Package 'DiPALM'

May 29, 2026

Type Package

Title Differential Pattern Analysis via Linear Modeling

Version 1.3

Date 2026-05-21

Description Individual gene expression patterns are encoded into a series of eigenvector patterns ('WGCNA' package). Using the framework of linear model-based differential expression comparisons ('limma' package), time-course expression patterns for genes in different conditions are compared and analyzed for significant pattern changes. For reference, see: Greenham K, Sartor RC, Zorich S, Lou P, Mockler TC and McClung CR. eLife. 2020 Sep 30;9(4). <doi:10.7554/eLife.58993>.

License GPL (>= 2)

Imports limma, WGCNA, ggplot2, pwalign, Biostrings, hashr

Suggests edgeR

NeedsCompilation no

Depends R (>= 4.1.0)

Repository CRAN

Date/Publication 2026-05-29 09:40:02 UTC

Author Ryan C. Sartor [aut, cre],
Kathleen Greenham [aut],
William Gustafson [aut]

Maintainer Ryan C. Sartor <sartorry@gmail.com>

Contents

`DiPALM-package` *DiPALM - Differential Pattern Analysis via Linear Modeling*

Description

Individual gene expression patterns are encoded into a series of eigenvector patterns (WGCNA package). Using the framework of linear model-based differential expression comparisons (limma package), time-course expression patterns for genes in different conditions are compared and analyzed for significant pattern changes.

Author(s)

Ryan C. Sartor & Kathleen Greenham

Maintainer: Ryan C. Sartor <sartorry@gmail.com>

See Also

`limma` package `blockwiseModules` from the WGCNA package

`AdjustPvalue` *Determine p-values based on permutation test results*

Description

Used to calculate the false discovery rate associated with a specified cutoff. It uses a vector of actual test scores and a vector of permuted scores. The result can be considered a corrected p-value or q-value.

Usage

```
AdjustPvalue(tVal, tVec, pVec)
```

Arguments

tVal	The threshold or cutoff value to use (given this value, the function returns the FDR associated with using any score above this one as a positive result).
tVec	A numeric vector of actual test result values (used to determine true positives).
pVec	A numeric vector of permuted test result values (used to determine false positives).

Details

This function assumes all test result values above the given threshold are true positives (TP) and all permuted result values above the threshold are false positives (FP). It calculates FDR as (FP)/(TP). When applied to individual test results, this works to estimate an adjusted p-value (or q-value) for that result.

Value

A numeric value representing the FDR associated with the given cutoff (tVal).

Author(s)

Ryan C. Sartor

Examples

```
# Returns a vector of adjusted p-values from a vector of test results
data("testData")
AdjPval_kMEs<-sapply(testData$testResults[1:100],function(x)
AdjustPvalue(tVal = x, tVec = testData$testResults, pVec = testData$permutedResults))
```

BlankPlot *Generate a blank plot window*

Description

Creates a blank plotting window of desired size.

Usage

```
BlankPlot(xrng = c(0, 1), yrng = c(0, 1), Main = "", xlab = "", ylab = "", ...)
```

Arguments

xrng	The range of the x-axis.
yrng	The range of the y-axis.
Main	The main plot title.
xlab	X-axis label.
ylab	Y-axis label.
...	Additional arguments passed to the plot function.

Details

This function is used to initiate a blank plotting window in R. xrng and yrng are used to initiate the size of the plotting region.

Value

Generates a blank plot

Author(s)

Ryan C. Sartor

See Also

plot

Examples

```
BlankPlot(xrng=c(0,10),yrng=c(0,10),Main="Test",xlab="xlab",ylab="ylab")
```

`BuildLimmaLM` *Fit linear models to time-course expression patterns using the limma package*

Description

This function fits a linear model (using `lmFit` form the limma package) to each set of kME values returned from `BuildModMembership`.

Usage

```
BuildLimmaLM(dataMat, designMat, contrastStr)
```

Arguments

`dataMat`	a matrix containing module membership values (a single element of the list returned by `BuildModMembership`). Module membership is a Pearson correlation value for a single gene compared to a module eigengene.
`designMat`	a linear model design matrix containing binary values (0 and 1) with rows as samples and columns as treatments. Use the `model.matrix` function in the stats package.
`contrastStr`	a linear model contrast specifying the desired differential comparison to be made using the linear model (see the `limma` package documentation for more details).

