")
   h

   # printing options
   print(h)
   print(h, digits=6)
   # 'significant' distances only
   print(h, p=c(.05, .95))

   # access cells of distance matrix
   h[1,2]

   ### advanced  ###

   # histogram of Slater distances and indifference region
   h <- distanceHartmann(bell2010, distributions=TRUE)
   l <- attr(h, "distributions")
   hist(l$slater, breaks=100)
   hist(l$hartmann, breaks=100)

## End(Not run)
```

`distanceNormalized` *Standardized inter-element distances' (power transformed Hartmann distances).*

Description

Calculate power-transformed Hartmann distances.

Usage

```
distanceNormalized(
  x,
  reps = 1000,
  prob = NULL,
  progress = TRUE,
  distributions = TRUE
)
```

Arguments

`x`	`repgrid` object.
`reps`	Number of random grids to generate to produce sample distribution for Hartmann distances (default is `1000`). Note that a lot of samples may take a while to calculate.
`prob`	The probability of each rating value to occur. If `NULL` (default) the distribution is uniform. The number of values must match the length of the rating scale.
`progress`	Whether to show a progress bar during simulation (default is `TRUE`) (for `method="simulate"`). May be useful when the distribution is estimated on the basis of many quasis.
`distributions`	Whether to additionally return the values of the simulated distributions (Slater etc.) The default is `FALSE` as it will quickly boost the object size.

Details

Hartmann (1992) suggested a transformation of Slater (1977) distances to make them independent from the size of a grid. Hartmann distances are supposed to yield stable cutoff values used to determine 'significance' of inter-element distances. It can be shown that Hartmann distances are still affected by grid parameters like size and the range of the rating scale used (Heckmann, 2012). The function `distanceNormalize` applies a Box-Cox (1964) transformation to the Hartmann distances in order to remove the skew of the Hartmann distance distribution. The normalized values show to have more stable cutoffs (quantiles) and better properties for comparison across grids of different size and scale range.

The function `distanceNormalize` can also return the quantiles of the sample distribution and only the element distances considered 'significant' according to the quantiles defined.

Value

A matrix containing the standardized distances.
Further data is contained in the object's attributes:

`"arguments"`	A list of several parameters including `mean` and `sd` of Slater distribution.
`"quantiles"`	Quantiles for Slater, Hartmann and power transformed distance distributions.
`"distributions"`	List with values of the simulated distributions, if `distributions=TRUE`.

Calculations

The 'power transformed Hartmann distance' are calculated as follows: The simulated Hartmann distribution is added a constant as the Box-Cox transformation can only be applied to positive values. Then a range of values for lambda in the Box-Cox transformation (Box & Cox, 1964) are tried out. The best lambda is the one maximizing the correlation of the quantiles with the standard normal distribution. The lambda value maximizing normality is used to transform Hartmann distances. As the resulting scale of the power transformation depends on lambda, the resulting values are z-transformed to derive a common scaling.

The code for the calculation of the optimal lambda was written by Ioannis Kosmidis.

References

Box, G. E. P., & Cox, D. R. (1964). An Analysis of Transformations. *Journal of the Royal Statistical Society. Series B (Methodological), 26*(2), 211-252.

Hartmann, A. (1992). Element comparisons in repertory grid technique: Results and consequences of a Monte Carlo study. *International Journal of Personal Construct Psychology, 5*(1), 41-56.

Heckmann, M. (2012). Standardizing inter-element distances in grids - A revision of Hartmann's distances, 11th Biennal Conference of the European Personal Construct Association (EPCA), Dublin, Ireland, Paper presentation, July 2012.

Slater, P. (1977). *The measurement of intrapersonal space by Grid technique*. London: Wiley.

See Also

`distanceHartmann` and `distanceSlater`.

Examples

```
## Not run:

   ### basics  ###

   distanceNormalized(bell2010)
   n <- distanceNormalized(bell2010)
   n

   # printing options
   print(n)
```

```
    print(n, digits=4)
    # 'significant' distances only
    print(n, p=c(.05, .95))

    # access cells of distance matrix
    n[1,2]

    ### advanced  ###

    # histogram of Slater distances and indifference region
    n <- distanceNormalized(bell2010, distributions=TRUE)
    l <- attr(n, "distributions")
    hist(l$bc, breaks=100)


## End(Not run)
```

---

distanceSlater *'Slater distances' (standardized Euclidean distances).*

Description

Calculate Slater distance.

Usage

```
distanceSlater(x, trim = 20, index = TRUE)
```

Arguments

`x`	`repgrid` object.
`trim`	The number of characters a construct or element is trimmed to (default is `20`). If `NA` no trimming occurs. Trimming simply saves space when displaying correlation of constructs with long names.
`index`	Whether to print the number of the construct or element in front of the name (default is `TRUE`). This is useful to avoid identical row names, which may cause an error.

Details

The euclidean distance is often used as a measure of similarity between elements (see `distance`. A drawback of this measure is that it depends on the range of the rating scale and the number of constructs used, i. e. on the size of a grid.
An approach to standardize the euclidean distance to make it independent from size and range of ratings and was proposed by Slater (1977, pp. 94). The 'Slater distance' is the Euclidean distance divided by the expected distance. Slater distances bigger than 1 are greater than expected, lesser than 1 are smaller than expected. The minimum value is 0 and values bigger than 2 are rarely found. Slater

distances have been be used to compare inter-element distances between different grids, where the grids do not need to have the same constructs or elements. Hartmann (1992) showed that Slater distance is not independent of grid size. Also the distribution of the Slater distances is asymmetric. Hence, the upper and lower limit to infer 'significance' of distance is not symmetric. The practical relevance of Hartmann's findings have been demonstrated by Schoeneich and Klapp (1998). To calculate Hartmann's version of the standardized distances see distanceHartmann

Value

A matrix with Slater distances.

Calculation

The Slater distance is calculated as follows. For a derivation see Slater (1977, p.94).
Let matrix D contain the row centered ratings. Then

$$P = D^T D$$

and

$$S = tr(P)$$

The expected 'unit of expected distance' results as

$$U = (2S/(m-1))^{1/2}$$

where m denotes the number of elements of the grid. The standardized Slater distances is the matrix of Euclidean distances E divided by the expected distance U.

$$E/U$$

References

Hartmann, A. (1992). Element comparisons in repertory grid technique: Results and consequences of a Monte Carlo study. *International Journal of Personal Construct Psychology, 5*(1), 41-56.

Schoeneich, F., & Klapp, B. F. (1998). Standardization of interelement distances in repertory grid technique and its consequences for psychological interpretation of self-identity plots: An empirical study. *Journal of Constructivist Psychology, 11*(1), 49-58.

Slater, P. (1977). *The measurement of intrapersonal space by Grid technique.* Vol. II. London: Wiley.

See Also

distanceHartmann

Examples

```
distanceSlater(bell2010)
distanceSlater(bell2010, trim=40)

d <- distanceSlater(bell2010)
```

```
print(d)
print(d, digits=4)

# using Norris and Makhlouf-Norris (problematic) cutoffs
print(d, cutoffs=c(.8, 1.2))
```

`elementCor` *Calculate the correlations between elements.*

Description

Note that simple element correlations as a measure of similarity are flawed as they are not invariant to construct reflection (Mackay, 1992; Bell, 2010). A correlation index invariant to construct reflection is Cohen's rc measure (1969), which can be calculated using the argument `rc=TRUE` which is the default option.

Usage

```
elementCor(x, rc = TRUE, method = "pearson", trim = 20, index = TRUE)
```

Arguments

`x`	`repgrid` object.
`rc`	Use Cohen's rc which is invariant to construct reflection (see description above). It is used as the default.
`method`	A character string indicating which correlation coefficient is to be computed. One of `"pearson"` (default), `"kendall"` or `"spearman"`, can be abbreviated. The default is `"pearson"`.
`trim`	The number of characters a construct is trimmed to (default is `20`). If `NA` no trimming occurs. Trimming simply saves space when displaying correlation of constructs with long names.
`index`	Whether to print the number of the construct.

Value

`matrix` of element correlations

References

Bell, R. C. (2010). A note on aligning constructs. *Personal Construct Theory & Practice*, (7), 42-48.

Cohen, J. (1969). rc: A profile similarity coefficient invariant over variable reflection. *Psychological Bulletin, 71*(4), 281-284.

Mackay, N. (1992). Identification, Reflection, and Correlation: Problems In The Bases Of Repertory Grid Measures. *International Journal of Personal Construct Psychology, 5*(1), 57-75.

See Also

constructCor

Examples

```
elementCor(mackay1992)                      # Cohen's rc
elementCor(mackay1992, rc=FALSE)            # PM correlation
elementCor(mackay1992, rc=FALSE, method="spearman")  # Spearman correlation

# format output
elementCor(mackay1992, trim=6)
elementCor(mackay1992, index=FALSE, trim=6)

# save as object for further processing
r <- elementCor(mackay1992)
r

# change output of object
print(r, digits=5)
print(r, col.index=FALSE)
print(r, upper=FALSE)

# accessing elements of the correlation matrix
r[1,3]
```

---

elementRmsCor *Root mean square (RMS) of inter-element correlations.*

Description

The RMS is also known as 'quadratic mean' of the inter-element correlations. The RMS serves as a simplification of the correlation table. It reflects the average relation of one element with all other elements. Note that as the correlations are squared during its calculation, the RMS is not affected by the sign of the correlation (cf. Fransella, Bell & Bannister, 2003, p. 86).

Usage

```
elementRmsCor(x, rc = TRUE, method = "pearson", trim = NA)
```

Arguments

x	`repgrid` object.
rc	Whether to use Cohen's rc which is invariant to construct reflection (see description above). It is used as the default.
method	A character string indicating which correlation coefficient to be computed. One of `"pearson"` (default), `"kendall"` or `"spearman"`, can be abbreviated. The default is `"pearson"`.

`trim` The number of characters an element is trimmed to (default is `NA`). If `NA` no trimming occurs. Trimming simply saves space when displaying correlation of constructs with long names.

Details

Note that simple element correlations as a measure of similarity are flawed as they are not invariant to construct reflection (Mackay, 1992; Bell, 2010). A correlation index invariant to construct reflection is Cohen's rc measure (1969), which can be calculated using the argument `rc=TRUE` which is the default option in this function.

Value

`dataframe` of the RMS of inter-element correlations

References

Fransella, F., Bell, R. C., & Bannister, D. (2003). A Manual for Repertory Grid Technique (2. Ed.). Chichester: John Wiley & Sons.

See Also

`constructRmsCor`, `elementCor`

Examples

```
# data from grid manual by Fransella, Bell and Bannister
elementRmsCor(fbb2003)
elementRmsCor(fbb2003, trim=10)

# modify output
r <- elementRmsCor(fbb2003)
print(r, digits=5)

# access second row of calculation results
r[2, "RMS"]
```

`elements` *Get or replace element names*

Description

Allows to get and set element names. Replaces the older functions `getElementNames`, `getElementNames2`, and `eNames` which are deprecated.

Usage

```
elements(x)

elements(x, position) <- value
```

Arguments

`x`	A repgrid object.
`position`	Index where to insert element.
`value`	Character vector of element names.

Examples

```
# copy Boeker grid to x
x <- boeker

## get element names
e <- elements(x)
e

## replace element names
elements(x) <- rev(e)       # reverse all element names
elements(x)[1] <- "Hannes"  # replace name of first element

# replace names of elements 1 and 3
elements(x)[c(1,3)] <- c("element 1", "element 3")
```

`gridlist` *Add repgrids into a gridlist*

Description

Add repgrids into a gridlist

Test or create object of class ‘gridlist‘

Usage

```
gridlist(...)

