Package 'WGCNA'

May 12, 2010

Version 0.88-2

Date 2010-05-12

Title Weighted Gene Co-Expression Network Analysis

Author Peter Langfelder <Peter.Langfelder@gmail.com> and Steve Horvath <SHorvath@mednet.ucla.edu>

Maintainer Peter Langfelder <Peter.Langfelder@gmail.com>

Depends R (>= 2.3.0), stats, impute, grDevices, dynamicTreeCut (>= 1.20), utils, flashClust, qvalue, Hmisc, splines

Suggests GO.db, org.Hs.eg.db, org.Mm.eg.db, AnnotationDbi

ZipData no

License GPL (>= 2)

Description Functions necessary to perform Weighted Gene Co-Expression Network Analysis

URL http://www.genetics.ucla.edu/labs/horvath/CoexpressionNetwork/BranchCutting/

R topics documented:

`WGCNA-package` *Weighted Gene Co-Expression Network Analysis*

Description

Functions necessary to perform Weighted Correlation Network Analysis. WGCNA is also known as weighted gene co-expression network analysis when dealing with gene expression data. Many functions of WGCNA can also be used for general association networks specified by a symmetric adjacency matrix.

Details

Package:	WGCNA
Version:	0.88-2
Date:	2010-05-12
Depends:	R (>= 2.3.0), stats, fields, impute, grDevices, dynamicTr
Suggests: qvalue, AnnotationDbi, GO.db, org.*.eg.db ZipData:	no
License:	GPL (>= 2)
URL:	http://www.genetics.ucla.edu/labs/horvath/Coexpression

Index:

```
blockwiseModules        Automatic network construction and module
                        detection
checkAdjMat             Check adjacency matrix
checkSets               Check structure and retrieve sizes of a group
                        of datasets
clusterCoef             Clustering coefficient calculation
collectGarbage          Iterative garbage collection
colQuantileC            Fast colunm-wise quantile of a matrix
consensusMEDissimilarity
                        Consensus dissimilarity of module eigengenes.
consensusOrderMEs       Put close eigenvectors next to each other in
                        several sets.
consensusProjectiveKMeans
                        Consensus projective K-means (pre-)clustering
                        of expression data
cor                     Faster calculation of Pearson correlations
corAndPvalue            Correlation and the associated p-value
cor1                    Faster calculation of column correlations of a matrix
corFast                 Faster calculation of Pearson correlations
corPredictionSuccess    ~~function to do ... ~~
corPvalueFisher         Fisher's asymptotic p-value for correlation
corPvalueStudent        Student asymptotic p-value for correlation
correlationPreservation
                        Preservation of eigengene correlations
cutreeStatic            Constant height tree cut
cutreeStaticColor       Constant height tree cut using color labels
displayColors           Show colors used to label modules
dynamicMergeCut         Threshold for module merging
exportNetworkToVisANT   Export network data in format readable by VisANT
exportNetworkToCytoscape
                        Export network data in format readable by Cytoscape
fixDataStructure        Put single-set data into a form useful for multiset calc
fundamentalNetworkConcepts
Calculation of fundamental network concepts
GOenrichmentAnalysis    Calculate enrichment p-values of clusters in GO terms
goodGenes               Filter genes with too many missing entries
goodGenesMS             Filter genes with too many missing entries across multip
goodSamples             Filter samples with too many missing entries
goodSamplesGenes        Iterative filtering of samples and genes with
                        too many missing entries
goodSamplesGenesMS      Iterative filtering of samples and genes with too many m
                        multiple data sets
goodSamplesMS           Filter samples with too many missing entries across mult
greenBlackRed           Green-black-red color sequence
greenWhiteRed           Green-white-red color sequence
hubGeneSignificance     Hubgene significance
initProgInd             Inline display of progress
intramodularConnectivity
                        Calculation of intramodular connectivity
keepCommonProbes        Keep probes that are shared among given data sets
labeledBarplot          Barplot with text or color labels
labeledHeatmap          Produce a labeled heatmap plot
```

```
labelPoints Attempt to intelligently label points in a scatterplot
labels2colors           Convert numerical labels to colors
matchLabels             Relabel modules to best approximate a reference labeling
mergeCloseModules       Merge close modules in gene expression data
moduleColor.getMEprefix
                        Get the prefix used to label module eigengenes
moduleEigengenes        Calculate module eigengenes
moduleNumber            Fixed-height cut of a dendrogram
modulePreservation Calculation of module preservation statistics
multiSetMEs             Calculate module eigengenes
nPresent                Number of present data entries
nearestNeighborConnectivity
                        Connectivity to a constant number of nearest neighbors
nearestNeighborConnectivityMS
                        Connectivity to a constant number of nearest
                        neighbors across multiple data sets
networkConcepts         Calculations of network concepts
networkScreening        ~~function to do ... ~~
networkScreeningGS      ~~function to do ... ~~
normalizeLabels         Transform numerical labels into normal order
numbers2colors          Color representation for a numeric variable
orderMEs                Put close eigenvectors next to each other
overlapTable            Overlap counts and Fisher exact tests for two sets of mo
pickHardThreshold       Analysis of scale free topology for
                        hard-thresholding.
pickSoftThreshold       Analysis of scale free topology for
                        soft-thresholding
plotClusterTreeSamples
                        Annotated clustering dendrogram of microarray samples
plotColorUnderTree      Plot color rows under a dendrogram
plotDendroAndColors     Dendrogram plot with color annotation of objects
plotEigengeneNetworks   Eigengene network plot
plotMEpairs             Pairwise scatterplots of eigengenes
plotModuleSignificance
                        Barplot of module significance
plotNetworkHeatmap      Network heatmap plot
preservationNetworkConnectivity
                        Network preservation calculations
projectiveKMeans        Projective K-means (pre-)clustering of
                        expression data
propVarExplained        Proportion of variance explained by eigengenes
randIndex               ~~function to do ... ~~
recutBlockwiseTrees     Repeat blockwise module detection from
                        pre-calculated data
recutConsensusTrees     Repeat blockwise consensus module detection
                        from pre-calculated data
redWhiteGreen           Red-white-green color sequence
relativeCorPredictionSuccess
                        ~~function to do ... ~~
removeGreyME            Removes the grey eigengene from a given
                        collection of eigengenes.
scaleFreeFitIndex Calculation of fitting statistics for evaluating scale free to
```

```
scaleFreePlot           Visual check of scale-free topology
setCorrelationPreservation
                        Summary correlation preservation measure
sigmoidAdjacencyFunction
                        Sigmoid-type adacency function
signedKME               Signed eigengene-based connectivity
signumAdjacencyFunction
                        Hard-thresholding adjacency function
simulateDatExpr         Simulation of expression data
simulateDatExpr5Modules

simulateEigengeneNetwork
                        Simulate eigengene network from a causal model
simulateModule          Simulate a gene co-expression module
simulateMultiExpr       ~~function to do ... ~~
simulateSmallLayer      ~~function to do ... ~~
sizeGrWindow            Open a graphics window of given width and height
softConnectivity        Calculation of soft (weighted) connectevity
spaste                  Space-less paste
standardColors          Colors this library uses for labeling modules
standardScreeningBinaryTrait
                        Standard screening for a binary trait
standardScreeningCensoredTime
                        Standard screening with regard to a Censored Time Variab
stdErr                  ~~function to do ... ~~
unsignedAdjacency       ~~function to do ... ~~
vectorTOM               ~~function to do ... ~~
vectorizeMatrix         Turn a matrix into a vector of non-redundant components
verboseBarplot          Barplot with error bars, annotated by Kruskal-Wallis p-v
verboseBoxplot          Boxplot annotated by a Kruskal-Wallis p-value
verboseScatterplot      Scatterplot annotated by regression line and p-value
```

Author(s)

Peter Langfelder <Peter.Langfelder@gmail.com> and Steve Horvath <SHorvath@mednet.ucla.edu>, with contributions by Jun Dong, Andy Yip, and Bin Zhang.

Maintainer: Peter Langfelder <Peter.Langfelder@gmail.com>

References

Peter Langfelder and Steve Horvath (2008) WGCNA: an R package for weighted correlation network analysis. BMC Bioinformatics 2008, 9:559

Bin Zhang and Steve Horvath (2005) "A General Framework for Weighted Gene Co-Expression Network Analysis", Statistical Applications in Genetics and Molecular Biology: Vol. 4: No. 1, Article 17

Dong J, Horvath S (2007) Understanding Network Concepts in Modules, BMC Systems Biology 2007, 1:24

Horvath S, Dong J (2008) Geometric Interpretation of Gene Coexpression Network Analysis. PLoS Comput Biol 4(8): e1000117

Yip A, Horvath S (2007) Gene network interconnectedness and the generalized topological overlap measure. BMC Bioinformatics 2007, 8:22

Langfelder P, Zhang B, Horvath S (2007) Defining clusters from a hierarchical cluster tree: the Dynamic Tree Cut library for R. Bioinformatics. November/btm563

Langfelder P, Horvath S (2007) Eigengene networks for studying the relationships between co-expression modules. BMC Systems Biology 2007, 1:54

`addErrorBars` *Add error bars to a barplot.*

Description

This function adds error bars to an existing barplot.

Usage

```
addErrorBars(means, errors, two.side = FALSE)
```

Arguments

`means`	vector of means plotted in the barplot
`errors`	vector of standard errors (signle positive values) to be plotted.
`two.side`	should the error bars be two-sided?

Value

None.

Author(s)

Steve Horvath and Peter Langfelder

`addGrid` *Add grid lines to an existing plot.*

Description

This function adds horizontal and/or vertical grid lines to an existing plot. The grid lines are aligned with tick marks.

Usage

```
addGrid(linesPerTick = NULL, horiz = TRUE, vert = FALSE, col = "grey30", lty = 3
```

Arguments

`linesPerTick`	Number of lines between successive tick marks (including the line on the tick-marks themselves)
`horiz`	Draw horizontal grid lines?
`vert`	Draw vertical tick lines?
`col`	Specifies color of the grid lines
`lty`	Specifies line type of grid lines. See `par`.

Details

If `linesPerTick` is not specified, it is set to 5 if number of tick s is 5 or less, and it is set to 2 if number of ticks is greater than 5.

Note

The function does not work whenever logarithmic scales are in use.

Author(s)

Peter Langfelder

Examples

```
plot(c(1:10), c(1:10))
addGrid();
```

---

addGuideLines *Add vertical "guide lines" to a dendrogram plot*

Description

Adds vertical "guide lines" to a dendrogram plot.

Usage

```
addGuideLines(dendro,
              all = FALSE,
              count = 50,
              positions = NULL,
              col = "grey30",
              lty = 3,
              hang = 0)
```

Arguments

`dendro`	The dendrogram (see `hclust`) to which the guide lines are to be added.
`all`	Add a guide line to every object on the dendrogram? Useful if the number of objects is relatively low.
`count`	Number of guide lines to be plotted. The lines will be equidistantly spaced.
`positions`	Horizontal positions of the added guide lines. If given, overrides `count`.
`col`	Color of the guide lines
`lty`	Line type of the guide lines. See `par`.
`hang`	Fraction of the figure height that will separate top ends of guide lines and the merge heights of the corresponding objects.

Author(s)

Peter Langfelder

addTraitToMEs *Add trait information to multi-set module eigengene structure*

Description

Adds trait information to multi-set module eigengene structure.

Usage

```
addTraitToMEs(multiME, multiTraits)
```

Arguments

multiME	Module eigengenes in multi-set format. A vector of lists, one list per set. Each list must contain an element named data that is a data frame with module eigengenes.
multiTraits	Microarray sample trait(s) in multi-set format. A vector of lists, one list per set. Each list must contain an element named data that is a data frame in which each column corresponds to a trait, and each row to an individual sample.

Details

The function simply cbind's the module eigengenes and traits for each set. The number of sets and numbers of samples in each set must be consistent between multiMEs and multiTraits.

Value

A multi-set structure analogous to the input: a vector of lists, one list per set. Each list will contain a component data with the merged eigengenes and traits for the corresponding set.

Author(s)

Peter Langfelder

See Also

checkSets, moduleEigengenes

adjacency *Calculate network adjacency*

Description

Calculates network adjacency from given expression data.

Usage

```
adjacency(datExpr, selectCols = NULL, power = 6, type = "unsigned", corFnc = "co
```

Arguments

datExpr	data frame containing expression data. Columns correspond to genes and rows to samples.
selectCols	can be used to select genes whose adjacencies will be calculated. Should be either a numeric vector giving the indices of the genes to be used, or a boolean vector indicating which genes are to be used.
power	soft thresholding power.
type	network type. Allowed values are (unique abbreviations of) "unsigned", "signed", "signed hybrid".
corFnc	character string specifying the function to be used to calculate co-expression similarity. Defaults to Pearson correlation. Any function returning values between -1 and 1 can be used.
corOptions	character string specifying additional arguments to be passed to the function given by corFnc. Use "use = 'p', method = 'Spearman'" to obtain Spearman correlation.

Details

The function calculates the similarity of columns (genes) in datExpr by calling the function given in corFnc, transforms the similarity according to type and raises it to power, resulting in a weighted network adjacency matrix. If selectCols is given, the corFnc function will be given arguments (datExpr, datExpr[selectCols], ...); hence the returned adjacency will have rows corresponding to all genes and columns corresponding to genes selected by selectCols.

Value

Adjacency matrix of dimensions nrow(datExpr) times nrow(datExpr). If selectCols was given, the number of columns will be the length (if numeric) or sum (if boolean) of selectCols.

Author(s)

Peter Langfelder and Steve Horvath

References

Bin Zhang and Steve Horvath (2005) A General Framework for Weighted Gene Co-Expression Network Analysis, Statistical Applications in Genetics and Molecular Biology, Vol. 4 No. 1, Article 17

Langfelder P, Horvath S (2007) Eigengene networks for studying the relationships between co-expression modules. BMC Systems Biology 2007, 1:54

alignExpr *Align expression data with given vector*

Description

Multiplies genes (columns) in given expression data such that their correlation with given reference vector is non-negative.

Usage

```
alignExpr(datExpr, y = NULL)
```

Arguments

datExpr	expression data to be aligned. A data frame with columns corresponding to genes and rows to samples.
y	reference vector of length equal the number of samples (rows) in datExpr

Details

The function basically multiplies each column in datExpr by the sign of its correlation with y. If y is not given, the first column in datExpr will be used as the reference vector.

Value

A data frame containing the aligned expression data, of the same dimensions as the input data frame.

Author(s)

Steve Horvath and Peter Langfelder

automaticNetworkScreening *~~function to do ... ~~*

Description

~~ A concise (1-5 lines) description of what the function does. ~~

Usage

```
automaticNetworkScreening(datExpr, y, power = 6, networkType = "unsigned", detec
minModuleSize = min(20, ncol(as.matrix(datExpr))/2), datME = NULL, getQValues =
```

Arguments

`datExpr`	~~Describe `datExpr` here~~
`y`	~~Describe `y` here~~
`power`	~~Describe `power` here~~
`networkType`	~~Describe `networkType` here~~
`detectCutHeight`	~~Describe `detectCutHeight` here~~
`minModuleSize`	~~Describe `minModuleSize` here~~
`datME`	~~Describe `datME` here~~
`getQValues`	~~Describe `datME` here~~
`...`	~~Describe `...` here~~

Details

~~ If necessary, more details than the description above ~~

Value

~Describe the value returned If it is a LIST, use

`comp1`	Description of 'comp1'
`comp2`	Description of 'comp2'

...

Note

~~further notes~~

Author(s)

~~who you are~~

References

~put references to the literature/web site here ~

See Also

~~objects to See Also as `help`, ~~~

Examples

```
# Nothing here yet.
```

`automaticNetworkScreeningGS` *One-step automatic network gene screening with external gene significance*

Description

This function performs gene screening based on external gene significance and their network properties.

Usage

```
automaticNetworkScreeningGS(
     datExpr, GS,
     power = 6, networkType = "unsigned",
     detectCutHeight = 0.995, minModuleSize = min(20, ncol(as.matrix(datExpr))/2
     datME = NULL)
```

Arguments

`datExpr`	data frame containing the expression data, columns corresponding to genes and rows to samples
`GS`	vector containing gene significance for all genes given in `datExpr`
`power`	soft thresholding power used in network construction
`networkType`	character string specifying network type. Allowed values are (unique abbreviations of) `"unsigned"`, `"signed"`, `"hybrid"`.
`detectCutHeight`	cut height of the gene hierarchical clustering dendrogram. See `cutreeDynamic` for details.
`minModuleSize`	minimum module size to be used in module detection procedure.
`datME`	optional specification of module eigengenes. A data frame whose columns are the module eigengenes. If given, module analysis will not be performed.

Details

Network screening is a method for identifying genes that have a high gene significance and are members of important modules at the same time. If `datME` is given, the function calls `networkScreeningGS` with the default parameters. If `datME` is not given, module eigengenes are first calculated using network analysis based on supplied parameters.

Value

A list with the following components:

`networkScreening`	a data frame containing results of the network screening procedure. See `networkScreeningGS` for more details.
`datME`	calculated module eigengenes (or a copy of the input `datME`, if given).
`hubGeneSignificance`	hub gene significance for all calculated modules. See `hubGeneSignificance`.

Author(s)

Steve Horvath

See Also

networkScreening, hubGeneSignificance, networkScreening, cutreeDynamic

---

bicor *Biweight Midcorrelation*

Description

Calculate biweight midcorrelation efficiently for matrices.

Usage

```
bicor(x, y = NULL,
      robustX = TRUE, robustY = TRUE,
      use = "all.obs",
      maxPOutliers = 1,
      quick = 0,
      pearsonFallback = "individual",
      nThreads = 0,
      verbose = 0, indent = 0)
```

Arguments

x	a vector or matrix-like numeric object
y	a vector or matrix-like numeric object
robustX	use robust calculation for x?
robustY	use robust calculation for y?
use	specifies handling of NAs. One of (unique abbreviations of) "all.obs", "pairwise.complete.obs".
maxPOutliers	specifies the maximum percentile of data that can be considered outliers on either side of the median separately. For each side of the median, if higher percentile than maxPOutliers is considered an outlier by the weight function based on 9*mad(x), the width of the weight function is increased such that the percentile of outliers on that side of the median equals maxPOutliers. Using maxPOutliers=1 will effectively disable all weight function broadening; using maxPOutliers=0 will give results that are quite similar (but not equal to) Pearson correlation.
quick	real number between 0 and 1 that controls the handling of missing data in the calculation of correlations. See details.
nThreads	non-negative integer specifying the number of parallel threads to be used by certain parts of correlation calculations. This option only has an effect on systems on which a POSIX thread library is available (which currently includes Linux and Mac OSX, but excludes Windows). If zero, the number of online processors will be used if it can be determined dynamically, otherwise correlation calculations will use 2 threads.

pearsonFallback
: Specifies whether the bicor calculation should revert to Pearson when median absolute deviation (mad) is zero. Recongnized values are (abbreviations of) "none", "individual", "all". If set to "none", zero mad will result in NA for the corresponding correlation. If set to "individual", Pearson calculation will be used only for columns that have zero mad. If set to "all", the presence of a single zero mad will cause the whole variable to be treated in Pearson correlation manner (as if the corresponding robust option was set to FALSE).

verbose
: if non-zero, the underlying C function will print some diagnostics.

indent
: indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

This function implements biweight midcorrelation calculation (see references). If y is not supplied, midcorrelation of columns of x will be calculated; otherwise, the midcorrelation between columns of x and y will be calculated. Thus, bicor(x) is equivalent to bicor(x,x) but is more efficient.

The options robustX, robustY allow the user to revert the calculation to standard correlation calculation. This is important, for example, if any of the variables is binary (or, more generally, discrete) as in such cases the robust methods produce meaningless results. If both robustX, robustY are set to FALSE, the function calculates the standard Pearson correlation (but is slower than the function cor).

The argument quick specifies the precision of handling of missing data in the correlation calculations. Value quick = 0 will cause all calculations to be executed accurately, which may be significantly slower than calculations without missing data. Progressively higher values will speed up the calculations but introduce progressively larger errors. Without missing data, all column meadians and median absolute deviations (MADs) can be pre-calculated before the covariances are calculated. When missing data are present, exact calculations require the column medians and MADs to be calculated for each covariance. The approximate calculation uses the pre-calculated median and MAD and simply ignores missing data in the covariance calculation. If the number of missing data is high, the pre-calculated medians and MADs may be very different from the actual ones, thus potentially introducing large errors. The quick value times the number of rows specifies the maximum difference in the number of missing entries for median and MAD calculations on the one hand and covariance on the other hand that will be tolerated before a recalculation is triggered. The hope is that if only a few missing data are treated approximately, the error introduced will be small but the potential speedup can be significant.

The choice "all" for pearsonFallback is not fully implemented in the sense that there are rare but possible cases in which the calculation is equivalent to "individual". This may happen if the use option is set to "pairwise.complete.obs" and the missing data are arranged such that each individual mad is non-zero, but when two columns are analyzed together, the missing data from both columns may make a mad zero. In such a case, the calculation is treated as Pearson, but other columns will be treated as bicor.

Value

A matrix of biweight midcorrelations. Dimnames on the result are set appropriately.

Author(s)

Peter Langfelder

References

"Dealing with Outliers in Bivariate Data: Robust Correlation", Rich Herrington, http://www.unt.edu/benchmarks/archives

"Introduction to Robust Estimation and Hypothesis Testing", Rand Wilcox, Academic Press, 1997.

"Data Analysis and Regression: A Second Course in Statistics", Mosteller and Tukey, Addison-Wesley, 1977, pp. 203-209.

`bicorAndPvalue` *Calculation of biweight midcorrelations and associated p-values*

Description

A faster, one-step calculation of Student correlation p-values for multiple biweight midcorrelations, properly taking into account the actual number of observations.

Usage

```
bicorAndPvalue(x, y,
               use = "pairwise.complete.obs",
               alternative = c("two.sided", "less", "greater"),
               ...)
```

Arguments

`x`	a vector or a matrix
`y`	a vector or a matrix. If `NULL`, the correlation of columns of `x` will be calculated.
`use`	determines handling of missing data. See `bicor` for details.
`alternative`	specifies the alternative hypothesis and must be (a unique abbreviation of) one of `"two.sided"`, `"greater"` or `"less"`. the initial letter. `"greater"` corresponds to positive association, `"less"` to negative association.
`...`	other arguments to the function `bicor`.

Details

The function calculates the biweight midcorrelations of a matrix or of two matrices and the corresponding Student p-values. The output is not as full-featured as `cor.test`, but can work with matrices as input.

Value

A list with the following components

`bicor`	the calculated correlations
`p`	the Student p-values corresponding to the calculated correlations

Author(s)

Peter Langfelder and Steve Horvath

See Also

bicor for calculation of correlations only;

cor.test for another function for significance test of correlations

---

`blockwiseConsensusModules` *Find consensus modules across several datasets.*

Description

Perform network construction and consensus module detection across several datasets.

Usage

```
blockwiseConsensusModules(
  multiExpr, blocks = NULL,
  maxBlockSize = 5000,
  randomSeed = 12345,
  corType = "pearson",
  power = 6,
  consensusQuantile = 0,
  networkType = "unsigned",
  TOMType = "unsigned",
  TOMDenom = "min",
  scaleTOMs = TRUE, scaleQuantile = 0.95,
  sampleForScaling = TRUE, sampleForScalingFactor = 1000,
  useDiskCache = TRUE, chunkSize = NULL,
  cacheBase = ".blockConsModsCache",
  deepSplit = 2,
  detectCutHeight = 0.995, minModuleSize = 20,
  checkMinModuleSize = TRUE,
  maxCoreScatter = NULL, minGap = NULL,
  maxAbsCoreScatter = NULL, minAbsGap = NULL,
  pamStage = TRUE, pamRespectsDendro = TRUE,
  minKMEtoJoin =0.7,
  minCoreKME = 0.5, minCoreKMESize = minModuleSize/3,
  minKMEtoStay = 0.2,
  reassignThresholdPS = 1e-4,
  mergeCutHeight = 0.15,
  impute = TRUE,
  getTOMs = NULL,
  saveTOMs = FALSE,
  saveTOMFileBase = "consensusTOM",
  getTOMScalingSamples = FALSE,
  trapErrors = FALSE,
  checkPower = TRUE,
  numericLabels = FALSE,
  checkMissingData = TRUE,
  maxPOutliers = 1,
  quickCor = 0,
```

```
  pearsonFallback = "individual",
  nThreads = 0,
  verbose = 2, indent = 0)
```

Arguments

multiExpr	expression data in the multi-set format (see checkSets). A vector of lists, one per set. Each set must contain a component data that contains the expression data, with rows corresponding to samples and columns to genes or probes.
blocks	optional specification of blocks in which hierarchical clustering and module detection should be performed. If given, must be a numeric vector with one entry per gene of multiExpr giving the number of the block to which the corresponding gene belongs.
maxBlockSize	integer giving maximum block size for module detection. Ignored if blocks above is non-NULL. Otherwise, if the number of genes in datExpr exceeds maxBlockSize, genes will be pre-clustered into blocks whose size should not exceed maxBlockSize.
randomSeed	integer to be used as seed for the random number generator before the function starts. If a current seed exists, it is saved and restored upon exit. If NULL is given, the function will not save and restore the seed.
corType	character string specifying the correlation to be used. Allowed values are (unique abbreviations of) "pearson" and "bicor", corresponding to Pearson and bidweight midcorrelation, respectively. Missing values are handled using the pariwise.complete.obs option.
power	soft-thresholding power for netwoek construction.
consensusQuantile	qunatile at which consensus is to be defined. See details.
networkType	network type. Allowed values are (unique abbreviations of) "unsigned", "signed", "signed hybrid". See adjacency.
TOMType	one of "none", "unsigned", "signed". If "none", adjacency will be used for clustering. If "unsigned", the standard TOM will be used (more generally, TOM function will receive the adjacency as input). If "signed", TOM will keep track of the sign of correlations between neighbors.
TOMDenom	a character string specifying the TOM variant to be used. Recognized values are "min" giving the standard TOM described in Zhang and Horvath (2005), and "mean" in which the min function in the denominator is replaced by mean. The "mean" may produce better results but at this time should be considered experimental.
scaleTOMs	should set-specific TOM matrices be scaled to the same scale?
scaleQuantile	if scaleTOMs is TRUE, topological overlaps (or adjacencies if TOMs are not computed) will be scaled such that their scaleQuantile quantiles will agree.
sampleForScaling	if TRUE, scale quantiles will be determined from a sample of network similarities. Note that using all data can double the memory footprint of the function and the function may fail.
sampleForScalingFactor	determines the number of samples for scaling: the number is 1/scaleQuantile * sampleForScalingFactor. Should be set well above 1 to ensure accuracy of the sampled quantile.

useDiskCache should calculated network similarities in individual sets be temporarilly saved to disk? Saving to disk is somewhat slower than keeping all data in memory, but for large blocks and/or many sets the memory footprint may be too big.

chunkSize network similarities are saved in smaller chunks of size chunkSize.

cacheBase character string containing the desired name for the cache files. The actual file names will consists of cacheBase and a suffix to make the file names unique.

deepSplit integer value between 0 and 4. Provides a simplified control over how sensitive module detection should be to module splitting, with 0 least and 4 most sensitive. See cutreeDynamic for more details.

detectCutHeight
dendrogram cut height for module detection. See cutreeDynamic for more details.

minModuleSize
minimum module size for module detection. See cutreeDynamic for more details.

checkMinModuleSize
logical: should sanity checks be performed on minModuleSize?

maxCoreScatter
maximum scatter of the core for a branch to be a cluster, given as the fraction of cutHeight relative to the 5th percentile of joining heights. See cutreeDynamic for more details.

minGap minimum cluster gap given as the fraction of the difference between cutHeight and the 5th percentile of joining heights. See cutreeDynamic for more details.

maxAbsCoreScatter
maximum scatter of the core for a branch to be a cluster given as absolute heights. If given, overrides maxCoreScatter. See cutreeDynamic for more details.

minAbsGap minimum cluster gap given as absolute height difference. If given, overrides minGap. See cutreeDynamic for more details.

pamStage logical. If TRUE, the second (PAM-like) stage of module detection will be performed. See cutreeDynamic for more details.

pamRespectsDendro
Logical, only used when pamStage is TRUE. If TRUE, the PAM stage will respect the dendrogram in the sense an object can be PAM-assigned only to clusters that lie below it on the branch that the object is merged into. See cutreeDynamic for more details.

minKMEtoJoin a number between 0 and 1. Genes with eigengene connectivity higher than minKMEtoJoin are automatically assigned to their closest module.

minCoreKME a number between 0 and 1. If a detected module does not have at least minModuleKMESize genes with eigengene connectivity at least minCoreKME, the module is disbanded (its genes are unlabeled and returned to the pool of genes waiting for mofule detection).

minCoreKMESize
see minCoreKME above.

minKMEtoStay genes whose eigengene connectivity to their module eigengene is lower than minKMEtoStay are removed from the module.

reassignThresholdPS
per-set p-value ratio threshold for reassigning genes between modules. See Details.

mergeCutHeight
: dendrogram cut height for module merging.

impute
: logical: should imputation be used for module eigengene calculation? See moduleEigengenes for more details.

getTOMs
: deprecated, please use saveTOMs below.

saveTOMs
: logical: should the consensus topological overlap matrices for each block be saved and returned?

saveTOMFileBase
: character string containing the file name base for files containing the consensus topological overlaps. The full file names have "block.1.RData", "block.2.RData" etc. appended. These files are standard R data files and can be loaded using the load function.

getTOMScalingSamples
: logical: should samples used for TOM scaling be saved for future analysis? This option is only available when sampleForScaling is TRUE.

trapErrors
: logical: should errors in calculations be trapped?

checkPower
: logical: should basic sanity check be performed on the supplied power? If you would like to experiment with unusual powers, set the argument to FALSE and proceed with caution.

numericLabels
: logical: should the returned modules be labeled by colors (FALSE), or by numbers (TRUE)?

checkMissingData
: logical: should data be checked for excessive numbers of missing entries in genes and samples, and for genes with zero variance? See details.

maxPOutliers
: only used for corType=="bicor". Specifies the maximum percentile of data that can be considered outliers on either side of the median separately. For each side of the median, if higher percentile than maxPOutliers is considered an outlier by the weight function based on 9*mad(x), the width of the weight function is increased such that the percentile of outliers on that side of the median equals maxPOutliers. Using maxPOutliers=1 will effectively disable all weight function broadening; using maxPOutliers=0 will give results that are quite similar (but not equal to) Pearson correlation.

quickCor
: real number between 0 and 1 that controls the handling of missing data in the calculation of correlations. See details.

pearsonFallback
: Specifies whether the bicor calculation, if used, should revert to Pearson when median absolute deviation (mad) is zero. Recongnized values are (abbreviations of) "none", "individual", "all". If set to "none", zero mad will result in NA for the corresponding correlation. If set to "individual", Pearson calculation will be used only for columns that have zero mad. If set to "all", the presence of a single zero mad will cause the whole variable to be treated in Pearson correlation manner (as if the corresponding robust option was set to FALSE). Has no effect for Pearson correlation. See bicor.

nThreads
: non-negative integer specifying the number of parallel threads to be used by certain parts of correlation calculations. This option only has an effect on systems on which a POSIX thread library is available (which currently includes Linux and Mac OSX, but excludes Windows). If zero, the number of online processors will be used if it can be determined dynamically, otherwise correlation calculations will use 2 threads.

verbose	integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
indent	indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

The function starts by optionally filtering out samples that have too many missing entries and genes that have either too many missing entries or zero variance in at least one set. Genes that are filtered out are left unassigned by the module detection. Returned eigengenes will contain NA in entries corresponding to filtered-out samples.

If blocks is not given and the number of genes exceeds maxBlockSize, genes are pre-clustered into blocks using the function consensusProjectiveKMeans; otherwise all genes are treated in a single block.

For each block of genes, the network is constructed and (if requested) topological overlap is calculated in each set. To minimize memory usage, calculated topological overlaps are optionally saved to disk in chunks until they are needed again for the calculation of the consensus network topological overlap. If requested, the consensus topological overlaps are saved to disk for later use. Genes are then clustered using average linkage hierarchical clustering and modules are identified in the resulting dendrogram by the Dynamic Hybrid tree cut. Found modules are trimmed of genes whose correlation with module eigengene (KME) is less than minKMEtoStay in any of the sets. Modules in which fewer than minCoreKMESize genes have KME higher than minCoreKME (in all sets) are disbanded, i.e., their constituent genes are pronounced unassigned. Conversely, any unassigned genes with KME higher than minKMEtoJoin in all sets are automatically assigned to their nearest module.

After all blocks have been processed, the function checks whether there are genes whose KME in the module they assigned is lower than KME to another module. If p-values of the higher correlations are smaller than those of the native module by the factor reassignThresholdPS (in every set), the gene is re-assigned to the closer module.

In the last step, modules whose eigengenes are highly correlated are merged. This is achieved by clustering module eigengenes using the dissimilarity given by one minus their correlation, cutting the dendrogram at the height mergeCutHeight and merging all modules on each branch. The process is iterated until no modules are merged. See mergeCloseModules for more details on module merging.

The argument quick specifies the precision of handling of missing data in the correlation calculations. Zero will cause all calculations to be executed precisely, which may be significantly slower than calculations without missing data. Progressively higher values will speed up the calculations but introduce progressively larger errors. Without missing data, all column means and variances can be pre-calculated before the covariances are calculated. When missing data are present, exact calculations require the column means and variances to be calculated for each covariance. The approximate calculation uses the pre-calculated mean and variance and simply ignores missing data in the covariance calculation. If the number of missing data is high, the pre-calculated means and variances may be very different from the actual ones, thus potentially introducing large errors. The quick value times the number of rows specifies the maximum difference in the number of missing entries for mean and variance calculations on the one hand and covariance on the other hand that will be tolerated before a recalculation is triggered. The hope is that if only a few missing data are treated approximately, the error introduced will be small but the potential speedup can be significant.

Value

A list with the following components:

`colors` module assignment of all input genes. A vector containing either character strings with module colors (if input `numericLabels` was unset) or numeric module labels (if `numericLabels` was set to `TRUE`). The color "grey" and the numeric label 0 are reserved for unassigned genes.

`unmergedColors` module colors or numeric labels before the module merging step.

