

Package ‘WGCNA’

February 18, 2009

Version 0.72

Date 2009-02-18

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, fields, impute, grDevices, dynamicTreeCut (>= 1.20), qvalue, utils,
flashClust

Suggests

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:

	addErrorBars	4
	addGrid	4
	addGuideLines	5
	addTraitToMEs	6
	adjacency	6
	alignExpr	8
	automaticNetworkScreeningGS	8
	automaticNetworkScreening	10
0.1	Warning	11
	bicor	12
	blockwiseConsensusModules	13
	blockwiseModules	19
	checkAdjMat	23
	checkSets	24
	clusterCoef	25
	collectGarbage	25
	colQuantileC	26

	consensusMEDissimilarity	26
	consensusOrderMEs	27
	consensusProjectiveKMeans	28
	corPredictionSuccess	30
0.2	Warning	31
	corPvalueFisher	32
	corPvalueStudent	33
	correlationPreservation	34
	cutreeStaticColor	35
	cutreeStatic	35
	displayColors	36
	dynamicMergeCut	37
	exportNetworkToCytoscape	38
	exportNetworkToVisANT	39
	fixDataStructure	40
	goodGenesMS	41
	goodGenes	42
	goodSamplesGenesMS	43
	goodSamplesGenes	44
	goodSamplesMS	45
	goodSamples	46
	greenBlackRed	48
	greenWhiteRed	48
	GTOMdist	49
	hubGeneSignificance	50
	Inline display of progress	51
	intramodularConnectivity	52
	keepCommonProbes	53
	labeledBarplot	54
	labeledHeatmap	55
	labels2colors	58
	matchLabels	59
	mergeCloseModules	60
	moduleColor.getMEprefix	62
	moduleEigengenes	63
	moduleNumber	66
	multiSetMEs	67
	nearestNeighborConnectivityMS	70
	nearestNeighborConnectivity	71
	networkConcepts	73
	networkScreeningGS	74
0.3	Warning	74
	networkScreening	76
0.4	Warning	77
	normalizeLabels	80
	nPresent	80
	numbers2colors	81
	orderMEs	82
	overlapTable	83
	pickHardThreshold	84
	pickSoftThreshold	85
	plotClusterTreeSamples	86

	plotColorUnderTree	88
	plotDendroAndColors	89
	plotEigengeneNetworks	91
	plotMEpairs	94
	plotModuleSignificance	95
	plotNetworkHeatmap	96
	preservationNetworkConnectivity	97
	projectiveKMeans	99
	propVarExplained	101
	randIndex	102
0.5	Warning	102
	recutBlockwiseTrees	103
	recutConsensusTrees	107
	redWhiteGreen	110
	relativeCorPredictionSuccess	111
0.6	Warning	112
	removeGreyME	113
	scaleFreePlot	114
	setCorrelationPreservation	115
	sigmoidAdjacencyFunction	116
	signedKME	117
	signumAdjacencyFunction	118
	simulateDatExpr5Modules	118
	simulateDatExpr	120
	simulateEigengeneNetwork	122
	simulateModule	123
	simulateMultiExpr	125
0.7	Warning	126
	simulateSmallLayer	127
	sizeGrWindow	129
	softConnectivity	129
	standardColors	130
	stdErr	131
	TOMplot	132
	TOMsimilarityFromExpr	133
	TOMsimilarity	134
	unsignedAdjacency	135
	vectorTOM	136
	verboseBoxplot	137
0.8	Warning	138
	verboseScatterplot	139
0.9	Warning	140
	WGCNA-package	141

`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

<code>means</code>	vector of means plotted in the barplot
<code>errors</code>	vector of standard errors (single positive values) to be plotted.
<code>two.side</code>	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 = "grey", lty = 3)
```

Arguments

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

Details

If `linesPerTick` is not specified, it is set to 5 if number of ticks 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	<i>Add vertical “guide lines” to a dendrogram plot</i>
---------------	--

Description

Adds vertical “guide lines” to a dendrogram plot.

Usage

```
addGuideLines(dendro,
              all = FALSE,
              count = 50,
              positions = NULL,
              col = "grey60",
              lty = 3,
              hang = 0)
```

Arguments

<code>dendro</code>	The dendrogram (see hclust) to which the guide lines are to be added.
<code>all</code>	Add a guide line to every object on the dendrogram? Useful if the number of objects is relatively low.
<code>count</code>	Number of guide lines to be plotted. The lines will be equidistantly spaced.
<code>positions</code>	Horizontal positions of the added guide lines. If given, overrides <code>count</code> .
<code>col</code>	Color of the guide lines
<code>lty</code>	Line type of the guide lines. See par .
<code>hang</code>	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

<code>addTraitToMEs</code>	<i>Add trait information to multi-set module eigengene structure</i>
----------------------------	--

Description

Adds trait information to multi-set module eigengene structure.

Usage

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

Arguments

<code>multiME</code>	Module eigengenes in multi-set format. A vector of lists, one list per set. Each list must contain an element named <code>data</code> that is a data frame with module eigengenes.
<code>multiTraits</code>	Microarray sample trait(s) in multi-set format. A vector of lists, one list per set. Each list must contain an element named <code>data</code> 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](#)

<code>adjacency</code>	<i>Calculate network adjacency</i>
------------------------	------------------------------------

Description

Calculates network adjacency from given expression data.

Usage

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

Arguments

<code>datExpr</code>	data frame containing expression data. Columns correspond to genes and rows to samples.
<code>selectCols</code>	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.
<code>power</code>	soft thresholding power.
<code>type</code>	network type. Allowed values are (unique abbreviations of) "unsigned", "signed", "signed hybrid".
<code>corFnc</code>	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.
<code>corOptions</code>	character string specifying additional arguments to be passed to the function given by <code>corFnc</code> . 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

`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

<code>datExpr</code>	data frame containing the expression data, columns corresponding to genes and rows to samples
<code>GS</code>	vector containing gene significance for all genes given in <code>datExpr</code>
<code>power</code>	soft thresholding power used in network construction
<code>networkType</code>	character string specifying network type. Allowed values are (unique abbreviations of) "unsigned", "signed", <code>"hybrid"</code> .
<code>detectCutHeight</code>	cut height of the gene hierarchical clustering dendrogram. See <code>cutreeDynamic</code> for details.
<code>minModuleSize</code>	minimum module size to be used in module detection procedure.
<code>datME</code>	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:

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

Author(s)

Steve Horvath

See Also

[networkScreening](#), [hubGeneSignificance](#), [networkScreening](#), [cutreeDynamic](#)

```
automaticNetworkScreening
    function to do ...
```

Description

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

Usage

```
automaticNetworkScreening(datExpr, y, power = 6, networkType = "unsigned", detectCutHeight = 1,
minModuleSize = min(20, ncol(as.matrix(datExpr))/2), datME = NULL, getQValues = 0.01)
```

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 getQValues 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'
...	

Warning

....

Note

further notes

Make other sections like Warning with

0.1 Warning

....

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 (datExpr, y, power = 6, networkType = "unsigned", detectCutHeight = 0.995,
        minModuleSize = min(20, ncol(as.matrix(datExpr))/2), datME = NULL,
        ...)
{
  y = as.numeric(as.character(y))
  if (length(y) != dim(as.matrix(datExpr))[[1]])
    stop("Number of samples in 'y' and 'datExpr' disagree: length(y) != dim(as.matrix(
nAvailable = apply(as.matrix(!is.na(datExpr)), 2, sum)
ExprVariance = apply(as.matrix(datExpr), 2, var, na.rm = T)
restrictGenes = (nAvailable >= ..minNSamples) & (ExprVariance >
  0)
numberUsefulGenes = sum(restrictGenes, na.rm = T)
if (numberUsefulGenes < 3) {
  stop(paste("IMPORTANT: there are not enough useful genes. \n",
    "    Your input genes have fewer than 4 observations or they are constant.\n",
    "    WGCNA cannot be used for these data. Hint: collect more arrays or input
  })
datExprUsefulGenes = as.matrix(datExpr)[, restrictGenes &
  !is.na(restrictGenes)]
if (is.null(datME)) {
  mergeCutHeight1 = DynamicMergeCut(n = dim(as.matrix(datExprUsefulGenes))[[1]])
  B = blockwiseModules(datExprUsefulGenes, mergeCutHeight = mergeCutHeight1,
    TOMLevel = 0, power = power, networkType = networkType,
    detectCutHeight = detectCutHeight, minModuleSize = minModuleSize)
  datME = data.frame(B$MEs)
}
if (dim(as.matrix(datME))[[1]] != dim(as.matrix(datExpr))[[1]])
  stop(paste("Numbers of samples in 'datME' and 'datExpr' are incompatible:",
    "dim(as.matrix(datME))[[1]] != dim(as.matrix(datExpr))[[1]]"))
MMdata = signedKME(datExpr = datExpr, datME = datME, outputColumnName = "MM.")
MMdataPvalue = as.matrix(FisherTransformP(as.matrix(MMdata),
  n = dim(as.matrix(datExpr))[[1]]))
dimnames(MMdataPvalue)[[2]] = paste("Pvalue", names(MMdata),
```

```

    sep = ".")
  NS1 = networkScreening(y = y, datME = datME, datExpr = datExpr)
  ES = data.frame(cor(y, datME, use = "p"))
  rr = max(abs(ES), na.rm = T)
  AACriterion = sqrt(length(y) - 2) * rr/sqrt(1 - rr^2)
  ESy = (1 + max(abs(ES), na.rm = T))/2
  ES = data.frame(ES, ESy = ESy)
  ES.999 = as.numeric(as.vector(ES))
  ES.999[!is.na(ES) & ES > 0.9999] = 0.9999
  ES.pvalue = FisherTransformP(r = abs(ES.999), n = sum(!is.na(y)))
  ES.pvalue[length(ES.999)] = 0
  EigengeneSignificance.pvalue = data.frame(matrix(ES.pvalue,
    nrow = 1))
  names(EigengeneSignificance.pvalue) = names(ES)
  datME = data.frame(datME, y = y)
  names(ES) = paste("ES", substr(names(ES), 3, 100), sep = "")
  print(signif(ES, 2))
  output = list(networkScreening = data.frame(NS1, MMdata,
    MMdataPvalue), datME = data.frame(datME), eigengeneSignificance = data.frame(ES),
    eigengeneSignificance.pvalue = EigengeneSignificance.pvalue,
    AACriterion = AACriterion)
  output
}

```

 bicor

Biweight Midcorrelation

Description

Calculate biweight midcorrelation efficiently for matrices.

Usage

```
bicor(x, y = NULL, robustX = TRUE, robustY = TRUE, use = "all.obs", verbose = 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".
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 rows 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 calculation to revert to standard covariance and 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.

Value

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

Author(s)

Peter Langfelder, based on code by Rich Herrington

References

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

"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.

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",
  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,
verbose = 2, indent = 0)

```

Arguments

<code>multiExpr</code>	expression data in the multi-set format (see checkSets). A vector of lists, one per set. Each set must contain a component <code>data</code> that contains the expression data, with rows corresponding to samples and columns to genes or probes.
<code>blocks</code>	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 <code>multiExpr</code> giving the number of the block to which the corresponding gene belongs.
<code>maxBlockSize</code>	integer giving maximum block size for module detection. Ignored if <code>blocks</code> above is non-NULL. Otherwise, if the number of genes in <code>dataExpr</code> exceeds <code>maxBlockSize</code> , genes will be pre-clustered into blocks whose size should not exceed <code>maxBlockSize</code> .
<code>randomSeed</code>	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.
<code>corType</code>	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 <code>parwise.complete.obs</code> option.
<code>power</code>	soft-thresholding power for network construction.
<code>consensusQuantile</code>	quantile at which consensus is to be defined. See details.
<code>networkType</code>	network type. Allowed values are (unique abbreviations of) "unsigned", "signed", "signed hybrid". See adjacency .
<code>TOMType</code>	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.
<code>scaleTOMs</code>	should set-specific TOM matrices be scaled to the same scale?

