Package 'blockmodeling'

July 22, 2025

Type Package

Title Generalized and Classical Blockmodeling of Valued Networks

Version 1.1.5

Date 2023-08-23

Imports stats, methods, Matrix, parallel

Suggests sna, doRNG, doParallel, foreach

Depends R (>= 2.10)

Maintainer Aleš Žiberna <ales.ziberna@gmail.com>

Description

This is primarily meant as an implementation of generalized blockmodeling for valued networks. In addition, measures of similarity or dissimilarity based on structural equivalence and regular equivalence (REGE algorithms) can be computed and partitioned matrices can be plotted: Žiberna (2007)<doi:10.1016/j.socnet.2006.04.002>, Žiberna (2008)<doi:10.1080/00222500701790207>, Žiberna (2014)<doi:10.1016/j.socnet.2014.04.002>.

License GPL (>= 2)

Encoding UTF-8

NeedsCompilation yes

Author Aleš Žiberna [aut, cre],
Marjan Cugmas [ctb]

Repository CRAN

Date/Publication 2023-08-23 15:30:02 UTC

Contents

`baker` *Citation data between social work journals for the 1985-86 period*

Description

This example consists of the citation data between social work journals for the 1985-86 period, collected and analyzed in Baker (1992)

Usage

```
data(baker)
```

Format

An object of class `matrix` (inherits from `array`) with 20 rows and 20 columns.

References

Baker, D. R. (1992). A Structural Analysis of Social Work Journal Network: 1985-1986. Journal of Social Service Research, 15(3-4), 153-168. doi: 10.1300/J079v15n03_09

Examples

```
# data(baker)
# Transforming it to matrix format
# baker <- as.matrix(baker)
# putting zeros on the diagonal
# diag(baker) <- 0
```

`blockmodeling` *An R package for Generalized and classical blockmodeling of valued networks*

Description

This package is primarily meant as an implementation of Generalized blockmodeling. In addition, functions for computation of (dis)similarities in terms of structural and regular equivalence, plotting and other "utility" functions are provided.

Author(s)

Aleš Žiberna

References

Doreian, P., Batagelj, V., & Ferligoj, A. (2005). Generalized blockmodeling, (Structural analysis in the social sciences, 25). Cambridge [etc.]: Cambridge University Press.

Žiberna, A. (2007). Generalized Blockmodeling of Valued Networks. Social Networks, 29(1), 105-126. doi: 10.1016/j.socnet.2006.04.002

Žiberna, A. (2008). Direct and indirect approaches to blockmodeling of valued networks in terms of regular equivalence. Journal of Mathematical Sociology, 32(1), 57-84. doi: 10.1080/00222500701790207

Žiberna, A. (2014). Blockmodeling of multilevel networks. Social Networks, 39(1), 46-61. doi: 10.1016/j.socnet.2014.04.002

See Also

`optRandomParC`, `critFunC`, `optParC`, `IM`, `clu`, `err`, `plotMat`

Examples

```
#Generating a simple network corresponding to the simple Sum of squares
# Structural equivalence with blockmodel:
# nul com
# nul nul
n <- 20
net <- matrix(NA, ncol = n, nrow = n)
clu <- rep(1:2, times = c(5, 15))
tclu <- table(clu)
net[clu == 1, clu == 1] <- rnorm(n = tclu[1] * tclu[1], mean = 0, sd = 1)
```

```
net[clu == 1, clu == 2] <- rnorm(n = tclu[1] * tclu[2], mean = 4, sd = 1)
net[clu == 2, clu == 1] <- rnorm(n = tclu[2] * tclu[1], mean = 0, sd = 1)
net[clu == 2, clu == 2] <- rnorm(n = tclu[2] * tclu[2], mean = 0, sd = 1)

# Computation of criterion function with the correct partition
res <- critFunC(M = net, clu = clu, approaches = "hom", homFun = "ss", blocks = "com")
res$err # The error is relatively small
plot(res)

# Computation of criterion function with the correct partition and correct pre-specified blockmodel
# Prespecified blockmodel used
# nul com
# nul nul
B <- array(NA, dim = c(1, 1, 2, 2))
B[1, 1, , ] <- "nul"
B[1, 1, 1, 2] <- "com"
B[1, 1, , ]
res <- critFunC(M = net, clu = clu, approaches = "hom", homFun = "ss", blocks = B)
err(res) # The error is relatively small
IM(res)
plot(res)

# Computation of criterion function with the correct partition
# and pre-specified blockmodel with some alternatives
# Prespecified blockmodel used
# nul nul|com
# nul nul
B <- array(NA, dim = c(2, 2, 2))
B[1, , ] <- "nul"
B[2, 1, 2] <- "com"
res <- critFunC(M = net, clu = clu, approaches = "hom", homFun = "ss", blocks = B)
err(res) # The error is relatively small
IM(res)
plot(res)


# Optimizing a very bad partition
cluStart <- rep(1:2, times = 10)
res <- optParC(M = net,
               clu = cluStart,
               approaches = "hom", homFun = "ss", blocks = "com")
clu(res) # Hopefully we get the original partition)
err(res)
plot(res)

# Optimizing 10 random chosen partitions with optRandomParC
res <- optRandomParC(M = net, k = 2, rep = 10,
approaches = "hom", homFun = "ss", blocks = "com")
clu(res) # Hopefully we get the original partition)
err(res)
plot(res)

# Adapt network for Valued blockmodeling with the same model
```

```
net[net > 4] <- 4
net[net < 0] <- 0

# Computation of criterion function with the correct partition
res <- critFunC(M = net, clu = clu, approaches = "val",
                blocks = c("nul", "com"), preSpecM = 4)
err(res) # The error is relatively small
IM(res)
# The image corresponds to the one used for generation of
# The network
plot(res)
```

---

canClu *Create canonical partition and find unique canonical partitions in a list of partitions.*

Description

It is used to convert any partition to a canonical partition. A canonical partition is a partition where the first unit is in cluster 1, the next unit that is not in cluster 1 in in cluster 2 and so on. So if we would take first appearances of clusters in the order they appear in the partition vector, we would get integers from 1 to the number of clusters.

Usage

```
canClu(clu)

canCluUniqe(cluList)
```

Arguments

clu	A partition - a vector or a list of vectors/partitions.
cluList	A list of partitions(vectors).

Value

For function canClu - a canonical partition or a list of such partitions. For function canCluUniqe - A list of unique canonical partitions.

See Also

clu

Examples

```
clu<-c(3,2,2,3,1,2)
canClu(clu)
```

clu *Function for extraction of some elements for objects, returend by functions for Generalized blockmodeling*

Description

Functions for extraction of partition (clu), all best partitions (partitions), image or blockmodel (IM)) and total error or inconsistency (err) for objects, returned by functions critFunC or optRandomParC.

Usage

```
clu(res, which = 1, ...)

partitions(res)

err(res, ...)

IM(res, which = 1, drop = TRUE, ...)

EM(res, which = 1, drop = TRUE, ...)
```

Arguments

res	Result of function critFunC or optRandomParC.
which	From which (if there are more than one) "best" solution should the element be extracted. Warning! which grater than the number of "best" partitions produces an error.
...	Not used.
drop	If TRUE (default), dimensions that have only one level are dropped (drop function is applied to the final result).

Value

The desired element.

Author(s)

Aleš Žiberna

References

Doreian, P., Batagelj, V., & Ferligoj, A. (2005). Generalized blockmodeling, (Structural analysis in the social sciences, 25). Cambridge [etc.]: Cambridge University Press.

Žiberna, A. (2007). Generalized Blockmodeling of Valued Networks. Social Networks, 29(1), 105-126. doi: 10.1016/j.socnet.2006.04.002

Žiberna, A. (2008). Direct and indirect approaches to blockmodeling of valued networks in terms of regular equivalence. Journal of Mathematical Sociology, 32(1), 57-84. doi: 10.1080/00222500701790207

See Also

critFunC, plot.mat, optRandomParC

Examples

```
n <- 8 # If larger, the number of partitions increases dramatically,
# as does if we increase the number of clusters
net <- matrix(NA, ncol = n, nrow = n)
clu <- rep(1:2, times = c(3, 5))
tclu <- table(clu)
net[clu == 1, clu == 1] <- rnorm(n = tclu[1] * tclu[1], mean = 0, sd = 1)
net[clu == 1, clu == 2] <- rnorm(n = tclu[1] * tclu[2], mean = 4, sd = 1)
net[clu == 2, clu == 1] <- rnorm(n = tclu[2] * tclu[1], mean = 0, sd = 1)
net[clu == 2, clu == 2] <- rnorm(n = tclu[2] * tclu[2], mean = 0, sd = 1)

# We select a random partition and then optimize it
all.par <- nkpartitions(n = n, k = length(tclu))
# Forming the partitions
all.par <- lapply(apply(all.par, 1, list),function(x) x[[1]])
# to make a list out of the matrix
res <- optParC(M = net,
   clu = all.par[[sample(1:length(all.par), size = 1)]],
    approaches = "hom", homFun = "ss", blocks = "com")
plot(res) # Hopefully we get the original partition
clu(res) # Hopefully we get the original partition
err(res) # Error
IM(res) # Image matrix/array.
EM(res) # Error matrix/array.
```

---

critFunC *Functions for Generalized blockmodeling for valued networks*

Description

Functions for implementation of Generalized blockmodeling for valued networks where the values of the ties are assumed to be measured on at least interval scale. critFunC calculates the criterion function, based on the network, partition and blockmodel/equivalece. optParC optimizes a partition based on the criterion function based on a local search algorithm.

Usage

```
critFunC(
  M,
  clu,
  approaches,
  blocks,
  isTwoMode = NULL,
```

```
  isSym = NULL,
  diag = 1,
  IM = NULL,
  EM = NULL,
  Earr = NULL,
  justChange = FALSE,
  rowCluChange = c(0, 0),
  colCluChange = c(0, 0),
  sameIM = FALSE,
  regFun = "max",
  homFun = "ss",
  usePreSpecM = NULL,
  preSpecM = NULL,
  save.initial.param = TRUE,
  relWeights = 1,
  posWeights = 1,
  blockTypeWeights = 1,
  combWeights = NULL,
  returnEnv = FALSE,
  mulReg = TRUE,
  addGroupLlErr = TRUE
)

optParC(
  M,
  clu,
  approaches,
  blocks,
  nMode = NULL,
  isSym = NULL,
  diag = 1,
  useMulti = FALSE,
  maxPar = 50,
  IM = NULL,
  EM = NULL,
  Earr = NULL,
  justChange = TRUE,
  sameIM = FALSE,
  regFun = "max",
  homFun = "ss",
  usePreSpecM = NULL,
  preSpecM = NULL,
  minUnitsRowCluster = 1,
  minUnitsColCluster = 1,
  maxUnitsRowCluster = 9999,
  maxUnitsColCluster = 9999,
  relWeights = 1,
  posWeights = 1,
```

```
  blockTypeWeights = 1,
  combWeights = NULL,
  exchageClusters = "all",
  fixClusters = NULL,
  save.initial.param = TRUE,
  mulReg = TRUE,
  addGroupLlErr = TRUE
)
```

Arguments

M	A matrix representing the (usually valued) network. For multi-relational networks, this should be an array with the third dimension representing the relation. The network can have one or more modes (diferent kinds of units with no ties among themselves). If the network is not two-mode, the matrix must be square.
clu	A partition. Each unique value represents one cluster. If the nework is one-mode, than this should be a vector, else a list of vectors, one for each mode. Similarly, if units are comprised of several sets, clu should be the list containing one vector for each set.
approaches	One of the approaches (for each relation in multi-relational netowrks in a vector) described in Žiberna (2007). Possible values are: "bin" - binary blockmodeling, "val" - valued blockmodeling, "hom" - homogeneity blockmodeling, "ss" - sum of squares homogeneity blockmodeling, and "ad" - absolute deviations homogeneity blockmodeling. The last two options are "shorthand" for specifying approaches="hom" and homFun to either "ss" or "ad".
blocks	A vector, a list of vectors or an array with names of allowed blocy types. Only listing of allowed block types (blockmodel is not pre-specified). A vector with names of allowed block types. For multi-relational networks, it can be a list of such vectors. For approaches = "bin" or approaches = "val", at least two should be selected. Possible values are: "nul" - null or empty block "com" - complete block "rdo", "cdo" - row and column-dominant blocks (binary and valued approach only) "reg" - (f-)regular block "rre", "cre" - row and column-(f-)regular blocks "rfn", "cfn" - row and column-dominant blocks (binary, valued only) "den" - density block (binary approach only) "avg" - average block (valued approach only) "dnc" - do not care block - the error is always zero The ordering is important, since if several block types have identical error, the first on the list is selected.