`multiMEs` module eigengenes corresponding to the modules returned in `colors`, in multi-set format. A vector of lists, one per set, containing eigengenes, proportion of variance explained and other information. See `multiSetMEs` for a detailed description.

`goodSamples` a list, with one component per input set. Each component is a logical vector with one entry per sample from the corresponding set. The entry indicates whether the sample in the set passed basic quality control criteria.

`goodGenes` a logical vector with one entry per input gene indicating whether the gene passed basic quality control criteria in all sets.

`dendrograms` a list with one component for each block of genes. Each component is the hierarchical clustering dendrogram obtained by clustering the consensus gene dissimilarity in the corresponding block.

`TOMFiles` if `saveTOMs==TRUE`, a vector of character strings, one string per block, giving the file names of files (relative to current directory) in which blockwise topological overlaps were saved.

`blockGenes` a list with one component for each block of genes. Each component is a vector giving the indices (relative to the input `multiExpr`) of genes in the corresponding block.

`blocks` if input `blocks` was given, its copy; otherwise a vector of length equal number of genes giving the block label for each gene. Note that block labels are not necessarilly sorted in the order in which the blocks were processed (since we do not require this for the input `blocks`). See `blockOrder` below.

`blockOrder` a vector giving the order in which blocks were processed and in which `blockGenes` above is returned. For example, `blockOrder[1]` contains the label of the first-processed block.

`originCount` if the input `consensusQuantile==0`, this vector will contain counts of how many times each set contributed the consensus gene similarity value. If the counts are highly unbalanced, the consensus may be biased.

`TOMScalingSamples` if the input `getTOMScalingSamples` is `TRUE`, this component is a list with one component per block. Each component is again a list with two components: `sampleIndex` contains indices of the distance structure in which TOM is stored that were sampled, and `TOMSamples` is a matrix whose rows correspond to TOM samples and columns to individual set. Hence, `TOMScalingSamples[[blockNo]]$TO setNo]` contains the TOM entry that corresponds to element `TOMScalingSamples[[blockNo` of the TOM distance structure in block `blockNo` and set `setNo`. (For details on the distance structure, see `dist`.)

Note

If the input datasets have large numbers of genes, consider carefully the `maxBlockSize` as it significantly affects the memory footprint (and whether the function will fail with a memory allocation error). From a theoretical point of view it is advantageous to use blocks as large as possible; on the other hand, using smaller blocks is substantially faster and often the only way to work with large

numbers of genes. As a rough guide, it is unlikely a standard desktop computer with 4GB memory or less will be able to work with blocks larger than 7000 genes.

Author(s)

Peter Langfelder

References

Langfelder P, Horvath S (2007) Eigengene networks for studying the relationships between co-expression modules. BMC Systems Biology 2007, 1:54

See Also

`goodSamplesGenesMS` for basic quality control and filtering;

`adjacency`, `TOMsimilarity` for network construction;

`hclust` for hierarchical clustering;

`cutreeDynamic` for adaptive branch cutting in hierarchical clustering dendrograms;

`mergeCloseModules` for merging of close modules.

`blockwiseModules` *Automatic network construction and module detection*

Description

This function performs automatic network construction and module detection on large expression datasets in a block-wise manner.

Usage

```
blockwiseModules(
  datExpr,
  blocks = NULL,
  maxBlockSize = 5000,
  randomSeed = 12345,
  corType = "pearson",
  power = 6,
  networkType = "unsigned",
  TOMType = "signed",
  TOMDenom = "min",
  deepSplit = 2,
  detectCutHeight = 0.995, minModuleSize = min(20, ncol(datExpr)/2 ),
  maxCoreScatter = NULL, minGap = NULL,
  maxAbsCoreScatter = NULL, minAbsGap = NULL,
  pamStage = TRUE, pamRespectsDendro = TRUE,
  minKMEtoJoin =0.7,
  minCoreKME = 0.5, minCoreKMESize = minModuleSize/3,
  minKMEtoStay = 0.3,
  reassignThreshold = 1e-6,
  mergeCutHeight = 0.15, impute = TRUE,
```

```
  getTOMs = NULL,
  saveTOMs = FALSE,
  saveTOMFileBase = "blockwiseTOM",
  trapErrors = FALSE, numericLabels = FALSE,
  checkMissingData = TRUE,
  maxPOutliers = 1,
  quickCor = 0,
  pearsonFallback = "individual",
  nThreads = 0,
  verbose = 0, indent = 0)
```

Arguments

datExpr	expression data. A data frame in which columns are genes and rows ar samples. NAs are allowed, but not too many.
blocks	optional specification of blocks in which hierarchical clustering and module detection should be performed. If given, must be a numeric vector with one entry per column (gene) of exprData giving the number of the block to which the corresponding gene belongs.
maxBlockSize	integer giving maximum block size for module detection. Ignored if blocks above is non-NULL. Otherwise, if the number of genes in datExpr exceeds maxBlockSize, genes will be pre-clustered into blocks whose size should not exceed maxBlockSize.
randomSeed	integer to be used as seed for the random number generator before the function starts. If a current seed exists, it is saved and restored upon exit. If NULL is given, the function will not save and restore the seed.
corType	character string specifying the correlation to be used. Allowed values are (unique abbreviations of) "pearson" and "bicor", corresponding to Pearson and bidweight midcorrelation, respectively. Missing values are handled using the pairwise.complete.obs option.
power	soft-thresholding power for network construction.
networkType	network type. Allowed values are (unique abbreviations of) "unsigned", "signed", "signed hybrid". See adjacency.
TOMType	one of "none", "unsigned", "signed". If "none", adjacency will be used for clustering. If "unsigned", the standard TOM will be used (more generally, TOM function will receive the adjacency as input). If "signed", TOM will keep track of the sign of correlations between neighbors.
TOMDenom	a character string specifying the TOM variant to be used. Recognized values are "min" giving the standard TOM described in Zhang and Horvath (2005), and "mean" in which the min function in the denominator is replaced by mean. The "mean" may produce better results but at this time should be considered experimental.
deepSplit	integer value between 0 and 4. Provides a simplified control over how sensitive module detection should be to module splitting, with 0 least and 4 most sensitive. See cutreeDynamic for more details.
detectCutHeight	dendrogram cut height for module detection. See cutreeDynamic for more details.

`minModuleSize`
: minimum module size for module detection. See `cutreeDynamic` for more details.

`maxCoreScatter`
: maximum scatter of the core for a branch to be a cluster, given as the fraction of `cutHeight` relative to the 5th percentile of joining heights. See `cutreeDynamic` for more details.

`minGap`
: minimum cluster gap given as the fraction of the difference between `cutHeight` and the 5th percentile of joining heights. See `cutreeDynamic` for more details.

`maxAbsCoreScatter`
: maximum scatter of the core for a branch to be a cluster given as absolute heights. If given, overrides `maxCoreScatter`. See `cutreeDynamic` for more details.

`minAbsGap`
: minimum cluster gap given as absolute height difference. If given, overrides `minGap`. See `cutreeDynamic` for more details.

`pamStage`
: logical. If TRUE, the second (PAM-like) stage of module detection will be performed. See `cutreeDynamic` for more details.

`pamRespectsDendro`
: Logical, only used when `pamStage` is `TRUE`. If `TRUE`, the PAM stage will respect the dendrogram in the sense an object can be PAM-assigned only to clusters that lie below it on the branch that the object is merged into. See `cutreeDynamic` for more details.

`minKMEtoJoin`
: a number between 0 and 1. Genes with eigengene connectivity higher than `minKMEtoJoin` are automatically assigned to their closest module.

`minCoreKME`
: a number between 0 and 1. If a detected module does not have at least `minModuleKMESize` genes with eigengene connectivity at least `minCoreKME`, the module is disbanded (its genes are unlabeled and returned to the pool of genes waiting for mofule detection).

`minCoreKMESize`
: see `minCoreKME` above.

`minKMEtoStay`
: genes whose eigengene connectivity to their module eigengene is lower than `minKMEtoStay` are removed from the module.

`reassignThreshold`
: p-value ratio threshold for reassigning genes between modules. See Details.

`mergeCutHeight`
: dendrogram cut height for module merging.

`impute`
: logical: should imputation be used for module eigengene calculation? See `moduleEigengenes` for more details.

`getTOMs`
: deprecated, please use saveTOMs below.

`saveTOMs`
: logical: should the consensus topological overlap matrices for each block be saved and returned?

`saveTOMFileBase`
: character string containing the file name base for files containing the consensus topological overlaps. The full file names have `"block.1.RData"`, `"block.2.RData"` etc. appended. These files are standard R data files and can be loaded using the `load` function.

`trapErrors`
: logical: should errors in calculations be trapped?

numericLabels
: logical: should the returned modules be labeled by colors (FALSE), or by numbers (TRUE)?

checkMissingData
: logical: should data be checked for excessive numbers of missing entries in genes and samples, and for genes with zero variance? See details.

maxPOutliers
: only used for corType=="bicor". Specifies the maximum percentile of data that can be considered outliers on either side of the median separately. For each side of the median, if higher percentile than maxPOutliers is considered an outlier by the weight function based on 9*mad(x), the width of the weight function is increased such that the percentile of outliers on that side of the median equals maxPOutliers. Using maxPOutliers=1 will effectively disable all weight function broadening; using maxPOutliers=0 will give results that are quite similar (but not equal to) Pearson correlation.

quickCor
: real number between 0 and 1 that controls the handling of missing data in the calculation of correlations. See details.

pearsonFallback
: Specifies whether the bicor calculation, if used, should revert to Pearson when median absolute deviation (mad) is zero. Recongnized values are (abbreviations of) "none", "individual", "all". If set to "none", zero mad will result in NA for the corresponding correlation. If set to "individual", Pearson calculation will be used only for columns that have zero mad. If set to "all", the presence of a single zero mad will cause the whole variable to be treated in Pearson correlation manner (as if the corresponding robust option was set to FALSE). Has no effect for Pearson correlation. See bicor.

nThreads
: non-negative integer specifying the number of parallel threads to be used by certain parts of correlation calculations. This option only has an effect on systems on which a POSIX thread library is available (which currently includes Linux and Mac OSX, but excludes Windows). If zero, the number of online processors will be used if it can be determined dynamically, otherwise correlation calculations will use 2 threads.

verbose
: integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.

indent
: indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

Before module detection starts, genes and samples are optionally checked for the presence of NAs. Genes and/or samples that have too many NAs are flagged as bad and removed from the analysis; bad genes will be automatically labeled as unassigned, while the returned eigengenes will have NA entries for all bad samples.

If blocks is not given and the number of genes exceeds maxBlockSize, genes are pre-clustered into blocks using the function projectiveKMeans; otherwise all genes are treated in a single block.

For each block of genes, the network is constructed and (if requested) topological overlap is calculated. If requested, the topological overlaps are returned as part of the return value list. Genes are then clustered using average linkage hierarchical clustering and modules are identified in the resulting dendrogram by the Dynamic Hybrid tree cut. Found modules are trimmed of genes whose correlation with module eigengene (KME) is less than minKMEtoStay. Modules in which fewer than minCoreKMESize genes have KME higher than minCoreKME are disbanded, i.e., their

constituent genes are pronounced unassigned. Conversely, any unassigned genes with KME higher than `minKMEtoJoin` are automatically assigned to their nearest module.

After all blocks have been processed, the function checks whether there are genes whose KME in the module they assigned is lower than KME to another module. If p-values of the higher correlations are smaller than those of the native module by the factor `reassignThresholdPS`, the gene is re-assigned to the closer module.

In the last step, modules whose eigengenes are highly correlated are merged. This is achieved by clustering module eigengenes using the dissimilarity given by one minus their correlation, cutting the dendrogram at the height `mergeCutHeight` and merging all modules on each branch. The process is iterated until no modules are merged. See `mergeCloseModules` for more details on module merging.

The argument `quick` specifies the precision of handling of missing data in the correlation calculations. Zero will cause all calculations to be executed precisely, which may be significantly slower than calculations without missing data. Progressively higher values will speed up the calculations but introduce progressively larger errors. Without missing data, all column means and variances can be pre-calculated before the covariances are calculated. When missing data are present, exact calculations require the column means and variances to be calculated for each covariance. The approximate calculation uses the pre-calculated mean and variance and simply ignores missing data in the covariance calculation. If the number of missing data is high, the pre-calculated means and variances may be very different from the actual ones, thus potentially introducing large errors. The `quick` value times the number of rows specifies the maximum difference in the number of missing entries for mean and variance calculations on the one hand and covariance on the other hand that will be tolerated before a recalculation is triggered. The hope is that if only a few missing data are treated approximately, the error introduced will be small but the potential speedup can be significant.

Value

A list with the following components:

`colors` a vector of color or numeric module labels for all genes.

`unmergedColors` a vector of color or numeric module labels for all genes before module merging.

`MEs` a data frame containing module eigengenes of the found modules (given by `colors`).

`goodSamples` numeric vector giving indices of good samples, that is samples that do not have too many missing entries.

`goodGenes` numeric vector giving indices of good genes, that is genes that do not have too many missing entries.

`dendrograms` a list whose components conatain hierarchical clustering dendrograms of genes in each block.

`TOMFiles` if `saveTOMs==TRUE`, a vector of character strings, one string per block, giving the file names of files (relative to current directory) in which blockwise topological overlaps were saved.

`blockGenes` a list whose components give the indices of genes in each block.

`blocks` if input `blocks` was given, its copy; otherwise a vector of length equal number of genes giving the block label for each gene. Note that block labels are not necessarilly sorted in the order in which the blocks were processed (since we do not require this for the input `blocks`). See `blockOrder` below.

blockOrder	a vector giving the order in which blocks were processed and in which blockGenes above is returned. For example, blockOrder[1] contains the label of the first-processed block.
MEsOK	logical indicating whether the module eigengenes were calculated without errors.

Note

If the input dataset has a large number of genes, consider carefully the maxBlockSize as it significantly affects the memory footprint (and whether the function will fail with a memory allocation error). From a theoretical point of view it is advantageous to use blocks as large as possible; on the other hand, using smaller blocks is substantially faster and often the only way to work with large numbers of genes. As a rough guide, it is unlikely a standard desktop computer with 4GB memory or less will be able to work with blocks larger than 8000 genes.

Author(s)

Peter Langfelder

References

Bin Zhang and Steve Horvath (2005) "A General Framework for Weighted Gene Co-Expression Network Analysis", Statistical Applications in Genetics and Molecular Biology: Vol. 4: No. 1, Article 17

See Also

goodSamplesGenes for basic quality control and filtering;

adjacency, TOMsimilarity for network construction;

hclust for hierarchical clustering;

cutreeDynamic for adaptive branch cutting in hierarchical clustering dendrograms;

mergeCloseModules for merging of close modules.

---

checkAdjMat *Check adjacency matrix*

Description

Checks a given matrix for properties that an adjacency matrix must satisfy.

Usage

```
checkAdjMat(adjMat, min = 0, max = 1)
```

Arguments

adjMat	matrix to be checked
min	minimum allowed value for entries of adjMat
max	maximum allowed value for entries of adjMat

Details

The function checks whether the given matrix really is a 2-dimensional numeric matrix, whether it is square, symmetric, and all finite entries are between `min` and `max`. If any of the conditions is not met, the function issues an error.

Value

None. The function returns normally if all conditions are met.

Author(s)

Peter Langfelder

See Also

`adjacency`

---

checkSets *Check structure and retrieve sizes of a group of datasets.*

Description

Checks whether given sets have the correct format and retrieves dimensions.

Usage

```
checkSets(data, checkStructure = FALSE, useSets = NULL)
```

Arguments

`data`	A vector of lists; in each list there must be a component named `data` whose content is a matrix or dataframe or array of dimension 2.
`checkStructure`	If `FALSE`, incorrect structure of `data` will trigger an error. If `TRUE`, an appropriate flag (see output) will be set to indicate whether `data` has correct structure.
`useSets`	Optional specification of entries of the vector `data` that are to be checked. Defaults to all components. This may be useful when `data` only contains information for some of the sets.

Details

For multiset calculations, many quantities (such as expression data, traits, module eigengenes etc) are presented by a common structure, a vector of lists (one list for each set) where each list has a component `data` that contains the actual (expression, trait, eigengene) data for the corresponding set in the form of a dataframe. This funtion checks whether `data` conforms to this convention and retrieves some basic dimension information (see output).

Value

A list with components

`nSets`	Number of sets (length of the vector `data`).
`nGenes`	Number of columns in the `data` components in the lists. This number must be the same for all sets.
`nSamples`	A vector of length `nSets` giving the number of rows in the `data` components.
`structureOK`	Only set if the argument `checkStructure` equals `TRUE`. The value is `TRUE` if the paramter `data` passes a few tests of its structure, and `FALSE` otherwise. The tests are not exhaustive and are meant to catch obvious user errors rather than be bulletproof.

Author(s)

Peter Langfelder, `<Peter.Langfelder@gmail.com>`

`clusterCoef` *Clustering coefficient calculation*

Description

This function calculates the clustering coefficients for all nodes in the network given by the input adjacency matrix.

Usage

```
clusterCoef(adjMat)
```

Arguments

`adjMat`	adjacency matrix

Value

A vector of clustering coefficients for each node.

Author(s)

Steve Horvath

collectGarbage *Iterative garbage collection.*

Description

Performs garbage collection until free memory idicators show no change.

Usage

```
collectGarbage()
```

Value

None.

Author(s)

Steve Horvath

colQuantileC *Fast colunm-wise quantile of a matrix.*

Description

Fast calculation of column-wise quantiles of a matrix at a single probability. Implemented via compiled code, it is much faster than the equivalent `apply(data, 2, quantile, prob = p)`.

Usage

```
colQuantileC(data, p)
```

Arguments

`data`	a numerical matrix column-wise quantiles are desired. Missing values are corrently not allowed.
`p`	a single probability at which the quantile is to be calculated.

Value

A vector of length equal the number of columns in `data` containing the column-wise quantiles.

Author(s)

Peter Langfelder

See Also

`quantile`

`conformityBasedNetworkConcepts` *Calculation of conformity-based network concepts.*

Description

This function computes 3 types of network concepts (also known as network indices or statistics) based on an adjacency matrix and optionally a node significance measure.

Usage

```
conformityBasedNetworkConcepts(adj, GS = NULL)
```

Arguments

`adj`	adjacency matrix. A symmetric matrix with components between 0 and 1.
`GS`	optional node significance measure. A vector with length equal the dimension of `adj`.

Details

This function computes 3 types of network concepts (also known as network indices or statistics) based on an adjacency matrix and optionally a node significance measure. Specifically, it computes I) fundamental network concepts, II) conformity based network concepts, and III) approximate conformity based network concepts. These network concepts are defined for any symmetric adjacency matrix (weighted and unweighted). The network concepts are described in Dong and Horvath (2007) and Horvath and Dong (2008). In the following, we use the term gene and node interchangeably since these methods were originally developed for gene networks. In the following, we briefly describe the 3 types of network concepts:

Type I: fundamental network concepts are defined as a function of the off-diagonal elements of an adjacency matrix A and/or a node significance measure GS. Type II: conformity-based network concepts are functions of the off-diagonal elements of the conformity based adjacency matrix A.CF=CF*t(CF) and/or the node significance measure. These network concepts are defined for any network for which a conformity vector can be defined. Details: For any adjacency matrix A, the conformity vector CF is calculated by requiring that A[i,j] is approximately equal to CF[i]*CF[j]. Using the conformity one can define the matrix A.CF=CF*t(CF) which is the outer product of the conformity vector with itself. In general, A.CF is not an adjacency matrix since its diagonal elements are different from 1. If the off-diagonal elements of A.CF are similar to those of A according to the Frobenius matrix norm, then A is approximately factorizable. To measure the factorizability of a network, one can calculate the Factorizability, which is a number between 0 and 1 (Dong and Horvath 2007). The conformity is defined using a monotonic, iterative algorithm that maximizes the factorizability measure. Type III: approximate conformity based network concepts are functions of all elements of the conformity based adjacency matrix A.CF (including the diagonal) and/or the node significance measure GS. These network concepts are very useful for deriving relationships between network concepts in networks that are approximately factorizable.

Value

A list with the following components:

`fundamentalNCs`
: fundamental network concepts, that is network concepts calculated directly from the given adjacency matrix `adj`. A list with components `ScaledConnectivity` (giving the scaled connectivity of each node), `Connectivity` (connectivity of each node), `ClusterCoef` (the clustering coefficient of each node), `MAR` (maximum adjacency ratio of each node), `Density` (the mean density of the network), `Centralization` (the centralization of the network), `Heterogeneity` (the heterogeneity of the network). If the input node significance `GS` is specified, the following additional components are included: `NetworkSignificance` (network significance, the mean node significance), and `HubNodeSignificance` (hub node significance given by the linear regression of node significance on connectivity).

`conformityBasedNCs`
: network concepts based on an approximate adjacency matrix given by the outer product of the conformity vector but with unit diagonal. A list with components `Conformity` (the conformity vector) and `Connectivity.CF, ClusterCoef.CF, MAR.CF, Density.CF, Centralization.CF, Heterogeneity.CF` giving the conformity-based analogs of the above network concepts.

`approximateConformityBasedNCs`
: network concepts based on an approximate adjacency matrix given by the outer product of the conformity vector. A list with components `Conformity` (the conformity vector) and `Connectivity.CF.App, ClusterCoef.CF.App, MAR.CF.App, Density.CF.App, Centralization.CF.App, Heterogeneity.CF` giving the conformity-based analogs of the above network concepts.

Author(s)

Steve Horvath

References

Dong J, Horvath S (2007) Understanding Network Concepts in Modules, BMC Systems Biology 2007, 1:24 Horvath S, Dong J (2008) Geometric Interpretation of Gene Coexpression Network Analysis. PLoS Comput Biol 4(8): e1000117

See Also

`networkConcepts` for calculation of eigennode based network concepts for a correlation network;

`fundamentalNetworkConcepts` for calculation of fundamental network concepts only.

---

consensusMEDissimilarity

Consensus dissimilarity of module eigengenes.

Description

Calculates consensus dissimilarity `(1-cor)` of given module eigengenes relaized in several sets.

Usage

```
consensusMEDissimilarity(MEs, useAbs = FALSE, useSets = NULL, method = "consensu
```

Arguments

MEs	Module eigengenes of the same modules in several sets.
useAbs	Controls whether absolute value of correlation should be used instead of correlation in the calculation of dissimilarity.
useSets	If the consensus is to include only a selection of the given sets, this vector (or scalar in the case of a single set) can be used to specify the selection. If NULL, all sets will be used.
method	A character string giving the method to use. Allowed values are (abbreviations of) "consensus" and "majority". The consensus dissimilarity is calculated as the minimum of given set dissimilarities for "consensus" and as the average for "majority".

Details

This function calculates the individual set dissimilarities of the given eigengenes in each set, then takes the (parallel) maximum or average over all sets. For details on the structure of imput data, see checkSets.

Value

A dataframe containing the matrix of dissimilarities, with names and rownames set appropriately.

Author(s)

Peter Langfelder, <Peter.Langfelder@gmail.com>

See Also

checkSets

---

consensusOrderMEs *Put close eigenvectors next to each other in several sets.*

Description

Reorder given (eigen-)vectors such that similar ones (as measured by correlation) are next to each other. This is a multi-set version of orderMEs; the dissimilarity used can be of consensus type (for each pair of eigenvectors the consensus dissimilarity is the maximum of individual set dissimilarities over all sets) or of majority type (for each pair of eigenvectors the consensus dissimilarity is the average of individual set dissimilarities over all sets).

Usage

```
consensusOrderMEs(MEs, useAbs = FALSE, useSets = NULL,
                  greyLast = TRUE,
                  greyName = paste(moduleColor.getMEprefix(), "grey", sep=""),
                  method = "consensus")
```

Arguments

MEs	Module eigengenes of several sets in a multi-set format (see checkSets). A vector of lists, with each list corresponding to one dataset and the module eigengenes in the component data, that is MEs[[set]]$data[sample, module] is the expression of the eigengene of module module in sample sample in dataset set. The number of samples can be different between the sets, but the modules must be the same.
useAbs	Controls whether vector similarity should be given by absolute value of correlation or plain correlation.
useSets	Allows the user to specify for which sets the eigengene ordering is to be performed.
greyLast	Normally the color grey is reserved for unassigned genes; hence the grey module is not a proper module and it is conventional to put it last. If this is not desired, set the parameter to FALSE.
greyName	Name of the grey module eigengene.
method	A character string giving the method to be used calculating the consensus dissimilarity. Allowed values are (abbreviations of) "consensus" and "majority". The consensus dissimilarity is calculated as the maximum of given set dissimilarities for "consensus" and as the average for "majority".

Details

Ordering module eigengenes is useful for plotting purposes. This function calculates the consensus or majority dissimilarity of given eigengenes over the sets specified by useSets (defaults to all sets). A hierarchical dendrogram is calculated using the dissimilarity and the order given by the dendrogram is used for the eigengenes in all other sets.

Value

A vector of lists of the same type as MEs containing the re-ordered eigengenes.

Author(s)

Peter Langfelder, <Peter.Langfelder@gmail.com>

See Also

moduleEigengenes, multiSetMEs, orderMEs

---

consensusProjectiveKMeans

Consensus projective K-means (pre-)clustering of expression data

---

Description

Implementation of a consensus variant of K-means clustering for expression data across multiple data sets.

Usage

```
consensusProjectiveKMeans(
  multiExpr,
  preferredSize = 5000,
  nCenters = NULL,
  sizePenaltyPower = 4,
  networkType = "unsigned",
  randomSeed = 54321,
  checkData = TRUE,
  useMean = (length(multiExpr) > 3),
  maxIterations = 1000,
  verbose = 0, indent = 0)
```

Arguments

multiExpr	expression data in the multi-set format (see checkSets). A vector of lists, one per set. Each set must contain a component data that contains the expression data, with rows corresponding to samples and columns to genes or probes.
preferredSize	preferred maximum size of clusters.
nCenters	number of initial clusters. Empirical evidence suggests that more centers will give a better preclustering; the default is as.integer(min(nGenes/20, preferredSize^2/nGenes)) and is an attempt to arrive at a reasonable number given the resources available.
sizePenaltyPower	parameter specifying how severe is the penalty for clusters that exceed preferredSize.
networkType	network type. Allowed values are (unique abbreviations of) "unsigned", "signed", "signed hybrid". See adjacency.
randomSeed	integer to be used as seed for the random number generator before the function starts. If a current seed exists, it is saved and restored upon exit.
checkData	logical: should data be checked for genes with zero variance and genes and samples with excessive numbers of missing samples? Bad samples are ignored; returned cluster assignment for bad genes will be NA.
useMean	logical: should mean distance across sets be used instead of maximum? See details.
maxIterations	maximum iterations to be attempted.
verbose	integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
indent	indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

The principal aim of this function within WGCNA is to pre-cluster a large number of genes into smaller blocks that can be handled using standard WGCNA techniques.

This function implements a variant of K-means clustering that is suitable for co-expression analysis. Cluster centers are defined by the first principal component, and distances by correlation. Consensus distance across several sets is defined as the maximum of the corresponding distances in individual sets; however, if useMean is set, the mean distance will be used instead of the

maximum. The distance between a gene and a center of a cluster is multiplied by a factor of $max(clusterSize/preferredSize, 1)^{sizePenaltyPower}$, thus penalizing clusters whose size exceeds `preferredSize`. The function starts with randomly generated cluster assignment (hence the need to set the random seed for repeatability) and executes interations of calculating new centers and reassigning genes to nearest (in the consensus sense) center until the clustering becomes stable. Before returning, nearby clusters are iteratively combined if their combined size is below `preferredSize`.

Consensus distance defined as maximum of distances in all sets is consistent with the approach taken in `blockwiseConsensusModules`, but the procedure may not converge. Hence it is advisable to use the mean as consensus in cases where there are multiple data sets (4 or more, say) and/or if the input data sets are very different.

The standard principal component calculation via the function `svd` fails from time to time (likely a convergence problem of the underlying lapack functions). Such errors are trapped and the principal component is approximated by a weighted average of expression profiles in the cluster. If `verbose` is set above 2, an informational message is printed whenever this approximation is used.

Value

A list with the following components:

`clusters` a numerical vector with one component per input gene, giving the cluster number in which the gene is assigned.

`centers` a vector of lists, one list per set. Each list contains a component `data` that contains a matrix whose columns are the cluster centers in the corresponding set.

`unmergedClusters` a numerical vector with one component per input gene, giving the cluster number in which the gene was assigned before the final merging step.

`unmergedCenters` a vector of lists, one list per set. Each list contains a component `data` that contains a matrix whose columns are the cluster centers before merging in the corresponding set.

Author(s)

Peter Langfelder

See Also

`projectiveKMeans`

---

cor *Fast calculations of Pearson correlation.*

Description

These functions implements a faster calculation of Pearson correlation.

The speedup against the R's standard `cor` function will be substantial particularly if the input matrix only contains a small number of missing data. If there are no missing data, or the missing data are numerous, the speedup will be smaller but still present.

Usage

```
cor(x, y = NULL,
    use = "all.obs",
    method = c("pearson", "kendall", "spearman"),
    quick = 0, nThreads = 0,
    verbose = 0, indent = 0)

corFast(x, y = NULL,
    use = "all.obs",
    quick = 0, nThreads = 0,
    verbose = 0, indent = 0)

cor1(x, use = "all.obs", verbose = 0, indent = 0)
```

Arguments

`x`	a numeric vector or a matrix. If `y` is null, `x` must be a matrix.
`y`	a numeric vector or a matrix. If not given, correlations of columns of `x` will be calculated.
`use`	a character string specifying the handling of missing data. The fast calculations currently support `"all.obs"` and `"pairwise.complete.obs"`; for other options, see R's standard correlation function `cor`. Abbreviations are allowed.
`method`	a character string specifying the method to be used. Fast calculations are currently available only for `"pearson"`.
`quick`	real number between 0 and 1 that controls the precision of handling of missing data in the calculation of correlations. See details.
`nThreads`	non-negative integer specifying the number of parallel threads to be used by certain parts of correlation calculations. This option only has an effect on systems on which a POSIX thread library is available (which currently includes Linux and Mac OSX, but excludes Windows). If zero, the number of online processors will be used if it can be determined dynamically, otherwise correlation calculations will use 2 threads.
`verbose`	Controls the level of verbosity. Values above zero will cause a small amount of diagnostic messages to be printed.
`indent`	Indentation of printed diagnostic messages. Each unit above zero adds two spaces.

Details

The fast calculations are currently implemented only for `method="pearson"` and `use` either `"all.obs"` or `"pairwise.complete.obs"`. The `corFast` function is a wrapper that calls the function `cor`. If the combination of `method` and `use` is implemented by the fast calculations, the fast code is executed; otherwise, R's own correlation `cor` is executed.

The argument `quick` specifies the precision of handling of missing data. Zero will cause all calculations to be executed precisely, which may be significantly slower than calculations without missing data. Progressively higher values will speed up the calculations but introduce progressively larger errors. Without missing data, all column means and variances can be pre-calculated before

the covariances are calculated. When missing data are present, exact calculations require the column means and variances to be calculated for each covariance. The approximate calculation uses the pre-calculated mean and variance and simply ignores missing data in the covariance calculation. If the number of missing data is high, the pre-calculated means and variances may be very different from the actual ones, thus potentially introducing large errors. The `quick` value times the number of rows specifies the maximum difference in the number of missing entries for mean and variance calculations on the one hand and covariance on the other hand that will be tolerated before a recalculation is triggered. The hope is that if only a few missing data are treated approximately, the error introduced will be small but the potential speedup can be significant.

Value

The matrix of the Pearson correlations of the columns of `x` with columns of `y` if `y` is given, and the correlations of the columns of `x` if `y` is not given.

Note

The implementation uses the BLAS library matrix multiplication function for the most expensive step of the calculation. Using a tuned, architecture-specific BLAS may significantly improve the performance of this function.

The values returned by the corFast function may differ from the values returned by R's function `cor` by rounding errors on the order of 1e-15.

Author(s)

Peter Langfelder

See Also

R's standard Pearson correlation function `cor`.

Examples

```
## Test the speedup compared to standard function cor

# Generate a random matrix with 200 rows and 1000 columns

set.seed(10)
nrow = 200;
ncol = 1000;
data = matrix(rnorm(nrow*ncol), nrow, ncol);

## First test: no missing data

system.time( {corStd = stats::cor(data)} );

system.time( {corFast = cor(data)} );

all.equal(corStd, corFast)

# Here R's standard correlation performs very well.

# We now add a few missing entries.
```

```
data[sample(nrow, 10), 1] = NA;

# And test the correlations again...

system.time( {corStd = stats::cor(data, use ='p')} );

system.time( {corFast = cor(data, use = 'p')} );

all.equal(corStd, corFast)

# Here the R's standard correlation slows down considerably, while corFast still retains
```

`corAndPvalue` *Calculation of correlations and associated p-values*

Description

A faster, one-step calculation of Student correlation p-values for multiple correlations, properly taking into account the actual number of observations.

Usage

```
corAndPvalue(x, y,
             use = "pairwise.complete.obs",
             alternative = c("two.sided", "less", "greater"),
             ...)
```

Arguments

`x`	a vector or a matrix
`y`	a vector or a matrix. If `NULL`, the correlation of columns of `x` will be calculated.
`use`	determines handling of missing data. See `cor` for details.
`alternative`	specifies the alternative hypothesis and must be (a unique abbreviation of) one of `"two.sided"`, `"greater"` or `"less"`. the initial letter. `"greater"` corresponds to positive association, `"less"` to negative association.
`...`	other arguments to the function `cor`.

Details

The function calculates correlations of a matrix or of two matrices and the corresponding Student p-values. The output is not as full-featured as `cor.test`, but can work with matrices as input.

Value

A list with the following components

`cor`	the calculated correlations
`p`	the Student p-values corresponding to the calculated correlations

Author(s)

Peter Langfelder and Steve Horvath

See Also

`cor` for calculation of correlations only;

`cor.test` for another function for significance test of correlations

---

`corPredictionSuccess` *~~function to do ... ~~*

Description

~~ A concise (1-5 lines) description of what the function does. ~~

Usage

```
corPredictionSuccess(corPrediction, corTestSet, topNumber = 100)
```

Arguments

`corPrediction`	~~Describe `corPrediction` here~~
`corTestSet`	~~Describe `corTestSet` here~~
`topNumber`	~~Describe `topNumber` here~~

Details

~~ If necessary, more details than the description above ~~

Value

~Describe the value returned If it is a LIST, use

`comp1`	Description of 'comp1'
`comp2`	Description of 'comp2'

...