is.gridlist(x)

as.gridlist(x)
```

Arguments

...	Objects to be converted into ‘gridlist‘
x	Any object.

---

grids_leave_n_out   *Resample constructs*

Description

The goal of resampling is to build variations of a single grid. Two variants are implemented: The first is the *leave-n-out* approach which builds all possible grids when dropping n constructs. The second is a *bootstrap* approach, randomly drawing n constructs from the grid.

Usage

```
grids_leave_n_out(x, n = 0)

grids_bootstrap(x, n = nrow(x), reps = 100, replace = TRUE)
```

Arguments

x	A repgrid object.
n	Number of constructs to drop or to sample in each generated grid.
reps	Number of grids to generate.
replace	Resample constructs with replacement?

Value

List of grids.

Examples

```
## All results for PVAFF index when one construct is left out
p <- indexPvaff(boeker)
l <- grids_leave_n_out(boeker, n = 1)
pp <- sapply(l, indexPvaff)  # apply indexPvaff function to all grids
range(pp)   # min and max PVAFF
hist(pp, xlab = "PVAFF values")    # visualize
abline(v = p, col = "blue", lty = 2)
```

home *Rotate the interactive 3D device to default views.*

Description

Rotate the interactive 3D device to a default viewpoint or to a position defined by `theta` and `phi` in Euler angles. Three default viewpoints are implemented rendering a view so that two axes span a plane and the third axis is pointing out of the screen.

Usage

```
home(view = 1, theta = NULL, phi = NULL)
```

Arguments

`view`	Numeric. Specifying one of three default views. 1 = XY, 2=XZ and 3=YZ-plane.
`theta`	Numeric. Euler angle. Overrides view setting.
`phi`	Numeric. Euler angle. Overrides view setting. return `NULL`.

See Also

Interactive 3D biplots: `biplot3d`, `biplotSlater3d`, `biplotEsa3d`.

Examples

```
## Not run:

   biplot3d(boeker)
   home(2)
   home(3)
   home(1)
   home(theta=45, phi=45)


## End(Not run)
```

`importExcel` *Import grid data from an Excel file.*

Description

You can define a grid using Microsoft Excel and by saving it as a `.xlsx` file. The `.xlsx` file has to be in a specified fixed format (see section Details).

Usage

```
importExcel(file, dir = NULL, sheetIndex = 1, min = NULL, max = NULL)
```

Arguments

`file`	A vector of filenames including the full path if file is not in current working directory. The file suffix has to be `.xlsx` (used since Excel 2007).
`dir`	Alternative way to supply the directory where the file is located (default `NULL`).
`sheetIndex`	The number of the Excel sheet that contains the grid data.
`min`	Optional argument (`numeric`, default `NULL`) for minimum rating value in grid.
`max`	Optional argument (`numeric`, default `NULL`) for maximum rating value in grid.

Details

Excel file structure: The first row contains the minimum of the rating scale, the names of the elements and the maximum of the rating scale. Below every row contains the left construct pole, the ratings and the right construct pole.

1	E1	E2	E3	E4	5
left pole 1	1	5	3	4	right pole 1
left pole 2	3	1	1	3	right pole 2
left pole 3	4	2	5	1	right pole 3

Note that the maximum and minimum value has to be defined using the `min` and `max` arguments if no values are supplied at the beginning and end of the first row. Otherwise the scaling range is inferred from the available data and a warning is issued as the range may be erroneous. This may effect other functions that depend on knowing the correct range and it is thus strongly recommended to set the scale range correctly.

Value

A single `repgrid` object in case one file and a list of `repgrid` objects in case multiple files are imported.

See Also

importGridcor, importGridstat, importScivesco, importGridsuite, importTxt

Examples

```
## Not run:

# Open Excel file delivered along with the package
file <- system.file("extdata", "grid_01.xlsx", package = "OpenRepGrid")
rg <- importExcel(file)

# To see the structure of the Excel file try to open it as follows.
# Requires Excel to be installed.
system2("open", file)

# Import more than one Excel file
files <- system.file("extdata", c("grid_01.xlsx", "grid_02.xlsx") , package = "OpenRepGrid")
rg <- importExcel(files)


## End(Not run)
```

---

importGridcor *Import GRIDCOR data files.*

Description

Reads the file format that is used by the grid program GRIDCOR (Feixas & Cornejo, 2002).

Usage

```
importGridcor(file, dir = NULL)
```

Arguments

file	filename including path if file is not in current working directory. File can also be a complete URL. The fileformat is .dat.
dir	alternative way to supply the directory where the file is located (default NULL).

Value

a single repgrid object in case one file and a list of repgrid objects in case multiple files are imported.

Note

Note that the GRIDCOR data sets the minimum ratings scale range to 1. The maximum value can differ and is defined in the data file.

Also note that both Gridcor and Gridstat data files do have the same suffix .dat. Make sure not to mix them up.

References

Feixas, G., & Cornejo, J. M. (2002). GRIDCOR: Correspondence Analysis for Grid Data (version 4.0). Barcelona: Centro de Terapia Cognitiva. Retrieved from https://www.ub.edu/terdep/pag/index.html.

See Also

importGridcor, importGridstat, importScivesco, importGridsuite, importTxt, importExcel

Examples

```
## Not run:

# supposing that the data file gridcor.dat is in the current directory
file <- "gridcor.dat"
rg <- importGridcor(file)

# specifying a directory (arbitrary example directory)
dir <- "/Users/markheckmann/data"
rg <- importGridcor(file, dir)

# using a full path
rg <- importGridcor("/Users/markheckmann/data/gridcor.dat")


## End(Not run)
```

---

importGridstat — *Import Gridstat data files.*

Description

Reads the file format that is used by the latest version of the grid program gridstat (Bell, 1998).

Usage

```
importGridstat(file, dir = NULL, min = NULL, max = NULL)
```

Arguments

`file`	Filename including path if file is not in current working directory. File can also be a complete URL. The fileformat is `.dat`.
`dir`	Alternative way to supply the directory where the file is located (default `NULL`).
`min`	Optional argument (`numeric`, default `NULL`) for minimum rating value in grid.
`max`	Optional argument (`numeric`, default `NULL`) for maximum rating value in grid.

Value

A single `repgrid` object in case one file and a list of `repgrid` objects in case multiple files are imported.

Note

Note that the gridstat data format does not contain explicit information about the range of the rating scale used (minimum and maximum). By default the range is inferred by scanning the ratings and picking the minimal and maximal values as rating range. You can set the minimal and maximal value by hand using the `min` and `max` arguments or by using the `setScale()` function. Note that if the rating range is not set, it may cause several functions to not work properly. A warning will be issued if the range is not set explicitly when using the importing function.

The function only reads data from the latest GridStat version. The latest version allows the separation of the left and right pole by using on of the following symbols `/:-` (hyphen, colon and dash). Older versions may not separate the left and right pole. This will cause all labels to be assigned to the left pole only when importing. You may fix this by simply entering one of the construct separator symbols into the GridStat file between each left and right construct pole.

The third line of a GridStat file may contain a no labels statement (i.e. a line containing any string of 'NOLA', 'NO L', 'NoLa', 'No L', 'Nola', 'No l', 'nola' or 'no l'). In this case only ratings are supplied, hence, default names are assigned to elements and constructs.

References

Bell, R. C. (1998) GRIDSTAT: A program for analyzing the data of a repertory grid. Melbourne: Author.

See Also

`importGridcor`, `importGridstat`, `importScivesco`, `importGridsuite`, `importTxt`, `importExcel`

Examples

```
## Not run:

# supposing that the data file gridstat.dat is in the current working directory
file <- "gridstat.dat"
rg <- importGridstat(file)

# specifying a directory (example)
dir <- "/Users/markheckmann/data"
```

```
rg <- importGridstat(file, dir)

# using a full path (example)
rg <- importGridstat("/Users/markheckmann/data/gridstat.dat")

# setting rating scale range
rg <- importGridstat(file, dir, min=1, max=6)

## End(Not run)
```

---

importGridsuite *Import Gridsuite data files.*

Description

Import Gridsuite data files.

Usage

```
importGridsuite(file, dir = NULL)
```

Arguments

file	Filename including path if file is not in current working directory. File can also be a complete URL. The fileformat is .dat.
dir	Alternative way to supply the directory where the file is located (default NULL).

Value

A single repgrid object in case one file and a list of repgrid objects in case multiple files are imported.

Note

The developers of Gridsuite have proposed to use an XML scheme as a standard exchange format for repertory grid data (Walter, Bacher & Fromm, 2004). This approach is also embraced by the OpenRepGrid package.

TODO: The element and construct IDs are not used yet. Thus, if the output should be in different order the current mechanism will cause false assignments.

References

http://www.gridsuite.de/

Walter, O. B., Bacher, A., & Fromm, M. (2004). A proposal for a common data exchange format for repertory grid data. *Journal of Constructivist Psychology, 17*(3), 247. doi:10.1080/10720530490447167

See Also

importGridcor, importGridstat, importScivesco, importGridsuite, importTxt, importExcel

Examples

```
## Not run:

# supposing that the data file gridsuite.xml is in the current directory
file <- "gridsuite.xml"
rg <- importGridsuite(file)

# specifying a directory (arbitrary example directory)
dir <- "/Users/markheckmann/data"
rg <- importGridsuite(file, dir)

# using a full path
rg <- importGridsuite("/Users/markheckmann/data/gridsuite.xml")


## End(Not run)
```

---

importScivesco *Import sci:vesco data files.*

Description

Import sci:vesco data files.

Usage

```
importScivesco(file, dir = NULL)
```

Arguments

file	Filename including path if file is not in current working directory. File can also be a complete URL. The fileformat is .dat.
dir	Alternative way to supply the directory where the file is located (default NULL).

Value

A single repgrid object in case one file and a list of repgrid objects in case multiple files are imported.

Note

Sci:Vesco offers the options to rate the construct poles separately or using a bipolar scale. The separated rating is done using the "tetralemma" field. The field is a bivariate plane on which each of the four (tetra) corners has a different meaning in terms of rating. Using this approach also allows ratings like: "both poles apply", "none of the poles apply" and all intermediate ratings can be chosen. This relaxes the bipolarity assumption often assumed in grid theory and allows for deviation from a strict bipolar rating if the constructs are not applied in a bipolar way. Using the tetralemma field for rating requires to analyze each construct separately though. This means we get a double entry grid where the emergent and contrast pole ratings might not simply be a reflection of on another. The tetralemma field is not yet supported and importing will fail. Currently only bipolar ratings are supported.

If a tetralemma field has been used for rating, `OpenRepGrid` will offer the option to transform the scores into "normal" grid ratings (i.e. restricted to bipolarity) by projecting the ratings from the bivariate tetralemma field onto the diagonal of the tetralemma field and thus forcing a bipolar rating type. This option is not recommended due to the fact that the conversion is susceptible to error when both ratings are near to zero.

TODO: For developers: The element IDs are not used yet. This might cause wrong assignments.

References

Menzel, F., Rosenberger, M., Buve, J. (2007). Emotionale, intuitive und rationale Konstrukte verstehen. *Personalfuehrung, 4*(7), 91-99.

See Also

`importGridcor`, `importGridstat`, `importScivesco`, `importGridsuite`, `importTxt`, `importExcel`

Examples

```
## Not run:

# supposing that the data file scivesco.scires is in the current directory
file <- "scivesco.scires"
rg <- importScivesco(file)

# specifying a directory (arbitrary example directory)
dir <- "/Users/markheckmann/data"
rg <- importScivesco(file, dir)

# using a full path
rg <- importScivesco("/Users/markheckmann/data/scivesco.scires")


## End(Not run)
```

importTxt *Import grid data from a text file.*

Description

If you do not have a grid program at hand you can define a grid using a standard text editor and by saving it as a `.txt` file. The `.txt` file has to be in a fixed format. There are three mandatory blocks each starting and ending with a predefined tag in uppercase letters. The first block starts with `ELEMENTS` and ends with `END ELEMENTS` and contains one element in each line. The other mandatory blocks contain the constructs and ratings (see below). In the block containing the constructs the left and right pole are separated by a colon (:). To define missing values use `NA` like in the example below. One optional block contains the range of the rating scale used defined by two numbers. The order of the blocks is arbitrary. All text not contained within the blocks is discarded and can thus be used for comments.

Usage