<code>scaleQuantile</code>	if <code>scaleTOMs</code> is TRUE, topological overlaps (or adjacencies if TOMs are not computed) will be scaled such that their <code>scaleQuantile</code> quantiles will agree.
<code>sampleForScaling</code>	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.
<code>sampleForScalingFactor</code>	determines the number of samples for scaling: the number is $1/\text{scaleQuantile} * \text{sampleForScalingFactor}$. Should be set well above 1 to ensure accuracy of the sampled quantile.
<code>useDiskCache</code>	should calculated network similarities in individual sets be temporarily 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.
<code>chunkSize</code>	network similarities are saved in smaller chunks of size <code>chunkSize</code> .
<code>cacheBase</code>	character string containing the desired name for the cache files. The actual file names will consists of <code>cacheBase</code> and a suffix to make the file names unique.
<code>deepSplit</code>	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.
<code>detectCutHeight</code>	dendrogram cut height for module detection. See cutreeDynamic for more details.
<code>minModuleSize</code>	minimum module size for module detection. See cutreeDynamic for more details.
<code>checkMinModuleSize</code>	logical: should sanity checks be performed on <code>minModuleSize</code> ?
<code>maxCoreScatter</code>	maximum scatter of the core for a branch to be a cluster, given as the fraction of <code>cutHeight</code> relative to the 5th percentile of joining heights. See cutreeDynamic for more details.
<code>minGap</code>	minimum cluster gap given as the fraction of the difference between <code>cutHeight</code> and the 5th percentile of joining heights. See cutreeDynamic for more details.
<code>maxAbsCoreScatter</code>	maximum scatter of the core for a branch to be a cluster given as absolute heights. If given, overrides <code>maxCoreScatter</code> . See cutreeDynamic for more details.
<code>minAbsGap</code>	minimum cluster gap given as absolute height difference. If given, overrides <code>minGap</code> . See cutreeDynamic for more details.
<code>pamStage</code>	logical. If TRUE, the second (PAM-like) stage of module detection will be performed. See cutreeDynamic for more details.
<code>pamRespectsDendro</code>	Logical, only used when <code>pamStage</code> 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.
<code>minKMEtoJoin</code>	a number between 0 and 1. Genes with eigengene connectivity higher than <code>minKMEtoJoin</code> are automatically assigned to their closest module.

<code>minCoreKME</code>	a number between 0 and 1. If a detected module does not have at least <code>minModuleKMESize</code> genes with eigengene connectivity at least <code>minCoreKME</code> , the module is dis-banded (its genes are unlabeled and returned to the pool of genes waiting for module detection).
<code>minCoreKMESize</code>	see <code>minCoreKME</code> above.
<code>minKMEtoStay</code>	genes whose eigengene connectivity to their module eigengene is lower than <code>minKMEtoStay</code> are removed from the module.
<code>reassignThresholdPS</code>	per-set p-value ratio threshold for reassigning genes between modules. See De-tails.
<code>mergeCutHeight</code>	dendrogram cut height for module merging.
<code>impute</code>	logical: should imputation be used for module eigengene calculation? See moduleEigengenes for more details.
<code>getTOMs</code>	deprecated, please use <code>saveTOMs</code> below.
<code>saveTOMs</code>	logical: should the consensus topological overlap matrices for each block be saved and returned?
<code>saveTOMfileBase</code>	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.
<code>getTOMScalingSamples</code>	logical: should samples used for TOM scaling be saved for future analysis? This option is only available when <code>sampleForScaling</code> is TRUE.
<code>trapErrors</code>	logical: should errors in calculations be trapped?
<code>checkPower</code>	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.
<code>numericLabels</code>	logical: should the returned modules be labeled by colors (FALSE), or by num-bers (TRUE)?
<code>checkMissingData</code>	logical: should data be checked for excessive numbers of missing entries in genes and samples, and for genes with zero variance? See details.
<code>verbose</code>	integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
<code>indent</code>	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.

Value

A list with the following components:

<code>colors</code>	module assignment of all input genes. A vector containing either character strings with module colors (if <code>input numericLabels</code> was unset) or numeric module labels (if <code>numericLabels</code> was set to <code>TRUE</code>). The color "grey" and the numeric label 0 are reserved for unassigned genes.
<code>unmergedColors</code>	module colors or numeric labels before the module merging step.
<code>multiMEs</code>	module eigengenes corresponding to the modules returned in <code>colors</code> , in multi-set format. A vector of lists, one per set, containing eigengenes, proportion of variance explained and other information. See <code>multiSetMEs</code> for a detailed description.
<code>goodSamples</code>	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.
<code>goodGenes</code>	a logical vector with one entry per input gene indicating whether the gene passed basic quality control criteria in all sets.
<code>dendrograms</code>	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.
<code>TOMfiles</code>	if <code>saveTOMs==TRUE</code> , 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.
<code>blockGenes</code>	a list with one component for each block of genes. Each component is a vector giving the indices (relative to the input <code>multiExpr</code>) of genes in the corresponding block.

<code>blocks</code>	if input <code>blocks</code> 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 necessarily sorted in the order in which the blocks were processed (since we do not require this for the input <code>blocks</code>). See <code>blockOrder</code> below.
<code>blockOrder</code>	a vector giving the order in which blocks were processed and in which <code>blockGenes</code> above is returned. For example, <code>blockOrder[1]</code> contains the label of the first-processed block.
<code>originCount</code>	if the input <code>consensusQuantile==0</code> , 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.
<code>TOMScalingSamples</code>	if the input <code>getTOMScalingSamples</code> is <code>TRUE</code> , this component is a list with one component per block. Each component is again a list with two components: <code>sampleIndex</code> contains indices of the distance structure in which TOM is stored that were sampled, and <code>TOMSamples</code> is a matrix whose rows correspond to TOM samples and columns to individual set. Hence, <code>TOMScalingSamples[[blockNo]]\$TOM[setNo]</code> contains the TOM entry that corresponds to element <code>TOMScalingSamples[[blockNo]]\$sampleIndex[setNo]</code> of the TOM distance structure in block <code>blockNo</code> and set <code>setNo</code> . (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",  
  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,  
  verbose = 0, indent = 0)
```

Arguments

- | | |
|---------------------------|--|
| <code>datExpr</code> | expression data. A data frame in which columns are genes and rows are samples. NAs are allowed, but not too many. |
| <code>blocks</code> | 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 <code>exprData</code> giving the number of the block to which the corresponding gene belongs. |
| <code>maxBlockSize</code> | integer giving maximum block size for module detection. Ignored if <code>blocks</code> above is non-NULL. Otherwise, if the number of genes in <code>datExpr</code> exceeds <code>maxBlockSize</code> , genes will be pre-clustered into blocks whose size should not exceed <code>maxBlockSize</code> . |

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 <code>pairwise.complete.obs</code> 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.
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 <code>cutHeight</code> 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 <code>cutHeight</code> 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 <code>maxCoreScatter</code> . See cutreeDynamic for more details.
minAbsGap	minimum cluster gap given as absolute height difference. If given, overrides <code>minGap</code> . 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 <code>pamStage</code> 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 <code>minKMEtoJoin</code> are automatically assigned to their closest module.
minCoreKME	a number between 0 and 1. If a detected module does not have at least <code>minModuleKMESize</code> genes with eigengene connectivity at least <code>minCoreKME</code> , the module is disbanded (its genes are unlabeled and returned to the pool of genes waiting for module detection).

<code>minCoreKMESize</code>	see <code>minCoreKME</code> above.
<code>minKMEtoStay</code>	genes whose eigengene connectivity to their module eigengene is lower than <code>minKMEtoStay</code> are removed from the module.
<code>reassignThreshold</code>	p-value ratio threshold for reassigning genes between modules. See Details.
<code>mergeCutHeight</code>	dendrogram cut height for module merging.
<code>impute</code>	logical: should imputation be used for module eigengene calculation? See moduleEigengenes for more details.
<code>getTOMs</code>	deprecated, please use <code>saveTOMs</code> below.
<code>saveTOMs</code>	logical: should the consensus topological overlap matrices for each block be saved and returned?
<code>saveTOMfileBase</code>	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.
<code>trapErrors</code>	logical: should errors in calculations be trapped?
<code>numericLabels</code>	logical: should the returned modules be labeled by colors (FALSE), or by numbers (TRUE)?
<code>checkMissingData</code>	logical: should data be checked for excessive numbers of missing entries in genes and samples, and for genes with zero variance? See details.
<code>verbose</code>	integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
<code>indent</code>	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.

Value

A list with the following components:

<code>colors</code>	a vector of color or numeric module labels for all genes.
<code>unmergedColors</code>	a vector of color or numeric module labels for all genes before module merging.
<code>MEs</code>	a data frame containing module eigengenes of the found modules (given by <code>colors</code>).
<code>goodSamples</code>	numeric vector giving indices of good samples, that is samples that do not have too many missing entries.
<code>goodGenes</code>	numeric vector giving indices of good genes, that is genes that do not have too many missing entries.
<code>dendrograms</code>	a list whose components contain hierarchical clustering dendrograms of genes in each block.
<code>TOMfiles</code>	if <code>saveTOMs==TRUE</code> , 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.
<code>blockGenes</code>	a list whose components give the indices of genes in each block.
<code>blocks</code>	if input <code>blocks</code> 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 necessarily sorted in the order in which the blocks were processed (since we do not require this for the input <code>blocks</code>). See <code>blockOrder</code> below.
<code>blockOrder</code>	a vector giving the order in which blocks were processed and in which <code>blockGenes</code> above is returned. For example, <code>blockOrder[1]</code> contains the label of the first-processed block.
<code>MEsOK</code>	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	<i>Check adjacency matrix</i>
-------------	-------------------------------

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

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

Value

A list with components

<code>nSets</code>	Number of sets (length of the vector <code>data</code>).
<code>nGenes</code>	Number of columns in the <code>data</code> components in the lists. This number must be the same for all sets.
<code>nSamples</code>	A vector of length <code>nSets</code> giving the number of rows in the <code>data</code> components.
<code>structureOK</code>	Only set if the argument <code>checkStructure</code> equals <code>TRUE</code> . The value is <code>TRUE</code> if the parameter <code>data</code> passes a few tests of its structure, and <code>FALSE</code> 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	<i>Clustering coefficient calculation</i>
-------------	---

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	<i>Iterative garbage collection.</i>
----------------	--------------------------------------

Description

Performs garbage collection until free memory indicators show no change.

Usage