Note

~~further notes~~

Author(s)

~~who you are~~

References

~put references to the literature/web site here ~

See Also

~~objects to See Also as help, ~~~

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.
```

`corPvalueFisher` *Fisher's asymptotic p-value for correlation*

Description

Calculates Fisher's asymptotic p-value for given correlations.

Usage

```
corPvalueFisher(cor, nSamples, twoSided = TRUE)
```

Arguments

`cor`	A vector of correlation values whose corresponding p-values are to be calculated
`nSamples`	Number of samples from which the correlations were calculated
`twoSided`	logical: should the calculated p-values be two sided?

Value

A vector of p-values of the same length as the input correlations.

Author(s)

Steve Horvath and Peter Langfelder

`corPvalueStudent` *Student asymptotic p-value for correlation*

Description

Calculates Student asymptotic p-value for given correlations.

Usage

```
corPvalueStudent(cor, nSamples)
```

Arguments

`cor`	A vector of correlation values whose corresponding p-values are to be calculated
`nSamples`	Number of samples from which the correlations were calculated

Value

A vector of p-values of the same length as the input correlations.

Author(s)

Steve Horvath and Peter Langfelder

`correlationPreservation`
Preservation of eigengene correlations

Description

Calculates a summary measure of preservation of eigengene correlations across data sets

Usage

```
correlationPreservation(multiME, setLabels, excludeGrey = TRUE, greyLabel = "gre
```

Arguments

`multiME`	consensus module eigengenes in a multi-set format. A vector of lists with one list corresponding to each set. Each list must contain a component `data` that is a data frame whose columns are consensus module eigengenes.
`setLabels`	names to be used for the sets represented in `multiME`.
`excludeGrey`	logical: exclude the 'grey' eigengene from preservation measure?
`greyLabel`	module label corresponding to the 'grey' module. Usually this will be the character string `"grey"` if the labels are colors, and the number 0 if the labels are numeric.

Details

The function calculates the preservation of correlation of each eigengene with all other eigengenes (optionally except the 'grey' eigengene) in all pairs of sets.

Value

A data frame whose rows correspond to consensus module eigengenes given in the input `multiME`, and columns correspond to all possible set comparisons. The two sets compared in each column are indicated in the column name.

Author(s)

Peter Langfelder

References

Langfelder P, Horvath S (2007) Eigengene networks for studying the relationships between co-expression modules. BMC Systems Biology 2007, 1:54

See Also

multiSetMEs and modulecheckSets in package moduleColor for more on eigengenes and the multi-set format

---

cutreeStatic *Constant-height tree cut*

Description

Module detection in hierarchical dendrograms using a constant-height tree cut. Only branches whose size is at least minSize are retained.

Usage

```
cutreeStatic(dendro, cutHeight = 0.9, minSize = 50)
```

Arguments

dendro	a hierarchical clustering dendrogram such as returned by hclust.
cutHeight	height at which branches are to be cut.
minSize	minimum number of object on a branch to be considered a cluster.

Details

This function performs a straightforward constant-height cut as implemented by cutree, then calculates the number of objects on each branch and only keeps branches that have at least minSize objects on them.

Value

A numeric vector giving labels of objects, with 0 meaning unassigned. The largest cluster is conventionally labeled 1, the next largest 2, etc.

Author(s)

Peter Langfelder

See Also

hclust for hierarchical clustering, cutree and cutreeStatic for other constant-height branch cuts, standardColors to convert the retuned numerical lables into colors for easier visualization.

cutreeStaticColor *Constant height tree cut using color labels*

Description

Cluster detection by a constant height cut of a hierarchical clustering dendrogram.

Usage

```
cutreeStaticColor(dendro, cutHeight = 0.9, minSize = 50)
```

Arguments

dendro	a hierarchical clustering dendrogram such as returned by hclust.
cutHeight	height at which branches are to be cut.
minSize	minimum number of object on a branch to be considered a cluster.

Details

This function performs a straightforward constant-height cut as implemented by cutree, then calculates the number of objects on each branch and only keeps branches that have at least minSize objects on them.

Value

A character vector giving color labels of objects, with "grey" meaning unassigned. The largest cluster is conventionally labeled "turquoise", next "blue" etc. Run standardColors() to see the sequence of standard color labels.

Author(s)

Peter Langfelder

See Also

hclust for hierarchical clustering, cutree and cutreeStatic for other constant-height branch cuts, standardColors to see the sequence of color labels that can be assigned.

displayColors *Show colors used to label modules*

Description

The function plots a barplot using colors that label modules.

Usage

```
displayColors(colors = NULL)
```

Arguments

colors	colors to be displayed. Defaults to all colors available for module labeling.

Details

To see the first n colors, use argument colors = standardColors(n).

Value

None.

Author(s)

Peter Langfelder

See Also

standardColors

Examples

```
displayColors(standardColors(10))
```

---

dynamicMergeCut *Threshold for module merging*

Description

Calculate a suitable threshold for module merging based on the number of samples and a desired Z quantile.

Usage

```
dynamicMergeCut(n, mergeCor = 0.9, Zquantile = 2.35)
```

Arguments

n	number of samples
mergeCor	theoretical correlation threshold for module merging
Zquantile	Z quantile for module merging

Details

This function calculates the threshold for module merging. The threshold is calculated as the lower boundary of the interval around the theoretical correlation mergeCor whose width is given by the Z value Zquantile.

Value

The correlation threshold for module merging; a single number.

Author(s)

Steve Horvath

See Also

moduleEigengenes, mergeCloseModules

Examples

```
dynamicMergeCut(20)
dynamicMergeCut(50)
dynamicMergeCut(100)
```

---

exportNetworkToCytoscape

Export network to Cytoscape

Description

This function exports a network in edge and node list files in a format suitable for importing to Cytoscape.

Usage

```
exportNetworkToCytoscape(adjMat, edgeFile = NULL, nodeFile = NULL, weighted = TR
nodeNames = NULL, altNodeNames = NULL, nodeAttr = NULL, includeColNames = TRUE)
```

Arguments

adjMat	adjacency matrix giving connection strengths among the nodes in the network.
edgeFile	file name of the file to contain the edge information.
nodeFile	file name of the file to contain the node information.
weighted	logical: should the exported network be weighted?
threshold	adjacency threshold for including edges in the output.
nodeNames	names of the nodes. If not given, dimnames of adjMat will be used.
altNodeNames	optional alternate names for the nodes, for example gene names if nodes are labeled by probe IDs.
nodeAttr	optional node attribute, for example module color. Can be a vector or a data frame.
includeColNames	logical: should column names be included in the output files? Note that Cytoscape can read files both with and without column names.

Details

If the corresponding file names are supplied, the edge and node data is written to the appropriate files. The edge and node data is also returned as return value (see below).

Value

A list with the following componens:

egdeData	a data frame containing the edge data, with one row per edge
nodeData	a data frame containing the node data, with one row per node

Author(s)

Peter Langfelder

See Also

exportNetworkToVisANT

---

exportNetworkToVisANT *Export network data in format readable by VisANT*

Description

Exports network data in a format readable and displayable by the VisANT software.

Usage

```
exportNetworkToVisANT(
  adjMat,
  file = NULL,
  weighted = TRUE,
  threshold = 0.5,
  probeToGene = NULL)
```

Arguments

adjMat	adjacency matrix of the network to be exported.
file	character string specifying the file name of the file in which the data should be written. If not given, no file will be created. The file is in a plain text format.
weighted	logical: should the exported network by weighted?
threshold	adjacency threshold for including edges in the output.
probeToGene	optional specification of a conversion between probe names (that label columns and rows of adjacency) and gene names (that should label nodes in the output).

Details

The adjacency matrix is checked for validity. The entries can be negative, however. The adjacency matrix is expected to also have valid names or dimnames[[2]] that represent the probe names of the corresponding edges.

Whether the output is a weighted network or not, only edges whose (absolute value of) adjacency are above threshold will be included in the output.

If probeToGene is given, it is expected to have two columns, the first one corresponding to the probe names, the second to their corresponding gene names that will be used in the output.

Value

A data frame containing the network information suitable as input to VisANT. The same data frame is also written into a file specified by `file`, if given.

Author(s)

Peter Langfelder

References

VisANT software is available from http://visant.bu.edu/.

`fixDataStructure` *Put single-set data into a form useful for multiset calculations.*

Description

Encapsulates single-set data in a wrapper that makes the data suitable for functions working on multiset data collections.

Usage

```
fixDataStructure(data, verbose = 0, indent = 0)
```

Arguments

`data`	A dataframe, matrix or array with two dimensions to be encapsulated.
`verbose`	Controls verbosity. 0 is silent.
`indent`	Controls indentation of printed progress messages. 0 means no indentation, every unit adds two spaces.

Details

For multiset calculations, many quantities (such as expression data, traits, module eigengenes etc) are presented by a common structure, a vector of lists (one list for each set) where each list has a component `data` that contains the actual (expression, trait, eigengene) data for the corresponding set in the form of a dataframe. This funtion creates a vector of lists of length 1 and fills the component `data` with the content of parameter `data`.

Value

As described above, input data in a format suitable for functions operating on multiset data collections.

Author(s)

Peter Langfelder, `<Peter.Langfelder@gmail.com>`

See Also

`checkSets`

Examples

```
singleSetData = matrix(rnorm(100), 10,10);
encapsData = fixDataStructure(singleSetData);
length(encapsData)
names(encapsData[[1]])
dim(encapsData[[1]]$data)
all.equal(encapsData[[1]]$data, singleSetData);
```

---

fundamentalNetworkConcepts

Calculation of fundamental network concepts from an adjacency matrix.

Description

This function computes fundamental network concepts (also known as network indices or statistics) based on an adjacency matrix and optionally a node significance measure. These network concepts are defined for any symmetric adjacency matrix (weighted and unweighted). The network concepts are described in Dong and Horvath (2007) and Horvath and Dong (2008). Fundamental network concepts are defined as a function of the off-diagonal elements of an adjacency matrix `adj` and/or a node significance measure `GS`.

Usage

```
fundamentalNetworkConcepts(adj, GS = NULL)
```

Arguments

`adj`	an adjacency matrix, that is a square, symmetric matrix with entries between 0 and 1
`GS`	a node significance measure: a vector of the same length as the number of rows (and columns) of the adjacency matrix.

Value

A list with the following components:

`Connectivity`	a numerical vector that reports the connectivity (also known as degree) of each node. This fundamental network concept is also known as whole network connectivity. One can also define the scaled connectivity `K=Connectivity/max(Connectivity)` which is used for computing the hub gene significance.
`ScaledConnectivity`	the `Connectivity` vector scaled by the highest connectivity in the network, i.e., `Connectivity/max(Connectivity)`.
`ClusterCoef`	a numerical vector that reports the cluster coefficient for each node. This fundamental network concept measures the cliquishness of each node.

`MAR` a numerical vector that reports the maximum adjacency ratio for each node. `MAR[i]` equals 1 if all non-zero adjacencies between node `i` and the remaining network nodes equal 1. This fundamental network concept is always 1 for nodes of an unweighted network. This is a useful measure for weighted networks since it allows one to determine whether a node has high connectivity because of many weak connections (small MAR) or because of strong (but few) connections (high MAR), see Horvath and Dong 2008.

`Density` the density of the network.

`Centralization` the centralization of the network.

`Heterogeneity` the heterogeneity of the network.

Author(s)

Steve Horvath

References

Dong J, Horvath S (2007) Understanding Network Concepts in Modules, BMC Systems Biology 2007, 1:24

Horvath S, Dong J (2008) Geometric Interpretation of Gene Coexpression Network Analysis. PLoS Comput Biol 4(8): e1000117

See Also

`conformityBasedNetworkConcepts` for calculation of conformity based network concepts for a network adjacency matrix;

`networkConcepts`, for calculation of conformity based and eigennode based network concepts for a correlation network.

`GOenrichmentAnalysis` *Calculation of GO enrichment (experimental)*

Description

WARNING: This function should be considered experimental. The arguments and resulting values (in particular, the enrichment p-values) are not yet finalized and may change in the future. The function should only be used to get a quick and rough overview of GO enrichment in the modules in a data set; for a publication-quality analysis, please use an established tool.

Using Bioconductor's annotation packages, this function calculates enrichments and returns terms with best enrichment values.

Usage

```
GOenrichmentAnalysis(labels,
                     entrezCodes,
                     yeastORFs = NULL,
                     organism = "human",
                     ontologies = c("BP", "CC", "MF"),
                     evidence = c("IMP", "IGI", "IPI", "ISS", "IDA", "IEA", "TAS
                     includeOffspring = TRUE,
                     backgroundType = "givenInGO",
                     removeDuplicates = TRUE,
                     leaveOutLabel = NULL,
                     nBestP = 10, pCut = NULL,
                     nBiggest = 0,
                     verbose = 2, indent = 0)
```

Arguments

`labels`	cluster (module, group) labels of genes to be analyzed. Either a single vector, or a matrix. In the matrix case, each column will be analyzed separately; analyzing a collection of module assignments in one function call will be faster than calling the function several tinmes. For each row, the labels in all columns must correspond to the same gene specified in `entrezCodes`.
`entrezCodes`	Entrez (a.k.a. LocusLink) codes of the genes whose labels are given in `labels`. A single vector; the i-th entry corresponds to row i of the matrix `labels` (or to the i-the entry if `labels` is a vector).
`yeastORFs`	if `organism=="yeast"` (below), this argument can be used to input yeast open reading frame (ORF) identifiers instead of Entrez codes. Since the GO mappings for yeast are provided in terms of ORF identifiers, this may lead to a more accurate GO enrichment analysis. If given, the argument `entrezCodes` is ignored.
`organism`	character string specifying the organism for which to perform the analysis. Recognized values are (unique abbreviations of) `"human", "mouse", "rat", "malaria", "yeast", "fly", "bovine", "worm", "canine", "zebrafish", "chicken"`.
`ontologies`	vector of character strings specifying GO ontologies to be included in the analysis. Can be any subset of `"BP", "CC", "MF"`. The result will contain the terms with highest enrichment in each specified category, plus a separate list of terms with best enrichment in all ontologies combined.
`evidence`	vector of character strings specifying admissible evidence for each gene in its specific term. GO uses the following codes: IMP: inferred from mutant phenotype; IGI: inferred from genetic interaction; IPI: inferred from physical interaction; ISS: inferred from sequence similarity; IDA: inferred from direct assay; IEP: inferred from expression pattern; IEA: inferred from electronic annotation; TAS: traceable author statement; NAS: non-traceable author statement; ND: no biological data available; IC: inferred by curator. The default is to use all evidence types.
`includeOffspring`	logical: should genes belonging to the offspring of each term be included in the term? As a default, only genes belonging directly to each term are associated with the term. Note that the calculation of enrichments with offspring included can be quite slow for large data sets.

`backgroundType`
: specification of the background to use. Recognized values are (unique abbreviations of) `"allGiven", "allInGO", "givenInGO"`, meaning that the functions will take all genes given in `labels` as backround (`"allGiven"`), all genes present in any of the GO categories (`"allInGO"`), or the intersection of given genes and genes present in GO (`"givenInGO"`). The default is recommended for genome-wide enrichment studies.

`removeDuplicates`
: logical: should duplicate entries in `entrezCodes` be removed? If `TRUE`, only the first occurence of each unique Entrez code will be kept. The cluster labels `labels` will be adjusted accordingly.

`leaveOutLabel`
: optional specifications of module labels for which enrichment calculation is not desired. Can be a single label or a vector of labels to be ignored. However, if in any of the sets no labels are left to calculate enrichment of, the function will stop with an error.

`nBestP`
: specifies the number of terms with highest enrichment whose detailed information will be returned.

`pCut`
: alternative specification of terms to be returned: all terms whose enrichment p-value is more significant than `pCut` will be returned. If `pCut` is given, `nBestP` is ignored.

`nBiggest`
: in addition to returning terms with highest enrichment, terms that contain most of the genes in each cluster can be returned by specifying the number of biggest terms per cluster to be returned. This may be useful for development and testing purposes.

`verbose`
: integer specifying the verbosity of the function. Zero means silent, positive values will cause the function to print progress reports.

`indent`
: integer specifying indentation of the diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

This function is basically a wrapper for the annotation packages available from Bioconductor. It requires the packages GO.db, AnnotationDbi, and org.xx.eg.db, where xx is the code corresponding to the organism that the user wishes to analyze (e.g., Hs for human Homo Sapiens, Mm for mouse Mus Musculus etc). For each cluster specified in the input, the function calculates all enrichments in the specified ontologies, and collects information about the terms with highest enrichment. The enrichment p-value is calculated using Fisher exact test. As background we use all of the supplied genes that are present in at least one term in GO (in any of the ontologies).

For best results, the newest annotation libraries should be used. Because of the way Bioconductor is set up, to get the newest annotation libraries you may have to use the current version of R.

Value

A list with the following components:

`keptForAnalysis`
: logical vector with one entry per given gene. `TRUE` if the entry was used for enrichment analysis. Depending on the setting of `removeDuplicates` above, only a single entry per gene may be used.

`inGO` logical vector with one entry per given gene. `TRUE` if the gene belongs to any GO term, `FALSE` otherwise. Also `FALSE` for genes not used for the analysis because of duplication.

If input `labels` contained only one vector of labels, the following components:

`countsInTerms` a matrix whose rows correspond to given cluster, and whose columns correspond to GO terms, contaning number of genes in the intersection of the corresponding module and GO term. Row and column names are set appropriately.

`enrichmentP` a matrix whose rows correspond to given cluster, and whose columns correspond to GO terms, contaning enrichment p-values of each term in each cluster. Row and column names are set appropriately.

`bestPTerms` a list of lists with each inner list corresponding to an ontology given in `ontologies` in input, plus one component corresponding to all given ontologies combined. The name of each component is set appropriately. Each inner list contains two components: `enrichment` is a data frame containing the highest enriched terms for each module; and `forModule` is a list of lists with one inner list per module, appropriately named. Each inner list contains one component per term. This component is yet another list and contains components `termName` (term name), `enrichmentP` (enrichment P value), `termDefinition` (GO term definition), `termOntology` (GO term ontology), `geneCodes` (Entrez codes of module genes in this term), `genePositions` (indices of the genes listed in `geneCodes` within the given `labels`). Thus, to obtain information on say the second term of the 5th module in ontology BP, one can look at the appropriate row of `bestPTerms$BP$enrichment`, or one can reference `bestPTerms$BP$forModule[[5]][[2]]`. The author of the function apologizes for any confusion this structure of the output may cause.

`biggestTerms` a list of the same format as `bestPTerms`, containing information about the terms with most genes in the module for each supplied ontology.

If input `labels` contained more than one vector, instead of the above components the return value contains a list named `setResults` that has one component per given set; each component is a list containing the above components for the corresponding set.

Author(s)

Peter Langfelder

See Also

Bioconductor's annotation packages such as GO.db and organism-specific annotation packages such as org.Hs.eg.db.

---

goodGenes *Filter genes with too many missing entries*

Description

This function checks data for missing entries and returns a list of genes that have non-zero variance and pass two criteria on maximum number of missing values: the fraction of missing values must be below a given threshold and the total number of missing samples must be below a given threshold.

Usage

```
goodGenes(datExpr,
          useSamples = NULL,
          useGenes = NULL,
          minFraction = 1/2,
          minNSamples = ..minNSamples,
          minNGenes = ..minNGenes,
          verbose = 1, indent = 0)
```

Arguments

datExpr	expression data. A data frame in which columns are genes and rows ar samples.
useSamples	optional specifications of which samples to use for the check. Should be a logical vector; samples whose entries are FALSE will be ignored for the missing value counts. Defaults to using all samples.
useGenes	optional specifications of genes for which to perform the check. Should be a logical vector; genes whose entries are FALSE will be ignored. Defaults to using all genes.
minFraction	minimum fraction of non-missing samples for a gene to be considered good.
minNSamples	minimum number of non-missing samples for a gene to be considered good.
minNGenes	minimum number of good genes for the data set to be considered fit for analysis. If the actual number of good genes falls below this threshold, an error will be issued.
verbose	integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
indent	indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

The constants ..minNSamples and ..minNGenes are both set to the value 4. For most data sets, the fraction of missing samples criterion will be much more stringent than the absolute number of missing samples criterion.

Value

A logical vector with one entry per gene that is TRUE if the gene is considered good and FALSE otherwise. Note that all genes excluded by useGenes are automatically assigned FALSE.

Author(s)

Peter Langfelder and Steve Horvath

See Also

goodSamples, goodSamplesGenes

goodGenesMS *Filter genes with too many missing entries across multiple sets*

Description

This function checks data for missing entries and returns a list of genes that have non-zero variance in all sets and pass two criteria on maximum number of missing values in each given set: the fraction of missing values must be below a given threshold and the total number of missing samples must be below a given threshold

Usage

```
goodGenesMS(multiExpr,
            useSamples = NULL,
            useGenes = NULL,
            minFraction = 1/2,
            minNSamples = ..minNSamples,
            minNGenes = ..minNGenes,
            verbose = 1, indent = 0)
```

Arguments

multiExpr	expression data in the multi-set format (see checkSets). A vector of lists, one per set. Each set must contain a component data that contains the expression data, with rows corresponding to samples and columns to genes or probes.
useSamples	optional specifications of which samples to use for the check. Should be a logical vector; samples whose entries are FALSE will be ignored for the missing value counts. Defaults to using all samples.
useGenes	optional specifications of genes for which to perform the check. Should be a logical vector; genes whose entries are FALSE will be ignored. Defaults to using all genes.
minFraction	minimum fraction of non-missing samples for a gene to be considered good.
minNSamples	minimum number of non-missing samples for a gene to be considered good.
minNGenes	minimum number of good genes for the data set to be considered fit for analysis. If the actual number of good genes falls below this threshold, an error will be issued.
verbose	integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
indent	indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

The constants ..minNSamples and ..minNGenes are both set to the value 4. For most data sets, the fraction of missing samples criterion will be much more stringent than the absolute number of missing samples criterion.

Value

A logical vector with one entry per gene that is TRUE if the gene is considered good and FALSE otherwise. Note that all genes excluded by useGenes are automatically assigned FALSE.

Author(s)

Peter Langfelder

See Also

goodGenes, goodSamples, goodSamplesGenes for cleaning individual sets separately;

goodSamplesMS, goodSamplesGenesMS for additional cleaning of multiple data sets together.

---

goodSamples *Filter samples with too many missing entries*

Description

This function checks data for missing entries and returns a list of samples that pass two criteria on maximum number of missing values: the fraction of missing values must be below a given threshold and the total number of missing genes must be below a given threshold.

Usage

```
goodSamples(datExpr,
      useSamples = NULL,
      useGenes = NULL,
      minFraction = 1/2,
      minNSamples = ..minNSamples,
      minNGenes = ..minNGenes,
      verbose = 1, indent = 0)
```

Arguments

datExpr	expression data. A data frame in which columns are genes and rows ar samples.
useSamples	optional specifications of which samples to use for the check. Should be a logical vector; samples whose entries are FALSE will be ignored for the missing value counts. Defaults to using all samples.
useGenes	optional specifications of genes for which to perform the check. Should be a logical vector; genes whose entries are FALSE will be ignored. Defaults to using all genes.
minFraction	minimum fraction of non-missing samples for a gene to be considered good.
minNSamples	minimum number of good samples for the data set to be considered fit for analysis. If the actual number of good samples falls below this threshold, an error will be issued.
minNGenes	minimum number of non-missing samples for a sample to be considered good.
verbose	integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
indent	indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

The constants `..minNSamples` and `..minNGenes` are both set to the value 4. For most data sets, the fraction of missing samples criterion will be much more stringent than the absolute number of missing samples criterion.

Value

A logical vector with one entry per sample that is `TRUE` if the sample is considered good and `FALSE` otherwise. Note that all samples excluded by `useSamples` are automatically assigned `FALSE`.

Author(s)

Peter Langfelder and Steve Horvath

See Also

`goodSamples`, `goodSamplesGenes`

---

goodSamplesGenes *Iterative filtering of samples and genes with too many missing entries*

Description

This function checks data for missing entries and zero-variance genes, and returns a list of samples and genes that pass criteria maximum number of missing values. If necessary, the filtering is iterated.

Usage

```
goodSamplesGenes(
  datExpr,
  minFraction = 1/2,
  minNSamples = ..minNSamples,
  minNGenes = ..minNGenes,
  verbose = 1, indent = 0)
```

Arguments

`datExpr`	expression data. A data frame in which columns are genes and rows ar samples.
`minFraction`	minimum fraction of non-missing samples for a gene to be considered good.
`minNSamples`	minimum number of non-missing samples for a gene to be considered good.
`minNGenes`	minimum number of good genes for the data set to be considered fit for analysis. If the actual number of good genes falls below this threshold, an error will be issued.
`verbose`	integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
`indent`	indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

This function iteratively identifies samples and genes with too many missing entries and genes with zero variance. Iterations may be required since excluding samples effectively changes criteria on genes and vice versa. The process is repeated until the lists of good samples and genes are stable. The constants `..minNSamples` and `..minNGenes` are both set to the value 4.

Value

A list with the foolowing components:

`goodSamples`	A logical vector with one entry per sample that is `TRUE` if the sample is considered good and `FALSE` otherwise.
`goodGenes`	A logical vector with one entry per gene that is `TRUE` if the gene is considered good and `FALSE` otherwise.

Author(s)

Peter Langfelder

See Also

`goodSamples`, `goodGenes`

---

goodSamplesGenesMS *Iterative filtering of samples and genes with too many missing entries across multiple data sets*

Description

This function checks data for missing entries and zero variance across multiple data sets and returns a list of samples and genes that pass criteria maximum number of missing values. If necessary, the filtering is iterated.

Usage

```
goodSamplesGenesMS(
  multiExpr,
  minFraction = 1/2,
  minNSamples = ..minNSamples,
  minNGenes = ..minNGenes,
  verbose = 2, indent = 0)
```

Arguments

`multiExpr`	expression data in the multi-set format (see `checkSets`). A vector of lists, one per set. Each set must contain a component `data` that contains the expression data, with rows corresponding to samples and columns to genes or probes.
`minFraction`	minimum fraction of non-missing samples for a gene to be considered good.
`minNSamples`	minimum number of non-missing samples for a gene to be considered good.

minNGenes	minimum number of good genes for the data set to be considered fit for analysis. If the actual number of good genes falls below this threshold, an error will be issued.
verbose	integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
indent	indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

This function iteratively identifies samples and genes with too many missing entries, and genes with zero variance. Iterations may be required since excluding samples effectively changes criteria on genes and vice versa. The process is repeated until the lists of good samples and genes are stable. The constants `..minNSamples` and `..minNGenes` are both set to the value 4.

Value

A list with the foolowing components:

goodSamples	A list with one component per given set. Each component is a logical vector with one entry per sample in the corresponding set that is `TRUE` if the sample is considered good and `FALSE` otherwise.
goodGenes	A logical vector with one entry per gene that is `TRUE` if the gene is considered good and `FALSE` otherwise.

Author(s)

Peter Langfelder

See Also

`goodGenes`, `goodSamples`, `goodSamplesGenes` for cleaning individual sets separately;

`goodSamplesMS`, `goodGenesMS` for additional cleaning of multiple data sets together.

---

goodSamplesMS *Filter samples with too many missing entries across multiple data sets*

Description

This function checks data for missing entries and returns a list of samples that pass two criteria on maximum number of missing values: the fraction of missing values must be below a given threshold and the total number of missing genes must be below a given threshold.

Usage

```
goodSamplesMS(multiExpr,
      useSamples = NULL,
      useGenes = NULL,
      minFraction = 1/2,
      minNSamples = ..minNSamples,
      minNGenes = ..minNGenes,
      verbose = 1, indent = 0)
```

Arguments

multiExpr	expression data in the multi-set format (see checkSets). A vector of lists, one per set. Each set must contain a component data that contains the expression data, with rows corresponding to samples and columns to genes or probes.
useSamples	optional specifications of which samples to use for the check. Should be a logical vector; samples whose entries are FALSE will be ignored for the missing value counts. Defaults to using all samples.
useGenes	optional specifications of genes for which to perform the check. Should be a logical vector; genes whose entries are FALSE will be ignored. Defaults to using all genes.
minFraction	minimum fraction of non-missing samples for a gene to be considered good.
minNSamples	minimum number of good samples for the data set to be considered fit for analysis. If the actual number of good samples falls below this threshold, an error will be issued.
minNGenes	minimum number of non-missing samples for a sample to be considered good.
verbose	integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
indent	indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

The constants ..minNSamples and ..minNGenes are both set to the value 4. For most data sets, the fraction of missing samples criterion will be much more stringent than the absolute number of missing samples criterion.

Value

A list with one component per input set. Each component is a logical vector with one entry per sample in the corresponding set, indicating whether the sample passed the missing value criteria.

Author(s)

Peter Langfelder and Steve Horvath

See Also

goodGenes, goodSamples, goodSamplesGenes for cleaning individual sets separately;

goodGenesMS, goodSamplesGenesMS for additional cleaning of multiple data sets together.

---

greenBlackRed *Green-black-red color sequence*

Description

Generate a green-black-red color sequence of a given length.

Usage

```
greenBlackRed(n, gamma = 1)
```

Arguments

n	number of colors to be returned
gamma	color correction power

Details

The function returns a color vector that starts with pure green, gradually turns into black and then to red. The power gamma can be used to control the behaviour of the quarter- and three quarter-values (between green and black, and black and red, respectively). Higher powers will make the mid-colors more green and red, respectively.

Value

A vector of colors of length n.

Author(s)

Peter Langfelder

Examples

```
par(mfrow = c(3, 1))
displayColors(greenBlackRed(50));
displayColors(greenBlackRed(50, 2));
displayColors(greenBlackRed(50, 0.5));
```

---

greenWhiteRed *Green-white-red color sequence*

Description

Generate a green-white-red color sequence of a given length.

Usage

```
greenWhiteRed(n, gamma = 1)
```

Arguments

n	number of colors to be returned
gamma	color correction power

Details

The function returns a color vector that starts with pure green, gradually turns into white and then to red. The power gamma can be used to control the behaviour of the quarter- and three quarter-values (between green and white, and white and red, respectively). Higher powers will make the mid-colors more white, while lower powers will make the colors more saturated, respectively.

Value

A vector of colors of length `n`.

Author(s)

Peter Langfelder

Examples

```
par(mfrow = c(3, 1))
displayColors(greenWhiteRed(50));
displayColors(greenWhiteRed(50, 3));
displayColors(greenWhiteRed(50, 0.5));
```

GTOMdist *Generalized Topological Overlap Measure*

Description

Generalized Topological Overlap Measure, taking into account interactions of higher degree.

Usage

```
GTOMdist(adjMat, degree = 1)
```

Arguments

`adjMat`	adjacency matrix. See details below.
`degree`	integer specifying the maximum degree to be calculated.

Value

Matrix of the same dimension as the input `adjMat`.

Author(s)

Steve Horvath and Andy Yip

References

Yip A, Horvath S (2007) Gene network interconnectedness and the generalized topological overlap measure. BMC Bioinformatics 2007, 8:22

hubGeneSignificance
Hubgene significance

Description

Calculate approximate hub gene significance for all modules in network.

Usage

```
hubGeneSignificance(datKME, GS)
```

Arguments

datKME	a data frame (or a matrix-like object) containing eigengene-based connectivities of all genes in the network.
GS	a vector with one entry for every gene containing its gene significance.

Details

In datKME rows correspond to genes and columns to modules.

Value

A vector whose entries are the hub gene significances for each module.

Author(s)

Steve Horvath

References

Dong J, Horvath S (2007) Understanding Network Concepts in Modules, BMC Systems Biology 2007, 1:24

Inline display of progress
Inline display of progress

Description

These functions provide an inline display of pregress.

Usage

```
initProgInd(leadStr = "..", trailStr = "", quiet = !interactive())
updateProgInd(newFrac, progInd, quiet = !interactive())
```

Arguments

leadStr	character string that will be printed before the actual progress number.
trailStr	character string that will be printed after the actual progress number.
quiet	can be used to silence the indicator for non-interactive sessions whose output is typically redirected to a file.
newFrac	new fraction of progress to be displayed.
progInd	an object of class progressIndicator that encodes previously printed message.

Details

A progress indicator is a simple inline display of progress intended to satisfy impatient users during lengthy operations. The function initProgInd initializes a progress indicator (at zero); updateProgInd updates it to a specified fraction.

Value

Both functions return an object of class progressIndicator that holds information on the last printed value and should be used for subsequent updates of the indicator. Note that excessive use of updateProgInd may lead to a performance penalty if a substantial amount of CPU time has to be invested into console output. See examples.

Author(s)

Peter Langfelder

Examples

```
if (TRUE)
{
  max = 20;
  prog = initProgInd("Counting: ", "done");
  for (c in 1:max)
  {
    Sys.sleep(0.3);
    prog = updateProgInd(c/max, prog);
  }
  printFlush("");
}

if (TRUE)
{
  max = 20;
  printFlush("Example 2:");
  prog = initProgInd();
  for (c in 1:max)
  {
    Sys.sleep(0.3);
    prog = updateProgInd(c/max, prog);
  }
  printFlush("");
}

## Example of a significant slowdown:
```

```
## Without progress indicator:

system.time( {a = 0; for (i in 1:100000) a = a+i; } )

## With progress indicator, some 100 times slower:

system.time(
{
  prog = initProgInd("Counting: ", "done");
  a = 0;
  for (i in 1:100000)
  {
    a = a+i;
    prog = updateProgInd(i/100000, prog);
  }
}
)
```

`intramodularConnectivity`
Calculation of intramodular connectivity

Description

Calculates intramodular connectivity, i.e., connectivity of nodes to other nodes within the same module.

Usage