```
importTxt(file, dir = NULL, min = NULL, max = NULL)
```

Arguments

`file`	A vector of filenames including the full path if file is not in current working directory. File can also be a complete URL. The file suffix has to be `.txt`.
`dir`	Alternative way to supply the directory where the file is located (default `NULL`).
`min`	Optional argument (`numeric`, default `NULL`) for minimum rating value in grid.
`max`	Optional argument (`numeric`, default `NULL`) for maximum rating value in grid.

Details

```
---------------- .txt file -----------------

anything not contained within the tags will be discarded
ELEMENTS
element 1
element 2
element 3
END ELEMENTS

CONSTRUCTS
left pole 1 : right pole 1
left pole 2 : right pole 2
left pole 3 : right pole 3
left pole 4 : right pole 4
END CONSTRUCTS

RATINGS
```

```
1 3 2
4 1 1
1 4 4
3 1 1
END RATINGS

RANGE
1 4
END RANGE
---------------- end of file ----------------
```

Note that the maximum and minimum value has to be defined using the `min` and `max` arguments if no `RANGE` block is contained in the data file. Otherwise the scaling range is inferred from the available data and a warning is issued as the range may be erroneous. This may effect other functions that depend on knowing the correct range and it is thus strongly recommended to set the scale range correctly.

Value

A single `repgrid` object in case one file and a list of `repgrid` objects in case multiple files are imported.

See Also

`importGridcor`, `importGridstat`, `importScivesco`, `importGridsuite`, `importTxt`, `importExcel`

Examples

```
## Not run:

# supposing that the data file sample.txt is in the current directory
file <- "sample.txt"
rg <- importTxt(file)

# specifying a directory (arbitrary example directory)
dir <- "/Users/markheckmann/data"
rg <- importTxt(file, dir)

# using a full path
rg <- importTxt("/Users/markheckmann/data/sample.txt")

# importing more than one .txt file via R code
files <- c("sample.txt", "sample_2.txt")
rg <- importTxt(files)

## End(Not run)
```

indexBias *Calculate 'bias' of grid as defined by Slater (1977).*

Description

"Bias records a tendency for responses to accumulate at one end of the grading scale" (Slater, 1977, p.88).

Usage

```
indexBias(x, min = NULL, max = NULL, digits = 2)
```

Arguments

x	repgrid object.
min, max	Minimum and maximum grid scale values. Nor needed if they are set for the grid.
digits	Numeric. Number of digits to round to (default is 2).

Value

Numeric.

Note

STATUS: Working and checked against example in Slater, 1977, p. 87.

References

Slater, P. (1977). *The measurement of intrapersonal space by Grid technique*. London: Wiley.

See Also

indexVariability

Examples

```
indexBias(boeker)
```

`indexBieri` *Bieri's index of cognitive complexity*

Description

The index builds on the number of rating matches between pairs of constructs. It is the relation between the total number of matches and the possible number of matches.

Usage

```
indexBieri(x, deviation = 0)
```

Arguments

`x`	A `repgrid` object.
`deviation`	Maximal difference between ratings to be considered a match (default `0` = identical scores for a match).

Details

CAVEAT: The Bieri index will change when constructs are reversed.

Value

List of class `indexBieri`:

- `grid`: The grid used to calculate the index
- `deviation` The deviation parameter.
- `matches_max` Maximum possible number of matches across constructs.
- `matches` Total number of matches across constructs.
- `constructs`: Matrix with no. of matches for constructs.
- `bieri`: Bieri index (= matches / matches_max)

Examples

```
m <- indexBieri(boeker)

# several output options
print(m)
print(m, output = "IC")  # construct matches

# extract the matrix of matches
m$constructs

# CAVEAT: Bieri's index changes when constructs are reversed
nr <- nrow(boeker)
l <- replicate(1000, swapPoles(boeker, sample(nr, sample(nr, 1))))
```

```
bieri <- sapply(l, function(x) indexBieri(x)$bieri)
hist(bieri, breaks = 50)
abline(v = mean(bieri), col = "red", lty = 2)
```

---

indexConflict1 — *Conflict measure for grids (Slade & Sheehan, 1979) based on correlations.*

Description

Conflict measure as proposed by Slade and Sheehan (1979)

Usage

```
indexConflict1(x)
```

Arguments

x	`repgrid` object.

Details

The first approach to mathematically derive a conflict measure based on grid data was presented by Slade and Sheehan (1979). Their operationalization is based on an approach by Lauterbach (1975) who applied the *balance theory* (Heider, 1958) for a quantitative assessment of psychological conflict. It is based on a count of balanced and imbalanced triads of construct correlations. A triad is imbalanced if one or all three of the correlations are negative, i. e. leading to contrary implications. This approach was shown by Winter (1982) to be flawed. An improved version was proposed by Bassler et al. (1992) and has been implemented in the function `indexConflict2`.

The table below shows when a triad made up of the constructs A, B, and C is balanced and imbalanced.

cor(A,B)	cor(A,C)	cor(B,C)	Triad characteristic
+	+	+	balanced
+	+	-	imbalanced
+	-	+	imbalanced
+	-	-	balanced
-	+	+	imbalanced
-	+	-	balanced
-	-	+	balanced
-	-	-	imbalanced

Value

A list with the following elements:

`total`	Total number of triads
`imbalanced`	Number of imbalanced triads
`prop.balanced`	Proportion of balanced triads
`prop.imbalanced`	Proportion of imbalanced triads

References

Bassler, M., Krauthauser, H., & Hoffmann, S. O. (1992). A new approach to the identification of cognitive conflicts in the repertory grid: An illustrative case study. *Journal of Constructivist Psychology, 5*(1), 95-111.

Heider, F. (1958). *The Psychology of Interpersonal Relation*. John Wiley & Sons.

Lauterbach, W. (1975). Assessing psychological conflict. *The British Journal of Social and Clinical Psychology, 14*(1), 43-47.

Slade, P. D., & Sheehan, M. J. (1979). The measurement of 'conflict' in repertory grids. *British Journal of Psychology, 70*(4), 519-524.

Winter, D. A. (1982). Construct relationships, psychological disorder and therapeutic change. *The British Journal of Medical Psychology, 55* (Pt 3), 257-269.

See Also

`indexConflict2` for an improved version of this measure; see `indexConflict3` for a measure based on distances.

Examples

```
indexConflict1(feixas2004)
indexConflict1(boeker)
```

`indexConflict2` *Conflict measure for grids (Bassler et al., 1992) based on correlations.*

Description

Conflict measure as proposed by Bassler et al. (1992).

Usage

```
indexConflict2(x, crit = 0.03)
```

Arguments

`x`	`repgrid` object.
`crit`	Sensitivity criterion with which triads are marked as unbalanced. A bigger values will lead to less imbalanced triads. The default is `0.03`. The value should be adjusted with regard to the researchers interest.

Details

The function calculates the conflict measure as devised by Bassler et al. (1992). It is an improved version of the ideas by Slade and Sheehan (1979) that have been implemented in the function indexConflict1. The new approach also takes into account the magnitude of the correlations in a trait to assess whether it is balanced or imbalanced. As a result, small correlations that are psychologically meaningless are considered accordingly. Also, correlations with a small magnitude, i. e. near zero, which may be positive or negative due to chance alone will no longer distort the measure (Bassler et al., 1992).

Description of the balance / imbalance assessment:

1. Order correlations of the triad by absolute magnitude, so that $r_{max} > r_{mdn} > r_{min}$.
2. Apply Fisher's Z-transformation and division by 3 to yield values between 1 and -1
3. Check whether the triad is balanced by assessing if the following relation holds:
   - If $Z_{max} Z_{mdn} > 0$, the triad is balanced if $Z_{max} Z_{mdn} - Z_{min} <= crit$ Z_max x Z_mdn - Z_min <= crit .
   - If $Z_{max} Z_{mdn} < 0$, the triad is balanced if $Z_{min} - Z_{max} Z_{mdn} <= crit$ Z_min - Z_max x Z_mdn <= crit .

Personal remarks (MH)

I am a bit suspicious about step 2 from above. To devide by 3 appears pretty arbitrary. The r for a z-values of 3 is 0.9950548 and not 1. The r for 4 is 0.9993293. Hence, why not a value of 4, 5, or 6? Denoting the value to devide by with `a`, the relation for the first case translates into $a Z_{max} Z_{mdn} <= \frac{crit}{a} + Z_{min}$ a x Z_max x Z_mdn =< crit/a + Z_min. This shows that a bigger value of `a` will make it more improbable that the relation will hold.

References

Bassler, M., Krauthauser, H., & Hoffmann, S. O. (1992). A new approach to the identification of cognitive conflicts in the repertory grid: An illustrative case study. *Journal of Constructivist Psychology, 5*(1), 95-111.

Slade, P. D., & Sheehan, M. J. (1979). The measurement of 'conflict' in repertory grids. *British Journal of Psychology, 70*(4), 519-524.

See Also

See indexConflict1 for the older version of this measure; see indexConflict3 for a measure based on distances instead of correlations.

Examples

```
## Not run:

 indexConflict2(bell2010)

 x <- indexConflict2(bell2010)
 print(x)
```

```
 # show conflictive triads
 print(x, output = 2)

 # accessing the calculations for further use
 x$total
 x$imbalanced
 x$prop.balanced
 x$prop.imbalanced
 x$triads.imbalanced

## End(Not run)
```

`indexConflict3` *Conflict or inconsistency measure for grids (Bell, 2004) based on distances.*

Description

Conflict measure as proposed by Bell (2004).

Usage

```
indexConflict3(
  x,
  p = 2,
  e.out = NA,
  e.threshold = NA,
  c.out = NA,
  c.threshold = NA,
  trim = 20
)
```

Arguments

x	`repgrid` object.
p	The power of the Minkowski distance. `p=2` (default) will result in euclidean distances, `p=1` in city block distances.
e.out	Numeric. A vector giving the indexes of the elements for which detailed stats (number of conflicts per element, discrepancies for triangles etc.) are prompted (default `NA`, i.e. no detailed stats for any element).
e.threshold	Numeric. Detailed stats are prompted for those elements with a an attributable percentage to the overall conflicts higher than the supplied threshold (default `NA`).
c.out	Numeric. A vector giving the indexes of the constructs for which detailed stats (discrepancies for triangles etc.) are prompted (default `NA`, i. e. no detailed stats).

`c.threshold`	Numeric. Detailed stats are prompted for those constructs with a an attributable percentage to the overall conflicts higher than the supplied threshold (default `NA`).
`trim`	The number of characters a construct (element) is trimmed to (default is `10`). If `NA` no trimming is done. Trimming simply saves space when displaying the output.

Details

Measure of conflict or inconsistency as proposed by Bell (2004). The identification of conflict is based on distances rather than correlations as in other measures of conflict `indexConflict1` and `indexConflict2`. It assesses if the distances between all components of a triad, made up of one element and two constructs, satisfies the "triangle inequality" (cf. Bell, 2004). If not, a triad is regarded as conflictive. An advantage of the measure is that it can be interpreted not only as a global measure for a grid but also on an element, construct, and element by construct level making it valuable for detailed feedback. Also, differences in conflict can be submitted to statistical testing procedures.

Status: working; output for euclidean and manhattan distance checked against Gridstat output.
TODO: standardization and z-test for discrepancies; Index of Conflict Variation.

Value

A list (invisibly) containing containing:

`potential`	number of potential conflicts
`actual`	count of actual conflicts
`overall`	percentage of conflictive relations
`e.count`	number of involvements of each element in conflictive relations
`e.perc`	percentage of involvement of each element in total of conflictive relations
`e.count`	number of involvements of each construct in conflictive relation
`c.perc`	percentage of involvement of each construct in total of conflictive relations
`e.stats`	detailed statistics for prompted elements
`c.stats`	detailed statistics for prompted constructs
`e.threshold`	threshold percentage. Used by print method
`c.threshold`	threshold percentage. Used by print method
`enames`	trimmed element names. Used by print method
`cnames`	trimmed construct names. Used by print method

output

For further control over the output see `print.indexConflict3`.

References

Bell, R. C. (2004). A new approach to measuring inconsistency or conflict in grids. Personal Construct Theory & Practice, (1), 53-59.

See Also

See `indexConflict1` and `indexConflict2` for conflict measures based on triads of correlations.

Examples

