

Package ‘seriation’

May 9, 2017

Type Package

Title Infrastructure for Ordering Objects Using Seriation

Version 1.2-2

Date 2017-05-08

Description Infrastructure for seriation with an implementation of several seriation/sequencing techniques to reorder matrices, dissimilarity matrices, and dendrograms. Also provides (optimally) reordered heatmaps, color images and clustering visualizations like dissimilarity plots, and visual assessment of cluster tendency plots (VAT and iVAT).

Classification/ACM G.1.6, G.2.1, G.4

URL <http://lyle.smu.edu/IDA/seriation>

BugReports <https://github.com/mhahsler/seriation>

Depends R (>= 2.14.0)

Imports TSP, qap, grid, cluster, gclus, dendextend, colorspace, MASS, gplots, registry, methods, stats, grDevices

Suggests biclust, testthat, DendSer, GA

License GPL-3

Copyright The code in src/bea.f is Copyright (C) 1991 F. Murtagh; src/bbwreg.f, src/arsa.f and src/bbureg.f are Copyright (C) 2005 M. Brusco, H.F. Koehn, and S. Stahl. All other code is Copyright (C) Michael Hahsler, Christian Buchta, and Kurt Hornik.

NeedsCompilation yes

Author Michael Hahsler [aut, cre, cph],
Christian Buchta [aut, cph],
Kurt Hornik [aut, cph],
Fionn Murtagh [ctb, cph],
Michael Brusco [ctb, cph],
Stephanie Stahl [ctb, cph],
Hans-Friedrich Koehn [ctb, cph]

Maintainer Michael Hahsler <mhahsler@lyle.smu.edu>

Repository CRAN

Date/Publication 2017-05-09 06:56:01 UTC

R topics documented:

bertinplot	2
Chameleon	4
color_palettes	5
criterion	7
criterion_methods	12
dissimilarity	14
dissplot	16
get_order	20
hmap	21
Irish	23
Munsingen	24
permutation	26
permutation_matrix	27
permutation_vector	28
permute	29
pimage	30
Psych24	33
register_DendSer	34
register_GA	35
reorder.hclust	37
Robinson	38
seriate	39
seriation_data	46
seriation_methods	48
SupremeCourt	50
Townships	51
uniscale	52
VAT	53
Wood	54
Zoo	55
Index	57

bertinplot

Plot a Bertin Matrix

Description

Plot a data matrix of cases and variables. Each value is represented by a symbol. Large values are highlighted. The matrix can be rearranged to make structure in the data visible (see Falguerolles et al 1997). `bertin_cut_line()` can be used to add cut lines (see Details).

Usage

```
bertinplot(x, order = NULL, highlight = TRUE, options = NULL)
```

Arguments

- x** a data matrix. Note that following Bertin, columns are variables and rows are cases. This behavior can be reversed using `reverse = TRUE` in options.
- order** an object of class `ser_permutation` to rearrange `x` before plotting. If `NULL`, no rearrangement is performed.
- highlight** a logical scalar indicating whether to use highlighting. If `TRUE`, all variables with values greater than the variable-wise mean are highlighted. To control highlighting, also a logical matrix with the same dimensions as `x` can be supplied.
- options** a list with options for plotting. The list can contain the following elements:
- `panel.function` a function to produce the symbols. Currently available functions are `panel.bars` (default), `panel.circles`, `panel.squares`, `panel.blocks` and `panel.lines`. For circles and squares `neg.` values are represented by a dashed border. For blocks all blocks are the same size (can be used with `shading=TRUE`).
 - `reverse` logical indicating whether to swap cases and variables in the plot. The default (`FALSE`) is to plot cases as columns and variables as rows.
 - `xlab`, `ylab` labels (default: use labels from `x`).
 - `spacing` relative space between symbols (default: 0.2).
 - `shading` use gray shades to encode value instead of highlighting (default: `FALSE`).
 - `frame` plot a grid to separate symbols (default: `codeFALSE`).
 - `mar` margins (see `par`).
 - `gp_labels` `gpar` object for labels (see `gpar`).
 - `gp_panels` `gpar` object for panels (see `gpar`).
 - `newpage` a logical indicating whether to start the plot on a new page (see `grid.newpage`).
 - `pop` a logical indicating whether to pop the created viewports (see `pop.viewport`)?

Details

The plot is organized as a matrix of symbols. The symbols are drawn by a panel function, where all symbols of a row are drawn by one call of the function (using vectorization). The interface for the panel function is `panel.myfunction(value, spacing, hl)`. `value` is the vector of values for a row scaled between 0 and 1, `spacing` contains the relative space between symbols and `hl` is a logical vector indicating which symbol should be highlighted.

Cut lines can be added to an existing bertin plot using `bertin_cut_line(x=NULL, y=NULL)`. `x/y` is can be a number indicating where to draw the cut line between two columns/rows. If both `x` and `y` is specified then one can select a row/column and the other can select a range to draw a line which does only span a part of the row/column. It is important to call `bertinplot()` with the option `pop=FALSE`.

Author(s)

Michael Hahsler

References

de Falguerolles, A., Friedrich, F., Sawitzki, G. (1997): A Tribute to J. Bertin's Graphical Data Analysis. In: Proceedings of the SoftStat '97 (Advances in Statistical Software 6), 11–20.

See Also

[ser_permutation](#), [seriate](#), Package [grid](#).

Examples

```
data("Irish")
scale_by_rank <- function(x) apply(x, 2, rank)
x <- scale_by_rank(Irish[,-6])

## use the the sum of absolute rank differences
order <- c(
  seriate(dist(x, "minkowski", p = 1)),
  seriate(dist(t(x), "minkowski", p = 1))
)

## plot
bertinplot(x, order)

## some alternative displays
bertinplot(x, order, options = list(shading = TRUE, panel = panel.blocks))
bertinplot(x, order, options = list(panel = panel.lines))
bertinplot(x, order, options = list(panel = panel.squares))
bertinplot(x, order,
  options = list(panel = panel.circles, spacing = -0.5))

## plot with cut lines (we manually set the order here)
order <- ser_permutation(c(21, 16, 19, 18, 14, 12, 20, 15,
  17, 26, 13, 41, 7, 11, 5, 23, 28, 34, 31, 1, 38, 40,
  3, 39, 4, 27, 24, 8, 37, 36, 25, 30, 33, 35, 2,
  22, 32, 29, 10, 6, 9),
  c(4, 2, 1, 6, 8, 7, 5, 3))

bertinplot(x, order, options=list(pop=FALSE))
bertin_cut_line(,4) ## horizontal line between rows 4 and 5
bertin_cut_line(,7) ## separate "Right to Life" from the rest
bertin_cut_line(14,c(0,4)) ## separate a block of large values (vertically)
```

Description

Several 2D data sets used to evaluate the CHAMELEON clustering algorithm in the paper by Karypis et al (1999) and used by iVAT, an ordering-based tool to assess cluster tendency (Havens and Bezdek, 2012).

Usage

```
data(Chameleon)
```

Format

chameleon_ds4: The format is a 8,000 x 2 data.frame.

chameleon_ds5: The format is a 8,000 x 2 data.frame.

chameleon_ds7: The format is a 10,000 x 2 data.frame.

chameleon_ds8: The format is a 8,000 x 2 data.frame.

Source

The data was obtained from <http://glaros.dtc.umn.edu/gkhome/cluto/cluto/download>

References

Karypis, G., EH. Han, V. Kumar (1999): CHAMELEON: A Hierarchical Clustering Algorithm Using Dynamic Modeling, *IEEE Computer*, **32**(8): 68–75.

Havens, T.C. and Bezdek, J.C. (2012): An Efficient Formulation of the Improved Visual Assessment of Cluster Tendency (iVAT) Algorithm, *IEEE Transactions on Knowledge and Data Engineering*, **24**(5), 813–822.

Examples

```
data(Chameleon)

plot(chameleon_ds4, cex=.1)
```

color_palettes

Different Useful Color Palettes

Description

Defines several color palettes for pimage, disssplot and hmap.

Usage

```
bluered(n, bias = 1)
greenred(n, bias = 1)
grays(n, power = 1)
greys(n, power = 1)
```

Arguments

n	number of different colors produces.
bias	a positive number. Higher values give more widely spaced colors at the high end.
power	control parameter determining how luminance should be increased (1 = linear, 2 = quadratic, etc.).

Details

bluered creates a blue-red color palette.

greenred creates a green-red color palette.

greys and grays creates gray scales.

See colorRampPalette to create your own color palettes.

Value

A vector with n colors.

Author(s)

Michael Hahsler

See Also

[colorRampPalette](#), [pimage](#), [dissplot](#), [hmap](#).

Examples

```
pimage(rbind(1:100))
pimage(rbind(1:100), col = greys(100, power=2))
pimage(rbind(1:100), col = bluered(100))
pimage(rbind(1:100), col = bluered(100, bias = 2))
pimage(rbind(-100:100), col = greenred(10))

## create your own color palettes
## red to green (with 10 colors)
pimage(rbind(1:100),
  col = colorRampPalette(colors = c("red", "yellow", "green"))(10))
## white to blue (with 100 colors)
pimage(rbind(1:100),
  col = colorRampPalette(colors = c("white", "blue"))(100))
```

criterion	<i>Criterion for a Loss/Merit Function for Data Given a Permutation</i>
-----------	-------------------------------------------------------------------------

Description

Compute the value for different loss functions L and merit function M for data given a permutation.

Usage

```
criterion(x, order = NULL, method = NULL, force_loss = FALSE, ...)
```

Arguments

x	an object of class <code>dist</code> or a matrix (currently no functions are implemented for array).
order	an object of class <code>ser_permutation</code> suitable for x. If <code>NULL</code> , the identity permutation is used.
method	a character vector with the names of the criteria to be employed, or <code>NULL</code> (default) in which case all available criteria are used.
...	additional parameters passed on to the criterion method.
force_loss	logical; should merit function be converted into loss functions by multiplying with -1?

Details

For a symmetric dissimilarity matrix D with elements $d(i, j)$ where $i, j = 1 \dots n$, the aim is generally to place low distance values close to the diagonal. The following criteria to judge the quality of a certain permutation of the objects in a dissimilarity matrix are currently implemented (for a more detailed description and an experimental comparison see Hahsler (2017)):

"Gradient_raw", "Gradient_weighted" Gradient measures (Hubert et al 2001). A symmetric dissimilarity matrix where the values in all rows and columns only increase when moving away from the main diagonal is called a perfect *anti-Robinson matrix* (Robinson 1951). A suitable merit measure which quantifies the divergence of a matrix from the anti-Robinson form is

$$M(D) = \sum_{i=1}^n \sum_{i < k < j} f(d_{ij}, d_{ik}) + \sum_{i < k < j} f(d_{ij}, d_{kj})$$

where $f(., .)$ is a function which defines how a violation or satisfaction of a gradient condition for an object triple (O_i, O_k, O_j) is counted.

Hubert et al (2001) suggest two functions. The first function is given by:

$$f(z, y) = \text{sign}(y - z) = +1 \text{ if } z < y; 0 \text{ if } z = y; \text{ and } -1 \text{ if } z > y.$$

It results in raw number of triples satisfying the gradient constraints minus triples which violate the constraints.

The second function is defined as:

$$f(z, y) = |y - z| \text{sign}(y - z) = y - z$$

It weights the each satisfaction or violation by the difference by its magnitude given by the absolute difference between the values.

"AR_events", "AR_deviations" Anti-Robinson events (Chen 2002). An even simpler loss function can be created in the same way as the gradient measures above by concentrating on violations only.

$$L(D) = \sum_{i=1}^n \sum_{i < k < j} f(d_{ik}, d_{ij}) + \sum_{i < k < j} f(d_{kj}, d_{ij})$$

To only count the violations we use

$$f(z, y) = I(z, y) = 1 \text{ if } z < y \text{ and } 0 \text{ otherwise.}$$

$I(\cdot)$ is an indicator function returning 1 only for violations. Chen (2002) presented a formulation for an equivalent loss function and called the violations *anti-Robinson events* and also introduced a weighted versions of the loss function resulting in

$$f(z, y) = |y - z| I(z, y)$$

using the absolute deviations as weights.

"RGAR" Relative generalized Anti-Robinson events (Tien et al 2008). Counts Anti-Robinson events in a variable band (window specified by w defaults to the maximum of $n - 1$) around the main diagonal and normalizes by the maximum of possible events.

$$L(D) = 1/m \sum_{i=1}^n \sum_{(i-w) \leq j < k < i} I(d_{ij} < d_{ik}) + \sum_{i < j < k \leq (i+w)} I(d_{ij} > d_{ik})$$

where $m = (2/3 - n)w + nw^2 - 2/3w^3$, the maximal number of possible anti-Robinson events in the window. The window size w represents the number of neighboring objects (number of entries from the diagonal of the distance matrix) are considered. The window size is $2 \leq w < n$, where smaller values result in focusing on the local structure while larger values look at the global structure. Alternatively, pct can be used instead of w to specify the window as a percentage of n . `relative=FALSE` can be to get the GAR, i.e., the absolute number of AR events in the window.