```
intramodularConnectivity(adjMat, colors, scaleByMax = FALSE)
```

Arguments

`adjMat`	adjacency matrix, a square, symmetric matrix with entries between 0 and 1.
`colors`	module labels. A vector of length `ncol(adjMat)` giving a module label for each gene (node) of the network.
`scaleByMax`	logical: should intramodular connectivities be scaled by the maximum IM connectivity in each module?

Details

The module labels can be numeric or character. For each node (gene), the function sums adjacency entries (excluding the diagonal) to other nodes within the same module. Optionally, the connectivities can be scaled by the maximum connectivy in each module.

Value

A data frame with 4 columns giving the total connectivity, intramodular connectivity, extra-modular connectivity, and the difference of the intra- and extra-modular connectivities for all genes.

Author(s)

Steve Horvath and Peter Langfelder

References

Dong J, Horvath S (2007) Understanding Network Concepts in Modules, BMC Systems Biology 2007, 1:24

See Also

adjacency

---

keepCommonProbes *Keep probes that are shared among given data sets*

Description

This function strips out probes that are not shared by all given data sets, and orders the remaining common probes using the same order in all sets.

Usage

```
keepCommonProbes(multiExpr, orderBy = 1)
```

Arguments

multiExpr	expression data in the multi-set format (see checkSets). A vector of lists, one per set. Each set must contain a component data that contains the expression data, with rows corresponding to samples and columns to genes or probes.
orderBy	index of the set by which probes are to be ordered.

Value

Expression data in the same format as the input data, containing only common probes.

Author(s)

Peter Langfelder

See Also

checkSets

`labeledBarplot` *Barplot with text or color labels.*

Description

Produce a barplot with extra annotation.

Usage

```
labeledBarplot(
      Matrix, labels,
      colorLabels = FALSE,
      colored = TRUE,
      setStdMargins = TRUE,
      stdErrors = NULL,
      cex.lab = NULL,
      xLabelsAngle = 45,
      ...)
```

Arguments

`Matrix`	vector or a matrix to be plotted.
`labels`	labels to annotate the bars underneath the barplot.
`colorLabels`	logical: should the labels be interpreted as colors? If `TRUE`, the bars will be labeled by colored squares instead of text. See details.
`colored`	logical: should the bars be divided into segments and colored? If `TRUE`, assumes the `labels` can be interpreted as colors, and the input `Matrix` is square and the rows have the same labels as the columns. See details.
`setStdMargins`	if `TRUE`, the function wil set margins `c(3, 3, 2, 2)+0.2`.
`stdErrors`	if given, error bars corresponding to `1.96*stdErrors` will be plotted on top of the bars.
`cex.lab`	character expansion factor for axis labels, including the text labels underneath the barplot.
`xLabelsAngle`	angle at which text labels under the barplot will be printed.
`...`	other parameters for the function `barplot`.

Details

Individual bars in the barplot can be identified either by printing the text of the corresponding entry in `labels` underneath the bar at the angle specified by `xLabelsAngle`, or by interpreting the `labels` entry as a color (see below) and drawing a correspondingly colored square underneath the bar.

For reasons of compatibility with other functions, `labels` are interpreted as colors after stripping the first two characters from each label. For example, the label `"MEturquoise"` is interpreted as the color turquoise.

If `colored` is set, the code assumes that `labels` can be interpreted as colors, and the input `Matrix` is square and the rows have the same labels as the columns. Each bar in the barplot is then sectioned into contributions from each row entry in `Matrix` and is colored by the color given by the entry in `labels` that corresponds to the row.

Value

None.

Author(s)

Peter Langfelder

`labeledHeatmap` *Produce a labeled heatmap plot*

Description

Plots a heatmap plot with color legend, row and column annotation, and optional text within th heatmap.

Usage

```
labeledHeatmap(
  Matrix,
  xLabels, yLabels = NULL,
  xSymbols = NULL, ySymbols = NULL,
  colorLabels = NULL,
  xColorLabels = FALSE, yColorLabels = FALSE,
  checkColorsValid = TRUE,
  invertColors = FALSE,
  setStdMargins = TRUE,
  xLabelsPosition = "bottom",
  xLabelsAngle = 45,
  xLabelsAdj = 1,
  xColorWidth = 0.05,
  yColorWidth = 0.05,
  colors = NULL,
  textMatrix = NULL,
  cex.text = NULL, cex.lab = NULL,
  plotLegend = TRUE, ...)
```

Arguments

`Matrix`	numerical matrix to be plotted in the heatmap.
`xLabels`	labels for the columns. See Details.
`yLabels`	labels for the rows. See Details.
`xSymbols`	additional labels used when `xLabels` are interpreted as colors. See Details.
`ySymbols`	additional labels used when `yLabels` are interpreted as colors. See Details.
`colorLabels`	logical: should `xLabels` and `yLabels` be interpreted as colors? If given, overrides `xColorLabels` and `yColorLabels` below.
`xColorLabels`	logical: should `xLabels` be interpreted as colors?
`yColorLabels`	logical: should `yLabels` be interpreted as colors?

checkColorsValid
: logical: should given colors be checked for validity against the output of `colors()` ? If this argument is `FALSE`, invalid color specification will trigger an error.

`invertColors` logical: should the color order be inverted?

setStdMargins
: logical: should standard margins be set before calling the plot function? Standard margins depend on `colorLabels`: they are wider for text labels and narrower for color labels. The defaults are static, that is the function does not attempt to guess the optimal margins.

xLabelsPosition
: a character string specifying the position of labels for the columns. Recognized values are (unique abbreviations of) `"top", "bottom"`.

`xLabelsAngle` angle by which the column labels should be rotated.

`xLabelsAdj` justification parameter for column labels. See `par` and the description of parameter `"adj"`.

`xColorWidth` width of the color labels for the x axis expressed as a fraction of the smaller of the range of the x and y axis.

`yColorWidth` width of the color labels for the y axis expressed as a fraction of the smaller of the range of the x and y axis.

`colors` color pallette to be used in the heatmap. Defaults to `heat.colors`.

`textMatrix` optional matrix of text entries of the same dimensions as `Matrix`.

`cex.text` character expansion factor for `textMatrix`.

`cex.lab` character expansion factor for text labels labeling the axes

`plotLegend` logical: should a color legend be plotted?

`...` other arguments to functions `image.plot` (for `plotLegend==TRUE`) or `heatmap` (for `plotLegend==FALSE`).

Details

The function basically plots a standard heatmap plot of the given `Matrix` and embellishes it with row and column labels and/or with text within the heatmap entries. Row and column labels can be either character strings or color squares, or both.

To get simple text labels, use `colorLabels=FALSE` and pass the desired row and column labels in `yLabels` and `xLabels`, respectively.

To label rows and columns by color squares, use `colorLabels=TRUE`; `yLabels` and `xLabels` are then expected to represent valid colors. For reasons of compatibility with other functions, each entry in `yLabels` and `xLabels` is expected to consist of a color designation preceded by 2 characters: an example would be `MEturquoise`. The first two characters can be arbitrary, they are stripped. Any labels that do not represent valid colors will be considered text labels and printed in full, allowing the user to mix text and color labels.

It is also possible to label rows and columns by both color squares and additional text annotation. To achieve this, use the above technique to get color labels and, additionally, pass the desired text annotation in the `xSymbols` and `ySymbols` arguments.

Value

None.

Author(s)

Peter Langfelder

See Also

image.plot, heatmap, colors

Examples

```
# This example illustrates 4 main ways of annotating columns and rows of a heatmap.
# Copy and paste the whole example into an R session with an interactive plot window;
# alternatively, you may replace the command sizeGrWindow below by opening another graphi
# as pdf.

# Generate a matrix to be plotted

nCol = 8; nRow = 7;
mat = matrix(runif(nCol*nRow, min = -1, max = 1), nRow, nCol);

rowColors = standardColors(nRow);
colColors = standardColors(nRow + nCol)[(nRow+1):(nRow + nCol)];

rowColors;
colColors;

sizeGrWindow(9,7)
par(mfrow = c(2,2))
par(mar = c(4, 5, 4, 6));

# Label rows and columns by text:

labeledHeatmap(mat, xLabels = colColors, yLabels = rowColors,
               colors = greenWhiteRed(50),
               setStdMargins = FALSE,
               textMatrix = signif(mat, 2),
               main = "Text-labeled heatmap");

# Label rows and columns by colors:

rowLabels = paste("ME", rowColors, sep="");
colLabels = paste("ME", colColors, sep="");

labeledHeatmap(mat, xLabels = colLabels, yLabels = rowLabels,
               colorLabels = TRUE,
               colors = greenWhiteRed(50),
               setStdMargins = FALSE,
               textMatrix = signif(mat, 2),
               main = "Color-labeled heatmap");

# Mix text and color labels:

rowLabels[3] = "Row 3";
colLabels[1] = "Column 1";

labeledHeatmap(mat, xLabels = colLabels, yLabels = rowLabels,
```

```
                colorLabels = TRUE,
                colors = greenWhiteRed(50),
                setStdMargins = FALSE,
                textMatrix = signif(mat, 2),
                main = "Mix-labeled heatmap");

# Color labels and additional text labels

rowLabels = paste("ME", rowColors, sep="");
colLabels = paste("ME", colColors, sep="");

extraRowLabels = paste("Row", c(1:nRow));
extraColLabels = paste("Column", c(1:nCol));

# Extend margins to fit all labels
par(mar = c(6, 6, 4, 6));
labeledHeatmap(mat, xLabels = colLabels, yLabels = rowLabels,
                xSymbols = extraColLabels,
                ySymbols = extraRowLabels,
                colorLabels = TRUE,
                colors = greenWhiteRed(50),
                setStdMargins = FALSE,
                textMatrix = signif(mat, 2),
                main = "Text- + color-labeled heatmap");
```

`labelPoints` *Label scatterplot points*

Description

Given scatterplot point coordinates, the function tries to place labels near the points such that the labels do not become scrambled.

Usage

```
labelPoints(x, y, labels, cex = 0.7, offs = 0.01, xpd = TRUE, jiggle = 0, ...)
```

Arguments

`x`	a vector of x coordinates of the points
`y`	a vector of y coordinates of the points
`labels`	labels to be placed next to the points
`cex`	character expansion factor for the labels
`offs`	offset of the labels from the plotted coordinates in inches
`xpd`	logical: controls truncating labels to fit within the plotting region. See `par`.
`jiggle`	amount of random noise to be added to the coordinates. This may be useful if the scatterplot is too regular (such as all points on one straight line).
`...`	other arguments to function `text`.

Details

The algorithm basically works by finding the direction of most surrounding points, and attempting to place the label in the opposite direction. There are (not uncommon) situations in which this placement is suboptimal; the author promises to further develop the function sometime in the future.

Note that this function does not plot the actual scatterplot; only the labels are plotted. Plotting the scatterplot is the responsibility of the user.

The argument offset needs to be carefully tuned to the size of the plotted symbols. Sorry, no automation here yet.

Value

None.

Author(s)

Peter Langfelder

See Also

plot.default, text

Examples

```
# generate some random points
   set.seed(11);
   n = 20;
   x = runif(n);
   y = runif(n);

# Create a basic scatterplot
   col = standardColors(n);
   plot(x,y, pch = 21, col =1, bg = col, cex = 2.6, xlim = c(-0.1, 1.1), ylim = c(-0.1, 1
   labelPoints(x, y, paste("Pt", c(1:n), sep=""), offs = 0.10, cex = 1);

# label points using longer text labels. Note the positioning is not perfect, but close e

   plot(x,y, pch = 21, col =1, bg = col, cex = 2.6, xlim = c(-0.1, 1.1), ylim = c(-0.1, 1
   labelPoints(x, y, col, offs = 0.10, cex = 0.8);
```

---

labels2colors *Convert numerical labels to colors.*

Description

Converts a vector or array of numerical labels into a corresponding vector or array of colors corresponding to the labels.

Usage

```
labels2colors(labels, zeroIsGrey = TRUE, colorSeq = NULL)
```

Arguments

labels	Vector of non-negative integer labels.
zeroIsGrey	If TRUE, labels 0 will be assigned color grey. Otherwise, labels below 1 will trigger an error.
colorSeq	Color sequence corresponding to labels. If not given, a standard sequence will be used.

Details

The standard sequence start with well-distinguishable colors, and after about 40 turns into a quasi-random sampling of all colors available in R with the exception of all shades of grey (and gray).

If the input labels have a dimension attribute, it is copied into the output, meaning the dimensions of the returned value are the same as those of the input labels.

Value

A vector or array of character strings of the same length or dimensions as labels.

Author(s)

Peter Langfelder, <Peter.Langfelder@gmail.com>

Examples

```
labels = c(0:20);
labels2colors(labels);
```

---

matchLabels *Relabel module labels to best match the given reference labels*

Description

Given a source and reference vectors of module labels, the function produces a module labeling that is equivalent to source, but individual modules are re-labeled so that modules with significant overlap in source and reference have the same labels.

Usage

```
matchLabels(source, reference, pThreshold = 5e-2)
```

Arguments

source	a vector or a matrix of reference labels. The labels may be numeric or character.
reference	a vector of reference labels.
pThreshold	threshold of Fisher's exact test for considering modules to have a significant overlap.

Details

Each column of `source` is treated separately. Source and reference labels are assumed to be of the same type, that is noth should be either numeric or character. If the labels are character, they are assumed to be color labels such as the ones returned by `standardColors`.

The function calculates the overlap of the `source` and `reference` modules using Fisher's exact test. It then attempts to relabel `source` modules such that modules with the highest overlap with the `reference` modules have the same color. Where this is not possible (for example because one reference module has the highest overlap with two source modules), the source modules will be relabeled using labels that are not present among the reference labels.

Value

A vector (if the input `source` labels are a vector) or a matrix (if the input `source` labels are a matrix) of the new labels.

Author(s)

Peter Langfelder

See Also

`standardColors`

`mergeCloseModules` *Merge close modules in gene expression data*

Description

Merges modules in gene expression networks that are too close as measured by the correlation of their eigengenes.

Usage

```
mergeCloseModules(exprData, colors,
                  cutHeight = 0.2,
                  MEs = NULL,
                  impute = TRUE,
                  useAbs = FALSE,
                  iterate = TRUE,
                  relabel = FALSE,
                  colorSeq = NULL,
                  getNewMEs = TRUE,
                  getNewUnassdME = TRUE,
                  useSets = NULL,
                  checkDataFormat = TRUE,
                  unassdColor = ifelse(is.numeric(colors), 0, "grey"),
                  trapErrors = FALSE,
                  verbose = 1, indent = 0)
```

Arguments

exprData	Expression data, either a single data frame with rows corresponding to samples and columns to genes, or in a multi-set format (see checkSets). See checkDataStructure below.
colors	A vector (numeric, character or a factor) giving module colors for genes. The method only makes sense when genes have the same color label in all sets, hence a single vector.
cutHeight	Maximum dissimilarity (i.e., 1-correlation) that qualifies modules for merging.
MEs	If module eigengenes have been calculated before, the user can save some computational time by inputting them. MEs should have the same format as exprData. If they are not given, they will be calculated.
impute	Should missing values be imputed in eigengene calculation? If imputation is disabled, the presence of NA entries will cause the eigengene calculation to fail and eigengenes will be replaced by their hubgene approximation. See moduleEigengenes for more details.
useAbs	Specifies whether absolute value of correlation or plain correlation (of module eigengenes) should be used in calculating module dissimilarity.
iterate	Controls whether the merging procedure should be repeated until there is no change. If FALSE, only one iteration will be executed.
relabel	Controls whether, after merging, color labels should be ordered by module size.
colorSeq	Color labels to be used for relabeling. Defaults to the standard color order used in this package if colors are not numeric, and to integers starting from 1 if colors is numeric.
getNewMEs	Controls whether module eigengenes of merged modules should be calculated and returned.
getNewUnassdME	When doing module eigengene manipulations, the function does not normally calculate the eigengene of the 'module' of unassigned ('grey') genes. Setting this option to TRUE will force the calculation of the unassigned eigengene in the returned newMEs, but not in the returned oldMEs.
useSets	A vector of scalar allowing the user to specify which sets will be used to calculate the consensus dissimilarity of module eigengenes. Defaults to all given sets.
checkDataFormat	If TRUE, the function will check exprData and MEs for correct multi-set structure. If single set data is given, it will be converted into a format usable for the function. If FALSE, incorrect structure of input data will trigger an error.
unassdColor	Specifies the string that labels unassigned genes. Module of this color will not enter the module eigengene clustering and will not be merged with other modules.
trapErrors	Controls whether computational errors in calculating module eigengenes, their dissimilarity, and merging trees should be trapped. If TRUE, errors will be trapped and the function will return the input colors. If FALSE, errors will cause the function to stop.
verbose	Controls verbosity of printed progress messages. 0 means silent, up to (about) 5 the verbosity gradually increases.
indent	A single non-negative integer controlling indentation of printed messages. 0 means no indentation, each unit above that adds two spaces.

Details

This function returns the color labels for modules that are obtained from the input modules by merging ones that are closely related. The relationships are quantified by correlations of module eigengenes; a "consensus" measure is defined as the minimum over the corresponding relationship in each set. Once the (dis-)similarity is calculated, average linkage hierarchical clustering of the module eigengenes is performed, the dendrogram is cut at the height `cutHeight` and modules on each branch are merged. The process is (optionally) repeated until no more modules are merged.

If, for a particular module, the module eigengene calculation fails, a hubgene approximation will be used.

The user should be aware that if a computational error occurs and `trapErrors==TRUE`, the returned list (see below) will not contain all of the components returned upon normal execution.

Value

If no errors occurred, a list with components

`colors`	Color labels for the genes corresponding to merged modules. The function attempts to mimic the mode of the input `colors`: if the input `colors` is numeric, character and factor, respectively, so is the output. Note, however, that if the fnction performs relabeling, a standard sequence of labels will be used: integers starting at 1 if the input `colors` is numeric, and a sequence of color labels otherwise (see `colorSeq` above).
`dendro`	Hierarchical clustering dendrogram (average linkage) of the eigengenes of the most recently computed tree. If `iterate` was set TRUE, this will be the dendrogram of the merged modules, otherwise it will be the dendrogram of the original modules.
`oldDendro`	Hierarchical clustering dendrogram (average linkage) of the eigengenes of the original modules.
`cutHeight`	The input cutHeight.
`oldMEs`	Module eigengenes of the original modules in the sets given by `useSets`.
`newMEs`	Module eigengenes of the merged modules in the sets given by `useSets`.
`allOK`	A boolean set to `TRUE`.

If an error occurred and `trapErrors==TRUE`, the list only contains these components:

`colors`	A copy of the input colors.
`allOK`	a boolean set to `FALSE`.

Author(s)

Peter Langfelder, `<Peter.Langfelder@gmail.com>`

`moduleColor.getMEprefix` *Get the prefix used to label module eigengenes.*

Description

Returns the currently used prefix used to label module eigengenes. When returning module eigengenes in a dataframe, names of the corresponding columns will start with the given prefix.

Usage

```
moduleColor.getMEprefix()
```

Details

Returns the prefix used to label module eigengenes. When returning module eigengenes in a dataframe, names of the corresponding columns will consist of the corresponfing color label preceded by the given prefix. For example, if the prefix is "PC" and the module is turquoise, the corresponding module eigengene will be labeled "PCturquoise". Most of old code assumes "PC", but "ME" is more instructive and used in some newer analyses.

Value

A character string.

Note

Currently the standard prefix is `"ME"` and there is no way to change it.

Author(s)

Peter Langfelder, `<Peter.Langfelder@gmail.com>`

See Also

`moduleEigengenes`

`moduleEigengenes` *Calculate module eigengenes.*

Description

Calculates module eigengenes (1st principal component) of modules in a given single dataset.

Usage

```
moduleEigengenes(expr,
                 colors,
                 impute = TRUE,
                 nPC = 1,
                 align = "along average",
                 excludeGrey = FALSE,
                 grey = ifelse(is.numeric(colors),  0, "grey"),
                 subHubs = TRUE,
                 trapErrors = FALSE,
                 returnValidOnly = trapErrors,
                 softPower = 6,
                 scale = TRUE,
                 verbose = 0, indent = 0)
```

Arguments

`expr`	Expression data for a single set in the form of a data frame where rows are samples and columns are genes (probes).
`colors`	A vector of the same length as the number of probes in `expr`, giving module color for all probes (genes). Color `"grey"` is reserved for unassigned genes.
`impute`	If `TRUE`, expression data will be checked for the presence of `NA` entries and if the latter are present, numerical data will be imputed, using function `impute.knn` and probes from the same module as the missing datum. The function `impute.knn` uses a fixed random seed giving repeatable results.
`nPC`	Number of principal components and variance explained entries to be calculated. Note that only the first principal component is returned; the rest are used only for the calculation of proportion of variance explained. The number of returned variance explained entries is currently `min(nPC, 10)`. If given `nPC` is greater than 10, a warning is issued.
`align`	Controls whether eigengenes, whose orientation is undetermined, should be aligned with average expression (`align = "along average"`, the default) or left as they are (`align = ""`). Any other value will trigger an error.
`excludeGrey`	Should the improper module consisting of 'grey' genes be excluded from the eigengenes?
`grey`	Value of `colors` designating the improper module. Note that if `colors` is a factor of numbers, the default value will be incorrect.
`subHubs`	Controls whether hub genes should be substituted for missing eigengenes. If `TRUE`, each missing eigengene (i.e., eigengene whose calculation failed and the error was trapped) will be replaced by a weighted average of the most connected hub genes in the corresponding module. If this calculation fails, or if `subHubs==FALSE`, the value of `trapErrors` will determine whether the offending module will be removed or whether the function will issue an error and stop.
`trapErrors`	Controls handling of errors from that may arise when there are too many `NA` entries in expression data. If `TRUE`, errors from calling these functions will be trapped without abnormal exit. If `FALSE`, errors will cause the function to stop. Note, however, that `subHubs` takes precedence in the sense that if `subHubs==TRUE` and `trapErrors==FALSE`, an error will be issued only if both the principal component and the hubgene calculations have failed.

returnValidOnly
: logical; controls whether the returned data frame of module eigengenes contains columns corresponding only to modules whose eigengenes or hub genes could be calculated correctly (TRUE), or whether the data frame should have columns for each of the input color labels (FALSE).

softPower
: The power used in soft-thresholding the adjacency matrix. Only used when the hubgene approximation is necessary because the principal component calculation failed. It must be non-negative. The default value should only be changed if there is a clear indication that it leads to incorrect results.

scale
: logical; can be used to turn off scaling of the expression data before calculating the singular value decomposition. The scaling should only be turned off if the data has been scaled previously, in which case the function can run a bit faster. Note however that the function first imputes, then scales the expression data in each module. If the expression contain missing data, scaling outside of the function and letting the function impute missing data may lead to slightly different results than if the data is scaled within the function.

verbose
: Controls verbosity of printed progress messages. 0 means silent, up to (about) 5 the verbosity gradually increases.

indent
: A single non-negative integer controlling indentation of printed messages. 0 means no indentation, each unit above that adds two spaces.

Details

Module eigengene is defined as the first principal component of the expression matrix of the corresponding module. The calculation may fail if the expression data has too many missing entries. Handling of such errors is controlled by the arguments subHubs and trapErrors. If subHubs==TRUE, errors in principal component calculation will be trapped and a substitute calculation of hubgenes will be attempted. If this fails as well, behaviour depends on trapErrors: if TRUE, the offending module will be ignored and the return value will allow the user to remove the module from further analysis; if FALSE, the function will stop.

From the user's point of view, setting trapErrors=FALSE ensures that if the function returns normally, there will be a valid eigengene (principal component or hubgene) for each of the input colors. If the user sets trapErrors=TRUE, all calculational (but not input) errors will be trapped, but the user should check the output (see below) to make sure all modules have a valid returned eigengene.

While the principal component calculation can fail even on relatively sound data (it does not take all that many "well-placed" NA to torpedo the calculation), it takes many more irregularities in the data for the hubgene calculation to fail. In fact such a failure signals there likely is something seriously wrong with the data.

Value

A list with the following components:

eigengenes
: Module eigengenes in a dataframe, with each column corresponding to one eigengene. The columns are named by the corresponding color with an "ME" prepended, e.g., MEturquoise etc. If returnValidOnly==FALSE, module eigengenes whose calculation failed have all components set to NA.

averageExpr
: If align == "along average", a dataframe containing average normalized expression in each module. The columns are named by the corresponding color with an "AE" prepended, e.g., AEturquoise etc.

varExplained A dataframe in which each column corresponds to a module, with the component varExplained[PC, module] giving the variance of module module explained by the principal component no. PC. The calculation is exact irrespective of the number of computed principal components. At most 10 variance explained values are recorded in this dataframe.

nPC A copy of the input nPC.

validMEs A boolean vector. Each component (corresponding to the columns in data) is TRUE if the corresponding eigengene is valid, and FALSE if it is invalid. Valid eigengenes include both principal components and their hubgene approximations. When returnValidOnly==FALSE, by definition all returned eigengenes are valid and the entries of validMEs are all TRUE.

validColors A copy of the input colors with entries corresponding to invalid modules set to grey if given, otherwise 0 if colors is numeric and "grey" otherwise.

allOK Boolean flag signalling whether all eigengenes have been calculated correctly, either as principal components or as the hubgene average approximation.

allPC Boolean flag signalling whether all returned eigengenes are principal components.

isPC Boolean vector. Each component (corresponding to the columns in eigengenes) is TRUE if the corresponding eigengene is the first principal component and FALSE if it is the hubgene approximation or is invalid.

isHub Boolean vector. Each component (corresponding to the columns in eigengenes) is TRUE if the corresponding eigengene is the hubgene approximation and FALSE if it is the first principal component or is invalid.

validAEs Boolean vector. Each component (corresponding to the columns in eigengenes) is TRUE if the corresponding module average expression is valid.

allAEOK Boolean flag signalling whether all returned module average expressions contain valid data. Note that returnValidOnly==TRUE does not imply allAEOK==TRUE: some invalid average expressions may be returned if their corresponding eigengenes have been calculated correctly.

Author(s)

Steve Horvath <SHorvath@mednet.ucla.edu>, Peter Langfelder <Peter.Langfelder@gmail.com>

References

Zhang, B. and Horvath, S. (2005), "A General Framework for Weighted Gene Co-Expression Network Analysis", Statistical Applications in Genetics and Molecular Biology: Vol. 4: No. 1, Article 17

See Also

svd, impute.knn

`moduleNumber` *Fixed-height cut of a dendrogram.*

Description

Detects branches of on the input dendrogram by performing a fixed-height cut.

Usage

```
moduleNumber(dendro, cutHeight = 0.9, minSize = 50)
```

Arguments

`dendro`	a hierarchical clustering dendorgram such as one returned by `hclust`.
`cutHeight`	Maximum joining heights that will be considered.
`minSize`	Minimum cluster size.

Details

All contiguous branches below the height `cutHeight` that contain at least `minSize` objects are assigned unique positive numerical labels; all unassigned objects are assigned label 0.

Value

A vector of numerical labels giving the assigment of each object.

Note

The numerical labels may not be sequential. See `normalizeLabels` for a way to put the labels into a standard order.

Author(s)

Peter Langfelder, `<Peter.Langfelder@gmail.com>`

See Also

`hclust`, `cutree`, `normalizeLabels`

`modulePreservation` *Calculation of module preservation statistics*

Description

Calculations of module preservation statistics between independent data sets.

Usage

```
modulePreservation(
   multiData,
   multiColor,
   dataIsExpr = TRUE,
   networkType = "unsigned",
   corFnc = "cor",
   corOptions = "use = 'p'",
   referenceNetworks = 1,
   nPermutations = 100,
   randomSeed = 12345,
   maxGoldModuleSize = 1000,
   maxModuleSize = 1000,
   quickCor = 1,
   ccTupletSize = 2,
   calculateCor.kIMall = TRUE,
   useInterpolation = FALSE,
   checkData = TRUE,
   greyName = NULL,
   savePermutedStatistics = TRUE,
   loadPermutedStatistics = FALSE,
   permutedStatisticsFile = if (useInterpolation) "permutedStats-intrModules.RDa
                                    else "permutedStats-actualModules.RData",
   plotInterpolation = TRUE,
   interpolationPlotFile = "modulePreservationInterpolationPlots.pdf",
   discardInvalidOutput = TRUE,
   verbose = 1, indent = 0)
```

Arguments

`multiData`	expression data or adjacency data in the multi-set format (see `checkSets`). A vector of lists, one per set. Each set must contain a component `data` that contains the expression or adjacency data. If expression data are used, rows correspond to samples and columns to genes or probes. In case of adjacencies, each `data` matrix should be a symmetric matrix ith entries between 0 and 1 and unit diagonal. Each component of the outermost list should be named.
`multiColor`	a list in which every component is a vector giving the module labels of genes in `multiExpr`. The components must be named using the same names that are used in `multiExpr`; these names are used top match labels to expression data sets. See details.
`dataIsExpr`	logical: if `TRUE`, `multiData` will be interpreted as expression data; if `FALSE`, `multiData` will be interpreted as adjacencies.

networkType network type. Allowed values are (unique abbreviations of) "unsigned", "signed", "signed hybrid". See adjacency.

corFnc character string specifying the function to be used to calculate co-expression similarity. Defaults to Pearson correlation. Another useful choice is bicor. More generally, any function returning values between -1 and 1 can be used.

corOptions character string specifying additional arguments to be passed to the function given by corFnc. Use "use = 'p', method = 'Spearman'" to obtain Spearman correlation.

referenceNetworks
a vector giving the indices of expression data to be used as reference networks. Reference networks must have their module labels given in multiColor.

nPermutations
specifies the number of permutations that will be calculated in the permutation test.

randomSeed seed for the random number generator. If NULL, the seed will not be set. If non-NULL and the random generator has been initialized prior to the function call, the latter's state is saved and restored upon exit

maxGoldModuleSize
maximum size of the "gold" module, i.e., the random sample of all network genes.

maxModuleSize
maximum module size used for calculations. Modules larger than maxModuleSize will be reduced by randomly sampling maxModuleSize genes.

quickCor number between 0 and 1 specifying the handling of missing data in calculation of correlation. Zero means exact but potentially slower calculations; one means potentially faster calculations, but with potentially inaccurate results if the proportion of missing data is large. See cor for more details.

ccTupletSize tuplet size for co-clustering calculations.

calculateCor.kIMall
logical: should cor.kMEall be calculated? This option is only valid for adjacency input. If FALSE, cor.kIMall will not be calculated, potentially saving significant amount of time if the input adjacencies are large and contain many modules.

checkData logical: should data be checked for excessive number of missing entries? See goodSamplesGenesMS for details.

greyName label used for unassigned genes. Traditionally such genes are labeled by grey color or numeric label 0. These values are the default when multiColor contains character or numeric vectors, respectively.

savePermutedStatistics
logical: should calculated permutation statistics be saved? Saved statistics may be re-used if the calculation needs to be repeated.

permutedStatisticsFile
file name to save the permutation statistics into.

loadPermutedStatistics
logical: should permutation statistics be loaded? If a previously executed calculation needs to be repeated, loading permutation study results can cut the calculation time many-fold.

useInterpolation
logical: should permutation statistics be calculated by interpolating an artificial set of evenly spaced modules? This option may potentially speed up the calculations, but it restricts calculations to density measures.

`plotInterpolation`
: logical: should interpolation plots be saved? If interpolation is used (see `useInterpolation` above), the function can optionally generate diagnostic plots that can be used to assess whether the interpolation makes sense.

`interpolationPlotFile`
: file name to save the interpolation plots into.

`discardInvalidOutput`
: logical: should output columns containing no valid data be discarded? This option may be useful when input `dataIsExpr` is `FALSE` and some of the output statistics cannot be calculated. This option causes such statistics to be dropped from output.

`verbose`
: integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.

`indent`
: indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

This function calculates module preservation statistics pair-wise between given reference sets and all other sets in `multiExpr`. Reference sets must have their corresponding module assignment specified in `multiColor`; module assignment is optional for test sets. Individual expression sets and their module labels are matched using `names` of the corresponding components in `multiExpr` and `multiColor`.

For each reference-test pair, the function calculates module preservation statistics that measure how well the modules of the reference set are preserved in the test set. If the `multiColor` also contains module assignment for the test set, the calculated statistics also include cross-tabulation statistics that make use of the test module assignment.

For each reference-test pair, the function only uses genes (columns of the `data` component of each component of `multiExpr`) that are in common between the reference and test set. Columns are matched by column names, so column names must be valid.

In addition to preservation statistics, the function also calculates several statistics of module quality, that is measures of how well-defined modules are in the reference set. The quality statistics are calculated with respect to genes in common with with a test set; thus the function calculates a set of quality statistics for each reference-test pair. This may be somewhat counter-intuitive, but it allows a direct comparison of corresponding quality and preservation statistics.

Missing data are removed (but see `quickCor` above).

Value

The function returns a nested list of preservation statistics. At the top level, the list components are:

`quality`
: observed values and Z scores of quality statistics

`preservation`
: observed values and Z scores of density and connectivity preservation statistics

`accuracy`
: observed values and Z scores of cross-tabulation statistics

`referenceSeparability`
: observed values and Z scores of module separability in the reference network

`testSeparability`
: observed values and Z scores of module separability in the test network

`permutationDetails`
: results of individual permutations, useful for diagnostics

All of the above are lists. The lists `quality`, `preservation`, `referenceSeparability`, and `testSeparability` each contain 2 components: `observed` contains observed values, and `Z` contains the corresponding Z scores. The list `accuracy` contains `observed`, `Z`, and an additional components `observedOverlapCounts` and `observedFisherPvalues` that contain the observed matrices of overlap counts and Fisher test p-values.

Each of the lists `observed`, `Z`, `observedOverlapCounts` and `observedFisherPvalues` is structured as a 2-level list where the outer components correspond to reference sets and the inner components to tests sets. As an example, `preservation$observed[[1]][[2]]` contains the density and connectivity preservation statistics for the preservation of set 1 modules in set 2, that is set 1 is the reference set and set 2 is the test set. `preservation$observed[[1]][[2]]` is a data frame in which each row corresponds to a module in the reference network 1 plus one row for the unassigned objects, and one row for a "module" that contains randomly sampled objects and that represents a whole-network average. Each column corresponds to a statistic as indicated by the column name.

Note

For large data sets, the permutation study may take a while (typically on the order of several hours). Use `verbose = 3` to get detailed progress report as the calculations advance.

Author(s)

Rui Luo and Peter Langfelder

References

Peter Langfelder, Rui Luo, Michael C. Oldham, and Steve Horvath, to appear

See Also

Network construction and module detection functions in the WGCNA package such as `adjacency`, `blockwiseModules`; rudimentary cleaning in `goodSamplesGenesMS`; the WGCNA implementation of correlation in `cor`.

`multiSetMEs` *Calculate module eigengenes.*

Description

Calculates module eigengenes for several sets.

Usage