A pre-specified blockmodel.
An array with four dimensions (see example below). The third and the fourth represent the clusters (for rows and columns). The first is as long as the maximum number of allows block types for a given block. If some block has less possible block types, the empty slots should have values NA. The second dimension is the number of relations (1 for single-relational networks). The values in the array should be the ones from above. The array can have only three dimensions in case of one-relational networks or if the same pre-specified blockmodel is assumed for all relations. Further, it can have only two dimensions, if in addition only one block type is allowed per block.

`isTwoMode` 1 for one-mode networks and 2 for two-mode networks. The default value is set to NULL.

`isSym` Specifying if the matrix (for each relation) is symmetric.

`diag` Should the special status of diagonal be acknowledged. A single number or a vector equal to the number of relation. The default value is set to 1. Codes:
0 - diagonal is treated in the same way as other values
1 - diagonal is treated separately, or
2 - diagonal values are ignored.

`IM` The obtained image for objects. For debugging purposes only.

`EM` Block errors by blocks. For debugging purposes only.

`Earr` The array of errors for all allowed block types by next dimensions: allowed block types, relations, row clusters and column clusters. The dimensions should match the dimensions of the block argument if specified as an array. For debugging purposes only.

`justChange` Value specifying if only the errors for changed clusters should be computed. Used only for debugging purposes by developers.

`rowCluChange` An array holding the two row clusters where the change occured. Used only for debugging purposes by developers.

`colCluChange` An array holding the col row clusters where the change occured. Used only for debugging purposes by developers.

`sameIM` Should we demand the same blockmodel image for all relations. The default value is set to FALSE.

`regFun` Function f used in row-f-regular, column-f-regular, and f-regular blocks. Not used in binary approach. For multi-relational networks, it can be a vector of such character strings. The default value is set to "max".

`homFun` In case of homogeneity blockmodeling two variability criteria can be used: "ss" - sum of squares (set by default), "ad" - absolute deviations and "bll" - - (minus) binary log-likelihood.

`usePreSpecM` Specifying weather a pre-specified value should be used when computing inconsistency.

`preSpecM` Sufficient value for individual cells for valued approach. Can be a number or a character string giving the name of a function. Set to "max" for implicit approach. For multi-relational networks, it can be a vector of such values. In case

ob binary blockmodeling this argument is a threshold used for binerizing the network. Therefore all values with values lower than `preSpecM` are recoded into 0s, all other into 1s. For multi-relational networks, it can be a vector of such values. In case of pre-specified blockmodeling, it can have the same dimensions as `blocks`.

`save.initial.param`
: Should the inital parameters (`approaches`, ...) be saved. The default value is `TRUE`.

`relWeights`
: Weights for all type of relations in a blockmodel. The default value is set to `1`.

`posWeights`
: Weigths for positions in the blockmodel (the dimensions must be the same as the error matrix (rows, columns)). For now this is a matix (two-dimensional) even for multi-relational networks.

`blockTypeWeights`
: Weights for each type of block used, if they are to be different across block types (see `blocks` above). It must be suplied in form of a named vector, where the names are one or all allowed block types from `blocks`. If only some block types are specified, the other have a default weight of 1. The default value is set to `1`.

`combWeights`
: Weights for all type of block used, The default value is set to `NULL`.The dimension must be the same as `blocks`, if `blocks` would be specified in array format (which is usual in pre-specified case).

`returnEnv`
: Should the function also return the environment after its completion.

`mulReg`
: Should the errors that apply to rows/columns (and not to cells) should be multiplied by number of rows/columns. Defaults to TRUE.

`addGroupLlErr`
: Used only when stochastic generalized blockmodeling is used. Should the total error included the part based on sizes of groups. Defaults to TRUE. Will return wrong results for two-mode networks if critFunC is called directly (should be fine if called via optParC function).

`nMode`
: Number of nodes. If `NULL`, then determined from `clu`.

`useMulti`
: Which version of local search should be used. The default value is set to `FALSE`. If `FALSE`, first possible all moves in random order and then all possible exchanges in random order are tired. When a move with lower value of criterion function is found, the algorithm moves to this new partition. If `TRUE` the version of local search where all possible moves and exchanges are tired first and then the one with the lowest error is selected and used. In this case, several optimal partitions are found. `maxPar` best partitions are returned.

`maxPar`
: The number of partitions with optimal criterion fuction to be returned. Only used If `useMulti` is `TRUE`.

`minUnitsRowCluster`
: Minimum number of units in row cluster.

`minUnitsColCluster`
: Minimum number of units in col cluster.

`maxUnitsRowCluster`
: Maximum number of units in row cluster.

`maxUnitsColCluster`
: Maximum number of units in col cluster.

`exchageClusters`
: A matrix of dimensions "number of clusters" x "number of clusters" indicating to which clusters can units from a specific cluster be moved. Useful for multi-level blockmodeling or/in some other cases where some units cannot mix.

`fixClusters`
: Clusters to be fixed. Used only if `exchageClusters = "all"`. A vector of integers that specify clusters to be fixed, where clusters are numbered from 1 to the total (in all modes or sets) number of clusters.

Value

`critFunC` returns a list containing:

`M`
: The matrix of the network analyzed.

`err`
: The error or inconsistency emplirical network with the ideal network for a given blockmodel (model, approach,...) and paritition.

`clu`
: The analyzed partition.

`EM`
: Block errors by blocks.

`IM`
: The obtained image for objects.

`BM`
: Block means by block - only for Homogeneity blockmodeling.

`Earr`
: The array of errors for all allowed block types by next dimensions: allowed block types, relations, row clusters and column clusters. The dimensions should match the dimensions of the block argument if specified as an array.

`optParC` returns a list containing:

`M`
: The matrix of the network analyzed.

`err`
: The error or inconsistency emplirical network with the ideal network for a given blockmodel (model, approach,...) and paritition.

`clu`
: The analyzed partition.

`EM`
: Block errors by blocks.

`IM`
: The obtained image for objects.

`BM`
: Block means by block - only for Homogeneity blockmodeling.

`Earr`
: The array of errors for all allowed block types by next dimensions: allowed block types, relations, row clusters and column clusters. The dimensions should match the dimensions of the block argument if specified as an array.

`useMulti`
: The value of the input paramter `useMulti`.

`bestRowParMatrix`
: (If `useMulti = TRUE`) Matrix, where there are different solutions for columns, where rows represent units.

`sameErr`
: The number of partitions with the minimum value of the criterion function.

Author(s)

Aleš, Žiberna

References

Doreian, P., Batagelj, V., & Ferligoj, A. (2005). Generalized blockmodeling, (Structural analysis in the social sciences, 25). Cambridge [etc.]: Cambridge University Press.

Žiberna, A. (2007). Generalized Blockmodeling of Valued Networks. Social Networks, 29(1), 105-126. doi: 10.1016/j.socnet.2006.04.002

Žiberna, A. (2008). Direct and indirect approaches to blockmodeling of valued networks in terms of regular equivalence. Journal of Mathematical Sociology, 32(1), 57-84. doi: 10.1080/00222500701790207

Žiberna, A. (2014). Blockmodeling of multilevel networks. Social Networks, 39(1), 46-61. doi: 10.1016/j.socnet.2014.04.002

See Also

optRandomParC, IM, clu, err, plot.critFun

Examples

```
# Generating a simple network corresponding to the simple Sum of squares
# Structural equivalence with blockmodel:
# nul com
# nul nul
n <- 20
net <- matrix(NA, ncol = n, nrow = n)
clu <- rep(1:2, times = c(5, 15))
tclu <- table(clu)
net[clu == 1, clu == 1] <- rnorm(n = tclu[1] * tclu[1], mean = 0, sd = 1)
net[clu == 1, clu == 2] <- rnorm(n = tclu[1] * tclu[2], mean = 4, sd = 1)
net[clu == 2, clu == 1] <- rnorm(n = tclu[2] * tclu[1], mean = 0, sd = 1)
net[clu == 2, clu == 2] <- rnorm(n = tclu[2] * tclu[2], mean = 0, sd = 1)

# Computation of criterion function with the correct partition
res <- critFunC(M = net, clu = clu, approaches = "hom", homFun = "ss", blocks = "com")
res$err # The error is relatively small
plot(res)

# Computation of criterion function with the correct partition and correct pre-specified blockmodel
# Prespecified blockmodel used
# nul com
# nul nul
B <- array(NA, dim = c(1, 1, 2, 2))
B[1, 1, , ] <- "nul"
B[1, 1, 1, 2] <- "com"
B[1, 1, , ]
res <- critFunC(M = net, clu = clu, approaches = "hom", homFun = "ss", blocks = B)
res$err # The error is relatively small
res$IM
plot(res)

# Computation of criterion function with the correct partition
# and pre-specified blockmodel with some alternatives
# Prespecified blockmodel used
```

```
# nul nul|com
# nul nul
B <- array(NA, dim = c(2, 2, 2))
B[1, , ] <- "nul"
B[2, 1, 2] <- "com"
res <- critFunC(M = net, clu = clu, approaches = "hom", homFun = "ss", blocks = B)
res$err # The error is relatively small
res$IM
plot(res)

# Computation of criterion function with random partition
set.seed(1)
clu.rnd <- sample(1:2, size = n, replace = TRUE)
res.rnd <- critFunC(M = net, clu = clu.rnd, approaches = "hom",
homFun = "ss", blocks = "com")
res.rnd$err # The error is larger
plot(res.rnd)

# Adapt network for Valued blockmodeling with the same model
net[net > 4] <- 4
net[net < 0] <- 0

# Computation of criterion function with the correct partition
res <- critFunC(M = net, clu = clu, approaches = "val",
blocks = c("nul", "com"), preSpecM = 4)
res$err # The error is relatively small
res$IM
# The image corresponds to the one used for generation of
# The network
plot(res)

# Optimizing one partition
res <- optParC(M = net, clu = clu.rnd,
   approaches = "hom", homFun = "ss", blocks = "com")
plot(res) # Hopefully we get the original partition
```

`expandMat` *Expands a square matrix by repeating each row/column the specified number of times.*

Description

Expands a square matrix by repeating each row/column the specified number of times.