```
## Not run:

 # calculate conflicts
 indexConflict3(bell2010)

 # show additional stats for elements 1 to 3
 indexConflict3(bell2010, e.out = 1:3)

 # show additional stats for constructs 1 and 5
 indexConflict3(bell2010, c.out = c(1,5))

 # finetune output
 ## change number of digits
 x <- indexConflict3(bell2010)
 print(x, digits = 4)

 ## omit discrepancy matrices for constructs
 x <- indexConflict3(bell2010, c.out = 5:6)
 print(x, discrepancies = FALSE)


## End(Not run)
```

`indexDilemma` *Detect implicative dilemmas (conflicts).*

Description

Implicative Dilemmas

Usage

```
indexDilemma(
  x,
  self = 1,
  ideal = ncol(x),
  diff.mode = 1,
  diff.congruent = NA,
  diff.discrepant = NA,
  diff.poles = 1,
  r.min = 0.35,
  exclude = FALSE,
```

```
    digits = 2,
    show = FALSE,
    output = 1,
    index = TRUE,
    trim = 20
)
```

Arguments

`x`	`repgrid` object.
`self`	Numeric. Index of self element.
`ideal`	Numeric. Index of ideal self element.
`diff.mode`	Numeric. Method adopted to classify construct pairs into congruent and discrepant. With `diff.mode=1`, the minimal and maximal score difference criterion is applied. With `diff.mode=0` the Mid-point rating criterion is applied. Default is `diff.mode=1`.
`diff.congruent`	Is used if `diff.mode=1`. Maximal difference between element ratings to define construct as congruent (default `diff.congruent=1`). Note that the value needs to be adjusted by the user according to the rating scale used.
`diff.discrepant`	Is used if `diff.mode=1`. Minimal difference between element ratings to define construct as discrepant (default `diff.discrepant=3`). Note that the value needs to be adjusted by the user according to the rating scale used.
`diff.poles`	Not yet implemented.
`r.min`	Minimal correlation to determine implications between constructs.
`exclude`	Whether to exclude the elements self and ideal self during the calculation of the inter-construct correlations. (default is `FALSE`).
`digits`	Numeric. Number of digits to round to (default is `2`).
`show`	Whether to additionally plot the distribution of correlations to help the user assess what level is adequate for `r.min`.
`output`	The type of output to return.
`index`	Whether to print index numbers in front of each construct (default is `TRUE`).
`trim`	The number of characters a construct (element) is trimmed to (default is `20`). If `NA` no trimming is done. Trimming simply saves space when displaying the output.

Details

Implicative dilemmas are closely related to the notion of conflict. An implicative dilemma arises when a desired change on one construct is associated with an undesired implication on another construct. E. g. a timid subject may want to become more socially skilled but associates being socially skilled with different negative characteristics (selfish, insensitive etc.). Hence, he may anticipate that becoming less timid will also make him more selfish (cf. Winter, 1982). As a consequence, the subject will resist to the change if the negative presumed implications will threaten the patients identity and the predictive power of his construct system. From this stance the resistance to change is a logical consequence coherent with the subjects construct system (Feixas, Saul, & Sanchez, 2000). The investigation of the role of cognitive dilemma in different disorders in the context of PCP is a current field of research (e.g. Feixas & Saul, 2004, Dorough et al. 2007).

The detection of implicative dilemmas happens in two steps. First the constructs are classified as being 'congruent' or 'discrepant'. Secondly, the correlation between a congruent and discrepant construct pair is assessed if it is big enough to indicate an implication.

Classifying the construct
To detect implicit dilemmas the construct pairs are first identified as 'congruent' or 'discrepant'. The assessment is based on the rating differences between the elements 'self' and 'ideal self'. A construct is 'congruent' if the construction of the 'self' and the preferred state (i.e. ideal self) are the same or similar. A construct is discrepant if the construction of the 'self' and the 'ideal' is dissimilar.

There are two popular accepted methods to identify congruent and discrepant constructs:

1. "Scale Midpoint criterion" (cf. Grice 2008)
2. "Minimal and maximal score difference" (cf. Feixas & Saul, 2004)

"Scale Midpoint criterion" (cf. Grice 2008)

As reported in the Idiogrid (v. 2.4) manual: "[..] The Scale Midpoint uses the scales as the 'dividing line' for discrepancies; for example, if the actual element is rated above the midpoint, then the discrepancy exists (and vice versa). If the two selves are the same as the actual side of the scale, then a discrepancy does not exist". As an example: Assuming a scoring range of 1-7, the midpoint score will be 4, we then look at self and ideal-self scoring on any given construct and we proceed as follow:

- If the scoring of Self AND Ideal Self are both < 4: construct is "Congruent"
- If the scoring of Self AND Ideal Self are both > 4: construct is "Congruent"
- If the scoring of Self is < 4 AND Ideal Self is > 4 (OR vice versa): construct is "discrepant"
- If scoring Self OR Ideal Self = 4 then the construct is NOT Discrepant and it is "Undifferentiated"

Minimal and maximal score difference criterion (cf. Feixas & Saul, 2004)

This other method is more conservative and it is designed to minimize Type I errors by a) setting a default minimum correlation between constructs of `r = .34`; b) discarding cases where the ideal Self and self are neither congruent or discrepant; c) discarding cases where ideal self is "not oriented", i.e. scored at the midpoint.

E.g. suppose the element 'self' is rated 2 and 'ideal self' 5 on a scale from 1 to 6. The ratings differences are 5-2 = 3. If this difference is smaller than e.g. 1 the construct is 'congruent', if it is bigger than 3 it is 'discrepant'.

The values used to classify the constructs 'congruent' or 'discrepant' can be determined in several ways (cf. Bell, 2009):

1. They are set 'a priori'.
2. They are implicitly derived by taking into account the rating differences to the other constructs. (Not yet implemented)

The value mode is determined via the argument `diff.mode`.
If no 'a priori' criteria to determine whether the construct is congruent or discrepant is supplied as an argument, the values are chosen according to the range of the rating scale used. For the following scales the defaults are chosen as:

Scale	'A priori' criteria
1 2	–> con: <=0 disc: >=1
1 2 3	–> con: <=0 disc: >=2
1 2 3 4	–> con: <=0 disc: >=2
1 2 3 4 5	–> con: <=1 disc: >=3
1 2 3 4 5 6	–> con: <=1 disc: >=3
1 2 3 4 5 6 7	–> con: <=1 disc: >=4
1 2 3 4 5 6 7 8	–> con: <=1 disc: >=5
1 2 3 4 5 6 7 8 9	–> con: <=2 disc: >=5
1 2 3 4 5 6 7 8 9 10	–> con: <=2 disc: >=6

Defining the correlations
As the implications between constructs cannot be derived from a rating grid directly, the correlation between two constructs is used as an indicator for implication. A large correlation means that one construct pole implies the other. A small correlation indicates a lack of implication. The minimum criterion for a correlation to indicate implication is set to .35 (cf. Feixas & Saul, 2004). The user may also choose another value. To get a an impression of the distribution of correlations in the grid, a visualization can be prompted via the argument `show`. When calculating the correlation used to assess if an implication is given or not, the elements under consideration (i. e. self and ideal self) can be included (default) or excluded. The options will cause different correlations (see argument `exclude`).

Example of an implicative dilemma
A depressive person considers herself as 'timid' and wished to change to the opposite pole she defines as 'extraverted'. This construct is called discrepant as the construction of the 'self' and the desired state (e.g. described by the 'ideal self') on this construct differ. The person also considers herself as 'sensitive' (preferred pole) for which the opposite pole is 'selfish'. This construct is congruent, as the person construes herself as she would like to be. If the person now changed on the discrepant construct from the undesired to the desired pole, i.e. from timid to extraverted, the question can be asked what consequences such a change has. If the person construes being timid and being sensitive as related and that someone who is extraverted will not be timid, a change on the first construct will imply a change on the congruent construct as well. Hence, the positive shift from timid to extraverted is presumed to have a undesired effect in moving from sensitive towards selfish. This relation is called an implicative dilemma. As the implications of change on a construct cannot be derived from a rating grid directly, the correlation between two constructs is used as an indicator of implication.

Value

List object of class `indexDilemma`, containing the result from the calculations.

Author(s)

Mark Heckmann, Alejandro García, Diego Vitali

References

Bell, R. C. (2009). *Gridstat version 5 - A Program for Analyzing the Data of A Repertory Grid* (manual). University of Melbourne, Australia: Department of Psychology.

Dorough, S., Grice, J. W., & Parker, J. (2007). Implicative dilemmas and psychological well-being. *Personal Construct Theory & Practice*, (4), 83-101.

Feixas, G., & Saul, L. A. (2004). The Multi-Center Dilemma Project: an investigation on the role of cognitive conflicts in health. *The Spanish Journal of Psychology, 7*(1), 69-78.

Feixas, G., Saul, L. A., & Sanchez, V. (2000). Detection and analysis of implicative dilemmas: implications for the therapeutic process. In J. W. Scheer (Ed.), *The Person in Society: Challenges to a Constructivist Theory*. Giessen: Psychosozial-Verlag.

Winter, D. A. (1982). Construct relationships, psychological disorder and therapeutic change. *British Journal of Medical Psychology, 55* (Pt 3), 257-269.

Grice, J. W. (2008). Idiogrid: Idiographic Analysis with Repertory Grids (Version 2.4). Oklahoma: Oklahoma State University.

See Also

print.indexDilemma, plot.indexDilemma

Examples

```
id <- indexDilemma(boeker, self = 1, ideal = 2)
id

# adjust minimal correlation
indexDilemma(boeker, self = 1, ideal = 2, r.min = .5)

# adjust congruence and discrepance ranges
indexDilemma(boeker, self = 1, ideal = 2, diff.congruent = 0, diff.discrepant = 4)

# print options (see ?print.indexDilemma for help)
print(id, output = "D")   # dilemmas only
print(id, output = "OD")  # overview and dilemmas

# plot dilemmas as network graph (see ?plot.indexDilemma for help)
# set a seed for reproducibility
plot(id, layout = "rows")
plot(id, layout = "circle")
plot(id, layout = "star")
```

`indexDilemmatic` *Dilemmatic constructs*

Description

A Dilemmatic Construct (DC) is one where the ideal element is rated on the scale midpoint. This means, the person cannot decide which of the poles is preferable. Such constructs are called "dilemmatic". For example, on a rating scale from 1 to 7, a rating of 4 on the ideal element means that the construct is dilemmatic. By definition, DCs can only emerge in scales with an uneven number of rating options, i.e. 5-point scale, 7-point scale etc. However, the function makes it possible to allow for a deviation from the midpoint, to still count as dilemmatic. This is useful if the grid uses a large rating scale, e.g. from 0 to 100 or a visual analog scale, as some grid administration programs do. In this case you may want to set ratings, for example, between 45 and 55 as close enough to the midpoint to indicate that both poles are equally desirable.

Usage