Details

This function uses the limma package functionality to carry out an analysis of differential patterning. Asking if a gene displays a significantly different expression pattern across time when looking at one condition compared to another. The concept of differential patterning is similar to differential expression. However, the expression level of most genes fluctuates significantly throughout the day. Most of this is due to diurnal or circadian rhythms. These regular changes in expression severely complicate differential expression analysis and give rise to the need for examining differential patterning. By "encoding" the expression pattern of each gene into a vector of module membership (kME) values using the `BuildModMembership` function, the DiPALM analysis provides a way to use the limma method to examine differential patterning.

Value

This function returns an `MArrayLM-class` object. This is a custom class from the `limma` package. It is an object that contains all information and results of individual linear models built for each gene in the input matrix.

Author(s)

Ryan C. Sartor

See Also

`BuildLimmaLM` , `lmFit` , `makeContrasts` , `contrasts.fit` , `eBayes`

Examples

```
data("testData")
# First calculate the module memberships
kMEsList<-BuildModMembership(MeMat=testData$moduleEigengenes, TCsLst=testData$timeCourseList)

# Then build the limma models
LimmaMods<-lapply(kMEsList, function(x)
BuildLimmaLM(dataMat = x, designMat = testData$modelDesign, contrastStr = testData$modelContrast))
```

BuildModMembership *Calculate module membership (kME) for all genes in a time-course data set*

Description

This function calculates module membership (kME) between each gene's expression vector throughout a time-course and the module eigengenes of the data set (from the WGCNA package, using `blockwiseModules`). Module membership is defined as the Pearson correlation between the expression of a single gene and the expression of a single eigengene.

Usage

```
BuildModMembership(MeMat, TCsLst)
```

Arguments

MeMat	A data frame of eigengenes with each column being a different eigengene vector.
TCsLst	A named list of time-course matrices. Each element of the list is a time-course expression matrix (genes as rows and time-points as columns).

Details

The matrix of eigengene vectors, MeMat (see blockwiseModules from the WGCNA package) is used as a convenient way to summarize all the unique expression patterns (across time) that are present in a time-course data set. Any large gene expression data set from a complex organism will contain tens of thousands of genes. However, extensive coordinated regulation is present in all biological systems. Because of this co-regulation, usually only dozens of distinct expression patterns exist and most genes can be categorized by one (or a few) of them. By using a relatively small set of descriptive expression vectors (eigengenes), we can "encode" the pattern of expression for any single gene into a length N vector where N is the number of eigengenes and the values of the vector are the Pearson correlation of the gene to each eigengene (kME). These kME values can then be compared using the limma package to determine differential patterning similar to how expression values are compared to determine differential expression. This function is the first step to calculate all kMEs for each gene in a list of time-course matrices relative to the set of eigengenes for the whole data set.

Value

This function returns a list of kME matrices. The list contains a separate element for each eigengene. Each element is a matrix with a row for each gene and a column for each member of the input TCsLst.

Author(s)

Ryan C. Sartor

See Also

blockwiseModules

Examples

```
data("testData")
kMEsList<-BuildModMembership(MeMat=testData$moduleEigengenes, TCsLst=testData$timeCourseList)
```

---

col_format_sub *Substitute values in for the time, treatment, and replicate portion of a column format.*

Description

Given a column format as a string this function substitutes each instance of '[TREATMENT]', '[REPLICATE]' and '[TIME]' with the corresponding given values.

Usage

```
col_format_sub(
col.format = "[TREATMENT].[REPLICATE].[TIME]",
tr = ".*",
re = ".*",
ti = ".*",
escape = TRUE
)
```

Arguments

col.format	A string used to describe the column format of your dataset whose samples are assumed to be organized along 3 axes: treatment, time and replicate. The column format uses placeholders '[TREATMENT]', '[REPLICATE]' and '[TIME]' to denote corresponding information. The placeholders in the format must be separated, for instance a column format of '[TREATMENT].[TIME][REPLICATE]' will not behave as expected; instead '[TREATMENT].[TIME].[REPLICATE]' would be appropriate. This defaults to STD_FORMAT.
tr	The value to substitute for the placeholder '[TREATMENT]'.
re	The value to substitute for the placeholder '[REPLICATE]'.
ti	The value to substitute for the placeholder '[TIME]'.
escape	Whether to escape regex special characters in the arguments tr, re and ti.

Value

A new string with the placeholders '[TREATMENT]', '[REPLICATE]' and '[TIME]' substituted.

