Package 'puma'

October 29, 2025

Type Package

Title Propagating Uncertainty in Microarray Analysis(including Affymetrix tranditional 3' arrays and exon arrays and Human Transcriptome Array 2.0)

Version 3.51.0

Date 2015-7-29

Author Richard D. Pearson, Xuejun Liu, Magnus Rattray, Marta Milo, Neil D. Lawrence, Guido Sanguinetti, Li Zhang

Maintainer Xuejun Liu <xuejun.liu@nuaa.edu.cn>

Depends R (>= 3.2.0), oligo (>= 1.32.0),graphics,grDevices, methods, stats, utils, mclust, oligoClasses

Imports Biobase (>= 2.5.5), affy (>= 1.46.0), affyio, oligoClasses

Suggests pumadata, affydata, snow, limma, ROCR,annotate

Description Most analyses of Affymetrix GeneChip data (including tranditional 3' arrays and exon arrays and Human Transcriptome Array 2.0) are based on point estimates of expression levels and ignore the uncertainty of such estimates. By propagating uncertainty to downstream analyses we can improve results from microarray analyses. For the first time, the puma package makes a suite of uncertainty propagation methods available to a general audience. In additon to calculte gene expression from Affymetrix 3' arrays, puma also provides methods to process exon arrays and produces gene and isoform expression for alternative splicing study. puma also offers improvements in terms of scope and speed of execution over previously available uncertainty propagation methods. Included are summarisation, differential expression detection, clustering and PCA methods, together with useful plotting functions.

License LGPL

biocViews Microarray, OneChannel, Preprocessing, DifferentialExpression, Clustering, ExonArray, GeneExpression, mRNAMicroarray, ChipOnChip, AlternativeSplicing, DifferentialSplicing, Bayesian, TwoChannel, DataImport, HTA2.0

URL http://umber.sbs.man.ac.uk/resources/puma

NeedsCompilation yes

git_url https://git.bioconductor.org/packages/puma

git_branch devel

git_last_commit 6c57f13

git_last_commit_date 2025-04-15

Repository Bioconductor 3.22

Date/Publication 2025-10-28

Contents

---

puma-package *puma - Propagating Uncertainty in Microarray Analysis*

Description

Most analyses of Affymetrix GeneChip data (including tranditional 3' arrays and exon arrays) are based on point estimates of expression levels and ignore the uncertainty of such estimates. By propagating uncertainty to downstream analyses we can improve results from microarray analyses. For the first time, the puma package makes a suite of uncertainty propagation methods available to a general audience. In additon to calculte gene expression from Affymetrix 3' arrays, puma also provides methods to process exon arrays and produces gene and isoform expression for alternative splicing study. puma also offers improvements in terms of scope and speed of execution over previously available uncertainty propagation methods. Included are summarisation, differential expression detection, clustering and PCA methods, together with useful plotting functions.

Details

Package:	puma
Type:	Package
Version:	3.4.3
Date:	2013-11-04
License:	LGPL excluding donlp2

For details of using the package please refer to the Vignette

Author(s)

Richard Pearson, Xuejun Liu, Guido Sanguinetti, Marta Milo, Neil D. Lawrence, Magnus Rattray, Li Zhang

Maintainer: Richard Pearson <richard.pearson@postgrad.manchester.ac.uk>, Li Zhang <leo.zhang@nuaa.edu.cn>

References

Milo, M., Niranjan, M., Holley, M. C., Rattray, M. and Lawrence, N. D. (2004) A probabilistic approach for summarising oligonucleotide gene expression data, technical report available upon request.

Liu, X., Milo, M., Lawrence, N. D. and Rattray, M. (2005) A tractable probabilistic model for Affymetrix probe-level analysis across multiple chips, Bioinformatics, 21(18):3637-3644.

Sanguinetti, G., Milo, M., Rattray, M. and Lawrence, N. D. (2005) Accounting for probe-level noise in principal component analysis of microarray data, Bioinformatics, 21(19):3748-3754.

Rattray, M., Liu, X., Sanguinetti, G., Milo, M. and Lawrence, N. D. (2006) Propagating uncertainty in Microarray data analysis, Briefings in Bioinformatics, 7(1):37-47.

Liu, X., Milo, M., Lawrence, N. D. and Rattray, M. (2006) Probe-level measurement error improves accuracy in detecting differential gene expression, Bioinformatics, 22(17):2107-2113.

Liu, X. Lin, K., Andersen, B. Rattray, M. (2007) Including probe-level uncertainty in model-based gene expression clustering, BMC Bioinformatics, 8(98).

Pearson, R. D., Liu, X., Sanguinetti, G., Milo, M., Lawrence, N. D., Rattray, M. (2008) puma: a Bioconductor package for Propagating Uncertainty in Microarray Analysis, BMC Bioinformatics, 2009, 10:211.

Zhang,L. and Liu,X. (2009) An improved probabilistic model for finding differential gene expression, the 2nd BMEI 17-19 oct. 2009. Tianjin. China.

Liu,X. and Rattray,M. (2009) Including probe-level measurement error in robust mixture clustering of replicated microarray gene expression, Statistical Application in Genetics and Molecular Biology, 9(1), Article 42.

puma 3.0: improved uncertainty propagation methods for gene and transcript expression analysis, Liu et al. BMC Bioinformatics, 2013, 14:39.

Examples

```
# Next 4 lines commented out to save time in package checks, and saved version used
   # if (require(affydata)) {
# data(Dilution)
# eset_mmgmos <- mmgmos(Dilution)
# }
data(eset_mmgmos)
pumapca_mmgmos <- pumaPCA(eset_mmgmos)
plot(pumapca_mmgmos)
eset_mmgmos_100 <- eset_mmgmos[1:100,]
eset_comb <- pumaComb(eset_mmgmos_100)
       eset_combImproved <- pumaCombImproved(eset_mmgmos_100)
esetDE <- pumaDE(eset_comb)
       esetDEImproved <- pumaDE(eset_combImproved)
```

---

bcomb *Combining replicates for each condition*

Description

This function calculates the combined signal for each condition from replicates using Bayesian models. The inputs are gene expression levels and the probe-level standard deviation associated with expression measurement for each gene on each chip. The outputs include gene expression levels and standard deviation for each condition. This function was originally part of the **pplr** package. Although this function can be called directly, it is recommended to use the `pumaComb` function instead, which can work directly on `ExpressionSet` objects, and can automatically determine which arrays are replicates.

Usage

```
bcomb(e, se, replicates, method=c("map","em"),
      gsnorm=FALSE, nsample=1000, eps=1.0e-6)
```

Arguments

`e`	a data frame containing the expression level for each gene on each chip.
`se`	a data frame containing the standard deviation of gene expression levels.
`replicates`	a vector indicating which chip belongs to which condition.
`method`	character specifying the method algorithm used.
`gsnorm`	logical specifying whether do global scaling normalisation or not.
`nsample`	integer. The number of sampling in parameter estimation.
`eps`	a numeric, optimisation parameter.

Details

Each element in replicate represents the condition of the chip which is in the same column order as in the expression and standard deviation matrix files.

Method "map" uses MAP of a hierarchical Bayesion model with Gamma prior on the between-replicate variance (Gelman et.al. p.285) and shares the same variance across conditions. This method is fast and suitable for the case where there are many conditions.

Method "em" uses variational inference of the same hierarchical Bayesion model as in method "map" but with conjugate prior on between-replicate variance and shares the variance across conditions.

The parameter nsample should be large enough to ensure stable parameter estimates. Should be at least 1000.

Value

The result is a data frame with components named 'M1', 'M2', and so on, which represent the mean expression values for condition 1, condition 2, and so on. It also has components named 'Std1', 'Std2', and so on, which represent the standard deviation of the gene expression values for condition 1, condtion 2, and so on.

Author(s)

Xuejun Liu, Marta Milo, Neil D. Lawrence, Magnus Rattray

References

Gelman,A., Carlin,J.B., Stern,H.S., Rubin,D.B., Bayesian data analysis. London: Chapman & Hall; 1995.

Liu,X., Milo,M., Lawrence,N.D. and Rattray,M. (2006) Probe-level variances improve accuracy in detecting differential gene expression, Bioinformatics, 22:2107-2113.

See Also

Related methods pumaComb, mmgmos and pplr

Examples

```
#  data(exampleE)
#  data(exampleStd)
#  r<-bcomb(exampleE,exampleStd,replicates=c(1,1,1,2,2,2),method="map")
```

`calcAUC` *Calculate Area Under Curve (AUC) for a standard ROC plot.*

Description

Calculates the AUC values for one or more ROC plots.

Usage

```
calcAUC(scores, truthValues, includedProbesets = 1:length(truthValues))
```

Arguments

`scores`	A vector of scores. This could be, e.g. one of the columns of the statistics of a `DEResult` object.
`truthValues`	A boolean vector indicating which scores are True Positives.
`includedProbesets`	A vector of indices indicating which scores (and truthValues) are to be used in the calculation. The default is to use all, but a subset can be used if, for example, you only want a subset of the probesets which are not True Positives to be treated as False Positives.

Value

A single number which is the AUC value.

Author(s)

Richard D. Pearson

See Also

Related methods `plotROC` and `numFP`.

Examples

```
#class1a <- rnorm(1000,0.2,0.1)
#class2a <- rnorm(1000,0.6,0.2)
#class1b <- rnorm(1000,0.3,0.1)
#class2b <- rnorm(1000,0.5,0.2)
#scores_a <- c(class1a, class2a)
#scores_b <- c(class1b, class2b)
#classElts <- c(rep(FALSE,1000), rep(TRUE,1000))
#print(calcAUC(scores_a, classElts))
#print(calcAUC(scores_b, classElts))
```

---

calculateFC *Calculate differential expression between conditions using FC*

Description

Automatically creates design and contrast matrices if not specified. This function is useful for comparing fold change results with those of other differential expression (DE) methods such as pumaDE.

Usage

```
calculateFC(
 eset
, design.matrix = createDesignMatrix(eset)
, contrast.matrix = createContrastMatrix(eset)
)
```

Arguments

eset	An object of class ExpressionSet
design.matrix	A design matrix
contrast.matrix	A contrast matrix

Details

The eset argument must be supplied, and must be a valid ExpressionSet object. Design and contrast matrices can be supplied, but if not, default matrices will be used. These should usually be sufficient for most analyses.

Value

An object of class DEResult.

Author(s)

Richard D. Pearson

See Also

Related methods pumaDE, calculateLimma, calculateTtest, createDesignMatrix and createContrastMatrix and class DEResult

Examples

```
#if (require(affydata)) {
# data(Dilution)
# eset_rma <- affy:::rma(Dilution)
# # Next line used so eset_rma only has information about the liver factor
# # The scanner factor will thus be ignored, and the two arrays of each level
# # of the liver factor will be treated as replicates
# pData(eset_rma) <- pData(eset_rma)[,1, drop=FALSE]
# FCRes <- calculateFC(eset_rma)
# topGeneIDs(FCRes,numberOfGenes=6)
# plotErrorBars(eset_rma, topGenes(FCRes))
#}
```

`calculateLimma` *Calculate differential expression between conditions using limma*

Description

Runs a default analysis using the **limma** package. Automatically creates design and contrast matrices if not specified. This function is useful for comparing **limma** results with those of other differential expression (DE) methods such as pumaDE.