```
indexDilemmatic(x, ideal, deviation = 0, warn = TRUE)
```

Arguments

`x`	A `repgrid` object.
`ideal`	Index of ideal element.
`deviation`	The maximal deviation from the scale midpoint for an ideal rating to be considered dilemmatic (default = `0`). For scales larger than a 17-point rating scale a warning is raised, if deviation is `0` (see details).
`warn`	Show warnings?

Value

List of class `indexDilemmatic`:

- `ideal`: Name of the ideal element.
- `n_constructs` Number of grid's constructs.
- `scale`: Minimum and maximum of grid rating scale.
- `midpoint`: Midpoint of rating scale.
- `lower,upper`: Lower and upper value to for a rating to be considered in the midpoint range.
- `midpoint_range`: Midpoint range as interval.
- `n_dilemmatic`: Number of dilemmatic constructs.
- `perc_dilemmatic`: Percentage of constructs which are dilemmatic.
- `i_dilemmatic`: Index of dilemmatic constructs.
- `dilemmatic_constructs`: Labels of dilemmatic constructs.
- `summary`: Summary dataframe.

Examples

```
dc <- indexDilemmatic(feixas2004, ideal = 13)
dc

# control the output
print(dc, output = "S")  # Summary
print(dc, output = "D")  # Details
```

---

indexIntensity *Intensity index*

Description

Calculate intensity index.

Usage

```
indexIntensity(x, rc = FALSE, trim = 30)
```

Arguments

x	repgrid object.
rc	Whether to use Cohen's rc for the calculation of inter-element correlations. See elementCor for further explanations of this measure.
trim	The number of characters a construct is trimmed to (default is 30). If NA no trimming occurs. Trimming simply saves space when displaying correlation of constructs or elements with long names.

Details

The Intensity index has been suggested by Bannister (1960) as a measure of the amount of construct linkage. Bannister suggested that the score reflects the degree of organization of the construct system under investigation (Bannister & Mair, 1968). The index resulted from his and his colleagues work on construction systems of patient suffering schizophrenic thought disorder. The concept of intensity has a theoretical connection to the notion of "tight" and "loose" construing as proposed by Kelly (1991). While tight constructs lead to unvarying prediction, loose constructs allow for varying predictions. Bannister hypothesized that schizophrenic thought disorder is liked to a process of extremely loose construing leading to a loss of predictive power of the subject's construct system. The Intensity score as a structural measure is thought to reflect this type of system disintegration (Bannister, 1960).

Implementation as in the Gridcor program and explained on the correspoding help pages: "...the sum of the squared values of the correlations of each construct with the rest of the constructs, averaged by the total number of constructs minus one. This process is repeated with each element, and the overall Intensity is calculated by averaging the intensity scores of constructs and elements." https://www.ub.edu/terdep/pag/man11.html. Currently the total is calculated as the unweighted average of all single scores (for elements and construct).

Value

An object of class `indexIntensity` containing a list with the following elements:

`c.int`	Intensity scores by construct.
`e.int`	Intensity scores by element.
`c.int.mean`	Average intensity score for constructs.
`e.int.mean`	Average intensity score for elements.
`total.int`	Total intensity score.

Development

TODO: Results have not been tested against other programs' results.

References

Bannister, D. (1960). Conceptual structure in thought-disordered schizophrenics. *The Journal of mental science*, 106, 1230-49.

Examples

```
indexIntensity(bell2010)
indexIntensity(bell2010, trim = NA)

# using Cohen's rc for element correlations
indexIntensity(bell2010, rc = TRUE)

# save output
x <- indexIntensity(bell2010)
x

# printing options
print(x, digits=4)

# accessing the objects' content
x$c.int
x$e.int
x$c.int.mean
x$e.int.mean
x$total.int
```

`indexPolarization` *Polarization (percentage of extreme ratings)*

Description

Polarization is the percentage of extreme ratings, e.g. the values 1 and 7 for a grid with a 7-point ratings scale.

Usage

```
indexPolarization(x, deviation = 0)
```

Arguments

`x`	A `repgrid` object.
`deviation`	The maximal deviation from the end of the rating scale for values to be considered an 'extreme' rating. By default only values that lie directly on ends of the ratings scales are considered 'extreme' (default = `0`).

Value

List of class `indexPolarization`:

- `scale`: Minimum and maximum of grid rating scale.
- `lower,upper` Lower and upper value to decide which ratings are considered extreme.
- `polarization_total`: Grid's overall polarization.
- `polarization_constructs`: Polarization per construct.
- `polarization_elements`: Polarization per element.

Examples

```
p <- indexPolarization(boeker)
p

# control the output
print(p, output = "T")  # total polarization
print(p, output = "C")  # construct polarization
print(p, output = "E")  # element polarization
```

`indexPvaff` *Percentage of Variance Accounted for by the First Factor (PVAFF)*

Description

The PVAFF is used as a measure of cognitive complexity. It was introduced in an unpublished PhD thesis by Jones (1954, cit. Bonarius, 1965). To calculate it, the 'first factor' two different methods may be used. One applies principal component analysis (PCA) to the construct centered raw data (default), the second applies SVD to the construct correlation matrix. The PVAFF reflects the amount of variation that is accounted for by a single linear component. If a single latent component is able to explain the variation in the grid, the cognitive complexity is said to be low. In this case the construct system is regarded as 'simple' (Bell, 2003).

Usage

```
indexPvaff(x, method = 1)
```

Arguments

`x`	`repgrid` object.
`method`	Method to compute PVAFF: `1` = PCA is applied to raw data with centered constructs (default), `2` = SVD of construct correlation matrix.

References

Bell, R. C. (2003). An evaluation of indices used to represent construct structure. In G. Chiari & M. L. Nuzzo (Eds.), *Psychological Constructivism and the Social World* (pp. 297-305). Milan: FrancoAngeli.

Bonarius, J. C. J. (1965). Research in the personal construct theory of George A. Kelly: role construct repertory test and basic theory. In B. A. Maher (Ed.), *Progress in experimental personality research* (Vol. 2). New York: Academic Press.

James, R. E. (1954). *Identification in terms of personal constructs* (Unpublished doctoral thesis). Ohio State University, Columbus, OH.

Examples

```
indexPvaff(bell2010)
```

`indexSelfConstruction` *Self construction profile*

Description

TBD

Usage

```
indexSelfConstruction(
  x,
  self,
  ideal,
  others = c(-self, -ideal),
  method = "euclidean",
  p = 2,
  normalize = TRUE,
  round = FALSE
)
```

Arguments

x	A repgrid object.
self	Numeric. Index of self element.
ideal	Numeric. Index of ideal element.
others	Numeric. Index(es) of self related "other" elements (e.g. father, friend).
method	The distance or correlation measure: • Distances: euclidean, manhattan, maximum, canberra, binary, minkowski • Correlations: pearson, kendall, spearman
p	The power of the Minkowski distance, in case minkowski is used as argument for method, otherwise it is ignored.
normalize	Normalize values?
round	Round average rating scores for 'others' to closest integer?

Value

List object of class indexSelfConstruction, containing the results from the calculations:

- grid: Reduced grid with self, ideal and others
- method_type: method type (correlation or distance)
- method: correlation or distance method used
- self_element: name of the self element
- ideal_element: name of the ideal element

- `other_elements`: name(s) of other elements
- `self_ideal`: measure between self and ideal
- `self_others`: measure between self and others
- `ideal_others`: measure betwen ideal and others

Author(s)

Mark Heckmann, José Antonio González Del Puerto

References

TBD

Examples

```
# using distance measures
indexSelfConstruction(boeker, 1, 2, c(3:11), method = "euclidean")
indexSelfConstruction(boeker, 1, 2, c(3:11), method = "manhattan")
indexSelfConstruction(boeker, 1, 2, c(3:11), method = "minkowski", p = 3)

# using correlation measures
indexSelfConstruction(boeker, 1, 2, c(3:11), method = "pearson")
indexSelfConstruction(boeker, 1, 2, c(3:11), method = "spearman")

# using not-normalized distances
indexSelfConstruction(boeker, 1, 2, c(3:11), method = "euclidean", normalize = FALSE)

# printing the results (biplot only works with)
cp <- indexSelfConstruction(boeker, 1, 2, c(3:11))
cp$grid  # grid with self, ideal and others
biplot2d(cp$grid, center = 4)  # midopoint centering
```

`indexVariability` *Calculate 'variability' of a grid as defined by Slater (1977).*

Description

Variability records a tendency for the responses to gravitate towards both end of the gradings scale. (Slater, 1977, p.88).

Usage

```
indexVariability(x, min = NULL, max = NULL, digits = 2)
```

Arguments

`x`	`repgrid` object.
`min, max`	Minimum and maximum grid scale values. Nor needed if they are set for the grid.
`digits`	Numeric. Number of digits to round to (default is 2).

Value

Numeric.

Note

STATUS: working and checked against example in Slater, 1977 , p.88.

References

Slater, P. (1977). *The measurement of intrapersonal space by Grid technique*. London: Wiley.

See Also

`indexBias`

Examples

```
indexVariability(boeker)
```

`is.repgrid` *Test if object has class repgrid*

Description

Test if object has class repgrid

Usage

```
is.repgrid(x)
```

Arguments

`x`	Any object.

left *Move construct or element in grid to the left, right, up or down.*

Description

Move element in grid to the right.

Usage

```
left(x, pos = 0)

right(x, pos = 0)

up(x, pos = 0)

down(x, pos = 0)
```

Arguments

x	`repgrid` object.
pos	Row (column) number of construct (element) to be moved leftwards, rightwards, upwards or downwards. The default is `0`. For indexes outside the range of the grid no moving is done.

Value

`repgrid` object.

`repgrid` object

`repgrid` object

`repgrid` object

Examples

```
## Not run:
   x <- randomGrid()
   left(x, 2)    # 2nd element to the left
   right(x, 1)   # 1st element to the right
   up(x, 2)      # 2nd construct upwards
   down(x, 1)    # 1st construct downwards

## End(Not run)
```

makeRepgrid *Make a new repgrid object.*

Description

The function creates a repgrid object from scratch. A number of parameters have to be defined in order to make a new grid (see parameters).

Usage

```
makeRepgrid(args)
```

Arguments

args	Arguments needed for the construction of the grid (list). These include name followed by a vector containing the element names. l.name followed by a vector with the left construct poles. r.name followed by a vector with the right construct poles. scores followed by a vector containing the rating scores row wise.

Value

NULL

Examples

```
## Not run:

   # make list object containing the arguments
   args <- list( name=c("element_1", "element_2", "element_3", "element_4"),
              l.name=c("left_1", "left_2", "left_3"),
             r.name=c("right_1", "right_2", "right_3"),
             scores=c( 1,0,1,0,
                1,1,1,0,
                1,0,1,0 ) )
   # make grid object
   x <- makeRepgrid(args)
   x

## End(Not run)
```

`matches` *Number of matches in ratings*

Description

Count the number of matches, i.e. (near) identical ratings between two elements or constructs. Matches are used as the basis for the calculation of grid indexes.

Usage

```
matches(x, deviation = 0, diag.na = TRUE)
```

Arguments

`x`	A `repgrid` object.
`deviation`	Maximal difference between ratings to be considered a match (default `0` = only identical rating scores are a match). Especially useful for long rating scale (e.g. 0 to 100).
`diag.na`	Whether to set the diagonal of the matrices to `NA` (default is `TRUE`).

Value

A list of class `org.matches` with:

- `grid`: The grid used to calculate the matches.
- `deviation` The deviation parameter.
- `max_constructs` Maximum possible number of matches across constructs.
- `max_elements` Maximum possible number of matches across elements.
- `total_constructs` Total number of matches across constructs.
- `total_elements` Total number of matches across elements.
- `constructs`: Matrix with no. of matches for constructs.
- `elements`: Matrix with no. of matches for elements.

Examples