```
multiSetMEs(exprData,
            colors,
            universalColors = NULL,
            useSets = NULL,
            useGenes = NULL,
            impute = TRUE,
            nPC = 1,
            align = "along average",
```

```
                excludeGrey = FALSE,
                grey = ifelse(is.null(universalColors), ifelse(is.numeric(colors), 0
                              ifelse(is.numeric(universalColors), 0, "grey")),
                subHubs = TRUE,
                trapErrors = FALSE,
                returnValidOnly = trapErrors,
                softPower = 6,
                verbose = 1, indent = 0)
```

Arguments

exprData	Expression data in a multi-set format (see checkSets). A vector of lists, with each list corresponding to one microarray dataset and expression data in the component data, that is expr[[set]]$data[sample, probe] is the expression of probe probe in sample sample in dataset set. The number of samples can be different between the sets, but the probes must be the same.
colors	A matrix of dimensions (number of probes, number of sets) giving the module assignment of each gene in each set. The color "grey" is interpreted as unassigned.
universalColors	Alternative specification of module assignment. A single vector of length (number of probes) giving the module assignment of each gene in all sets (that is the modules are common to all sets). If given, takes precedence over color.
useSets	If calculations are requested in (a) selected set(s) only, the set(s) can be specified here. Defaults to all sets.
useGenes	Can be used to restrict calculation to a subset of genes (the same subset in all sets). If given, validColors in the returned list will only contain colors for the genes specified in useGenes.
impute	Logical. If TRUE, expression data will be checked for the presence of NA entries and if the latter are present, numerical data will be imputed, using function impute.knn and probes from the same module as the missing datum. The function impute.knn uses a fixed random seed giving repeatable results.
nPC	Number of principal components to be calculated. If only eigengenes are needed, it is best to set it to 1 (default). If variance explained is needed as well, use value NULL. This will cause all principal components to be computed, which is slower.
align	Controls whether eigengenes, whose orientation is undetermined, should be aligned with average expression (align = "along average", the default) or left as they are (align = ""). Any other value will trigger an error.
excludeGrey	Should the improper module consisting of 'grey' genes be excluded from the eigengenes?
grey	Value of colors or universalColors (whichever applies) designating the improper module. Note that if the appropriate colors argument is a factor of numbers, the default value will be incorrect.
subHubs	Controls whether hub genes should be substituted for missing eigengenes. If TRUE, each missing eigengene (i.e., eigengene whose calculation failed and the error was trapped) will be replaced by a weighted average of the most connected hub genes in the corresponding module. If this calculation fails, or if subHubs==FALSE, the value of trapErrors will determine whether the offending module will be removed or whether the function will issue an error and stop.

trapErrors Controls handling of errors from that may arise when there are too many NA entries in expression data. If TRUE, errors from calling these functions will be trapped without abnormal exit. If FALSE, errors will cause the function to stop. Note, however, that subHubs takes precedence in the sense that if subHubs==TRUE and trapErrors==FALSE, an error will be issued only if both the principal component and the hubgene calculations have failed.

returnValidOnly Boolean. Controls whether the returned data frames of module eigengenes contain columns corresponding only to modules whose eigengenes or hub genes could be calculated correctly in every set (TRUE), or whether the data frame should have columns for each of the input color labels (FALSE).

softPower The power used in soft-thresholding the adjacency matrix. Only used when the hubgene approximation is necessary because the principal component calculation failed. It must be non-negative. The default value should only be changed if there is a clear indication that it leads to incorrect results.

verbose Controls verbosity of printed progress messages. 0 means silent, up to (about) 5 the verbosity gradually increases.

indent A single non-negative integer controlling indentation of printed messages. 0 means no indentation, each unit above that adds two spaces.

Details

This function calls moduleEigengenes for each set in exprData.

Module eigengene is defined as the first principal component of the expression matrix of the corresponding module. The calculation may fail if the expression data has too many missing entries. Handling of such errors is controlled by the arguments subHubs and trapErrors. If subHubs==TRUE, errors in principal component calculation will be trapped and a substitute calculation of hubgenes will be attempted. If this fails as well, behaviour depends on trapErrors: if TRUE, the offending module will be ignored and the return value will allow the user to remove the module from further analysis; if FALSE, the function will stop. If universalColors is given, any offending module will be removed from all sets (see validMEs in return value below).

From the user's point of view, setting trapErrors=FALSE ensures that if the function returns normally, there will be a valid eigengene (principal component or hubgene) for each of the input colors. If the user sets trapErrors=TRUE, all calculational (but not input) errors will be trapped, but the user should check the output (see below) to make sure all modules have a valid returned eigengene.

While the principal component calculation can fail even on relatively sound data (it does not take all that many "well-placed" NA to torpedo the calculation), it takes many more irregularities in the data for the hubgene calculation to fail. In fact such a failure signals there likely is something seriously wrong with the data.

Value

A vector of lists similar in spirit to the input exprData. For each set there is a list with the following components:

data Module eigengenes in a data frame, with each column corresponding to one eigengene. The columns are named by the corresponding color with an "ME" prepended, e.g., MEturquoise etc. Note that, when trapErrors == TRUE and returnValidOnly==FALSE, this data frame also contains entries corresponding to removed modules, if any. (validMEs below indicates which

eigengenes are valid and `allOK` whether all module eigengens were successfully calculated.)

`averageExpr` If `align == "along average"`, a dataframe containing average normalized expression in each module. The columns are named by the corresponding color with an `"AE"` prepended, e.g., `AEturquoise` etc.

`varExplained` A dataframe in which each column corresponds to a module, with the component `varExplained[PC, module]` giving the variance of module `module` explained by the principal component no. `PC`. This is only accurate if all principal components have been computed (input `nPC = NULL`). At most 5 principal components are recorded in this dataframe.

`nPC` A copy of the input `nPC`.

`validMEs` A boolean vector. Each component (corresponding to the columns in `data`) is `TRUE` if the corresponding eigengene is valid, and `FALSE` if it is invalid. Valid eigengenes include both principal components and their hubgene approximations. When `returnValidOnly==FALSE`, by definition all returned eigengenes are valid and the entries of `validMEs` are all `TRUE`.

`validColors` A copy of the input colors (`universalColors` if set, otherwise `colors[, set]`) with entries corresponding to invalid modules set to `grey` if given, otherwise 0 if the appropriate input colors are numeric and "grey" otherwise.

`allOK` Boolean flag signalling whether all eigengenes have been calculated correctly, either as principal components or as the hubgene approximation. If `universalColors` is set, this flag signals whether all eigengenes are valid in all sets.

`allPC` Boolean flag signalling whether all returned eigengenes are principal components. This flag (as well as the subsequent ones) is set independently for each set.

`isPC` Boolean vector. Each component (corresponding to the columns in `eigengenes`) is `TRUE` if the corresponding eigengene is the first principal component and `FALSE` if it is the hubgene approximation or is invalid.

`isHub` Boolean vector. Each component (corresponding to the columns in `eigengenes`) is `TRUE` if the corresponding eigengene is the hubgene approximation and `FALSE` if it is the first principal component or is invalid.

`validAEs` Boolean vector. Each component (corresponding to the columns in `eigengenes`) is `TRUE` if the corresponding module average expression is valid.

`allAEOK` Boolean flag signalling whether all returned module average expressions contain valid data. Note that `returnValidOnly==TRUE` does not imply `allAEOK==TRUE`: some invalid average expressions may be returned if their corresponding eigengenes have been calculated correctly.

Author(s)

Peter Langfelder, `<Peter.Langfelder@gmail.com>`

See Also

`moduleEigengenes`

na *Basic Statistical Functions for Handling Missing Values*

Description

Basic statistical functions for handling missing values or NA.
In `log.na`, `sum.na`, `mean.na` and `var.na`, `quantile.na`, `length.na`, missing values are omitted from the calculation.
The function `cor.na` calls `cor` with the argument `use="pairwise.complete.obs"`.
The function `order.na` only handles vector arguments and not lists. However, it gives the option of omitting the NAs (`na.last=NA`), of placing the NAs at the start of the ordered vector (`na.last=F`) or at the end (`na.last=T`).
The function `scale.na` is a modified version of `scale` which allows NAs in the variance calculation. If `scale = T`, the function `f` in `scale.na` uses `var.na` to perform the variance calculation. The function `prod.na` is similar to the `prod` function with `na.rm=TRUE`. This function returns the product of all the values present in its arguments, omitting any missing values.

Author(s)

Yee Hwa Yang, <yeehwa@stat.berkeley.edu>
Sandrine Dudoit, <sandrine@stat.berkeley.edu>

See Also

`log`, `sum`, `mean`, `var`, `cor`, `order`, `scale`, `link{prod}`.

nearestNeighborConnectivity *Connectivity to a constant number of nearest neighbors*

Description

Given expression data and basic network parameters, the function calculates connectivity of each gene to a given number of nearest neighbors.

Usage

```
nearestNeighborConnectivity(datExpr,
         nNeighbors = 50, power = 6, type = "unsigned",
         corFnc = "cor", corOptions = "use = 'p'",
         blockSize = 1000,
         sampleLinks = NULL, nLinks = 5000, setSeed = 38457,
         verbose = 1, indent = 0)
```

Arguments

datExpr	a data frame containing expression data, with rows corresponding to samples and columns to genes. Missing values are allowed and will be ignored.
nNeighbors	number of nearest neighbors to use.
power	soft thresholding power for network construction. Should be a number greater than 1.
type	a character string encoding network type. Recognized values are (unique abbreviations of) "unsigned", "signed", and "signed hybrid".
corFnc	character string containing the name of the function to calculate correlation. Suggested functions include "cor" and "bicor".
corOptions	further argument to the correlation function.
blockSize	correlation calculations will be split into square blocks of this size, to prevent running out of memory for large gene sets.
sampleLinks	logical: should network connections be sampled (TRUE) or should all connections be used systematically (FALSE)?
nLinks	number of links to be sampled. Should be set such that nLinks * nNeighbors be several times larger than the number of genes.
setSeed	seed to be used for sampling, for repeatability. If a seed already exists, it is saved before the sampling starts and restored upon exit.
verbose	integer controlling the level of verbosity. 0 means silent.
indent	integer controlling indentation of output. Each unit above 0 adds two spaces.

Details

Connectivity of gene i is the sum of adjacency strengths between gene i and other genes; in this case we take the nNeighbors nodes with the highest connection strength to gene i. The adjacency strengths are calculated by correlating the given expression data using the function supplied in corFNC and transforming them into adjacency according to the given network type and power.

Value

A vector with one component for each gene containing the nearest neighbor connectivity.

Author(s)

Peter Langfelder

See Also

adjacency, softConnectivity

nearestNeighborConnectivityMS
Connectivity to a constant number of nearest neighbors across multiple data sets

Description

Given expression data from several sets and basic network parameters, the function calculates connectivity of each gene to a given number of nearest neighbors in each set.

Usage

```
nearestNeighborConnectivityMS(multiExpr, nNeighbors = 50, power = 6,
          type = "unsigned", corFnc = "cor", corOptions = "use = 'p'",
          blockSize = 1000,
          sampleLinks = NULL, nLinks = 5000, setSeed = 36492,
          verbose = 1, indent = 0)
```

Arguments

multiExpr	expression data in multi-set format. A vector of lists, one list per set. In each list there must be a component named data whose content is a matrix or dataframe or array of dimension 2 containing the expression data. Rows correspond to samples and columns to genes (probes).
nNeighbors	number of nearest neighbors to use.
power	soft thresholding power for network construction. Should be a number greater than 1.
type	a character string encoding network type. Recognized values are (unique abbreviations of) "unsigned", "signed", and "signed hybrid".
corFnc	character string containing the name of the function to calculate correlation. Suggested functions include "cor" and "bicor".
corOptions	further argument to the correlation function.
blockSize	correlation calculations will be split into square blocks of this size, to prevent running out of memory for large gene sets.
sampleLinks	logical: should network connections be sampled (TRUE) or should all connections be used systematically (FALSE)?
nLinks	number of links to be sampled. Should be set such that nLinks * nNeighbors be several times larger than the number of genes.
setSeed	seed to be used for sampling, for repeatability. If a seed already exists, it is saved before the sampling starts and restored after.
verbose	integer controlling the level of verbosity. 0 means silent.
indent	integer controlling indentation of output. Each unit above 0 adds two spaces.

Details

Connectivity of gene i is the sum of adjacency strengths between gene i and other genes; in this case we take the nNeighbors nodes with the highest connection strength to gene i. The adjacency strengths are calculated by correlating the given expression data using the function supplied in corFNC and transforming them into adjacency according to the given network type and power.

Value

A matrix in which columns correspond to sets and rows to genes; each entry contains the nearest neighbor connectivity of the corresponding gene.

Author(s)

Peter Langfelder

See Also

adjacency, softConnectivity, nearestNeighborConnectivity

---

networkConcepts *Calculations of network concepts*

Description

This functions calculates various network concepts (topological properties, network indices) of a network calculated from expression data. See details for a detailed description.

Usage

```
networkConcepts(datExpr, power = 1, trait = NULL, networkType = "unsigned")
```

Arguments

datExpr	a data frame containg the expression data, with rows corresponding to samples and columns to genes (nodes).
power	soft thresholding power.
trait	optional specification of a sample trait. A vector of length equal the number of samples in datExpr.
networkType	network type. Recognized values are (unique abbreviations of) "unsigned", "signed", and "signed hybrid".

Details

This function computes various network concepts (also known as network statistics, topological properties, or network indices) for a weighted correlation network. The nodes of the weighted correlation network will be constructed between the columns (interpreted as nodes) of the input datExpr. If the option networkType="unsigned" then the adjacency between nodes i and j is defined as A[i,j]=abs(cor(datExpr[,i],datExpr[,j]))^power. In the following, we use the term gene and node interchangeably since these methods were originally developed for gene networks. The function computes the following 4 types of network concepts (introduced in Horvath and Dong 2008):

Type I: fundamental network concepts are defined as a function of the off-diagonal elements of an adjacency matrix A and/or a node significance measure GS. These network concepts can be defined for any network (not just correlation networks). The adjacency matrix of an unsigned weighted correlation network is given by A=abs(cor(datExpr,use="p"))^power and the trait based gene significance measure is given by GS= abs(cor(datExpr,trait, use="p"))^power where datExpr, trait, power are input parameters.

Type II: conformity-based network concepts are functions of the off-diagonal elements of the conformity based adjacency matrix `A.CF=CF*t(CF)` and/or the node significance measure. These network concepts are defined for any network for which a conformity vector can be defined. Details: For any adjacency matrix `A`, the conformity vector `CF` is calculated by requiring that `A[i,j]` is approximately equal to `CF[i]*CF[j]`. Using the conformity one can define the matrix `A.CF=CF*t(CF)` which is the outer product of the conformity vector with itself. In general, `A.CF` is not an adjacency matrix since its diagonal elements are different from 1. If the off-diagonal elements of `A.CF` are similar to those of `A` according to the Frobenius matrix norm, then `A` is approximately factorizable. To measure the factorizability of a network, one can calculate the `Factorizability`, which is a number between 0 and 1 (Dong and Horvath 2007). T he conformity is defined using a monotonic, iterative algorithm that maximizes the factorizability measure.

Type III: approximate conformity based network concepts are functions of all elements of the conformity based adjacency matrix `A.CF` (including the diagonal) and/or the node significance measure `GS`. These network concepts are very useful for deriving relationships between network concepts in networks that are approximately factorizable.

Type IV: eigengene-based (also known as eigennode-based) network concepts are functions of the eigengene-based adjacency matrix `A.E=ConformityE*t(ConformityE)` (diagonal included) and/or the corresponding eigengene-based gene significance measure `GSE`. These network concepts can only be defined for correlation networks. Details: The columns (nodes) of `datExpr` can be summarized with the first principal component, which is referred to as Eigengene in coexpression network analysis. In general correlation networks, it is called eigennode. The eigengene-based conformity `ConformityE[i]` is defined as `abs(cor(datE[,i], Eigengene))^power` where the power corresponds to the power used for defining the weighted adjacency matrix `A`. The eigengene-based conformity can also be used to define an eigengene-based adjacency matrix `A.E=ConformityE*t(ConformityE)`. The eigengene based factorizability `EF(datE)` is a number between 0 and 1 that measures how well `A.E` approximates `A` when the power parameter equals 1. `EF(datE)` is defined with respect to the singular values of `datExpr`. For a trait based node significance measure `GS=abs(cor(datE,trait))^power`, one can also define an eigengene-based node significance measure `GSE[i]=ConformityE[i]*EigengeneSignificance` where the eigengene significance `abs(cor(Eigengene,trait))^power` is defined as power of the absolute value of the correlation between eigengene and trait. Eigengene-based network concepts are very useful for providing a geometric interpretation of network concepts and for deriving relationships between network concepts. For example, the hub gene significance measure and its eigengene-based analog have been used to characterize networks where highly connected hub genes are important with regard to a trait based gene significance measure (Horvath and Dong 2008).

Value

A list with the following components:

`Summary` — a data frame whose rows report network concepts that only depend on the adjacency matrix. Density (mean adjacency), Centralization , Heterogeneity (coefficient of variation of the connectivity), Mean ClusterCoef, Mean Connectivity. The columns of the data frame report the 4 types of network concepts mentioned in the description: Fundamental concepts, eigengene-based concepts, conformity-based concepts, and approximate conformity-based concepts.

`Size` — reports the network size, i.e. the number of nodes, which equals the number of columns of the input data frame `datExpr`.

`Factorizability` — a number between 0 and 1. The closer it is to 1, the better the off-diagonal elements of the conformity based network `A.CF` approximate those of `A` (according to the Frobenius norm).

`Eigengene` the first principal component of the standardized columns of `datExpr`. The number of components of this vector equals the number of rows of `datExpr`.

`VarExplained` the proportion of variance explained by the first principal component (the `Eigengene`). It is numerically different from the eigengene based factorizability. While `VarExplained` is based on the squares of the singular values of `datExpr`, the eigengene-based factorizability is based on fourth powers of the singular values.

`Conformity` numerical vector giving the conformity. The number of components of the conformity vector equals the number of columns in `datExpr`. The conformity is often highly correlated with the vector of node connectivities. The conformity is computed using an iterative algorithm for maximizing the factorizability measure. The algorithm and related network concepts are described in Dong and Horvath 2007.

`ClusterCoef` a numerical vector that reports the cluster coefficient for each node. This fundamental network concept measures the cliquishness of each node.

`Connectivity` a numerical vector that reports the connectivity (also known as degree) of each node. This fundamental network concept is also known as whole network connectivity. One can also define the scaled connectivity `K=Connectivity/max(Connectivity)` which is used for computing the hub gene significance.

`MAR` a numerical vector that reports the maximum adjacency ratio for each node. `MAR[i]` equals 1 if all non-zero adjacencies between node `i` and the remaining network nodes equal 1. This fundamental network concept is always 1 for nodes of an unweighted network. This is a useful measure for weighted networks since it allows one to determine whether a node has high connectivity because of many weak connections (small MAR) or because of strong (but few) connections (high MAR), see Horvath and Dong 2008.

`ConformityE` a numerical vector that reports the eigengene based (aka eigenenode based) conformity for the correlation network. The number of components equals the number of columns of `datExpr`.

`GS` a numerical vector that encodes the node (gene) significance. The i-th component equals the node significance of the i-th column of `datExpr` if a sample trait was supplied to the function (input trait). `GS[i]=abs(cor(datE[,i], trait, use="p"))^power`

`GSE` a numerical vector that reports the eigengene based gene significance measure. Its i-th component is given by `GSE[i]=ConformityE[i]*EigengeneSignificance` where the eigengene significance `abs(cor(Eigengene,trait))^power` is defined as power of the absolute value of the correlation between eigengene and trait.

`Significance` a data frame whose rows report network concepts that also depend on the trait based node significance measure. The rows correspond to network concepts and the columns correspond to the type of network concept (fundamental versus eigengene based). The first row of the data frame reports the network significance. The fundamental version of this network concepts is the average gene significance=mean(GS). The eigengene based analog of this concept is defined as mean(GSE). The second row reports the hub gene significance which is defined as slope of the intercept only regression model that regresses the gene significance on the scaled network connectivity K. The third row reports the eigengene significance `abs(cor(Eigengene,trait))^power`. More details can be found in Horvath and Dong (2008).

Author(s)

Jun Dong, Steve Horvath, Peter Langfelder

References

Bin Zhang and Steve Horvath (2005) "A General Framework for Weighted Gene Co-Expression Network Analysis", Statistical Applications in Genetics and Molecular Biology: Vol. 4: No. 1, Article 17

Dong J, Horvath S (2007) Understanding Network Concepts in Modules, BMC Systems Biology 2007, 1:24

Horvath S, Dong J (2008) Geometric Interpretation of Gene Coexpression Network Analysis. PLoS Comput Biol 4(8): e1000117

---

networkScreening *~~function to do ... ~~*

Description

~~ A concise (1-5 lines) description of what the function does. ~~

Usage

```
networkScreening(y, datME, datExpr, oddPower = 3, blockSize = 1000, minimumSampl
addMEy = TRUE, removeDiag = FALSE, weightESy = 0.5, getQValues = TRUE)
```

Arguments

`y`	~~Describe `y` here~~
`datME`	~~Describe `datME` here~~
`datExpr`	~~Describe `datExpr` here~~
`oddPower`	~~Describe `oddPower` here~~
`blockSize`	~~Describe `blockSize` here~~
`minimumSampleSize`	~~Describe `minimumSampleSize` here~~
`addMEy`	~~Describe `addMEy` here~~
`removeDiag`	~~Describe `removeDiag` here~~
`weightESy`	~~Describe `weightESy` here~~
`getQValues`	~~Describe `weightESy` here~~

Details

~~ If necessary, more details than the description above ~~

Value

~Describe the value returned If it is a LIST, use

`comp1`	Description of 'comp1'
`comp2`	Description of 'comp2'

...

Note

~~further notes~~

Author(s)

~~who you are~~

References

~put references to the literature/web site here ~

See Also

~~objects to See Also as help, ~~~

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (y, datME, datExpr, oddPower = 3, blockSize = 1000,
    MinimumSampleSize = ..minNSamples, addMEy = TRUE, removeDiag = FALSE,
    weightESy = 0.5)
{
    oddPower = as.integer(oddPower)
    if (as.integer(oddPower/2) == oddPower/2) {
        oddPower = oddPower + 1
    }
    nMEs = dim(as.matrix(datME))[[2]]
    nGenes = dim(as.matrix(datExpr))[[2]]
    if (nGenes > nMEs & addMEy) {
        datME = data.frame(y, datME)
    }
    nMEs = dim(as.matrix(datME))[[2]]
    RawCor.Weighted = rep(0, nGenes)
    Cor.Standard = as.numeric(cor(y, datExpr, use = "p"))
    NoAvailable = apply(!is.na(datExpr), 2, sum)
    Cor.Standard[NoAvailable < MinimumSampleSize] = NA
    if (nGenes == 1)
        RawCor.Weighted = as.numeric(cor(y, datExpr, use = "p"))
    nBlocks = as.integer(nMEs/blockSize)
    if (nBlocks > 0)
        for (i in 1:nBlocks) {
            printFlush(paste("block number = ", i))
            index1 = c(1:blockSize) + (i - 1) * blockSize
            datMEBatch = datME[, index1]
            datKMEBatch = as.matrix(signedKME(datExpr, datMEBatch,
                outputColumnName = "MM."))
            ES.CorBatch = as.vector(cor(as.numeric(as.character(y)),
                datMEBatch, use = "p"))
            ES.CorBatch[ES.CorBatch > 0.999] = weightESy * 1 +
                (1 - weightESy) * max(abs(ES.CorBatch[ES.CorBatch <
                  0.999]), na.rm = T)
            if (nGenes == nMEs & removeDiag) {
```

```
            diag(datKMEBatch[index1, ]) = 0
        }
        if (nGenes == nMEs) {
            datKMEBatch[is.na(datKMEBatch)] = 0
            ES.CorBatch[is.na(ES.CorBatch)] = 0
        }
        RawCor.WeightedBatch = as.matrix(datKMEBatch)^oddPower %*%
            as.matrix(ES.CorBatch^oddPower)
        RawCor.Weighted = RawCor.Weighted + RawCor.WeightedBatch
    }
if (nMEs - nBlocks * blockSize > 0) {
    restindex = c((nBlocks * blockSize + 1):nMEs)
    datMEBatch = datME[, restindex]
    datKMEBatch = as.matrix(signedKME(datExpr, datMEBatch,
        outputColumnName = "MM."))
    ES.CorBatch = as.vector(cor(as.numeric(as.character(y)),
        datMEBatch, use = "p"))
    ES.CorBatch[ES.CorBatch > 0.999] = weightESy * 1 + (1 -
        weightESy) * max(abs(ES.CorBatch[ES.CorBatch < 0.999]),
        na.rm = T)
    if (nGenes == nMEs & removeDiag) {
        diag(datKMEBatch[restindex, ]) = 0
    }
    if (nGenes == nMEs) {
        datKMEBatch[is.na(datKMEBatch)] = 0
        ES.CorBatch[is.na(ES.CorBatch)] = 0
    }
    RawCor.WeightedBatch = as.matrix(datKMEBatch)^oddPower %*%
        ES.CorBatch^oddPower
    RawCor.Weighted = RawCor.Weighted + RawCor.WeightedBatch
}
RawCor.Weighted = RawCor.Weighted/nMEs
RawCor.Weighted[NoAvailable < MinimumSampleSize] = NA
if (max(abs(RawCor.Weighted), na.rm = T) == 1)
    RawCor.Weighted = RawCor.Weighted/1.0000001
if (max(abs(Cor.Standard), na.rm = T) == 1)
    Cor.Standard = Cor.Standard/1.0000001
RawZ.Weighted = sqrt(NoAvailable - 2) * RawCor.Weighted/sqrt(1 -
    RawCor.Weighted^2)
Z.Standard = sqrt(NoAvailable - 2) * Cor.Standard/sqrt(1 -
    Cor.Standard^2)
if (sum(abs(Z.Standard), na.rm = T) > 0) {
    Z.Weighted = RawZ.Weighted/sum(abs(RawZ.Weighted), na.rm = T) *
        sum(abs(Z.Standard), na.rm = T)
}
h1 = Z.Weighted/sqrt(NoAvailable - 2)
Cor.Weighted = h1/sqrt(1 + h1^2)
p.Weighted = as.numeric(2 * (1 - pt(abs(Z.Weighted), NoAvailable -
    2)))
p.Standard = 2 * (1 - pt(abs(Z.Standard), NoAvailable - 2))
p.Weighted2 = p.Weighted
p.Standard2 = p.Standard
p.Weighted2[is.na(p.Weighted)] = 1
p.Standard2[is.na(p.Standard)] = 1
q.Weighted = try(qvalue(p.Weighted2)$qvalues)
q.Standard = try(qvalue(p.Standard2)$qvalues)
if (class(q.Weighted) == "try-error")
```

```
        q.Weighted = rep(NA, length(p.Weighted))
    if (class(q.Standard) == "try-error")
        q.Standard = rep(NA, length(p.Standard))
    rankCor.Weighted = rank(-abs(Cor.Weighted), ties.method = "first")
    rankCor.Standard = rank(-abs(Cor.Standard), ties.method = "first")
    printFlush(paste("Proportion of agreement between lists based on abs(Cor.Weighted) ar
    for (i in c(10, 20, 50, 100, 200, 500, 1000)) {
        printFlush(paste("Top ", i, " list of genes: prop. agree = ",
            signif(sum(rankCor.Weighted <= i & rankCor.Standard <=
                i, na.rm = T)/i, 3)))
    }
    datout = data.frame(p.Weighted, q.Weighted, Cor.Weighted,
        Z.Weighted, p.Standard, q.Standard, Cor.Standard, Z.Standard)
    datout
  }
```

`networkScreeningGS` *~~function to do ... ~~*

Description

~~ A concise (1-5 lines) description of what the function does. ~~

Usage

```
networkScreeningGS(datExpr, datME, GS, oddPower = 3, blockSize = 1000, minimumSa
addGS = TRUE)
```

Arguments

`datExpr`	~~Describe `datExpr` here~~
`datME`	~~Describe `datME` here~~
`GS`	~~Describe `GS` here~~
`oddPower`	~~Describe `oddPower` here~~
`blockSize`	~~Describe `blockSize` here~~
`minimumSampleSize`	~~Describe `minimumSampleSize` here~~
`addGS`	~~Describe `addGS` here~~

Details

~~ If necessary, more details than the description above ~~

Value

~Describe the value returned If it is a LIST, use

`comp1`	Description of 'comp1'
`comp2`	Description of 'comp2'

...

Note

~~further notes~~

Author(s)

~~who you are~~

References

~put references to the literature/web site here ~

See Also

~~objects to See Also as help, ~~~

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.
```

---

normalizeLabels *Transform numerical labels into normal order.*

Description

Transforms numerical labels into normal order, that is the largest group will be labeled 1, next largest 2 etc. Label 0 is optionally preserved.

Usage

```
normalizeLabels(labels, keepZero = TRUE)
```

Arguments

labels	Numerical labels.
keepZero	If TRUE (the default), labels 0 are preserved.

Value

A vector of the same length as input, containing the normalized labels.

Author(s)

Peter Langfelder, <Peter.Langfelder@gmail.com>

nPresent *Number of present data entries.*

Description

A simple sum of present entries in the argument.

Usage

```
nPresent(x)
```

Arguments

x	data in which to count number of present entries.

Value

A single number giving the number of present entries in x.

Author(s)

Steve Horvath

numbers2colors *Color representation for a numeric variable*

Description

The function creates a color represenation for the given numeric input.

Usage

```
numbers2colors(
   x,
   signed,
   centered = signed,
   lim = NULL,
   colors = if (signed) greenWhiteRed(100) else greenWhiteRed(100)[50:100],
   naColor = "grey")
```

Arguments

x	a vector or matrix of numbers. Missing values are allowed and will be assigned the color given in naColor. If a matrix, each column of the matrix is processed separately and the return value will be a matrix of colors.
signed	logical: should x be considered signed? If TRUE, the default setting is to use to use a palette that starts with green for the most negative values, continues with white for values around zero and turns red for positive values. If FALSE, the default palette ranges from white for minimum values to red for maximum values.

centered	logical. If TRUE and signed==TRUE, numeric value zero will correspond to the middle of the color palette. If FALSE or signed==FALSE, the middle of the color palette will correspond to the average of the minimum and maximum value.
lim	optional specification of limits, that is numeric values that should correspond to the first and last entry of colors.
colors	color palette to represent the given numbers.
naColor	color to represent missing values in x.

Details

Each column of x is processed individually, meaning that the color palette is adjusted individually for each column of x.

Value

A vector or matrix (of the same dimensions as x) of colors.

Author(s)

Peter Langfelder

See Also

labels2colors for color coding of ordinal labels.

---

orderMEs *Put close eigenvectors next to each other*

Description

Reorder given (eigen-)vectors such that similar ones (as measured by correlation) are next to each other.

Usage

```
orderMEs(MEs, greyLast = TRUE,
         greyName = paste(moduleColor.getMEprefix(), "grey", sep=""),
         orderBy = 1, order = NULL,
         useSets = NULL,  verbose = 0, indent = 0)
```

Arguments

MEs	Module eigengenes in a multi-set format (see checkSets). A vector of lists, with each list corresponding to one dataset and the module eigengenes in the component data, that is MEs[[set]]$data[sample, module] is the expression of the eigengene of module module in sample sample in dataset set. The number of samples can be different between the sets, but the modules must be the same.

greyLast	Normally the color grey is reserved for unassigned genes; hence the grey module is not a proper module and it is conventional to put it last. If this is not desired, set the parameter to FALSE.
greyName	Name of the grey module eigengene.
orderBy	Specifies the set by which the eigengenes are to be ordered (in all other sets as well). Defaults to the first set in useSets (or the first set, if useSets is not given).
order	Allows the user to specify a custom ordering.
useSets	Allows the user to specify for which sets the eigengene ordering is to be performed.
verbose	Controls verbostity of printed progress messages. 0 means silent, nonzero verbose.
indent	A single non-negative integer controling indentation of printed messages. 0 means no indentation, each unit above zero adds two spaces.

Details

Ordering module eigengenes is useful for plotting purposes. For this function the order can be specified explicitly, or a set can be given in which the correlations of the eigengenes will determine the order. For the latter, a hierarchical dendrogram is calculated and the order given by the dendrogram is used for the eigengenes in all other sets.

Value

A vector of lists of the same type as MEs containing the re-ordered eigengenes.

Author(s)

Peter Langfelder, <Peter.Langfelder@gmail.com>

See Also

moduleEigengenes, multiSetMEs, consensusOrderMEs

---

overlapTable *Calculate overlap of modules*

Description

The function calculates overlap counts and Fisher exact test p-values for the given two sets of module assignments.

Usage

```
overlapTable(labels1, labels2)
```

Arguments

labels1	a vector containing module labels.
labels2	a vector containing module labels to be compared to labels1.

Value

A list with the following components:

countTable	a matrix whose rows correspond to modules (unique labels) in labels1 and whose columns correspond to modules (unique labels) in labels2, giving the number of objects in the intersection of the two respective modules.
pTable	a matrix whose rows correspond to modules (unique labels) in labels1 and whose columns correspond to modules (unique labels) in labels2, giving Fisher's exact test significance p-values for the overlap of the two respective modules.

Author(s)

Peter Langfelder

See Also

fisher.test, matchLabels

---

pickHardThreshold *Analysis of scale free topology for hard-thresholding.*

Description

Analysis of scale free topology for multiple hard thresholds. The aim is to help the user pick an appropriate threshold for network construction.

Usage