"BAR" Banded Anti-Robinson Form (Earle and Hurley 2015).

Simplified measure for closeness to the anti-Robinson form in a band of size b with $1 \leq b < n$ around the diagonal.

$$L(D) = \sum_{|i-j| \leq b} (b + 1 - |i - j|) d_{ij}$$

For $b = 1$ the measure reduces to the Hamiltonian path length. For $b = n - 1$ the measure is equivalent to ARc defined (Earle and Hurley, 2015). Note that ARc is equivalent to the Linear Seriation criterion (scaled by 1/2).

b defaults to a band of 20% of n .

"Path_length" Hamiltonian path length (Caraux and Pinloche 2005).

The order of the objects in a dissimilarity matrix corresponds to a path through a graph where each node represents an object and is visited exactly once, i.e., a Hamilton path. The length of the path is defined as the sum of the edge weights, i.e., dissimilarities.

$$L(D) = \sum_{i=1}^{n-1} d_{i,i+1}$$

The length of the Hamiltonian path is equal to the value of the minimal span loss function (as used by Chen 2002). Both notions are related to the *traveling salesperson problem (TSP)*.

If order is not unique or there are non-finite distance values NA is returned.

"Lazy_path_length" Lazy path length (Earl and Hurley 2015).

A weighted version of the Hamiltonian path criterion. This loss function postpones larger distances to later in the order (i.e., a lazy traveling sales person).

$$L(D) = \sum_{i=1}^{n-1} (n-i)d_{i,i+1}$$

Earl and Hurley (2015) proposed this criterion for reordering in visualizations to concentrate on closer objects first.

"Inertia" Inertia criterion (Caraux and Pinloche 2005).

Measures the moment of the inertia of dissimilarity values around the diagonal as

$$M(D) = \sum_{i=1}^n \sum_{j=1}^n d(i,j)|i-j|^2$$

$|i-j|$ is used as a measure for the distance to the diagonal and $d(i,j)$ gives the weight. This criterion gives higher weight to values farther away from the diagonal. It increases with quality.

"Least_squares" Least squares criterion (Caraux and Pinloche 2005).

The sum of squares of deviations between the dissimilarities and rank differences (in the matrix) between two elements:

$$L(D) = \sum_{i=1}^n \sum_{j=1}^n (d(i,j) - |i-j|)^2,$$

where $d(i,j)$ is an element of the dissimilarity matrix D and $|i-j|$ is the rank difference between the objects.

Note that if Euclidean distance is used to calculate D from a data matrix X , the order of the elements in X by projecting them on the first principal component of X minimizes this criterion. The least squares criterion is related to *unidimensional scaling*.

"LS" Linear Seriation Criterion (Hubert and Schultz 1976).

Weights the distances with the absolute rank differences.

$$L(D) = \sum_{i,j=1}^n d(i,j)(-|i-j|)$$

"2SUM" 2-Sum Criterion (Barnard, Pothen, and Simon 1993).

The 2-Sum loss criterion multiplies the similarity between objects with the squared rank differences.

$$L(D) = \sum_{i,j=1}^n 1/(1+d(i,j))(i-j)^2,$$

where $s(i,j) = 1/(1+d(i,j))$ represents the similarity between objects i and j .

"ME", "Moore_stress", "Neumann_stress", "Cor_R" These criteria are defined on general matrices (see below for definitions). The dissimilarity matrix is first converted into a similarity matrix using $S = 1/(1+D)$. If a different transformation is required, then perform the transformation first and supply a matrix instead of a dist object.

For a general matrix $X = x_{ij}$, $i = 1 \dots n$ and $j = 1 \dots m$, currently the following loss/merit functions are implemented:

"ME" Measure of Effectiveness (McCormick 1972).

The measure of effectiveness (ME) for matrix X , is defined as

$$M(X) = 1/2 \sum_{i=1}^n \sum_{j=1}^m x_{i,j} (x_{i,j-1} + x_{i,j+1} + x_{i-1,j} + x_{i+1,j})$$

with, by convention

$$x_{0,j} = x_{m+1,j} = x_{i,0} = x_{i,n+1} = 0.$$

ME is a merit measure, i.e. a higher ME indicates a better arrangement. Maximizing ME is the objective of the bond energy algorithm (BEA).

"Cor_R" Weighted correlation coefficient R developed as the Measure of Effectiveness for the Moment Ordering Algorithm (Deutsch and Martin 1971).

R is a merit measure normalized so that its value always lies in $[-1, 1]$. For the special case of a square matrix $R = 1$ corresponds to only the main diagonal being filled, $R = 0$ to a random distribution of value throughout the array, and $R = -1$ to the opposite diagonal only being filled.

"Moore_stress", "Neumann_stress" Stress (Niermann 2005).

Stress measures the conciseness of the presentation of a matrix/table and can be seen as a purity function which compares the values in a matrix/table with its neighbors. The stress measure used here is computed as the sum of squared distances of each matrix entry from its adjacent entries.

$$L(X) = \sum_{i=1}^n \sum_{j=1}^m \sigma_{ij}$$

The following types of neighborhoods are available:

Moore: comprises the eight adjacent entries.

$$\sigma_{ij} = \sum_{k=\max(1,i-1)}^{\min(n,i+1)} \sum_{l=\max(1,j-1)}^{\min(m,j+1)} (x_{ij} - x_{kl})^2$$

Neumann: comprises the four adjacent entries.

$$\sigma_{ij} = \sum_{k=\max(1,i-1)}^{\min(n,i+1)} (x_{ij} - x_{kj})^2 + \sum_{l=\max(1,j-1)}^{\min(m,j+1)} (x_{ij} - x_{il})^2$$

The major difference between the Moore and the Neumann neighborhood is that for the later the contribution of row and column permutations to stress are independent and thus can be optimized independently.

Value

A named vector of real values.

Author(s)

Christian Buchta and Michael Hahsler

References

- Barnard, S.T., A. Pothen, and H. D. Simon (1993): A Spectral Algorithm for Envelope Reduction of Sparse Matrices. *In Proceedings of the 1993 ACM/IEEE Conference on Supercomputing*, 493–502. Supercomputing '93. New York, NY, USA: ACM.
- Caraux, G. and S. Pinloche (2005): Permutmatrix: A Graphical Environment to Arrange Gene Expression Profiles in Optimal Linear Order, *Bioinformatics*, **21**(7), 1280–1281.
- Chen, C.-H. (2002): Generalized association plots: Information visualization via iteratively generated correlation matrices, *Statistica Sinica*, **12**(1), 7–29.
- Deutsch, S.B. and J.J. Martin (1971): An ordering algorithm for analysis of data arrays. *Operational Research*, **19**(6), 1350–1362.
- Earle, D. and C.B. Hurley (2015): Advances in Dendrogram Seriation for Application to Visualization. *Journal of Computational and Graphical Statistics*, **24**(1), 1–25.
- Hahsler, M. (2017): An experimental comparison of seriation methods for one-mode two-way data. *European Journal of Operational Research*, **257**, 133–143.
- Hubert, L. and J. Schultz (1976): Quadratic Assignment as a General Data Analysis Strategy. *British Journal of Mathematical and Statistical Psychology*, **29**(2). Blackwell Publishing Ltd. 190–241.
- Hubert, L., P. Arabie, and J. Meulman (2001): *Combinatorial Data Analysis: Optimization by Dynamic Programming*. Society for Industrial Mathematics.
- Niermann, S. (2005): Optimizing the Ordering of Tables With Evolutionary Computation, *The American Statistician*, **59**(1), 41–46.
- McCormick, W.T., P.J. Schweitzer and T.W. White (1972): Problem decomposition and data reorganization by a clustering technique, *Operations Research*, **20**(5), 993–1009.
- Robinson, W.S. (1951): A method for chronologically ordering archaeological deposits, *American Antiquity*, **16**, 293–301.
- Tien, Y.-J., Yun-Shien Lee, Han-Ming Wu and Chun-Houh Chen (2008): Methods for simultaneously identifying coherent local clusters with smooth global patterns in gene expression profiles, *BMC Bioinformatics*, **9**(155), 1–16.

See Also

[list_criterion_methods](#) to query the criterion registry.

Examples

```
## create random data and calculate distances
m <- matrix(runif(20),ncol=2)
d <- dist(m)

## get an order for rows (optimal for the least squares criterion)
o <- seriate(d, method = "MDS")
o

## compare the values for all available criteria
rbind(
  unordered = criterion(d),
  ordered = criterion(d, o)
)

## compare RGAR by window size (from local to global)
w <- 2:(nrow(m)-1)
RGAR <- sapply(w, FUN = function (w)
  criterion(d, o, method="RGAR", w = w))
plot(w, RGAR, type = "b", ylim = c(0,1),
  xlab = "Windows size (w)", main = "RGAR by window size")
```

criterion_methods *Registry for Criterion Methods*

Description

A registry to manage methods to calculate a criterion value given data and a permutation.

Usage

```
list_criterion_methods(kind)
show_criterion_methods(kind)
get_criterion_method(kind, name)
set_criterion_method(kind, name, fun, description = NULL, merit = NA, ...)
```

Arguments

kind	the data type the method works on. For example, "dist", "matrix" or "array".
name	a short name for the method used to refer to the method in the function <code>criterion()</code> .
fun	a function containing the method's code.
description	a description of the method. For example, a long name.
merit	a boolean indicating if the criterion measure is a merit (TRUE) or a loss (FALSE) measure.
...	further information that is stored for the method in the registry.

Details

All methods below are convenience methods for the registry named `registry_criterion`.

`list_criterion_method()` lists all available methods for a given data type (`kind`). The result is a vector of character strings with the short names of the methods.

`show_criterion_method()` shows all available methods for a given data type (`kind`) including a description.

`get_criterion_method()` returns information (including the implementing function) about a given method in form of an object of class "criterion_method".

With `set_criterion_method()` new criterion methods can be added by the user. The implementing function (`fun`) needs to have the formal arguments `x`, `order`, `...`, where `x` is the data object, `order` is an object of class `permutation_vector` and `...` can contain additional information for the method passed on from `criterion()`. The implementation has to return the criterion value as a scalar.

Author(s)

Michael Hahsler

Examples

```
## the registry
registry_criterion

## use the convenience functions
list_criterion_methods("dist")

show_criterion_methods("dist")

get_criterion_method("dist", "AR_d")

## define a new method

## a function that return sum of the diagonal elements
criterion_method_matrix_foo <- function(x, order, ...) {
  if(!is.null(order)) x <- permute(x,order)
  sum(diag(x))
}

## set new method
set_criterion_method("matrix", "foo", criterion_method_matrix_foo,
  "foo: a useless demo criterion", FALSE)

list_criterion_methods("matrix")

##use all criterion methods (including the new one)
criterion(matrix(1:9, ncol=3))
```

dissimilarity

Dissimilarities and Correlations Between Seriation Orders

Description

Calculates dissimilarities/correlations between seriation orders in a list.

Usage

```
ser_cor(x, y = NULL, method = "spearman", reverse = TRUE, test = FALSE)
ser_dist(x, y = NULL, method = "spearman", reverse = TRUE, ...)
ser_align(x, method = "spearman")
```

Arguments

x	set of seriation orders as a list with elements which can be coerced into <code>ser_permutation_vector</code> objects.
y	if not NULL then a single seriation order can be specified. In this case x has to be a single seriation order and not a list.
method	a character string with the name of the used measure. Available measures are: "kendall", "spearman", "manhattan", "euclidean", "hamming", "ppc" (positional proximity coefficient), and "apr" (absolute pairwise rank differences).
reverse	a logical indicating if the orders should also be checked in reverse order and the best value (highest correlation, lowest distance) is reported. This only affect ranking-based measures and not precedence invariant measures (e.g., ppc, apr).
test	a logical indicating if a correlation test should be performed.
...	Further arguments passed on to the method.

Details

`ser_cor` calculates the correlation between two sequences (orders). Note that a seriation order and its reverse are identical and purely an artifact due to the method that creates the order. This is a major difference to rankings. For ranking-based correlation measures (Spearman and Kendall) the absolute value of the correlation is returned for `reverse = TRUE` (in effect returning the correlation for the reversed order). If `test = TRUE` then the appropriate test for association is performed and a matrix with p-values is returned as the attribute "p-value". Note that no correction for multiple testing is performed.