```
collectGarbage()
```

Value

None.

Author(s)

Steve Horvath

colQuantileC *Fast column-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 currently 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](#)

consensusMEDissimilarity
Consensus dissimilarity of module eigengenes.

Description

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

Usage

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

Arguments

<code>MEs</code>	Module eigengenes of the same modules in several sets.
<code>useAbs</code>	Controls whether absolute value of correlation should be used instead of correlation in the calculation of dissimilarity.
<code>useSets</code>	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 <code>NULL</code> , all sets will be used.
<code>method</code>	A character string giving the method to use. Allowed values are (abbreviations of) <code>"consensus"</code> and <code>"majority"</code> . The consensus dissimilarity is calculated as the minimum of given set dissimilarities for <code>"consensus"</code> and as the average for <code>"majority"</code> .

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 input 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⟩](mailto: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

<code>MEs</code>	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 <code>data</code> , that is <code>MEs[[set]]\$data[sample, module]</code> is the expression of the eigengene of module <code>module</code> in sample <code>sample</code> in dataset <code>set</code> . The number of samples can be different between the sets, but the modules must be the same.
<code>useAbs</code>	Controls whether vector similarity should be given by absolute value of correlation or plain correlation.
<code>useSets</code>	Allows the user to specify for which sets the eigengene ordering is to be performed.
<code>greyLast</code>	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 <code>FALSE</code> .
<code>greyName</code>	Name of the grey module eigengene.
<code>method</code>	A character string giving the method to be used calculating the consensus dissimilarity. Allowed values are (abbreviations of) <code>"consensus"</code> and <code>"majority"</code> . The consensus dissimilarity is calculated as the maximum of given set dissimilarities for <code>"consensus"</code> and as the average for <code>"majority"</code> .

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

<code>multiExpr</code>	expression data in the multi-set format (see checkSets). A vector of lists, one per set. Each set must contain a component <code>data</code> that contains the expression data, with rows corresponding to samples and columns to genes or probes.
<code>preferredSize</code>	preferred maximum size of clusters.
<code>nCenters</code>	number of initial clusters.
<code>sizePenaltyPower</code>	parameter specifying how severe is the penalty for clusters that exceed <code>preferredSize</code> .
<code>networkType</code>	network type. Allowed values are (unique abbreviations of) "unsigned", "signed", "signed hybrid". See adjacency .
<code>randomSeed</code>	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.
<code>checkData</code>	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.
<code>useMean</code>	logical: should mean distance across sets be used instead of maximum? See details.
<code>maxIterations</code>	maximum iterations to be attempted.
<code>verbose</code>	integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
<code>indent</code>	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 $\max(\text{clusterSize}/\text{preferredSize}, 1)^{\text{sizePenaltyPower}}$, 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 iterations 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:

<code>clusters</code>	a numerical vector with one component per input gene, giving the cluster number in which the gene is assigned.
<code>centers</code>	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.
<code>unmergedClusters</code>	a numerical vector with one component per input gene, giving the cluster number in which the gene was assigned before the final merging step.
<code>unmergedCenters</code>	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](#)

`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'

...

Warning

....

Note

further notes

Make other sections like Warning with

0.2 Warning

....

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 (corPrediction, corTestSet, topNumber = 100)
{
  nPredictors = dim(as.matrix(corPrediction))[[2]]
  nGenes = dim(as.matrix(corPrediction))[[1]]
  if (length(as.numeric(corTestSet)) != nGenes)
    stop("non-compatible dimensions of 'corPrediction' and 'corTestSet'")
  out1 = rep(NA, nPredictors)
  meancorTestSetPositive = matrix(NA, ncol = nPredictors, nrow = length(topNumber))
  meancorTestSetNegative = matrix(NA, ncol = nPredictors, nrow = length(topNumber))
  for (i in c(1:nPredictors)) {
    rankpositive = rank(-as.matrix(corPrediction)[, i], ties.method = "first")
    ranknegative = rank(as.matrix(corPrediction)[, i], ties.method = "first")
    for (j in c(1:length(topNumber))) {
      meancorTestSetPositive[j, i] = mean(corTestSet[rankpositive <=
        topNumber[j]], na.rm = T)
      meancorTestSetNegative[j, i] = mean(corTestSet[ranknegative <=
        topNumber[j]], na.rm = T)
    }
  }
  meancorTestSetOverall = data.frame((meancorTestSetPositive -
    meancorTestSetNegative)/2)
  dimnames(meancorTestSetOverall)[[2]] = names(data.frame(corPrediction))
  meancorTestSetOverall = data.frame(topNumber = topNumber,
    meancorTestSetOverall)
  meancorTestSetPositive = data.frame(meancorTestSetPositive)
  dimnames(meancorTestSetPositive)[[2]] = names(data.frame(corPrediction))
  meancorTestSetPositive = data.frame(topNumber = topNumber,
    meancorTestSetPositive)
  meancorTestSetNegative = data.frame(meancorTestSetNegative)
  dimnames(meancorTestSetNegative)[[2]] = names(data.frame(corPrediction))
  meancorTestSetNegative = data.frame(topNumber = topNumber,
    meancorTestSetNegative)
  datout = list(meancorTestSetOverall = meancorTestSetOverall,
    meancorTestSetPositive = meancorTestSetPositive, meancorTestSetNegative = meancor
  datout
}

```

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

<code>cor</code>	A vector of correlation values whose corresponding p-values are to be calculated
<code>nSamples</code>	Number of samples from which the correlations were calculated
<code>twoSided</code>	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

<code>cor</code>	A vector of correlation values whose corresponding p-values are to be calculated
<code>nSamples</code>	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

<code>multiME</code>	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 <code>data</code> that is a data frame whose columns are consensus module eigengenes.
<code>setLabels</code>	names to be used for the sets represented in <code>multiME</code> .
<code>excludeGrey</code>	logical: exclude the 'grey' eigengene from preservation measure?
<code>greyLabel</code>	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

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.

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 returned numerical labels into colors for easier visualization.

`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	<i>Threshold for module merging</i>
-----------------	-------------------------------------

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

<code>adjMat</code>	adjacency matrix of the network to be exported.
<code>file</code>	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.
<code>weighted</code>	logical: should the exported network be weighted?
<code>threshold</code>	adjacency threshold for including edges in the output.
<code>probeToGene</code>	optional specification of a conversion between probe names (that label columns and rows of <code>adjacency</code>) 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 function 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);
```


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 <code>data</code> 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 <code>FALSE</code> 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 <code>FALSE</code> 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.

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

<code>datExpr</code>	expression data. A data frame in which columns are genes and rows are samples.
<code>useSamples</code>	optional specifications of which samples to use for the check. Should be a logical vector; samples whose entries are <code>FALSE</code> will be ignored for the missing value counts. Defaults to using all samples.
<code>useGenes</code>	optional specifications of genes for which to perform the check. Should be a logical vector; genes whose entries are <code>FALSE</code> will be ignored. Defaults to using all genes.
<code>minFraction</code>	minimum fraction of non-missing samples for a gene to be considered good.
<code>minNSamples</code>	minimum number of non-missing samples for a gene to be considered good.
<code>minNGenes</code>	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.
<code>verbose</code>	integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
<code>indent</code>	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](#)

`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

<code>multiExpr</code>	expression data in the multi-set format (see checkSets). A vector of lists, one per set. Each set must contain a component <code>data</code> that contains the expression data, with rows corresponding to samples and columns to genes or probes.
<code>minFraction</code>	minimum fraction of non-missing samples for a gene to be considered good.
<code>minNSamples</code>	minimum number of non-missing samples for a gene to be considered good.
<code>minNGenes</code>	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.
<code>verbose</code>	integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
<code>indent</code>	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 following 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.

`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 are 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 following components:

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

Author(s)

Peter Langfelder

See Also

[goodSamples](#), [goodGenes](#)

goodSamplesMS	<i>Filter samples with too many missing entries across multiple data sets</i>
---------------	---

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

<code>multiExpr</code>	expression data in the multi-set format (see checkSets). A vector of lists, one per set. Each set must contain a component <code>data</code> that contains the expression data, with rows corresponding to samples and columns to genes or probes.
<code>useSamples</code>	optional specifications of which samples to use for the check. Should be a logical vector; samples whose entries are <code>FALSE</code> will be ignored for the missing value counts. Defaults to using all samples.
<code>useGenes</code>	optional specifications of genes for which to perform the check. Should be a logical vector; genes whose entries are <code>FALSE</code> will be ignored. Defaults to using all genes.
<code>minFraction</code>	minimum fraction of non-missing samples for a gene to be considered good.
<code>minNSamples</code>	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.
<code>minNGenes</code>	minimum number of non-missing samples for a sample to be considered good.
<code>verbose</code>	integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
<code>indent</code>	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.

`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

<code>datExpr</code>	expression data. A data frame in which columns are genes and rows are samples.
<code>useSamples</code>	optional specifications of which samples to use for the check. Should be a logical vector; samples whose entries are <code>FALSE</code> will be ignored for the missing value counts. Defaults to using all samples.
<code>useGenes</code>	optional specifications of genes for which to perform the check. Should be a logical vector; genes whose entries are <code>FALSE</code> will be ignored. Defaults to using all genes.
<code>minFraction</code>	minimum fraction of non-missing samples for a gene to be considered good.
<code>minNSamples</code>	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.
<code>minNGenes</code>	minimum number of non-missing samples for a sample to be considered good.
<code>verbose</code>	integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
<code>indent</code>	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](#)

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 progress.

Usage

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

Arguments

<code>leadStr</code>	character string that will be printed before the actual progress number.
<code>trailStr</code>	character string that will be printed after the actual progress number.
<code>quiet</code>	can be used to silence the indicator for non-interactive sessions whose output is typically redirected to a file.
<code>newFrac</code>	new fraction of progress to be displayed.
<code>progInd</code>	an object of class <code>progressIndicator</code> 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 connectivity 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	<i>Barplot with text or color labels.</i>
----------------	---

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

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

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	<i>Produce a labeled heatmap plot</i>
----------------	---------------------------------------

Description

Plots a heatmap plot with color legend, row and column annotation, and optional text within the 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,
  colors = NULL,
  textMatrix = NULL,
  cex.text = NULL, cex.lab = NULL,
  plotLegend = TRUE, ...)
```

Arguments

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

<code>yColorLabels</code>	logical: should <code>yLabels</code> be interpreted as colors?
<code>checkColorsValid</code>	logical: should given colors be checked for validity against the output of <code>colors()</code> ? If this argument is <code>FALSE</code> , invalid color specification will trigger an error.
<code>invertColors</code>	logical: should the color order be inverted?
<code>setStdMargins</code>	logical: should standard margins be set before calling the plot function? Standard margins depend on <code>colorLabels</code> : 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.
<code>xLabelsPosition</code>	a character string specifying the position of labels for the columns. Recognized values are (unique abbreviations of) "top", "bottom".
<code>xLabelsAngle</code>	angle by which the column labels should be rotated.
<code>xLabelsAdj</code>	justification parameter for column labels. See <code>par</code> and the description of parameter "adj".
<code>colors</code>	color palette to be used in the heatmap. Defaults to <code>heat.colors</code> .
<code>textMatrix</code>	optional matrix of text entries of the same dimensions as <code>Matrix</code> .
<code>cex.text</code>	character expansion factor for <code>textMatrix</code> .
<code>cex.lab</code>	character expansion factor for text labels labeling the axes
<code>plotLegend</code>	logical: should a color legend be plotted?
<code>...</code>	other arguments to functions <code>image.plot</code> (for <code>plotLegend==TRUE</code>) or <code>heatmap</code> (for <code>plotLegend==FALSE</code>).

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");

```

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 = 1e-5)
```

Arguments

<code>source</code>	a vector or a matrix of reference labels. The labels may be numeric or character.
<code>reference</code>	a vector of reference labels.
<code>pThreshold</code>	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 <code>exprData</code> . 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.