Usage

```
expandMat(mat, nn)
```

Arguments

mat	A square matrix to be exapanded
nn	A vector of number of times each row/column must be repeated. Its length must match the number of rows/columns

Value

Sum of squared deviations from the mean using only valid (non NA) values.

Author(s)

Aleš Žiberna

---

find.cut *Computing the threshold*

Description

The functions compute the maximum value of m/cut where a certain block is still classified as alt.blocks and not "null". The difference between find.m and find.m2 it that find.m uses an optimization approach and is faster and more precise than find.m2. However, find.m only supports regular ("reg") and complete ("com") as alt.blocks, while find.m2 supports all block types. Also, find.m does not always work, especially if cormet is not "none".

Usage

```
find.cut(M, clu, alt.blocks = "reg", cuts = "all", ...)

find.m(
  M,
  clu,
  alt.blocks = "reg",
  diag = !is.list(clu),
  cormet = "none",
  half = TRUE,
  FUN = "max"
)

find.m2(M, clu, alt.blocks = "reg", neval = 100, half = TRUE, ms = NULL, ...)
```

Arguments

M	A matrix representing the (usually valued) network. For now, only one-relational networks are supported. The network can have one or more modes (different kinds of units with no ties among themselves. If the network is not two-mode, the matrix must be square.

clu	A partition. Each unique value represents one cluster. If the network is one-mode, then this should be a vector, else a list of vectors, one for each mode.
alt.blocks	Only one of allowed blocktypes, as alternative to the null block: "com" - complete block "rdo", "cdo" - row and column-dominant blocks (binary, valued, and implicit approach only) "reg" - (f-)regular block "rre", "cre" - row and column-(f-)regular blocks "rfn", "cfn" - row and column-dominant blocks (binary, valued, and implicit approach only) "den" - density block (binary approach only) "avg" - average block (valued approach only).
cuts	The cuts, which should be evaluated. If `cuts="all"` (default), all unique values are evaluated.
...	Other parameters to `critFunC`.
diag	(default = `TRUE`) Should the special status of diagonal be acknowledged.
cormet	Which method should be used to correct for different maximum error contributions "none" - no correction "censor" - censor values larger than `M` "correct" - so that the maximum possible error contribution of the cell is the same regardless of a condition (either that something must be 0 or at least `M`).
half	Should the returned value of m be one half of the value where the inconsistencies are the same.
FUN	(default = "max") Function f used in row-f-regular, column-f-regular, and f-regular blocks.
neval	A number of different `m` values to be evaluated.
ms	The values of m where the function should be evaluated.

Value

A matrix of maximal `m/cut` values.

Author(s)

Aleš Žiberna

References

Doreian, P., Batagelj, V. & Ferligoj, A. Anuška (2005). Generalized blockmodeling, (Structural analysis in the social sciences, 25). Cambridge [etc.]: Cambridge University Press.

Žiberna, A. (2007). Generalized Blockmodeling of Valued Networks. Social Networks, 29(1), 105-126. doi: 10.1016/j.socnet.2006.04.002

Žiberna, A. (2008). Direct and indirect approaches to blockmodeling of valued networks in terms of regular equivalence. Journal of Mathematical Sociology, 32(1), 57-84. doi: 10.1080/00222500701790207

Žiberna, A. (2014). Blockmodeling of multilevel networks. Social Networks, 39(1), 46-61. doi: 10.1016/j.socnet.2014.04.002

See Also

critFunC and maybe also optParC, plotMat

---

formatA *A formating function for numbers*

Description

Formats a vector or matrix of numbers so that all have equal length (digits). This is especially suitable for printing tables.

Usage

```
formatA(x, digits = 2, FUN = round, ...)
```

Arguments

x	A numerical vector or matrix.
digits	The number of desired digits.
FUN	Function used for "shortening" the numbers.
...	Additional arguments to format.

Value

A character vector or matrix.

Author(s)

Aleš Žiberna

See Also

find.m, find.m2, find.cut

Examples

```
A <- matrix(c(1, 1.02002, 0.2, 10.3), ncol = 2)
formatA(A)
```

`funByBlocks.default` *Computation of function values by blocks*

Description

Computes a value of a function over blocks of a matrix, defined by a partition.

Usage

```
## Default S3 method:
funByBlocks(
  x = M,
  clu,
  M = x,
  ignore.diag = "default",
  sortNames = TRUE,
  FUN = "mean",
  ...
)

## S3 method for class 'optMorePar'
funByBlocks(x, which = 1, orderClu = FALSE, sortNames = NULL, ...)

## S3 method for class 'opt.more.par'
funByBlocks(x, which = 1, orderClu = FALSE, sortNames = NULL, ...)

funByBlocks(x, ...)

fun.by.blocks(x, ...)
```

Arguments

`x`	An object of suitable class or a matrix/array representing the (usually valued) network. For multi-relational networks, this should be an array with the third dimension representing the relation. The network can have one or more modes (different kinds of units with no ties among themselves. If the network is not two-mode, the matrix must be square.
`clu`	A partition. Each unique value represents one cluster. If the network is one-mode, then this should be a vector, else a list of vectors, one for each mode.
`M`	A matrix representing the (usually valued) network. For multi-relational networks, this should be an array with the third dimension representing the relation. The network can have one or more modes (different kinds of units with no ties among themselves. If the network is not two-mode, the matrix must be square.
`ignore.diag`	Should the diagonal be ignored.
`sortNames`	Should the rows and columns of the matrix be sorted based on their names.

FUN	The function to be computed over the blocks.
...	Further arguments to funByBlocks.default.
which	Which (if several) of the "best" solutions should be used.
orderClu	Should the partition be ordered before computing. FALSE by default. If TRUE, orderClu is used (using default arguments) to order the clusters in a partition in "decearsing" (see orderClu for interpretation) order. If TRUE, sortNames is set to FALSE.

Value

A numerical matrix of FUN values by blocks, induced by a partition clu.

Author(s)

Aleš Žiberna

References

Žiberna, A. (2007). Generalized Blockmodeling of Valued Networks. Social Networks, 29(1), 105-126. doi: 10.1016/j.socnet.2006.04.002

Žiberna, A. (2008). Direct and indirect approaches to blockmodeling of valued networks in terms of regular equivalence. Journal of Mathematical Sociology, 32(1), 57-84. doi: 10.1080/00222500701790207

See Also

optRandomParC, optParC

Examples

```
n <- 8 # If larger, the number of partitions increases dramatically,
# as does if we increase the number of clusters
net <- matrix(NA, ncol = n, nrow = n)
clu <- rep(1:2, times = c(3, 5))
tclu <- table(clu)
net[clu == 1, clu == 1] <- rnorm(n = tclu[1] * tclu[1], mean = 0, sd = 1)
net[clu == 1, clu == 2] <- rnorm(n = tclu[1] * tclu[2], mean = 4, sd = 1)
net[clu == 2, clu == 1] <- rnorm(n = tclu[2] * tclu[1], mean = 0, sd = 1)
net[clu == 2, clu == 2] <- rnorm(n = tclu[2] * tclu[2], mean = 0, sd = 1)
# Optimizing 10 random partitions with optRandomParC
res <- optRandomParC(M = net, k = 2, rep = 10, approaches = "hom", homFun = "ss", blocks = "com")
plot(res) # Hopefully we get the original partition
funByBlocks(res)
# Computing mean by blocks, ignoring the diagonal (default)
```

genMatrixMult *Generalized matrix multiplication*

Description

Computes a generalized matrix multiplication, where sum and product functions (elemet-wise and summary functions) can be replaced by arbitrary functions.

Usage

```
genMatrixMult(A, B, FUNelement = "*", FUNsummary = sum)
```

Arguments

A	The first matrix.
B	The second matrix.
FUNelement	Element-wise operator.
FUNsummary	Summary function.

Value

A character vector or matrix.

Author(s)

Aleš Žiberna

See Also

matmult

Examples

```
# Operations can be anything
x <- matrix(letters[1:8], ncol = 2)
y <- matrix(1:10, nrow = 2)

genMatrixMult(x, y, FUNelement = paste,
FUNsummary = function(x) paste(x, collapse = "|"))

# Binary logic
set.seed(1)
x <- matrix(rbinom(8, size = 1, prob = 0.5) == 1, ncol = 2)
y <- matrix(rbinom(10, size = 1, prob = 0.5) == 1, nrow = 2)
genMatrixMult(x, y, FUNelement = "*", FUNsummary = any)
```

genRandomPar *The function for generating random partitions*

Description

The function generates random partitions. The function is meant to be called by the function optRandomParC.

Usage

```
genRandomPar(
  k,
  n,
  seed = NULL,
  mingr = 1,
  maxgr = Inf,
  addParam = list(genPajekPar = TRUE, probGenMech = NULL)
)
```

Arguments

k	Number of clusters (by modes).
n	Number of units (by modes).
seed	Seed for generating random numbers (partitions).
mingr	Minimal allowed group size.
maxgr	Maximal allowed group size.
addParam	This has to be a list with the following parameters (any or all can be missing, then the default values (see usage) are used): "genPajekPar" - Should the partitions be generated as in Pajek (Batagelj & Mrvar, 2006). If FALSE, all partitions are selected completely at random while making sure that the partitions have the required number of clusters. probGenMech - Here the probabilities for 4 different generating mechanisms can be specified. If this is not specified, the value is set to c(1/3, 1/3, 1/3, 0) if genPajekPar is TRUE and to c(0, 0, 0, 1) if genPajekPar is FALSE. The first 3 mechanisms are the same as implemented in Pajek (the second one has almost all units in only one cluster) and the fourth is completely random (from uniform distribution).

Value

A random partition in the format required by optRandomParC. If a network has several modes, then a list of partitions, one for each mode.

Author(s)

Aleš Žiberna

References

Batagelj, V., & Mrvar, A. (2006). Pajek 1.11. Retrieved from http://vlado.fmf.uni-lj.si/pub/networks/pajek/

`gplot1` *A wrapper for function gplot - Two-Dimensional Visualization of Graphs*

Description

The function calls function `gplot` from the library `sna` with different defaults. Use fun for plotting image graphs.