Usage

```
calculateLimma(
 eset
, design.matrix = createDesignMatrix(eset)
, contrast.matrix = createContrastMatrix(eset)
)
```

Arguments

`eset`	An object of class ExpressionSet
`design.matrix`	A design matrix
`contrast.matrix`	A contrast matrix

Details

The `eset` argument must be supplied, and must be a valid ExpressionSet object. Design and contrast matrices can be supplied, but if not, default matrices will be used. These should usually be sufficient for most analyses.

Value

An object of class DEResult.

Author(s)

Richard D. Pearson

See Also

Related methods pumaDE, calculateTtest, calculateFC, createDesignMatrix and createContrastMatrix and class DEResult

Examples

```
#if (require(affydata)) {
# data(Dilution)
# eset_rma <- affy:::rma(Dilution)
# # Next line used so eset_rma only has information about the liver factor
# # The scanner factor will thus be ignored, and the two arrays of each level
# # of the liver factor will be treated as replicates
# pData(eset_rma) <- pData(eset_rma)[,1, drop=FALSE]
# limmaRes <- calculateLimma(eset_rma)
# topGeneIDs(limmaRes,numberOfGenes=6)
# plotErrorBars(eset_rma, topGenes(limmaRes))
#}
```

---

calculateTtest *Calculate differential expression between conditions using T-test*

Description

Automatically creates design and contrast matrices if not specified. This function is useful for comparing T-test results with those of other differential expression (DE) methods such as pumaDE.

Usage

```
calculateTtest(
 eset
, design.matrix = createDesignMatrix(eset)
, contrast.matrix = createContrastMatrix(eset)
)
```

Arguments

eset	An object of class ExpressionSet
design.matrix	A design matrix
contrast.matrix	A contrast matrix

Details

The eset argument must be supplied, and must be a valid ExpressionSet object. Design and contrast matrices can be supplied, but if not, default matrices will be used. These should usually be sufficient for most analyses.

Value

An object of class DEResult.

Author(s)

Richard D. Pearson

See Also

Related methods pumaDE, calculateLimma, calculateFC, createDesignMatrix and createContrastMatrix and class DEResult

Examples

```
# eset_test <- new("ExpressionSet", exprs=matrix(rnorm(400,8,2),100,4))
# pData(eset_test) <- data.frame("class"=c("A", "A", "B", "B"))
# TtestRes <- calculateTtest(eset_test)
# plotErrorBars(eset_test, topGenes(TtestRes))
```

Clust.exampleE *The example data of the mean gene expression levels*

Description

This data is an artificial example of the mean gene expression levels.

Usage

```
data(Clust.exampleE)
```

Format

A 700x20 matrix including 700 genes and 20 chips. Every 100 genes belong to one cluster from the first gene. There are 7 clusters.

Source

Liu, X. Lin, K., Andersen, B. Rattray, M. (2007) Including probe-level uncertainty in model-based gene expression clustering, BMC Bioinformatics, 8(98).

See Also

Clust.exampleStd

Clust.exampleStd *The example data of the standard deviation for gene expression levels*

Description

This data is an artificial example of the standard deviation for gene expression levels.

Usage

```
data(Clust.exampleStd)
```

Format

A 700x20 matrix including 700 genes and 20 chips. Every 100 genes belong to one cluster from the first gene. There are 7 true clusters.

Source

Liu, X. Lin, K., Andersen, B. Rattray, M. (2007) Including probe-level uncertainty in model-based gene expression clustering, BMC Bioinformatics, 8(98).

See Also

Clust.exampleE

clusterApplyLBDots *clusterApplyLB with dots to indicate progress*

Description

This is basically the clusterApplyLB function from the **snow** package, but with dots displayed to indicate progress.

Usage

```
clusterApplyLBDots(cl, x, fun, ...)
```

Arguments

cl	cluster object
x	array
fun	function or character string naming a function
...	additional arguments to pass to standard function

Author(s)

Richard D. Pearson (modified from original **snow** function)

clusterNormE *Zero-centered normalisation*

Description

This function normalise the data vector to have zero mean.

Usage

```
clusterNormE(x)
```

Arguments

x	a vector which contains gene expression level on log2 scale.

Details

Vector x is related to a gene and each element is related to a chip.

Value

The return vector is in the same format as the input x.

Author(s)

Xuejun Liu, Magnus Rattray

See Also

See Also as pumaClust and pumaClustii

Examples

```
#data(Clust.exampleE)
#Clust.exampleE.centered<-t(apply(Clust.exampleE, 1, clusterNormE))
```

clusterNormVar *Adjusting expression variance for zero-centered normalisation*

Description

This function adjusts the variance of the gene expression according to the zero-centered normalisation.

Usage

```
clusterNormVar(x)
```

Arguments

x	a vector which contains the variance of gene expression level on log2 scale.

Details

Vector x is related to a gene and each element is related to a chip.

Value

The return vector is in the same format as the input x.

Author(s)

Xuejun Liu, Magnus Rattray

See Also

See Also as pumaClust and pumaClustii

Examples

```
#data(Clust.exampleE)
#data(Clust.exampleStd)
#Clust.exampleVar<-Clust.exampleStd^2
#Clust.exampleStd.centered<-t(apply(cbind(Clust.exampleE,Clust.exampleVar), 1, clusterNormVar))
```

Clustii.exampleE *The example data of the mean gene expression levels*

Description

This data is an artificial example of the mean gene exapression levels generated by package mmgmos.

Usage

```
data(Clustii.exampleE)
```

Format

A 600x80 matrix including 600 genes and 20 conditions. Each condition has 4 replicates. Every 100 genes belong to one cluster from the first gene. There are 6 clusters.

Source

Liu,X. and Rattray,M. (2009) Including probe-level measurement error in robust mixture clustering of replicated microarray gene expression, Statistical Application in Genetics and Molecular Biology, 9(1), Article 42.

See Also

Clustii.exampleStd

Clustii.exampleStd *The example data of the standard deviation for gene expression levels*

Description

This data is an artificial example of the standard deviation for gene exapression levels generated by package mmgmos.

Usage

```
data(Clustii.exampleStd)
```

Format

A 600x80 matrix including 600 genes and 20 conditions. Each condition has 4 replicates. Every 100 genes belong to one cluster from the first gene. There are 6 clusters.

Source

Liu,X. and Rattray,M. (2009) Including probe-level measurement error in robust mixture clustering of replicated microarray gene expression, technical report available upon request.

See Also

Clustii.exampleE

compareLimmapumaDE *Compare pumaDE with a default Limma model*

Description

This function compares the identification of differentially expressed (DE) genes using the pumaDE function and the **limma** package.

Usage

```
compareLimmapumaDE(
 eset_mmgmos
, eset_comb = NULL
, eset_other = eset_mmgmos
, limmaRes = calculateLimma(eset_other)
, pumaDERes = pumaDE(eset_comb)
, contrastMatrix = createContrastMatrix(eset_mmgmos)
, numberToCompareForContrasts = 3
, numberToCompareForVenn = 100
, plotContrasts = TRUE
, contrastsFilename = NULL
, plotOther = FALSE
, otherFilename = "other"
, plotBcombContrasts = FALSE
```

```
, bcombContrastsFilename = "bcomb_contrasts"
, plotVenn = FALSE
, vennFilename = "venn.pdf"
, showTopMatches = FALSE
, returnResults = FALSE
)
```

Arguments

eset_mmgmos	An object of class ExpressionSet, that includes both expression levels as well as standard errors of the expression levels. This will often have been created using mmgmos, but might also have been created by mgmos, or any other method capable of providing standard errors.
eset_comb	An object of class ExpressionSet, includes both expression levels as well as standard errors of the expression levels for each unique condition in an experiment (i.e. created from combining the information from each replicate). This will usually have been created using pumaComb.
eset_other	An object of class ExpressionSet, that includes expression levels , and may optionally also include standard errors of the expression levels. This is used for comparison with eset_mmgmos, and might have been created by any summarisation method, e.g. rma.
limmaRes	A list with two elements, usually created using the function calculateLimma. The first element is a matrix of p-values. Each column represent one contrast. Within each column the p-values are ordered. The second element is a matrix of row numbers, which can be used to map p-values back to probe sets. If not supplied this will be automatically created from eset_other.
pumaDERes	A list with two elements, usually created using the function pumaDE. The first element is a matrix of PPLR values. Each column represent one contrast. Within each column the PPLR values are ordered. The second element is a matrix of row numbers, which can be used to map PPLR values back to probe sets. If not supplied this will be automatically created from eset_comb.
contrastMatrix	A contrast matrix. If not supplied this will be created from eset_mmgmos
numberToCompareForContrasts	An integer specifying the number of most differentially expressed probe sets (genes) that will be used in comparison charts.
numberToCompareForVenn	An integer specifying the number of most differentially expressed probe sets (genes) that will be used for comparison in the Venn diagram.
plotContrasts	A boolean specifying whether or not to plot the most differentially expressed probe sets (genes) for each contrast for the eset_mmgmos ExpressionSet.
contrastsFilename	A character string specifying a file name stem for the PDF files which will be created to hold the contrast plots for the eset_mmgmos ExpressionSet. The actually filenames will have the name of the contrast appended to this stem.
plotOther	A boolean specifying whether or not to plot the most differentially expressed probe sets (genes) for each contrast for the eset_other ExpressionSet.
otherFilename	A character string specifying a file name stem for the PDF files which will be created to hold the contrast plots for the eset_other ExpressionSet. The actually filenames will have the name of the contrast appended to this stem.

plotBcombContrasts
: A boolean specifying whether or not to plot the most differentially expressed probe sets (genes) for each contrast for the eset_comb ExpressionSet.

bcombContrastsFilename
: A character string specifying a file name stem for the PDF files which will be created to hold the contrast plots for the eset_comb ExpressionSet. The actually filenames will have the name of the contrast appended to this stem.

plotVenn
: A boolean specifying whether or not to plot a Venn diagram showing the overlap in the most differentially expressed probe sets (genes) as identified from the two different methods being compared.

vennFilename
: A character string specifying the filename for the PDF file which will hold the Venn diagram showing the overlap in the most differentially expressed probe sets (genes) as identified from the two different methods being compared.

showTopMatches
: A boolean specifying whether or not to show the probe sets which are deemed most likely to be differentially expressed.

returnResults
: A boolean specifying whether or not to return a list containing results generated.

Value

The main outputs from this function are a number of PDF files.

The function only returns results if returnResults=TRUE

Author(s)

Richard D. Pearson

See Also

Related methods pumaDE and calculateLimma

---

createContrastMatrix *Automatically create a contrast matrix from an ExpressionSet and optional design matrix*

Description

To appear

Usage

```
createContrastMatrix(eset, design=NULL)
```

Arguments

eset
: An object of class ExpressionSet.

design
: A design matrix

Details

The **puma** package has been designed to be as easy to use as possible, while not compromising on power and flexibility. One of the most difficult tasks for many users, particularly those new to microarray analysis, or statistical analysis in general, is setting up design and contrast matrices. The **puma** package will automatically create such matrices, and we believe the way this is done will suffice for most users' needs.