Author(s)

William Gustafson

See Also

re_esc

Examples

```
column.names<-c(
'CTL.ZT1.1',
'CTL.ZT1.2',
'FRZ.ZT1.1',
'FRZ.ZT1.2',
'CTL.ZT4.1',
'CTL.ZT4.2',
'FRZ.ZT4.1',
'FRZ.ZT4.2'
)
col.format<-'[TREATMENT].ZT[TIME].[REPLICATE]'
#grab time 1 samples
```

```
time1.regex<-re_esc(col_format_sub(col.format=col.format, ti='1'));
column.names[grep(time1.regex,column.names)]

#remove replicate
time.trt.regex<-re_esc( col_format_sub(col.format=col.format,ti='(.*)',tr='(.*)') )
column.names<-sub(time.trt.regex,'\1.ZT\2',column.names)
```

`dipalm` *Run a single instance of the DiPALM pipeline with replicates permuted.*

Description

Applys the DiPALM pipeline to a given set of gene expression data with replicates permuted within each timepoint. The user may want to instead use the frontend function `main` which runs this function many times in parallel.

Usage

```
dipalm(
expr.file,
 wgcna.power,
 threads,
 col.format,
 ctrl.treatment,
 p.value,
 .log = open_log()
)
```

Arguments

`expr.file`	Name of a csv file containing the gene expression data. It is assumed the first column consists of rownames,which are indexed by genes, and the columns are indexed by samples.
`wgcna.power`	Softthresholding power used with WGCNA.
`threads`	Number of threads to use when creating WGCNA modules.
`col.format`	String describing the column format of the data. The column format uses placeholders `'[TREATMENT]'`, `'[REPLICATE]'` and `'[TIME]'` to denote the location of the corresponding fields in the format.
`ctrl.treatment`	Name of the control treatment.
`p.value`	A p-value threshold used for calling genes as exhibiting differential pattern of expression, with and without regarding magnitude.
`.log`	Function to write log statements to, e.g. the output of the function `open_log`.

Value

A list with the following components:

AdjkMEs	Vector of adjusted kME p-values which represent significance of differential pattern of expression with no regard for magnitude.
AdjMed	Vector of adjusted kMed p-values which represent significance of differential pattern of expression considering magnitude.
patternCor	A matrix of correlations between genes called significant with respect to the AdjkMEs. This correlation matrix is used downstream as a distance measure in hierarchical clustering.
args	The list of arguments this function was called with along with a hash of the gene expression matrix for logging purposes.

Author(s)

William Gustafson

Examples

```
expr.file<-tempfile();
data('smallTestData')
write.csv(smallTestData,file=expr.file);

dipalm(
expr.file,
7,
parallel::detectCores(),
'[TREATMENT].[REPLICATE].[TIME]',
'CTL',
0.05,
open_log()
);
```

---

exampleData *Example Data: Data for use with the DiPALM vignette*

Description

A list of data objects that are used to run through the DiPALM example vignette.

Usage

```
data("exampleData")
```

Format

A List of 3 objects:

1. $rawCounts - A data frame [42383 rows X 48 columns] containing raw counts for mRNA expression data on *Brassica rapa* R500.
   - rows : Each row represents the expression of one gene across many different samples.
   - columns: Each column represents a different sample. Both drought-treated and properly watered samples are present, with two replicates of each time-course. Tissue was sampled at 4-hour intervals and column names are encoded as follows: S[replicate number] [D-Drought / W-Watered] [Time - sample] Example: S2D10 is part of the second replcate of the drought data and is the 10th time-point (ZT13, 13th hour after dawn on day two).
2. $moduleEigengenes - A dataframe [12 rows and 90 columns] representing 90 different expression vectors that describe descrete expression patterns found in the whole dataset ($timeCourseList).
   - Rows: each row represents a single time-point.
   - Columns: each column represents a distinct expression pattern seen repeatedly in the full data.
3. $geneAnnotations - A data frame [42383 rows and 5 columns] giving the locations within the *Brassica rapa* R500 genome where each gene is found. This is in SAF format which is used by the Linux Subread software package.
   - rows: Each row represents a single gene.
   - $GeneID: (character string) Unique gene accession.
   - $Chr: (character string) Chromosome number.
   - $Start: (integer) The position where the gene starts (first basepair).
   - $End: (integer) The position where the gene ends (last basepair).
   - $Strand: (character) is the gene on the positive [+] or negative [-] strand.

Details

Raw count data was summarized by mapping RNA-seq counts to the R500 *Brassica rapa* genome using the hisat2 aligner. Mapped reads were counted using the featureCounts function from the Linux Subread software package.