For `ser_dist`, the correlation coefficients (Kendall's tau and Spearman's rho) are converted into a dissimilarity by taking one minus the correlation value. Note that Manhattan distance between the ranks in a linear order is equivalent to Spearman's footrule metric (Diaconis 1988). `reverse = TRUE` returns the pairwise minima using also reversed orders.

The positional proximity coefficient (ppc) is a precedence invariant measure based on product of the squared positional distances in two permutations defined as (see Goulermas et al 2016):

$$d_{ppc}(R, S) = 1/h \sum_{j=2}^n \sum_{i=1}^{j-1} (\pi_R(i) - \pi_R(j))^2 * (\pi_S(i) - \pi_S(j))^2,$$

where R and S are two seriation orders, π_R and π_S are the associated permutation vectors and h is a normalization factor. The associated generalized correlation coefficient is defined as $1 - d_{ppc}$. For this precedence invariant measure reverse is ignored.

The absolute pairwise rank difference (aprd) is also precedence invariant and defined as a distance measure:

$$d_{aprd}(R, S) = \sum_{j=2}^n \sum_{i=1}^{j-1} ||\pi_R(i) - \pi_R(j)| - |\pi_S(i) - \pi_S(j)||^p,$$

where p is the power which can be passed on as parameter p and is by default set to 2. For this precedence invariant measure reverse is ignored.

`ser_align` tries to normalize the direction in a list of seriations such that ranking-based methods can be used. We add for each permutation also the reversed order to the set and then use a modified version of Prim's algorithm for finding a minimum spanning tree (MST) to choose if the original seriation order or its reverse should be used. We use the orders first added to the MST. Every time an order is added, its reverse is removed from the possible remaining orders.

Value

`ser_dist` returns an object of class `dist`. `ser_align` returns a new list with elements of class `ser_permutation`.

Author(s)

Michael Hahsler

References

P. Diaconis (1988): Group Representations in Probability and Statistics. Institute of Mathematical Statistics, Hayward, CA.

J.Y. Goulermas, A. Kostopoulos, and T. Mu (2016): A New Measure for Analyzing and Fusing Sequences of Objects. IEEE Transactions on Pattern Analysis and Machine Intelligence 38(5):833-48.

See Also

[ser_permutation_vector](#)

Examples

```

set.seed(1234)
## seriate dist of 50 flowers from the iris data set
data("iris")
x <- as.matrix(iris[-5])
x <- x[sample(1:nrow(x), 50),]
rownames(x) <- 1:50
d <- dist(x)

## Create a list of different seriations
methods <- c("HC_single", "HC_complete", "OLO", "GW", "R2E", "VAT",
            "TSP", "Spectral", "SPIN", "MDS", "Identity", "Random")

os <- sapply(methods, function(m) {
  cat("Doing", m, "... ")
  tm <- system.time(o <- seriate(d, method = m))
  cat("took", tm[3], "s.\n")
  o
})

## Compare the methods using distances. Default is based on
## Spearman's rank correlation coefficient. Reverse orders are considered
## equivalent.
ds <- ser_dist(os)
hmap(ds, margin=c(7,7))

## Compare using actual correlation between orders. Reversed orders have
## negative correlation!
cs <- ser_cor(os, reverse = FALSE)
hmap(cs, margin=c(7,7))

## Also check reversed seriation orders.
## Now all but random and identity are highly positive correlated
cs2 <- ser_cor(os, reverse = TRUE)
hmap(cs2, margin=c(7,7))

## Use Manhattan distance of the ranks (i.e., Spearman's foot rule)
ds <- ser_dist(os, method="manhattan")
plot(hclust(ds))

```

dissplot

Dissimilarity Plot

Description

Visualizes a dissimilarity matrix using seriation and matrix shading using the method developed by Hahsler and Hornik (2011). Entries with lower dissimilarities (higher similarity) are plotted darker. Such a plot can be used to uncover hidden structure in the data.

The plot can also be used to visualize cluster quality (see Ling 1973). Objects belonging to the same cluster are displayed in consecutive order. The placement of clusters and the within cluster order is

obtained by a seriation algorithm which tries to place large similarities/small dissimilarities close to the diagonal. Compact clusters are visible as dark squares (low dissimilarity) on the diagonal of the plot. Additionally, a Silhouette plot (Rousseeuw 1987) is added. This visualization is similar to CLUSION (see Strehl and Ghosh 2002), however, allows for using arbitrary seriating algorithms.

Usage

```
dissplot(x, labels = NULL, method = "Spectral",
         control = NULL, options = NULL, ...)
```

Arguments

- | | |
|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x | an object of class <code>dist</code> . |
| labels | NULL or an integer vector of the same length as rows/columns in <code>x</code> indicating the cluster membership for each object in <code>x</code> as consecutive integers starting with one. The labels are used to reorder the matrix. |
| method | <p>a list with up to three elements or a single character string. Use a single character string to apply the same algorithm to reorder the clusters (inter cluster seriation) as well as the objects within each cluster (intra cluster seriation).</p> <p>If separate algorithms for inter and intra cluster seriation are required, <code>method</code> can be a list of two named elements (<code>inter_cluster</code> and <code>intra_cluster</code> each containing the name of the respective seriation method. See <code>seriate.dist</code> for available algorithms.</p> <p>Set <code>method</code> to <code>NA</code> to plot the matrix as is (no or only coarse seriation). For intra cluster reordering the special method <code>"silhouette width"</code> is available. Objects in clusters are then ordered by silhouette width (from silhouette plots). If no <code>method</code> is given, the default method of <code>seriate.dist</code> is used.</p> <p>The third list element (named <code>aggregation</code>) controls how inter cluster dissimilarities are computed from from the given dissimilarity matrix. The choices are <code>"avg"</code> (average pairwise dissimilarities; average-link), <code>"min"</code> (minimal pairwise dissimilarities; single-link), <code>"max"</code> (maximal pairwise dissimilarities; complete-link), and <code>"Hausdorff"</code> (pairs up each point from one cluster with the most similar point from the other cluster and then uses the largest dissimilarity of paired up points).</p> |
| control | a list of control options passed on to the seriation algorithm. In case of two different seriation algorithms, <code>control</code> can contain a list of two named elements (<code>inter_cluster</code> and <code>intra_cluster</code>) containing each a list with the control options for the respective algorithm. |
| options | <p>a list with options for plotting the matrix. The list can contain the following elements:</p> <p><code>plot</code> a logical indicating if a plot should be produced. if <code>FALSE</code>, the returned object can be plotted later using the function <code>plot</code> which takes as the second argument a list of plotting options (see options below).</p> <p><code>cluster_labels</code> a logical indicating whether to display cluster labels in the plot.</p> |

`averages` a logical vector of length two. The first element controls the upper triangle and the second element the lower triangle of the plot. `FALSE` displays the original dissimilarity between objects, `TRUE` displays cluster-wise average dissimilarities, and `NA` leaves the triangle white (default: `c(FALSE, TRUE)`, i.e., the lower triangle displays averages)

`lines` a logical indicating whether to draw lines to separate clusters.

`flip` a logical indicating if the clusters are displayed on the diagonal from north-west to south-east (`FALSE`; default) or from north-east to south-west (`TRUE`).

`silhouettes` a logical indicating whether to include a silhouette plot (see Rousseeuw, 1987).

`threshold` a numeric. If used, only plot distances below the threshold are displayed. Consider also using `zlim` for this purpose.

`col` colors used for the image plot.

`key` a logical indicating whether to place a color key below the plot.

`zlim` range of values to display (defaults to range `x`).

`axes` "auto" (default; enabled for less than 25 objects), "y" or "none".

`main` title for the plot.

`newpage` a logical indicating whether to start plot on a new page (see `grid.newpage` in package **grid**).

`pop` a logical indicating whether to pop the created viewports (see package **grid**)?

`gp`, `gp_lines`, `gp_labels` objects of class `gpar` containing graphical parameters (see `gpar` in package **grid**).

... further arguments are added to options.

Value

An invisible object of class `cluster_proximity_matrix` with the following elements:

<code>order</code>	NULL or integer vector giving the order used to plot <code>x</code> .
<code>cluster_order</code>	NULL or integer vector giving the order of the clusters as plotted.
<code>method</code>	vector of character strings indicating the seriation methods used for plotting <code>x</code> .
<code>k</code>	NULL or integer scalar giving the number of clusters generated.
<code>description</code>	a <code>data.frame</code> containing information (label, size, average intra-cluster dissimilarity and the average silhouette) for the clusters as displayed in the plot (from top/left to bottom/right).

This object can be used for plotting via `plot(x, options = NULL, ...)`, where `x` is the object and `options` contains a list with plotting options (see above).

Author(s)

Michael Hahsler

References

- Hahsler, M. and Hornik, K. (2011): Dissimilarity plots: A visual exploration tool for partitional clustering. *Journal of Computational and Graphical Statistics*, **10**(2):335–354.
- Ling, R.F. (1973): A computer generated aid for cluster analysis. *Communications of the ACM*, **16**(6), 355–361.
- Rousseeuw, P.J. (1987): Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, **20**(1), 53–65.
- Strehl, A. and Ghosh, J. (2003): Relationship-based clustering and visualization for high-dimensional data mining. *INFORMS Journal on Computing*, **15**(2), 208–230.

See Also

[dist](#), [seriate](#), [pimage](#) and [hmap](#).

Examples

```
data("iris")
d <- dist(iris[-5])

## plot original matrix
res <- dissplot(d, method = NA)

## plot reordered matrix using the nearest insertion algorithm (from tsp)
res <- dissplot(d, method = "TSP",
  options = list(main = "Seriation (TSP)"))

## cluster with pam (we know iris has 3 clusters)
library("cluster")
l <- pam(d, 3, cluster.only = TRUE)

## we use a grid layout to place several plots on a page
library("grid")
grid.newpage()
pushViewport(viewport(layout=grid.layout(nrow = 2, ncol = 2),
  gp = gpar(fontsize = 8)))
pushViewport(viewport(layout.pos.row = 1, layout.pos.col = 1))

## visualize the clustering (using Spectral between clusters and MDS within)
res <- dissplot(d, l, method = list(inter = "Spectral", intra = "MDS"),
  options = list(main = "PAM + Seriation - standard",
  newpage = FALSE))

popViewport()
pushViewport(viewport(layout.pos.row = 1, layout.pos.col = 2))

## more visualization options. Note that we reuse the reordered object res!
## color: use 10 shades red-blue
plot(res, options = list(main = "PAM + Seriation",
  col= bluered(10, bias=.5), newpage = FALSE))
```

```

popViewport()
pushViewport(viewport(layout.pos.row = 2, layout.pos.col = 1))

## threshold (using zlim) and cubic scale to highlight differences
plot(res, options = list(main = "PAM + Seriation - threshold",
  zlim = c(0, 1.5), col = greys(100, power = 2), newpage = FALSE))

popViewport()
pushViewport(viewport(layout.pos.row = 2, layout.pos.col = 2))

## use custom (logistic) scale
plot(res, options = list(main = "PAM + Seriation - logistic scale",
  col= hcl(c = 0, l = (plogis(seq(10, 0, length=100),
location = 2, scale = 1/2, log = FALSE))*100),
newpage = FALSE))

popViewport(2)

## the reordered_cluster_dissimilarity_matrix object
res
names(res)

```

get_order

Extracting Order Information from a Permutation Object

Description

Method to get the order information from an object of class `ser_permutation` or `ser_permutation_vector`. Order information can be extracted as an integer permutation vector, a vector containing the object ranks or a permutation matrix.

Usage

```

get_order(x, ...)
## S3 method for class 'ser_permutation_vector'
get_order(x, ...)
## S3 method for class 'ser_permutation'
get_order(x, dim = 1, ...)

get_rank(x, ...)

get_permutation_matrix(x, ...)

```

Arguments

<code>x</code>	an object of class <code>ser_permutation</code> or <code>ser_permutation_vector</code> .
<code>dim</code>	order information for which dimension should be returned?
<code>...</code>	further arguments are ignored for <code>get_order</code> . For <code>get_rank</code> and for <code>get_permutation_matrix</code> the additional arguments are passed on to <code>get_order</code> (e.g., as <code>dim</code>).

Details

`get_order` returns the seriation as an integer vector containing the order of the objects after permutation. That is, the index of the first, second, ..., n -th object. These permutation vectors can directly be used to reorder objects using subsetting with "[". *Note:* In **seriation** we usually use these order-based permutation vectors.

`get_rank` returns the seriation as an integer vector containing the rank/position for each objects in the permutation. That is, position of the first, second, ..., n -th object after permutation. *Note:* Use `order()` to convert ranks back to an order.

`get_permutation_matrix` returns a $n \times n$ permutation matrix.

Value

Returns an integer permutation vector/a permutation matrix.