It is important to recognise that the automatic creation of design and contrast matrices will only happen if appropriate information about the levels of each factor is available for each array in the experimental design. This data should be held in an `AnnotatedDataFrame` class. The easiest way of doing this is to ensure that the `AnnotatedDataFrame` object holding the raw CEL file data has an appropriate `phenoData` slot. This information will then be passed through to any `ExpressionSet` object created, for example through the use of `mmgmos`. The `phenoData` slot of an `ExpressionSet` object can also be manipulated directly if necessary.

Design and contrast matrices are dependent on the experimental design. The simplest experimental designs have just one factor, and hence the `phenoData` slot will have a matrix with just one column. In this case, each unique value in that column will be treated as a distinct level of the factor, and hence `pumaComb` will group arrays according to these levels. If there are just two levels of the factor, e.g. A and B, the contrast matrix will also be very simple, with the only contrast of interest being A vs B. For factors with more than two levels, a contrast matrix will be created which reflects all possible combinations of levels. For example, if we have three levels A, B and C, the contrasts of interest will be A vs B, A vs C and B vs C. In addition, if the `others` argument is set to TRUE, the following additional contrasts will be created: A vs other (i.e. A vs B \& C), B vs other and C vs other. Note that these additional contrasts are experimental, and not currently recommended for use in calculating differential expression.

If we now consider the case of two or more factors, things become more complicated. There are now two cases to be considered: factorial experiments, and non-factorial experiments. A factorial experiment is one where all the combinations of the levels of each factor are tested by at least one array (though ideally we would have a number of biological replicates for each combination of factor levels). The `estrogen` case study from the package vignette is an example of a factorial experiment.

A non-factorial experiment is one where at least one combination of levels is not tested. If we treat the example used in the `puma-package` help page as a two-factor experiment (with factors "level" and "batch"), we can see that this is not a factorial experiment as we have no array to test the conditions "level=ten" and "batch=B". We will treat the factorial and non-factorial cases separately in the following sections.

Factorial experiments

For factorial experiments, the design matrix will use all columns from the `phenoData` slot. This will mean that `pumaComb` will group arrays according to a combination of the levels of all the factors.

Non-factorial designs

For non-factorial designed experiments, we will simply ignore columns (right to left) from the `phenoData` slot until we have a factorial design or a single factor. We can see this in the example used in the `puma-package` help page. Here we have ignored the "batch" factor, and modelled the experiment as a single-factor experiment (with that single factor being "level").

Value

The result is a matrix. See the code below for an example.

Author(s)

Richard D. Pearson

See Also

Related methods createDesignMatrix and pumaDE

Examples

```
if(FALSE){
# This is a simple example based on a real data set. Note that this is an "unbalanced" design, the "level" factor

# Next 4 lines commented out to save time in package checks, and saved version used
 if (require(affydata)) {
 data(Dilution)
 eset_mmgmos <- mmgmos(Dilution)
 }
data(eset_mmgmos)
createContrastMatrix(eset_mmgmos)

# The following shows a set of 15 synthetic data sets with increasing complexity. We first create the data sets,

# single 2-level factor
eset1 <- new("ExpressionSet", exprs=matrix(0,100,4))
pData(eset1) <- data.frame("class"=c(1,1,2,2))

# single 2-level factor - unbalanced design
eset2 <- new("ExpressionSet", exprs=matrix(0,100,4))
pData(eset2) <- data.frame("class"=c(1,2,2,2))

# single 3-level factor
eset3 <- new("ExpressionSet", exprs=matrix(0,100,6))
pData(eset3) <- data.frame("class"=c(1,1,2,2,3,3))

# single 4-level factor
eset4 <- new("ExpressionSet", exprs=matrix(0,100,8))
pData(eset4) <- data.frame("class"=c(1,1,2,2,3,3,4,4))

# 2x2 factorial
eset5 <- new("ExpressionSet", exprs=matrix(0,100,8))
pData(eset5) <- data.frame("fac1"=c("a","a","a","a","b","b","b","b"), "fac2"=c(1,1,2,2,1,1,2,2))

# 2x2 factorial - unbalanced design
eset6 <- new("ExpressionSet", exprs=matrix(0,100,10))
pData(eset6) <- data.frame("fac1"=c("a","a","a","b","b","b","b","b","b","b"), "fac2"=c(1,2,2,1,1,1,2,2,2,2)

# 3x2 factorial
eset7 <- new("ExpressionSet", exprs=matrix(0,100,12))
pData(eset7) <- data.frame("fac1"=c("a","a","a","a","b","b","b","b","c","c","c","c"), "fac2"=c(1,1,2,2,1,1,2

# 2x3 factorial
eset8 <- new("ExpressionSet", exprs=matrix(0,100,12))
pData(eset8) <- data.frame(
 "fac1"=c("a","a","a","a","a","a","b","b","b","b","b","b")
, "fac2"=c(1,1,2,2,3,3,1,1,2,2,3,3) )
```

```
# 2x2x2 factorial
eset9 <- new("ExpressionSet", exprs=matrix(0,100,8))
pData(eset9) <- data.frame(
 "fac1"=c("a","a","a","a","b","b","b","b")
, "fac2"=c(1,1,2,2,1,1,2,2)
, "fac3"=c("X","Y","X","Y","X","Y","X","Y") )

# 3x2x2 factorial
eset10 <- new("ExpressionSet", exprs=matrix(0,100,12))
pData(eset10) <- data.frame(
 "fac1"=c("a","a","a","a","b","b","b","b","c","c","c","c")
, "fac2"=c(1,1,2,2,1,1,2,2,1,1,2,2)
, "fac3"=c("X","Y","X","Y","X","Y","X","Y","X","Y","X","Y") )

# 3x2x2 factorial
eset11 <- new("ExpressionSet", exprs=matrix(0,100,12))
pData(eset11) <- data.frame(
 "fac1"=c("a","a","a","a","a","a","b","b","b","b","b","b")
, "fac2"=c(1,1,2,2,3,3,1,1,2,2,3,3)
, "fac3"=c("X","Y","X","Y","X","Y","X","Y","X","Y","X","Y") )

# 3x2x2 factorial
eset12 <- new("ExpressionSet", exprs=matrix(0,100,18))
pData(eset12) <- data.frame(
 "fac1"=c("a","a","a","a","a","a","b","b","b","b","b","b","c","c","c","c","c","c")
, "fac2"=c(1,1,2,2,3,3,1,1,2,2,3,3,1,1,2,2,3,3)
, "fac3"=c("X","Y","X","Y","X","Y","X","Y","X","Y","X","Y","X","Y","X","Y","X","Y") )

# 2x2x2x2 factorial
eset13 <- new("ExpressionSet", exprs=matrix(0,100,16))
pData(eset13) <- data.frame(
 "fac1"=c("a","a","a","a","a","a","a","a","b","b","b","b","b","b","b","b")
, "fac2"=c(0,0,0,0,1,1,1,1,0,0,0,0,1,1,1,1)
, "fac3"=c(2,2,3,3,2,2,3,3,2,2,3,3,2,2,3,3)
, "fac4"=c("X","Y","X","Y","X","Y","X","Y","X","Y","X","Y","X","Y","X","Y") )

# "Un-analysable" data set - all arrays are from the same class
eset14 <- new("ExpressionSet", exprs=matrix(0,100,4))
pData(eset14) <- data.frame("class"=c(1,1,1,1))

# "Non-factorial" data set - there are no arrays for fac1="b" and fac2=2. In this case only the first factor (fac
eset15 <- new("ExpressionSet", exprs=matrix(0,100,6))
pData(eset15) <- data.frame("fac1"=c("a","a","a","a","b","b"), "fac2"=c(1,1,2,2,1,1))

createContrastMatrix(eset1)
createContrastMatrix(eset2)
createContrastMatrix(eset3)
createContrastMatrix(eset4)
createContrastMatrix(eset5)
createContrastMatrix(eset6)
createContrastMatrix(eset7)
createContrastMatrix(eset8)
createContrastMatrix(eset9)
# For the last 4 data sets, the contrast matrices get pretty big, so we'll just show the names of each contrast
colnames(createContrastMatrix(eset10))
colnames(createContrastMatrix(eset11))
# Note that the number of contrasts can rapidly get very large for multi-factorial experiments!
```

```
colnames(createContrastMatrix(eset12))
# For this final data set, note that the puma package does not currently create interaction terms for data sets w
colnames(createContrastMatrix(eset13))

# "Un-analysable" data set - all arrays are from the same class - gives an error.  Note that we've commented this
createContrastMatrix(eset14)
# "Non-factorial" data set - there are no arrays for fac1="b" and fac2=2. In this case only the first factor (fac
createContrastMatrix(eset15)
}
```

`createDesignMatrix` *Automatically create a design matrix from an ExpressionSet*

Description

Automatically create a design matrix from an ExpressionSet.

Usage

```
createDesignMatrix(eset)
```

Arguments

`eset`	An object of class `ExpressionSet`.

Details

The **puma** package has been designed to be as easy to use as possible, while not compromising on power and flexibility. One of the most difficult tasks for many users, particularly those new to microarray analysis, or statistical analysis in general, is setting up design and contrast matrices. The **puma** package will automatically create such matrices, and we believe the way this is done will suffice for most users' needs.

It is important to recognise that the automatic creation of design and contrast matrices will only happen if appropriate information about the levels of each factor is available for each array in the experimental design. This data should be held in an `AnnotatedDataFrame` class. The easiest way of doing this is to ensure that the `AnnotatedDataFrame` object holding the raw CEL file data has an appropriate `phenoData` slot. This information will then be passed through to any `ExpressionSet` object created, for example through the use of `mmgmos`. The `phenoData` slot of an `ExpressionSet` object can also be manipulated directly if necessary.

Design and contrast matrices are dependent on the experimental design. The simplest experimental designs have just one factor, and hence the `phenoData` slot will have a matrix with just one column. In this case, each unique value in that column will be treated as a distinct level of the factor, and hence `pumaComb` will group arrays according to these levels. If there are just two levels of the factor, e.g. A and B, the contrast matrix will also be very simple, with the only contrast of interest being A vs B. For factors with more than two levels, a contrast matrix will be created which reflects all possible combinations of levels. For example, if we have three levels A, B and C, the contrasts of interest will be A vs B, A vs C and B vs C.

If we now consider the case of two or more factors, things become more complicated. There are now two cases to be considered: factorial experiments, and non-factorial experiments. A factorial experiment is one where all the combinations of the levels of each factor are tested by at least one array (though ideally we would have a number of biological replicates for each combination of

factor levels). The estrogen case study from the package vignette is an example of a factorial experiment.

A non-factorial experiment is one where at least one combination of levels is not tested. If we treat the example used in the puma-package help page as a two-factor experiment (with factors "level" and "batch"), we can see that this is not a factorial experiment as we have no array to test the conditions "level=ten" and "batch=B". We will treat the factorial and non-factorial cases separately in the following sections.

Factorial experiments

For factorial experiments, the design matrix will use all columns from the phenoData slot. This will mean that pumaComb will group arrays according to a combination of the levels of all the factors.

Non-factorial designs

For non-factorial designed experiments, we will simply ignore columns (right to left) from the phenoData slot until we have a factorial design or a single factor. We can see this in the example used in the puma-package help page. Here we have ignored the "batch" factor, and modelled the experiment as a single-factor experiment (with that single factor being "level").

Value

The result is a matrix. See the code below for an example.