Source

Data is from: Greenham et al., eLife 2017;6:e29655

`ggPlotMultiDensities` *Plot multiple density distributions*

Description

This function generates one or more smoothed density distributions using `ggplot2` functionality

Usage

```
ggPlotMultiDensities(denslist, main = "", xlab = "", ylab = "Normalized Frequency",
 scale = T, cols = c("red", "blue", "grey50", "black", "skyblue", "orange"),
 ledgx = "topright", ltype = NULL, lwidth = NULL, xrng = NULL, yrng = NULL, Ledge = T)
```

Arguments

`denslist`	A named list of populations. Each element is a vector of numbers. A separate density plot will be generated for each population.
`main`	The main plot title.
`xlab`	The x-axis label.
`ylab`	The y-axis label.
`scale`	When multiple densities are plotted, should they be scaled to represent the relative number of samples in each population.
`cols`	A vector of colors (one for each population in `denslist`).
`ledgx`	The x value from `legend` function. Determines where the legend is plotted.
`ltype`	A vector of integers specifying the line types to be used for each population (repeated if necessary for multiple populations).
`lwidth`	The line width to be used in the plot.
`xrng`	A numeric vector of 2, the x-axis range to be plotted.
`yrng`	A numeric vector of 2, the y-axis range to be plotted.
`Ledge`	A logical: should the legend be plotted?

Details

This function takes in a list of one or more numeric vectors. Each vector is considered a separate population and a smoothed density plot (similar to a histogram) will be generated for each and plotted together. The actual numeric y-axis values are meaningless. A density plot is normalized so that the total area under the curve is equal to 1. When scale = T, the total area under the density plots may not equal one, but instead the total area is scaled to the relative proportions of the total population sizes of each population.

Value

Generates multiple density plots on the same graph.

Author(s)

Ryan C. Sartor

See Also

`ggplot2`

Examples

```
data(testData)
require(ggplot2)

# Two populations of the same size, unscaled
ggPlotMultiDensities(denslist = list(Test=testData$testResults,Permuted=testData$permutedResults),
main = "Pattern Change Scores", xlab = "Differential Pattern Score",lwidth = 1)

# Two populations of different sizes, scaled
ggPlotMultiDensities(denslist = list(Test=testData$testResults[1:3000],
Permuted=testData$permutedResults), scale=TRUE , main = "Pattern Change Scores (Scaled)",
xlab = "Differential Pattern Score",lwidth = 1)
```

`joinOutputs` *Joins outputs from multiple invocations of the function* `dipalm`.

Description

Given a collection of RData files containing the output of multiple invocations of the function `dipalm` this function combines them into a single set of results.

Usage

```
joinOutputs(files, p, n, .log = open_log())
```

Arguments

`files`	A vector or list of RData files containing the outputs to be joined.
`p`	The p-value to use when calling significance.
`n`	Genes must be called significant in at least `n` iterations to be called significant in the return value.
`.log`	Function used to write log statements.

Value

A list with the following components.

`sig.genes`	Vector of genes called significant with respect to AdjkMEs at the given p-value in at least `n` iterations.
`sig.MedGenes`	Vector of genes called significant with respect to AdjkMEds at the given p-value in at least `n` iterations.
`patternCor`	Average of the matrices named `patternCor` that were returned by the individual invocations of the function `dipalm`.
`AdjkMEs`	Matrix of the AdjkME values. Columns correspond to the indiviual invocations of the function `dipalm` and rows correspond to genes.
`AdjMEd`	Matrix of the AdjkMEd values. Columns correspond to the indiviual invocations of the function `dipalm` and rows correspond to genes.

Author(s)

William Gustafson

Examples

```
expr.file<-tempfile();
dipalm.file.1<-tempfile();
dipalm.file.2<-tempfile();
data('smallTestData')
write.csv(smallTestData,file=expr.file);
tmp<-dipalm(
expr.file,
7,
parallel::detectCores(),
'[TREATMENT].[REPLICATE].[TIME]',
'CTL',
0.05,
open_log()
);
#save each component of the list returned by the function dipalm
do.call(save,as.list(c(names(tmp),envir=as.environment(tmp),file=dipalm.file.1)))
tmp<-dipalm(
expr.file,
7,
parallel::detectCores(),
'[TREATMENT].[REPLICATE].[TIME]',
'CTL',
0.05,
open_log()
);
#save each component of the list returned by the function dipalm
do.call(save,as.list(c(names(tmp),envir=as.environment(tmp),file=dipalm.file.2)))
results<-joinOutputs(c(dipalm.file.1,dipalm.file.2),0.05,2);
```

---

main *Runs the full DiPALM pipeline with replicate permutation in parallel.*

Description

The DiPALM pipeline uses timecourse gene expression data to test whether genes exhibit a differential pattern of expression between two treatments. The method is sensitive to the labeling of replicates which is typically arbitrary and thus unrelated between different timepoints. This function runs the pipeline a specified number of times in parallel and on each individual invocation of the pipeline permutes replicates separately within each timepoint.