<code>colorSeq</code>	Color labels to be used for relabeling. Defaults to the standard color order used in this package if <code>colors</code> are not numeric, and to integers starting from 1 if <code>colors</code> is numeric.
<code>getNewMEs</code>	Controls whether module eigengenes of merged modules should be calculated and returned.
<code>getNewUnassdME</code>	When doing module eigengene manipulations, the function does not normally calculate the eigengene of the 'module' of unassigned ('grey') genes. Setting this option to <code>TRUE</code> will force the calculation of the unassigned eigengene in the returned <code>newMEs</code> , but not in the returned <code>oldMEs</code> .
<code>useSets</code>	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.
<code>checkDataFormat</code>	If <code>TRUE</code> , the function will check <code>exprData</code> and <code>MEs</code> for correct multi-set structure. If single set data is given, it will be converted into a format usable for the function. If <code>FALSE</code> , incorrect structure of input data will trigger an error.
<code>unassdColor</code>	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.
<code>trapErrors</code>	Controls whether computational errors in calculating module eigengenes, their dissimilarity, and merging trees should be trapped. If <code>TRUE</code> , errors will be trapped and the function will return the input <code>colors</code> . If <code>FALSE</code> , errors will cause the function to stop.
<code>verbose</code>	Controls verbosity of printed progress messages. 0 means silent, up to (about) 5 the verbosity gradually increases.
<code>indent</code>	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

<code>colors</code>	Color labels for the genes corresponding to merged modules. The function attempts to mimic the mode of the input <code>colors</code> : if the input <code>colors</code> is numeric, character and factor, respectively, so is the output. Note, however, that if the function performs relabeling, a standard sequence of labels will be used: integers starting at 1 if the input <code>colors</code> is numeric, and a sequence of color labels otherwise (see <code>colorSeq</code> above).
---------------------	--

dendro	Hierarchical clustering dendrogram (average linkage) of the eigengenes of the most recently computed tree. If <code>iterate</code> was set <code>TRUE</code> , 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 <code>cutHeight</code> .
oldMEs	Module eigengenes of the original modules in the sets given by <code>useSets</code> .
newMEs	Module eigengenes of the merged modules in the sets given by <code>useSets</code> .
allOK	A boolean set to <code>TRUE</code> .
colors	A copy of the input colors.
allOK	a boolean set to <code>FALSE</code> .

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 corresponding 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,
                 verbose = 0, indent = 0)
```

Arguments

<code>expr</code>	Expression data for a single set in the form of a data frame where rows are samples and columns are genes (probes).
<code>colors</code>	A vector of the same length as the number of probes in <code>expr</code> , giving module color for all probes (genes). Color "grey" is reserved for unassigned genes.
<code>impute</code>	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 <code>impute.knn</code> and probes from the same module as the missing datum. The function <code>impute.knn</code> uses a fixed random seed giving repeatable results.
<code>nPC</code>	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 <code>min(nPC, 10)</code> . If given <code>nPC</code> is greater than 10, a warning is issued.
<code>align</code>	Controls whether eigengenes, whose orientation is undetermined, should be aligned with average expression (<code>align = "along average"</code> , the default) or left as they are (<code>align = ""</code>). Any other value will trigger an error.
<code>excludeGrey</code>	Should the improper module consisting of 'grey' genes be excluded from the eigengenes?
<code>grey</code>	Value of <code>colors</code> designating the improper module. Note that if <code>colors</code> 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 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.
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 <code>returnValidOnly==FALSE</code> , module eigengenes whose calculation failed have all components set to NA.
<code>averageExpr</code>	If <code>align == "along average"</code> , 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.
<code>varExplained</code>	A dataframe in which each column corresponds to a module, with the component <code>varExplained[PC, module]</code> giving the variance of module <code>module</code> explained by the principal component no. <code>PC</code> . The calculation is exact irrespective of the number of computed principal components. At most 10 variance explained values are recorded in this dataframe.
<code>nPC</code>	A copy of the input <code>nPC</code> .
<code>validMEs</code>	A boolean vector. Each component (corresponding to the columns in <code>data</code>) is <code>TRUE</code> if the corresponding eigengene is valid, and <code>FALSE</code> if it is invalid. Valid eigengenes include both principal components and their hubgene approximations. When <code>returnValidOnly==FALSE</code> , by definition all returned eigengenes are valid and the entries of <code>validMEs</code> are all <code>TRUE</code> .
<code>validColors</code>	A copy of the input <code>colors</code> with entries corresponding to invalid modules set to <code>grey</code> if given, otherwise 0 if <code>colors</code> is numeric and "grey" otherwise.
<code>allOK</code>	Boolean flag signalling whether all eigengenes have been calculated correctly, either as principal components or as the hubgene average approximation.
<code>allPC</code>	Boolean flag signalling whether all returned eigengenes are principal components.
<code>isPC</code>	Boolean vector. Each component (corresponding to the columns in <code>eigengenes</code>) is <code>TRUE</code> if the corresponding eigengene is the first principal component and <code>FALSE</code> if it is the hubgene approximation or is invalid.
<code>isHub</code>	Boolean vector. Each component (corresponding to the columns in <code>eigengenes</code>) is <code>TRUE</code> if the corresponding eigengene is the hubgene approximation and <code>FALSE</code> if it is the first principal component or is invalid.
<code>validAEs</code>	Boolean vector. Each component (corresponding to the columns in <code>eigengenes</code>) is <code>TRUE</code> if the corresponding module average expression is valid.
<code>allAEOK</code>	Boolean flag signalling whether all returned module average expressions contain valid data. Note that <code>returnValidOnly==TRUE</code> does not imply <code>allAEOK==TRUE</code> : 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 dendrogram 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 assignment 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](#)

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

<code>exprData</code>	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 <code>expr[[set]]\$data[sample, probe]</code> is the expression of probe <code>probe</code> in sample <code>sample</code> in dataset <code>set</code> . The number of samples can be different between the sets, but the probes must be the same.
<code>colors</code>	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.
<code>universalColors</code>	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 <code>color</code> .
<code>useSets</code>	If calculations are requested in (a) selected set(s) only, the set(s) can be specified here. Defaults to all sets.
<code>useGenes</code>	Can be used to restrict calculation to a subset of genes (the same subset in all sets). If given, <code>validColors</code> in the returned list will only contain colors for the genes specified in <code>useGenes</code> .
<code>impute</code>	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 <code>impute.knn</code> and probes from the same module as the missing datum. The function <code>impute.knn</code> 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 (<code>align = "along average"</code> , the default) or left as they are (<code>align = ""</code>). Any other value will trigger an error.
excludeGrey	Should the improper module consisting of 'grey' genes be excluded from the eigengenes?
grey	Value of <code>colors</code> or <code>universalColors</code> (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 <code>subHubs==FALSE</code> , the value of <code>trapErrors</code> 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 <code>subHubs</code> takes precedence in the sense that if <code>subHubs==TRUE</code> and <code>trapErrors==FALSE</code> , 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:

<code>data</code>	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., <code>MEturquoise</code> etc. Note that, when <code>trapErrors == TRUE</code> and <code>returnValidOnly==FALSE</code> , this data frame also contains entries corresponding to removed modules, if any. (<code>validMEs</code> below indicates which eigengenes are valid and <code>allOK</code> whether all module eigengenes were successfully calculated.)
<code>averageExpr</code>	If <code>align == "along average"</code> , a dataframe containing average normalized expression in each module. The columns are named by the corresponding color with an "AE" prepended, e.g., <code>AEturquoise</code> etc.
<code>varExplained</code>	A dataframe in which each column corresponds to a module, with the component <code>varExplained[PC, module]</code> giving the variance of module <code>module</code> explained by the principal component no. <code>PC</code> . This is only accurate if all principal components have been computed (input <code>nPC = NULL</code>). At most 5 principal components are recorded in this dataframe.
<code>nPC</code>	A copy of the input <code>nPC</code> .
<code>validMEs</code>	A boolean vector. Each component (corresponding to the columns in <code>data</code>) is <code>TRUE</code> if the corresponding eigengene is valid, and <code>FALSE</code> if it is invalid. Valid eigengenes include both principal components and their hubgene approximations. When <code>returnValidOnly==FALSE</code> , by definition all returned eigengenes are valid and the entries of <code>validMEs</code> are all <code>TRUE</code> .
<code>validColors</code>	A copy of the input colors (<code>universalColors</code> if set, otherwise <code>colors[, set]</code>) with entries corresponding to invalid modules set to <code>grey</code> if given, otherwise 0 if the appropriate input colors are numeric and "grey" otherwise.
<code>allOK</code>	Boolean flag signalling whether all eigengenes have been calculated correctly, either as principal components or as the hubgene approximation. If <code>universalColors</code> is set, this flag signals whether all eigengenes are valid in all sets.
<code>allPC</code>	Boolean flag signalling whether all returned eigengenes are principal components. This flag (as well as the subsequent ones) is set independently for each set.
<code>isPC</code>	Boolean vector. Each component (corresponding to the columns in <code>eigengenes</code>) is <code>TRUE</code> if the corresponding eigengene is the first principal component and <code>FALSE</code> if it is the hubgene approximation or is invalid.
<code>isHub</code>	Boolean vector. Each component (corresponding to the columns in <code>eigengenes</code>) is <code>TRUE</code> if the corresponding eigengene is the hubgene approximation and <code>FALSE</code> 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 <code>returnValidOnly==TRUE</code> does not imply <code>allAEOK==TRUE</code> : 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](#)

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 <code>data</code> 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](#)

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

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

networkConcepts	<i>Calculations of network concepts</i>
-----------------	---

Description

This functions calculates various network concepts (topological properties, network indices) of a network calculated from expression data.

Usage

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

Arguments

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

Value

A list containing the values of various network concepts. The component names indicate the meaning of each component.

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

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

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'
...	

Warning

....

Note

further notes

Make other sections like Warning with

0.3 Warning

....

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 (datExpr, datME, GS, oddPower = 3, blockSize = 1000,
  MinimumSampleSize = 4, addGS = TRUE)
{
  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]]
  GS.Weighted = rep(0, nGenes)
  if (dim(as.matrix(datExpr))[[1]] != dim(as.matrix(datME))[[1]])
    stop(paste("Expression data and the module eigengenes have different\n",
      "numbers of observations (arrays). Specifically:\n",
      "dim(as.matrix(datExpr))[[1]] != dim(as.matrix(datME))[[1]] "))
  if (dim(as.matrix(datExpr))[[2]] != length(GS))
    stop(paste("The number of genes in the expression data does not match\n",
      "the length of the genes significance variable. Specifically:\n",
      "dim(as.matrix(datExpr))[[2]] != length(GS)  "))
  nAvailable = apply(as.matrix(!is.na(datExpr)), 2, sum)
  ExprVariance = apply(as.matrix(datExpr), 2, var, na.rm = T)
  restrictGenes = nAvailable >= 4 & ExprVariance > 0
  numberUsefulGenes = sum(restrictGenes, na.rm = T)
  if (numberUsefulGenes < 3) {
    stop(paste("IMPORTANT: there are fewer than 3 useful genes. \n",
      "Violations: either fewer than 4 observations or they are constant.\n",
      "WGCNA cannot be used for these data. Hint: collect more arrays or input
    "))
  }
  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."))
      ESBatch = hubGeneSignificance(datKMEBatch^oddPower,
        GS^oddPower)
      if (nGenes == nMEs) {
```

```

        diag(datKMEBatch[index1, ]) = 0
        datKMEBatch[is.na(datKMEBatch)] = 0
        ESBatch[is.na(ESBatch)] = 0
    }
    GS.WeightedBatch = as.matrix(datKMEBatch)^oddPower **%
        as.matrix(ESBatch)
    GS.Weighted = GS.Weighted + GS.WeightedBatch
}
if (nMEs - nBlocks * blockSize > 0) {
    restindex = c((nBlocks * blockSize + 1):nMEs)
    datMEBatch = datME[, restindex]
    datKMEBatch = as.matrix(signedKME(datExpr, datMEBatch,
        outputColumnName = "MM."))
    ESBatch = hubGeneSignificance(datKMEBatch^oddPower, GS^oddPower)
    if (nGenes == nMEs) {
        diag(datKMEBatch[restindex, ]) = 0
        datKMEBatch[is.na(datKMEBatch)] = 0
        ESBatch[is.na(ESBatch)] = 0
    }
    GS.WeightedBatch = as.matrix(datKMEBatch)^oddPower **%
        ESBatch
    GS.Weighted = GS.Weighted + GS.WeightedBatch
}
GS.Weighted = GS.Weighted/nMEs
GS.Weighted[nAvailable < MinimumSampleSize] = NA
rankGS.Weighted = rank(-GS.Weighted, ties.method = "first")
rankGS = rank(-GS, ties.method = "first")
printFlush(paste("Proportion of agreement between GS.Weighted and GS:"))
for (i in c(10, 20, 50, 100, 200, 500, 1000)) {
    printFlush(paste("Top ", i, " list of genes: prop. of agreement = ",
        signif(sum(rankGS.Weighted <= i & rankGS <= i, na.rm = T)/i,
            3)))
}
if (mean(abs(GS.Weighted), na.rm = T) > 0) {
    GS.Weighted = GS.Weighted/mean(abs(GS.Weighted), na.rm = T) *
        mean(abs(GS), na.rm = T)
}
if (addGS)
    GS.Weighted = apply(data.frame(GS.Weighted, GS), 1, mean,
        na.rm = T)
datout = data.frame(GS.Weighted, GS)
datout
}

```

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, removedDiag = 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
removedDiag	Describe removedDiag 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'
...	