Usage

```
gplot1(
  M,
  diag = TRUE,
  displaylabels = TRUE,
  boxed.labels = FALSE,
  loop.cex = 4,
  edge.lwd = 1,
  edge.col = "default",
  rel.thresh = 0.05,
  ...
)

gplot2(
  M,
  uselen = TRUE,
  usecurve = TRUE,
  edge.len = 0.001,
  diag = TRUE,
  displaylabels = TRUE,
  boxed.labels = FALSE,
  loop.cex = 4,
  arrowhead.cex = 2.5,
  edge.lwd = 1,
  edge.col = "default",
  rel.thresh = 0.05,
  ...
)
```

Arguments

`M`	A matrix (array) of a graph or set thereof. This data may be valued.
`diag`	Boolean indicating whether or not the diagonal should be treated as valid data Set this `TRUE` if and only if the data can contain loops. `diag` is `FALSE` by default.

displaylabels	Boolean; should vertex labels be displayed.
boxed.labels	Boolean; place vertex labels within boxes.
loop.cex	An expansion factor for loops; may be given as a vector, if loops are to be of different sizes.
edge.lwd	Line width scale for edges; if set greater than 0, edge widths are scaled by edge.lwd*dat. May be given as a vector or adjacency matrix, if edges are to have different line widths.
edge.col	Color for edges; may be given as a vector or adjacency matrix, if edges are to be of different colors.
rel.thresh	Real number indicating the lower relative (compared to the highest value) threshold for tie values. Only ties of value thresh are displayed. By default, thresh = 0.
...	Additional arguments to plot or link{sna::gplot}: mode: the vertex placement algorithm; this must correspond to a gplot.layout function from package sna.
uselen	Boolean; should we use edge.len to rescale edge lengths.
usecurve	Boolean; should we use edge.curve.
edge.len	If uselen == TRUE, curved edge lengths are scaled by edge.len.
arrowhead.cex	An expansion factor for edge arrowheads.

Value

Plots a graph.

Author(s)

Aleš Žiberna

See Also

link{sna::gplot}

---

ircNorm *Function for iterated row and column normalization of valued matrices*

Description

The aim is to obtain a matrix with row and column sums equal to 1. This is achieved by iterating row and column normalization. This is usually not possible if any row or column has only 1 non-zero cell.

Usage

```
ircNorm(M, eps = 10^-12, maxiter = 1000)
```

Arguments

M	A non-negative valued matrix to be normalized.
eps	The maximum allows squared deviation of a row or column's maximum from 1 (if not exactly 0). Also, if the all deviations in two consequtive iterations are smaller, the process is terminated.
maxiter	Maximum number of iterations. If reached, the process is terminated and the current solution returned.

Value

Normalized matrix.

Author(s)

Aleš Žiberna

Examples

```
A <- matrix(runif(100), ncol = 10)
A # A non-normalized matrix with different row and column sums.
apply(A, 1, sum)
apply(A, 2, sum)
A.norm <- ircNorm(A)
A.norm # Normalized matrix with all row and column sums approximately 1.
apply(A.norm, 1, sum)
apply(A.norm, 2, sum)
```

---

loadmatrix *Functions for loading and writing Pajek files*

Description

`loadmatrix` - Loads a Pajek ".mat" filename as a matrix.

Functions for reading/loading and writing Pajek files:

`loadnetwork` - Loads a Pajek ".net" filename as a matrix. For now, only simple one and two-mode networks are supported (eg. only single relations, no time information).

`loadnetwork2` - The same as above, but adapted to be called within `loadpajek`.

`loadnetwork3` - Another version for reading networks.

`loadnetwork4` - Another version for reading networks.

`loadpajek` - Loads a Pajek project file name (".paj") as a list with the following components: Networks, Partitions, Vectors and Clusters. Clusters and hierarchies are dismissed.

loadvector - Loads a Pajek ".clu" filename as a vector.

loadvector2 - The same as above, but adapted to be called within loadpajek - as a consequence not suited for reading clusters.

savematrix - Saves a matrix into a Pajek ".mat" filename.

savenetwork - Saves a matrix into a Pajek ".net" filename.

savevector - Saves a vector into a Pajek ".clu" filename.

Usage

```
loadmatrix(filename)

loadnetwork(filename, useSparseMatrix = NULL, minN = 50)

loadnetwork2(
  filename,
  useSparseMatrix = NULL,
  minN = 50,
  safe = TRUE,
  closeFile = TRUE
)

loadnetwork3(filename, useSparseMatrix = NULL, minN = 50)

loadnetwork4(filename, useSparseMatrix = NULL, minN = 50, fill = FALSE)

loadpajek(filename)

loadvector(filename)

loadvector2(filename)

savematrix(n, filename, twomode = 1)

savenetwork(n, filename, twomode = "default", symetric = NULL)

savevector(v, filename)
```

Arguments

filename	The name of the file to be loaded or saved to or an open file object.
useSparseMatrix	Should a sparse matrix be use instead of the ordinary one? Sparse matrices can only be used if package Matrix is installed. The default NULL uses sparse matrices for networks with more that minN vertices.
minN	The minimal number of units in the network to use sparse matrices.
safe	If FALSE error will occur if not all vertices have labels. If TRUE reading works faster.

closeFile	Should the connection be closed at the end. Should be always TRUE if function is used directly.
fill	If TRUE, then in case the rows have unequal length, blank fields are added.
n	A matrix representing the network.
twomode	1 for one-mode networks and 2 for two-mode networks. Default sets the argument to 1 for square matrices and to 2 for others.
symetric	If TRUE, only the lower part of the matrix is used and the values are interpreted as "Edges", not "Arcs".
v	A vector.

Value

NULL, a matrix or a vector.

Author(s)

Vladimir Batagelj & Andrej Mrvar (most functions), Aleš Žiberna (loadnetwork, loadpajek and modification of others)

References

Batagelj, V., & Mrvar. A. (1999). Pajek - Program for Large Network Analysis. Retrieved from http://vlado.fmf.uni-lj.si/pub/networks/pajek/.

de Nooy, W., Mrvar, A., & Batagelj. V. (2005). Exploratory Social Network Analysis with Pajek. London: SAGE Publications.

See Also

plot.mat, critFunC, optRandomParC

---

nanRep *Replaces NaN values by the speficied values (0 by default)*

Description

Replaces NaN values by the speficied values (0 by default)

Usage

```
nanRep(x, rep = 0)
```

Arguments

x	A vector or similar where the NaNs are to be replaced.
rep	A value that should replace the NaNs (0 by default).

Value

x with NaNs replaced by rep.

Author(s)

Aleš Žiberna

---

nkpar *Functions for listing all possible partitions or just counting the number of them*

Description

The function nkpartitions lists all possible partitions of n objects in to k clusters.

Usage

```
nkpar(n, k)

nkpartitions(n, k, exact = TRUE, print = FALSE)
```

Arguments

n	Number of units/objects.
k	Number of clusters/groups.
exact	Search for partitions with exactly k or at most k clusters.
print	Print results as they are found.

Value

The matrix or number of possible partitions.

Author(s)

Chris Andrews

Examples

```
n <- 8 # If larger, the number of partitions increases dramatically,
# as does if we increase the number of clusters
net <- matrix(NA, ncol = n, nrow = n)
clu <- rep(1:2, times = c(3, 5))
tclu <- table(clu)
net[clu == 1, clu == 1] <- rnorm(n = tclu[1] * tclu[1], mean = 0, sd = 1)
net[clu == 1, clu == 2] <- rnorm(n = tclu[1] * tclu[2], mean = 4, sd = 1)
net[clu == 2, clu == 1] <- rnorm(n = tclu[2] * tclu[1], mean = 0, sd = 1)
net[clu == 2, clu == 2] <- rnorm(n = tclu[2] * tclu[2], mean = 0, sd = 1)
```

```
# Computation of criterion function with the correct partition
nkpar(n = n, k = length(tclu)) # Computing the number of partitions
all.par <- nkpartitions(n = n, k = length(tclu)) # Forming the partitions
all.par <- lapply(apply(all.par, 1, list), function(x) x[[1]])
# to make a list out of the matrix
res <- critFunC(M = net, clu = clu, approaches = "val",
                blocks = c("nul", "com"), preSpecM = 4)
plot(res) # We get the original partition
```

---

notesBorrowing *The notes borrowing network between social-informatics students*

Description

The data come from a survey conducted in May 1993 on 13 social-informatics students (Hlebec, 1996). The network was constructed from answers to the question, "How often did you borrow notes from this person?" for each of the fellow students. The respondents indicated the frequency of borrowing by choosing (on a computer) a line of length 1-20, where 1 meant no borrowing. 1 was deducted from all answers, so that 0 now means no borrowing. The data was first used for blockmodeling in Žiberna (2007).

Usage

```
data("notesBorrowing")
```

Format

The data set is a valued matrix with 13 rows and columns.

References

Hlebec, V., (1996). *Metodološke značilnosti anketnega zbiranja podatkov v analizi omrežji: Magistersko delo*. FDV, Ljubljana.

Žiberna, A. (2007). Generalized blockmodeling of valued networks. *Social Networks*, 29, 105-126. https://doi.org/10.1016/j.socnet.2006.04.002

Examples

```
data(notesBorrowing)

# Plot the network.
# (The function plotMat is from blockmodeling package.)
# plotMat(nyt)
```

one2two *Two-mode network conversions*

Description

Converting two mode networks from two to one mode matrix representation and vice versa. If a two-mode matrix is converted into a one-mode matrix, the original two-mode matrix lies in the upper right corner of the one-mode matrix.

Usage

```
one2two(M, clu = NULL)

two2one(M, clu = NULL)
```

Arguments

M	A matrix representing the (usually valued) network.
clu	A partition. Each unique value represents one cluster. This should be a list of two vectors, one for each mode.

Value

Function returns list with the elements: a two mode matrix of a the two mode network in its upper left corner.

M	The matrix.
clu	The partition, in form appropriate for the mode of the matrix.

Author(s)

Aleš Žiberna

See Also

optParC, optParC, optRandomParC, plot.mat

Examples

```
# Generating a simple network corresponding to the simple Sum of squares
# Structural equivalence with blockmodel:
# null com
# null null
n <- c(7, 13)
net <- matrix(NA, nrow = n[1], ncol = n[2])
clu <- list(rep(1:2, times = c(3, 4)), rep(1:2, times = c(5, 8)))
tclu <- lapply(clu, table)
net[clu[[1]] == 1, clu[[2]] == 1] <- rnorm(n = tclu[[1]][1] * tclu[[2]][1],
```

```
   mean = 0, sd = 1)
net[clu[[1]] == 1, clu[[2]] == 2] <- rnorm(n = tclu[[1]][1] * tclu[[2]][2],
   mean = 4, sd = 1)
net[clu[[1]] == 2, clu[[2]] == 1] <- rnorm(n = tclu[[1]][2] * tclu[[2]][1],
   mean = 4, sd = 1)
net[clu[[1]] == 2, clu[[2]] == 2] <- rnorm(n = tclu[[1]][2] * tclu[[2]][2],
   mean = 0, sd = 1)
plot.mat(net, clu = clu) # Two mode matrix of a two mode network

# Converting to one mode network
M1 <- two2one(net)$M
plot.mat(M1, clu = two2one(net)$clu) # Plotting one mode matrix
# Converting one to two mode matrix and plotting
plot.mat(one2two(M1, clu = clu)$M, clu = clu)
```

---

optRandomParC *Optimizing a set of partitions based on the value of a criterion function*

Description

The function optimizes a set of partitions based on the value of a criterion function (see critFunC for details on the criterion function) for a given network and blockmodel for Generalized blockmodeling (Žiberna, 2007) based on other parameters (see below). The optimization is done through local optimization, where the neighborhood of a partition includes all partitions that can be obtained by moving one unit from one cluster to another or by exchanging two units (from different clusters). The number of clusters and a number of partitions to generate can be specified (optParC).