Usage

```
main(
expr.file,
```

```
threads,
wgcna.power,
iterations,
p.value,
sig.ratio,
col.format = "[TREATMENT].[REPLICATE].[TIME]",
ctrl.treatment = c(),
file.patt = "%d.dipalm.RData",
tmpdir = ".",
logdir = "logs",
clear.old.files = FALSE
)
```

Arguments

`expr.file`	Name of a csv file containing gene expression data. Columns in the file represent samples and the column names should prescribe to the format specified in the argument `col.format`. Rows in the file represent genes.
`threads`	An integer vector of length one or two giving the number of processes to use for computation. The first entry specifies the number of separate processes used to run the DiPALM pipeline (i.e. to call the function `dipalm`). The second entry, which defaults to 1, is the number of threads used for the function `blockwiseModules`.
`wgcna.power`	Soft thresholding power used with the function `blockwiseModules`.
`iterations`	The number of iterations of the DiPALM pipeline to perform.
`p.value`	A p-value used to call significance of differential pattern of expression.
`sig.ratio`	A proportion specifying how many iterations a gene must be called significant in to be considered significant in the final results. For example, when `n=100` and `sig.ratio=0.95` a gene must be called significant in 95 of the individual iterations to be called significant in the final results.
`col.format`	A format string that uses placeholders to describe the column format. Samples are grouped by treatment, time and replicate and these groupings are denoted by the placeholders `'[TREATMENT]'`, `'[TIME]'` and `'[REPLICATE]'`. For example, the sample name `'R1_Drought_17'` fits the format string `'R[REPLICATE]_[TREATMENT]_[TIME]'`. Note, there must be at least one character separating each placeholder so that each field can be captured with a regular expression.
`ctrl.treatment`	Name of the control treatment.
`file.patt`	A pattern describing how the files storing the outputs of the individual iterations are named. In the pattern the string `'%d'` is a placeholder for the iteration number. If the string `'%d'` is not found in the file pattern it is appended to the beginning of the file pattern.
`tmpdir`	Directory to store the outputs of the individual iterations.
`logdir`	Directory to write log files to. These log files record some information which may be useful when debugging user code or other issues such as corrupted outputs. The name of logfiles include the function the log file was generated by,

including an iteration number for the function dipalm along with a timestamp in the format '%Y.%m.%d.%H.%M'.

clear.old.files
: Whether to remove any output files from previous invocations of this function meaning the files recording the results of the individual iterations of the DiPALM pipeline. Log files are never cleared. When this argument is false output files are kept only if the arguments match. As a special case, a hash of the matrix containing gene expression is also recorded. Thus, if the gene expression data is different between invocations of this function the output files will not be kept even if all arguments (including the file name for the gene expression data) are the same.

Value

Returns the result of calling joinOutputs on the files containing the iteration outputs. This is a list with the following values:

AdjkMEs
: Conjoined adjusted kME matrix. The matrix has columns indexed by the individual invocations of the function dipalm and rows indexed by genes.

AdjMed
: Conjoined adjusted kMEd matrix. This has the same form as the component AdjkMEs.

patternCor
: Conjoined pattern correlation matrix. Both rows and columns are indexed by the genes called significant with respect to kMEs by this function; that is, by genes that were called significant in at least n of the individual invocations of the function dipalm at a p-value equal to p. The entries are the average of the pattern correlations returned by the function dipalm.

sig.genes
: A vector of genes found to exhibit differential pattern of expression disregarding magnitude of expression (i.e. "kME" genes) in at least n of the individual invocations of the function dipalm at a p-value equal to the argument p.

sig.Medgenes
: A vector of genes found to exhibit differential pattern of expression when considering magnitude (i.e. "kMed" genes) using the same conventions as for the return value sig.genes.

Author(s)

William Gustafson

Examples

```
data("smallTestData");
expr.file<-tempfile();
dipalm.dir<-tempdir();
logdir<-tempdir();
write.csv(smallTestData,file=expr.file);

main(
expr.file,
2,
7,
```

```
2,
0.05,
0.9,
'[TREATMENT].[REPLICATE].[TIME]',
'CTL',
tmpdir=dipalm.dir,
logdir=logdir
);
```

`make_TCs` *Break a data matrix into time courses separated by treatment and replicate.*

Description

Given a data set split into treatment groups, replicates and timepoints this function separates the dataset by treatment groups and replicate labels.