```
m <- matches(boeker)

# several output options
print(m, index = FALSE, names = FALSE, upper = FALSE)
print(m, output = "C")  # construct matches
print(m, output = "E")  # element matches

# extract the matrices
m$constructs
m$elements
```

`midpoint` *Midpoint of the grid rating scale*

Description

Midpoint of the grid rating scale

Usage

```
midpoint(x)
```

Arguments

x	`repgrid` object.

Value

Midpoint of scale.

Examples

```
midpoint(bell2010)
```

`normalize` *Normalize rows or columns by its standard deviation.*

Description

Normalize rows or columns by its standard deviation.

Usage

```
normalize(x, normalize = 0, ...)
```

Arguments

x	`matrix`
normalize	A numeric value indicating along what direction (rows, columns) to normalize by standard deviations. `0 = none, 1= rows, 2 = columns` (default is `0`).
...	Not evaluated.

Value

Not yet defined TODO!

Examples

```
## Not run:

 x <- matrix(sample(1:5, 20, rep=T), 4)
 normalize(x, 1)       # normalizing rows
 normalize(x, 2)       # normalizing columns

## End(Not run)
```

OpenRepGrid   **OpenRepGrid***: an R package for the analysis of repertory grids.*

Description

The **OpenRepGrid** package provides tools for the analysis of repertory grid data. The repertory grid is a method devised by George Alexander Kelly in his seminal work "The Psychology of Personal Constructs" published in 1955. The repertory grid has been used in and outside the context of Personal Construct Psychology (PCP) in a broad range of fields. For an introduction into the technique see e.g. Fransella, Bell and Bannister (2003).

Note

To get started with **OpenRepGrid** visit the project's home under openrepgrid.org. On this site you will find tutorials, explanation about the theory, the analysis methods and the corresponding R code.

To see the preferable citation of the **OpenRepGrid** package, type `citation("OpenRepGrid")` into the R console.

Disclaimer: Note that the package is distributed under the GPL 2 license. It is work in progress and is continuously being improved by hopefully numerous contributors. It may contain bugs and errors. There is no warranty whatsoever for the program.

Author(s)

OpenRepGrid is maintained by Mark Heckmann. Other contributors: Alejandro García, Diego Vitali. Researchers interested in contributing to the package are welcome.

The **OpenRepGrid** code is hosted on GitHub. The GitHub site provides additional information and allows to file bug reports or feature requests. Bug reports can also be emailed to the package maintainer or issued on openrepgrid.org under section *Suggestions/Issues*. The package maintainer is Mark Heckmann <heckmann(dot)mark(at)gmail(dot)com>.

References

Fransella, F., Bell, R. C., & Bannister, D. (2003). *A Manual for Repertory Grid Technique (2. Ed.)*. Chichester: John Wiley & Sons.

Kelly, G. A. (1955). *The psychology of personal constructs. Vol. I, II.* New York: Norton, (2nd printing: 1991, Routledge, London, New York).

See Also

Useful links:

- https://github.com/markheckmann/OpenRepGrid

OpenRepGrid-overview **OpenRepGrid***: Annotated overview of package functions.*

Description

This documentation page contains an overview over the package functions ordered by topics. The best place to start learning OpenRepGrid will be the package website https://openrepgrid.org though.

Functions sorted by topic

Manipulating grids

left	Move construct(s) to the left
right	Move construct(s) to the right
up	Move construct(s) upwards
down	Move construct(s) downwards

Loading and saving data

importGridcor	Import GRIDCOR data files
importGridstat	Import Gridstat data files
importGridsuite	Import Gridsuite data files
importScivesco	Import sci:vesco data files
importTxt	Import grid data from a text file
saveAsTxt	Save grid in a text file (txt)

Analyzing constructs

Descriptive statistics of constructs Construct correlations distance Root mean square of inter-construct correlations Somers' D Principal component analysis (PCA) of construct correlation matrix Cluster analysis of constructs

Analyzing elements

Visual representation

Bertin plots

bertin	Make Bertin display of grid data
bertinCluster	Bertin display with corresponding cluster analysis

Biplots

biplot2d	Draw a two-dimensional biplot
biplotEsa2d	Plot an eigenstructure analysis (ESA) biplot in 2D
biplotSlater2d	Draws Slater's INGRID biplot in 2D
biplotPseudo3d	See 'biplotPseudo3d' for its use. Draws a biplot of the grid in 2D with depth impression (pseudo 3
biplotEsaPseudo3d	Plot an eigenstructure analysis (ESA) in 2D grid with 3D impression (pseudo 3D)
biplotSlaterPseudo3d	Draws Slater's biplot in 2D with depth impression (pseudo 3D)
biplot3d	Draw grid in rgl (3D device)
biplotEsa3d	Draw the eigenstructure analysis (ESA) biplot in rgl (3D device)
biplotSlater3d	Draw the Slater's INGRID biplot in rgl (3D device)
biplotSimple	A graphically unsophisticated version of a biplot

Index measures

indexConflict1	Conflict measure for grids (Slade & Sheehan, 1979) based on correlations
indexConflict2	Conflict measure for grids (Bassler et al., 1992) based on correlations
indexConflict3	Conflict or inconsistency measure for grids (Bell, 2004) based on distances
indexDilemma	Detect implicative dilemmas (conflicts)
indexIntensity	Intensity index
indexPvaff	Percentage of Variance Accounted for by the First Factor (PVAFF)
indexBias	Calculate 'bias' of grid as defined by Slater (1977)
indexVariability	Calculate 'variability' of a grid as defined by Slater (1977)

Special features

alignByIdeal	Align constructs using the ideal element to gain pole preferences
alignByLoadings	Align constructs by loadings on first principal component
reorder2d	Order grid by angles between construct and/or elements in 2D

Settings

OpenRepGrid uses several default settings e.g. to determine how many construct characters to display by default when displaying a grid. The function `settings` can be used to show and change these settings. Also it is possible to store the settings to a file and load the settings file to restore the settings.

`settings`	Show and modify global settings for OpenRepGrid
`settingsSave`	Save OpenRepGrid settings to file
`settingsLoad`	Load OpenRepGrid settings from file

Grid datasets

OpenRepGrid already contains some ready to use grid data sets. Most of the datasets are taken from the literature. To output the data simply type Type the name of the dataset to the console and press enter.

Single grids

`bell2010`	Grid data from a study by Haritos et al. (2004) on role titles; used for demonstration of construct
`bellmcgorry1992`	Grid from a psychotic patient used in Bell (1997, p. 6). Data originated from a study by Bell and
`boeker`	Grid from seventeen year old female schizophrenic patient undergoing last stage of psychoanalyt
`fbb2003`	Dataset used in *A manual for Repertory Grid Technique* (Fransella, Bell, & Bannister, 2003b, p. (
`feixas2004`	Grid from a 22 year old Spanish girl suffering self-worth problems (Feixas & Saul, 2004, p. 77).
`mackay1992`	Dataset *Grid C* used in Mackay's paper on inter-element correlation (1992, p. 65).
`leach2001a`, `leach2001b`	Pre- (a) and post-therapy (b) dataset from sexual child abuse survivor (Leach, Freshwater, Aldrid
`raeithel`	Grid data to demonstrate the use of Bertin diagrams (Raeithel, 1998, p. 223). The context of its a
`slater1977a`	Drug addict grid dataset from (Slater, 1977, p. 32).
`slater1977b`	Grid dataset (ranked) from a seventeen year old female psychiatric patient (Slater, 1977, p. 110) s

Multiple grids

NOT YET AVAILABLE

Functions for developers

OpenRepGrid: internal functions overview for developers.

Below you find a guide for developers: these functions are usually not needed by the casual user. The internal functions have a twofold goal 1) to provide means for advanced numerical grid analysis and 2) to facilitate function development. The function for these purposes are internal, i.e. they are not visible in the package documentation. Nonetheless they do have a documentation that can be accesses in the same way as for other functions. More in the details section.

Functions for advanced grid analysis

The package provides functions to facilitate numerical research for grids. These comprise the generation of random data, permutation of grids etc. to facilitate Monte Carlo simulations, batch analysis of grids and other methods. With R as an underlying framework, the results of grid analysis easily lend themselves to further statistical processing and analysis within R. This is one of the central advantages for researchers compared to other standard grid software. The following table lists several functions for these purposes.

randomGrid
randomGrids
permuteConstructs
permuteGrid
quasiDistributionDistanceSlater

Modules for function development

Beside the advanced analysis feature the developer's functions comprise low-level modules to create new functions for grid analysis. Though the internal structure of a repgrid object in R is simple (type e.g. `str(bell2010, 2)` to get an impression), it is convenient to not have to deal with access on this level. Several function like e.g. `getElementNames` are convenient wrappers that perform standard tasks needed when implementing new functions. The following table lists several functions for these purposes.

getRatingLayer	Retrieve grid scores from grid object.
getNoOfConstructs	Get the number of constructs in a grid object.
getNoOfElements	Get the number of elements in a grid object.
dim	Get grid dimensions, i.e. constructs x elements.
getScale	Get minimum and maximum scale value used in grid.
getScaleMidpoint	Get midpoint of the grid rating scale.
getConstructNames	Get construct names.
getConstructNames2	Get construct names (another newer version).
getElementNames	Retrieve element names of repgrid object.
bindConstructs	Concatenate the constructs of two grids.
doubleEntry	Join the constructs of a grid with the same reversed constructs.

Other internal functions

importTxtInternal

Author(s)

Current members of the **OpenRepGrid** development team: Mark Heckmann. Everyone who is interested in developing the package is invited to join.

The **OpenRepGrid** package development is hosted on github (https://github.com/markheckmann/OpenRepGrid). The github site provides information and allows to file bug reports or feature requests. Bug reports can also be emailed to the package maintainer or issued on https://openrepgrid.org under section *Suggestions/Issues*. The package maintainer is Mark Heckmann <heckmann(dot)mark(at)gmail(dot)com>.

`permuteConstructs` *Generate a list with all possible construct reflections of a grid.*

Description

Generate a list with all possible construct reflections of a grid.

Usage

```
permuteConstructs(x, progress = TRUE)
```

Arguments

`x`	`repgrid` object.
`progress`	Whether to show a progress bar (default is `TRUE`). This may be sensible for a larger number of elements.

Value

A list of `repgrid` objects with all possible permutations of the grid.

Examples

```
## Not run:

  l <- permuteConstructs(mackay1992)
  l


## End(Not run)
```

`perturbate` *Perturbate grid ratings*

Description

Randomly subtract or add an amount to a proportion of the grid ratings. This emulates randomness during the rating process, producing a grid which might also have resulted.

Usage

```
perturbate(x, prop = 0.1, amount = c(-1, 1), prob = c(0.5, 0.5))

grids_perturbate(x, n = 10, prop = 0.1, amount = c(-1, 1), prob = c(0.5, 0.5))
```

Arguments

`x`	A `repgrid` object.
`prop`	The proportion of ratings to be perturbated.
`amount`	The amount set of possible perturbations. Will depend on scale range. Usually -1, 1 are reasonable settings
`prob`	Probability for each amount to occur.
`n`	Number of perturbated grid to generate.

Examples

```
## All results for PVAFF index when ratings are slightly perturbated
p <- indexPvaff(boeker)
l <- grids_perturbate(boeker, n = 100, prop = .1)
pp <- sapply(l, indexPvaff)  # apply indexPvaff function to all perturbated grids
range(pp)   # min and max PVAFF
hist(pp, xlab = "PVAFF values")    # visualize
abline(v = p, col = "blue", lty = 2)
```

`randomGrid` *Generate a random grid (quasis) of prompted size.*

Description

This feature is useful for research purposes like exploring distributions of indexes etc.

Usage