Usage

```
optRandomParC(
  M,
  k,
  approaches,
  blocks,
  rep,
  save.initial.param = TRUE,
  save.initial.param.opt = FALSE,
  deleteMs = TRUE,
  max.iden = 10,
  switch.names = NULL,
  return.all = FALSE,
  return.err = TRUE,
  seed = NULL,
  RandomSeed = NULL,
  parGenFun = genRandomPar,
  mingr = NULL,
  maxgr = NULL,
```

```
  addParam = list(genPajekPar = TRUE, probGenMech = NULL),
  maxTriesToFindNewPar = rep * 10,
  skip.par = NULL,
  useOptParMultiC = FALSE,
  useMulti = useOptParMultiC,
  printRep = ifelse(rep <= 10, 1, round(rep/10)),
  n = NULL,
  nCores = 1,
  useParLapply = FALSE,
  useLB = NULL,
  chunk.size = 1,
  cl = NULL,
  stopcl = is.null(cl),
  useRegParrallaBackend = FALSE,
  ...
)

## S3 method for class 'optMorePar'
print(x, ...)
```

Arguments

M	A matrix representing the (usually valued) network. For multi-relational networks, this should be an array with the third dimension representing the relation. The network can have one or more modes (diferent kinds of units with no ties among themselves). If the network is not two-mode, the matrix must be square.
k	The number of clusters used in the generation of partitions.
approaches	One of the approaches (for each relation in multi-relational netowrks in a vector) described in Žiberna (2007). Possible values are: "bin" - binary blockmodeling, "val" - valued blockmodeling, "hom" - homogeneity blockmodeling, "ss" - sum of squares homogeneity blockmodeling, and "ad" - absolute deviations homogeneity blockmodeling. The last two options are "shorthand" for specifying approaches="hom" and homFun to either "ss" or "ad".
blocks	A vector, a list of vectors or an array with names of allowed blocy types. Only listing of allowed block types (blockmodel is not pre-specified). A vector with names of allowed block types. For multi-relational networks, it can be a list of such vectors. For approaches = "bin" or approaches = "val", at least two should be selected. Possible values are: "nul" - null or empty block "com" - complete block "rdo", "cdo" - row and column-dominant blocks (binary and valued approach only) "reg" - (f-)regular block

"rre", "cre" - row and column-(f-)regular blocks
"rfn", "cfn" - row and column-dominant blocks (binary, valued only)
"den" - density block (binary approach only)
"avg" - average block (valued approach only)
"dnc" - do not care block - the error is always zero
The ordering is important, since if several block types have identical error, the first on the list is selected.

A pre-specified blockmodel.
An array with four dimensions (see example below). The third and the fourth represent the clusters (for rows and columns). The first is as long as the maximum number of allows block types for a given block. If some block has less possible block types, the empty slots should have values NA. The second dimension is the number of relations (1 for single-relational networks). The values in the array should be the ones from above. The array can have only three dimensions in case of one-relational networks or if the same pre-specified blockmodel is assumed for all relations. Further, it can have only two dimensions, if in addition only one block type is allowed per block.

`rep` The number of repetitions/different starting partitions to check.

`save.initial.param`
Should the inital parameters (approaches, ...) be saved. The default value is TRUE.

`save.initial.param.opt`
Should the inital parameters(approaches, ...) of using optParC be saved. The default value is FALSE.

`deleteMs` Delete networks/matrices from the results of to save space.

`max.iden` Maximum number of results that should be saved (in case there are more than max.iden results with minimal error, only the first max.iden will be saved).

`switch.names` Should partitions that only differ in group names be considered equal. By default it is set to TRUE if blocks is either a vector or a list of vectors and to FALSE otherwise.

`return.all` If FALSE, solution for only the best (one or more) partition/s is/are returned.

`return.err` Should the error for each optimized partition be returned.

`seed` Optional. The seed for random generation of partitions.

`RandomSeed` Optional. Integer vector, containing the random number generator. It is only looked for in the user's workspace.

`parGenFun` The function (object) that will generate random partitions. The default function is genRandomPar. The function has to accept the following parameters: k (number o of partitions by modes, n (number of units by modes), seed (seed value for random generation of partition), addParam (a list of additional parameters).

`mingr` Minimal allowed group size.

`maxgr` Maximal allowed group size.

`addParam` A list of additional parameters for function specified above. In the usage section they are specified for the default function genRandomPar.

maxTriesToFindNewPar
: The maximum number of partition try when trying to find a new partition to optimize that was not yet checked before - the default value is `rep * 1000`.

skip.par
: The partitions that are not allowed or were already checked and should therefore be skipped.

useOptParMultiC
: For backward compatibility. May be removed soon. See next argument.

useMulti
: Which version of local search should be used. Default is currently `FALSE`. If `FALSE`, first possible all moves in random order and then all possible exchanges in random order are tried. When a move with lower value of criterion function is found, the algorithm moves to this new partition. If `TRUE` the version of local search where all possible moves and exchanges are tried first and then the one with the lowest error is selected and used. In this case, several optimal partitions are found. `maxPar` best partitions are returned.

printRep
: Should some information about each optimization be printed.

n
: The number of units by "modes". It is used only for generating random partitions. It has to be set only if there are more than two modes or if there are two modes, but the matrix representing the network is one mode (both modes are in rows and columns).

nCores
: Number of cores to be used. Value `0` means all available cores. It can also be a cluster object.

useParLapply
: Should `parLapplyLB` or `parLapply` (see `useLB`) be used for parallel execution (on multiple cores). Otherwise `mforeach` is used. Defaults to FALSE. If `useParLapply = TRUE` and `useLB = TRUE`, results are not reproducible.

useLB
: Should be logical if set. Only used if `useParLapply = TRUE`. Should load balancing be used (`parLapplyLB` instead of `parLapply`). Using load balancing usually means faster execution, but results are with not reproducible. Defaults to `NULL`, which is changed to `TRUE`, but a warning.

chunk.size
: chunk.size used in `parLapplyLB` if it is used, otherwise ignored. Defaults to 1.

cl
: The cluster to use (if formed beforehand). Defaults to `NULL`. Ignored if useParLapply=FALSE (default) and foreach::getDoParRegistered is true

stopcl
: Should the cluster be stoped after the function finishes. Defaults to `is.null(cl)`.

useRegParrallaBackend
: Should the function use already registered parallel backend. Defaults to `FALSE`. If `TRUE`, you must make sure that an appropriate backend is correctly set up and registered. Use only if `useParLapply = FALSE` (default) and `nCore` is not 1.

...
: Arguments passed to other functions, see `critFunC`.

x
: The result of `optRandomParC`.

genPajekPar
: Should the partitions be generated as in Pajek.

probGenMech
: Should the probabilities for different mechanisms for specifying the partitions be set. If `probGenMech` is not set, it is determined based on the parameter `genPajekPar`.

Value

M	The matrix of the network analyzed.
res	If return.all = TRUE - A list of results the same as best - one best for each partition optimized.
best	A list of results from optParC, only without M.
err	If return.err = TRUE - The vector of errors or inconsistencies of the empirical network with the ideal network for a given blockmodel (model,approach,...) and parititions.
nIter	The vector of the number of iterations used - one value for each starting partition that was optimized. It can show that maxiter is too low if a lot of these values have the value of maxiter.
checked.par	If selected - A list of checked partitions. If merge.save.skip.par is TRUE, this list also includes the partitions in skip.par.
call	The call used to call the function.
initial.param	If selected - The initial parameters are used.

Warning

It should be noted that the time complexity of package blockmodeling is increasing with the number of units and the number of clusters (due to its algorithm). Therefore the analysis of network with more than 100 units can take a lot of time (from a few hours to a few days).

Author(s)

Aleš, Žiberna

References

Batagelj, V., & Mrvar, A. (2006). Pajek 1.11. Retrieved from http://vlado.fmf.uni-lj.si/pub/networks/pajek/

Doreian, P., Batagelj, V. & Ferligoj, A. (2005). Generalized blockmodeling, (Structural analysis in the social sciences, 25). Cambridge [etc.]: Cambridge University Press.

Žiberna, A. (2007). Generalized Blockmodeling of Valued Networks. Social Networks, 29(1), 105-126. doi: 10.1016/j.socnet.2006.04.002

Žiberna, A. (2008). Direct and indirect approaches to blockmodeling of valued networks in terms of regular equivalence. Journal of Mathematical Sociology, 32(1), 57-84. doi: 10.1080/00222500701790207

Žiberna, A. (2014). Blockmodeling of multilevel networks. Social Networks, 39(1), 46-61. doi: 10.1016/j.socnet.2014.04.002

See Also

critFunC, IM, clu, err, plot.optMorePar

Examples

```
n <- 8 # If larger, the number of partitions increases dramatically
# as does if we increase the number of clusters
net <- matrix(NA, ncol = n, nrow = n)
clu <- rep(1:2, times = c(3, 5))
tclu <- table(clu)
net[clu == 1, clu == 1] <- rnorm(n = tclu[1] * tclu[1], mean = 0, sd = 1)
net[clu == 1, clu == 2] <- rnorm(n = tclu[1] * tclu[2], mean = 4, sd = 1)
net[clu == 2, clu == 1] <- rnorm(n = tclu[2] * tclu[1], mean = 0, sd = 1)
net[clu == 2, clu == 2] <- rnorm(n = tclu[2] * tclu[2], mean = 0, sd = 1)

# Optimizing 10 random chosen partitions with optRandomParC
res <- optRandomParC(M = net, k = 2, rep = 10,
approaches = "hom", homFun = "ss", blocks = "com")
plot(res) # Hopefully we get the original partition
```

`orderClu` *Orders the partition so that mean values of* `fun` *applied to columns (if* `funWay=2`*, default), rows (if* `funWay=1`*) or both (if* `funWay=c(1,2)`*) is decreasing by clusters.*

Description

Orders the partition so that mean values of `fun` applied to columns (if `funWay=2`, default), rows (if `funWay=1`) or both (if `funWay=c(1,2)`) is decreasing by clusters. The function can be used on the results of `critFunC`, `optRandomParC` or similar, or matrix and a partition can be supplied. It should also work on multirelational and lined networks.

Usage

```
orderClu(
  x,
  clu = NULL,
  fun = sum,
  funWay = 2,
  nn = NULL,
  returnList = TRUE,
  scale = TRUE
)
```

Arguments

`x`	A result of `critFunC`, `optRandomParC` or similar (something containing M (matrix) and clu (partition)) or a matrix (or array for multirelational networks).
`clu`	A partition - a vector or a list of vectors/partitions. It must be supplied only if `x` is a matrix or array.

fun	A function used to summarize rows or columns. sum by default.
funWay	In which "way" should fun be appluied - to columns (if funWay=2, default), rows (if funWay=1) or both (if funWay=c(1,2))
nn	The numbers of untis by sets of units. In principle, the function should determin this automatically.
returnList	Logical. Should the partition be returned in form of a list (for lined networks only). TRUE by default.
scale	Only used in case of multirelational networks. Should relations be scaled (TRUE by default) before summation. It can also be a vector of weights by relations.

Value

An ordered partition. In an attribute ("reorder"). the information on how things were reordered.