Warning

....

Note

further notes

Make other sections like Warning with

0.4 Warning

....

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, removedDiag = 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 & removedDiag) {
        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) an
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
}

```

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 representation 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 <code>naColor</code> . If a matrix, each column of the matrix is processed separately and the return value will be a matrix of colors.
signed	logical: should <code>x</code> be considered signed? If <code>TRUE</code> , 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 <code>FALSE</code> , the default palette ranges from white for minimum values to red for maximum values.
centered	logical. If <code>TRUE</code> and <code>signed==TRUE</code> , numeric value zero will correspond to the middle of the color palette. If <code>FALSE</code> or <code>signed==FALSE</code> , 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 <code>colors</code> .
colors	color palette to represent the given numbers.
naColor	color to represent missing values in <code>x</code> .

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	<i>Put close eigenvectors next to each other</i>
----------	--

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 <code>MEs[[set]]\$data[sample, module]</code> is the expression of the eigengene of module <code>module</code> in sample <code>sample</code> in dataset <code>set</code> . The number of samples can be different between the sets, but the modules must be the same.
greyLast	Normally the color <code>grey</code> is reserved for unassigned genes; hence the <code>grey</code> module is not a proper module and it is conventional to put it last. If this is not desired, set the parameter to <code>FALSE</code> .
greyName	Name of the <code>grey</code> 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 <code>useSets</code> (or the first set, if <code>useSets</code> 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 verbosity of printed progress messages. 0 means silent, nonzero verbose.
indent	A single non-negative integer controlling 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	<i>Calculate overlap of modules</i>
--------------	-------------------------------------

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

<code>labels1</code>	a vector containing module labels.
<code>labels2</code>	a vector containing module labels to be compared to <code>labels1</code> .

Value

A list with the following components:

<code>countTable</code>	a matrix whose rows correspond to modules (unique labels) in <code>labels1</code> and whose columns correspond to modules (unique labels) in <code>labels2</code> , giving the number of objects in the intersection of the two respective modules.
<code>pTable</code>	a matrix whose rows correspond to modules (unique labels) in <code>labels1</code> and whose columns correspond to modules (unique labels) in <code>labels2</code> , 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),
  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.
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.

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

[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",
  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 .

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:

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

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](#), [softConnectivity](#)

`plotClusterTreeSamples`

Annotated clustering dendrogram of microarray samples

Description

This function plots an annotated clustering dendrogram of microarray samples.

Usage

```
plotClusterTreeSamples(
  datExpr,
  y = NULL,
  traitLabels = 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

<code>datExpr</code>	a data frame containing expression data, with rows corresponding to samples and columns to genes. Missing values are allowed and will be ignored.
<code>y</code>	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.
<code>traitLabels</code>	labels to be printed next to the color rows depicting sample traits. Defaults to column names of <code>y</code> .
<code>main</code>	title for the plot.
<code>setLayout</code>	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.
<code>autoColorHeight</code>	logical: should the height of the color area below the dendrogram be automatically adjusted for the number of traits? Only effective if <code>setLayout</code> is TRUE.
<code>colorHeight</code>	Specifies the height of the color area under dendrogram as a fraction of the height of the dendrogram area. Only effective when <code>autoColorHeight</code> above is FALSE.
<code>dendroLabels</code>	dendrogram labels. Set to FALSE to disable dendrogram labels altogether; set to NULL to use row labels of <code>datExpr</code> .
<code>addGuide</code>	logical: should vertical "guide lines" be added to the dendrogram plot? The lines make it easier to identify color codes with individual samples.
<code>guideAll</code>	logical: add a guide line for every sample? Only effective for <code>addGuide</code> set TRUE.
<code>guideCount</code>	number of guide lines to be plotted. Only effective when <code>addGuide</code> is TRUE and <code>guideAll</code> is FALSE.
<code>guideHang</code>	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 <code>guideHang</code> to leave more space for the labels.
<code>cex.traitLabels</code>	character expansion factor for trait labels.

<code>cex.dendroLabels</code>	character expansion factor for dendrogram (sample) labels.
<code>marAll</code>	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 <code>par</code> , because the bottom margin of the dendrogram and the top margin of the color underneath are always zero.
<code>saveMar</code>	logical: save margins setting before starting the plot and restore on exit?
<code>abHeight</code>	optional specification of the height for a horizontal line in the dendrogram, see <code>abline</code> .
<code>abCol</code>	color for plotting the horizontal line.
<code>...</code>	other graphical parameters to <code>plot.hclust</code> .

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 continuous, 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, ...)
```


Arguments

<code>dendro</code>	A dendrogram such as returned by <code>hclust</code> .
<code>colors</code>	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.
<code>rowLabels</code>	Labels for the colorings given in <code>colors</code> . 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 <code>colors</code> . If not given, <code>names(colors)</code> will be used if available. If not, sequential numbers starting from 1 will be used.
<code>cex.rowLabels</code>	Font size scale factor for the row labels. See <code>par</code> .
<code>...</code>	Other parameters to be passed on to the plotting method (such as <code>main</code> 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,
  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

<code>dendro</code>	a hierarchical clustering dendrogram such as one produced by <code>hclust</code> .
<code>colors</code>	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.
<code>groupLabels</code>	Labels for the colorings given in <code>colors</code> . 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 <code>colors</code> . If not given, <code>names(colors)</code> will be used if available. If not, sequential numbers starting from 1 will be used.
<code>setLayout</code>	logical: should the plotting device be partitioned into a standard layout? If <code>FALSE</code> , the user is responsible for partitioning. The function expects two regions of the same width, the first one immediately above the second one.
<code>autoColorHeight</code>	logical: should the height of the color area below the dendrogram be automatically adjusted for the number of traits? Only effective if <code>setLayout</code> is <code>TRUE</code> .
<code>colorHeight</code>	Specifies the height of the color area under dendrogram as a fraction of the height of the dendrogram area. Only effective when <code>autoColorHeight</code> above is <code>FALSE</code> .
<code>dendroLabels</code>	dendrogram labels. Set to <code>FALSE</code> to disable dendrogram labels altogether; set to <code>NULL</code> to use row labels of <code>datExpr</code> .
<code>addGuide</code>	logical: should vertical "guide lines" be added to the dendrogram plot? The lines make it easier to identify color codes with individual samples.
<code>guideAll</code>	logical: add a guide line for every sample? Only effective for <code>addGuide</code> set <code>TRUE</code> .
<code>guideCount</code>	number of guide lines to be plotted. Only effective when <code>addGuide</code> is <code>TRUE</code> and <code>guideAll</code> is <code>FALSE</code> .
<code>guideHang</code>	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 <code>guideHang</code> to leave more space for the labels.
<code>cex.colorLabels</code>	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 splits 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 consensus 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,
```

```

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)?
- `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 triangle 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 assignment, 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

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

<code>multiExpr</code>	expression data in the multi-set format (see checkSets). A vector of lists, one per set. Each set must contain a component <code>data</code> that contains the expression data, with rows corresponding to samples and columns to genes or probes.
<code>useSets</code>	optional specification of sets to be used for the preservation calculation. Defaults to using all sets.
<code>useGenes</code>	optional specification of genes to be used for the preservation calculation. Defaults to all genes.
<code>corFnc</code>	character string containing the name of the function to calculate correlation. Suggested functions include <code>"cor"</code> and <code>"bicor"</code> .
<code>corOptions</code>	further argument to the correlation function.
<code>networkType</code>	a character string encoding network type. Recognized values are (unique abbreviations of) <code>"unsigned"</code> , <code>"signed"</code> , and <code>"signed hybrid"</code> .

<code>power</code>	soft thresholding power for network construction. Should be a number greater than 1.
<code>sampleLinks</code>	logical: should network connections be sampled (TRUE) or should all connections be used systematically (FALSE)?
<code>nLinks</code>	number of links to be sampled. Should be set such that <code>nLinks * nNeighbors</code> be several times larger than the number of genes.
<code>blockSize</code>	correlation calculations will be split into square blocks of this size, to prevent running out of memory for large gene sets.
<code>setSeed</code>	seed to be used for sampling, for repeatability. If a seed already exists, it is saved before the sampling starts and restored upon exit.
<code>weightPower</code>	power with which higher adjacencies will be weighted in weighted means
<code>verbose</code>	integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
<code>indent</code>	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 $[(A_{ij}^{(1)} + A_{ij}^{(2)})/2]^{weightPower}$. 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:

<code>pairwise</code>	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.
<code>complete</code>	a vector with one entry for each input gene containing the complete mean preservation of the adjacencies connecting the gene to all other genes.
<code>pairwiseWeighted</code>	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.
<code>completeWeighted</code>	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.
<code>pairwiseHyperbolic</code>	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.
<code>completeHyperbolic</code>	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 = ceiling(10 * ncol(datExpr)/preferredSize),
  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 are samples. NAs are allowed, but not too many.

`preferredSize` preferred maximum size of clusters.

`nCenters` number of initial clusters.

`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 $\max(\text{clusterSize}/\text{preferredSize}, 1)^{\text{sizePenaltyPower}}$, 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 iterations 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 are 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

<code>tab</code>	Describe <code>tab</code> here
<code>adjust</code>	Describe <code>adjust</code> here

Details

If necessary, more details than the description above

Value

Describe the value returned If it is a LIST, use

<code>comp1</code>	Description of ' <code>comp1</code> '
--------------------	---------------------------------------

<code>comp2</code>	Description of ' <code>comp2</code> '
--------------------	---------------------------------------

...

Warning

....

Note

further notes

Make other sections like Warning with

0.5 Warning

....

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(tab, adjust=TRUE)
{
  a <- 0; b <- 0; c <- 0; d <- 0; nn <- 0
  m <- nrow(tab);
  n <- ncol(tab);
  for (i in 1:m) {
    c<-0
    for(j in 1:n) {
      a <- a+.choosenew(tab[i,j],2)
      nj <- sum(tab[,j])
      c <- c+.choosenew(nj,2)
    }
    ni <- sum(tab[i,])
    b <- b+.choosenew(ni,2)
    nn <- nn+ni
  }
  if(adjust==T) {
    d <- .choosenew(nn,2)
    adrand <- (a-(b*c)/d)/(0.5*(b+c)-(b*c)/d)
    adrand
  } else {
    b <- b-a
    c <- c-a
    d <- .choosenew(nn,2)-a-b-c
    rand <- (a+d)/(a+b+c+d)
    rand
  }
}
```

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

<code>datExpr</code>	expression data. A data frame in which columns are genes and rows are samples. NAs are allowed, but not too many.
<code>goodSamples</code>	a logical vector specifying which samples are considered "good" for the analysis. See goodSamplesGenes .
<code>goodGenes</code>	a logical vector with length equal number of genes in <code>multiExpr</code> that specifies which genes are considered "good" for the analysis. See goodSamplesGenes .
<code>blocks</code>	specification of blocks in which hierarchical clustering and module detection should be performed. A numeric vector with one entry per gene of <code>multiExpr</code> giving the number of the block to which the corresponding gene belongs.
<code>TOMfiles</code>	a vector of character strings specifying file names in which the block-wise topological overlaps are saved.
<code>dendrograms</code>	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.
<code>corType</code>	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 <code>parwise.complete.obs</code> option.
<code>networkType</code>	network type. Allowed values are (unique abbreviations of) "unsigned", "signed", "signed hybrid". See adjacency .
<code>deepSplit</code>	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.
<code>detectCutHeight</code>	dendrogram cut height for module detection. See cutreeDynamic for more details.