Author(s)

Michael Hahsler

See Also

[ser_permutation_vector](#), [ser_permutation](#)

Examples

```
## permutation_vector
o <- ser_permutation_vector(sample(10))
o

get_order(o)
get_rank(o)
get_permutation_matrix(o)

## permutation
o2 <- ser_permutation(o, sample(5))
o2

get_order(o2, dim = 2)
get_rank(o2, dim = 2)
get_permutation_matrix(o2, dim = 2)
```

Description

Provides heat maps reordered using several different seriation methods. This includes dendrogram based reordering with optimal leaf order and matrix seriation based heat maps.

Usage

```
hmap(x, distfun = dist, method = "OLO", control = NULL, zlim = NULL, ...)
```

Arguments

<code>x</code>	a matrix or a dissimilarity matrix of class <code>dist</code> . If a dissimilarity matrix is used, then the <code>distfun</code> is ignored.
<code>distfun</code>	function used to compute the distance (dissimilarity) between both rows and columns (default: <code>dist</code>).
<code>method</code>	a character strings indicating the used seriation algorithm (see <code>seriate.dist</code>). If the method results in a dendrogram then heatmap in stats is used to show the dendrograms, otherwise reordered distance matrices are shown instead.
<code>control</code>	a list of control options passed on to the seriation algorithm specified in <code>method</code> .
<code>zlim</code>	range of values to display (defaults to the range of <code>x</code>).
<code>...</code>	further arguments.

Details

For dendrogram based heat maps the arguments are passed on to `heatmap.2` in **gplots**. See for example `margins` and `col`. The following arguments for `heatmap.2` cannot be used: `Rowv`, `Colv`, `hclustfun`, `reorderfun`.

For seriation-based heat maps further arguments include:

`gp` an object of class `gpar` containing graphical parameters (see `gpar` in package **grid**).

`newpage` a logical indicating whether to start plot on a new page (see `gpar` in package **grid**).

`prop` a logical indicating whether the height and width of `x` should be plotted proportional to its dimensions.

`showdist` Display seriated dissimilarity matrices? Values are "none", "both", "rows" or "columns".

`key` logical; show a colorkey?.

key.lab string plotted next to the color key.

`axes` one of "auto" (default; show axis labels if there are less than 25 labels), "x", "y", "both" and "none".

`margins` bottom and right-hand-side margins are calculated automatically or can be specifies as a vector of two numbers (in lines).

`zlim` range of values displayed.

`col, col_dist` color palettes used.

For `dendrogram = TRUE`, `seriate.hclust` with the default method "optimal" is used for arranging the dendrograms and `x`. `heatmap` is used for plotting.

For `dendrogram = FALSE`, `seriate.dist` with the default method "tsp" (a traveling salesperson solver) for arranging `x` is used. **grid** code implemented in this package is used to produce the plot.

Note that unlike the default behavior of `heatmap`, scaling is not automatically applied. The data have to be scaled before using `hmap`.

Value

An invisible list with elements:

`rowInd`, `colInd` index permutation vectors.

`reorder_method` name of the method used to reorder the matrix.

The list may contain additional elements (dendrograms, colors, etc).

Author(s)

Michael Hahsler

See Also

[seriate](#), [pimage](#), [dissplot](#), [heatmap.2](#) in [gplots](#).

Examples

```
data("Wood")

## default heatmap does Euclidean distance, hierarchical clustering with
## average-link and optimal leaf ordering
hmap(Wood)

## heatmap with correlation-based distance, green-red color (greenred is
## predefined) and optimal leaf ordering and no row label
dist_cor <- function(x) as.dist(1-cor(t(x)))

hmap(Wood, method="OLO", distfun = dist_cor, col=greenred(100), labRow=FALSE)

## order-based heatmap
hmap(Wood, method="MDS_angle", distfun = dist_cor, col=greenred(100))

## order-based with dissimilarity matrices
hmap(Wood, method="MDS_angle", distfun = dist_cor, showdist = "both",
     col=greenred(100))
```

Irish

Irish Referendum Data Set

Description

A data matrix containing the results of 8 referenda for 41 Irish communities used in Falguerolles et al (1997).

Usage

```
data(Irish)
```

Format

The format is a 41 x 9 matrix. Two values are missing.

Details

Column 6 contains the size of the Electorate in 1992.

Source

The data was kindly provided by Guenter Sawitzki.

References

de Falguerolles, A., Friedrich, F., Sawitzki, G. (1997): A Tribute to J. Bertin's Graphical Data Analysis. In: Proceedings of the SoftStat '97 (Advances in Statistical Software 6), 11–20.

Examples

```
data(Irish)
```

Munsingen

Hodson's Munsingen Data Set

Description

This data set contains a grave times artifact incidence matrix for the Celtic Münsingen-Rain cemetery in Switzerland as provided by Hodson (1968) and published by Kendall 1971.

Usage

```
data("Munsingen")
```

Format

A 59 x 70 0-1 matrix. Rows (graves) and columns (artifacts) are in the order determined by Hodson (1968).

References

Hodson, F.R. (1968): *The La Tene Cemetery at Münsingen-Rain*. Stämpfli, Bern.

Kendall, D.G. (1971): Seriation from abundance matrices. In: Hodson, F.R., Kendall, D.G. and Tautu, P., (Editors). *Mathematics in the Archaeological and Historical Sciences*, Edinburgh University Press, Edinburgh, 215–232.

Examples

```

data("Munsingen")

## Seriation method after Kendall (1971)
## Kendall's square symmetric matrix S and SoS
S <- function(x, w = 1) {
  sij <- function(i, j) w * sum(pmin(x[i,], x[j,]))
  h <- nrow(x)
  r <- matrix(ncol = h, nrow = h)
  for(i in 1:h) for (j in 1:h) r[i,j] <- sij(i,j)
  r
}

SoS <- function(x) S(S(x))

## Kendall's horse shoe (Hamiltonian arc)
horse_shoe_plot <- function(mds, sigma, threshold = mean(sigma), ...) {
  plot(mds, main = paste("Kendall's horse shoe with th =", threshold), ...)
  l <- which(sigma > threshold, arr.ind=TRUE)
  for(i in 1:nrow(l)) lines(rbind(mds[l[i,1],], mds[l[i,2],]))
}

## shuffle data
x <- Munsingen[sample(nrow(Munsingen)),]

## calculate matrix and do isoMDS (from package MASS)
sigma <- SoS(x)
library("MASS")
mds <- isoMDS(1/(1+sigma))$points

## plot Kendall's horse shoe
horse_shoe_plot(mds, sigma)

## find order using a TSP
library("TSP")
tour <- solve_TSP(insert_dummy(TSP(dist(mds)), label = "cut"),
  method = "2-opt", control = list(rep = 15))
tour <- cut_tour(tour, "cut")
lines(mds[tour,], col = "red", lwd = 2)

## create and plot order
order <- ser_permutation(tour, 1:ncol(x))
bertinplot(x, order, options= list(panel=panel.circles,
  rev = TRUE))

## compare criterion values
rbind(
  random = criterion(x),
  reordered = criterion(x, order),
  Hodson = criterion(Munsingen)
)

```

permutation	<i>Class ser_permutation – A Collection of Permutation Vectors for Seriation</i>
-------------	----------------------------------------------------------------------------------

Description

The class `ser_permutation` is a collection of permutation vectors (see class `ser_permutation_vector`), one for each dimension (mode) of the data to be permuted.

Usage

```
## constructor  
ser_permutation(x, ...)
```

Arguments

<code>x</code>	an object of class <code>ser_permutation_vector</code> or any object which can be converted into a object of class <code>ser_permutation</code> (e.g. an integer vector).
<code>...</code>	permutation vectors for further dimensions

Details

The basic functions `print`, `"["`, `"[["` and `c` are provided.

Value

An object of class `ser_permutation`.

Author(s)

Michael Hahsler

See Also

[ser_permutation_vector](#), [get_order](#), [get_permutation_matrix](#)

Examples

```
o <- ser_permutation(1:5, 10:1)  
o  
  
## length (number of dimensions)  
length(o)  
  
## get permutation vector for 2nd dimension  
get_order(o, 2)  
  
## reverse dimensions  
o[2:1]
```

```
## combine
o <- c(o, ser_permutation(1:15))
o

## get an individual permutation
o[[2]]

## reverse the order of a permutation
o[[2]] <- rev(o[[2]])
get_order(o,2)
```

permutation_matrix *Conversion Between Permutation Vector and Permutation Matrix*

Description

Converts between permutation vectors and matrices.

Usage

```
permutation_matrix2vector(x)
permutation_vector2matrix(x)
```

Arguments

x A permutation vector (any object that can be converted into a permutation vector, e.g., a integer vector or a hclust object) or a matrix representing a permutation. Arguments are checked.

Author(s)

Michael Hahsler

See Also

[ser_permutation](#), [permute](#)

Examples

```
## create a random permutation vector
pv <- sample(1:5)
pv

## convert into a permutation matrix
pm <- permutation_vector2matrix(pv)
pm

## convert back
permutation_matrix2vector(pm)
```

permutation_vector	<i>Class ser_permutation_vector – A Single Permutation Vector for Seriation</i>
--------------------	---------------------------------------------------------------------------------

Description

The class `ser_permutation_vector` represents a single permutation vector.

Usage

```
## constructor
ser_permutation_vector(x, method = NULL)
```

Arguments

x	an object which contains a permutation vector (currently an integer vector or an object of class <code>hclust</code>). The value <code>NA</code> creates an identity permutation.
method	a string representing the method used to obtain the permutation vector

Details

A permutation vector maps a set of n objects $\{O_1, O_2, \dots, O_n\}$ onto itself. In **seriation** we represent a permutation π as a vector which lists the objects in their permuted order. For example, the permutation vector $\langle 3, 1, 2 \rangle$ indicates that in first position is the object with index 3 then the object with index 1 and finally the object with index 2. A permutation vector can be extracted from a permutation vector object via `get_order()`. Such a permutation vector can be directly used to subset the list of original objects with `"["` to apply the permutation. *Note:* An alternative way to specify a permutation is via a list of the ranks of the objects after permutation (see `get_rank()`).

`ser_permutation_vector` objects are usually packed into a `ser_permutation` object which is a collection of k permutation vectors for k -mode data.

The constructor `ser_permutation_vector` checks if the permutation vector is valid (i.e. if all integers occur exactly once).

The following functions are implemented: `print`, `rev`, `length`, `get_order`, `get_rank`, `get_method`.

Value

An object of class `ser_permutation_vector`.

Author(s)

Michael Hahsler

See Also

[ser_permutation](#), [get_order](#), [get_rank](#), [get_permutation_matrix](#), [permutation_vector2matrix](#).

Examples

```

p <- ser_permutation_vector(sample(10), "random")
p

## some methods
length(p)
get_method(p)
get_order(p)
get_rank(p)
get_permutation_matrix(p)

r <- rev(p)
r
get_order(r)

## create a identity permutation vector (with unknown length)
ip <- ser_permutation_vector(NA)
ip

```

permutate

*Permute the Order in Various Objects***Description**

Provides the generic function and methods for permuting the order of various objects including vectors, dendrograms (also hclust objects), the order of observations in a dist object, the rows and columns of a matrix, all dimensions of an array given a suitable ser_permutation object.

Usage

```
permutate(x, order, ...)
```

Arguments

x	an object (a list, a vector, a dist object, a matrix, an array or any other object which provides dim and standard subsetting with "[").
order	an object of class ser_permutation which contains suitable permutation vectors for x.
...	additional arguments for the permutation function.

Details

The permutation vectors in ser_permutation are suitable if the number of permutation vectors matches the number of dimensions of x and if the length of each permutation vector has the same length as the corresponding dimension of x.

For 1-dimensional/1-mode data (list, vector, dist), order can also be a single permutation vector of class ser_permutation_vector or data which can be automatically coerced to this class (e.g. a numeric vector).

For dendrograms and `hclust`, subtrees are rotated to represent the order best possible. If the order is not achieved perfectly then the user is warned. This behavior can be changed with the extra parameter `incompatible` which can take the values "warn" (default), "stop" or "ignore".

Author(s)

Michael Hahsler

See Also

[ser_permutation](#), [dist](#) in package `stats`.

Examples

```
## permute matrix
m <- matrix(rnorm(10), 5, 2, dimnames = list(1:5, 1:2))
m

## permute rows and columns
permute(m, ser_permutation(5:1, 2:1))
## permute only columns
permute(m, ser_permutation(NA, 2:1))

## permute objects in a dist object
d <- dist(m)
d

permute(d, ser_permutation(c(3,2,1,4,5)))

## permute a list
l <- list(a=1:5, b=letters[1:3], c=0)
l

permute(l, c(2,3,1))

## permute a dendrogram
hc <- hclust(d)
plot(hc)
plot(permute(hc, 5:1))
```

pimage

Permutation Image Plot

Description

Provides methods for plotting image plots for matrix and `dist` objects given a permutation. By default, no permutation is performed. This plot can also be used as a more versatile replacement of image plot in `graphics` based on `grid`.