Author(s)

Richard D. Pearson

See Also

Related methods createContrastMatrix, pumaComb, pumaDE and pumaCombImproved

Examples

```
if(FALSE){
# This is a simple example based on a real data set. Note that this is an "unbalanced" design, the "level" factor

# Next 4 lines commented out to save time in package checks, and saved version used
   # if (require(affydata)) {
# data(Dilution)
# eset_mmgmos <- mmgmos(Dilution)
# }
data(eset_mmgmos)
createDesignMatrix(eset_mmgmos)

# The following shows a set of 15 synthetic data sets with increasing complexity. We first create the data sets,

# single 2-level factor
eset1 <- new("ExpressionSet", exprs=matrix(0,100,4))
pData(eset1) <- data.frame("class"=c(1,1,2,2))

# single 2-level factor - unbalanced design
eset2 <- new("ExpressionSet", exprs=matrix(0,100,4))
pData(eset2) <- data.frame("class"=c(1,2,2,2))

# single 3-level factor
eset3 <- new("ExpressionSet", exprs=matrix(0,100,6))
pData(eset3) <- data.frame("class"=c(1,1,2,2,3,3))
```

```
# single 4-level factor
eset4 <- new("ExpressionSet", exprs=matrix(0,100,8))
pData(eset4) <- data.frame("class"=c(1,1,2,2,3,3,4,4))

# 2x2 factorial
eset5 <- new("ExpressionSet", exprs=matrix(0,100,8))
pData(eset5) <- data.frame("fac1"=c("a","a","a","a","b","b","b","b"), "fac2"=c(1,1,2,2,1,1,2,2))

# 2x2 factorial - unbalanced design
eset6 <- new("ExpressionSet", exprs=matrix(0,100,10))
pData(eset6) <- data.frame("fac1"=c("a","a","a","b","b","b","b","b","b","b"), "fac2"=c(1,2,2,1,1,1,2,2,2,2)

# 3x2 factorial
eset7 <- new("ExpressionSet", exprs=matrix(0,100,12))
pData(eset7) <- data.frame("fac1"=c("a","a","a","a","b","b","b","b","c","c","c","c"), "fac2"=c(1,1,2,2,1,1,

# 2x3 factorial
eset8 <- new("ExpressionSet", exprs=matrix(0,100,12))
pData(eset8) <- data.frame(
 "fac1"=c("a","a","a","a","a","a","b","b","b","b","b","b")
, "fac2"=c(1,1,2,2,3,3,1,1,2,2,3,3) )

# 2x2x2 factorial
eset9 <- new("ExpressionSet", exprs=matrix(0,100,8))
pData(eset9) <- data.frame(
 "fac1"=c("a","a","a","a","b","b","b","b")
, "fac2"=c(1,1,2,2,1,1,2,2)
, "fac3"=c("X","Y","X","Y","X","Y","X","Y") )

# 3x2x2 factorial
eset10 <- new("ExpressionSet", exprs=matrix(0,100,12))
pData(eset10) <- data.frame(
 "fac1"=c("a","a","a","a","b","b","b","b","c","c","c","c")
, "fac2"=c(1,1,2,2,1,1,2,2,1,1,2,2)
, "fac3"=c("X","Y","X","Y","X","Y","X","Y","X","Y","X","Y") )

# 3x2x2 factorial
eset11 <- new("ExpressionSet", exprs=matrix(0,100,12))
pData(eset11) <- data.frame(
 "fac1"=c("a","a","a","a","a","a","b","b","b","b","b","b")
, "fac2"=c(1,1,2,2,3,3,1,1,2,2,3,3)
, "fac3"=c("X","Y","X","Y","X","Y","X","Y","X","Y","X","Y") )

# 3x2x2 factorial
eset12 <- new("ExpressionSet", exprs=matrix(0,100,18))
pData(eset12) <- data.frame(
 "fac1"=c("a","a","a","a","a","a","b","b","b","b","b","b","c","c","c","c","c","c")
, "fac2"=c(1,1,2,2,3,3,1,1,2,2,3,3,1,1,2,2,3,3)
, "fac3"=c("X","Y","X","Y","X","Y","X","Y","X","Y","X","Y","X","Y","X","Y","X","Y") )

# 2x2x2x2 factorial
eset13 <- new("ExpressionSet", exprs=matrix(0,100,16))
pData(eset13) <- data.frame(
 "fac1"=c("a","a","a","a","a","a","a","a","b","b","b","b","b","b","b","b")
, "fac2"=c(0,0,0,0,1,1,1,1,0,0,0,0,1,1,1,1)
, "fac3"=c(2,2,3,3,2,2,3,3,2,2,3,3,2,2,3,3)
```

```
 , "fac4"=c("X","Y","X","Y","X","Y","X","Y","X","Y","X","Y","X","Y","X","Y") )

 # "Un-analysable" data set - all arrays are from the same class
 eset14 <- new("ExpressionSet", exprs=matrix(0,100,4))
 pData(eset14) <- data.frame("class"=c(1,1,1,1))

# "Non-factorial" data set - there are no arrays for fac1="b" and fac2=2. In this case only the first factor (fac
 eset15 <- new("ExpressionSet", exprs=matrix(0,100,6))
 pData(eset15) <- data.frame("fac1"=c("a","a","a","a","b","b"), "fac2"=c(1,1,2,2,1,1))

 # "pseduo 2 factor" data set - second factor is informative
 eset16 <- new("ExpressionSet", exprs=matrix(0,100,8))
 pData(eset16) <- data.frame("fac1"=c("a","a","b","b"), "fac2"=c(1,1,1,1))

 # "pseduo 2 factor" data set - first factor is informative
 eset17 <- new("ExpressionSet", exprs=matrix(0,100,8))
 pData(eset17) <- data.frame("fac1"=c("a","a","a","a"), "fac2"=c(1,1,2,2))

 # "pseudo 3 factor" data set - first factor is uninformative so actually a 2x2 factorial
 eset18 <- new("ExpressionSet", exprs=matrix(0,100,8))
 pData(eset18) <- data.frame(
  "fac1"=c("a","a","a","a","a","a","a","a")
 , "fac2"=c(1,1,2,2,1,1,2,2)
 , "fac3"=c("X","Y","X","Y","X","Y","X","Y") )

# "pseudo 3 factor" data set - first and third factors are uninformative so actually a single factor
 eset19 <- new("ExpressionSet", exprs=matrix(0,100,8))
 pData(eset19) <- data.frame(
  "fac1"=c("a","a","a","a","a","a","a","a")
 , "fac2"=c(1,1,2,2,1,1,2,2)
 , "fac3"=c("X","X","X","X","X","X","X","X") )

 createDesignMatrix(eset1)
 createDesignMatrix(eset2)
 createDesignMatrix(eset3)
 createDesignMatrix(eset4)
 createDesignMatrix(eset5)
 createDesignMatrix(eset6)
 createDesignMatrix(eset7)
 createDesignMatrix(eset8)
 createDesignMatrix(eset9)
 createDesignMatrix(eset10)
 createDesignMatrix(eset11)
 createDesignMatrix(eset12)
 createDesignMatrix(eset13)

 # "Un-analysable" data set - all arrays are from the same class - gives an error. Note that we've commented this
  # createDesignMatrix(eset14)
 # "Non-factorial" data set - there are no arrays for fac1="b" and fac2=2. In this case only the first factor (fac
  createDesignMatrix(eset15)
}
```

---

create_eset_r *Create an ExpressionSet from a PPLR matrix*

Description

This is really an internal function called from pumaComb. It is used to create an ExpressionSet object from the output of the bcomb function (which was originally part of the **pplr** package. Don't worry about it!

Usage

```
create_eset_r(
 eset
, r
, design.matrix=createDesignMatrix(eset)
)
```

Arguments

eset	An object of class ExpressionSet. The phenotype information from this is used as the phenotype information of the returned object
r	A data frame with components named 'M1', 'M2', and so on, which represent the mean expression values for condition 1, condition 2, and so on. It also has components named 'Std1', 'Std2', and so on, which represent the standard deviation of the gene expression values for condition 1, condtion 2, and so on. This type of data frame is output by function bcomb and hcomb
design.matrix	A design matrix.

Value

An object of class ExpressionSet.

Author(s)

Richard D. Pearson

See Also

Related methods bcomb, hcomb, pumaComb and pumaCombImproved

DEResult *Class DEResult*

Description

Class to contain and describe results of a differential expression (DE) analysis. The main components are statistic which hold the results of any statistic (e.g. p-values, PPLR values, etc.), and FC which hold the fold changes.

Creating Objects

DEResult objects will generally be created using one of the functions pumaDE, calculateLimma, calculateFC or calculateTtest.

Objects can also be created from scratch:

```
new("DEResult")
new("DEResult",statistic=matrix() ,FC=matrix() ,statisticDescription="unknown" ,DEMethod="unknown
)
```

Slots

statistic: Object of class "matrix" holding the statistics returned by the DE method.

FC: Object of class "matrix" holding the fold changes returned by the DE method.

statisticDescription: A text description of the contents of the statistic slot.

DEMethod: A string indicating which DE method was used to create the object.

Methods

Class-specific methods.

statistic(DEResult), statistic(DEResult,matrix)<- Access and set the statistic slot.

FC(DEResult), FC(DEResult,matrix)<- Access and set the FC slot.

statisticDescription(DEResult), statisticDescription(DEResult,character)<- Access and set the statisticDescription slot.

DEMethod(DEResult), DEMethod(DEResult,character)<- Access and set the DEMethod slot.

pLikeValues(object, contrast=1, direction="either") Access the statistics of an object of class DEResult, converted to "p-like values". If the object holds information on more than one contrast, only the values of the statistic for contrast number contrast are given. Direction can be "either" (meaning we want order genes by probability of being either up- or down-regulated), "up" (meaning we want to order genes by probability of being up-regulated), or "down" (meaning we want to order genes by probability of being down-regulated). "p-like values" are defined as values between 0 and 1, where 0 identifies the highest probability of being differentially expressed, and 1 identifies the lowest probability of being differentially expressed. We use this so that we can easily compare results from methods that provide true p-values (e.g. calculateLimma) and methods methods that do not provide p-values (e.g. pumaDE). For objects created using pumaDE, this returns 1-PPLR if the direction is "up", PPLR if direction is "down", and 1-abs(2*(PPLR-0.5)) if direction is "either". For objects created using calculateLimma or calculateTtest, this returns the p-value if direction is "either", ((p-1 * sign(FC))/2)+ 0.5, if the direction is "up", and ((1-p * sign(FC))/2)+ 0.5 if the direction is "down". For all other methods, this returns the rank of the appropriate statistic, scaled to lie between 0 and 1. contrast will be returned.

topGenes(object, numberOfGenes=1, contrast=1, direction="either") Returns the index numbers (row numbers) of the genes determined to be most likely to be differentially expressed. numberOfGenes specifies the number of genes to be returned by the function. If the object holds information on more than one contrast, only the values of the statistic for contrast number contrast are given. Direction can be "either" (meaning we want order genes by probability of being either up- or down-regulated), "up" (meaning we want to order genes by probability of being up-ragulated), or "down" (meaning we want to order genes by probability of being down-regulated). Note that genes are ordered by "p-like values" (see pLikeValues). object is an object of class DEResult.