Usage

```
make_TCs(
gene.expr,
time_order = NULL,
col.format = "[TREATMENT].[REPLICATE].[TIME]",
.log = open_log()
)
```

Arguments

`gene.expr`	Input data set as a `matrix` or `data.frame`. Rows should be indexed by genes and columns by samples.
`time_order`	Optional argument specifying the order of the timepoints in the returned matrices.
`col.format`	String describing the column format of the data. The column format uses placeholders `'[TREATMENT]'`, `'[REPLICATE]'` and `'[TIME]'` to denote the location of the corresponding fields in the format.
`.log`	A function to write log statements to such as the return value of `open_log`.

Value

A list indexed by all pairs of a treatment and a replicate label from the input dataset. Each component of the returned list is a matrix containing the samples from the corresponding treatment and replicate label pair. The column names of these matrices are the timepoints from the input dataset.

Author(s)

William Gustafson

See Also

open_log

Examples

```
gene.expr<-matrix(1:16,nrow=2)
colnames(gene.expr)<-c(
'CTL.1.0',
'FRZ.1.0',
'CTL.1.4',
'FRZ.1.4',
'CTL.2.0',
'FRZ.2.0',
'CTL.2.4',
'FRZ.2.4'
)
make_TCs(gene.expr,col.format='[TREATMENT].[REPLICATE].[TIME]',.log=open_log())
```

---

msub *Regex substitution applied to a vector or list.*

Description

A convenient frontend for the function sub that applies the function to each element of a vector or list and returns the result in the same form.

Usage

```
msub(regex, repl, ...)
```

Arguments

regex	Regular expression to use for matching.
repl	Replacement string.
...	The first variadic argument is the object to perform the regular expression substitution to. This can be a vector or list of strings. Any remaining arguments are passed to the function sub.

Value

A vector of strings resulting from applying the regular expression substutition to each component of the given vector.

Author(s)

William Gustafson

See Also

sub

Examples

```
msub("^CTL\\.(.*)\\.(.*)$","\\1.\\2",c("CTL.ZT1.1","FRZ.ZT1.1","CTL.ZT4.2","FRZ.ZT4.2"));
```

---

open_log *Open a log*

Description

Returns a function, which is a wrapper around the function cat, to be used for logging.

Usage

```
open_log(log_file = NULL)
```

Arguments

log_file	A character vector giving the name of a file to write the output of the log function to.

Value

A function that can be used to write to the log file passed to open_log. This function is a wrapper for cat that sets the argument file to the value passed for the parameter log_file. The function also writes a newline provided the argument sep was not set to be a newline (by default sep is a single space). Calling the returned function with NULL as the first argument will close the log file.

If the value of log_file is NULL the returned function will print to standard output instead of writing to a file.

Author(s)

William Gustafson

See Also

cat

Examples

```
logfile<-tempfile()
.log<-open_log(logfile)
.log('This string is written to the file `log`.');
.log(NULL); #close the file `log`
.log<-open_log()
.log('This string is written to standard output.');
```

PlotTCs *Plot multiple time-course expression patterns for one gene*

Description

A plotting function that generates a line plot for a single gene in multiple time-course expression sets. Each time-course is overlaid on the same plot.

Usage

```
PlotTCs(TClst, tgene, main = "", scale = TRUE, xlab = "Time",
    ylab = "", xAxsLabs = colnames(TClst[[1]]), ledgeX = "top",
   colAdj = 0.4, tcols = c("red", "red", "blue", "blue"), tltys = c(1, 1, 2, 2))
```

Arguments

TClst	A named list of time-course matrices. Each element of the list is a time-course expression matrix (Genes as rows and time points as columns).
tgene	A string of a single gene accession (found in the row names of one element of TClst).
main	Main figure title.
scale	Logical: should the expression values of each time-course be scaled and centered?
xlab	The X-axis label.
ylab	The Y-axis label.
xAxsLabs	A vector of names to be associated with each x-axis point.
ledgeX	The x value from legend function. Determines where the legend is plotted.
colAdj	Controls the alpha channel for the line colors using adjustcolor.
tcols	A vector of colors controlling the colors of the lines. Must list one color for each time-course in TClst.
tltys	A vector of integers controlling the line type (lty) of the lines. Must list one lty for each time-course in TClst.

Details

This function is used to view the expression pattern of a single gene across multiple time-course samples.

Value

The function generates a plot with multiple lines.

Author(s)

Ryan C. Sartor

Examples

```
data(testData)
PlotTCs(TClst = testData$timeCourseList, tgene = "BraA05g36370R" ,
scale = TRUE,tcols = c("red","red","blue","blue"), tltys = c(1,2,1,2), ledgeX = "topleft")
```

`PlotTCsRibbon` *Plot multiple averaged time-course expression patterns for a group of genes*

Description

A plotting function that generates a ribbon plot representing the summarized expression of a set of genes. A separate ribbon is generated for each time-course. The ribbon represents + & - 1 standard deviation from the mean at each point.