```
pickHardThreshold(
  datExpr,
  RsquaredCut = 0.85,
  cutVector = seq(0.1, 0.9, by = 0.05),
  moreNetworkConcepts = FALSE,
  removeFirst = FALSE, nBreaks = 10,
  corFnc = "cor", corOptions = "use = 'p'")
```

Arguments

datExpr	expression data in a matrix or data frame. Rows correspond to samples and columns to genes.
RsquaredCut	desired minimum scale free topology fitting index R^2.
cutVector	a vector of hard threshold cuts for which the scale free topology fit indices are to be calculated.
moreNetworkConcepts	logical: should additional network concepts be calculated? If TRUE, the function will calculate how the network density, the network heterogeneity, and the network centralization depend on the power. For the definition of these additional network concepts, see Horvath and Dong (2008). PloS Comp Biol.
removeFirst	should the first bin be removed from the connectivity histogram?

nBreaks	number of bins in connectivity histograms
corFnc	a character string giving the correlation function to be used in adjacency calculation.
corOptions	further options to the correlation function specified in corFnc.

Details

The function calculates unsigned networks by thresholding the correlation matrix using thresholds given in cutVector. For each power the scale free topology fit index is calculated and returned along with other information on connectivity.

Value

A list with the following components:

cutEstimate	estimate of an appropriate hard-thresholding cut: the lowest cut for which the scale free topology fit R^2 exceeds RsquaredCut. If R^2 is below RsquaredCut for all cuts, NA is returned.
fitIndices	a data frame containing the fit indices for scale free topology. The columns contain the hard threshold, adjusted R^2 for the linear fit, the linear coefficient, adjusted R^2 for a more complicated fit models, mean connectivity, median connectivity and maximum connectivity. If input moreNetworkConcepts is TRUE, 3 additional columns containing network density, centralization, and heterogeneity.

Author(s)

Steve Horvath

References

Bin Zhang and Steve Horvath (2005) "A General Framework for Weighted Gene Co-Expression Network Analysis", Statistical Applications in Genetics and Molecular Biology: Vol. 4: No. 1, Article 17

Horvath S, Dong J (2008) Geometric Interpretation of Gene Coexpression Network Analysis. PLoS Comput Biol 4(8): e1000117

See Also

signumAdjacencyFunction

---

pickSoftThreshold *Analysis of scale free topology for soft-thresholding*

Description

Analysis of scale free topology for multiple soft thresholding powers. The aim is to help the user pick an appropriate soft-thresholding power for network construction.

Usage

```
pickSoftThreshold(
  datExpr,
  RsquaredCut = 0.85,
  powerVector = c(seq(1, 10, by = 1), seq(12, 20, by = 2)),
  removeFirst = FALSE, nBreaks = 10, blockSize = 1000,
  corFnc = "cor", corOptions = "use = 'p'",
  networkType = "unsigned",
  moreNetworkConcepts = FALSE,
  verbose = 0, indent = 0)
```

Arguments

`datExpr`	expression data in a matrix or data frame. Rows correspond to samples and columns to genes.
`RsquaredCut`	desired minimum scale free topology fitting index R^2.
`powerVector`	a vector of soft thresholding powers for which the scale free topology fit indices are to be calculated.
`removeFirst`	should the first bin be removed from the connectivity histogram?
`nBreaks`	number of bins in connectivity histograms
`blockSize`	block size into which the calculation of connectivity should be broken up. If R runs into memory problems, decrease this value.
`corFnc`	a character string giving the correlation function to be used in adjacency calculation.
`corOptions`	further options to the correlation function specified in `corFnc`.
`networkType`	network type. Allowed values are (unique abbreviations of) `"unsigned"`, `"signed"`, `"signed hybrid"`. See `adjacency`.
`moreNetworkConcepts`	logical: should additional network concepts be calculated? If `TRUE`, the function will calculate how the network density, the network heterogeneity, and the network centralization depend on the power. For the definition of these additional network concepts, see Horvath and Dong (2008). PloS Comp Biol.
`verbose`	integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
`indent`	indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

The function calculates unsigned networks by raising the absolute value of the correlation matrix to the powers given in `powerVector`. For each power the scale free topology fit index is calculated and returned along with other information on connectivity.

Value

A list with the following components:

`powerEstimate`	estimate of an appropriate soft-thresholding power: the lowest power for which the scale free topology fit R^2 exceeds `RsquaredCut`. If R^2 is below `RsquaredCut` for all powers, `NA` is returned.

`fitIndices` a data frame containing the fit indices for scale free topology. The columns contain the soft-thresholding power, adjusted R^2 for the linear fit, the linear coefficient, adjusted R^2 for a more complicated fit models, mean connectivity, median connectivity and maximum connectivity. If input `moreNetworkConcepts` is `TRUE`, 3 additional columns containing network density, centralization, and heterogeneity.

Author(s)

Steve Horvath and Peter Langfelder

References

Bin Zhang and Steve Horvath (2005) "A General Framework for Weighted Gene Co-Expression Network Analysis", Statistical Applications in Genetics and Molecular Biology: Vol. 4: No. 1, Article 17

Horvath S, Dong J (2008) Geometric Interpretation of Gene Coexpression Network Analysis. PLoS Comput Biol 4(8): e1000117

See Also

`adjacency`, `softConnectivity`

---

plot.cor *Red and Green Color Image of Correlation Matrix*

Description

This function produces a red and green color image of a correlation matrix using an RGB color specification. Increasingly positive correlations are represented with reds of increasing intensity, and increasingly negative correlations are represented with greens of increasing intensity.

Usage

```
plot.cor(x, new=FALSE, nrgcols=50, labels=FALSE, labcols=1, title="", ...)
```

Arguments

`x`	a matrix of numerical values.
`new`	If `new=F`, `x` must already be a correlation matrix. If `new=T`, the correlation matrix for the columns of `x` is computed and displayed in the image.
`nrgcols`	the number of colors (>= 1) to be used in the red and green palette.
`labels`	vector of character strings to be placed at the tickpoints, labels for the columns of `x`.
`labcols`	colors to be used for the labels of the columns of `x`. `labcols` can have either length 1, in which case all the labels are displayed using the same color, or the same length as `labels`, in which case a color is specified for the label of each column of `x`.
`title`	character string, overall title for the plot.
`...`	graphical parameters may also be supplied as arguments to the function (see `par`). For comparison purposes, it is good to set `zlim=c(-1,1)`.

Author(s)

Sandrine Dudoit, <sandrine@stat.berkeley.edu>

See Also

plot.mat,rgcolors.func, cor.na, cor, image, rgb.

---

plot.mat *Red and Green Color Image of Data Matrix*

Description

This function produces a red and green color image of a data matrix using an RGB color specification. Larger entries are represented with reds of increasing intensity, and smaller entries are represented with greens of increasing intensity.

Usage

```
plot.mat(x, nrgcols=50, rlabels=FALSE, clabels=FALSE, rcols=1, ccols=1, title=""
```

Arguments

x	a matrix of numbers.
nrgcols	the number of colors (>= 1) to be used in the red and green palette.
rlabels	vector of character strings to be placed at the row tickpoints, labels for the rows of x.
clabels	vector of character strings to be placed at the column tickpoints, labels for the columns of x.
rcols	colors to be used for the labels of the rows of x. rcols can have either length 1, in which case all the labels are displayed using the same color, or the same length as rlabels, in which case a color is specified for the label of each row of x.
ccols	colors to be used for the labels of the columns of x. ccols can have either length 1, in which case all the labels are displayed using the same color, or the same length as clabels, in which case a color is specified for the label of each column of x.
title	character string, overall title for the plot.
...	graphical parameters may also be supplied as arguments to the function (see par). E.g. zlim=c(-3,3)

Author(s)

Sandrine Dudoit, <sandrine@stat.berkeley.edu>

See Also

plot.cor, rgcolors.func, cor.na, cor, image, rgb.

`plotClusterTreeSamples` *Annotated clustering dendrogram of microarray samples*

Description

This function plots an annotated clustering dendorgram of microarray samples.

Usage

```
plotClusterTreeSamples(
  datExpr,
  y = NULL,
  traitLabels = NULL,
  yLabels = NULL,
  main = if (is.null(y)) "Sample dendrogram" else "Sample dendrogram and trait i
  setLayout = TRUE, autoColorHeight = TRUE, colorHeight = 0.3,
  dendroLabels = NULL,
  addGuide = FALSE, guideAll = TRUE,
  guideCount = NULL, guideHang = 0.2,
  cex.traitLabels = 0.8,
  cex.dendroLabels = 0.9,
  marAll = c(1, 5, 3, 1),
  saveMar = TRUE,
  abHeight = NULL, abCol = "red",
  ...)
```

Arguments

`datExpr`	a data frame containing expression data, with rows corresponding to samples and columns to genes. Missing values are allowed and will be ignored.
`y`	microarray sample trait. Either a vector with one entry per sample, or a matrix in which each column corresponds to a (different) trait and each row to a sample.
`traitLabels`	labels to be printed next to the color rows depicting sample traits. Defaults to column names of `y`.
`yLabels`	Optional labels to identify colors in the row identifying the sample classes. If given, must be of the same dimensions as `y`. Each label that occurs will be displayed once.
`main`	title for the plot.
`setLayout`	logical: should the plotting device be partitioned into a standard layout? If `FALSE`, the user is responsible for partitioning. The function expects two regions of the same width, the first one immediately above the second one.
`autoColorHeight`	logical: should the height of the color area below the dendrogram be automatically adjusted for the number of traits? Only effective if `setLayout` is `TRUE`.
`colorHeight`	Specifies the height of the color area under dendrogram as a fraction of the height of the dendrogram area. Only effective when `autoColorHeight` above is `FALSE`.

dendroLabels	dendrogram labels. Set to FALSE to disable dendrogram labels altogether; set to NULL to use row labels of datExpr.
addGuide	logical: should vertical "guide lines" be added to the dendrogram plot? The lines make it easier to identify color codes with individual samples.
guideAll	logical: add a guide line for every sample? Only effective for addGuide set TRUE.
guideCount	number of guide lines to be plotted. Only effective when addGuide is TRUE and guideAll is FALSE.
guideHang	fraction of the dendrogram height to leave between the top end of the guide line and the dendrogram merge height. If the guide lines overlap with dendrogram labels, increase guideHang to leave more space for the labels.
cex.traitLabels	character expansion factor for trait labels.
cex.dendroLabels	character expansion factor for dendrogram (sample) labels.
marAll	a 4-element vector giving the bottom, left, top and right margins around the combined plot. Note that this is not the same as setting the margins via a call to par, because the bottom margin of the dendrogram and the top margin of the color underneath are always zero.
saveMar	logical: save margins setting before starting the plot and restore on exit?
abHeight	optional specification of the height for a horizontal line in the dendrogram, see abline.
abCol	color for plotting the horizontal line.
...	other graphical parameters to plot.hclust.

Details

The function generates an average linkage hierarchical clustering dendrogram (see hclust) of samples from the given expression data, using Eclidean distance of samples. The dendrogram is plotted together with color annotation for the samples.

The trait y must be numeric. If y is integer, the colors will correspond to values. If y is continouos, it will be dichotomized to two classes, below and above median.

Value

None.

Author(s)

Steve Horvath and Peter Langfelder

See Also

dist, hclust, plotDendroAndColors

`plotColorUnderTree` *Plot color rows under a dendrogram*

Description

Plot color rows encoding information about objects in a clustering dendrogram, usually below the dendrogram.

Usage

```
plotColorUnderTree(dendro, colors, rowLabels = NULL, cex.rowLabels = 1, colorTex
```

Arguments

`dendro`	A dendrogram such as returned by `hclust`.
`colors`	Coloring of objects on the dendrogram. Either a vector (one color per object) or a matrix (can also be an array or a data frame) with each column giving one color per object. Each column will be plotted as a horizontal row of colors under the dendrogram.
`rowLabels`	Labels for the colorings given in `colors`. The labels will be printed to the left of the color rows in the plot. If the argument is given, it must be a vector of length equal to the number of columns in `colors`. If not given, `names(colors)` will be used if available. If not, sequential numbers starting from 1 will be used.
`cex.rowLabels`	Font size scale factor for the row labels. See `par`.
`colorText`	Optional labels to identify colors in the color rows. If given, must be of the same dimensions as `colors`. Each label that occurs will be displayed once.
`...`	Other parameters to be passed on to the plotting method (such as `main` for the main title etc).

Details

It is often useful to plot dendrograms of objects together with additional information about the objects, for example module assignment (by color) that was obtained by cutting a hierarchical dendrogram or external color-coded measures such as gene significance. This function provides a way to do so. The calling code should section the screen into two (or more) parts, plot the dendrogram (via `plot(hclust)`) in the upper section and use this function to plot color annotation in the order corresponding to the dendrogram in the lower section.

Value

None.

Note

This function is identical to the function `plotHclustColors` in package `moduleColor`.

Author(s)

Steve Horvath <SHorvath@mednet.ucla.edu> and Peter Langfelder <Peter.Langfelder@gmail.com>

See Also

cutreeDynamic for module detection in a dendrogram;

plotDendroAndColors for automated plotting of dendrograms and colors in one step.

---

plotDendroAndColors

Dendrogram plot with color annotation of objects

Description

This function plots a hierarchical clustering dendrogram and color annotation(s) of objects in the dendrogram underneath.

Usage

```
plotDendroAndColors(
  dendro,
  colors,
  groupLabels = NULL,
  colorText = NULL,
  setLayout = TRUE,
  autoColorHeight = TRUE,
  colorHeight = 0.2,
  dendroLabels = NULL,
  addGuide = FALSE, guideAll = FALSE,
  guideCount = 50, guideHang = 0.2,
  cex.colorLabels = 0.8, cex.dendroLabels = 0.9,
  marAll = c(1, 5, 3, 1), saveMar = TRUE,
  abHeight = NULL, abCol = "red", ...)
```

Arguments

dendro	a hierarchical clustering dendrogram such as one produced by hclust.
colors	Coloring of objects on the dendrogram. Either a vector (one color per object) or a matrix (can also be an array or a data frame) with each column giving one color per object. Each column will be plotted as a horizontal row of colors under the dendrogram.
groupLabels	Labels for the colorings given in colors. The labels will be printed to the left of the color rows in the plot. If the argument is given, it must be a vector of length equal to the number of columns in colors. If not given, names(colors) will be used if available. If not, sequential numbers starting from 1 will be used.
colorText	Optional labels to identify colors in the color rows. If given, must be of the same dimensions as colors. Each label that occurs will be displayed once.
setLayout	logical: should the plotting device be partitioned into a standard layout? If FALSE, the user is responsible for partitioning. The function expects two regions of the same width, the first one immediately above the second one.
autoColorHeight	logical: should the height of the color area below the dendrogram be automatically adjusted for the number of traits? Only effective if setLayout is TRUE.

colorHeight	Specifies the height of the color area under dendrogram as a fraction of the height of the dendrogram area. Only effective when autoColorHeight above is FALSE.
dendroLabels	dendrogram labels. Set to FALSE to disable dendrogram labels altogether; set to NULL to use row labels of datExpr.
addGuide	logical: should vertical "guide lines" be added to the dendrogram plot? The lines make it easier to identify color codes with individual samples.
guideAll	logical: add a guide line for every sample? Only effective for addGuide set TRUE.
guideCount	number of guide lines to be plotted. Only effective when addGuide is TRUE and guideAll is FALSE.
guideHang	fraction of the dendrogram height to leave between the top end of the guide line and the dendrogram merge height. If the guide lines overlap with dendrogram labels, increase guideHang to leave more space for the labels.
cex.colorLabels	character expansion factor for trait labels.
cex.dendroLabels	character expansion factor for dendrogram (sample) labels.
marAll	a vector of length 4 giving the bottom, left, top and right margins of the combined plot. There is no margin between the dendrogram and the color plot underneath.
saveMar	logical: save margins setting before starting the plot and restore on exit?
abHeight	optional specification of the height for a horizontal line in the dendrogram, see abline.
abCol	color for plotting the horizontal line.
...	other graphical parameters to plot.hclust.

Details

The function slits the plotting device into two regions, plots the given dendrogram in the upper region, then plots color rows in the region below the dendrogram.

Value

None.

Author(s)

Peter Langfelder

See Also

plotColorUnderTree

`plotEigengeneNetworks` *Eigengene network plot*

Description

This function plots dendrogram and eigengene representations of (consensus) eigengenes networks. In the case of conensus eigengene networks the function also plots pairwise preservation measures between consensus networks in different sets.

Usage

```
plotEigengeneNetworks(
  multiME,
  setLabels,
  letterSubPlots = FALSE, Letters = NULL,
  excludeGrey = TRUE, greyLabel = "grey",
  plotDendrograms = TRUE, plotHeatmaps = TRUE,
  setMargins = TRUE, marDendro = NULL, marHeatmap = NULL,
  colorLabels = TRUE, signed = TRUE,
  heatmapColors = NULL,
  plotAdjacency = TRUE,
  printAdjacency = FALSE, cex.adjacency = 0.9,
  coloredBarplot = TRUE, barplotMeans = TRUE, barplotErrors = FALSE,
  plotPreservation = "standard",
  zlimPreservation = c(0, 1),
  printPreservation = FALSE, cex.preservation = 0.9,
  ...)
```

Arguments

`multiME`	either a single data frame containing the module eigengenes, or module eigengenes in the multi-set format (see `checkSets`). The multi-set format is a vector of lists, one per set. Each set must contain a component `data` whose rows correspond to samples and columns to eigengenes.
`setLabels`	A vector of character strings that label sets in `multiME`.
`letterSubPlots`	logical: should subplots be lettered?
`Letters`	optional specification of a sequence of letters for lettering. Defaults to "ABCD"...
`excludeGrey`	logical: should the grey module eigengene be excluded from the plots?
`greyLabel`	label for the grey module. Usually either "grey" or the number 0.
`plotDendrograms`	logical: should eigengene dendrograms be plotted?
`plotHeatmaps`	logical: should eigengene network heatmaps be plotted?
`setMargins`	logical: should margins be set? See `par`.
`marDendro`	a vector of length 4 giving the margin setting for dendrogram plots. See `par`. If `setMargins` is `TRUE` and `marDendro` is not given, the function will provide reasonable default values.

`marHeatmap` a vector of length 4 giving the margin setting for heatmap plots. See `par`. If `setMargins` is `TRUE` and `marDendro` is not given, the function will provide reasonable default values.

`colorLabels` logical: should module eigengene names be interpreted as color names and the colors used to label heatmap plots and barplots?

`signed` logical: should eigengene networks be constructed as signed?

`heatmapColors` color palette for heatmaps. Defaults to `heat.colors` when `signed` is `FALSE`, and to `redWhiteGreen` when `signed` is `TRUE`.

`plotAdjacency` logical: should module eigengene heatmaps plot adjacency (ranging from 0 to 1), or correlation (ranging from -1 to 1)?

`printAdjacency` logical: should the numerical values be printed into the adjacency or correlation heatmap?

`cex.adjacency` character expansion factor for printing of numerical values into the adjacency or correlation heatmap

`coloredBarplot` logical: should the barplot of eigengene adjacency preservation distinguish individual contributions by color? This is possible only if `colorLabels` is `TRUE` and module eigengene names encode valid colors.

`barplotMeans` logical: plot mean preservation in the barplot? This option effectively rescales the preservation by the number of eigengenes in the network. If means are plotted, the barplot is not colored.

`barplotErrors` logical: should standard errors of the mean preservation be plotted?

`plotPreservation` a character string specifying which type of preservation measure to plot. Allowed values are (unique abbreviations of) `"standard"`, `"hyperbolic"`, `"both"`.

`zlimPreservation` a vector of length 2 giving the value limits for the preservation heatmaps.

`printPreservation` logical: should preservation values be printed within the heatmap?

`cex.preservation` character expansion factor for preservation display.

`...` other graphical arguments to function `link[fields]{image.plot}`.

Details

Consensus eigengene networks consist of a fixed set of eigengenes "expressed" in several different sets. Network connection strengths are given by eigengene correlations. This function aims to visualize the networks as well as their similarities and differences across sets.

The function partitions the screen appropriately and plots eigengene dendrograms in the top row, then a square matrix of plots: heatmap plots of eigengene networks in each set on the diagonal, heatmap plots of pairwise preservation networks below the diagonal, and barplots of aggregate network preservation of individual eigengenes above the diagonal. A preservation plot or barplot in the row i and column j of the square matrix represents the preservation between sets i and j.

Individual eigengenes are labeled by their name in the dendrograms; in the heatmaps and barplots they can optionally be labeled by color squares. For compatibility with other functions, the color labels are encoded in the eigengene names by prefixing the color with two letters, such as "MEturquoise".

Two types of network preservation can be plotted: the "standard" is simply the difference between adjacencies in the two compared sets. The "hyperbolic" difference de-emphasizes the preservation of low adjacencies. When "both" is specified, standard preservation is plotted in the lower triangle and hyperbolic in the upper triangle of each preservation heatmap.

If the eigengenes are labeled by color, the bars in the barplot can be split into segments representing the contribution of each eigengene and labeled by the contribution. For example, a yellow segment in a bar labeled by a turquoise square represents the preservation of the adjacency between the yellow and turquoise eigengenes in the two networks compared by the barplot.

For large numbers of eigengenes and/or sets, it may be difficult to get a meaningful plot fit a standard computer screen. In such cases we recommend using a device such as postscript or pdf where the user can specify large dimensions; such plots can be conveniently viewed in standard pdf or postscript viewers.

Value

None.

Author(s)

Peter Langfelder

References

For theory and applications of consensus eigengene networks, see

Langfelder P, Horvath S (2007) Eigengene networks for studying the relationships between co-expression modules. BMC Systems Biology 2007, 1:54

See Also

labeledHeatmap, labeledBarplot for annotated heatmaps and barplots;

hclust for hierarchical clustering and dendrogram plots

---

plotMEpairs *Pairwise scatterplots of eigengenes*

Description

The function produces a matrix of plots containing pairwise scatterplots of given eigengenes, the distribution of their values and their pairwise correlations.

Usage

```
plotMEpairs(
   datME,
   y = NULL,
   main = "Relationship between module eigengenes",
   clusterMEs = TRUE,
   ...)
```

Arguments

datME	a data frame containing expression data, with rows corresponding to samples and columns to genes. Missing values are allowed and will be ignored.
y	optional microarray sample trait vector. Will be treated as an additional eigengene.
main	main title for the plot.
clusterMEs	logical: should the module eigengenes be ordered by their dendrogram?
...	additional graphical parameters to the function pairs

Details

The function produces an NxN matrix of plots, where N is the number of eigengenes. In the upper traingle it plots pairwise scatterplots of module eigengenes (plus the trait y, if given). On the diagonal it plots histograms of sample values for each eigengene. Below the diagonal, it displays the pairwise correlations of the eigengenes.

Value

None.

Author(s)

Steve Horvath

See Also

pairs

---

plotModuleSignificance

Barplot of module significance

Description

Plot a barplot of gene significance.

Usage

```
plotModuleSignificance(
  geneSignificance,
  colors,
  boxplot = FALSE,
  main = "Gene significance across modules,",
  ylab = "Gene Significance", ...)
```

Arguments

geneSignificance	a numeric vector giving gene significances.
colors	a character vector specifying module assignment for the genes whose significance is given in geneSignificance . The modules should be labeled by colors.
boxplot	logical: should a boxplot be produced instead of a barplot?
main	main title for the plot.
ylab	y axis label for the plot.
...	other graphical parameters to plot.

Details

Given individual gene significances and their module assigment, the function calculates the module significance for each module as the average gene significance of the genes within the module. The result is plotted in a barplot or boxplot form. Each bar or box is labeled by the corresponding module color.

Value

None.

Author(s)

Steve Horvath

References

Bin Zhang and Steve Horvath (2005) "A General Framework for Weighted Gene Co-Expression Network Analysis", Statistical Applications in Genetics and Molecular Biology: Vol. 4: No. 1, Article 17

Dong J, Horvath S (2007) Understanding Network Concepts in Modules, BMC Systems Biology 2007, 1:24

See Also

barplot, boxplot

---

plotNetworkHeatmap *Network heatmap plot*

Description

Network heatmap plot.

Usage

```
plotNetworkHeatmap(
  datExpr,
  plotGenes,
  useTOM = TRUE,
  power = 6,
  networkType = "unsigned",
  main = "Heatmap of the network")
```

Arguments

datExpr	a data frame containing expression data, with rows corresponding to samples and columns to genes. Missing values are allowed and will be ignored.
plotGenes	a character vector giving the names of genes to be included in the plot. The names will be matched against names(datExpr).
useTOM	logical: should TOM be plotted (TRUE), or correlation-based adjacency (FALSE)?
power	soft-thresholding power for network construction.
networkType	a character string giving the newtork type. Recognized values are (unique abbreviations of) "unsigned", "signed", and "signed hybrid".
main	main title for the plot.

Details

The function constructs a network from the given expression data (selected by plotGenes) using the soft-thresholding procedure, optionally calculates Topological Overlap (TOM) and plots a heatmap of the network.

Note that all network calculations are done in one block and may fail due to memory allocation issues for large numbers of genes.

Value

None.

Author(s)

Steve Horvath

References

Bin Zhang and Steve Horvath (2005) "A General Framework for Weighted Gene Co-Expression Network Analysis", Statistical Applications in Genetics and Molecular Biology: Vol. 4: No. 1, Article 17

See Also

adjacency, TOMsimilarity

`preservationNetworkConnectivity`
Network preservation calculations

Description

This function calculates several measures of gene network preservation. Given gene expression data in several individual data sets, it calculates the individual adjacency matrices, forms the preservation network and finally forms several summary measures of adjacency preservation for each node (gene) in the network.

Usage

```
preservationNetworkConnectivity(
   multiExpr,
   useSets = NULL, useGenes = NULL,
   corFnc = "cor", corOptions = "use='p'",
   networkType = "unsigned",
   power = 6,
   sampleLinks = NULL, nLinks = 5000,
   blockSize = 1000,
   setSeed = 12345,
   weightPower = 2,
   verbose = 2, indent = 0)
```

Arguments

`multiExpr`	expression data in the multi-set format (see `checkSets`). A vector of lists, one per set. Each set must contain a component `data` that contains the expression data, with rows corresponding to samples and columns to genes or probes.
`useSets`	optional specification of sets to be used for the preservation calculation. Defaults to using all sets.
`useGenes`	optional specification of genes to be used for the preservation calculation. Defaults to all genes.
`corFnc`	character string containing the name of the function to calculate correlation. Suggested functions include `"cor"` and `"bicor"`.
`corOptions`	further argument to the correlation function.
`networkType`	a character string encoding network type. Recognized values are (unique abbreviations of) `"unsigned"`, `"signed"`, and `"signed hybrid"`.
`power`	soft thresholding power for network construction. Should be a number greater than 1.
`sampleLinks`	logical: should network connections be sampled (`TRUE`) or should all connections be used systematically (`FALSE`)?
`nLinks`	number of links to be sampled. Should be set such that `nLinks * nNeighbors` be several times larger than the number of genes.
`blockSize`	correlation calculations will be split into square blocks of this size, to prevent running out of memory for large gene sets.

`setSeed` seed to be used for sampling, for repeatability. If a seed already exists, it is saved before the sampling starts and restored upon exit.

`weightPower` power with which higher adjacencies will be weighted in weighted means

`verbose` integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.

`indent` indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

The preservation network is formed from adjacencies of compared sets. For 'complete' preservations, all given sets are compared at once; for 'pairwise' preservations, the sets are compared in pairs. Unweighted preservations are simple mean preservations for each node; their weighted counterparts are weighted averages in which a preservation of adjacencies $A_{ij}^{(1)}$ and $A_{ij}^{(2)}$ of nodes i, j between sets 1 and 2 is weighted by $[(A_{ij}^{(1)} + A_{ij}^{(2)})/2]^{w}eightPower$. The hyperbolic preservation is based on $tanh[(max - min)/(max + min)^2]$, where max and min are the componentwise maximum and minimum of the compared adjacencies, respectively.

Value

A list with the following components:

`pairwise` a matrix with rows corresponding to genes and columns to unique pairs of given sets, giving the pairwise preservation of the adjacencies connecting the gene to all other genes.

`complete` a vector with one entry for each input gene containing the complete mean preservation of the adjacencies connecting the gene to all other genes.

`pairwiseWeighted`
a matrix with rows corresponding to genes and columns to unique pairs of given sets, giving the pairwise weighted preservation of the adjacencies connecting the gene to all other genes.

`completeWeighted`
a vector with one entry for each input gene containing the complete weighted mean preservation of the adjacencies connecting the gene to all other genes.

`pairwiseHyperbolic`
a matrix with rows corresponding to genes and columns to unique pairs of given sets, giving the pairwise hyperbolic preservation of the adjacencies connecting the gene to all other genes.

`completeHyperbolic`
a vector with one entry for each input gene containing the complete mean hyperbolic preservation of the adjacencies connecting the gene to all other genes.

`pairwiseWeightedHyperbolic`
a matrix with rows corresponding to genes and columns to unique pairs of given sets, giving the pairwise weighted hyperbolic preservation of the adjacencies connecting the gene to all other genes.

`completeWeightedHyperbolic`
a vector with one entry for each input gene containing the complete weighted hyperbolic mean preservation of the adjacencies connecting the gene to all other genes.

Author(s)

Peter Langfelder

References

Langfelder P, Horvath S (2007) Eigengene networks for studying the relationships between co-expression modules. BMC Systems Biology 2007, 1:54

See Also

`adjacency` for calculation of adjacency;

---

projectiveKMeans *Projective K-means (pre-)clustering of expression data*

Description

Implementation of a variant of K-means clustering for expression data.

Usage

```
projectiveKMeans(
  datExpr,
  preferredSize = 5000,
  nCenters = as.integer(min(ncol(datExpr)/20, preferredSize^2/ncol(datExpr))),
  sizePenaltyPower = 4,
  networkType = "unsigned",
  randomSeed = 54321,
  checkData = TRUE,
  maxIterations = 1000,
  verbose = 0, indent = 0)
```

Arguments

`datExpr`	expression data. A data frame in which columns are genes and rows ar samples. NAs are allowed, but not too many.
`preferredSize`	preferred maximum size of clusters.
`nCenters`	number of initial clusters. Empirical evidence suggests that more centers will give a better preclustering; the default is an attempt to arrive at a reasonable number.
`sizePenaltyPower`	parameter specifying how severe is the penalty for clusters that exceed `preferredSize`.
`networkType`	network type. Allowed values are (unique abbreviations of) `"unsigned"`, `"signed"`, `"signed hybrid"`. See `adjacency`.
`randomSeed`	integer to be used as seed for the random number generator before the function starts. If a current seed exists, it is saved and restored upon exit.
`checkData`	logical: should data be checked for genes with zero variance and genes and samples with excessive numbers of missing samples? Bad samples are ignored; returned cluster assignment for bad genes will be `NA`.

`maxIterations`
: maximum iterations to be attempted.

`verbose`
: integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.

`indent`
: indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

The principal aim of this function within WGCNA is to pre-cluster a large number of genes into smaller blocks that can be handled using standard WGCNA techniques.

This function implements a variant of K-means clustering that is suitable for co-expression analysis. Cluster centers are defined by the first principal component, and distances by correlation (more precisely, 1-correlation). The distance between a gene and a cluster is multiplied by a factor of $max(clusterSize/preferredSize, 1)^{sizePenaltyPower}$, thus penalizing clusters whose size exceeds `preferredSize`. The function starts with randomly generated cluster assignment (hence the need to set the random seed for repeatability) and executes interations of calculating new centers and reassigning genes to nearest center until the clustering becomes stable. Before returning, nearby clusters are iteratively combined if their combined size is below `preferredSize`.

The standard principal component calculation via the function `svd` fails from time to time (likely a convergence problem of the underlying lapack functions). Such errors are trapped and the principal component is approximated by a weighted average of expression profiles in the cluster. If `verbose` is set above 2, an informational message is printed whenever this approximation is used.

Value

A list with the following components:

`clusters`
: a numerical vector with one component per input gene, giving the cluster number in which the gene is assigned.

`centers`
: cluster centers, that is their first principal components.

Author(s)

Peter Langfelder

`propVarExplained` *Proportion of variance explained by eigengenes.*

Description

This function calculates the proportion of variance of genes in each module explained by the respective module eigengene.

Usage

```
propVarExplained(datExpr, colors, MEs, corFnc = "cor", corOptions = "use = 'p'")
```

Arguments

datExpr	expression data. A data frame in which columns are genes and rows ar samples. NAs are allowed and will be ignored.
colors	a vector giving module assignment for genes given in datExpr. Unique values should correspond to the names of the eigengenes in MEs.
MEs	a data frame of module eigengenes in which each column is an eigengene and each row corresponds to a sample.
corFnc	character string containing the name of the function to calculate correlation. Suggested functions include "cor" and "bicor".
corOptions	further argument to the correlation function.

Details

For compatibility with other functions, entries in color are matched to a substring of names(MEs) starting at position 3. For example, the entry "turquoise" in colors will be matched to the eigengene named "MEturquoise". The first two characters of the eigengene name are ignored and can be arbitrary.

Value

A vector with one entry per eigengene containing the proportion of variance of the module explained by the eigengene.

Author(s)

Peter Langfelder

See Also

moduleEigengenes

---

randIndex *~~function to do ... ~~*

Description

~~ A concise (1-5 lines) description of what the function does. ~~

Usage

```
randIndex(tab, adjust = TRUE)
```

Arguments

tab	~~Describe tab here~~
adjust	~~Describe adjust here~~

Details

~~ If necessary, more details than the description above ~~

Value

~Describe the value returned If it is a LIST, use

comp1	Description of 'comp1'
comp2	Description of 'comp2'

...

Note

~~further notes~~

Author(s)

~~who you are~~

References

~put references to the literature/web site here ~

See Also

~~objects to See Also as help, ~~~

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.
```

---

recutBlockwiseTrees

Repeat blockwise module detection from pre-calculated data

Description

Given consensus networks constructed for example using blockwiseModules, this function (re-)detects modules in them by branch cutting of the corresponding dendrograms. If repeated branch cuts of the same gene network dendrograms are desired, this function can save substantial time by re-using already calculated networks and dendrograms.