```
randomGrid(
  nc = 10,
  ne = 15,
  nwc = 8,
  nwe = 5,
  range = c(1, 5),
  prob = NULL,
  options = 1
)
```

Arguments

nc	Number of constructs (default 10).
ne	Number of elements (default 15).
nwc	Number of random words per construct.
nwe	Number of random words per element.
range	Minimal and maximal scale value (default c(1, 5)).
prob	The probability of each rating value to occur. If NULL (default) the distribution is uniform.
options	Use random sentences as constructs and elements (1) or not (0). If not, the elements and constructs are given default names and are numbered.

Value

repgrid object.

Examples

```
## Not run:

    x <- randomGrid()
    x
    x <- randomGrid(10, 25)
    x
    x <- randomGrid(10, 25, options=0)
    x

## End(Not run)
```

---

randomGrids *Generate a list of random grids (quasis) of prompted size.*

Description

This feature is useful for research purposes like exploring distributions of indexes etc. The function is a simple wrapper around randomGrid.

Usage

```
randomGrids(
  rep = 3,
  nc = 10,
  ne = 15,
  nwc = 8,
  nwe = 5,
  range = c(1, 5),
  prob = NULL,
  options = 1
)
```

Arguments

rep	Number of grids to be produced (default is 3).
nc	Number of constructs (default 10).
ne	Number of elements (default 15).
nwc	Number of random words per construct.
nwe	Number of random words per element.
range	Minimal and maximal scale value (default c(1, 5)).
prob	The probability of each rating value to occur. If NULL (default) the distribution is uniform.
options	Use random sentences as constructs and elements (1) or not (0). If not, the elements and constructs are given default names and are numbered.

Value

A list of repgrid objects.

Examples

```
## Not run:

    x <- randomGrids()
    x
    x <- randomGrids(5, 3, 3)
    x
```

```
    x <- randomGrids(5, 3, 3, options=0)
    x

## End(Not run)
```

---

ratings *Extract ratings (wide or long format)*

Description

Extract ratings (wide or long format)

Usage

```
ratings(x, names = TRUE, trim = 10)

ratings_df(x, long = FALSE, names = TRUE, trim = NA)

ratings(x, i, j) <- value
```

Arguments

x	A `repgrid` object.
names	Extract row and columns names (constructs and elements).
trim	The number of characters a row or column name is trimmed to (default is `10`). If `NA` no trimming is done. Trimming simply saves space when displaying the output.
long	Return as long format? (default `FALSE`)
i, j	Row and column indices.
value	Numeric replacement value(s).

Value

A `matrix`.#'

See Also

`` `[<--method` ``

Examples

```
## store Bell's dataset in x
x <- bell2010

## get ratings
ratings(x)


## replace ratings

ratings(x)[1,1] <- 1
# noet that this is even simpler using the repgrid object directly
x[1,1] <- 2

#replace several values

ratings(x)[1,1:5] <- 1
x[1,1:5] <- 2   # the same

ratings(x)[1:3,5:6] <- matrix(5, 3, 2)
x[1:3,5:6] <- matrix(5, 3, 2)   # the same


## ratings as dataframe in wide or long format

ratings_df(x)
ratings_df(x, long = TRUE)
```

---

reorder.repgrid *Invert construct and element order*

Description

Invert construct and element order

Usage

```
## S3 method for class 'repgrid'
reorder(x, what = "CE", ...)
```

Arguments

x	A repgrid object.
what	A string or numeric to indicate if constructs ("C", 1) or elements ("C", 1), or both ("CE", 12) should be reversed.
...	Ignored.

Examples

```
# invert order of constructs
reorder(boeker, "C")
reorder(boeker, 1)

# invert order of elements
reorder(boeker, "E")
reorder(boeker, 2)

# invert both (default)
reorder(boeker)
reorder(boeker, "CE")
reorder(boeker, 12)

# not reordering
reorder(boeker, NA)
```

---

reorder2d — *Order grid by angles between construct and/or elements in 2D.*

Description

The approach is to reorder the grid matrix by their polar angles on the first two principal components from a data reduction technique (here the biplot, i.e. SVD). The function `reorder2d` reorders the grid according to the angles between the x-axis and the element (construct) vectors derived from a 2D biplot solution. This approach is apt to identify circumplex structures in data indicated by the diagonal stripe in the display (see examples).

Usage

```
reorder2d(
  x,
  dim = c(1, 2),
  center = 1,
  normalize = 0,
  g = 0,
  h = 1 - g,
  rc = TRUE,
  re = TRUE,
  ...
)
```

Arguments

`x`	`repgrid` object.
`dim`	Dimension of 2D solution used to calculate angles (default `c(1,2)`).

`center`	Numeric. The type of centering to be performed. 0= no centering, 1= row mean centering (construct), 2= column mean centering (elements), 3= double-centering (construct and element means), 4= midpoint centering of rows (constructs). The default is `1` (row centering).
`normalize`	A numeric value indicating along what direction (rows, columns) to normalize by standard deviations. `0 = none, 1= rows, 2 = columns` (default is `0`).
`g`	Power of the singular value matrix assigned to the left singular vectors, i.e. the constructs.
`h`	Power of the singular value matrix assigned to the right singular vectors, i.e. the elements.
`rc`	Logical. Reorder constructs by similarity (default `TRUE`).
`re`	Logical. Reorder elements by similarity (default `TRUE`).
`...`	Not evaluated.

Value

Reordered `repgrid` object.

Examples

```
## Not run:

   x <- feixas2004
   reorder2d(x)           # reorder grid by angles in first two dimensions
   reorder2d(x, rc=F)     # reorder elements only
   reorder2d(x, re=F)     # reorder constructs only

## End(Not run)
```

`saveAsExcel` *Save grid in a Microsoft Excel file (.xlsx)*

Description

`saveAsExcel` will save the grid as a Microsoft Excel file (`.xlsx`).

Usage

```
saveAsExcel(x, file, sheet = 1)
```

Arguments

`x`	A `repgrid` object.
`file`	Filename to save the grid to. The name should have the suffix `.xlsx`.
`sheet`	Index of the sheet to write to.

Value

Invisibly returns the name of the file.

See Also

importExcel

Examples

```
## Not run:

 x <- randomGrid(options=0)
 saveAsExcel(x, "grid.xlsx")


## End(Not run)
```

---

saveAsTxt *Save grid in a text file (txt).*

Description

saveAsTxt will save the grid as a .txt file in format used by **OpenRepGrid**. This file format can also easily be edited by hand (see importTxt for a description).

Usage

```
saveAsTxt(x, file = NA)
```

Arguments

x	repgrid object.
file	Filename to save the grid to. The name should have the suffix .txt.

Value

Invisibly returns the name of the file.

Note

Structure of a txt file that can be read by importTxt.

---------------- .txt file -----------------

anything not contained within the tags will be discarded

ELEMENTS
element 1

```
element 2
element 3
END ELEMENTS

CONSTRUCTS
left pole 1 : right pole 1
left pole 2 : right pole 2
left pole 3 : right pole 3
left pole 4 : right pole 4
END CONSTRUCTS

RATINGS
1 3 2
4 1 1
1 4 4
3 1 1
END RATINGS

RANGE
1 4
END RANGE
```

```
---------------- end of file ----------------
```

See Also

importTxt

Examples

```
## Not run:

 x <- randomGrid()
 saveAsTxt(x, "random.txt")


## End(Not run)
```

---

setConstructAttr *Set the attributes of a construct*

Description

Set the attributes of a construct i.e. name, abbreviation, status etc.

Usage

```
setConstructAttr(
  x,
  pos,
  l.name,
  r.name,
  l.preferred,
  r.preferred,
  l.emerged,
  r.emerged
)
```

Arguments

x	repgrid object.
pos	Row number of construct in the grid to be changed
l.name	Name of the left pole (string) (optional).
r.name	Name of the right pole (string) (optional).
l.preferred	Logical. Is the left one the preferred pole? (optional).
r.preferred	Logical. Is the right one the preferred pole? (optional).
l.emerged	Logical. Is the left one the emergent pole? (optional).
r.emerged	Logical. Is the right one the emergent pole? (optional).

Value

repgrid object

See Also

setElementAttr

Examples

```
## Not run:

   x <- setConstructAttr(bell2010, 1,
                 "new left pole", "new right pole")
   x

## End(Not run)
```

`setElementAttr` *Set the attributes of an element*

Description

Set the attributes of an element i.e. name, abbreviation, status etc.

Usage

```
setElementAttr(x, pos, name, abb, status)
```

Arguments

x	`repgrid` object.
pos	Column number of element in the grid whose attributes are changed.
name	New element name (optional).
abb	Abbreviation of element name (optional).
status	Status of element (e.g. ideal etc.) (optional).

Value

`repgrid` object

Note

Currently the main purpose is to change element names. Future implementations will allow to set further attributes.

See Also

`setConstructAttr`

Examples

```
## Not run: 

   x <- setElementAttr(boeker, 1, "new name")   # change name of first element
   x

## End(Not run)
```

setScale *Set the scale range of a grid.*

Description

The scale must be known for certain operations, e.g. to swap the construct poles. If the user construes a grid he should make sure that the scale range is set correctly.

Usage

```
setScale(x, min, max, step, ...)
```

Arguments

`x`	`repgrid` object.
`min`	Minimal possible scale value for ratings.
`max`	Maximal possible scale value for ratings.
`step`	Steps the scales uses (not yet in use).
`...`	Not evaluated.

Value

`repgrid` object

Examples

```
## Not run:

   x <- bell2010
   x <- setScale(x, 0, 8)   # not set correctly
   x
   x <- setScale(x, 1, 7)   # set correctly
   x

## End(Not run)
```

`settings` *global settings for OpenRepGrid*

Description

global settings for OpenRepGrid

Usage

```
settings(...)
```

Arguments

`...`	Use parameter value pairs (`par1=val1, par2=val2`) to change a parameter. Use parameter names to request parameter's value (`"par1", "par2"`).

Note

Currently the following parameters can be changed, ordered by topic. The default value is shown in the brackets at the end of a line.

Printing grid to the console

- `show.scale` Show grid scale info? (`TRUE`)
- `show.meta` Show grid meta data? (`TRUE`)
- `show.trim` Number of chars to trim strings to (`30`)
- `show.cut` Maximum number of characters printed on the sides of a grid (`20`)
- `c.no` Print construct ID number? (`TRUE`)
- `e.no` Print element ID number? (`TRUE`)

Examples

```
## Not run:
# get current settings
settings()

# get some parameters
settings("show.scale", "show.meta")

# change parameters
bell2010

settings(show.meta=F)
bell2010

settings(show.scale=F, show.cut=30)
bell2010

## End(Not run)
```

`settingsLoad` *Load OpenRepGrid settings*

Description

OpenRepGrid settings saved in an a settings file with the extension `.orgset` can be loaded to restore the settings.

Usage

```
settingsLoad(file)
```

Arguments

`file`	Path of the file to be loaded.

`settingsSave` *Save OpenRepGrid settings*

Description

The current settings of OpenRepGrid can be saved into a file with the extension `.orgset`.

Usage

```
settingsSave(file)
```

Arguments

`file`	Path of the file to be saved to.

`shift` *Shift construct or element to first position.*

Description

Shifts the whole grid vertically or horizontally so that the order remains the same but the prompted element or construct appears in first position.

Usage

```
shift(x, c = 1, e = 1)
```

Arguments

x	`repgrid` object.
c	Index of construct to be shifted to first position.
e	Index of element to be shifted to first position.

Value

`repgrid` object.

Examples

```
## Not run:

   # shift element 13: 'Ideal self' to first position
   shift(feixas2004, 13)

   x <- randomGrid(5,10)
   shift(x, 3, 5)

## End(Not run)
```

`show,repgrid-method` *Show method for repgrid*

Description

Show method for repgrid

Usage

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

Arguments

object	A `repgrid` object.

`statsElements` *Descriptive statistics for constructs and elements of a grid.*

Description

Descriptive statistics for constructs and elements of a grid.

Descriptive statistics for constructs and elements of a grid.

Usage