Usage

```
pimage(x, order = NULL, col = NULL, main = "", xlab = "", ylab = "",
       axes = "auto", zlim=NULL, key=TRUE, key.lab="", symkey=TRUE,
       upper.tri = TRUE, lower.tri = TRUE, prop = NULL,
       ..., newpage=TRUE, pop=TRUE, gp=NULL)
```

Arguments

x	a matrix or an object of class <code>dist</code> .
order	an object of class <code>ser_permutation</code> . If <code>NULL</code> the order in <code>x</code> is plotted.
col	a list of colors used. If <code>NULL</code> , a gray scale is used (for matrix larger values are displayed darker and for <code>dist</code> smaller distances are darker). For matrices containing logical data, black and white is used. For matrices containing negative values a symmetric diverging color palette is used.
main	plot title.
xlab, ylab	labels for the x and y axes.
axes	a character string indicating if axes labels (column and row names of <code>x</code>) should be potted. Possible values are "auto" (only plot if less then 25 labels), "x", "y", "both" and "none".
zlim	vector with two elements giving the range (min, max) for representing the values in the matrix.
key	logical; add a color key? No key is available for logical matrices.
key.lab	string plotted next to the color key.
symkey	logical; if <code>x</code> contains negative values, should the color palate be symmetric (zero is in the middle)>
upper.tri, lower.tri	a logical indicating whether to show the upper or lower triangle of the distance matrix.
prop	logical; draw the cells in the image proportional (defaults to <code>TRUE</code> for <code>dist</code> and <code>FALSE</code> for matrix).
...	further arguments passed on to <code>image</code> in graphics .
newpage, pop	two logical. Start plot on a new page and pop the viewports after plotting (see grid).
gp	a <code>gpar</code> object (see grid).

Details

Plots a matrix in its original row and column orientation. This means, in a plot the columns become the x-coordinates and the reversed rows the y-coordinates.

If `x` is of class `dist` it is converted to full-storage representation before plotting.

The viewports used for plotting are called: "plot", "image" and "colorkey".

Author(s)

Christian Buchta and Michael Hahsler

See Also

[seriate](#), [hmap](#), [dissplot](#) and [image](#).

Examples

```
x <- matrix(sample(c(FALSE, TRUE), 300, rep=TRUE), ncol=10,
  dimnames = list(1:30, LETTERS[1:10]))

## matrix (large values are dark/black)
pimage(x, main = "Random data", key = FALSE)

## plot seriated matrix (use red, proportional display and plot all axes)
pimage(x, seriate(x), col = c("white", "red"),
  prop = TRUE, axes="both", main = "Reordered data", key = FALSE)

## show correlation (for neg. values a diverging color scheme is
## used automatically)
pimage(cor(x), prop=TRUE)

## distances (note that low distances are represented dark!)
d <- dist(x, method = "binary")
pimage(d, upper.tri = FALSE, main = "Distances")

pimage(d, seriate(d), upper.tri = FALSE, main = "Distances", axes = "both")

## add to the plot using grid (use pop = FALSE)
library(grid)
pimage(x, pop = FALSE)
downViewport(name = "image")

## highlight cell 7/5 with a red arrow
grid.lines(x = c(5, 7), y = c(3, 5), arrow = arrow(),
  default.units = "native", gp = gpar(col="red", lwd = 3))

## add a red box around rows 15 and 16
grid.rect(x = 0.5, y = 15.5, width = ncol(x), height = 2,
  just = "left",
  default.units = "native", gp = gpar(col="red", lwd = 3, fill = NA))

## remove the viewports
popViewport(0)

## put several pimages on a page (uses viewports and newpage = FALSE)
library(grid)
grid.newpage()
pushViewport(viewport(layout=grid.layout(nrow = 1, ncol = 2)))
pushViewport(viewport(layout.pos.row = 1, layout.pos.col = 1))

## seriate matrix
o <- seriate(x)
pimage(x, o, main = "Random data", prop = TRUE, axes = "both", key = FALSE,
  newpage = FALSE)
```



```

upViewport(1)
pushViewport(viewport(layout.pos.row = 1, layout.pos.col = 2))

## add the reordered dissimilarity matrix for rowa
pimage(d, o[[1]], upper.tri = FALSE, main = "Distances", axes = "both",
       key = FALSE, newpage = FALSE)

upViewport(1)
popViewport(0)

```

Psych24

Results of 24 Psychological Test for 8th Grade Students

Description

A data set collected by Holzinger and Swineford (1939) which consists of the results of 24 psychological tests given to 145 seventh and eighth grade students in a Chicago suburb. This data set contains the correlation matrix for the 24 test results.

The data set was also used as an example for visualization of cluster analysis by Ling (1973).

Usage

```
data("Psych24")
```

Format

A 24 x 24 correlation matrix.

References

Holzinger, K. L., Swineford, F. (1939): A study in factor analysis: The stability of a bi-factor solution. *Supplementary Educational Monograph*, No. **48**. Chicago: University of Chicago Press.

Ling, R. L. (1973): A computer generated aid for cluster analysis. *Communications of the ACM*, **16**(6), pp. 355–361.

Examples

```

data("Psych24")

## create a dist object and also get rid of the one negative entry in the
## correlation matrix
d <- as.dist(1 - abs(Psych24))

pimage(d)

## do hclust as in Ling (1973)
hc <- hclust(d, method = "complete")
plot(hc)

```

```
pimage(d, hc)

## use seriation
order <- seriate(d, method = "tsp")
#order <- seriate(d, method = "tsp", control = list(method = "concorde"))
pimage(d, order)
```

register_DendSer *Register Seriation Methods from Package DendSer*

Description

Register the DendSer dendrogram seriation method and the ARc criterion (Earle and Hurley, 2015).

Usage

```
register_DendSer()
```

Details

Registers the method "DendSer" for `seriate`. DendSer is a fast heuristic for reordering dendrograms developed by Earle and Hurley (2015) able to use different criteria. `control` for `seriate` with method "DendSer" accepts the following parameters:

"h" or "method" A dendrogram or a method for hierarchical clustering (see `hclust`). Default: `complete-link`.

"criterion" A seriation criterion to optimize (see `list_criterion_methods("dist")`). Default: "BAR" (Banded anti-Robinson from with 20% band width).

"verbose" print progress information.

"DendSer_args" additional arguments for DendSer.

For convenience the following methods (for different cost functions) are also provided: "DendSer_ARc" (anti-robinson form), "DendSer_BAR" (banded anti-Robinson form), "DendSer_LS" (leaf seriation), "DendSer_PL" (path length).

Note: Package **DendSer** needs to be installed.

Author(s)

Michael Hahsler based on code by Catherine B. Hurley and Denise Earle

References

D. Earle, C. B. Hurley (2015): Advances in dendrogram seriation for application to visualization. *Journal of Computational and Graphical Statistics*, **24**(1), 1–25.

See Also

[seriate](#), [DendSer](#) in **DendSer**.

Examples

```
## Not run:
register_DendSer()
list_seriation_methods("dist")

d <- dist(random.robinson(20, pre=TRUE))

## use Banded AR form with default clustering (complete-link)
o <- seriate(d, "DendSer_BAR")
pimage(d, o)

## use different hclust method (Ward) and AR as the cost function for
## dendrogram reordering
o <- seriate(d, "DendSer", control = list(method = "ward.D2", criterion = "AR"))
pimage(d, o)

## End(Not run)
```

register_GA

Register a Genetic Algorithm Seriation Method

Description

Register a GA-based seriation metaheuristic.

Usage

```
register_GA()
```

Details

Registers the method "GA" for seriate. This method can be used to optimize any criterion in package **seriation**. control for seriate with method "GA" accepts the following parameters:

"criterion" criterion to optimize. Default: BAR

"suggestions" suggestions to warm start the GA. NA means no warm start. Default: TSP, QAP_LS and Spectral.

"selection" Selection operator (see **GA**). Default: non-linear rank selection

"crossover" Crossover operator (see **GA**). Default: ordered crossover (OX)

"mutation" Mutation operator (see **GA**). Default: a mixture of the simple insertion (80% chance) and simple inversion (20% chance) operators.

"pmutation" probability for permutations. Default: .5

"pcrossover" probability for crossover. Default: .2

"popsize" the population size. Default: 100
 "maxiter" maximum number of generations. Default: 1000
 "run" stop after run generations without improvement. Default: 50
 "parallel" use multiple cores? Default: TRUE
 "verbose" Report progress? Default: TRUE

The GA uses by default the ordered cross-over (OX) operator. For mutation, the GA uses a mixture of simple insertion and simple inversion operators. This mixed operator is created using `seriation::gaperm_mixedMutation(ismProb = .8)`, where `ismProb` is the probability that the simple insertion mutation operator is used. See package **GA** for a description of other available cross-over and mutation operators for permutaitons. The appropriate operator functions in **GA** start with `gaperm_`.

We warm start the GA using "suggestions" given by several heuristics. Set "suggestions" to NA to start with a purely random initial population.

Note: Package **GA** needs to be installed.

Author(s)

Michael Hahsler

References

Luca Scrucca (2013): GA: A Package for Genetic Algorithms in R. *Journal of Statistical Software*, 53(4), 1–37. URL <http://www.jstatsoft.org/v53/i04/>.

See Also

[seriate](#), [ga](#) in **GA**.

Examples

```
## Not run:
register_GA()
list_seriation_methods("dist")

d <- dist(random.robinson(50, pre=TRUE, noise=.1))

## use default settings: Banded AR form
o <- seriate(d, "GA")
pimage(d, o)

## optimize for linear sertiation criterion (LS)
o <- seriate(d, "GA", control = list(criterion = "LS"))
pimage(d, o)

## no warm start
o <- seriate(d, "GA", control = list(criterion = "LS", suggestions = NA))
pimage(d, o)

## End(Not run)
```

reorder.hclust *Reorder Dendrograms using Optimal Leaf Ordering*

Description

Reorder method for dendrograms for optimal leaf ordering.

Usage

```
## S3 method for class 'hclust'  
reorder(x, dist, method = "OLO", ...)
```

Arguments

x	an object of class hclust.
dist	an object of class dist with dissimilarities between the objects in x.
method	a character string with the name of the used measure Available are: "OLO" (optimal leaf ordering; Bar-Joseph et al., 2001) and "GW" (Gruvaeus and Wainer, 1972).
...	further arguments are currently ignored.

Details

Minimizes the distance between neighboring objects (leaf nodes) in the dendrogram by flipping the order of subtrees. The algorithm by Gruvaeus and Wainer is implemented in package **gclus** (Hurley 2004).

Value

A reordered hclust object.

Author(s)

Michael Hahsler

References

Bar-Joseph, Z., E. D. Demaine, D. K. Gifford, and T. Jaakkola. (2001): Fast Optimal Leaf Ordering for Hierarchical Clustering. *Bioinformatics*, **17**(1), 22–29.

Gruvaeus, G. and Wainer, H. (1972): Two Additions to Hierarchical Cluster Analysis, *British Journal of Mathematical and Statistical Psychology*, **25**, 200–206.

Hurley, Catherine B. (2004): Clustering Visualizations of Multidimensional Data. *Journal of Computational and Graphical Statistics*, **13**(4), 788–806.

See Also

[reorder.hclust](#)

Examples

```
## cluster European cities by distance
data("eurodist")
d <- as.dist(eurodist)
hc <- hclust(eurodist)

## plot original dendrogram and the reordered dendrograms
plot(hc)
plot(reorder(hc, d, method = "GW"))
plot(reorder(hc, d, method = "OLO"))
```

Robinson

*Create and Recognize Robinson and Pre-Robinson Matrices***Description**

Provides functions to create and recognize (anti) Robinson and pre-Robinson matrices. A (anti) Robinson matrix has strictly decreasing (increasing) values when moving away from the main diagonal. A pre-Robinson matrix is a matrix which can be transformed into a perfect Robinson matrix using simultaneous permutations of rows and columns.

Usage

```
is.robinson(x, anti = TRUE, pre = FALSE)
random.robinson(n, anti = TRUE, pre = FALSE, noise = 0)
```

Arguments

<code>x</code>	a symmetric, positive matrix or a dissimilarity matrix (a <code>dist</code> object).
<code>anti</code>	logical; check for anti Robinson structure? Note that for distances, anti Robinson structure is appropriate.
<code>pre</code>	logical; recognize/create pre-Robinson matrices.
<code>n</code>	number of objects.
<code>noise</code>	noise intensity between 0 and 1. Zero means no noise. Noise more than zero results in non-Robinson matrices.

Details

Note that the default matrices are anti Robinson matrices. This is done because distance matrices (the default in R) are typically anti Robinson matrices with values increasing when moving away from the diagonal.