See Also

clu

---

plot.critFun *Functions for plotting a partitioned matrix (representing the network)*

Description

The main function plot.mat or plotMat plots a (optionally partitioned) matrix. If the matrix is partitioned, the rows and columns of the matrix are rearranged according to the partitions. Other functions are only wrappers for plot.mat or plotMat for convenience when plotting the results of the corresponding functions. The plotMatNm plots two matrices based on M, normalized by rows and columns, next to each other. The plotArray plots an array. plot.mat.nm has been replaced by plotMatNm.

Usage

```
## S3 method for class 'critFun'
plot(x, main = NULL, ...)

## S3 method for class 'crit.fun'
plot(x, main = NULL, ...)

plotMatNm(
  M = x,
  x = M,
  ...,
  main.title = NULL,
  title.row = "Row normalized",
  title.col = "Column normalized",
  main.title.line = -2,
```

```
  par.set = list(mfrow = c(1, 2))
)

## S3 method for class 'optMorePar'
plot(x, main = NULL, which = 1, ...)

## S3 method for class 'opt.more.par'
plot(x, main = NULL, which = 1, ...)

## S3 method for class 'optMoreParMode'
plot(x, main = NULL, which = 1, ...)

## S3 method for class 'opt.more.par.mode'
plot(x, main = NULL, which = 1, ...)

## S3 method for class 'optPar'
plot(x, main = NULL, which = 1, ...)

## S3 method for class 'opt.par'
plot(x, main = NULL, which = 1, ...)

## S3 method for class 'optParMode'
plot(x, main = NULL, which = 1, ...)

## S3 method for class 'opt.par.mode'
plot(x, main = NULL, which = 1, ...)

plotMat(
  x = M,
  clu = NULL,
  orderClu = FALSE,
  M = x,
  ylab = "",
  xlab = "",
  main = NULL,
  print.val = !length(table(M)) <= 2,
  print.0 = FALSE,
  plot.legend = !print.val && !length(table(M)) <= 2,
  print.legend.val = "out",
  print.digits.legend = 2,
  print.digits.cells = 2,
  print.cells.mf = NULL,
  outer.title = FALSE,
  title.line = ifelse(outer.title, -1.5, 7),
  mar = c(0.5, 7, 8.5, 0) + 0.1,
  cex.val = "default",
  val.y.coor.cor = 0,
  val.x.coor.cor = 0,
```

```
  cex.legend = 1,
  legend.title = "Legend",
  cex.axes = "default",
  print.axes.val = NULL,
  print.x.axis.val = !is.null(colnames(M)),
  print.y.axis.val = !is.null(rownames(M)),
  x.axis.val.pos = 1.01,
  y.axis.val.pos = -0.01,
  cex.main = par()$cex.main,
  cex.lab = par()$cex.lab,
  yaxis.line = -1.5,
  xaxis.line = -1,
  legend.left = 0.4,
  legend.up = 0.03,
  legend.size = 1/min(dim(M)),
  legend.text.hor.pos = 0.5,
  par.line.width = 3,
  par.line.width.newSet = par.line.width[1] * 2,
  par.line.col = "blue",
  par.line.col.newSet = "red",
  IM.dens = NULL,
  IM = NULL,
  wnet = NULL,
  wIM = NULL,
  use.IM = length(dim(IM)) == length(dim(M)) | !is.null(wIM),
  dens.leg = c(null = 100, nul = 100),
  blackdens = 70,
  plotLines = FALSE,
  frameMatrix = TRUE,
  x0ParLine = -0.1,
  x1ParLine = 1,
  y0ParLine = 0,
  y1ParLine = 1.1,
  colByUnits = NULL,
  colByRow = NULL,
  colByCol = NULL,
  mulCol = 2,
  joinColOperator = "+",
  colTies = FALSE,
  maxValPlot = NULL,
  printMultipliedMessage = TRUE,
  replaceNAdiagWith0 = TRUE,
  colLabels = FALSE,
  MplotValues = NULL,
  ...
)

plotArray(
```

```
  x = M,
  M = x,
  IM = NULL,
  ...,
  main.title = NULL,
  main.title.line = -2,
  mfrow = NULL
)

## S3 method for class 'mat'
plot(
  x = M,
  clu = NULL,
  orderClu = FALSE,
  M = x,
  ylab = "",
  xlab = "",
  main = NULL,
  print.val = !length(table(M)) <= 2,
  print.0 = FALSE,
  plot.legend = !print.val && !length(table(M)) <= 2,
  print.legend.val = "out",
  print.digits.legend = 2,
  print.digits.cells = 2,
  print.cells.mf = NULL,
  outer.title = FALSE,
  title.line = ifelse(outer.title, -1.5, 7),
  mar = c(0.5, 7, 8.5, 0) + 0.1,
  cex.val = "default",
  val.y.coor.cor = 0,
  val.x.coor.cor = 0,
  cex.legend = 1,
  legend.title = "Legend",
  cex.axes = "default",
  print.axes.val = NULL,
  print.x.axis.val = !is.null(colnames(M)),
  print.y.axis.val = !is.null(rownames(M)),
  x.axis.val.pos = 1.01,
  y.axis.val.pos = -0.01,
  cex.main = par()$cex.main,
  cex.lab = par()$cex.lab,
  yaxis.line = -1.5,
  xaxis.line = -1,
  legend.left = 0.4,
  legend.up = 0.03,
  legend.size = 1/min(dim(M)),
  legend.text.hor.pos = 0.5,
  par.line.width = 3,
```

```
  par.line.width.newSet = par.line.width[1] * 2,
  par.line.col = "blue",
  par.line.col.newSet = "red",
  IM.dens = NULL,
  IM = NULL,
  wnet = NULL,
  wIM = NULL,
  use.IM = length(dim(IM)) == length(dim(M)) | !is.null(wIM),
  dens.leg = c(null = 100, nul = 100),
  blackdens = 70,
  plotLines = FALSE,
  frameMatrix = TRUE,
  x0ParLine = -0.1,
  x1ParLine = 1,
  y0ParLine = 0,
  y1ParLine = 1.1,
  colByUnits = NULL,
  colByRow = NULL,
  colByCol = NULL,
  mulCol = 2,
  joinColOperator = "+",
  colTies = FALSE,
  maxValPlot = NULL,
  printMultipliedMessage = TRUE,
  replaceNAdiagWith0 = TRUE,
  colLabels = FALSE,
  MplotValues = NULL,
  ...
)
```

Arguments

x	A result from a corresponding function or a matrix or similar object representing a network.
main	Main title.
...	Additional arguments to plot.default for plotMat and also to plotMat for other functions.
M	A matrix or similar object representing a network - either x or M must be supplied - both are here to make the code compatible with generic and with older functions.
main.title	Main title in plotArray version.
title.row	Title for the row-normalized matrix in nm version
title.col	Title for the column-normalized matrix in nm version
main.title.line	The line in which main title is printed in plotArray version.
par.set	A list of possible plotting parameters (to par) to be used in nm version

`which` Which (if there are more than one) of optimal solutions to plot.

`clu` A partition. Each unique value represents one cluster. If the network is one-mode, then this should be a vector, else a list of vectors, one for each mode/set.

`orderClu` Should the partition be ordered before plotting. `FALSE` by default. If `TRUE`, `orderClu` is used (using default arguments) to order the clusters in a partition in "decreasing" (see `orderClu` for interpretation) order.

`ylab` Label for y axis.

`xlab` Label for x axis.

`print.val` Should the values be printed in the matrix.

`print.0` If `print.val = TRUE` Should the 0s be printed in the matrix.

`plot.legend` Should the legend for shades be plotted.

`print.legend.val`
Should the values be printed in the legend.

`print.digits.legend`
The number of digits that should appear in the legend.

`print.digits.cells`
The number of digits that should appear in the cells (of the matrix and/or legend).

`print.cells.mf` If not `NULL`, the above argument is ignored, the cell values are printed as the cell are multiplied by this factor and rounded.

`outer.title` Should the title be printed on the 'inner' or 'outer' margin of the plot, default is 'inner' margin.

`title.line` The line (from the top) where the title should be printed. The suitable values depend heavily on the displayed type.

`mar` A numerical vector of the form `c(bottom, left, top, right)` which gives the lines of margin to be specified on the four sides of the plot. The R default for ordinary plots is `c(5, 4, 4, 2) + 0.1`, while this function default is `c(0.5, 7, 8.5, 0) + 0.1`.

`cex.val` The size of the values printed. The default is `10 / 'number of units'`.

`val.y.coor.cor` Correction for centering the values in the squares in y direction.

`val.x.coor.cor` Correction for centering the values in the squares in x direction.

`cex.legend` Size of the text in the legend.

`legend.title` The title of the legend.

`cex.axes` Size of the characters in axes. Default makes the cex so small that all categories can be printed.

`print.axes.val` Should the axes values be printed. Default prints each axis if `rownames` or `colnames` is not `NULL`.

`print.x.axis.val`
Should the x axis values be printed. Default prints each axis if `rownames` or `colnames` is not `NULL`.

`print.y.axis.val`
Should the y axis values be printed. Default prints each axis if `rownames` or `colnames` is not `NULL`.

`x.axis.val.pos` The x coordinate of the y axis values.

`y.axis.val.pos` The y coordinate of the x axis values.

`cex.main` Size of the text in the main title.

`cex.lab` Size of the text in matrix.

`yaxis.line` The position of the y axis (the argument 'line').

`xaxis.line` The position of the x axis (the argument 'line').

`legend.left` How much left should the legend be from the matrix.

`legend.up` How much up should the legend be from the matrix.

`legend.size` Relative legend size.

`legend.text.hor.pos`
Horizontal position of the legend text (bottom) - 0 = bottom, 0.5 = middle,...

`par.line.width` The width of the line that separates the partitions.

`par.line.width.newSet`
The width of the line that separates that separates the sets/modes - only used when `clu` is a list and `par.line.width` has length 1.

`par.line.col` The color of the line that separates the partitions.

`par.line.col.newSet`
The color of the line that separates that separates the sets/modes - only used when `clu` is a list and `par.line.col` has length 1.

`IM.dens` The density of shading lines in each block.

`IM` The image (as obtained with `critFunC`) of the blockmodel. `dens.leg` is used to translate this image into `IM.dens`.

`wnet` Specifies which matrix (if more) should be plotted - used if `M` is an array.

`wIM` Specifies which `IM` (if more) should be used for plotting. The default value is set to `wnet`) - used if `IM` is an array.

`use.IM` Specifies if `IM` should be used for plotting.

`dens.leg` It is used to translate the `IM` into `IM.dens`.

`blackdens` At which density should the values on dark colors of lines be printed in white.

`plotLines` Should the lines in the matrix be printed. The default value is set to `FALSE`, best set to `TRUE` for very small networks.

`frameMatrix` Should the matrix be framed (if `plotLines` is `FALSE`). The default value is set to `TRUE`.

`x0ParLine` Coordinates for lines separating clusters.

`x1ParLine` Coordinates for lines separating clusters.

`y0ParLine` Coordinates for lines separating clusters.

`y1ParLine` Coordinates for lines separating clusters.

`colByUnits` Coloring units. It should be a vector of unit length.

`colByRow` Coloring units by rows. It should be a vector of unit length.

`colByCol` Coloring units by columns. It should be a vector of unit length.