```
statsElements(x, index = TRUE, trim = 20)

statsConstructs(x, index = T, trim = 20)
```

Arguments

`x`	`repgrid` object.
`index`	Whether to print the number of the element.
`trim`	The number of characters an element or a construct is trimmed to (default is `20`). If `NA` no trimming occurs. Trimming simply saves space when displaying correlation of constructs or elements with long names.

Value

A dataframe containing the following measures is returned invisibly (see `describe`):
item name
item number
number of valid cases
mean standard deviation
trimmed mean (with trim defaulting to .1)
median (standard or interpolated)
mad: median absolute deviation (from the median)
minimum
maximum
skew
kurtosis
standard error

Note

Note that standard deviation and variance are estimations, i.e. including Bessel's correction. For more info type `?describe`.

Note that standard deviation and variance are estimated ones, i.e. including Bessel's correction. For more info type `?describe`.

Examples

```
## Not run:

   statsConstructs(fbb2003)
   statsConstructs(fbb2003, trim=10)
   statsConstructs(fbb2003, trim=10, index=F)

   statsElements(fbb2003)
   statsElements(fbb2003, trim=10)
   statsElements(fbb2003, trim=10, index=F)

   # save the access the results
   d <- statsElements(fbb2003)
   d
   d["mean"]
   d[2, "mean"]  # mean rating of 2nd element

   d <- statsConstructs(fbb2003)
   d
   d["sd"]
   d[1, "sd"]   # sd of ratings on first construct

## End(Not run)
```

---

swapConstructs *Swap the position of two constructs in a grid.*

Description

Swap the position of two constructs in a grid.

Usage

```
swapConstructs(x, pos1 = 1, pos2 = 1)
```

Arguments

x	repgrid object.
pos1	Row number of first construct to be swapped (default=1).
pos2	Row number of second construct to be swapped (default=1).

Value

repgrid object

Examples

```
## Not run:

   x <- randomGrid()
   swapConstructs(x, 1, 3)       # swap constructs 1 and 3
   swapConstructs(x, 1:2, 3:4)   # swap construct 1 with 3 and 2 with 4

## End(Not run)
```

---

swapElements — *Swap the position of two elements in a grid.*

Description

Swap the position of two elements in a grid.

Usage

```
swapElements(x, pos1 = 1, pos2 = 1)
```

Arguments

`x`	`repgrid` object.
`pos1`	Column number of first element to be swapped (default=1).
`pos2`	Column number of second element to be swapped (default=1).

Value

`repgrid` object.

Examples

```
## Not run:
   x <- randomGrid()
   swapElements(x, 1, 3)       # swap elements 1 and 3
   swapElements(x, 1:2, 3:4)   # swap element 1 with 3 and 2 with 4

## End(Not run)
```

`swapPoles` *Reverse constructs / swaps construct poles*

Description

Constructs are bipolar. They can be reversed without a change in meaning. The function swaps the constructs poles and re-adjusts ratings accordingly, i.e. it reversed selected constructs.

Usage

```
swapPoles(x, pos)

reverse(x, pos = 1L:nrow(x))
```

Arguments

`x`	A `repgrid` object.
`pos`	Row indexes of constructs to reverse..

Value

A `repgrid` object with reversed constructs.

Note

Please note that the scale of the rating grid has to be set in order to reverse constructs. If the scale is unknown no reversal occurs and an error is raised.

Examples

```
x <- boeker

reverse(x)        # reverse all constructs
reverse(x, 1)     # reverse construct 1
reverse(x, 1:2)   # reverse constructs 1 and 2

# swapPoles will become deprecated, use reverse instead
swapPoles(x, 1)     # swap construct poles of construct
```

[,repgrid-method *Extract parts of the repgrid object.*

Description

Methods for "[", i.e., subsetting of repgrid objects.

Usage

```
## S4 method for signature 'repgrid'
x[i, j, ..., drop = TRUE]
```

Arguments

x	A repgrid object.
i, j	Row and column indices.
...	Not evaluated.
drop	Not used.

Examples

```
x <- randomGrid()
x[1:4, ]
x[ , 1:3]
x[1:4,1:3]
x[1,1]
```

[<-,repgrid-method *Method for "<-" assignment of the repgrid ratings.*

Description

It should be possible to use it for ratings on all layers.

Usage

```
## S4 replacement method for signature 'repgrid'
x[i, j, ...] <- value
```

Arguments

x	A repgrid object.
i, j	Row and column indices.
...	Not evaluated.
value	Numeric replacement value(s).

Examples

```
## Not run:
x <- randomGrid()
x[1,1] <- 2
x[1, ] <- 4
x[ ,2] <- 3

# settings values outside defined rating scale
# range throws an error
x[1,1] <- 999

# removing scale range allows arbitary values to be set
x <- setScale(x, min = NA, max=NA)
x[1,1] <- 999

## End(Not run)
```

Index

∗ **data**
data-bell2010, 48
data-bellmcgorry1992, 48
data-boeker, 49
data-fbb2003, 49
data-feixas2004, 49
data-leach2001, 50
data-mackay1992, 50
data-raeithel, 51
data-slater1977a, 51
data-slater1977b, 51
∗ **package**
OpenRepGrid, 99
OpenRepGrid-overview, 100
∗ **repgrid**
OpenRepGrid, 99
+,list,repgrid-method (+,repgrid,repgrid-method), 4
+,repgrid,list-method (+,repgrid,repgrid-method), 4
+,repgrid,repgrid-method, 4
[,repgrid-method, 124
[<-,repgrid-method, 124

addAvgElement, 5
addConstruct, 5, *7*
addElement, *6*, 7
alignByIdeal, 8, *10*, *101*
alignByLoadings, *9*, 9, *101*
as.gridlist (gridlist), 63

bell2010, *102*
bell2010 (data-bell2010), 48
bellmcgorry1992, *102*
bellmcgorry1992 (data-bellmcgorry1992), 48
bertin, 11, *14*, *101*
bertinCluster, 13, *38*, *101*
bindConstructs, 15, *103*
biplot2d, 17, *21*, *24–28*, *30–35*, *101*
biplot3d, *21*, 22, *24*, *26–28*, *30*, *32–35*, *65*, *101*
biplotEsa2d, *21*, *24*, 25, *26–28*, *30*, *32–35*, *101*
biplotEsa3d, *21*, *24*, *26*, 26, *27, 28*, *30*, *32–35*, *65*, *101*
biplotEsaPseudo3d, *21*, *24*, *26, 27*, 27, *28*, *30*, *32–35*, *101*
biplotPseudo3d, *21*, *24*, *26–28*, 28, *30*, *32–35*, *101*
biplotSimple, *21*, *24*, *26–28*, *30*, 31, *32–35*, *101*
biplotSlater2d, *21*, *24*, *26–28*, *30*, *32, 33*, 33, *34, 35*, *101*
biplotSlater3d, *21*, *24*, *26–28*, *30*, *32–34*, 34, *35*, *65*, *101*
biplotSlaterPseudo3d, *21*, *24*, *26–28*, *30*, *32–35*, 35, *101*
boeker, *102*
boeker (data-boeker), 49

center, 36
cluster, *14*, 37, *39*
clusterBoot, 39
constructCor, 41, *46*, *61*
constructD, 42
constructPca, 43, *45*
constructPcaLoadings, *44*, 45
constructRmsCor, 45, *62*
constructs, 46
constructs<- (constructs), 46

data-bell2010, 48
data-bellmcgorry1992, 48
data-boeker, 49
data-fbb2003, 49
data-feixas2004, 49
data-leach2001, 50
data-mackay1992, 50
data-raeithel, 51

data-slater1977a, 51
data-slater1977b, 51
describe, *120*
dim, *103*
distance, 52, *58*
distanceHartmann, 53, *57*, *59*
distanceNormalized, 56
distanceSlater, *54, 55*, *57*, 58
doubleEntry, *103*
down, *100*
down *(*left*)*, 95

elementCor, *41*, 60, *62*, *88*
elementRmsCor, *46*, 61
elements, 62
elements<- *(*elements*)*, 62

fbb2003, *102*
fbb2003 *(*data-fbb2003*)*, 49
feixas2004, *102*
feixas2004 *(*data-feixas2004*)*, 49

getConstructNames, *103*
getConstructNames2, *103*
getElementNames, *103*
getNoOfConstructs, *103*
getNoOfElements, *103*
getRatingLayer, *103*
getScale, *103*
getScaleMidpoint, *103*
gridlist, 63
grids_bootstrap *(*grids_leave_n_out*)*, 64
grids_leave_n_out, 64
grids_perturbate *(*perturbate*)*, 105

hclust, *14*
home, *21*, *24*, *26–28*, *30*, *32–35*, 65

importExcel, 66, *68, 69*, *71, 72*, *74*, *112*
importGridcor, *67*, 67, *68, 69*, *71, 72*, *74*, *100*
importGridstat, *67, 68*, 68, *69*, *71, 72*, *74*, *100*
importGridsuite, *67–69*, 70, *71, 72*, *74*, *100*
importScivesco, *67–69*, *71*, 71, *72*, *74*, *100*
importTxt, *67–69*, *71, 72*, 73, *74*, *100*, *112, 113*
importTxtInternal, *103*
indexBias, 75, *94*, *101*
indexBieri, 76
indexConflict1, 77, *79*, *81, 82*, *101*
indexConflict2, *78*, 78, *81, 82*, *101*
indexConflict3, *78, 79*, 80, *101*
indexDilemma, 82, *101*
indexDilemmatic, 87
indexIntensity, 88, *101*
indexPolarization, 90
indexPvaff, 91, *101*
indexSelfConstruction, 92
indexVariability, *75*, 93, *101*
is.gridlist *(*gridlist*)*, 63
is.repgrid, 94

leach2001a, *102*
leach2001a *(*data-leach2001*)*, 50
leach2001b, *102*
leach2001b *(*data-leach2001*)*, 50
left, 95, *100*
leftpoles *(*constructs*)*, 46
leftpoles<- *(*constructs*)*, 46

mackay1992, *102*
mackay1992 *(*data-mackay1992*)*, 50
makeRepgrid, 96
matches, 97
midpoint, 98

normalize, 98

OpenRepGrid, 99
OpenRepGrid-overview, 100
OpenRepGrid-package *(*OpenRepGrid*)*, 99

permuteConstructs, *103*, 104
permuteGrid, *103*
perturbate, 105
plot.indexDilemma, *86*
print.indexConflict3, *81*
print.indexDilemma, *86*
pvclust, *39, 40*

quasiDistributionDistanceSlater, *103*

raeithel, *102*
raeithel *(*data-raeithel*)*, 51
randomGrid, *103*, 105, *107*
randomGrids, *103*, 107
ratings, 108
ratings<- *(*ratings*)*, 108
ratings_df *(*ratings*)*, 108

reorder.repgrid, 109
reorder2d, *101*, 110
reverse *(*swapPoles*)*, 123
right, *100*
right *(*left*)*, 95
rightpoles *(*constructs*)*, 46
rightpoles<- *(*constructs*)*, 46

saveAsExcel, 111
saveAsTxt, *100*, 112
set.seed, *39*
setConstructAttr, 113, *115*
setElementAttr, *114*, 115
setScale, 116
settings, *102*, 117
settingsLoad, *102*, 118
settingsSave, *102*, 118
shift, 118
show,repgrid-method, 119
slater1977a, *102*
slater1977a *(*data-slater1977a*)*, 51
slater1977b, *102*
slater1977b *(*data-slater1977b*)*, 51
statsConstructs *(*statsElements*)*, 120
statsElements, 120
swapConstructs, 121
swapElements, 122
swapPoles, *8*, 123

up, *100*
up *(*left*)*, 95