<code>minModuleSize</code>	minimum module size for module detection. See cutreeDynamic for more details.
<code>maxCoreScatter</code>	maximum scatter of the core for a branch to be a cluster, given as the fraction of <code>cutHeight</code> relative to the 5th percentile of joining heights. See cutreeDynamic for more details.
<code>minGap</code>	minimum cluster gap given as the fraction of the difference between <code>cutHeight</code> and the 5th percentile of joining heights. See cutreeDynamic for more details.
<code>maxAbsCoreScatter</code>	maximum scatter of the core for a branch to be a cluster given as absolute heights. If given, overrides <code>maxCoreScatter</code> . See cutreeDynamic for more details.
<code>minAbsGap</code>	minimum cluster gap given as absolute height difference. If given, overrides <code>minGap</code> . See cutreeDynamic for more details.
<code>pamStage</code>	logical. If TRUE, the second (PAM-like) stage of module detection will be performed. See cutreeDynamic for more details.
<code>pamRespectsDendro</code>	Logical, only used when <code>pamStage</code> 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.
<code>minKMEtoJoin</code>	a number between 0 and 1. Genes with eigengene connectivity higher than <code>minKMEtoJoin</code> are automatically assigned to their closest module.
<code>minCoreKME</code>	a number between 0 and 1. If a detected module does not have at least <code>minModuleKMESize</code> genes with eigengene connectivity at least <code>minCoreKME</code> , the module is disbanded (its genes are unlabeled and returned to the pool of genes waiting for module detection).
<code>minCoreKMESize</code>	see <code>minCoreKME</code> above.
<code>minKMEtoStay</code>	genes whose eigengene connectivity to their module eigengene is lower than <code>minKMEtoStay</code> are removed from the module.
<code>reassignThreshold</code>	p-value ratio threshold for reassigning genes between modules. See Details .
<code>mergeCutHeight</code>	dendrogram cut height for module merging.
<code>impute</code>	logical: should imputation be used for module eigengene calculation? See moduleEigengenes for more details.
<code>trapErrors</code>	logical: should errors in calculations be trapped?
<code>numericLabels</code>	logical: should the returned modules be labeled by colors (FALSE), or by numbers (TRUE)?
<code>verbose</code>	integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
<code>indent</code>	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:

<code>colors</code>	a vector of color or numeric module labels for all genes.
<code>unmergedColors</code>	a vector of color or numeric module labels for all genes before module merging.
<code>MEs</code>	a data frame containing module eigengenes of the found modules (given by <code>colors</code>).
<code>MEsOK</code>	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

- | | |
|--------------------------|---|
| <code>multiExpr</code> | expression data in the multi-set format (see checkSets). A vector of lists, one per set. Each set must contain a component <code>data</code> that contains the expression data, with rows corresponding to samples and columns to genes or probes. |
| <code>goodSamples</code> | a list with one component per set. Each component is a logical vector specifying which samples are considered "good" for the analysis. See goodSamplesGenesMS . |
| <code>goodGenes</code> | a logical vector with length equal number of genes in <code>multiExpr</code> that specifies which genes are considered "good" for the analysis. See goodSamplesGenesMS . |
| <code>blocks</code> | specification of blocks in which hierarchical clustering and module detection should be performed. A numeric vector with one entry per gene of <code>multiExpr</code> 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 <code>parwise.complete.obs</code> 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 <code>minModuleSize</code> ?
maxCoreScatter	maximum scatter of the core for a branch to be a cluster, given as the fraction of <code>cutHeight</code> 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 <code>cutHeight</code> 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 <code>maxCoreScatter</code> . See cutreeDynamic for more details.
minAbsGap	minimum cluster gap given as absolute height difference. If given, overrides <code>minGap</code> . 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 <code>pamStage</code> 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 <code>minKMEtoJoin</code> are automatically assigned to their closest module.
minCoreKME	a number between 0 and 1. If a detected module does not have at least <code>minModuleKMESize</code> genes with eigengene connectivity at least <code>minCoreKME</code> , the module is disbanded (its genes are unlabeled and returned to the pool of genes waiting for module detection).

<code>minCoreKMESize</code>	see <code>minCoreKME</code> above.
<code>minKMEtoStay</code>	genes whose eigengene connectivity to their module eigengene is lower than <code>minKMEtoStay</code> are removed from the module.
<code>reassignThresholdPS</code>	per-set p-value ratio threshold for reassigning genes between modules. See Details .
<code>mergeCutHeight</code>	dendrogram cut height for module merging.
<code>impute</code>	logical: should imputation be used for module eigengene calculation? See moduleEigengenes for more details.
<code>trapErrors</code>	logical: should errors in calculations be trapped?
<code>numericLabels</code>	logical: should the returned modules be labeled by colors (<code>FALSE</code>), or by numbers (<code>TRUE</code>)?
<code>verbose</code>	integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
<code>indent</code>	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:

<code>colors</code>	module assignment of all input genes. A vector containing either character strings with module colors (if input <code>numericLabels</code> was unset) or numeric module labels (if <code>numericLabels</code> was set to <code>TRUE</code>). 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.

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

<code>n</code>	number of colors to be returned
<code>gamma</code>	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'
...
```

Warning

....

Note

further notes

Make other sections like Warning with

0.6 Warning

....

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 (corPredictionNew, corPredictionStandard, corTestSet,
  topNumber = 100)
{
  nPredictors = dim(as.matrix(corPredictionNew))[[2]]
  nGenes = dim(as.matrix(corPredictionNew))[[1]]
  if (length(as.numeric(corTestSet)) != nGenes)
    stop("non-compatible dimensions of 'corPrediction' and 'corTestSet'.")
  if (length(as.numeric(corTestSet)) != length(corPredictionStandard))
    stop("non-compatible dimensions of 'corTestSet' and 'corPredictionStandard'.")
  kruskalp = matrix(NA, nrow = length(topNumber), ncol = nPredictors)
  for (i in c(1:nPredictors)) {
    rankhighNew = rank(-as.matrix(corPredictionNew)[, i],
      ties.method = "first")
    ranklowNew = rank(as.matrix(corPredictionNew)[, i], ties.method = "first")
    for (j in c(1:length(topNumber))) {
      highCorNew = as.numeric(corTestSet[rankhighNew <=
        topNumber[j]])
      lowCorNew = as.numeric(corTestSet[ranklowNew <= topNumber[j]])
      highCorStandard = as.numeric(corTestSet[rank(-as.numeric(corPredictionStandard)
        ties.method = "first") <= topNumber[j]])
      lowCorStandard = as.numeric(corTestSet[rank(as.numeric(corPredictionStandard)
        ties.method = "first") <= topNumber[j]])
      signedCorNew = c(highCorNew, -lowCorNew)
      signedCorStandard = c(highCorStandard, -lowCorStandard)
      x1 = c(signedCorNew, signedCorStandard)
      Grouping = rep(c(2, 1), c(length(signedCorNew), length(signedCorStandard)))
      sign1 = sign(cor(Grouping, x1, use = "p"))
    }
  }
}
```



```

        if (sign1 == 0)
            sign1 = 1
        kruskalp[j, i] = kruskal.test(x = x1, g = Grouping)$p.value *
            sign1
    }
}
kruskalp[kruskalp < 0] = 1
kruskalp = data.frame(kruskalp)
dimnames(kruskalp)[[2]] = paste(names(data.frame(corPredictionNew)),
    ".kruskalP", sep = "")
kruskalp = data.frame(topNumber = topNumber, kruskalp)
kruskalp
}

```

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 (typically 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)

`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 weightheted 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

<code>multiME</code>	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 <code>data</code> that is a data frame whose columns are consensus module eigengenes.
<code>setLabels</code>	names to be used for the sets represented in <code>multiME</code> .
<code>excludeGrey</code>	logical: exclude the 'grey' eigengene from preservation measure?
<code>greyLabel</code>	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.
<code>method</code>	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 absolute 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 adjacency 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	<i>Signed eigengene-based connectivity</i>
-----------	--

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

<code>corMat</code>	a matrix of correlations or other measures of similarity.
<code>threshold</code>	threshold for connecting nodes: all nodes whose <code>corMat</code> 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.

`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	background 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.

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.

<code>backgroundNoise</code>	amount of background noise to be added to the simulated expression data.
<code>leaveOut</code>	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.
<code>nSubmoduleLayers</code>	number of layers of ordered submodules to be added. See details.
<code>nScatteredModuleLayers</code>	number of layers of scattered submodules to be added. See details.
<code>averageNGenesInSubmodule</code>	average number of genes in a submodule. See details.
<code>averageExprInSubmodule</code>	average strength of submodule expression vectors.
<code>submoduleSpacing</code>	a number giving submodule spacing: this multiple of the submodule size will lie between the submodule and the next one.
<code>verbose</code>	integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
<code>indent</code>	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:

<code>datExpr</code>	simulated expression data in a data frame whose columns correspond genes and rows to samples.
<code>setLabels</code>	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 <code>leaveOut</code>) are indicated as 0 here.
<code>allLabels</code>	simulated module assignment. Genes that belong to leftout modules (specified in <code>leaveOut</code>) are indicated by their would-be assignment here.
<code>labelOrder</code>	a vector specifying the order in which labels correspond to the given eigengenes, that is <code>labelOrder[1]</code> is the label assigned to module whose seed is <code>eigengenes[, 1]</code> 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/Supplemental>

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.