Usage

```
recutBlockwiseTrees(
  datExpr,
  goodSamples, goodGenes,
  blocks,
  TOMFiles,
  dendrograms,
  corType = "pearson",
  networkType = "unsigned",
  deepSplit = 2,
```

```
  detectCutHeight = 0.995, minModuleSize = min(20, ncol(datExpr)/2 ),
  maxCoreScatter = NULL, minGap = NULL,
  maxAbsCoreScatter = NULL, minAbsGap = NULL,
  pamStage = TRUE, pamRespectsDendro = TRUE,
  minKMEtoJoin =0.7,
  minCoreKME = 0.5, minCoreKMESize = minModuleSize/3,
  minKMEtoStay = 0.3,
  reassignThreshold = 1e-6,
  mergeCutHeight = 0.15, impute = TRUE,
  trapErrors = FALSE, numericLabels = FALSE,
  verbose = 0, indent = 0)
```

Arguments

`datExpr`	expression data. A data frame in which columns are genes and rows ar samples. NAs are allowed, but not too many.
`goodSamples`	a logical vector specifying which samples are considered "good" for the analysis. See `goodSamplesGenes`.
`goodGenes`	a logical vector with length equal number of genes in `multiExpr` that specifies which genes are considered "good" for the analysis. See `goodSamplesGenes`.
`blocks`	specification of blocks in which hierarchical clustering and module detection should be performed. A numeric vector with one entry per gene of `multiExpr` giving the number of the block to which the corresponding gene belongs.
`TOMFiles`	a vector of character strings specifying file names in which the block-wise topological overlaps are saved.
`dendrograms`	a list of length equal the number of blocks, in which each component is a hierarchical clustering dendrograms of the genes that belong to the block.
`corType`	character string specifying the correlation to be used. Allowed values are (unique abbreviations of) `"pearson"` and `"bicor"`, corresponding to Pearson and bidweight midcorrelation, respectively. Missing values are handled using the `pariwise.complete.obs` option.
`networkType`	network type. Allowed values are (unique abbreviations of) `"unsigned"`, `"signed"`, `"signed hybrid"`. See `adjacency`.
`deepSplit`	integer value between 0 and 4. Provides a simplified control over how sensitive module detection should be to module splitting, with 0 least and 4 most sensitive. See `cutreeDynamic` for more details.
`detectCutHeight`	dendrogram cut height for module detection. See `cutreeDynamic` for more details.
`minModuleSize`	minimum module size for module detection. See `cutreeDynamic` for more details.
`maxCoreScatter`	maximum scatter of the core for a branch to be a cluster, given as the fraction of `cutHeight` relative to the 5th percentile of joining heights. See `cutreeDynamic` for more details.
`minGap`	minimum cluster gap given as the fraction of the difference between `cutHeight` and the 5th percentile of joining heights. See `cutreeDynamic` for more details.

maxAbsCoreScatter
: maximum scatter of the core for a branch to be a cluster given as absolute heights. If given, overrides maxCoreScatter. See cutreeDynamic for more details.

minAbsGap
: minimum cluster gap given as absolute height difference. If given, overrides minGap. See cutreeDynamic for more details.

pamStage
: logical. If TRUE, the second (PAM-like) stage of module detection will be performed. See cutreeDynamic for more details.

pamRespectsDendro
: Logical, only used when pamStage is TRUE. If TRUE, the PAM stage will respect the dendrogram in the sense an object can be PAM-assigned only to clusters that lie below it on the branch that the object is merged into. See cutreeDynamic for more details.

minKMEtoJoin
: a number between 0 and 1. Genes with eigengene connectivity higher than minKMEtoJoin are automatically assigned to their closest module.

minCoreKME
: a number between 0 and 1. If a detected module does not have at least minModuleKMESize genes with eigengene connectivity at least minCoreKME, the module is disbanded (its genes are unlabeled and returned to the pool of genes waiting for mofule detection).

minCoreKMESize
: see minCoreKME above.

minKMEtoStay
: genes whose eigengene connectivity to their module eigengene is lower than minKMEtoStay are removed from the module.

reassignThreshold
: p-value ratio threshold for reassigning genes between modules. See Details.

mergeCutHeight
: dendrogram cut height for module merging.

impute
: logical: should imputation be used for module eigengene calculation? See moduleEigengenes for more details.

trapErrors
: logical: should errors in calculations be trapped?

numericLabels
: logical: should the returned modules be labeled by colors (FALSE), or by numbers (TRUE)?

verbose
: integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.

indent
: indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

For details on blockwise module detection, see blockwiseModules. This function implements the module detection subset of the functionality of blockwiseModules; network construction and clustering must be performed in advance. The primary use of this function is to experiment with module detection settings without having to re-execute long network and clustering calculations whose results are not affected by the cutting parameters.

This function takes as input the networks and dendrograms that are produced by blockwiseModules. Working block by block, modules are identified in the dendrogram by the Dynamic Hybrid Tree Cut algorithm. Found modules are trimmed of genes whose correlation with module eigengene (KME) is less than minKMEtoStay. Modules in which fewer than minCoreKMESize genes have KME

higher than minCoreKME are disbanded, i.e., their constituent genes are pronounced unassigned. Conversely, any unassigned genes with KME higher than minKMEtoJoin are automatically assigned to their nearest module.

After all blocks have been processed, the function checks whether there are genes whose KME in the module they assigned is lower than KME to another module. If p-values of the higher correlations are smaller than those of the native module by the factor reassignThresholdPS, the gene is re-assigned to the closer module.

In the last step, modules whose eigengenes are highly correlated are merged. This is achieved by clustering module eigengenes using the dissimilarity given by one minus their correlation, cutting the dendrogram at the height mergeCutHeight and merging all modules on each branch. The process is iterated until no modules are merged. See mergeCloseModules for more details on module merging.

Value

A list with the following components:

colors	a vector of color or numeric module labels for all genes.
unmergedColors	a vector of color or numeric module labels for all genes before module merging.
MEs	a data frame containing module eigengenes of the found modules (given by colors).
MEsOK	logical indicating whether the module eigengenes were calculated without errors.

Author(s)

Peter Langfelder

References

Bin Zhang and Steve Horvath (2005) "A General Framework for Weighted Gene Co-Expression Network Analysis", Statistical Applications in Genetics and Molecular Biology: Vol. 4: No. 1, Article 17

See Also

blockwiseModules for full module calculation;

cutreeDynamic for adaptive branch cutting in hierarchical clustering dendrograms;

mergeCloseModules for merging of close modules.

---

recutConsensusTrees *Repeat blockwise consensus module detection from pre-calculated data*

Description

Given consensus networks constructed for example using `blockwiseConsensusModules`, this function (re-)detects modules in them by branch cutting of the corresponding dendrograms. If repeated branch cuts of the same gene network dendrograms are desired, this function can save substantial time by re-using already calculated networks and dendrograms.

Usage

```
recutConsensusTrees(
  multiExpr,
  goodSamples, goodGenes,
  blocks,
  TOMFiles,
  dendrograms,
  corType = "pearson",
  networkType = "unsigned",
  deepSplit = 2,
  detectCutHeight = 0.995, minModuleSize = 20,
  checkMinModuleSize = TRUE,
  maxCoreScatter = NULL, minGap = NULL,
  maxAbsCoreScatter = NULL, minAbsGap = NULL,
  pamStage = TRUE, pamRespectsDendro = TRUE,
  minKMEtoJoin =0.7,
  minCoreKME = 0.5, minCoreKMESize = minModuleSize/3,
  minKMEtoStay = 0.2,
  reassignThresholdPS = 1e-4,
  mergeCutHeight = 0.15,
  impute = TRUE,
  trapErrors = FALSE,
  numericLabels = FALSE,
  verbose = 2, indent = 0)
```

Arguments

`multiExpr`	expression data in the multi-set format (see `checkSets`). A vector of lists, one per set. Each set must contain a component `data` that contains the expression data, with rows corresponding to samples and columns to genes or probes.
`goodSamples`	a list with one component per set. Each component is a logical vector specifying which samples are considered "good" for the analysis. See `goodSamplesGenesMS`.
`goodGenes`	a logical vector with length equal number of genes in `multiExpr` that specifies which genes are considered "good" for the analysis. See `goodSamplesGenesMS`.
`blocks`	specification of blocks in which hierarchical clustering and module detection should be performed. A numeric vector with one entry per gene of `multiExpr` giving the number of the block to which the corresponding gene belongs.
`TOMFiles`	a vector of character strings specifying file names in which the block-wise topological overlaps are saved.
`dendrograms`	a list of length equal the number of blocks, in which each component is a hierarchical clustering dendrograms of the genes that belong to the block.
`corType`	character string specifying the correlation to be used. Allowed values are (unique abbreviations of) `"pearson"` and `"bicor"`, corresponding to Pearson and

bidweight midcorrelation, respectively. Missing values are handled using the `pariwise.complete.obs` option.

`networkType` network type. Allowed values are (unique abbreviations of) `"unsigned"`, `"signed"`, `"signed hybrid"`. See `adjacency`. Note that while no new networks are computed in this function, this parameter affects the interpretation of correlations in this function.

`deepSplit` integer value between 0 and 4. Provides a simplified control over how sensitive module detection should be to module splitting, with 0 least and 4 most sensitive. See `cutreeDynamic` for more details.

`detectCutHeight`
dendrogram cut height for module detection. See `cutreeDynamic` for more details.

`minModuleSize`
minimum module size for module detection. See `cutreeDynamic` for more details.

`checkMinModuleSize`
logical: should sanity checks be performed on `minModuleSize`?

`maxCoreScatter`
maximum scatter of the core for a branch to be a cluster, given as the fraction of `cutHeight` relative to the 5th percentile of joining heights. See `cutreeDynamic` for more details.

`minGap` minimum cluster gap given as the fraction of the difference between `cutHeight` and the 5th percentile of joining heights. See `cutreeDynamic` for more details.

`maxAbsCoreScatter`
maximum scatter of the core for a branch to be a cluster given as absolute heights. If given, overrides `maxCoreScatter`. See `cutreeDynamic` for more details.

`minAbsGap` minimum cluster gap given as absolute height difference. If given, overrides `minGap`. See `cutreeDynamic` for more details.

`pamStage` logical. If TRUE, the second (PAM-like) stage of module detection will be performed. See `cutreeDynamic` for more details.

`pamRespectsDendro`
Logical, only used when `pamStage` is `TRUE`. If `TRUE`, the PAM stage will respect the dendrogram in the sense an object can be PAM-assigned only to clusters that lie below it on the branch that the object is merged into. See `cutreeDynamic` for more details.

`minKMEtoJoin` a number between 0 and 1. Genes with eigengene connectivity higher than `minKMEtoJoin` are automatically assigned to their closest module.

`minCoreKME` a number between 0 and 1. If a detected module does not have at least `minModuleKMESize` genes with eigengene connectivity at least `minCoreKME`, the module is disbanded (its genes are unlabeled and returned to the pool of genes waiting for mofule detection).

`minCoreKMESize`
see `minCoreKME` above.

`minKMEtoStay` genes whose eigengene connectivity to their module eigengene is lower than `minKMEtoStay` are removed from the module.

`reassignThresholdPS`
per-set p-value ratio threshold for reassigning genes between modules. See Details.

mergeCutHeight
: dendrogram cut height for module merging.

impute
: logical: should imputation be used for module eigengene calculation? See moduleEigengenes for more details.

trapErrors
: logical: should errors in calculations be trapped?

numericLabels
: logical: should the returned modules be labeled by colors (FALSE), or by numbers (TRUE)?

verbose
: integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.

indent
: indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

For details on blockwise consensus module detection, see blockwiseConsensusModules. This function implements the module detection subset of the functionality of blockwiseConsensusModules; network construction and clustering must be performed in advance. The primary use of this function is to experiment with module detection settings without having to re-execute long network and clustering calculations whose results are not affected by the cutting parameters.

This function takes as input the networks and dendrograms that are produced by blockwiseConsensusModules. Working block by block, modules are identified in the dendrograms by the Dynamic Hybrid tree cut. Found modules are trimmed of genes whose correlation with module eigengene (KME) is less than minKMEtoStay in any of the sets. Modules in which fewer than minCoreKMESize genes have KME higher than minCoreKME (in all sets) are disbanded, i.e., their constituent genes are pronounced unassigned. Conversely, any unassigned genes with KME higher than minKMEtoJoin in all sets are automatically assigned to their nearest module.

After all blocks have been processed, the function checks whether there are genes whose KME in the module they assigned is lower than KME to another module. If p-values of the higher correlations are smaller than those of the native module by the factor reassignThresholdPS (in every set), the gene is re-assigned to the closer module.

In the last step, modules whose eigengenes are highly correlated are merged. This is achieved by clustering module eigengenes using the dissimilarity given by one minus their correlation, cutting the dendrogram at the height mergeCutHeight and merging all modules on each branch. The process is iterated until no modules are merged. See mergeCloseModules for more details on module merging.

Value

A list with the following components:

colors
: module assignment of all input genes. A vector containing either character strings with module colors (if input numericLabels was unset) or numeric module labels (if numericLabels was set to TRUE). The color "grey" and the numeric label 0 are reserved for unassigned genes.

unmergedColors
: module colors or numeric labels before the module merging step.

multiMEs
: module eigengenes corresponding to the modules returned in colors, in multiset format. A vector of lists, one per set, containing eigengenes, proportion of variance explained and other information. See multiSetMEs for a detailed description.

Note

Basic sanity checks are performed on given arguments, but it is left to the user's responsibility to provide valid input.

Author(s)

Peter Langfelder

References

Langfelder P, Horvath S (2007) Eigengene networks for studying the relationships between co-expression modules. BMC Systems Biology 2007, 1:54

See Also

`blockwiseConsensusModules` for the full blockwise modules calculation. Parts of its output are natural input for this function.

`cutreeDynamic` for adaptive branch cutting in hierarchical clustering dendrograms;

`mergeCloseModules` for merging of close modules.

`redWhiteGreen` *Red-white-green color sequence*

Description

Generate a red-white-green color sequence of a given length.

Usage

```
redWhiteGreen(n, gamma = 1)
```

Arguments

`n`	number of colors to be returned
`gamma`	color correction power

Details

The function returns a color vector that starts with pure green, gradually turns into white and then to red. The power `gamma` can be used to control the behaviour of the quarter- and three quarter-values (between red and white, and white and green, respectively). Higher powers will make the mid-colors more white, while lower powers will make the colors more saturated, respectively.

Value

A vector of colors of length `n`.

Author(s)

Peter Langfelder

Examples

```
par(mfrow = c(3, 1))
displayColors(redWhiteGreen(50));
displayColors(redWhiteGreen(50, 3));
displayColors(redWhiteGreen(50, 0.5));
```

---

relativeCorPredictionSuccess

~~function to do ... ~~

Description

~~ A concise (1-5 lines) description of what the function does. ~~

Usage

```
relativeCorPredictionSuccess(corPredictionNew, corPredictionStandard, corTestSet
```

Arguments

`corPredictionNew`	~~Describe `corPredictionNew` here~~
`corPredictionStandard`	~~Describe `corPredictionStandard` here~~
`corTestSet`	~~Describe `corTestSet` here~~
`topNumber`	~~Describe `topNumber` here~~

Details

~~ If necessary, more details than the description above ~~

Value

~Describe the value returned If it is a LIST, use

`comp1`	Description of 'comp1'
`comp2`	Description of 'comp2'

...

Note

~~further notes~~

Author(s)

~~who you are~~

References

~put references to the literature/web site here ~

See Also

~~objects to See Also as help, ~~~

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.
```

`removeGreyME` *Removes the grey eigengene from a given collection of eigengenes.*

Description

Given module eigengenes either in a single data frame or in a multi-set format, removes the grey eigengenes from each set. If the grey eigengenes are not found, a warning is issued.

Usage

```
removeGreyME(MEs, greyMEName = paste(moduleColor.getMEprefix(), "grey", sep=""))
```

Arguments

`MEs`	Module eigengenes, either in a single data frame (typicaly for a single set), or in a multi-set format. See `checkSets` for a description of the multi-set format.
`greyMEName`	Name of the module eigengene (in each corresponding data frame) that corresponds to the grey color. This will typically be "PCgrey" or "MEgrey". If the module eigengenes were calculated using standard functions in this library, the default should work.

Value

Module eigengenes in the same format as input (either a single data frame or a vector of lists) with the grey eigengene removed.

Author(s)

Peter Langfelder, `<Peter.Langfelder@gmail.com>`

rgcolors.func *Red and Green Color Specification*

Description

This function creates a vector of n "contiguous" colors, corresponding to n intensities (between 0 and 1) of the red, green and blue primaries, with the blue intensities set to zero. The values returned by `rgcolors.func` can be used with a `col=` specification in graphics functions or in `par`.

Usage

```
rgcolors.func(n=50)
```

Arguments

`n`	the number of colors (>= 1) to be used in the red and green palette.

Value

a character vector of color names. Colors are specified directly in terms of their RGB components with a string of the form "\#RRGGBB", where each of the pairs RR, GG, BB consist of two hexadecimal digits giving a value in the range 00 to FF.

Author(s)

Sandrine Dudoit, <sandrine@stat.berkeley.edu>
Jane Fridlyand, <janef@stat.berkeley.edu>

See Also

`plot.cor`, `plot.mat`, `colors`, `rgb`, `image`.

Examples

```
rgcolors.func(n=5)
## The following vector is returned:
## "#00FF00" "#40BF00" "#808000" "#BF4000" "#FF0000"
```

scaleFreeFitIndex *Calculation of fitting statistics for evaluating scale free topology fit.*

Description

The function scaleFreeFitIndex calculates several indices (fitting statistics) for evaluating scale free topology fit. The input is a vector (of connectivities) k. Next k is discretized into nBreaks number of equal-width bins. Let's denote the resulting vector dk. The relative frequency for each bin is denoted p.dk.

Usage

```
scaleFreeFitIndex(k, nBreaks = 10, removeFirst = FALSE)
```

Arguments

k	numeric vector whose components contain non-negative values
nBreaks	positive integer. This determines the number of equal width bins.
removeFirst	logical. If TRUE then the first bin will be removed.

Value

Data frame with columns

Rsquared.SFT	the model fitting index (R.squared) from the following model lm(log.p.dk ~ log.dk)
slope.SFT	the slope estimate from model lm(log(p(k))~log(k))
truncatedExponentialAdjRsquared	the adjusted R.squared measure from the truncated exponential model given by lm2 = lm(log.p.dk ~ log.dk + dk).
loglogAdjRsquared	the R-squared value resulting from the following model lm(log.p.dk ~ log.dk + log(log(1+log.dk)))

Author(s)

Steve Horvath

---

scaleFreePlot *Visual check of scale-free topology*

Description

A simple visula check of scale-free network ropology.

Usage

```
scaleFreePlot(connectivity, nBreaks = 10, truncated = FALSE, removeFirst = FALSE
```

Arguments

connectivity	vector containing network connectivities.
nBreaks	number of breaks in the connectivity dendrogram.
truncated	logical: should a truncated exponential fit be calculated and plotted in addition to the linear one?
removeFirst	logical: should the first bin be removed from the fit?
main	main title for the plot.
...	other graphical parameter to the plot function.

Details

The function plots a log-log plot of a histogram of the given connectivities, and fits a linear model plus optionally a truncated exponential model. The R^2 of the fit can be considered an index of the scale freedom of the network topology.

Value

None.

Author(s)

Steve Horvath

References

Bin Zhang and Steve Horvath (2005) "A General Framework for Weighted Gene Co-Expression Network Analysis", Statistical Applications in Genetics and Molecular Biology: Vol. 4: No. 1, Article 17

See Also

`softConnectivity` for connectivity calculation in weigheted networks.

---

setCorrelationPreservation

Summary correlation preservation measure

Description

Given consensus eigengenes, the function calculates the average correlation preservation pair-wise for all pairs of sets.

Usage

```
setCorrelationPreservation(multiME, setLabels, excludeGrey = TRUE, greyLabel = "
```

Arguments

`multiME`	consensus module eigengenes in a multi-set format. A vector of lists with one list corresponding to each set. Each list must contain a component `data` that is a data frame whose columns are consensus module eigengenes.
`setLabels`	names to be used for the sets represented in `multiME`.
`excludeGrey`	logical: exclude the 'grey' eigengene from preservation measure?
`greyLabel`	module label corresponding to the 'grey' module. Usually this will be the character string `"grey"` if the labels are colors, and the number 0 if the labels are numeric.
`method`	character string giving the correlation preservation measure to use. Recognized values are (unique abbreviations of) `"absolute"`, `"hyperbolic"`.

Details

For each pair of sets, the function calculates the average preservation of correlation among the eigengenes. Two preservation measures are available, the abosolute preservation (high if the two correlations are similar and low if they are different), and the hyperbolically scaled preservation, which de-emphasizes preservation of low correlation values.

Value

A data frame with each row and column corresponding to a set given in `multiME`, containing the pairwise average correlation preservation values. Names and rownames are set to entries of `setLabels`.

Author(s)

Peter Langfelder

References

Langfelder P, Horvath S (2007) Eigengene networks for studying the relationships between co-expression modules. BMC Systems Biology 2007, 1:54

See Also

`multiSetMEs` for module eigengene calculation;

`plotEigengeneNetworks` for eigengene network visualization.

`sigmoidAdjacencyFunction` *Sigmoid-type adacency function.*

Description

Sigmoid-type function that converts a similarity to a weighted network adjacency.

Usage

```
sigmoidAdjacencyFunction(ss, mu = 0.8, alpha = 20)
```

Arguments

`ss`	similarity, a number between 0 and 1. Can be given as a scalar, vector or a matrix.
`mu`	shift parameter.
`alpha`	slope parameter.

Details

The sigmoid adjacency function is defined as $1/(1 + \exp[-\alpha(ss - \mu)])$.

Value

Adjacencies returned in the same form as the input `ss`

Author(s)

Steve Horvath

References

Bin Zhang and Steve Horvath (2005) "A General Framework for Weighted Gene Co-Expression Network Analysis", Statistical Applications in Genetics and Molecular Biology: Vol. 4: No. 1, Article 17

`signedKME` *Signed eigengene-based connectivity*

Description

Calculation of (signed) eigengene-based connectivity, also known as module membership.

Usage

```
signedKME(datExpr, datME, outputColumnName = "kME")
```

Arguments

`datExpr`	a data frame containing the gene expression data. Rows correspond to samples and columns to genes. Missing values are allowed and will be ignored.
`datME`	a data frame containing module eigengenes. Rows correspond to samples and columns to module eigengenes.
`outputColumnName`	a character string specifying the prefix of column names of the output.

Details

Signed eigengene-based connectivity of a gene in a module is defined as the correlation of the gene with the corresponding module eigengene. The samples in `datExpr` and `datME` must be the same.

Value

A data frame in which rows correspond to input genes and columns to module eigengenes, giving the signed eigengene-based connectivity of each gene with respect to each eigengene.

Author(s)

Steve Horvath

References

Dong J, Horvath S (2007) Understanding Network Concepts in Modules, BMC Systems Biology 2007, 1:24

Horvath S, Dong J (2008) Geometric Interpretation of Gene Coexpression Network Analysis. PLoS Comput Biol 4(8): e1000117

`signumAdjacencyFunction` *Hard-thresholding adjacency function*

Description

This function transforms correlations or other measures of similarity into an unweighted network adjacency.

Usage

```
signumAdjacencyFunction(corMat, threshold)
```

Arguments

`corMat`	a matrix of correlations or other measures of similarity.
`threshold`	threshold for connecting nodes: all nodes whose `corMat` is above the threshold will be connected in the resulting network.

Value

An unweighted adjacency matrix of the same dimensions as the input `corMat`.

Author(s)

Steve Horvath

References

Bin Zhang and Steve Horvath (2005) "A General Framework for Weighted Gene Co-Expression Network Analysis", Statistical Applications in Genetics and Molecular Biology: Vol. 4: No. 1, Article 17

See Also

`adjacency` for soft-thresholding and creating weighted networks.

`simulateDatExpr` *Simulation of expression data*

Description

Simulation of expression data with a customizable modular structure and several different types of noise.

Usage

```
simulateDatExpr(
  eigengenes,
  nGenes,
  modProportions,
  minCor = 0.3,
  maxCor = 1,
  corPower = 1,
  signed = FALSE,
  propNegativeCor = 0.3,
  backgroundNoise = 0.1,
  leaveOut = NULL,
  nSubmoduleLayers = 0,
  nScatteredModuleLayers = 0,
  averageNGenesInSubmodule = 10,
  averageExprInSubmodule = 0.2,
  submoduleSpacing = 2,
  verbose = 1, indent = 0)
```

Arguments

`eigengenes`	a data frame containing the seed eigengenes for the simulated modules. Rows correspond to samples and columns to modules.
`nGenes`	total number of genes to be simulated.
`modProportions`	a numeric vector with length equal the number of eigengenes in `eigengenes` plus one, containing fractions of the total number of genes to be put into each of the modules and into the "grey module", which means genes not related to any of the modules. See details.
`minCor`	minimum correlation of module genes with the corresponding eigengene. See details.
`maxCor`	maximum correlation of module genes with the corresponding eigengene. See details.
`corPower`	controls the dropoff of gene-eigengene correlation. See details.
`signed`	logical: should the genes be simulated as belonging to a signed network? If `TRUE`, all genes will be simulated to have positive correlation with the eigengene. If `FALSE`, a proportion given by `propNegativeCor` will be simulated with negative correlations of the same absolute values.
`propNegativeCor`	proportion of genes to be simulated with negative gene-eigengene correlations. Only effective if `signed` is `FALSE`.
`backgroundNoise`	amount of background noise to be added to the simulated expression data.
`leaveOut`	optional specification of modules that should be left out of the simulation, that is their genes will be simulated as unrelated ("grey"). This can be useful when simulating several sets, in some which a module is present while in others it is absent.
`nSubmoduleLayers`	number of layers of ordered submodules to be added. See details.

`nScatteredModuleLayers`
: number of layers of scattered submodules to be added. See details.

`averageNGenesInSubmodule`
: average number of genes in a submodule. See details.

`averageExprInSubmodule`
: average strength of submodule expression vectors.

`submoduleSpacing`
: a number giving submodule spacing: this multiple of the submodule size will lie between the submodule and the next one.

`verbose`
: integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.

`indent`
: indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

Given `eigengenes` can be unrelated or they can exhibit non-trivial correlations. Each module is simulated separately from others. The expression profiles are chosen such that their correlations with the eigengene run from just below `maxCor` to `minCor` (hence minCor must be between 0 and 1, not including the bounds). The parameter `corPower` can be chosen to control the behaviour of the simulated correlation with the gene index; values higher than 1 will result in the correlation approaching `minCor` faster and lower than 1 slower.

Numbers of genes in each module are specified (as fractions of the total number of genes `nGenes`) by `modProportions`. The last entry in `modProportions` corresponds to the genes that will be simulated as unrelated to anything else ("grey" genes). The proportion must add up to 1 or less. If the sum is less than one, the remaining genes will be partitioned into groups and simulated to be "close" to the proper modules, that is with small but non-zero correlations (between `minCor` and 0) with the module eigengene.

If `signed` is set `FALSE`, the correlation for some of the module genes is chosen negative (but the absolute values remain the same as they would be for positively correlated genes). To ensure consistency for simulations of multiple sets, the indices of the negatively correlated genes are fixed and distributed evenly.

In addition to the primary module structure, a secondary structure can be optionally simulated. Modules in the secondary structure have sizes chosen from an exponential distribution with mean equal `averageNGenesInSubmodule`. Expression vectors simulated in the secondary structure are simulated with expected standard deviation chosen from an exponential distribution with mean equal `averageExprInSubmodule`; the higher this coefficient, the more pronounced will the submodules be in the main modules. The secondary structure can be simulated in several layers; their number is given by `SubmoduleLayers`. Genes in these submodules are ordered in the same order as in the main modules.

In addition to the ordered submodule structure, a scattered submodule structure can be simulated as well. This structure can be viewed as noise that tends to correlate random groups of genes. The size and effect parameters are the same as for the ordered submodules, and the number of layers added is controlled by `nScatteredModuleLayers`.

Value

A list with the following components:

`datExpr`
: simulated expression data in a data frame whose columns correspond genes and rows to samples.

`setLabels`	simulated module assignment. Module labels are numeric, starting from 1. Genes simulated to be outside of proper modules have label 0. Modules that are left out (specified in `leaveOut`) are indicated as 0 here.
`allLabels`	simulated module assignment. Genes that belong to leftout modules (specified in `leaveOut`) are indicated by their would-be assignment here.
`labelOrder`	a vector specifying the order in which labels correspond to the given eigengenes, that is `labelOrder[1]` is the label assigned to module whose seed is `eigengenes[, 1]` etc.

Author(s)

Peter Langfelder

References

A short description of the simulation method can also be found in the Supplementary Material to the article

Langfelder P, Horvath S (2007) Eigengene networks for studying the relationships between co-expression modules. BMC Systems Biology 2007, 1:54.

The material is posted at http://www.genetics.ucla.edu/labs/horvath/CoexpressionNetwork/EigengeneNetwork/Suppleme

See Also

`simulateEigengeneNetwork` for a simulation of eigengenes with a given causal structure;

`simulateModule` for simulations of individual modules;

`simulateDatExpr5Modules` for a simplified interface to expression simulations;

`simulateMultiExpr` for a simulation of several related data sets.

---

simulateDatExpr5Modules

Simplified simulation of expression data

Description

This function provides a simplified interface to the expression data simulation, at the cost of considerably less flexibility.

Usage

```
simulateDatExpr5Modules(
  nGenes = 2000,
  colorLabels = c("turquoise", "blue", "brown", "yellow", "green"),
  simulateProportions = c(0.1, 0.08, 0.06, 0.04, 0.02),
  MEturquoise, MEblue, MEbrown, MEyellow, MEgreen,
  SDnoise = 1, backgroundCor = 0.3)
```

Arguments

nGenes	total number of genes to be simulated.
colorLabels	labels for simulated modules.
simulateProportions	a vector of length 5 giving proportions of the total number of genes to be placed in each individual module. The entries must be positive and sum to at most 1. If the sum is less than 1, the leftover genes will be simulated outside of modules.
MEturquoise	seed module eigengene for the first module.
MEblue	seed module eigengene for the second module.
MEbrown	seed module eigengene for the third module.
MEyellow	seed module eigengene for the fourth module.
MEgreen	seed module eigengene for the fifth module.
SDnoise	level of noise to be added to the simulated expressions.
backgroundCor	backgrond correlation. If non-zero, a component will be added to all genes such that the average correlation of otherwise unrelated genes will be backgroundCor.

Details

Roughly one-third of the genes are simulated with a negative correlation to their seed eigengene. See the functions simulateModule and simulateDatExpr for more details.

Value

A list with the following components:

datExpr	the simulated expression data in a data frame, with rows corresponding to samples and columns to genes.
truemodule	a vector with one entry per gene containing the simulated module membership.
datME	a data frame containing a copy of the input module eigengenes.

Author(s)

Steve Horvath and Peter Langfelder

See Also

simulateModule for simulation of individual modules;

simulateDatExpr for a more comprehensive data simulation interface.

simulateEigengeneNetwork
Simulate eigengene network from a causal model

Description

Simulates a set of eigengenes (vectors) from a given set of causal anchors and a causal matrix.

Usage

```
simulateEigengeneNetwork(causeMat, anchorIndex, anchorVectors, noise = 1, verbos
```

Arguments

causeMat	causal matrix. The entry [i, j] is the influence (path coefficient) of vector j on vector i.
anchorIndex	specifies the indices of the anchor vectors.
anchorVectors	a matrix giving the actual anchor vectors as columns. Their number must equal the length of anchorIndex.
noise	standard deviation of the noise added to each simulated vector.
verbose	level of verbosity. 0 means silent.
indent	indentation for diagnostic messages. Zero means no indentation; each unit adds two spaces.

Details

The algorithm starts with the anchor vectors and iteratively generates the rest from the path coefficients given in the matrix causeMat.

Value

A list with the following components:

eigengenes	generated eigengenes.
causeMat	a copy of the input causal matrix
levels	useful for debugging. A vector with one entry for each eigengene giving the number of generations of parents of the eigengene. Anchors have level 0, their direct causal children have level 1 etc.
anchorIndex	a copy of the input anchorIndex.

Author(s)

Peter Langfelder

simulateModule *Simulate a gene co-expression module*

Description

Simulation of a single gene co-expression module.

Usage

```
simulateModule(
  ME,
  nGenes,
  nNearGenes = 0,
  minCor = 0.3, maxCor = 1, corPower = 1,
  signed = FALSE, propNegativeCor = 0.3,
  verbose = 0, indent = 0)
```

Arguments

ME	seed module eigengene.
nGenes	number of genes in the module to be simulated. Must be non-zero.
nNearGenes	number of genes to be simulated with low correlation with the seed eigengene.
minCor	minimum correlation of module genes with the eigengene. See details.
maxCor	maximum correlation of module genes with the eigengene. See details.
corPower	controls the dropoff of gene-eigengene correlation. See details.
signed	logical: should the genes be simulated as belonging to a signed network? If TRUE, all genes will be simulated to have positive correlation with the eigengene. If FALSE, a proportion given by propNegativeCor will be simulated with negative correlations of the same absolute values.
propNegativeCor	proportion of genes to be simulated with negative gene-eigengene correlations. Only effective if signed is FALSE.
verbose	integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
indent	indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

Module genes are simulated around the eigengene by choosing them such that their (expected) correlations with the seed eigengene decrease progressively from (just below) maxCor to minCor. The genes are otherwise independent from one another. The variable corPower determines how fast the correlation drops towards minCor. Higher powers lead to a faster frop-off; corPower must be above zero but need not be integer.

If signed is FALSE, the genes are simulated so as to be part of an unsigned network module, that is some genes will be simulated with a negative correlation with the seed eigengene (but of the same absolute value that a positively correlated gene would be simulated with). The proportion of genes with negative correlation is controlled by propNegativeCor.

Optionally, the function can also simulate genes that are "near" the module, meaning they are simulated with a low but non-zero correlation with the seed eigengene. The correlations run between `minCor` and zero.

Value

A matrix containing the expression data with rows corresponding to samples and columns to genes.

Author(s)

Peter Langfelder

References

A short description of the simulation method can also be found in the Supplementary Material to the article

Langfelder P, Horvath S (2007) Eigengene networks for studying the relationships between co-expression modules. BMC Systems Biology 2007, 1:54.

The material is posted at http://www.genetics.ucla.edu/labs/horvath/CoexpressionNetwork/EigengeneNetwork/Supplem

See Also

`simulateEigengeneNetwork` for a simulation of eigengenes with a given causal structure;

`simulateDatExpr` for simulations of whole datasets consisting of multiple modules;

`simulateDatExpr5Modules` for a simplified interface to expression simulations;

`simulateMultiExpr` for a simulation of several related data sets.

`simulateMultiExpr` *Simulate multi-set expression data*

Description

Simulation of expression data in several sets with relate module structure.

Usage