`topGeneIDs(object, numberOfGenes=1, contrast=1, direction="either")` Returns the Affy IDs (row names) of the genes determined to be most likely to be differentially expressed. `numberOfGenes` specifies the number of genes to be returned by the function. If the object holds information on more than one contrast, only the values of the statistic for contrast number `contrast` are given. Direction can be "either" (meaning we want order genes by probability of being either up- or down-regulated), "up" (meaning we want to order genes by probability of being up-ragulated), or "down" (meaning we want to order genes by probability of being down-regulated). Note that genes are ordered by "p-like values" (see `pLikeValues`). `object` is an object of class `DEResult`.

`numberOfProbesets(object)` Returns the number of probesets (number of rows) in an object of class `DEResult`. This method is synonymous with numberOfGenes.

`numberOfGenes(object)` Returns the number of probesets (number of rows) in an object of class `DEResult`. This method is synonymous with numberOfProbesets.

`numberOfContrasts(object)` Returns the number of contrasts (number of columns) in an object of class `DEResult`.

`write.reslts(object) signature(x = "DEResult")`: writes the statistics and related fold changes (FCs) to files. It takes the same arguments as `write.table`. The argument "file" does not need to set any extension. The different file marks and extension "csv" will be added automatically. The default file name is "tmp". In the final results, statistics are in the file "tmp_statistics.csv", and FCs are in "tmp_FCs.csv" respectively.

Standard generic methods:

`show(object)` Informatively display object contents.

Author(s)

Richard D. Pearson

See Also

Related methods `pumaDE`, `calculateLimma`, `calculateFC` or `calculateTtest`.

Examples

```
if(FALSE){
## Create an example DEResult object
# Next 4 lines commented out to save time in package checks, and saved version used
# if (require(affydata)) {
# data(Dilution)
# eset_mmgmos <- mmgmos(Dilution)
# }
data(eset_mmgmos)

# Next line used so eset_mmgmos only has information about the liver factor
# The scanner factor will thus be ignored, and the two arrays of each level
# of the liver factor will be treated as replicates
pData(eset_mmgmos) <- pData(eset_mmgmos)[,1,drop=FALSE]

# To save time we'll just use 100 probe sets for the example
eset_mmgmos_100 <- eset_mmgmos[1:100,]
eset_comb <- pumaComb(eset_mmgmos_100)

esetDE <- pumaDE(eset_comb)
```

```
## Use some of the methods
statisticDescription(esetDE)
DEMethod(esetDE)
numberOfProbesets(esetDE)
numberOfContrasts(esetDE)
topGenes(esetDE)
topGenes(esetDE, 3)
pLikeValues(esetDE)[topGenes(esetDE,3)]
topGeneIDs(esetDE, 3)
topGeneIDs(esetDE, 3, direction="down")

## save the expression results into files
write.reslts(esetDE, file="example")
}
```

`erfc` *The complementary error function*

Description

This function calculates the complementary error function of an input x.

Usage

```
erfc(x)
```

Arguments

x	a numeric, the input.

Details

erfc is implemented using the function qnorm.

Value

The return is a numeric.

Author(s)

Xuejun Liu

See Also

qnorm

Examples

```
erfc(0.5)
```

eset_mmgmos — *An example ExpressionSet created from the Dilution data with mmgmos*

Description

This data is created by applying mmgmos to the Dilution AffyBatch object from the affydata package.

Usage

```
data(eset_mmgmos)
```

Format

An object of class ExpressionSet.

Source

see Dilution

exampleE — *The example data of the mean gene expression levels*

Description

This data is an artificial example of the mean gene expression levels from golden spike-in data set in Choe et al. (2005).

Usage

```
data(exampleE)
```

Format

A 200x6 matrix including 200 genes and 6 chips. The first 3 chips are replicates for C condition and the last 3 chips are replicates for S conditon.

Source

Choe,S.E., Boutros,M., Michelson,A.M., Church,G.M., Halfon,M.S.: Preferred analysis methods for Affymetrix GeneChips revealed by a wholly defined control dataset. Genome Biology, 6 (2005) R16.

See Also

exampleStd

exampleStd *The example data of the standard deviation for gene expression levels*

Description

This data is an artificial example of the standard deviation for gene exapression levels from golden spike-in data set in Choe et al. (2005).

Usage

```
data(exampleStd)
```

Format

A 200x6 matrix including 200 genes and 6 chips. The first 3 chips are replicates for C condition and the last 3 chips are replicates for S conditon.

Source

Choe,S.E., Boutros,M., Michelson,A.M., Church,G.M., Halfon,M.S.: Preferred analysis methods for Affymetrix GeneChips revealed by a wholly defined control dataset. Genome Biology, 6 (2005) R16.

See Also

exampleE

exprReslt-class *Class exprReslt*

Description

This is a class representation for Affymetrix GeneChip probe level data. The main component are the intensities, estimated expression levels and the confidence of expression levels from multiple arrays of the same CDF type. In extends ExpressionSet.

Objects from the Class

Objects can be created by calls of the form new("exprReslt", ...).

Fields

prcfive: Object of class "matrix" representing the 5 percentile of the observed expression levels. This is a matrix with columns representing patients or cases and rows representing genes.

prctwfive: Object of class "matrix" representing the 25 percentile of the observed expression levels. This is a matrix with columns representing patients or cases and rows representing genes.

prcfifty: Object of class "matrix" representing the 50 percentile of the observed expression levels. This is a matrix with columns representing patients or cases and rows representing genes.

`prcsevfive`: Object of class "matrix" representing the 75 percentile of the observed expression levels. This is a matrix with columns representing patients or cases and rows representing genes.

`prcninfive`: Object of class "matrix" representing the 95 percentile of the observed expression levels. This is a matrix with columns representing patients or cases and rows representing genes.

`phenoData`: Object of class "phenoData" inherited from `ExpressionSet`.

`annotation`: A character string identifying the annotation that may be used for the `ExpressionSet` instance.

Extends

Class `"ExpressionSet"`, directly.

Methods

se.exprs `signature(object = "exprReslt")`: obtains the standard error of the estimated expression levels.

se.exprs<- `signature(object = "exprReslt")`: replaces the standard error of the estimated expression levels.

prcfifty `signature(object = "exprReslt")`: obtains the 50 percentile of the estimated expression levels.

prcfifty<- `signature(object = "exprReslt")`: replaces the 50 percentile of the estimated expression levels.

prcfive `signature(object = "exprReslt")`: obtains the 5 percentile of the estimated expression levels.

prcfive<- `signature(object = "exprReslt")`: replaces the 5 percentile of the estimated expression levels.

prcninfive `signature(object = "exprReslt")`: obtains the 95 percentile of the estimated expression levels.

prcninfive<- `signature(object = "exprReslt")`: replaces the 95 percentile of the estimated expression levels.

prcsevfive `signature(object = "exprReslt")`: obtains the 75 percentile of the estimated expression levels.

prcsevfive<- `signature(object = "exprReslt")`: replaces the 75 percentile of the estimated expression levels.

prctwfive `signature(object = "exprReslt")`: obtains the 25 percentile of the estimated expression levels.

prctwfive<- `signature(object = "exprReslt")`: replaces the 25 percentile of the estimated expression levels.

show `signature(object = "exprReslt")`: renders information about the exprReslt in a concise way on stdout.

write.reslts `signature(x = "exprReslt")`: writes the expression levels and related confidences to files. It takes the same arguments as `write.table`. The argument "file" does not need to set any extension. The different file marks and extension "csv" will be added automatically. The default file name is "tmp". In the final results, expression levels are in the file "tmp_exprs.csv", standard deviations in "tmp_se.csv", 5 percentiles in "tmp_prctile5.csv", likewise, 25, 50, 75 and 95 percentiles in "tmp_prctile25.csv", "tmp_prctile50.csv", "tmp_prctile75.csv" and "tmp_prctile95.csv" respectively.

Author(s)

Xuejun Liu, Magnus Rattray, Marta Milo, Neil D. Lawrence, Richard D. Pearson

See Also

Related method mmgmos and related class ExpressionSet.

Examples

```
if(FALSE){
## load example data from package affydata
# Next 4 lines commented out to save time in package checks, and saved version used
# if (require(affydata)) {
# data(Dilution)
# eset_mmgmos <- mmgmos(Dilution)
# }
data(eset_mmgmos)

## save the expression results into files
write.reslts(eset_mmgmos, file="example")
}
```

---

gmhta *Compute gene and transcript expression values and standard deviatons from hta2.0 CEL Files*

Description

This function converts an object of FeatureSet into an object of class exprReslt using the gamma model for hta2.0 chips. This function obtains confidence of measures, standard deviation and 5, 25, 50, 75 and 95 percentiles, as well as the estimated expression levels.

Usage

```
gmhta(
      object
     ,background=FALSE
     ,gsnorm=c("median", "none", "mean", "meanlog")
     ,savepar=FALSE
     ,eps=1.0e-6
     ,addConstant = 0
     ,cl=NULL
     ,BatchFold=10
)
```

Arguments

object	an object of FeatureSet
background	Logical value. If TRUE, perform background correction before applying gmhta.
gsnorm	character. specifying the algorithm of global scaling normalisation.

savepar	Logical value. If TRUE the estimated parameters of the model are saved in file par_gmhta.txt
eps	Optimisation termination criteria.
addConstant	numeric. This is an experimental feature and should not generally be changed from the default value.
cl	This function can be parallelised by setting parameter cl. For more details, please refer to the vignette.
BatchFold	we divide tasks into BatchFold*n jobs where n is the number of cluster nodes. The first n jobs are placed on the n nodes. When the first job is completed,the next job is placed on the available node. This continues until all jobs are completed. The default value is ten. The user also can change the value according to the number of cluster nodes n. We suggest that for bigger n BatchFold should be smaller.

Details

The obtained expression measures are in log base 2 scale. Using the known relationships between genes, transcripts and probes, we propose a gamma model for hta2.0 data to calculate transcript and gene expression levels. The algorithms of global scaling normalisation can be one of "median", "none", "mean", "meanlog". "mean" and "meanlog" are mean-centered normalisation on raw scale and log scale respectively, and "median" is median-centered normalisation. "none" will result in no global scaling normalisation being applied. This function can be parallelised by setting parameter cl. For more details, please refer to the vignette.

Value

A list of two object of class exprReslt.

Author(s)

Xuejun Liu,WuJun Zhang,Zhenzhu gao, Magnus Rattray

References

XueJun Liu. (2013) puma 3.0: improved uncertainty propagation methods for gene and transcript expression analysis, BMC Bioinformatics, 14:39.

Manhong Dai, Pinglang Wang,Andrew D. Boyd. (2005) Evolving gene/transcript definitions significantly alter the interpretation of GeneChip data,Nucleic Acid Research 33(20):e175.

See Also

Related class exprReslt-class

Examples

```
if(FALSE){
## The following scripts show the use of the method.
#library(puma)
## load CEL files
# object<-read.celfiles("celnames")
#eset<-gmhta(object,gsnorm="none",cl=cl)

}
```

gmoExon *Compute gene and transcript expression values and standard deviatons from exon CEL Files*

Description

This function converts an object of FeatureSet into an object of class exprReslt using the gamma model for exon chips. This function obtains confidence of measures, standard deviation and 5, 25, 50, 75 and 95 percentiles, as well as the estimated expression levels.