Usage

```
PlotTCsRibbon(TClst, tgenes, main = "", xlab = "Time", ylab = "",
 xAxsLabs = colnames(TClst[[1]]),scale = TRUE, alpha = 0.1, tcols, tltys,
 splits = rep(1, length(TClst)))
```

Arguments

`TClst`	A named list of time-course matrices. Each element of the list is a time-course expression matrix (Genes as rows and time points as columns).
`tgenes`	A vector of gene accessions (found in the row names of one element of TClst).
`main`	The main title of the plot, if splits are defined, multiple plots will be generated and a vector of titles may be defined here.
`xlab`	The X-axis label.
`ylab`	The Y-axis label.
`xAxsLabs`	The time point labels for the x-axis, by default they will be the column names of the first time-course in TClst.
`scale`	Logical: should the expression values of each time-course be scaled and centered?
`alpha`	The alpha channel to be used for the ribbons (number from 0 - 1 defining transparency). See `adjustcolor`.
`tcols`	A vector of colors controlling the colors of the lines. Must list one color for each time-course in TClst.
`tltys`	A vector of integers controlling the line type (lty) of the lines. Must list one lty for each time-course in TClst.
`splits`	A vector of integers defining groups to be plotted on separate plots if desired (see examples).

Details

This can be used to plot the summarized expression pattern of an entire cluster of genes and compare between multiple time courses. Line plots are generated representing the mean expression value at each time point for the group. Ribbons are plotted along the lines in semi-transparent colors. The ribbons represent + and - one standard deviation from the mean expression level of the group. #

Value

This function generates multiple line plots representing the average expression of a group of genes with ribbons representing standard deviation.

Author(s)

Ryan C. Sartor

Examples

```
data("testData")

# Calculate the module memberships
kMEsList<-BuildModMembership(MeMat=testData$moduleEigengenes, TCsLst=testData$timeCourseList)

# Build the limma models
LimmaMods<-lapply(kMEsList, function(x)
BuildLimmaLM(dataMat = x, designMat = testData$modelDesign, contrastStr = testData$modelContrast))

# Get the adjusted Pvalues for each gene
AdjPval_kMEs<-sapply(testData$testResults[rownames(LimmaMods[[1]])],function(x)
AdjustPvalue(tVal = x, tVec = testData$testResults, pVec = testData$permutedResults))

# Cluster the genes base on the limma results
LimmaSums<-do.call(cbind,lapply(LimmaMods,function(x) x$t))
LimmaModsSig<-LimmaSums[names(AdjPval_kMEs)[which(AdjPval_kMEs<0.01)],]
patternCor<-cor(t(LimmaModsSig))
patternTree<-hclust(as.dist(1-patternCor),method = "complete")
patternClusters<-cutree(tree = patternTree, k = 25)
patternClusters<-tapply(rownames(patternCor),INDEX = patternClusters,function(x) x)

#Replicates are grouped together (same color + lty values)
PlotTCsRibbon(TClst = testData$timeCourseList, main="Drought Time-Course",
xAxsLabs = c(seq(1,23,4),seq(1,23,4)), xlab="ZT Time (hours)", tgenes = patternClusters[[15]],
scale = TRUE, tcols = c("red","red","blue","blue"), tltys = c(1,1,1,1))

#Replicates are in different groups (same color but different lty values)
PlotTCsRibbon(TClst = testData$timeCourseList, main="Drought Time-Course",
xAxsLabs = c(seq(1,23,4),seq(1,23,4)), xlab="ZT Time (hours)", tgenes = patternClusters[[15]],
scale = TRUE, tcols = c("red","red","blue","blue"), tltys = c(1,2,1,2))

#Replicates are represented on differnt plots (using the splits argument)
PlotTCsRibbon(TClst = testData$timeCourseList, main=c("Rep1","Rep2"),
xAxsLabs = c(seq(1,23,4),seq(1,23,4)), xlab="ZT Time (hours)", tgenes = patternClusters[[15]],
scale = TRUE, tcols = c("red","red","blue","blue"), tltys = c(1,2,1,2), splits = c(1,2,1,2))
```

re_esc *Format a string as a regular expression literal.*

Description

Given a string returns escapes special characters used in regular expressions.

Usage

```
re_esc(s)
```

Arguments

s	String to be escaped.

Value

A string with special characters escaped.