```
simulateMultiExpr(eigengenes,
                  nGenes,
                  modProportions,
                  minCor = 0.5, maxCor = 1,
                  corPower = 1,
                  backgroundNoise = 0.1,
                  leaveOut = NULL,
                  signed = FALSE,
                  propNegativeCor = 0.3,
                  nSubmoduleLayers = 0,
                  nScatteredModuleLayers = 0,
                  averageNGenesInSubmodule = 10,
                  averageExprInSubmodule = 0.2,
                  submoduleSpacing = 2,
                  verbose = 1, indent = 0)
```

Arguments

eigengenes	the seed eigengenes for the simulated modules in a multi-set format. A list with one component per set. Each component is again a list that must contain a component data. This is a data frame of seed eigengenes for the corresponding data set. Columns correspond to modules, rows to samples. Number of samples in the simulated data is determined from the number of samples of the eigengenes.
nGenes	integer specifyin the number of simulated genes.
modProportions	a numeric vector with length equal the number of eigengenes in eigengenes plus one, containing fractions of the total number of genes to be put into each of the modules and into the "grey module", which means genes not related to any of the modules. See details.
minCor	minimum correlation of module genes with the corresponding eigengene. See details.
maxCor	maximum correlation of module genes with the corresponding eigengene. See details.
corPower	controls the dropoff of gene-eigengene correlation. See details.
backgroundNoise	amount of background noise to be added to the simulated expression data.
leaveOut	optional specification of modules that should be left out of the simulation, that is their genes will be simulated as unrelated ("grey"). A logical matrix in which columns correspond to sets and rows to modules. Wherever TRUE, the corresponding module in the corresponding data set will not be simulated, that is its genes will be simulated independently of the eigengene.
signed	logical: should the genes be simulated as belonging to a signed network? If TRUE, all genes will be simulated to have positive correlation with the eigengene. If FALSE, a proportion given by propNegativeCor will be simulated with negative correlations of the same absolute values.
propNegativeCor	proportion of genes to be simulated with negative gene-eigengene correlations. Only effective if signed is FALSE.
nSubmoduleLayers	number of layers of ordered submodules to be added. See details.
nScatteredModuleLayers	number of layers of scattered submodules to be added. See details.
averageNGenesInSubmodule	average number of genes in a submodule. See details.
averageExprInSubmodule	average strength of submodule expression vectors.
submoduleSpacing	a number giving submodule spacing: this multiple of the submodule size will lie between the submodule and the next one.
verbose	integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
indent	indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

For details of simulation of individual data sets and the meaning of individual set simulation arguments, see `simulateDatExpr`. This function simulates several data sets at a time and puts the result in a multi-set format. The number of genes is the same for all data sets. Module memberships are also the same, but modules can optionally be "dissolved", that is their genes will be simulated as unassigned. Such "dissolved", or left out, modules can be specified in the matrix `leaveOut`.

Value

A list with the following components:

`multiExpr` simulated expression data in multi-set format analogous to that of the input `eigengenes`. A list with one component per set. Each component is again a list that must contains a component `data`. This is a data frame of expression data for the corresponding data set. Columns correspond to genes, rows to samples.

`setLabels` a matrix of dimensions (number of genes) times (number of sets) that contains module labels for each genes in each simulated data set.

`allLabels` a matrix of dimensions (number of genes) times (number of sets) that contains the module labels that would be simulated if no module were left out using `leaveOut`. This means that all columns of the matrix are equal; the columns are repeated for convenience so `allLabels` has the same dimensions as `setLabels`.

`labelOrder` a matrix of dimensions (number of modules) times (number of sets) that contains the order in which module labels were assigned to genes in each set. The first label is assigned to genes 1...(module size of module labeled by first label), the second label to the following batch of genes etc.

Author(s)

Peter Langfelder

References

A short description of the simulation method can also be found in the Supplementary Material to the article

Langfelder P, Horvath S (2007) Eigengene networks for studying the relationships between co-expression modules. BMC Systems Biology 2007, 1:54.

The material is posted at http://www.genetics.ucla.edu/labs/horvath/CoexpressionNetwork/EigengeneNetwork/Suppleme

See Also

`simulateEigengeneNetwork` for a simulation of eigengenes with a given causal structure;

`simulateDatExpr` for simulation of individual data sets;

`simulateDatExpr5Modules` for a simple simulation of a data set consisting of 5 modules;

`simulateModule` for simulations of individual modules;

simulateSmallLayer *Simulate small modules*

Description

This function simulates a set of small modules. The primary purpose is to add a submodule structure to the main module structure simulated by simulateDatExpr.

Usage

```
simulateSmallLayer(
  order,
  nSamples,
  minCor = 0.3, maxCor = 0.5, corPower = 1,
  averageModuleSize,
  averageExpr,
  moduleSpacing,
  verbose = 4, indent = 0)
```

Arguments

order	a vector giving the simulation order for vectors. See details.
nSamples	integer giving the number of samples to be simulated.
minCor	a multiple of maxCor (see below) giving the minimum correlation of module genes with the corresponding eigengene. See details.
maxCor	maximum correlation of module genes with the corresponding eigengene. See details.
corPower	controls the dropoff of gene-eigengene correlation. See details.
averageModuleSize	average number of genes in a module. See details.
averageExpr	average strength of module expression vectors.
moduleSpacing	a number giving module spacing: this multiple of the module size will lie between the module and the next one.
verbose	integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
indent	indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

Module eigenvectors are chosen randomly and independently. Module sizes are chosen randomly from an exponential distribution with mean equal averageModuleSize. Two thirds of genes in each module are simulated as proper module genes and one third as near-module genes (see simulateModule for details). Between each successive pairs of modules a number of genes given by moduleSpacing will be left unsimulated (zero expression). Module expression, that is the expected standard deviation of the module expression vectors, is chosen randomly from an exponential distribution with mean equal averageExpr. The expression profiles are chosen such that their correlations with the eigengene run from just below maxCor to minCor * maxCor

(hence minCor must be between 0 and 1, not including the bounds). The parameter `corPower` can be chosen to control the behaviour of the simulated correlation with the gene index; values higher than 1 will result in the correlation approaching `minCor * maxCor` faster and lower than 1 slower.

The simulated genes will be returned in the order given in `order`.

Value

A matrix of simulated gene expressions, with dimension `(nSamples, length(order))`.

Author(s)

Peter Langfelder

See Also

`simulateModule` for simulation of individual modules;

`simulateDatExpr` for the main gene expression simulation function.

`sizeGrWindow` *Opens a graphics window with specified dimensions*

Description

If a graphic device window is already open, it is closed and re-opened with specified dimensions (in inches); otherwise a new window is opened.

Usage

```
sizeGrWindow(width, height)
```

Arguments

`width`	desired width of the window, in inches.
`height`	desired heigh of the window, in inches.

Value

None.

Author(s)

Peter Langfelder

softConnectivity *Calculates connectivity of a weighted network.*

Description

Given expression data, the function constructs the adjacency matrix and for each node calculates its connectivity, that is the sum of the adjacency to the other nodes.

Usage

```
softConnectivity(
  datExpr,
  corFnc = "cor", corOptions = "use = 'p'",
  type = "unsigned",
  power = if (type == "signed") 15 else 6,
  blockSize = 1500,
  minNSamples = NULL,
  verbose = 2, indent = 0)
```

Arguments

datExpr	a data frame containing the expression data, with rows corresponding to samples and columns to genes.
corFnc	character string giving the correlation function to be used for the adjacency calculation. Recommended choices are "cor" and "bicor", but other functions can be used as well.
corOptions	character string giving further options to be passed to the correlation function.
type	network type. Allowed values are (unique abbreviations of) "unsigned", "signed", "signed hybrid".
power	soft thresholding power.
blockSize	block size in which adjacency is to be calculated. Too low (say below 100) may make the calculation inefficient, while too high may cause R to run out of physical memory and slow down the computer. Should be chosen such that an array of doubles of size (number of genes) * (block size) fits into available physical memory.
minNSamples	minimum number of samples available for the calculation of adjacency for the adjacency to be considered valid. If not given, defaults to the greater of ..minNSamples (currently 4) and number of samples divided by 3. If the number of samples falls below this threshold, the connectivity of the corresponding gene will be returned as NA.
verbose	integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
indent	indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Value

A vector with one entry per gene giving the connectivity of each gene in the weighted network.

Author(s)

Steve Horvath

References

Bin Zhang and Steve Horvath (2005) "A General Framework for Weighted Gene Co-Expression Network Analysis", Statistical Applications in Genetics and Molecular Biology: Vol. 4: No. 1, Article 17

See Also

adjacency

---

spaste *Space-less paste*

Description

A convenient wrapper for the paste function with sep="".

Usage

```
spaste(...)
```

Arguments

...	standard arguments to function paste except sep.

Value

The result of the corresponding paste.

Note

Do not use the sep argument. Using will lead to an error.

Author(s)

Peter Langfelder

See Also

paste

Examples

```
a = 1;
paste("a=", a);
spaste("a=", a);
```

standardColors *Colors this library uses for labeling modules.*

Description

Returns the vector of color names in the order they are assigned by other functions in this library.

Usage

```
standardColors(n = NULL)
```

Arguments

n	Number of colors requested. If NULL, all (approx. 450) colors will be returned. Any other invalid argument such as less than one or more than maximum (length(standardColors())) will trigger an error.

Value

A vector of character color names of the requested length.

Author(s)

Peter Langfelder, <Peter.Langfelder@gmail.com>

Examples

```
standardColors(10);
```

standardScreeningBinaryTrait *Standard screening for binatry traits*

Description

The function standardScreeningBinaryTrait computes widely used statistics for relating the columns of the input data frame (argument datE) to a binary sample trait (argument y). The statistics include Student t-test p-value and the corresponding local false discovery rate (known as q-value, Storey et al 2004), the fold change, the area under the ROC curve (also known as C-index), mean values etc. If the input option KruskalTest is set to TRUE, it also computes the Kruskal Wallist test p-value and corresponding q-value. The Kruskal Wallis test is a non-parametric, rank-based group comparison test.

Usage

```
standardScreeningBinaryTrait(datExpr, y, kruskalTest = FALSE)
```

Arguments

datExpr	a data frame or matrix whose columns will be related to the binary trait
y	a binary vector whose length (number of components) equals the number of rows of datE
kruskalTest	logical: should the Kruskal test be performed?

Value

A data frame whose rows correspond to the columns of datE and whose columns report

ID	column names of the input datExpr.
corPearson	pearson correlation with a binary numeric version of the input variable. The numeric variable equals 1 for level 1 and 2 for level 2. The levels are given by levels(factor(y)).
pvalueStudent	two-sided Student t-test p-value.
qvalueStudent	q-value (local false discovery rate) based on the Student T-test p-value (Storey et al 2004).
foldChange	a (signed) ratio of mean values. If the mean in the first group (corresponding to level 1) is larger than that of the second group, it equals meanFirstGroup/meanSecondGroup. But if the mean of the second group is larger than that of the first group it equals -meanSecondGroup/meanFirstGroup (notice the minus sign).
meanFirstGroup	means of columns in input datExpr across samples in the first group.
meanSecondGroup	means of columns in input datExpr across samples in the second group.
areaUnderROC	the area under the ROC, also known as the concordance index or C.index. This is a measure of discriminatory power. The measure lies between 0 and 1 where 0.5 indicates no discriminatory power. 0 indicates that the "opposite" predictor has perfect discriminatory power. To compute it we use the function rcorr.cens with outx=T (from Frank Harrel's package Hmisc).

Author(s)

Steve Horvath

References

Storey JD, Taylor JE, and Siegmund D. (2004) Strong control, conservative point estimation, and simultaneous conservative consistency of false discovery rates: A unified approach. Journal of the Royal Statistical Society, Series B, 66: 187-205.

`standardScreeningCensoredTime`
Standard Screening with regard to a Censored Time Variable

Description

The function standardScreeningCensoredTime computes association measures between the columns of the input data datE and a censored time variable (e.g. survival time). The censored time is specified using two input variables "time" and "event". The event variable is binary where 1 indicates that the event took place (e.g. the person died) and 0 indicates censored (i.e. lost to follow up). The function fits univariate Cox regression models (one for each column of datE) and outputs a Wald test p-value, a logrank p-value, corresponding local false discovery rates (known as q-values, Storey et al 2004), hazard ratios. Further it reports the concordance index (also know as area under the ROC curve) and optionally results from dichotomizing the columns of datE.

Usage

```
standardScreeningCensoredTime(
   time,
   event,
   datExpr,
   percentiles = seq(from = 0.1, to = 0.9, by = 0.2),
   dichotomizationResults = FALSE,
   qValues = TRUE,
   fastCalculation = TRUE)
```

Arguments

`time`	numeric variable showing time to event or time to last follow up.
`event`	Input variable `time` specifies the time to event or time to last follow up. Input variable `event` indicates whether the event happend (=1) or whether there was censoring (=0).
`datExpr`	a data frame or matrix whose columns will be related to the censored time.
`percentiles`	numeric vector which is only used when dichotomizationResults=T. Each value should lie between 0 and 1. For each value specified in the vector percentiles, a binary vector will be defined by dichotomizing the column value according to the corresponding quantile. Next a corresponding p-value will be calculated.
`dichotomizationResults`	logical. If this option is set to TRUE then the values of the columns of datE will be dichotomized and corresponding Cox regression p-values will be calculated.
`qValues`	logical. If this option is set to TRUE (default) then q-values will be calculated for the Cox regression p-values.
`fastCalculation`	logical. If set to TRUE, the function outputs correlation test p-values (and q-values) for correlating the columns of datE with the expected hazard (if no covariate is fit). Specifically, the expected hazard is defined as the deviance residual of an intercept only Cox regression model. The results are very similar to those resulting from a univariate Cox model where the censored time is regressed on the columns of dat. Specifically, this computational speed up

is facilitated by the insight that the p-values resulting from a univariate Cox regression coxph(Surv(time,event)~datE[,i]) are very similar to those from corPvalueFisher(cor(devianceResidual,datE[,i]), nSamples).

Details

If input option fastCalculation=TRUE, then the function outputs correlation test p-values (and q-values) for correlating the columns of datE with the expected hazard (if no covariate is fit). Specifically, the expected hazard is defined as the deviance residual of an intercept only Cox regression model. The results are very similar to those resulting from a univariate Cox model where the censored time is regressed on the columns of dat. Specifically, this computational speed up is facilitated by the insight that the p-values resulting from a univariate Cox regression coxph(Surv(time,event)~datE[,i]) are very similar to those from corPvalueFisher(cor(devianceResidual,datE[,i]), nSamples)

Value

If `fastCalculation` is `FALSE`, the function outputs a data frame whose rows correspond to the columns of datE and whose columns report

`ID` column names of the input data datExpr.

`pvalueWald` Wald test p-value from fitting a univariate Cox regression model where the censored time is regressed on each column of datExpr.

`qValueWald` local false discovery rate (q-value) corresponding to the Wald test p-value.

`pvalueLogrank`
Logrank p-value resulting from the Cox regression model. Also known as score test p-value. For large sample sizes this sould be similar to the Wald test p-value.

`qValueLogrank`
local false discovery rate (q-value) corresponding to the Logrank test p-value.

`HazardRatio` hazard ratio resulting from the Cox model. If the value is larger than 1, then high values of the column are associated with shorter time, e.g. increased hazard of death. A hazard ratio equal to 1 means no relationship between the column and time. HR<1 means that high values are associated with longer time, i.e. lower hazard.

`CI.LowerLimitHR`
Lower bound of the 95 percent confidence interval of the hazard ratio.

`CI.UpperLimitHR`
Upper bound of the 95 percent confidence interval of the hazard ratio.

`C.index` concordance index, also known as C-index or area under the ROC curve. Calculated with the rcorr.cens option outx=TRUE (ties are ignored).

`MinimumDichotPvalue`
This is the smallest p-value from the dichotomization results. To see which dichotomized variable (and percentile) corresponds to the minimum, study the following columns.

`pValueDichot0.1`
This columns report the p-value when the column is dichotomized according to the specified percentile (here 0.1). The percentiles are specified in the input option percentiles.

`pvalueDeviance`
The p-value resulting from using a correlation test to relate the expected hazard (deviance residual) with each (undichotomized) column of datE. Specifically,

the Fisher transformation is used to calculate the p-value for the Pearson correlation. The resulting p-value should be very similar to that of a univariate Cox regression model.

`qvalueDeviance`
: Local false discovery rate (q-value) corresponding to pvalueDeviance.

`corDeviance`
: Pearson correlation between the expected hazard (deviance residual) with each (undichotomized) column of datExpr.

Author(s)

Steve Horvath

`stat.bwss` *Between and Within Group Sum of Squares Calculation*

Description

This function computes the between and within group sum of squares for each row of a matrix which may have NAs.

Usage

```
stat.bwss(x, cl)
```

Arguments

`x`
: a matrix, NAs allowed. In the microarray context, the rows of `X` could correspond to genes and the columns to different hybridizations.

`cl`
: column labels, must be consecutive integers.

Value

List containing the following components

`wn`
: matrix with class sizes for each row of `X`;

`BW`
: vector of BSS/WSS for each row of `X`;

`BSS`
: between group sum of squares for each row of `X`;

`WSS`
: within group sum of squares for each row of `X`;

`TSS`
: total sum of squares for each row of `X`;

`tvar`
: variance for each row of `X`.

Author(s)

Sandrine Dudoit, <sandrine@stat.berkeley.edu>
Jane Fridlyand, <janef@stat.berkeley.edu>

References

S. Dudoit, J. Fridlyand, and T. P. Speed. Comparison of Discrimination Methods for the Classification of Tumors Using Gene Expression Data. June 2000. (Statistics, UC Berkeley, Tech Report \# 576).

See Also

var.na, var, apply.

---

stat.diag.da *Diagonal Discriminant Analysis*

Description

This function implements a simple Gaussian maximum likelihood discriminant rule, for diagonal class covariance matrices.

Usage

```
stat.diag.da(ls, cll, ts, pool=1)
```

Arguments

ls	learning set data matrix, with rows corresponding to cases (i.e., mRNA samples) and columns to predictor variables (i.e., genes).
cll	class labels for learning set, must be consecutive integers.
ts	test set data matrix, with rows corresponding to cases and columns to predictor variables.
pool	logical flag. If pool=1, the covariance matrices are assumed to be constant across classes and the discriminant rule is linear in the data. If pool=0, the covariance matrices may vary across classes and the discriminant rule is quadratic in the data.

Value

List containing the following components

pred	vector of class predictions for the test set.

Author(s)

Sandrine Dudoit, <sandrine@stat.berkeley.edu>
Jane Fridlyand, <janef@stat.berkeley.edu>

References

S. Dudoit, J. Fridlyand, and T. P. Speed. Comparison of Discrimination Methods for the Classification of Tumors Using Gene Expression Data. June 2000. (Statistics, UC Berkeley, Tech Report \#576).

stdErr — *standard error of the mean of a given vector.*

Description

Returns the standard error of the mean of a given vector. Missing values are ignored.

Usage

```
stdErr(x)
```

Arguments

x	a numeric vector

Value

Standard error of the mean of x.

Author(s)

Steve Horvath

TOMplot — *Graphical representation of the Topological Overlap Matrix*

Description

Graphical representation of the Topological Overlap Matrix using a heatmap plot combined with the corresponding hierarchical clustering dendrogram and module colors.

Usage

```
TOMplot(
   dissim,
   dendro,
   colors = NULL,
   colorsLeft = colors,
   terrainColors = FALSE,
   setLayout = TRUE,
   ...)
```

Arguments

dissim	a matrix containing the topological overlap-based dissimilarity
dendro	the corresponding hierarchical clustering dendrogram
colors	optional specification of module colors to be plotted on top
colorsLeft	optional specification of module colors on the left side. If NULL, colors will be used.
terrainColors	logical: should terrain colors be used?
setLayout	logical: should layout be set? If TRUE, standard layout for one plot will be used. Note that this precludes multiple plots on one page. If FALSE, the user is responsible for setting the correct layout.
...	other graphical parameters to heatmap.

Details

The standard heatmap function uses the layout function to set the following layout (when colors is given):

```
0 0 5
0 0 2
4 1 3
```

To get a meaningful heatmap plot, user-set layout must respect this geometry.

Value

None.

Author(s)

Steve Horvath and Peter Langfelder

See Also

heatmap, the workhorse function doing the plotting.

TOMsimilarity *Topological overlap matrix similarity and dissimilarity*

Description

Calculation of the topological overlap matrix from a given adjacency matrix.

Usage

```
TOMsimilarity(adjMat, TOMType = "unsigned", TOMDenom = "min", verbose = 1, inden
TOMdist(adjMat, TOMType = "unsigned", TOMDenom = "min", verbose = 1, indent = 0)
```

Arguments

adjMat	adjacency matrix, that is a square, symmetric matrix with entries between 0 and 1 (negative values are allowed if TOMType=="signed").
TOMType	a character string specifying TOM type to be calculated. One of "unsigned", "signed". If "unsigned", the standard TOM will be used (more generally, TOM function will receive the adjacency as input). If "signed", TOM will keep track of the sign of the adjacency between neighbors.
TOMDenom	a character string specifying the TOM variant to be used. Recognized values are "min" giving the standard TOM described in Zhang and Horvath (2005), and "mean" in which the min function in the denominator is replaced by mean. The "mean" may produce better results but at this time should be considered experimental.
verbose	integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
indent	indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

The functions perform basically the same calculations of topological overlap. TOMdist turns the overlap (which is a measure of similarity) into a measure of dissimilarity by subtracting it from 1.

Basic checks on the adjacency matrix are performed and missing entries are replaced by zeros. If TOMType = "unsigned", entries of the adjacency matrix are required to lie between 0 and 1; for TOMType = "signed" they can be between -1 and 1. In both cases the resulting TOM entries, as well as the corresponding dissimilarities, lie between 0 and 1.

Value

A matrix holding the topological overlap.

Author(s)

Peter Langfelder

References

Bin Zhang and Steve Horvath (2005) "A General Framework for Weighted Gene Co-Expression Network Analysis", Statistical Applications in Genetics and Molecular Biology: Vol. 4: No. 1, Article 17

See Also

TOMsimilarityFromExpr

TOMsimilarityFromExpr
Topological overlap matrix

Description

Calculation of the topological overlap matrix from given expression data.

Usage

```
TOMsimilarityFromExpr(
  datExpr,
  corType = "pearson",
  networkType = "unsigned",
  power = 6,
  TOMType = "signed",
  TOMDenom = "min",
  maxPOutliers = 1,
  quickCor = 0,
  pearsonFallback = "individual",
  nThreads = 0,
  verbose = 1, indent = 0)
```

Arguments

datExpr	expression data. A data frame in which columns are genes and rows ar samples. NAs are allowed, but not too many.
corType	character string specifying the correlation to be used. Allowed values are (unique abbreviations of) "pearson" and "bicor", corresponding to Pearson and bidweight midcorrelation, respectively. Missing values are handled using the pairwise.complete.obs option.
networkType	network type. Allowed values are (unique abbreviations of) "unsigned", "signed", "signed hybrid". See adjacency.
power	soft-thresholding power for netwoek construction.
TOMType	one of "none", "unsigned", "signed". If "none", adjacency will be used for clustering. If "unsigned", the standard TOM will be used (more generally, TOM function will receive the adjacency as input). If "signed", TOM will keep track of the sign of correlations between neighbors.
TOMDenom	a character string specifying the TOM variant to be used. Recognized values are "min" giving the standard TOM described in Zhang and Horvath (2005), and "mean" in which the min function in the denominator is replaced by mean. The "mean" may produce better results but at this time should be considered experimental.
maxPOutliers	only used for corType=="bicor". Specifies the maximum percentile of data that can be considered outliers on either side of the median separately. For each side of the median, if higher percentile than maxPOutliers is considered an outlier by the weight function based on 9*mad(x), the width of the weight function is increased such that the percentile of outliers on that side of

the median equals maxPOutliers. Using maxPOutliers=1 will effectively disable all weight function broadening; using maxPOutliers=0 will give results that are quite similar (but not equal to) Pearson correlation.

quickCor — real number between 0 and 1 that controls the handling of missing data in the calculation of correlations. See details.

pearsonFallback — Specifies whether the bicor calculation, if used, should revert to Pearson when median absolute deviation (mad) is zero. Recongnized values are (abbreviations of) "none", "individual", "all". If set to "none", zero mad will result in NA for the corresponding correlation. If set to "individual", Pearson calculation will be used only for columns that have zero mad. If set to "all", the presence of a single zero mad will cause the whole variable to be treated in Pearson correlation manner (as if the corresponding robust option was set to FALSE). Has no effect for Pearson correlation. See bicor.

nThreads — non-negative integer specifying the number of parallel threads to be used by certain parts of correlation calculations. This option only has an effect on systems on which a POSIX thread library is available (which currently includes Linux and Mac OSX, but excludes Windows). If zero, the number of online processors will be used if it can be determined dynamically, otherwise correlation calculations will use 2 threads.

verbose — integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.

indent — indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Value

A matrix holding the topological overlap.

Author(s)

Peter Langfelder

References

Bin Zhang and Steve Horvath (2005) "A General Framework for Weighted Gene Co-Expression Network Analysis", Statistical Applications in Genetics and Molecular Biology: Vol. 4: No. 1, Article 17

See Also

TOMsimilarity

---

unsignedAdjacency *Calculation of unsigned adjacency*

Description

Calculation of the unsigned network adjacency from expression data. The restricted set of parameters for this function should allow a faster and less memory-hungry calculation.

Usage

```
unsignedAdjacency(
  datExpr,
  datExpr2 = NULL,
  power = 6,
  corFnc = "cor", corOptions = "use = 'p'")
```

Arguments

datExpr	expression data. A data frame in which columns are genes and rows ar samples. Missing values are ignored.
datExpr2	optional specification of a second set of expression data. See details.
power	soft-thresholding power for network construction.
corFnc	character string giving the correlation function to be used for the adjacency calculation. Recommended choices are "cor" and "bicor", but other functions can be used as well.
corOptions	character string giving further options to be passed to the correlation function

Details

The correlation function will be called with arguments datExpr, datExpr2 plus any extra arguments given in corOptions. If datExpr2 is NULL, the standard correlation functions will calculate the corelation of columns in datExpr.

Value

Adjacency matrix of dimensions n*n, where n is the number of genes in datExpr.

Author(s)

Steve Horvath and Peter Langfelder

References

Bin Zhang and Steve Horvath (2005) "A General Framework for Weighted Gene Co-Expression Network Analysis", Statistical Applications in Genetics and Molecular Biology: Vol. 4: No. 1, Article 17

See Also

adjacency

vectorizeMatrix *Turn a matrix into a vector of non-redundant components*

Description

A convenient function to turn a matrix into a vector of non-redundant components. If the matrix is non-symmetric, returns a vector containing all entries of the matrix. If the matrix is symmetric, only returns the upper triangle and optionally the diagonal.

Usage

```
vectorizeMatrix(M, diag = FALSE)
```

Arguments

M	the matrix or data frame to be vectorized.
diag	logical: should the diagonal be included in the output?

Value

A vector containing the non-redundant entries of the input matrix.

Author(s)

Steve Horvath

vectorTOM *Topological overlap for a subset of the whole set of genes*

Description

This function calculates topological overlap of a small set of vectors with respect to a whole data set.

Usage

```
vectorTOM(
  datExpr,
  vect,
  subtract1 = FALSE,
  blockSize = 2000,
  corFnc = "cor", corOptions = "use = 'p'",
  type = "unsigned",
  power = 6,
  verbose = 1, indent = 0)
```

Arguments

datExpr	a data frame containing the expression data of the whole set, with rows corresponding to samples and columns to genes.
vect	a single vector or a matrix-like object containing vectors whose topological overlap is to be calculated.
subtract1	logical: should calculation be corrected for self-correlation? Set this to TRUE if vect contains a subset of datExpr.
blockSize	maximum block size for correlation calculations. Only important if vect contains a large number of columns.
corFnc	character string giving the correlation function to be used for the adjacency calculation. Recommended choices are "cor" and "bicor", but other functions can be used as well.
corOptions	character string giving further options to be passed to the correlation function.
type	character string giving network type. Allowed values are (unique abbreviations of) "unsigned", "signed", "signed hybrid". See adjacency.
power	soft-thresholding power for network construction.
verbose	integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
indent	indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

Topological overlap can be viewed as the normalized count of shared neighbors encoded in an adjacency matrix. In this case, the adjacency matrix is calculated between the columns of vect and datExpr and the topological overlap of vectors in vect measures the number of shared neighbors in datExpr that vectors of vect share.

Value

A matrix of dimensions n*n, where n is the number of columns in vect.

Author(s)

Peter Langfelder

References

Bin Zhang and Steve Horvath (2005) "A General Framework for Weighted Gene Co-Expression Network Analysis", Statistical Applications in Genetics and Molecular Biology: Vol. 4: No. 1, Article 17

See Also

TOMsimilarity for standard calculation of topological overlap.

verboseBarplot *Barplot with error bars, annotated by Kruskal-Wallis or ANOVA p-value*

Description

Produce a barplot with error bars, annotated by Kruskal-Wallis or ANOVA p-value.

Usage

```
verboseBarplot(x, g,
               main = "", xlab = NA, ylab = NA,
               cex = 1, cex.axis = 1.5, cex.lab = 1.5, cex.main = 1.5,
               color = "grey", numberStandardErrors = 1,
               KruskalTest = TRUE, AnovaTest = FALSE, two.sided = TRUE, ...)
```

Arguments

x	numerical vector of data whose group means are to be plotted
g	a factor or a an object coercible to a factor giving the groups whose means are to be calculated.
main	main title for the plot.
xlab	label for the x-axis.
ylab	label for the y-axis.
cex	character expansion factor for plot annotations.
cex.axis	character expansion factor for axis annotations.
cex.lab	character expansion factor for axis labels.
cex.main	character expansion factor for the main title.
color	a vector giving the colors of the bars in the barplot.
numberStandardErrors	size of the error bars in terms of standard errors. See details.
KruskalTest	logical: should Kruskal-Wallis test be performed? See details.
AnovaTest	logical: should ANOVA be performed? See details.
two.sided	logical: should the printed p-value be two-sided? See details.
...	other parameters to function barplot

Details

This function creates a barplot of a numeric variable (input x) across the levels of a grouping variable (input g). The height of the bars equals the mean value of x across the observations with a given level of g. By default, the barplot also shows plus/minus one standard error. If you want only plus one standard error (not minus) choose two.sided=TRUE. But the number of standard errors can be determined with the input numberStandardErrors. For example, if you want a 95% confidence interval around the mean, choose numberStandardErrors=2. If you don't want any standard errors set numberStandardErrors=-1. The function also outputs the p-value of a Kruskal Wallis test, which is a non-parametric multi group comparison test. Alternatively, one can use Analysis of Variance (Anova) to compute a p-value by setting

AnovaTest=TRUE. Anova is a generalization of the Student t-test to multiple groups. In case of two groups, the Anova p-value equals the Student t-test p-value. Anova should only be used if x follows a normal distribution. Anova also assumes homoscedasticity (equal variances). The Kruskal Wallis test is often advantageous since it makes no distributional assumptions. Since the Kruskal Wallis test is based on the ranks of x, it is more robust with regard to outliers. All p-values are two-sided.

Value

None.

Author(s)

Steve Horvath

See Also

barplot

Examples

```
group=sample(c(1,2),100,replace=TRUE)

height=rnorm(100,mean=group)

par(mfrow=c(2,2))
verboseBarplot(height,group, main="1 SE, Kruskal Test")

verboseBarplot(height,group,numberStandardErrors=2,
               main="2 SE, Kruskal Test")

verboseBarplot(height,group,numberStandardErrors=2,AnovaTest=TRUE,
               main="2 SE, Anova")

verboseBarplot(height,group,numberStandardErrors=2,AnovaTest=TRUE,
               main="2 SE, Anova, only plus SE", two.sided=FALSE)
```

---

verboseBoxplot *Boxplot annotated by a Kruskal-Wallis p-value*

Description

Plot a boxplot annotated by the Kruskal-Wallis p-value.

Usage

```
verboseBoxplot(x, g, main = "", xlab = NA, ylab = NA,
               cex = 1, cex.axis = 1.5, cex.lab = 1.5, cex.main = 1.5, ...)
```

Arguments

x	numerical vector of data whose group means are to be plotted
g	a factor or a an object coercible to a factor giving the groups that will go into each box.
main	main title for the plot.
xlab	label for the x-axis.
ylab	label for the y-axis.
cex	character expansion factor for plot annotations.
cex.axis	character expansion factor for axis annotations.
cex.lab	character expansion factor for axis labels.
cex.main	character expansion factor for the main title.
...	other parameters to the function boxplot

Value

Returns the value returned by the function boxplot.

Author(s)

Steve Horvath

See Also

boxplot

---

verboseScatterplot *Scatterplot annotated by regression line and p-value*

Description

Produce a scatterplot annotated by the correlation, p-value, and regression line.

Usage

```
verboseScatterplot(x, y,
                   sample = NULL,
                   corFnc = "cor", corOptions = "use = 'p'",
                   main = "", xlab = NA, ylab = NA,
                   cex = 1, cex.axis = 1.5, cex.lab = 1.5, cex.main = 1.5,
                   abline = FALSE, corLabel = corFnc, ...)
```

Arguments

x	numerical vector to be plotted along the x axis.
y	numerical vector to be plotted along the y axis.
sample	determines whether x and y should be sampled for plotting, useful to keep the plot manageable when x and y are large vectors. The default NULL value implies no sampling. A single numeric value will be interpreted as the number of points to sample randomly. If a vector is given, it will be interpreted as the indices of the entries in x and y that should be plotted. In either case, the correlation and p value will be determined from the full vectors x and y.
corFnc	character string giving the correlation function to annotate the plot.
corOptions	character string giving further options to the correlation function.
main	main title for the plot.
xlab	label for the x-axis.
ylab	label for the y-axis.
cex	character expansion factor for plot annotations.
cex.axis	character expansion factor for axis annotations.
cex.lab	character expansion factor for axis labels.
cex.main	character expansion factor for the main title.
abline	logical: should the linear regression fit line be plotted?
corLabel	character string to be used as the label for the correlation value printed in the main title.
...	other arguments to the function plot.

Details

Irrespective of the specified correlation function, the p-value is always calculated for pearson correlation.

Value

If sample above is given, the indices of the plotted points are returned invisibly.

Author(s)

Steve Horvath and Peter Langfelder

See Also

plot.default for standard scatterplots

Index