Robinson matrices are recognized using the fact that they have zero anti Robinson events. For pre-Robinson matrices we use spectral seriation first since spectral seriation is guaranteed to perfectly reorder pre-Robinson matrices (see Laurent and Seminaroti, 2015).

Random pre-Robinson matrices are generated by reversing the process of unidimensional scaling. We randomly (uniform distribution with range $[0, 1]$) choose x coordinates for n points on a straight

line and calculate the pairwise distances. For Robinson matrices, the points are sorted first according to x . For noise, y coordinates is added. The coordinates are chosen uniformly between 0 and noise, with noise between 0 and 1.

Value

A single logical value.

References

M. Laurent, M. Seminaroti (2015): The quadratic assignment problem is easy for Robinsonian matrices with Toeplitz structure, *Operations Research Letters* 43(1), 103–109.

Examples

```
## create a perfect anti Robinson structure
m <- random.robinson(10)
pimage(m)

is.robinson(m)

## permute the structure to make it not Robinsonian. However,
## it is still pre-Robinson.
o <- sample(10)
m2 <- permute(m, ser_permutation(o,o))
pimage(m2)

is.robinson(m2)
is.robinson(m2, pre = TRUE)

## create a binary random Robinson matrix (not anti Robinson)
m3 <- random.robinson(10, anti = FALSE) > .7
pimage(m3)
is.robinson(m3, anti = FALSE)

## create matrices with noise (as distance matrices)
m4 <- as.dist(random.robinson(50, pre = FALSE, noise = .1))
pimage(m4)
criterion(m4, method = "AR")

m5 <- as.dist(random.robinson(50, pre = FALSE, noise = .5))
pimage(m5)
criterion(m5, method = "AR")
```

Description

Tries to find an linear order for objects using data in form of a dissimilarity matrix (two-way one mode data), a data matrix (two-way two-mode data) or a data array (k-way k-mode data).

Usage

```
## S3 method for class 'dist'
seriate(x, method = "Spectral", control = NULL, ...)
## S3 method for class 'matrix'
seriate(x, method = "PCA", control = NULL,
        margin = c(1,2), ...)
## S3 method for class 'array'
seriate(x, method = "PCA", control = NULL,
        margin = seq(length(dim(x))), ...)
```

Arguments

x	the data.
method	a character string with the name of the seriation method (default: varies by data type).
control	a list of control options passed on to the seriation algorithm.
margin	a vector giving the margins to be seriated. For matrix, 1 indicates rows, 2 indicates columns, c(1,2) indicates rows and columns. For array, margin gets a vector with the dimensions to seriate.
...	further arguments (unused).

Details

Seriation methods are available via a registry. See [list_seriation_methods](#) for help.

Many seriation methods (heuristically) optimize (minimize or maximize) an objective function. The value of the function for a given seriation can be calculated using [criterion](#). In this manual page we only state the measure which is optimized (using **bold font**). A definition of the measures can be found in the [criterion](#) manual page.

Two-way two-mode data has to be provided as a dist object (not as a symmetric matrix). Similarities have to be transformed in a suitable way into dissimilarities. Currently the following methods are implemented for dist (for a more detailed description and an experimental comparison see Hahsler (2017)):

"ARSA" Anti-Robinson seriation by simulated annealing to minimize the **linear seriation criterion** (simulated annealing initialization used in Brusco et al 2008).

Several control parameters are available: cool (cooling rate), tmin (minimum temperature), swap_to_inversion (proportion of swaps to inversions for local neighborhood search), try_multiplier (local search tries per temperature; multiplied with the number of objects), reps (repeat the algorithm with random initialization), verbose. Use verbose = TRUE to see the default values for the parameters.

"BBURCG" Anti-Robinson seriation by branch-and-bound to minimize the **unweighted gradient measure** (Brusco and Stahl 2005). This is only feasible for a relatively small number of objects.

"BBWRCG" Anti-Robinson seriation by branch-and-bound to minimize the **weighted gradient measure** (Brusco and Stahl 2005). This is only feasible for a relatively small number of objects.

"TSP" Traveling salesperson problem solver to minimize the **Hamiltonian path length**. The solvers in **TSP** are used (see `solve_TSP`). The solver method can be passed on via the `control` argument, e.g. `control = list(method = "two_opt")`. Default is the est of 10 runs of arbitrary insertion heuristic with 2-opt improvement.

Since a tour returned by a TSP solver is a connected circle and we are looking for a path representing a linear order, we need to find the best cutting point. Climer and Zhang (2006) suggest to add a dummy city with equal distance to each other city before generating the tour. The place of this dummy city in an optimal tour with minimal length is the best cutting point (it lies between the most distant cities).

"R2E" Rank-two ellipse seriation (Chen 2002).

This method starts with generating a sequence of correlation matrices R^1, R^2, \dots, R^n is the correlation matrix of the original distance matrix D (supplied to the function as x), and

$$R^{n+1} = \phi R^n,$$

where ϕ calculates the correlation matrix.

The rank of the matrix R^n falls with increasing n . The first R^n in the sequence which has a rank of 2 is found. Projecting all points in this matrix on the first two eigenvectors, all points fall on an ellipse. The order of the points on this ellipse is the resulting order.

The ellipse can be cut at the two interception points (top or bottom) of the vertical axis with the ellipse. In this implementation the top most cutting point is used.

"MDS", "MDS_metric", "MDS_nonmetric", "MDS_angle" Multidimensional scaling (MDS).

Use multidimensional scaling techniques to find an linear order by minimizing **stress**. Note MDS algorithms used for a single dimension tend to end up in local optima and unidimensional scaling (see Maier and De Leeuw, 2015) would be more appropriate. However, generally, ordering along the first component of MDS provides good results.

By default, metric MDS (`cmdscale` in **stats**) is used. In case of of general dissimilarities, non-metric MDS can be used. The choices are `isoMDS` and `sammon` from **MASS**. The method can be specified as the element `method` ("`cmdscale`", "`isoMDS`" or "`sammon`") in `control`.

For convenience, "`MDS_metric`" performs `cmdscale` and "`MDS_nonmetric`" performs `isoMDS`. "`MDS_angle`" projects the data on the first two components found by MDS and then orders by the angle in this space. The order is split by the larges gap between adjacent angles. A similar method was used for ordering correlation matrices by Friendly (2002).

"HC", "HC_single", "HC_complete", "HC_average", "HC_ward" Hierarchical clustering.

Using the order of the leaf nodes in a dendrogram obtained by hierarchical clustering can be used as a very simple seriation technique. This method applies hierarchical clustering (`hclust`) to x . The clustering method can be given using a "`method`" element in the `control` list. If omitted, the default "`average`" is used.

For convenience the other methods are provided as shortcuts.

"GW", "OLO" Hierarchical clustering (by default using average-link) with additional leaf-node re-ordering to minimize **Hamiltonian path length (restricted)**.

A dendrogram (binary tree) has 2^{n-1} internal nodes (subtrees) and the same number of leaf orderings. That is, at each internal node the left and right subtree (or leaves) can be swapped, or, in terms of a dendrogram, be flipped.

Method "`GW`" uses an algorithm developed by Gruvaeus and Wainer (1972) and implemented in package **gclus** (Hurley 2004). The clusters are ordered at each level so that the objects at

the edge of each cluster are adjacent to that object outside the cluster to which it is nearest. The method produces an unique order.

Method "OLO" (Optimal leaf ordering, Bar-Joseph et al., 2001) produces an optimal leaf ordering with respect to the minimizing the sum of the distances along the (Hamiltonian) path connecting the leaves in the given order. The time complexity of the algorithm is $O(n^3)$. Note that non-finite distance values are not allowed.

Both methods start with a dendrogram created by `hclust`. As the "method" element in the control list a clustering method (default "average") can be specified. Alternatively, a `hclust` object can be supplied using an element named "hclust".

For convenience "GW_single", "GW_average", "GW_complete", "GW_ward" and "OLO_single", "OLO_average", "OLO_complete", "OLO_ward" are provided.

"VAT" Visual Assessment of (Clustering) Tendency (Bezdek and Hathaway (2002)).

Creates an order based on Prim's algorithm for finding a minimum spanning tree (MST) in a weighted connected graph representing the distance matrix. The order is given by the order in which the nodes (objects) are added to the MST.

"SA" Simulated Annealing for diverse criterion measures.

Implement simulated annealing similar to the ARSA method, however, it works for any criterion measure defined in **seriation**. By default the algorithm optimizes for raw gradient measure and is warm started with the result of spectral seriation (2-Sum problem) since Hahsler (2017) shows that 2-Sum solutions are similar to solutions for the gradient measure.

Local neighborhood functions are `LS_insert`, `LS_swap`, `LS_reverse`, and `LS_mix` (1/3 insertion, 1/3 swap and 1/3 reverse). Any neighborhood function can be defined. It needs to take as the only argument the order (integer vector) and return a random neighbor.

Note that this is an R implementation repeatedly calling criterion, and therefore is relatively slow.

Several control parameters are available: `criterion` (criterion to optimize; default: "Gradient_raw"), `init` (initial order; default: "Spectral"), `localsearch` (neighborhood function; default: `LS_insert`), `cool` (cooling rate), `tmin` (minimum temperature), `swap_to_inversion` (proportion of swaps to inversions), `nlocal` (number of objects times `nlocal` is the number of search tries per temperature), `verbose`. Use `verbose = TRUE` to see the default values for the parameters.

"Spectral", "Spectral_norm" Spectral seriation (Ding and He 2004).

Spectral seriation uses a relaxation to minimize the **2-Sum Problem** (Barnard, Pothen, and Simon 1993). It uses the order of the Fiedler vector of the similarity matrix's (normalized) Laplacian.

Spectral seriation gives a good trade-off between seriation quality, speed and scalability (see Hahsler, 2017).

"SPIN_NH", "SPIN_STS" Sorting Points Into Neighborhoods (SPIN) (Tsafriir 2005). Given a weight matrix W , the algorithms try to minimize the energy for a permutation (matrix P) given by

$$F(P) = \text{tr}(PDP^TW),$$

where tr denotes the matrix trace.

"SPIN_STS" implements the Side-to-Side algorithm which tries to push out large distance values. The default weight matrix suggested in the paper with $W = XX^T$ and $X_i = i - (n + 1)/2$ is used. We run the algorithm from step (25) iteration and restart the algorithm

nstart (10) with random initial permutations (default values in parentheses). Via control the parameters step, nstart, X and verbose.

"SPIN_NH" implements the neighborhood algorithm (concentrate low distance values around the diagonal) with a Gaussian weight matrix $W_{ij} = \exp(-(i - j)^2/n\sigma)$, where n is the size of the dissimilarity matrix and σ is the variance around the diagonal that control the influence of global (large σ) or local (small σ) structure.

We use the heuristic suggested in the paper for the linear assignment problem. We do not terminate as indicated in the algorithm, but run all the iterations since the heuristic does not guarantee that the energy is strictly decreasing. We also implement the heuristic "annealing" scheme where σ is successively reduced. The parameters in control are sigma which can be a single value or a decreasing sequence (default: 20 to 1 in 10 steps) and step which defines how many update steps are performed before for each value of alpha. Via W_function a custom function to create W with the function signature function(n, sigma, verbose) can be specified. The parameter verbose can be used to display progress information.

"QAP_LS", "QAP_2SUM", "QAP_BAR", "QAP_Inertia" Quadratic assignment problem formulations for seriation using a simulated annealing solver. These methods minimize the **Linear Seriation Problem** (LS) formulation (Hubert and Schultz 1976), the **2-Sum Problem** formulation (Barnard, Pothen, and Simon 1993), the **banded anti-Robinson form** (BAR) or the **inertia criterion**.

The parameters in control are passed on to qap in **qap**. An important parameter is rep to return the best result out of the given number of repetitions with random restarts. Default is 1, but bigger numbers result in better and more stable results.

"GA" Use a genetic algorithm to optimize for various criteria. The GA code has to be first registered. A detailed description can be found in the manual page for [register_GA](#).

"DendSer" Use heuristic dendrogram seriation to optimize for various criteria. The DendSer code has to be first registered. A detailed description can be found in the manual page for [register_DendSer](#).

"Identity" Produces an identity permutation.

"Random" Produces a random permutation.

Two-way two mode data are general positive matrices. Currently the following methods are implemented for matrix:

"BEA" Bond Energy Algorithm (BEA; McCormick 1972). The algorithm tries to maximize the **Measure of Effectiveness** of a non-negative matrix. Due to the definition of this measure, the tasks of ordering rows and columns is separable and can be solved independently.