`mulCol` Multiply color when joining with row, column. Only used when when `colByUnits` is not `NULL`.

`joinColOperator`
Function to join `colByRow` and `colByCol`. The default value is set to `"+"`.

`colTies` If `TRUE`, ties are colored, if `FALSE`, 0-ties are colored.

`maxValPlot` The value to use as a maximum when computing colors (ties with maximal positive value are plotted as black).

`printMultipliedMessage`
Should the message '* all values in cells were multiplied by' be printed on the plot. The default value is set to `TRUE`.

`replaceNAdiagWith0`
If `replaceNAdiagWith0` = `TRUE` Should the `NA` values on the diagonal of a matrix be replaced with 0s.

`colLabels` Should the labels of units be colored. If `FALSE`, these are not colored, if `TRUE`, they are colored with colors of clusters as defined by palette. This can be also a vector of colors (or integers) for one-mode networks or a list of two such vectors for two-mode networks.

`MplotValues` A matrix to strings to plot in cells. Only to be used if other values than those in the original matrix (`x` or `M` arguments) should be used. Defaults to `NULL`, in which case the valued from original matrix are plotted (if this is not prevented by some other arguments). Overrides all other arguments that deal with cell values (e.g. `print.digits.cells`). Sets `print.val` to `TRUE` and `plot.legend` to `FALSE`.

`mfrow` `mfrow` Argument to `par` - number of row and column plots to be plotted on one figure.

Value

The functions are used for their side effect - plotting.

Author(s)

Aleš Žiberna

References

Žiberna, A. (2007). Generalized Blockmodeling of Valued Networks. Social Networks, 29(1), 105-126. doi: 10.1016/j.socnet.2006.04.002

Žiberna, A. (2008). Direct and indirect approaches to blockmodeling of valued networks in terms of regular equivalence. Journal of Mathematical Sociology, 32(1), 57-84. doi: 10.1080/00222500701790207

See Also

`critFunC`, `optRandomParC`

Examples

```
# Generation of the network
n <- 20
net <- matrix(NA, ncol = n, nrow = n)
clu <- rep(1:2, times = c(5, 15))
tclu <- table(clu)
net[clu == 1, clu == 1] <- rnorm(n = tclu[1] * tclu[1], mean = 0, sd = 1)
net[clu == 1, clu == 2] <- rnorm(n = tclu[1] * tclu[2], mean = 4, sd = 1)
net[clu == 2, clu == 1] <- rnorm(n = tclu[2] * tclu[1], mean = 0, sd = 1)
net[clu == 2, clu == 2] <- rnorm(n = tclu[2] * tclu[2], mean = 0, sd = 1)

# Ploting the network
plotMat(M = net, clu = clu, print.digits.cells = 3)
class(net) <- "mat"
plot(net, clu = clu)
# See corresponding functions for examples for other ploting
# functions
# presented, that are essentially only the wrappers for "plot.max"
```

---

printBlocks *Nice printing of the* blocks *parameter as used in* optRandomParC *and* critFunC.

Description

Nice printing of the blocks parameter as used in optRandomParC and critFunC.

Usage

```
printBlocks(blocks)
```

Arguments

blocks	blocks parameter as used in optRandomParC and critFunC.

Value

Used for side effects (printing)

Author(s)

Aleš, Žiberna

See Also

optRandomParC, critFunC

rand *Comparing partitions on one or multiple sets of units*

Description

Rand Index and Rand Index corrected/adjusted for chance for comparing partitions (Hubert & Arabie, 1985). The functions also support computing these indices on partitions on multiple sets (where a "combined" partition is a list of multiple partitions). The names of the clusters do not matter.

Usage

```
rand(clu1, clu2, tab)

crand(
  clu1,
  clu2,
  tab,
  multiSets = c("weights", "unlist"),
  weights = c("size", "equal"),
  returnIndividual = "attr"
)

rand2(clu1, clu2)

crand2(clu1, clu2)
```

Arguments

clu1	The first of the two partitions to be compared, given in the form of vectors, where for each unit a cluster membership is given. Alternatively, this can be a contingency table obtained as a `table(clu1, clu2)`. If a partition, `clu2` must also be provided. In case of multiple sets, this should be pa list of partitions.
clu2	If `clu1` is partition or a list of partitions, this must be a comaptible the second partition or list of partitions.
tab	A contingency table obtained as a `table(clu1, clu2)`. This is included for back-compatibility reasons. If this is present, all other arguments are ignored.
multiSets	How should we compute the index in case of multiple sets of unis (if `clu1` and `clu2` are lists of partitions)? Possible values are "unlist" and "weight" (the default).
weights	Weights to be used if `multiSets` is "weight". It can be "equal", "size" (default) or a numeric (non-negative) vector of the same length as the number of sets (the number of partitions in the list of partitions).
returnIndividual	If `multiSets` is "weight", should the indices for individual sets be also returned. If `TRUE`, the function returns a list instead of a single value. If the values is `"attr"` (the default), the indices by sets are given as an attribute `"bySets"`

Value

The value of Rand Index (corrected/adjusted for chance) unless `multiSets="weight"` and `returnIndividual=FALSE`. In this case, a list with two items is return. The "global" index is in `global`, while the the indices by sets are in `bySets`.

Author(s)

Aleš Žiberna

References

Hubert, L., & Arabie, P. (1985). Comparing Partitions. Journal of Classification, 2(1), 193-218.

`recode` *Recode*

Description

Recodes values in a vector.

Usage

```
recode(x, oldcode = sort(unique(x)), newcode)
```

Arguments

`x`	A vector.
`oldcode`	A vector of old codes.
`newcode`	A vector of new codes.

Value

A recoded vector.

Author(s)

Aleš Žiberna

Examples

```
x <- rep(1:3, times = 1:3)
newx <- recode(x, oldcode = 1:3, newcode = c("a", "b", "c"))
```

REGE.FC *REGE - Algorithms for compiting (dis)similarities in terms of regular equivalnece*

Description

REGE - Algorithms for compiting (dis)similarities in terms of regular equivalnece (White & Reitz, 1983). `REGE`, `REGE.for` - Classical REGE or REGGE, as also implemented in Ucinet. Similarities in terms of regular equivalence are computed. The `REGE.for` is a wrapper for calling the FORTRAN subrutine written by White (1985a), modified to be called by R. The `REGE` does the same, however it is written in R. The functions with and without ".for" differ only in whether they are implemented in R of FORTRAN. Needless to say, the functions implemented in FORTRAN are much faster. `REGE.ow`, `REGE.ow.for` - The above function, modified so that a best match is searched for each arc separately (and not for both arcs, if they exist, together). `REGE.nm.for` - REGE or REGGE, modified to use row and column normalized matrices instead of the original matrix. `REGE.ownm.for` - The above function, modified so that a best match for an outgoing ties is searched on row-normalized network and for incoming ties on column-normalized network. `REGD.for` - REGD or REGDI, a dissimilarity version of the classical REGE or REGGE. Dissimilarities in terms of regular equivalence are computed. The `REGD.for` is a wrapper for calling the FORTRAN subroutine written by White (1985b), modified to be called by R. `REGE.FC` - Actually an earlier version of REGE. The difference is in the denominator. See Žiberna (2007) for details. `REGE.FC.ow` - The above function, modified so that a best match is searched for each arc separately (and not for both arcs, if they exist, together). other - still in testing stage.

Usage

```
REGE.FC(
  M,
  E = 1,
  iter = 3,
  until.change = TRUE,
  use.diag = TRUE,
  normE = FALSE
)

REGE.FC.ow(
  M,
  E = 1,
  iter = 3,
  until.change = TRUE,
  use.diag = TRUE,
  normE = FALSE
)

REGE(M, E = 1, iter = 3, until.change = TRUE, use.diag = TRUE)

REGE.ow(M, E = 1, iter = 3, until.change = TRUE, use.diag = TRUE)
```

```
REGE.for(M, iter = 3, E = 1)

REGD.for(M, iter = 3, E = 0)

REGE.ow.for(M, iter = 3, E = 1)

REGD.ow.for(M, iter = 3, E = 0)

REGE.ownm.for(M, iter = 3, E = 1)

REGE.ownm.diag.for(M, iter = 3, E = 1)

REGE.nm.for(M, iter = 3, E = 1)

REGE.nm.diag.for(M, iter = 3, E = 1)

REGE.ne.for(M, iter = 3, E = 1)

REGE.ow.ne.for(M, iter = 3, E = 1)

REGE.ownm.ne.for(M, iter = 3, E = 1)

REGE.nm.ne.for(M, iter = 3, E = 1)

REGD.ne.for(M, iter = 3, E = 0)

REGD.ow.ne.for(M, iter = 3, E = 0)
```

Arguments

M	Matrix or a 3 dimensional array representing the network. The third dimension allows for several relations to be analyzed.
E	Initial (dis)similarity in terms of regular equivalnece.
iter	The desired number of iterations.
until.change	Should the iterations be stopped when no change occurs.
use.diag	Should the diagonal be used. If FALSE, all diagonal elements are set to 0.
normE	Should the equivalence matrix be normalized after each iteration.

Value

E	A matrix of (dis)similarities in terms of regular equivalnece.
Eall	An array of (dis)similarity matrices in terms of regular equivalence, each third dimension represets one iteration. For ".for" functions, only the initial and the final (dis)similarities are returned.
M	Matrix or a 3 dimensional array representing the network used in the call.
iter	The desired number of iterations.

use.diag Should the diagonal be used - for functions implemented in R only.

...

References

Žiberna, A. (2008). Direct and indirect approaches to blockmodeling of valued networks in terms of regular equivalence. Journal of Mathematical Sociology, 32(1), 57-84. doi: 10.1080/00222500701790207

White, D. R., & Reitz, K. P. (1983). Graph and semigroup homomorphisms on networks of relations. Social Networks, 5(2), 193-234.

White, D. R.(1985a). DOUG WHITE'S REGULAR EQUIVALENCE PROGRAM. Retrieved from http://eclectic.ss.uci.edu/~drwhite/REGGE/REGGE.FOR

White, D. R. (1985b). DOUG WHITE'S REGULAR DISTANCES PROGRAM. Retrieved from http://eclectic.ss.uci.edu/~drwhite/REGGE/REGDI.FOR

White, D. R. (2005). REGGE. Retrieved from http://eclectic.ss.uci.edu/~drwhite/REGGE/

#' @author Aleš Žiberna based on Douglas R. White's original REGE and REGD

See Also

sedist, critFunC, optParC, plot.mat

Examples

```
n <- 20
net <- matrix(NA, ncol = n, nrow = n)
clu <- rep(1:2, times = c(5, 15))
tclu <- table(clu)
net[clu == 1, clu == 1] <- 0
net[clu == 1, clu == 2] <- rnorm(n = tclu[1] * tclu[2], mean = 4, sd = 1) * sample(c(0, 1),
   size = tclu[1] * tclu[2], replace = TRUE, prob = c(3/5, 2/5))
net[clu == 2, clu == 1] <- 0
net[clu == 2, clu == 2] <- 0

D <- REGE.for(M = net)$E # Any other REGE function can be used
plot.mat(net, clu = cutree(hclust(d = as.dist(1 - D), method = "ward.D"),
   k = 2))
# REGE returns similarities, which have to be converted to
# disimilarities

res <- optRandomParC(M = net, k = 2, rep = 10, approaches = "hom", homFun = "ss", blocks = "reg")
plot(res) # Hopefully we get the original partition
```

relInv *Functions for computing "relative inverse" (x[1]/x).*

Description

For a vector x, it computes x[1]/x. For relInv2, if certain elements of the result are not finite (e.g. if certain elements of x are 0), these elements are replaced with 0s.