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, verbose = FALSE)
```

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	<i>Simulate a gene co-expression module</i>
----------------	---

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

<code>ME</code>	seed module eigengene.
<code>nGenes</code>	number of genes in the module to be simulated. Must be non-zero.
<code>nNearGenes</code>	number of genes to be simulated with low correlation with the seed eigengene.
<code>minCor</code>	minimum correlation of module genes with the eigengene. See details.
<code>maxCor</code>	maximum correlation of module genes with the eigengene. See details.
<code>corPower</code>	controls the dropoff of gene-eigengene correlation. See details.
<code>signed</code>	logical: should the genes be simulated as belonging to a signed network? If <code>TRUE</code> , all genes will be simulated to have positive correlation with the eigengene. If <code>FALSE</code> , a proportion given by <code>propNegativeCor</code> will be simulated with negative correlations of the same absolute values.
<code>propNegativeCor</code>	proportion of genes to be simulated with negative gene-eigengene correlations. Only effective if <code>signed</code> is <code>FALSE</code> .
<code>verbose</code>	integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
<code>indent</code>	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 drop-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/Supplemental>

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 *function to do ...*

Description

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

Usage

```
simulateMultiExpr(eigengenes, nGenes, modProportions, minCor = 0.5, maxCor = 1,
```

Arguments

eigengenes	Describe eigengenes here
nGenes	Describe nGenes here
modProportions	Describe modProportions here
minCor	Describe minCor here
maxCor	Describe maxCor here
corPower	Describe corPower here
backgroundNoise	Describe backgroundNoise here
leaveOut	Describe leaveOut here
signed	Describe signed here
propNegativeCor	Describe propNegativeCor here
nSubmoduleLayers	Describe nSubmoduleLayers here
nScatteredModuleLayers	Describe nScatteredModuleLayers here
averageNGenesInSubmodule	Describe averageNGenesInSubmodule here
averageExprInSubmodule	Describe averageExprInSubmodule here
submoduleSpacing	Describe submoduleSpacing here
verbose	Describe verbose here
indent	Describe indent 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'

...

Warning

....

Note

further notes

Make other sections like Warning with

0.7 Warning

....

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 (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)
{
  MSize = checkSets(eigengenes)
  NSets = MSize$nSets
  nMods = MSize$nGenes
```

```

nSamples = MSize$nSamples
nAllSamples = sum(nSamples)
d2 = length(modProportions) - 1
if (d2 != nMods)
  stop(paste("Incompatible numbers of modules in 'eigengenes' and 'modProportions'"))
d3 = dim(leaveOut)
if ((d3[1] != nMods) | (d3[2] != NSets))
  stop(paste("Incompatible dimensions of 'leaveOut' and set eigengenes."))
multiDatExpr = vector(mode = "list", length = NSets)
setLabels = NULL
allLabels = NULL
labelOrder = NULL
for (set in 1:NSets) {
  SetEigengenes = scale(eigengenes[[set]]$data)
  setLeaveOut = leaveOut[, set]
  SetMinCor = rep(minCor, nMods)
  SetMaxCor = rep(maxCor, nMods)
  SetLO = c(1:nMods)[setLeaveOut]
  setData = simulateDatExpr(SetEigengenes, nGenes, modProportions,
    minCor = SetMinCor, maxCor = SetMaxCor, corPower = corPower,
    signed = signed, propNegativeCor = propNegativeCor,
    backgroundNoise = backgroundNoise, leaveOut = SetLO,
    nSubmoduleLayers = nSubmoduleLayers, nScatteredModuleLayers = nScatteredModuleLayers,
    averageNGenesInSubmodule = averageNGenesInSubmodule,
    averageExprInSubmodule = averageExprInSubmodule,
    submoduleSpacing = submoduleSpacing, verbose = verbose -
      1, indent = indent + 1)
  multiDatExpr[[set]] = list(data = setData$datExpr)
  setLabels = cbind(setLabels, setData$setLabels)
  allLabels = cbind(allLabels, setData$allLabels)
  labelOrder = cbind(labelOrder, setData$labelOrder)
}
list(multiDatExpr = multiDatExpr, setLabels = setLabels,
  allLabels = allLabels, labelOrder = labelOrder)
}

```

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

<code>order</code>	a vector giving the simulation order for vectors. See details.
<code>nSamples</code>	integer giving the number of samples to be simulated.
<code>minCor</code>	a multiple of <code>maxCor</code> (see below) giving the minimum correlation of module genes with the corresponding eigengene. See details.
<code>maxCor</code>	maximum correlation of module genes with the corresponding eigengene. See details.
<code>corPower</code>	controls the dropoff of gene-eigengene correlation. See details.
<code>averageModuleSize</code>	average number of genes in a module. See details.
<code>averageExpr</code>	average strength of module expression vectors.
<code>moduleSpacing</code>	a number giving module spacing: this multiple of the module size will lie between the module and the next one.
<code>verbose</code>	integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
<code>indent</code>	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	<i>Opens a graphics window with specified dimensions</i>
--------------	--

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 height of the window, in inches.

Value

None.

Author(s)

Peter Langfelder

softConnectivity	<i>Calculates connectivity of a weighted network.</i>
------------------	---

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,  
  power = 6,  
  blockSize = 1500,  
  minNSamples = 10,  
  corFnc = "cor", corOptions = "use = 'p'",  
  verbose = 2, indent = 0)
```

Arguments

<code>datExpr</code>	a data frame containing the expression data, with rows corresponding to samples and columns to genes.
<code>power</code>	soft thresholding power.
<code>blockSize</code>	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.
<code>minNSamples</code>	minimum number of samples available for the calculation of adjacency for the adjacency to be considered valid. If the number of samples falls below this threshold, the adjacency will be excluded from the connectivity calculation.
<code>corFnc</code>	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.
<code>corOptions</code>	character string giving further options to be passed to the correlation function.
<code>verbose</code>	integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
<code>indent</code>	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](#)

`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);
```

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.

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",  
  verbose = 1, indent = 0)
```

Arguments

<code>datExpr</code>	expression data. A data frame in which columns are genes and rows are samples. NAs are allowed, but not too many.
<code>corType</code>	character string specifying the correlation to be used. Allowed values are (unique abbreviations of) "pearson" and "bicor", corresponding to Pearson and biweight midcorrelation, respectively. Missing values are handled using the <code>pairwise.complete.obs</code> option.
<code>networkType</code>	network type. Allowed values are (unique abbreviations of) "unsigned", "signed", "signed hybrid". See adjacency .
<code>power</code>	soft-thresholding power for network construction.
<code>TOMType</code>	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.
<code>verbose</code>	integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
<code>indent</code>	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](#)

TOMsimilarity *Topological overlap matrix similarity and dissimilarity*

Description

Calculation of the topological overlap matrix from a given adjacency matrix.

Usage

```
TOMsimilarity(adjMat, TOMType = "unsigned", verbose = 1, indent = 0)
TOMdist(adjMat, TOMType = "unsigned", 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.
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](#)

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

<code>datExpr</code>	expression data. A data frame in which columns are genes and rows are samples. Missing values are ignored.
<code>datExpr2</code>	optional specification of a second set of expression data. See details.
<code>power</code>	soft-thresholding power for network construction.
<code>corFnc</code>	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.
<code>corOptions</code>	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 correlation of columns in `datExpr`.

Value

Adjacency matrix of dimensions $n \times n$, 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](#)

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 <code>\vect</code> contains a subset of <code>datExpr</code> .
blockSize	maximum block size for correlation calculations. Only important if <code>vect</code> 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 $n \times n$, 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.

verboseBoxplot *function to do ...*

Description

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

Usage

```
verboseBoxplot(x, g, main = "", xlab = NA, ylab = NA, cex = 1, cex.axis = 1.5, c
```

Arguments

<code>x</code>	Describe <code>x</code> here
<code>g</code>	Describe <code>g</code> here
<code>main</code>	Describe <code>main</code> here
<code>xlab</code>	Describe <code>xlab</code> here
<code>ylab</code>	Describe <code>ylab</code> here
<code>cex</code>	Describe <code>cex</code> here
<code>cex.axis</code>	Describe <code>cex.axis</code> here
<code>cex.lab</code>	Describe <code>cex.lab</code> here
<code>cex.main</code>	Describe <code>cex.main</code> here
<code>ylim</code>	Describe <code>ylim</code> here
<code>...</code>	Describe <code>...</code> 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'
...	

Warning

....

Note

further notes

Make other sections like Warning with

0.8 Warning

....

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(x, g, main = "", xlab = NA, ylab = NA, cex = 1, cex.axis = 1.5,
        cex.lab = 1.5, cex.main = 1.5, ylim = -1, ...)
{
  if (is.na(xlab))
    xlab = as.character(match.call(expand.dots = FALSE)$x)
  if (is.na(ylab))
    ylab = as.character(match.call(expand.dots = FALSE)$g)
  p1 = signif(kruskal.test(x, factor(g))$p.value, 2)
  if (p1 < 5 * 10^(-22))
```

```

    p1 = "< 5e-22"
    boxplot(x ~ factor(g), notch = TRUE, varwidth = TRUE, main = paste(main,
      ", p =", as.character(p1)), xlab = xlab, ylab = ylab,
      cex = cex, cex.axis = cex.axis, cex.lab = cex.lab, cex.main = cex.main,
      ...)
  }

```

verboseScatterplot *function to do ...*

Description

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

Usage

```
verboseScatterplot(x, y, corFnc = "cor", corOptions = "use = 'p'", main = "", xl
```

Arguments

x	Describe x here
y	Describe y here
corFnc	Describe corFnc here
corOptions	Describe corOptions here
main	Describe main here
xlab	Describe xlab here
ylab	Describe ylab here
cex	Describe cex here
cex.axis	Describe cex.axis here
cex.lab	Describe cex.lab here
cex.main	Describe cex.main here
ylim	Describe ylim here
abline	Describe abline 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'
...	

Warning

....

Note

further notes

Make other sections like Warning with

0.9 Warning

....

Author(s)

who you are

References

put references to the literature/web site here

See Alsoobjects 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 (x, y, corFnc = "cor", corOptions = "use = 'p'", main = "",
  xlab = NA, ylab = NA, cex = 1, cex.axis = 1.5, cex.lab = 1.5,
  cex.main = 1.5, ylim = -1, ...)
{
  if (is.na(xlab))
    xlab1 = as.character(match.call(expand.dots = FALSE)$x)
  if (is.na(ylab))
    ylab1 = as.character(match.call(expand.dots = FALSE)$y)
  x = as.numeric(as.character(x))
  y = as.numeric(as.character(y))
  corExpr = parse(text = paste(corFnc, "(x, y, ", corOptions,
    ")"))
  cor1 = signif(eval(corExpr), 2)
  corpExpr = parse(text = paste("cor.test(x, y, ", corOptions,
    ")"))
  corp = signif(eval(corpExpr)$p.value, 2)
  if (corp < 10^(-20))
    corp = "<10^{-20}"
  if (length(ylim) == 2) {
    plot(x, y, main = paste(main, " cor=", cor1, " p=", corp,
      sep = ""), xlab = xlab, ylab = ylab, cex = cex, cex.axis = cex.axis,
      cex.lab = cex.lab, cex.main = cex.main, ylim = ylim,
      ...)
  }
}
```

```

    }
    else {
      plot(x, y, main = paste(main, " cor=", cor1, " p=", corp,
        sep = ""), col = as.character(col), xlab = xlab,
        ylab = ylab, cex = cex, cex.axis = cex.axis, cex.lab = cex.lab,
        cex.main = cex.main, ...)
    }
  }
}

```

WGCNA-package

Weighted Gene Co-Expression Network Analysis

Description

Functions necessary to perform Weighted Gene Co-Expression Network Analysis

Details

Package: WGCNA
 Version: 0.72
 Date: 2009-02-18
 Depends: R (>= 2.3.0), stats, fields, impute, grDevices, dynamicTreeCut (>= 1.20), qvalue, flashClust
 ZipData: no
 License: GPL (>= 2)
 URL: <http://www.genetics.ucla.edu/labs/horvath/CoexpressionNetwork/Rpackages/WGCNA/>

Index:

GTOMdist	Generalized Topological Overlap Measure
TOMdist	Topological overlap matrix dissimilarity
TOMplot	~~function to do ... ~~
TOMsimilarity	Topological overlap matrix similarity
TOMsimilarityFromExpr	Topological overlap matrix similarity
WGCNA-package	Weighted Gene Co-Expression Network Analysis
addErrorBars	Add error bars to a barplot.
addGrid	Add grid lines to an existing plot.
addGuideLines	Add vertical "guide lines" to a dendrogram plot
addTraitToMEs	Add trait information to multi-set module eigengene structure
adjacency	Calculate network adjacency
alignExpr	Align expression data with given vector
automaticNetworkScreening	~~function to do ... ~~
automaticNetworkScreeningGS	One-step automatic network gene screening with external gene significance
bicor	Biweight Midcorrelation
blockwiseConsensusModules	Find consensus modules across several datasets.
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 column-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
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 calculations
goodGenes	Filter genes with too many missing entries
goodGenesMS	Filter genes with too many missing entries across multiple data sets
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 missing entries across multiple data sets
goodSamplesMS	Filter samples with too many missing entries across multiple data sets
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
labels2colors	Convert numerical labels to colors
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
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
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.
scaleFreePlot	Visual check of scale-free topology
setCorrelationPreservation	Summary correlation preservation measure
sigmoidAdjacencyFunction	Sigmoid-type adjacency 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           ~~function to do ... ~~
softConnectivity       ~~function to do ... ~~
standardColors         Colors this library uses for labeling modules.
stdErr                 ~~function to do ... ~~
unsignedAdjacency      ~~function to do ... ~~
vectorTOM              ~~function to do ... ~~
verboseBoxplot         ~~function to do ... ~~
verboseScatterplot     ~~function to do ... ~~

```

Author(s)

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

Index

*Topic **kwd1**

- automaticNetworkScreening, 7
- corPredictionSuccess, 27
- networkScreening, 73
- networkScreeningGS, 71
- overlapTable, 80
- randIndex, 99
- relativeCorPredictionSuccess, 108
- verboseBoxplot, 134
- verboseScatterplot, 136

*Topic **kwd2**

- automaticNetworkScreening, 7
- corPredictionSuccess, 27
- networkScreening, 73
- networkScreeningGS, 71
- overlapTable, 80
- randIndex, 99
- relativeCorPredictionSuccess, 108
- verboseBoxplot, 134
- verboseScatterplot, 136

*Topic **cluster**

- consensusProjectiveKMeans, 25
- moduleNumber, 63
- projectiveKMeans, 96

*Topic **color**

- greenBlackRed, 45
- greenWhiteRed, 45
- labels2colors, 55
- redWhiteGreen, 107
- standardColors, 127

*Topic **hplot**

- addErrorBars, 1
- addGrid, 2
- addGuideLines, 3
- labeledBarplot, 51
- labeledHeatmap, 52
- plotClusterTreeSamples, 83
- plotColorUnderTree, 85
- plotDendroAndColors, 86
- plotEigengeneNetworks, 88
- plotMEpairs, 91

- plotModuleSignificance, 92
- plotNetworkHeatmap, 93

*Topic **misc**

- addTraitToMEs, 3
- adjacency, 4
- alignExpr, 5
- automaticNetworkScreeningGS, 6
- blockwiseConsensusModules, 10
- blockwiseModules, 16
- checkAdjMat, 20
- checkSets, 21
- clusterCoef, 22
- colQuantileC, 23
- consensusMEDissimilarity, 23
- consensusOrderMEs, 24
- corPvalueFisher, 29
- corPvalueStudent, 30
- correlationPreservation, 31
- cutreeStatic, 32
- cutreeStaticColor, 32
- displayColors, 33
- dynamicMergeCut, 34
- exportNetworkToCytoscape, 35
- exportNetworkToVisANT, 36
- fixDataStructure, 37
- goodGenes, 39
- goodGenesMS, 38
- goodSamples, 43
- goodSamplesGenes, 41
- goodSamplesGenesMS, 40
- goodSamplesMS, 42
- GTOMdist, 46
- hubGeneSignificance, 47
- Inline display of progress, 48
- intramodularConnectivity, 49
- keepCommonProbes, 50
- matchLabels, 56
- mergeCloseModules, 57
- moduleColor.getMEprefix, 59
- moduleEigengenes, 60
- multiSetMEs, 64

- nearestNeighborConnectivity, [68](#)
- nearestNeighborConnectivityMS, [67](#)
- networkConcepts, [70](#)
- normalizeLabels, [77](#)
- nPresent, [77](#)
- numbers2colors, [78](#)
- orderMEs, [79](#)
- pickHardThreshold, [81](#)
- pickSoftThreshold, [82](#)
- plotClusterTreeSamples, [83](#)
- plotModuleSignificance, [92](#)
- preservationNetworkConnectivity, [94](#)
- propVarExplained, [98](#)
- recutBlockwiseTrees, [100](#)
- recutConsensusTrees, [104](#)
- removeGreyME, [110](#)
- scaleFreePlot, [111](#)
- setCorrelationPreservation, [112](#)
- sigmoidAdjacencyFunction, [113](#)
- signedKME, [114](#)
- signumAdjacencyFunction, [115](#)
- simulateDatExpr, [117](#)
- simulateDatExpr5Modules, [115](#)
- simulateEigengeneNetwork, [119](#)
- simulateModule, [120](#)
- simulateMultiExpr, [122](#)
- simulateSmallLayer, [124](#)
- sizeGrWindow, [126](#)
- softConnectivity, [126](#)
- standardColors, [127](#)
- stdErr, [128](#)
- TOMplot, [129](#)
- TOMsimilarity, [131](#)
- TOMsimilarityFromExpr, [130](#)
- unsignedAdjacency, [132](#)
- vectorTOM, [133](#)
- *Topic package**
 - WGCNA-package, [138](#)
- *Topic robust**
 - bicor, [9](#)
- *Topic utilities**
 - collectGarbage, [22](#)
- abline, [85, 88](#)
- addErrorBars, [1](#)
- addGrid, [2](#)
- addGuideLines, [3](#)
- addTraitToMEs, [3](#)
- adjacency, [4, 12, 16, 17, 20, 26, 50, 68, 69, 82, 83, 94, 96, 97, 101, 105, 115, 127, 130, 133](#)
- alignExpr, [5](#)
- automaticNetworkScreening, [7](#)
- automaticNetworkScreeningGS, [6](#)
- barplot, [51, 93](#)
- bicor, [9](#)
- blockwiseConsensusModules, [10, 27, 104, 106, 107](#)
- blockwiseModules, [16, 100, 103](#)
- boxplot, [93](#)
- checkAdjMat, [20](#)
- checkSets, [4, 11, 21, 24–26, 31, 37, 38, 40, 43, 50, 51, 57, 64, 79, 89, 94, 104, 110](#)
- clusterCoef, [22](#)
- collectGarbage, [22](#)
- colors, [54](#)
- colQuantileC, [23](#)
- consensusMEDissimilarity, [23](#)
- consensusOrderMEs, [24, 80](#)
- consensusProjectiveKMeans, [14, 25](#)
- corPredictionSuccess, [27](#)
- corPvalueFisher, [29](#)
- corPvalueStudent, [30](#)
- correlationPreservation, [31](#)
- cutree, [32, 33, 63](#)
- cutreeDynamic, [7, 12, 13, 16, 17, 20, 86, 101–103, 105, 107](#)
- cutreeStatic, [32, 32, 33](#)
- cutreeStaticColor, [32](#)
- displayColors, [33](#)
- dist, [15, 85](#)
- dynamicMergeCut, [34](#)
- exportNetworkToCytoscape, [35](#)
- exportNetworkToVisANT, [35, 36](#)
- fisher.test, [80](#)
- fixDataStructure, [37](#)
- goodGenes, [39, 39, 41–43](#)
- goodGenesMS, [38, 41, 43](#)
- goodSamples, [39–42, 43, 43, 44](#)
- goodSamplesGenes, [20, 39, 40, 41, 41, 43, 44, 101](#)
- goodSamplesGenesMS, [16, 39, 40, 43, 104](#)
- goodSamplesMS, [39, 41, 42](#)
- greenBlackRed, [45](#)
- greenWhiteRed, [45](#)

- GTOMdist, 46
- hclust, 3, 16, 20, 32, 33, 63, 85–87, 91
- heat.colors, 53, 89
- heatmap, 53, 54, 129, 130
- help, 8, 28, 72, 74, 100, 109, 123, 135, 137
- hubGeneSignificance, 7, 47
- image.plot, 53, 54
- initProgInd(*Inline display of progress*), 48
- Inline display of progress, 48
- intramodularConnectivity, 49
- keepCommonProbes, 50
- labeledBarplot, 51, 91
- labeledHeatmap, 52, 91
- labels2colors, 55, 79
- layout, 129
- load, 13, 18
- matchLabels, 56, 80
- mergeCloseModules, 14, 16, 19, 20, 34, 57, 103, 106, 107
- moduleColor.getMEprefix, 59
- moduleEigengenes, 4, 13, 18, 25, 34, 57, 60, 60, 65, 67, 80, 98, 102, 106
- moduleNumber, 63
- multiSetMEs, 14, 25, 31, 64, 80, 107, 113
- nearestNeighborConnectivity, 68, 68
- nearestNeighborConnectivityMS, 67
- networkConcepts, 70
- networkScreening, 7, 73
- networkScreeningGS, 6, 7, 71
- normalizeLabels, 63, 77
- nPresent, 77
- numbers2colors, 78
- orderMEs, 24, 25, 79
- overlapTable, 80
- pairs, 91
- par, 2, 3, 53, 85, 86, 89
- pdf, 90
- pickHardThreshold, 81
- pickSoftThreshold, 82
- plot, 92
- plot.hclust, 85, 88
- plotClusterTreeSamples, 83
- plotColorUnderTree, 85, 88
- plotDendroAndColors, 85, 86, 86
- plotEigengeneNetworks, 88, 113
- plotMEpairs, 91
- plotModuleSignificance, 92
- plotNetworkHeatmap, 93
- postscript, 90
- preservationNetworkConnectivity, 94
- projectiveKMeans, 18, 27, 96
- propVarExplained, 98
- quantile, 23
- randIndex, 99
- recutBlockwiseTrees, 100
- recutConsensusTrees, 104
- redWhiteGreen, 89, 107
- relativeCorPredictionSuccess, 108
- removeGreyME, 110
- scaleFreePlot, 111
- setCorrelationPreservation, 112
- sigmoidAdjacencyFunction, 113
- signedKME, 114
- signumAdjacencyFunction, 82, 115
- simulateDatExpr, 116, 117, 122, 124, 125
- simulateDatExpr5Modules, 115, 119, 122
- simulateEigengeneNetwork, 119, 119, 122
- simulateModule, 116, 119, 120, 125
- simulateMultiExpr, 119, 122, 122
- simulateSmallLayer, 124
- sizeGrWindow, 126
- softConnectivity, 68, 69, 83, 111, 126
- standardColors, 32–34, 56, 127
- stdErr, 128
- svd, 62
- TOMdist (*TOMsimilarity*), 131
- TOMplot, 129
- TOMsimilarity, 16, 20, 94, 131, 131, 134
- TOMsimilarityFromExpr, 130, 132
- unsignedAdjacency, 132
- updateProgInd(*Inline display of progress*), 48
- vectorTOM, 133
- verboseBoxplot, 134
- verboseScatterplot, 136
- WGCNA (*WGCNA-package*), 138
- WGCNA-package, 138