A row is arbitrarily placed; then rows are positioned one by one. When this is completed, the columns are treated similarly. The overall procedure amounts to two approximate traveling salesperson problems (TSP), one on the rows and one on the columns. The so-called 'best insertion strategy' is used: rows (or columns) are inserted into the current permuted list of rows (or columns). Several consecutive runs of the algorithm might improve the energy.

Note that Arabie and Hubert (1990) question its use with non-binary data if the objective is to find a seriation or one-dimensional orderings of rows and columns.

The BEA code used in this package was implemented by Fionn Murtagh.

In control as element "rep" the number of runs can be specified. The results of the best run will be returned.

"BEA_TSP" Use a TSP to optimize the **Measure of Effectiveness** (Lenstra 1974).

In control as element "method" a TSP solver method can be specified (see package **TSP**).

"PCA", "PCA_angle" Principal component analysis.

Uses the projection of the data on its first principal component to determine the order.

Note that for a distance matrix calculated from x with Euclidean distance, this methods minimizes the least square criterion.

"PCA_angle" projects the data on the first two principal components and then orders by the angle in this space. The order is split by the largest gap between adjacent angles. A similar method was used for ordering correlation matrices by Friendly (2002).

"Identity" Produces an identity permutation.

"Random" Produces a random permutation.

For array no built-in methods are currently available.

Value

Returns an object of class `ser_permutation`.

Author(s)

Michael Hahsler

References

- Arabic, P. and L.J. Hubert (1990): The bond energy algorithm revisited, *IEEE Transactions on Systems, Man, and Cybernetics*, **20**(1), 268–274.
- Bar-Joseph, Z., E. D. Demaine, D. K. Gifford, and T. Jaakkola. (2001): Fast Optimal Leaf Ordering for Hierarchical Clustering. *Bioinformatics*, **17**(1), 22–29.
- Barnard, S. T., A. Pothen, and H. D. Simon (1993): A Spectral Algorithm for Envelope Reduction of Sparse Matrices. *In Proceedings of the 1993 ACM/IEEE Conference on Supercomputing*, 493–502. Supercomputing '93. New York, NY, USA: ACM.
- Bezdek, J.C. and Hathaway, R.J. (2002): VAT: a tool for visual assessment of (cluster) tendency. *Proceedings of the 2002 International Joint Conference on Neural Networks (IJCNN '02)*, Volume: 3, 2225–2230.
- Brusco, M., Koehn, H.F., and Stahl, S. (2008): Heuristic Implementation of Dynamic Programming for Matrix Permutation Problems in Combinatorial Data Analysis. *Psychometrika*, **73**(3), 503–522.
- Brusco, M., and Stahl, S. (2005): *Branch-and-Bound Applications in Combinatorial Data Analysis*. New York: Springer.
- Chen, C. H. (2002): Generalized Association Plots: Information Visualization via Iteratively Generated Correlation Matrices. *Statistica Sinica*, **12**(1), 7–29.
- Ding, C. and Xiaofeng He (2004): Linearized cluster assignment via spectral ordering. *Proceedings of the Twenty-first International Conference on Machine Learning (ICML '04)*.
- Climer, S. and Xiongnu Zhang (2006): Rearrangement Clustering: Pitfalls, Remedies, and Applications, *Journal of Machine Learning Research*, **7**(Jun), 919–943.

- Friendly, M. (2002): Corrgrams: Exploratory Displays for Correlation Matrices. *The American Statistician*, **56**(4), 316–324.
- Gruvaeus, G. and Wainer, H. (1972): Two Additions to Hierarchical Cluster Analysis, *British Journal of Mathematical and Statistical Psychology*, **25**, 200–206.
- Hahsler, M. (2017): An experimental comparison of seriation methods for one-mode two-way data. *European Journal of Operational Research*, **257**, 133–143.
- Hubert, Lawrence, and James Schultz (1976): Quadratic Assignment as a General Data Analysis Strategy. *British Journal of Mathematical and Statistical Psychology* **29**(2). Blackwell Publishing Ltd. 190–241.
- Hurley, Catherine B. (2004): Clustering Visualizations of Multidimensional Data. *Journal of Computational and Graphical Statistics*, **13**(4), 788–806.
- Lenstra, J.K (1974): Clustering a Data Array and the Traveling-Salesman Problem, *Operations Research*, **22**(2) 413–414.
- Mair P., De Leeuw J. (2015). Unidimensional scaling. In *Wiley StatsRef: Statistics Reference Online*, Wiley, New York.
- McCormick, W.T., P.J. Schweitzer and T.W. White (1972): Problem decomposition and data reorganization by a clustering technique, *Operations Research*, **20**(5), 993–1009.
- Tsafirir, D., Tsafirir, I., Ein-Dor, L., Zuk, O., Notterman, D.A. and Domany, E. (2005): Sorting points into neighborhoods (SPIN): data analysis and visualization by ordering distance matrices, *Bioinformatics*, **21**(10) 2301–8.

See Also

[list_seriation_methods](#), [criterion](#), [register_GA](#), [register_DendSer](#), [solve_TSP](#) in **TSP**, [hclust](#) in **stats**.

Examples

```
## show available seriation methods (for dist and matrix)
show_seriation_methods("dist")
show_seriation_methods("matrix")

##seriate dist
data("iris")
x <- as.matrix(iris[-5])
x <- x[sample(1:nrow(x)),]
d <- dist(x)

## default seriation
order <- seriate(d)
order

## plot
pimage(d, main = "Random")
pimage(d, order, main = "Reordered")

## compare quality
rbind(
```

```
        random = criterion(d),
        reordered = criterion(d, order)
    )

## seriate matrix
data("iris")
x <- as.matrix(iris[-5])

## to make the variables comparable, we scale the data
x <- scale(x, center = FALSE)

## try some methods
pimage(x, main = "original data")
criterion(x)

order <- seriate(x, method = "BEA_TSP")
pimage(x, order, main = "TSP to optimize ME")
criterion(x, order)

order <- seriate(x, method = "PCA")
pimage(x, order, main = "First principal component")
criterion(x, order)

## 2 TSPs
order <- c(
  seriate(dist(x), method = "TSP"),
  seriate(dist(t(x)), method = "TSP")
)
pimage(x, order, main = "2 TSPs")
criterion(x, order)
```

seriation_data

Create Simulated Data for Seriation Evaluation

Description

Several functions to create simulated data to evaluate different aspects of seriation algorithms and criterion functions.

Usage

```
create_lines_data(n = 250)
create_ordered_data(n = 250, k = 2, size = NULL, spacing = 6, path = "linear",
  sd1 = 1, sd2 = 0)
```

Arguments

n number of data points to create.
k number of Gaussian components.

size	relative size (number of points) of components (length of k). If NULL then all components have the same size.
spacing	space between the centers of components. The default of 6 means that the components will barely touch at ds1=1 (3 standard deviations for each Gaussian component).
path	Are the components arranged along a "linear" or "circular" path?
sd1	variation in the direction along the components. A value greater than one means the components are mixing.
sd2	variation perpendicular to the direction along the components. A value greater than 0 will introduce anti-Robinson violation events.

Details

create_lines_data creates the lines data set used in for iVAT in Havens and Bezdek (2012).

create_ordered_data is a versatile function which creates "orderable" 2D data using Gaussian components along a linear or circular path. The components are equally spaced (spacing) along the path. The default spacing of 6 ensures that 2 adjacent components with a standard deviation of one along the direction of the path will barely touch. The standard deviation along the path is set by sd1. The standard deviation perpendicular to the path is set by sd2. A value larger than zero will result in the data not being perfectly orderable (i.e., the resulting distance matrix will not be a perfect pre-anti-Robinson matrix and contain anti-Robinson violation events after seriation). Note that a circular path always creates anti-Robinson violation since the circle has to be broken at some point to create a linear order.

Author(s)

Michael Hahsler

References

Havens, T.C. and Bezdek, J.C. (2012): An Efficient Formulation of the Improved Visual Assessment of Cluster Tendency (iVAT) Algorithm, *IEEE Transactions on Knowledge and Data Engineering*, **24**(5), 813–822.

See Also

[seriate](#), [criterion](#), [VAT](#).

Examples

```
## lines data set from Havens and Bezdek (2011)
x <- create_lines_data(250)
plot(x, xlim=c(-5,5), ylim=c(-3,3), cex=.2, col = attr(x, "id"))
d <- dist(x)
pimage(d, seriate(d, "OLO_single"), col = bluered(100, bias=.5), key = TRUE)

## create_ordered_data can produce many types of "orderable" data

## perfect pre-Anti-Robinson matrix (with a single components)
```

```

x <- create_ordered_data(250, k = 1)
plot(x, cex=.2, col = attr(x, "id"))
d <- dist(x)
pimage(d, seriate(d, "MDS"), col = bluered(100, bias=.5), key = TRUE)

## separated components
x <- create_ordered_data(250, k = 5)
plot(x, cex=.2, col = attr(x, "id"))
d <- dist(x)
pimage(d, seriate(d, "MDS"), col = bluered(100, bias=.5), key = TRUE)

## overlapping components
x <- create_ordered_data(250, k = 5, sd1 = 2)
plot(x, cex=.2, col = attr(x, "id"))
d <- dist(x)
pimage(d, seriate(d, "MDS"), col = bluered(100, bias=.5), key = TRUE)

## introduce anti-Robinson violations (a non-zero y value)
x <- create_ordered_data(250, k = 5, sd1 = 2, sd2 = 5)
plot(x, cex=.2, col = attr(x, "id"))
d <- dist(x)
pimage(d, seriate(d, "MDS"), col = bluered(100, bias=.5), key = TRUE)

## circular path (has always violations)
x <- create_ordered_data(250, k = 5, path = "circular", sd1=2)
plot(x, cex=.2, col = attr(x, "id"))
d <- dist(x)
pimage(d, seriate(d, "OLO"), col = bluered(100, bias=.5), key = TRUE)

## circular path (with more violations violations)
x <- create_ordered_data(250, k = 5, path = "circular", sd1=2, sd2=1)
plot(x, cex=.2, col = attr(x, "id"))
d <- dist(x)
pimage(d, seriate(d, "OLO"), col = bluered(100, bias=.5), key = TRUE)

```

seriation_methods

Registry for Seriation Methods

Description

A registry to manage methods for seriation.

Usage

```

list_seriation_methods(kind)
show_seriation_methods(kind)
get_seriation_method(kind, name)
set_seriation_method(kind, name, definition, description = NULL, ...)

```


Arguments

kind	the data type the method works on. For example, "dist", "matrix" or "array".
name	a short name for the method used to refer to the method in <code>seriate()</code> .
definition	a function containing the method's code.
description	a description of the method. For example, a long name.
...	further information that is stored for the method in the registry.

Details

The functions below are convenience function for the registry `registry_seriate`.

`list_seriation_method()` lists all available methods for a given data type (kind). The result is a vector of character strings with the short names of the methods.

`show_seriation_method()` shows all available methods including a description.

`get_seriation_method()` returns information (including the implementing function) about a given method in form of an object of class "seriation_method".

With `set_seriation_method()` new seriation methods can be added by the user. The implementing function (definition) needs to have the formal arguments `x`, `control`, where `x` is the data object and `control` contains a list with additional information for the method passed on from `seriate()`. The implementation has to return a list of objects which can be coerced into `ser_permutation_vector` objects (e.g., integer vectors). The elements in the list have to be in corresponding order to the dimensions of `x`.

Author(s)

Michael Hahsler

Examples

```
## registry
registry_seriate

## convenience functions
show_seriation_methods("matrix")

list_seriation_methods("matrix")

get_seriation_method("matrix", "BEA")

## define a new method

## create a identity function which returns the identity order
seriation_method_identity <- function(x, control) {
  lapply(dim(x), seq)
}

## set new method
set_seriation_method("matrix", "identity", seriation_method_identity,
  "Identity order")
```

```

set_seriation_method("array", "identity", seriation_method_identity,
  "Identity order")

show_seriation_methods("matrix")

##use all criterion methods (including the new one)
seriate(matrix(1:12, ncol=3), "identity")

```

SupremeCourt

Voting Patterns in the Second Rehnquist U.S. Supreme Court

Description

Contains a (a subset of the) decisions for the stable 8-yr period 1995-2002 of the second Rehnquist Supreme Court. Decisions are aggregated to the joint probability for disagreement between judges.

Usage

```
data("SupremeCourt")
```

Format

A square, symmetric 9-by-9 matrix with the joint probability for disagreement.

Author(s)

Michael Hahsler

References

Sirovich, L. (2003). A pattern analysis of the second Rehnquist U.S. Supreme Court. *Proceedings of the National Academy of Sciences of the United States of America*, 100, 7432-7437. doi:10.1073/pnas.1132164100.