Usage

```
gmoExon(
      object
     ,exontype = c("Human", "Mouse", "Rat")
     ,background=FALSE
     ,gsnorm=c("median", "none", "mean", "meanlog")
     ,savepar=FALSE
     ,eps=1.0e-6
     ,addConstant = 0
     ,cl=NULL
     ,BatchFold=10
)
```

Arguments

object	an object of FeatureSet
exontype	character. specifying the type of exon chip.
background	Logical value. If TRUE, perform background correction before applying gmoExon.
gsnorm	character. specifying the algorithm of global scaling normalisation.
savepar	Logical value. If TRUE the estimated parameters of the model are saved in file par_gmoExon.txt
eps	Optimisation termination criteria.
addConstant	numeric. This is an experimental feature and should not generally be changed from the default value.
cl	This function can be parallelised by setting parameter cl. For more details, please refer to the vignette.
BatchFold	we divide tasks into BatchFold*n jobs where n is the number of cluster nodes. The first n jobs are placed on the n nodes. When the first job is completed,the next job is placed on the available node. This continues until all jobs are completed. The default value is ten. The user also can change the value according to the number of cluster nodes n. We suggest that for bigger n BatchFold should be smaller.

Details

The obtained expression measures are in log base 2 scale. Using the known relationships between genes, transcripts and probes, we propose a gamma model for exon array data to calculate transcript and gene expression levels. The algorithms of global scaling normalisation can be one of "median",

"none", "mean", "meanlog". "mean" and "meanlog" are mean-centered normalisation on raw scale and log scale respectively, and "median" is median-centered normalisation. "none" will result in no global scaling normalisation being applied. This function can be parallelised by setting parameter cl. For more details, please refer to the vignette.

Value

A list of two object of class `exprReslt`.

Author(s)

Xuejun Liu, Zhenzhu gao, Magnus Rattray, Marta Milo, Neil D. Lawrence

References

Liu,X., Milo,M., Lawrence,N.D. and Rattray,M. (2005) A tractable probabilistic model for Affymetrix probe-level analysis across multiple chips, Bioinformatics, 21:3637-3644.

Milo,M., Niranjan,M., Holley,M.C., Rattray,M. and Lawrence,N.D. (2004) A probabilistic approach for summarising oligonucleotide gene expression data, technical report available upon request.

Milo,M., Fazeli,A., Niranjan,M. and Lawrence,N.D. (2003) A probabilistic model for the extractioin of expression levels from oligonucleotide arrays, Biochemical Society Transactions, 31: 1510-1512.

Peter Spellucci. DONLP2 code and accompanying documentation. Electronically available via http://plato.la.asu.edu/donlp2.html

Risueno A, Fontanillo C, Dinger ME, De Las Rivas J. GATExplorer: genomic and transcriptomic explorer; mapping expression probes to gene loci, transcripts, exons and ncRNAs. BMC Bioinformatics.2010.

See Also

Related class `exprReslt-class`

Examples

```
if(FALSE){
## The following scripts show the use of the method.
## load CEL files
# celFiles<-c("SR20070419HEX01.CEL", "SR20070419HEX02.CEL","SR20070419HEX06.CEL","SR20070419HEX07.CEL)
#oligo_object.exon<-read.celfiles(celFiles);

## use method gmoExon to calculate the expression levels and related confidence
## of the measures for the example data
#eset_gmoExon<-gmoExon(oligo_object.exon,exontype="Human",gsnorm="none",cl=cl)
}
```

hcomb *Combining replicates for each condition with the true gene expression*

Description

This function calculates the combined (from replicates) signal for each condition using Bayesian models, which are added a hidden variable to represent the true expression for each gene on each chips. The inputs are gene expression levels and the probe-level standard deviations associated with expression measurements for each gene on each chip. The outputs include gene expression levels and standard deviation for each condition.

Usage

```
hcomb(e, se, replicates, max_num=c(200,500,1000),gsnorm=FALSE, eps=1.0e-6)
```

Arguments

e	a data frame containing the expression level for each gene on each chip.
se	a data frame containing the standard deviation of gene expression levels.
replicates	a vector indicating which chip belongs to which condition.
max_num	integer. The maximum number of iterations controls the convergence.
gsnorm	logical specifying whether do global scaling normalisation or not.
eps	a numeric, optimisation parameter.

Details

Each element in replicate represents the condition of the chip which is in the same column order as in the expression and standard deviation matrix files.

The max_num is used to control the maximum number of the iterations in the EM algorithm. The best value of the max_num is from 200 to 1000, and should be set 200 at least. The default value is 200.

Value

The result is a data frame with components named 'M1', 'M2', and so on, which represent the mean expression values for condition 1, condition 2, and so on. It also has components named 'Std1', 'Std2', and so on, which represent the standard deviation of the gene expression values for condition 1, condtion 2, and so on.

Author(s)

Li Zhang, Xuejun Liu

References

Gelman,A., Carlin,J.B., Stern,H.S., Rubin,D.B., Bayesian data analysis. London: Chapman & Hall; 1995.

Zhang,L. and Liu,X. (2009) An improved probabilistic model for finding differential gene expression, technical report available request.

Liu,X., Milo,M., Lawrence,N.D. and Rattray,M. (2006) Probe-level variances improve accuracy in detecting differential gene expression, Bioinformatics, 22(17):2107-13.

See Also

Related method pumaCombImproved, mmgmos and pplr

Examples

```
if(FALSE){
  data(exampleE)
  data(exampleStd)
  r<-hcomb(exampleE,exampleStd,replicates=c(1,1,1,2,2,2))
}
```

---

hgu95aphis *Estimated parameters of the distribution of phi*

Description

The pre-estimated parameters of log normal distribution of ϕ, which is the fraction of specific signal binding to mismatch probe.

Usage

```
data(hgu95aphis)
```

Format

The format is: num [1:3] 0.171 -1.341 0.653

Details

The current values of hgu95aphis are estimated from Affymetrix spike-in data sets. It was loaded in the method "mmgmos".

hgu95aphis[1:3] is respectively the mode, mean and variance of the log normal distribution of ϕ, and hgu95aphis[1] is also the intial value of ϕ in the model optimisation.

---

igmoExon *Separately Compute gene and transcript expression values and standard deviatons from exon CEL Files by the conditions.*

Description

The principle of this function is as same as the function gmoExon.This function separately calculates gene expression values by the conditions and then combined every condition's results, and normalises them finally.

Usage

```
igmoExon(
      cel.path
     ,SampleNameTable
     ,exontype = c("Human", "Mouse", "Rat")
     ,background=FALSE
     ,gsnorm=c("median", "none", "mean", "meanlog")
     ,savepar=FALSE
     ,eps=1.0e-6
     ,addConstant = 0
     ,condition=c("Yes","No")
     ,cl=NULL
     ,BatchFold=10
)
```

Arguments

cel.path	The directory where you put the CEL files.
SampleNameTable	It is a tab-separated table with two columns,ordered by "Celnames","Condition"
exontype	character. specifying the type of exon chip.
background	Logical value. If TRUE, perform background correction before applying gmoExon.
savepar	Logical value. If TRUE the estimated parameters of the model are saved in file par_gmoExon.txt
eps	Optimisation termination criteria.
addConstant	numeric. This is an experimental feature and should not generally be changed from the default value.
gsnorm	character. specifying the algorithm of global scaling normalisation.
condition	Yes or No. "Yes" means the igmoExon function separately calculates gene expression values by the conditions and then combined every condition's results, and normalises them finally. "No" means the igmoExon calulates the gene expression values as same as the gmoExon function.
cl	This function can be parallelised by setting parameter cl. For more details, please refer to the vignette.
BatchFold	we divide tasks into BatchFold*n jobs where n is the number of cluster nodes. The first n jobs are placed on the n nodes. When the first job is completed,the next job is placed on the available node. This continues until all jobs are completed. The default value is ten. The user also can change the value according to the number of cluster nodes n. We suggest that for bigger n BatchFold should be smaller.

Details

The obtained expression measures are in log base 2 scale. Using the known relationships between genes, transcripts and probes, we propose a gamma model for exon array data to calculate transcript and gene expression levels. The algorithms of global scaling normalisation can be one of "median", "none", "mean", "meanlog". "mean" and "meanlog" are mean-centered normalisation on raw scale and log scale respectively, and "median" is median-centered normalisation. "none" will result in no global scaling normalisation being applied.

Value

A list of two object of class `exprReslt`.

Author(s)

Xuejun Liu, Zhenzhu gao, Magnus Rattray, Marta Milo, Neil D. Lawrence

References

Liu,X., Milo,M., Lawrence,N.D. and Rattray,M. (2005) A tractable probabilistic model for Affymetrix probe-level analysis across multiple chips, Bioinformatics, 21:3637-3644.

Milo,M., Niranjan,M., Holley,M.C., Rattray,M. and Lawrence,N.D. (2004) A probabilistic approach for summarising oligonucleotide gene expression data, technical report available upon request.

Milo,M., Fazeli,A., Niranjan,M. and Lawrence,N.D. (2003) A probabilistic model for the extractioin of expression levels from oligonucleotide arrays, Biochemical Society Transactions, 31: 1510-1512.

Peter Spellucci. DONLP2 code and accompanying documentation. Electronically available via http://plato.la.asu.edu/donlp2.html

Risueno A, Fontanillo C, Dinger ME, De Las Rivas J. GATExplorer: genomic and transcriptomic explorer; mapping expression probes to gene loci, transcripts, exons and ncRNAs. BMC Bioinformatics.2010.

See Also

Related class `exprReslt-class`

Examples

```
if(FALSE){
## The following scripts show the use of the method.
## load CEL files
# cel.path<-cel.path;
# SampleNameTable<-"SampleNameTable"
#eset_igmoExon<-igmoExon(cel.path="cel.path"
                   # , SampleNameTable="SampleNameTable"
                   # , exontype="Human"
                   # , gsnorm="none", condition="Yes",cl=cl)
}
```

`justmgMOS` *Compute mgmos Directly from CEL Files*

Description

This function converts CEL files into an `exprReslt` using mgmos.

Usage

```
justmgMOS(..., filenames=character(0),
          widget=getOption("BioC")$affy$use.widgets,
          compress=getOption("BioC")$affy$compress.cel,
          celfile.path=getwd(),
          sampleNames=NULL,
          phenoData=NULL,
          description=NULL,
          notes="",
       background=TRUE, gsnorm=c("median", "none", "mean", "meanlog"), savepar=FALSE, eps=1.0e-6)

just.mgmos(..., filenames=character(0),
           phenoData=new("AnnotatedDataFrame"),
           description=NULL,
           notes="",
           compress=getOption("BioC")$affy$compress.cel,
        background=TRUE, gsnorm=c("median", "none", "mean", "meanlog"), savepar=FALSE, eps=1.0e-6)
```

Arguments

...	file names separated by comma.
filenames	file names in a character vector.
widget	a logical specifying if widgets should be used.
compress	are the CEL files compressed?
celfile.path	a character denoting the path where cel files locate.
sampleNames	a character vector of sample names to be used in the FeatureSet.
phenoData	an AnnotatedDataFrame object.
description	a MIAME object.
notes	notes.
background	Logical value. If TRUE, then perform background correction before applying mgmos.
gsnorm	character. specifying the algorithm of global scaling normalisation.
savepar	Logical value. If TRUE, then the estimated parameters of the model are saved in file par_mgmos.txt and phi_mgmos.txt.
eps	Optimisation termination criteria.