Author(s)

William Gustafson

Examples

```
re_esc('x.csv')
re_esc('abc')
re_esc('a\\(.*)\\')
```

sampleReplicates *Samples replicates from a dataset without replacement.*

Description

Given a dataset as a `matrix` or `data.frame` whose columns are samples grouped by treatments, times and replicates this function permutes the replicates which are typically arbitrary labels. Treatments and times of each sample are fixed.

Usage

```
sampleReplicates(gene.expr, col.format, .log)
```

Arguments

gene.expr	A matrix or data.frame containing the data to be permuted, typically this is gene expression data from a timecourse.
col.format	A column format string describing the formatting of the column names. The format string uses placeholders, '[TREATMENT]', '[REPLICATE]' and '[TIME]'.
.log	A function, returned by open_log to log information that can be helpful in debugging user code.

Value

A matrix containing the permuted dataset.

Author(s)

William Gustafson

See Also

col_format_sub open_log

Examples

```
set.seed(132)
gene.expr<-matrix(1:24,nrow=3)
colnames(gene.expr)<-c(paste0('CTL.ZT1.',1:4),paste0('CTL.ZT4.',1:4))
sampleReplicates(gene.expr,'[TREATMENT].ZT[TIME].[REPLICATE]',open_log());
```

---

smallTestData *Small test data for the DiPALM pipeline.*

Description

A selection of 250 Arabidopsis Thaliana transcript profiles from a timecourse study on cold acclimation. When applying the DiPALM pipeline roughly 100 genes should be called significant at a p-value of 0.05. Furthermore, the genes called significant are expected to be concentrated within the last 125 genes.

Usage

```
data("smallTestData")
```

Format

A data frame with 250 observations on the 48 variables. Observations, which correspond to genes, are labeled by locus ids, e.g. AT5G64650. Variables, which correspond to samples, are labeled in the format [TREATMENT].[REPLICATE].[TIME]. There are two treatments, CTL and FRZ, four replicates 1,2,3,4 and 6 times 1,5,9,13,17,21.

Examples

```
file<-tempfile();
data("smallTestData")
write.csv(smallTestData, file=file)
results<-dipalm(file, 7, 2, "[TREATMENT].[REPLICATE].[TIME]","CTL",0.05);
sum(results$AdjkMEs<0.05);
## maybe str(x) ; plot(x) ...
```

`testData` *Test Data: Data for function testing*

Description

A list of objects that can be used to test individual DiPALM functions.

Usage

```
data("testData")
```

Format

A List of 6 objects:

1. $timeCourseList - A List of 4 matrices containing mRNA expression data on *Brassica rapa* R500.
   - $Drought.R1 (Plants exposed to drought, Replicate 1) A matrix of time-course gene expression data [18428 Genes X 12 Time-points]
   - $Drought.R2 (Plants exposed to drought, Replicate 2) A matrix of time-course gene expression data [18428 Genes X 12 Time-points]
   - $Watered.R1 (Properly watered plants, Replicate 1) A matrix of time-course gene expression data [18428 Genes X 12 Time-points]
   - $Watered.R2 (Properly watered plants, Replicate 2) A matrix of time-course gene expression data [18428 Genes X 12 Time-points]
2. $moduleEigengenes - A dataframe [12 rows and 90 columns] representing 90 different expression vectors that describe descrete expression patterns found in the whole dataset ($timeCourseList).
   - Rows: each row represents a single time-point
   - Columns: each column represents a distinct expression pattern seen repeatedly in the full data
3. $modelDesign - A matrix [4 rows and 2 columns] used as a design matrix for a simple linear model. This matrix provides a mapping between the input instances (timeCourseList) and the two fixed effects fitted in the mode (Drought and Watered). This matrix was created with `model.matrix`.
4. $modelContrast - A character string specifying the contrast to be evaluated using the fitted model for each gene. This example is used to compare watered to drought conditions.

5. $testResults - A named numeric vector with 18428 values from DiPALM test results on actual data.
    - names: Gene accessions
    - values: The summed absolute values of t-values generated from limma model contrasts related to each eigengene pattern. This can be thought of as a DiPALM score for differential patterning.
6. $permutedResults- A named numeric vector with 18428 values from DiPALM test results on permuted data. This vector acts as a null distribution in order to estimate significance of actual test results.
    - names: Gene accessions
    - values: The summed absolute values of t-values generated from limma model contrasts related to each eigengene pattern using permuted expression data. This can be thought of as a null distribution of DiPALM scores for differential patterning.

Source

Data is from: Greenham et al., eLife 2017;6:e29655

Index