Examples

```

data("SupremeCourt")

SupremeCourt

d <- as.dist(SupremeCourt)
o <- seriate(d)

pimage(d, o)

plot(hclust(as.dist(SupremeCourt)))

```

Townships

Bertin's Characteristics of Townships

Description

This data contains nine characteristics for 16 townships. The data set was used by Bertin (1981) to illustrate that the conciseness of presentation can be improved by seriating the rows and columns.

Usage

```
data("Townships")
```

Format

A matrix with 16 0-1 variables (columns) indicating the presence (1) or absence (0) of characteristics of townships (rows).

Author(s)

Michael Hahsler

References

Bertin, J. (1981): *Graphics and Graphic Information Processing*. Berlin, Walter de Gruyter.

Examples

```
data("Townships")

## original data
pimage(Townships)
criterion(Townships)

## seriated data
order <- seriate(Townships, method = "BEA", control = list(rep = 5))
pimage(Townships, order)
criterion(Townships, order)
```

`uniscale`*Unidimensional Scaling from Seriation Results*

Description

Performs (approximate) unidimensional scaling by first performing seriation to obtain a permutation and then using the permutation to calculate the configuration.

Usage

```
uniscale(d, order = NULL, method = "QAP_LS", rep = 10, ...)
```

Arguments

<code>d</code>	a dissimilarity matrix.
<code>order</code>	a precomputed permutation (configuration) order. If <code>NULL</code> , then seriation is performed using the method specified in <code>method</code> .
<code>method</code>	seriation method used if <code>o</code> is <code>NULL</code> .
<code>rep</code>	Number of repetitions of the seriation heuristic.
<code>...</code>	additional arguments are passed on to the seriation method.

Details

Uses the method described in Maier and De Leeuw (2015) to calculate the minimum stress configuration for either a given configuration/permutation/order or for a permutation computed via a seriation method.

The code is similar to `uniscale` in **smacof**, but scales to larger datasets since it does not check all permutations.

Value

A vector with the fitted configuration.

Author(s)

Michael Hahsler with code from Patrick Mair (from **smacof**).

References

Mair P., De Leeuw J. (2015). Unidimensional scaling. In *Wiley StatsRef: Statistics Reference Online*, Wiley, New York.

See Also

[uniscale](#) in **smacof**.

Examples

```
data(SupremeCourt)

d <- as.dist(SupremeCourt)

sc <- uniscale(d)
sc

orderplot(sc)
```

VAT

Visual Analysis for Cluster Tendency Assessment (VAT/iVAT)

Description

Implements Visual Analysis for Cluster Tendency Assessment (VAT; Bezdek and Hathaway, 2002) and Improved Visual Analysis for Cluster Tendency Assessment (iVAT; Wang et al, 2010).

Usage

```
VAT(x, ...)
iVAT(x, ...)
path_dist(x)
```

Arguments

```
x          a dist object.
...        further arguments are passed on to pimage.
```

Details

`path_dist` redefines the distance between two objects as the minimum over the largest distances in all possible paths between the objects as used for iVAT.

Author(s)

Michael Hahsler

References

- Bezdek, J.C. and Hathaway, R.J. (2002): VAT: a tool for visual assessment of (cluster) tendency. *Proceedings of the 2002 International Joint Conference on Neural Networks (IJCNN '02)*, Volume: 3, 2225–2230.
- Havens, T.C. and Bezdek, J.C. (2012): An Efficient Formulation of the Improved Visual Assessment of Cluster Tendency (iVAT) Algorithm, *IEEE Transactions on Knowledge and Data Engineering*, **24**(5), 813–822.
- Wang L., U.T.V. Nguyen, J.C. Bezdek, C.A. Leckie and K. Ramamohanarao (2010): iVAT and aVAT: Enhanced Visual Analysis for Cluster Tendency Assessment, *Proceedings of the PAKDD 2010, Part I, LNAI 6118*, 16–27.

See Also

[seriate](#), [pimage](#), [create_lines_data](#).

Examples

```
## lines data set from Havens and Bezdek (2011)
x <- create_lines_data(250)
plot(x, xlim=c(-5,5), ylim=c(-3,3), cex=.2)
d <- dist(x)

## create regular VAT
VAT(d, colorkey = TRUE, main = "VAT")
## same as: pimage(d, seriate(d, "VAT"))

## create iVAT which shows visually the three lines
iVAT(d, main = "iVAT")
## same as:
## d_path <- path_dist(d)
## pimage(d_path, seriate(d_path, "VAT"))

## compare with dissplo (shows banded structures and relationship between
## center line and the two outer lines)
disssplot(d, method="OLO_single", main = "Disssplot", col = bluered(100, bias = .5))

## compare with optimally reordered heatmap
hmap(d, method="OLO_single", main = "Heat map (opt. leaf ordering)",
col = bluered(100, bias = .5))
```

Wood

Gene Expression Data for Wood Formation in Poplar Trees

Description

A data matrix containing a sample of the normalized gene expression data for 6 locations in the stem of Poplar trees published in the study by Herzberg et al (2001). The sample of 136 genes selected by Caraux and Pinloche (2005).

Usage

```
data(Wood)
```

Format

The format is a 136 x 6 matrix.

Source

The data was obtained from <http://www.atgc-montpellier.fr/permutmatrix/manual/Exemples/Wood/Wood.htm>.

References

Hertzberg M., H. Aspeborg, J. Schrader, A. Andersson, R. Erlandsson, K. Blomqvist, R. Bhalerao, M. Uhlen, T. T. Teeri, J. Lundeberg, Bjoern Sundberg, P. Nilsson and Goeran Sandberg (2001): A transcriptional roadmap to wood formation, *PNAS*, **98**(25), 14732–14737.

Caraux G. and Pinloche S. (2005): PermutMatrix: a graphical environment to arrange gene expression profiles in optimal linear order, *Bioinformatics*, **21**(7) 1280–1281.

Examples

```
data(Wood)
head(Wood)
```

Zoo

Zoo Data Set

Description

A database containing characteristics of different animals. The database was created and donated by Richard S. Forsyth and is available from the UCI Machine Learning Repository (Newman et al, 1998).

Usage

```
data("Zoo")
```

Format

A data frame with 101 observations on the following 17 variables.

hair {0, 1}

feathers {0, 1}

eggs {0, 1}

milk {0, 1}

airborne {0, 1}

aquatic {0, 1}

predator {0, 1}

toothed {0, 1}

backbone {0, 1}

breathes {0, 1}

venomous {0, 1}

fins {0, 1}

legs Numeric (set of values: {0, 2, 4, 5, 6, 8})

tail {0, 1}

domestic {0, 1}

catsize {0, 1}

class a factor with levels amphibian bird fish insect invertebrate mammal reptile

Source

D.J. Newman, S. Hettich, C.L. Blake and C.J. Merz (1998): UCI Repository of machine learning databases, <http://www.ics.uci.edu/~mllearn/MLRepository.html>, University of California, Irvine, Dept. of Information and Computer Sciences.

Examples

```
data("Zoo")  
x <- scale(Zoo[, -17])
```

```
d <- dist(x)  
pimage(d)
```

```
order <- seriate(d, method = "tsp")  
pimage(d, order)
```


Index

- *Topic **classes**
 - permutation, 26
 - permutation_vector, 28
- *Topic **cluster**
 - bertinplot, 2
 - criterion, 7
 - dissimilarity, 14
 - dissplot, 16
 - register_DendSer, 34
 - register_GA, 35
 - reorder.hclust, 37
 - seriate, 39
 - VAT, 53
- *Topic **datasets**
 - Chameleon, 4
 - Irish, 23
 - Munsingen, 24
 - Psych24, 33
 - SupremeCourt, 50
 - Townships, 51
 - Wood, 54
 - Zoo, 55
- *Topic **data**
 - seriation_data, 46
- *Topic **hplot**
 - bertinplot, 2
 - color_palettes, 5
 - dissplot, 16
 - hmap, 21
 - pimage, 30
- *Topic **manip**
 - get_order, 20
 - permutation_matrix, 27
 - permute, 29
 - VAT, 53
- *Topic **misc**
 - criterion_methods, 12
 - seriation_methods, 48
- *Topic **optimize**
 - register_DendSer, 34
 - register_GA, 35
 - reorder.hclust, 37
 - seriate, 39
 - uniscale, 52
- bertin_cut_line (bertinplot), 2
- bertinplot, 2
- bluered (color_palettes), 5
- c.ser_permutation (permutation), 26
- Chameleon, 4
- chameleon (Chameleon), 4
- chameleon_ds4 (Chameleon), 4
- chameleon_ds5 (Chameleon), 4
- chameleon_ds7 (Chameleon), 4
- chameleon_ds8 (Chameleon), 4
- color_palettes, 5
- colorRampPalette, 6
- create_lines_data, 54
- create_lines_data (seriation_data), 46
- create_ordered_data (seriation_data), 46
- criterion, 7, 40, 45, 47
- criterion_methods, 12
- DendSer, 35
- DendSer (register_DendSer), 34
- dendser (register_DendSer), 34
- dissimilarity, 14
- dissplot, 6, 16, 23, 32
- dist, 19, 30
- dist (dissimilarity), 14
- GA (register_GA), 35
- ga, 36
- ga (register_GA), 35
- gaperm_mixedMutation (register_GA), 35
- get_criterion_method (criterion_methods), 12
- get_method (permutation_vector), 28

- get_order, [20, 26, 28](#)
- get_permutation_matrix, [26, 28](#)
- get_permutation_matrix (get_order), [20](#)
- get_rank, [28](#)
- get_rank (get_order), [20](#)
- get_seriation_method
 - (seriation_methods), [48](#)
- grays (color_palettes), [5](#)
- greenred (color_palettes), [5](#)
- greys (color_palettes), [5](#)

- hclust, [45](#)
- heatmap.2, [23](#)
- hmap, [6, 19, 21, 32](#)

- image, [32](#)
- Irish, [23](#)
- is.robinson (Robinson), [38](#)
- iVAT (VAT), [53](#)

- length.ser_permutation_vector
 - (permutation_vector), [28](#)
- list_criterion_methods, [12](#)
- list_criterion_methods
 - (criterion_methods), [12](#)
- list_seriation_methods, [40, 45](#)
- list_seriation_methods
 - (seriation_methods), [48](#)
- LS_insert (seriate), [39](#)
- LS_mixed (seriate), [39](#)
- LS_reverse (seriate), [39](#)
- LS_swap (seriate), [39](#)

- Munsingen, [24](#)

- orderplot (uniscale), [52](#)

- panel.bars (bertinplot), [2](#)
- panel.blocks (bertinplot), [2](#)
- panel.circles (bertinplot), [2](#)
- panel.lines (bertinplot), [2](#)
- panel.squares (bertinplot), [2](#)
- path_dist (VAT), [53](#)
- permutation, [26](#)
- permutation_matrix, [27](#)
- permutation_matrix2vector
 - (permutation_matrix), [27](#)
- permutation_vector, [28](#)
- permutation_vector2matrix, [28](#)

- permutation_vector2matrix
 - (permutation_matrix), [27](#)
- permute, [27, 29](#)
- pimage, [6, 19, 23, 30, 54](#)
- plot.reordered_cluster_dissimilarity_matrix
 - (dissplot), [16](#)
- print.reordered_cluster_dissimilarity_matrix
 - (dissplot), [16](#)
- print.ser_permutation (permutation), [26](#)
- print.ser_permutation_vector
 - (permutation_vector), [28](#)
- Psych24, [33](#)

- random.robinson (Robinson), [38](#)
- register_DendSer, [34, 43, 45](#)
- register_GA, [35, 43, 45](#)
- registry_criterion (criterion_methods), [12](#)
- registry_seriate (seriation_methods), [48](#)
- reorder (reorder.hclust), [37](#)
- reorder.hclust, [37, 37](#)
- rev.ser_permutation_vector
 - (permutation_vector), [28](#)
- Robinson, [38](#)

- ser_align (dissimilarity), [14](#)
- ser_cor (dissimilarity), [14](#)
- ser_dist (dissimilarity), [14](#)
- ser_permutation, [4, 21, 27, 28, 30](#)
- ser_permutation (permutation), [26](#)
- ser_permutation_vector, [15, 21, 26](#)
- ser_permutation_vector
 - (permutation_vector), [28](#)
- seriate, [4, 19, 23, 32, 35, 36, 39, 47, 54](#)
- seriation_data, [46](#)
- seriation_methods, [48](#)
- set_criterion_method
 - (criterion_methods), [12](#)
- set_seriation_method
 - (seriation_methods), [48](#)
- show_criterion_methods
 - (criterion_methods), [12](#)
- show_seriation_methods
 - (seriation_methods), [48](#)
- solve_TSP, [45](#)
- SupremeCourt, [50](#)

- Townships, [51](#)
- uniscale, [52, 52](#)

INDEX

59

VAT, [47](#), [53](#)

Wood, [54](#)

Zoo, [55](#)