Details

This method should require much less RAM than the conventional method of first creating an FeatureSet and then running mgmos.

Note that this expression measure is given to you in log base 2 scale. This differs from most of the other expression measure methods.

The algorithms of global scaling normalisation can be one of "median", "none", "mean", "meanlog". "mean" and "meanlog" are mean-centered normalisation on raw scale and log scale respectively, and "median" is median-centered normalisation. "none" will result in no global scaling normalisation being applied.

Value

An exprReslt.

See Also

Related class exprReslt-class and related method mgmos

---

justmmgMOS *Compute mmgmos Directly from CEL Files*

Description

This function converts CEL files into an exprReslt using mmgmos.

Usage

```
justmmgMOS(..., filenames=character(0),
          widget=getOption("BioC")$affy$use.widgets,
          compress=getOption("BioC")$affy$compress.cel,
          celfile.path=getwd(),
          sampleNames=NULL,
          phenoData=NULL,
          description=NULL,
          notes="",
       background=TRUE, gsnorm=c("median", "none", "mean", "meanlog"), savepar=FALSE, eps=1.0e-6)

just.mmgmos(..., filenames=character(0),
           phenoData=new("AnnotatedDataFrame"),
           description=NULL,
           notes="",
           compress=getOption("BioC")$affy$compress.cel,
        background=TRUE, gsnorm=c("median", "none", "mean", "meanlog"), savepar=FALSE, eps=1.0e-6)
```

Arguments

...	file names separated by comma.
filenames	file names in a character vector.
widget	a logical specifying if widgets should be used.
compress	are the CEL files compressed?
celfile.path	a character denoting the path where cel files locate.
sampleNames	a character vector of sample names to be used in the FeatureSet.
phenoData	an AnnotatedDataFrame object.
description	a MIAME object
notes	notes
background	Logical value. If TRUE, then perform background correction before applying mmgmos.
gsnorm	character. specifying the algorithm of global scaling normalisation.
savepar	Logical value. If TRUE, the the estimated parameters of the model are saved in file par_mmgmos.txt and phi_mmgmos.txt.
eps	Optimisation termination criteria.

Details

This method should require much less RAM than the conventional method of first creating an FeatureSet and then running mmgmos.

Note that this expression measure is given to you in log base 2 scale. This differs from most of the other expression measure methods.

The algorithms of global scaling normalisation can be one of "median", "none", "mean", "meanlog". "mean" and "meanlog" are mean-centered normalisation on raw scale and log scale respectively, and "median" is median-centered normalisation. "none" will result in no global scaling normalisation being applied.

Value

An exprReslt.

See Also

Related class exprReslt-class and related method mmgmos

---

legend2 *A legend which allows longer lines*

Description

This function can be used to add legends to plots. This is almost identical to the legend function, accept it has an extra parameter, seg.len which allows the user to change the lengths of lines shown in legends.

Usage

```
legend2(x, y = NULL, legend, fill = NULL, col = par("col"),
    lty, lwd, pch, angle = 45, density = NULL, bty = "o", bg = par("bg"),
    box.lwd = par("lwd"), box.lty = par("lty"), pt.bg = NA, cex = 1,
    pt.cex = cex, pt.lwd = lwd, xjust = 0, yjust = 1, x.intersp = 1,
    y.intersp = 1, adj = c(0, 0.5), text.width = NULL, text.col = par("col"),
    merge = do.lines && has.pch, trace = FALSE, plot = TRUE,
    ncol = 1, horiz = FALSE, title = NULL, inset = 0, seg.len = 2)
```

Arguments

x, y	the x and y co-ordinates to be used to position the legend. They can be specified by keyword or in any way which is accepted by xy.coords: See Details.
legend	a character or expression vector. of length ≥ 1 to appear in the legend. Other objects will be coerced by as.graphicsAnnot.
fill	if specified, this argument will cause boxes filled with the specified colors (or shaded in the specified colors) to appear beside the legend text.
col	the color of points or lines appearing in the legend.
lty, lwd	the line types and widths for lines appearing in the legend. One of these two *must* be specified for line drawing.

pch	the plotting symbols appearing in the legend, either as vector of 1-character strings, or one (multi character) string. *Must* be specified for symbol drawing.
angle	angle of shading lines.
density	the density of shading lines, if numeric and positive. If NULL or negative or NA color filling is assumed.
bty	the type of box to be drawn around the legend. The allowed values are "o" (the default) and "n".
bg	the background color for the legend box. (Note that this is only used if bty != "n".)
box.lty, box.lwd	the line type and width for the legend box.
pt.bg	the background color for the points, corresponding to its argument bg.
cex	character expansion factor **relative** to current par("cex").
pt.cex	expansion factor(s) for the points.
pt.lwd	line width for the points, defaults to the one for lines, or if that is not set, to par("lwd").
xjust	how the legend is to be justified relative to the legend x location. A value of 0 means left justified, 0.5 means centered and 1 means right justified.
yjust	the same as xjust for the legend y location.
x.intersp	character interspacing factor for horizontal (x) spacing.
y.intersp	the same for vertical (y) line distances.
adj	numeric of length 1 or 2; the string adjustment for legend text. Useful for y-adjustment when labels are plotmath expressions.
text.width	the width of the legend text in x ("user") coordinates. (Should be positive even for a reversed x axis.) Defaults to the proper value computed by strwidth(legend).
text.col	the color used for the legend text.
merge	logical; if TRUE, "merge" points and lines but not filled boxes. Defaults to TRUE if there are points and lines.
trace	logical; if TRUE, shows how legend does all its magical computations.
plot	logical. If FALSE, nothing is plotted but the sizes are returned.
ncol	the number of columns in which to set the legend items (default is 1, a vertical legend).
horiz	logical; if TRUE, set the legend horizontally rather than vertically (specifying horiz overrides the ncol specification).
title	a character string or length-one expression giving a title to be placed at the top of the legend. Other objects will be coerced by as.graphicsAnnot.
inset	inset distance(s) from the margins as a fraction of the plot region when legend is placed by keyword.
seg.len	numeric specifying length of lines in legend.

Details

Arguments x, y, legend are interpreted in a non-standard way to allow the coordinates to be specified *via* one or two arguments. If legend is missing and y is not numeric, it is assumed that the second argument is intended to be legend and that the first argument specifies the coordinates.

The coordinates can be specified in any way which is accepted by `xy.coords`. If this gives the coordinates of one point, it is used as the top-left coordinate of the rectangle containing the legend. If it gives the coordinates of two points, these specify opposite corners of the rectangle (either pair of corners, in any order).

The location may also be specified by setting `x` to a single keyword from the list `"bottomright"`, `"bottom"`, `"bottomleft"`, `"left"`, `"topleft"`, `"top"`, `"topright"`, `"right"` and `"center"`. This places the legend on the inside of the plot frame at the given location. Partial argument matching is used. The optional `inset` argument specifies how far the legend is inset from the plot margins. If a single value is given, it is used for both margins; if two values are given, the first is used for `x`-distance, the second for `y`-distance.

"Attribute" arguments such as `col`, `pch`, `lty`, etc, are recycled if necessary. `merge` is not.

Points are drawn *after* lines in order that they can cover the line with their background color `pt.bg`, if applicable.

See the examples for how to right-justify labels.

Value

A list with list components

`rect`	a list with components
	`w, h` positive numbers giving **w**idth and **h**eight of the legend's box.
	`left, top` x and y coordinates of upper left corner of the box.
`text`	a list with components
	`x, y` numeric vectors of length `length(legend)`, giving the x and y coordinates of the legend's text(s).

returned invisibly.

Author(s)

Richard Pearson (modified from original **graphics** package function.)

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth \& Brooks/Cole.

Murrell, P. (2005) *R Graphics*. Chapman & Hall/CRC Press.

See Also

`legend`

Examples

```
if(FALSE){
x <- seq(-pi, pi, len = 65)
plot(x, sin(x), type = "l", ylim = c(-1.2, 1.8), col = 3, lty = 2)
points(x, cos(x), pch = 3, col = 4)
lines(x, tan(x), type = "b", lty = 1, pch = 4, col = 6)
title("legend(..., lty = c(2, -1, 1), pch = c(-1,3,4), merge = TRUE)",
      cex.main = 1.1)
legend2(-1, 1.9, c("sin", "cos", "tan"), col = c(3,4,6),
```

```
        text.col = "green4", lty = c(2, -1, 1), pch = c(-1, 3, 4),
        merge = TRUE, bg = 'gray90', seg.len=6)
}
```

`license.puma` *Print puma license*

Description

This function prints the license under which puma is made available.

Usage

```
license.puma()
```

Value

Null.

Author(s)

Richard Pearson (based on the license.cosmo function from the cosmo package)

Examples

```
license.puma()
```

`matrixDistance` *Calculate distance between two matrices*

Description

This calculates the mean Euclidean distance between the rows of two matrices. It is used in the function pumaPCA

Usage

```
matrixDistance(
    matrixA
,   matrixB
)
```

Arguments

`matrixA`	the first matrix
`matrixB`	the second matrix

Value

A numeric giving the mean distance

Author(s)

Richard D. Pearson

See Also

Related class pumaPCA

Examples

```
if(FALSE){
 show(matrixDistance(matrix(1,2,2),matrix(2,2,2)))
}
```

---

mgmos *modified gamma Model for Oligonucleotide Signal*

Description

This function converts an object of class FeatureSet into an object of class exprReslt using the modified gamma Model for Oligonucleotide Signal (multi-mgMOS). This function obtains confidence of measures, standard deviation and 5, 25, 50, 75 and 95 percentiles, as well as the estimated expression levels.

Usage

```
mgmos(
 object
, background=FALSE
, replaceZeroIntensities=TRUE
, gsnorm=c("median", "none", "mean", "meanlog")
, savepar=FALSE
, eps=1.0e-6
)
```

Arguments

object	an object of FeatureSet
background	Logical value. If TRUE, perform background correction before applying mmgmos.
replaceZeroIntensities	Logical value. If TRUE, replace 0 intensities with 1 before applying mmgmos.
gsnorm	character. specifying the algorithm of global scaling normalisation.
savepar	Logical value. If TRUE the estimated parameters of the model are saved in file par_mmgmos.txt and phi_mmgmos.txt.
eps	Optimisation termination criteria.

Details

The obtained expression measures are in log base 2 scale.

The algorithms of global scaling normalisation can be one of "median", "none", "mean", "meanlog". "mean" and "meanlog" are mean-centered normalisation on raw scale and log scale respectively, and "median" is median-centered normalisation. "none" will result in no global scaling normalisation being applied.

There are 4*n columns in file par_mgmos.txt, n is the number of chips. Every 4 columns are parameters for a chip. Among every 4 columns, the first one is for 'alpha' values, the 2nd one is for 'a' values, The 3rd column is for 'c' and the final column is values for 'd'.

Value

An object of class `exprReslt`.

Author(s)

Xuejun Liu, Magnus Rattray, Marta Milo, Neil D. Lawrence

References

Liu,X., Milo,M., Lawrence,N.D. and Rattray,M. (2005) A tractable probabilistic model for Affymetrix probe-level analysis across multiple chips, Bioinformatics, 21:3637-3644.