Usage

```
relInv(x)

relInv2(x)
```

Arguments

x	A numeric vector. For relInv it should not contain 0s (while for relInv2 it can).

Value

A vector computed as x[1]/x. For relInv2, if the non-finite elements are replaced with 0s.

Author(s)

Aleš Žiberna

reorderImage *Reordering an image matrix of the blockmodel (or an error matrix based on new and old partition*

Description

Reorders an image matrix of the blockmodel (or an error matrix based on new and old partition. The partitions should be the same, except that classes can have different labels. It is useful when we want to have a different order of classes in figures and then also in image matrices. Currently it is only suitable for one-mode blockmodels.

Usage

```
reorderImage(IM, oldClu, newClu)
```

Arguments

IM	An image or error matrix.
oldClu	Old partition.
newClu	New partition, the same as the old one except for class labeles.

Value

Reorder matrix (rows and columns are reordred).

Author(s)

Ales Ziberna

References

Žiberna, A. (2007). Generalized Blockmodeling of Valued Networks. Social Networks, 29(1), 105-126. doi: 10.1016/j.socnet.2006.04.002

Žiberna, A. (2008). Direct and indirect approaches to blockmodeling of valued networks in terms of regular equivalence. Journal of Mathematical Sociology, 32(1), 57-84. doi: 10.1080/00222500701790207

See Also

`critFunC`, `plot.mat`, `clu`, `IM`, `err`

RF *Calculate the value of the Relative Fit function*

Description

The function calculates the value of the Relative Fit function. Currently implemented only for one-relational one-mode or two-mode networks.

Usage

```
RF(res, m = 10, loops = NULL)
```

Arguments

`res`	An object returned by the function `optRandomParC`.
`m`	The number of randomized networks for the estimation of the expected value of a criterion function. It has to be as high as possible. Defaults to 10.
`loops`	Whether loops are treated the same as any other values or not.

Details

The function randomizes an empirical network to compute the value of the Relative Fit function. The networks are randomized in such a way that the values on the links are randomly relocated. Other approaches to randomization also exist and might be more appropriate in some cases, see Cugmas et al. (2021).

Value

- `RF` - The value of the Relative Fit function.
- `err` - The value of a criterion function that is used for blockmodeling (for empirical network).
- `rand.err` - A vector with the values of the criterion function that is used for blockmodeling (for randomized networks).

Author(s)

Marjan Cugmas and Aleš Žiberna

References

Cugmas, M., Žiberna, A., & Ferligoj, A. (2021). The Relative Fit measure for evaluating a blockmodel. Statistical Methods & Applications, 30(5), 1315-1335. doi:10.1007/s10260021005951

See Also

`optRandomParC`

Examples

```
n <- 8 # If larger, the number of partitions increases
# dramatically as does if we increase the number of clusters
net <- matrix(NA, ncol = n, nrow = n)
clu <- rep(1:2, times = c(3, 5))
tclu <- table(clu)
net[clu == 1, clu == 1] <- rnorm(n = tclu[1] * tclu[1], mean = 0, sd = 1)
net[clu == 1, clu == 2] <- rnorm(n = tclu[1] * tclu[2], mean = 4, sd = 1)
net[clu == 2, clu == 1] <- rnorm(n = tclu[2] * tclu[1], mean = 0, sd = 1)
net[clu == 2, clu == 2] <- rnorm(n = tclu[2] * tclu[2], mean = 0, sd = 1)

res <- optRandomParC(M = net, k = 2, rep = 10, approaches = "hom", homFun = "ss", blocks = "com")
RF(res = res, m = 100, loops = TRUE)
```

---

sedist *Computes distances in terms of Structural equivalence (Lorrain & White, 1971)*

Description

The functions compute the distances in terms of Structural equivalence (Lorrain and White, 1971) between the units of a one-mode network. Several options for treating the diagonal values are supported.

Usage

```
sedist(
  M,
  method = "default",
  fun = "default",
  fun.on.rows = "default",
  handle.interaction = "switch",
  use = "pairwise.complete.obs",
  ...
)
```

Arguments

M	A matrix representing the (usually valued) network. For now, only one-relational networks are supported. The network must be one-mode.
method	The method used to compute distances - any of the methods allowed by functions dist, "cor" or "cov" (all package::stats) or just "cor" or "cov" (given as a character).
fun	Which function should be used to compute distances (given as a character).
fun.on.rows	For non-standard function - does the function compute measure on rows (such as "cor", "cov",...) of the data matrix (as opposed to computing measure on columns (such as dist).
handle.interaction	How should the interaction between the vertices analysed be handled: "switch" (the default) - assumes that when comparing units i and j, M[i,i] should be compared with M[j,j] and M[i,j] with M[j,i]. These two comparisons are weighted by 2. This should be used with Euclidean distance to get the corrected Euclidean distance with p = 2. "switch2" - the same (alias) "switch1" - the same as above, only that the two comparisons are weighted by 1. This should be used with Euclidean distance to get the corrected Wuclidean distance with p = 1. "ignore" (diagonal) - Diagonal is ignored. This should be used with Euclidean distance to get the corrected Euclidean distance with p = 0. "none" - the matrix is used "as is"
use	For use with methods "cor" and "cov", for other methods (the default option should be used if handle.interaction == "ignore"), "pairwise.complete.obs" are always used, if stats.dist.cor.cov = TRUE.
...	Additional arguments to fun

Details

If both method and fun are "default", the Euclidean distances are computed. The "default" method for fun = "dist" is "euclidean" and for fun = "cor" "pearson".

Value

A matrix (usually of class dist) is returned.

Author(s)

Aleš Žiberna

References

Batagelj, V., Ferligoj, A., & Doreian, P. (1992). Direct and indirect methods for structural equivalence. Social Networks, 14(1-2), 63-90. doi: 10.1016/0378-8733(92)90014-X

Lorrain, F., & White, H. C. (1971). Structural equivalence of individuals in social networks. Journal of Mathematical Sociology, 1(1), 49-80. doi: 10.1080/0022250X.1971.9989788

See Also

dist, hclust, REGE, optParC, optParC, optRandomParC

Examples

```
# Generating a simple network corresponding to the simple Sum of squares
# Structural equivalence with blockmodel:
# null com
# null null
n <- 20
net <- matrix(NA, ncol = n, nrow = n)
clu <- rep(1:2, times = c(5, 15))
tclu <- table(clu)
net[clu == 1, clu == 1] <- rnorm(n = tclu[1] * tclu[1], mean = 0, sd = 1)
net[clu == 1, clu == 2] <- rnorm(n = tclu[1] * tclu[2], mean = 4, sd = 1)
net[clu == 2, clu == 1] <- rnorm(n = tclu[2] * tclu[1], mean = 0, sd = 1)
net[clu == 2, clu == 2] <- rnorm(n = tclu[2] * tclu[2], mean = 0, sd = 1)

D <- sedist(M = net)
plot.mat(net, clu = cutree(hclust(d = D, method = "ward"), k = 2))
```

---

splitClu — *Functions creating a list of partitions based on a single partition and information on the number of units in each set.*

Description

Function splitClu creates a list of partitions based on a single partition (clu) and information on the number of units in each set (n).

Function splitCluRes does the same but extracts the information from the result of (old versions of) functions critFunC, optParC, optRandomParC or similar (newer versions should already return a list of partitions in case they are used on networks with more sets of units.

Usage

```
splitClu(clu, n, renumber = FALSE)

splitCluRes(res, renumber = FALSE)
```

Arguments

clu	A vector representing a partition of units from different sets. Result of some legacy code for optRandomParC or optParC or similar functions.
n	A vector with number of units per set. The assuption is that the first n[1] elements of clu are for the first set, the second n[2] elements of clu are for the second set and so on. sum(n) must be equal to length(clu).
renumber	If TRUE, elements of each partition (for each set) in the list are renumbered to be from 1:"number of clusters" in that partition). Defaults to FALSE.
res	Result of (old versions of) functions critFunC, optParC, optRandomParC or similar.

Value

A list of partitions if clu, one for each set of units. A single vector if only one set of units is present.

Author(s)

Aleš Žiberna

See Also

clu, unlistClu, unlistCluInt

Examples

```
n <- c(8,8)
clu <- c(rep(1:2, times = c(3, 5)), rep(3:4, times = c(3, 5)))
splitClu(clu = clu, n = n )
splitClu(clu = clu, n = n, renumber = TRUE)
```

---

ss *Sum of Squared deviations from the mean and sum of Absolute Deviations from the median*

Description

Functions to compute Sum of Squared deviations from the mean and sum of Absolute Deviations from the median. ssNa removes missing values (NAs) before calling the ss function.

Usage

```
ss(x)

ssNa(x)

ad(x)
```

Arguments

x	A numeric vector.

Value

Sum of Squared deviations from the mean or sum of Absolute Deviations from the median.

Author(s)

Aleš Žiberna

---

unlistClu *Function for "unlisting" a partition.*

Description

Essentially, if the argument is a list (otherwise function just returns its argument), the function calls unlist on it. Before it, it however makes sure that names from different elements of the list to not repeat. The opposite of splitClu. The n argument of the splitClu is returned as an attribute. If renumber=TRUE (default), it is practically identical to unlistCluInt.

Usage

```
unlistClu(clu, renumber = FALSE)
```

Arguments

clu	A list representing a partition of units from different sets. Each element should be a partition for one set.
renumber	If TRUE (default), are renumbered so that they are 1:"total number of clusters". If any cluster "ID" is present in more than one set of units (one partition, one element of the list), this is done even if renumber = FALSE.

Value

A vector representing a partition. It also has an attribute n with the number of units that were in each set.

Author(s)

Aleš Žiberna

See Also

clu, splitClu, unlistCluInt

Examples

```
n <- c(8,8)
cluList <- c(rep(1:2, times = c(3, 5)), rep(5:6, times = c(3, 5)))
unlistClu(clu = clu)
unlistClu(clu = clu, renumber = FALSE)
```

---

unlistCluInt *Unlist a partition.*

Description

It is used to convert a partition by sets into a single "simple" partition. Simple partition is a partition of only one set, that is a vector where units with the same value are considered to belong to the same cluster. The partitions by sets is a list, where each element of a list is a "simple" partition that corresponds to one set. The function first converts all elements of the lists to integers, that makes sure that each set uses different integers and on the end uses unlist function on such list.

Usage

```
unlistCluInt(clu)
```

Arguments

clu	A partition by sets, that is a list of "simple" partitions.

Value

The unlisted partition - one vector containing only integers.

See Also

clu, splitClu, unlistClu

Examples

```
cluList<-list(c("a","b","a"),c("b","c","b","c"))
unlistCluInt(cluList)

cluList<-list(c(1,1,1,2,2,2),c(1,1,1,2,2,2,3,3))
unlistCluInt(cluList)
```

Index