Milo,M., Niranjan,M., Holley,M.C., Rattray,M. and Lawrence,N.D. (2004) A probabilistic approach for summarising oligonucleotide gene expression data, technical report available upon request.

Milo,M., Fazeli,A., Niranjan,M. and Lawrence,N.D. (2003) A probabilistic model for the extractioin of expression levels from oligonucleotide arrays, Biochemical Society Transactions, 31: 1510-1512.

Peter Spellucci. DONLP2 code and accompanying documentation. Electronically available via http://plato.la.asu.edu/donlp2.html

See Also

Related class `exprReslt-class` and related method `mmgmos`

Examples

```
if(FALSE){
## Code commented out to speed up checks
## load example data from package affydata
# if (require(pumadata)&&require(puma)){
  #  data(oligo.estrogen)
# use method mgMOS to calculate the expression levels and related confidence
# of the measures for the example data
  #  eset<-mgmos(oligo.estrogen,gsnorm="none")
#}
}
```

mmgmos *Multi-chip modified gamma Model for Oligonucleotide Signal*

Description

This function converts an object of class FeatureSet into an object of class exprReslt using the Multi-chip modified gamma Model for Oligonucleotide Signal (multi-mgMOS). This function obtains confidence of measures, standard deviation and 5, 25, 50, 75 and 95 percentiles, as well as the estimated expression levels.

Usage

```
mmgmos(
 object
, background=FALSE
, replaceZeroIntensities=TRUE
, gsnorm=c("median", "none", "mean", "meanlog")
, savepar=FALSE
, eps=1.0e-6
, addConstant = 0
)
```

Arguments

object	an object of FeatureSet
background	Logical value. If TRUE, perform background correction before applying mmgmos.
replaceZeroIntensities	Logical value. If TRUE, replace 0 intensities with 1 before applying mmgmos.
gsnorm	character. specifying the algorithm of global scaling normalisation.
savepar	Logical value. If TRUE the estimated parameters of the model are saved in file par_mmgmos.txt and phi_mmgmos.txt.
eps	Optimisation termination criteria.
addConstant	numeric. This is an experimental feature and should not generally be changed from the default value.

Details

The obtained expression measures are in log base 2 scale.

The algorithms of global scaling normalisation can be one of "median", "none", "mean", "meanlog". "mean" and "meanlog" are mean-centered normalisation on raw scale and log scale respectively, and "median" is median-centered normalisation. "none" will result in no global scaling normalisation being applied.

There are 2*n+2 columns in file par_mmgmos.txt, n is the number of chips. The first n columns are 'alpha' values for n chips, the next n columns are 'a' values for n chips, column 2*n+1 is 'c' values and the final column is values for 'd'. The file phi_mmgmos.txt keeps the final optimal value of 'phi'.

Value

An object of class `exprReslt`.

Author(s)

Xuejun Liu, Magnus Rattray, Marta Milo, Neil D. Lawrence

References

Liu,X., Milo,M., Lawrence,N.D. and Rattray,M. (2005) A tractable probabilistic model for Affymetrix probe-level analysis across multiple chips, Bioinformatics 21: 3637-3644.

Milo,M., Niranjan,M., Holley,M.C., Rattray,M. and Lawrence,N.D. (2004) A probabilistic approach for summarising oligonucleotide gene expression data, technical report available upon request.

Milo,M., Fazeli,A., Niranjan,M. and Lawrence,N.D. (2003) A probabilistic model for the extractioin of expression levels from oligonucleotide arrays, Biochemical Society Transactions, 31: 1510-1512.

Peter Spellucci. DONLP2 code and accompanying documentation. Electronically available via http://plato.la.asu.edu/donlp2.html

See Also

Related class `exprReslt-class` and related method `mgmos`

Examples

```
if(FALSE){
## Code commented out to speed up checks
## load example data from package affydata
# if (require(pumadata)&&require(puma)){
#  data(oligo.estrogen)
## use method mmgMOS to calculate the expression levels and related confidence
## of the measures for the example data
 # eset<-mmgmos(oligo.estrogen,gsnorm="none")
#}
}
```

`normalisation.gs` *Global scaling normalisation*

Description

This function is only included for backwards compatibility with the **pplr** package. This function is now superceded by `pumaNormalize`.

This function does the global scaling normalisation.

Usage

```
normalisation.gs(x)
```

Arguments

x	a matrix or data frame which contains gene expression level on log2 scale.

Details

Each row of x is related to a gene and each column is related to a chip.

Value

The return matrix is in the same format as the input x.

Author(s)

Xuejun Liu, Marta Milo, Neil D. Lawrence, Magnus Rattray

See Also

See Also as bcomb and hcomb

Examples

```
if(FALSE){
data(exampleE)
exampleE.normalised<-normalisation.gs(exampleE)
data(Clust.exampleE)
Clust.exampleE.normalised<-normalisation.gs(Clust.exampleE)
}
```

---

numFP *Number of False Positives for a given proportion of True Positives.*

Description

Often when evaluating a differential expression method, we are interested in how well a classifier performs for very small numbers of false positives. This method gives one way of calculating this, by determining the number of false positives for a set proportion of true positives.

Usage

```
numFP(scores, truthValues, TPRate = 0.5)
```

Arguments

scores	A vector of scores. This could be, e.g. one of the columns of the statistics of a DEResult object.
truthValues	A boolean vector indicating which scores are True Positives.
TPRate	A number between 0 and 1 identify the proportion of true positives for which we wish to determine the number of false positives.

Value

An integer giving the number of false positives.

Author(s)

Richard D. Pearson

See Also

Related methods plotROC and calcAUC.

Examples

```
if(FALSE){
class1a <- rnorm(1000,0.2,0.1)
class2a <- rnorm(1000,0.6,0.2)
class1b <- rnorm(1000,0.3,0.1)
class2b <- rnorm(1000,0.5,0.2)
scores_a <- c(class1a, class2a)
scores_b <- c(class1b, class2b)
classElts <- c(rep(FALSE,1000), rep(TRUE,1000))
print(numFP(scores_a, classElts))
print(numFP(scores_b, classElts))
}
```

---

numOfFactorsToUse *Determine number of factors to use from an ExpressionSet*

Description

This is really an internal function used to determine how many factors to use in design and contrast matrices

Usage

```
numOfFactorsToUse(eset)
```

Arguments

eset	An object of class ExpressionSet.

Value

An integer denoting the number of factors to be used.

Author(s)

Richard D. Pearson

See Also

Related methods createDesignMatrix and createContrastMatrix

Examples

```
if(FALSE){
 # Next 4 lines commented out to save time in package checks, and saved version used
    # if (require(affydata)) {
 # data(Dilution)
 # eset_mmgmos <- mmgmos(Dilution)
 # }
 data(eset_mmgmos)
 numOfFactorsToUse(eset_mmgmos)
}
```

---

numTP *Number of True Positives for a given proportion of False Positives.*

Description

Often when evaluating a differential expression method, we are interested in how well a classifier performs for very small numbers of true positives. This method gives one way of calculating this, by determining the number of true positives for a set proportion of false positives.

Usage

```
numTP(scores, truthValues, FPRate = 0.5)
```

Arguments

scores	A vector of scores. This could be, e.g. one of the columns of the statistics of a DEResult object.
truthValues	A boolean vector indicating which scores are True Positives.
FPRate	A number between 0 and 1 identify the proportion of flase positives for which we wish to determine the number of true positives.

Value

An integer giving the number of true positives.

Author(s)

Richard D. Pearson

See Also

Related methods numFP, plotROC and calcAUC.

Examples

```
if(FALSE){
class1a <- rnorm(1000,0.2,0.1)
class2a <- rnorm(1000,0.6,0.2)
class1b <- rnorm(1000,0.3,0.1)
class2b <- rnorm(1000,0.5,0.2)
scores_a <- c(class1a, class2a)
```

```
scores_b <- c(class1b, class2b)
classElts <- c(rep(FALSE,1000), rep(TRUE,1000))
print(numTP(scores_a, classElts))
print(numTP(scores_b, classElts))
}
```

`orig_pplr` *Probability of positive log-ratio*

Description

This is the original version of the pplr function as found in the **pplr** package. This should give exactly the same results as the `pplr` function. This function is only included for testing purposes and is not intended to be used. It will not be available in future versions of **puma**.

This function calculates the probability of positive log-ratio (PPLR) between any two specified conditions in the input data, mean and standard deviation of gene expression level for each condition.

Usage

```
orig_pplr(e, control, experiment)
```

Arguments

`e`	a data frame containing the mean and standard deviation of gene expression levels for each condition.
`control`	an integer denoting the control condition.
`experiment`	an integer denoting the experiment condition.

Details

The input of 'e' should be a data frame comprising of 2*n components, where n is the number of conditions. The first 1,2,...,n components include the mean of gene expression values for conditions 1,2,...,n, and the n+1, n+2,...,2*n components contain the standard deviation of expression levels for condition 1,2,...,n.

Value

The return is a data frame. The description of the components are below.

`index`	The original row number of genes.
`cM`	The mean expression levels under control condition.
`sM`	The mean expression levels under experiment condition.
`cStd`	The standard deviation of gene expression levels under control condition.
`sStd`	The standard deviation of gene expression levels under experiment condition.
`LRM`	The mean log-ratio between control and experiment genes.
`LRStd`	The standard deviation of log-ratio between control and experiment genes.
`stat`	A statistic value which is -mean/(sqrt(2)*standard deviation).
`PPLR`	Probability of positive log-ratio.

Author(s)

Xuejun Liu, Marta Milo, Neil D. Lawrence, Magnus Rattray

References

Liu,X., Milo,M., Lawrence,N.D. and Rattray,M. (2006) Probe-level variances improve accuracy in detecting differential gene expression, Bioinformatics, 22(17):2107-13.

See Also

Related method bcomb

Examples

```
if(FALSE){
  data(exampleE)
  data(exampleStd)
  r<-bcomb(exampleE,exampleStd,replicates=c(1,1,1,2,2,2),method="map")
  p<-orig_pplr(r,1,2)
}
```

---

plot-methods *Plot method for pumaPCARes objects*

Description

This is the method to plot objects of class pumaPCARes. It will produce a scatter plot of two of the principal components

Usage

```
## S4 method for signature 'pumaPCARes,missing'
plot(..., firstComponent = 1, secondComponent = 2, useFilenames = FALSE, phenotype = pData(pumaPCARe
```

Arguments

...	Optional graphical parameters to adjust different components of the plot
firstComponent	Integer identifying which principal component to plot on the x-axis
secondComponent	Integer identifying which principal component to plot on the x-axis
useFilenames	Boolean. If TRUE then use filenames as plot points. Otherwise just use points.
phenotype	Phenotype information
legend1pos	String indicating where to put legend for first factor
legend2pos	String indicating where to put legend for second factor

Examples

```
# Next 4 lines commented out to save time in package checks, and saved version used
   # if (require(affydata)) {
# data(Dilution)
# eset_mmgmos <- mmgmos(Dilution)
# }
data(eset_mmgmos)
pumapca_mmgmos <- pumaPCA(eset_mmgmos)
plot(pumapca_mmgmos)
```

---

plotErrorBars *Plot mean expression levels and error bars for one or more probesets*

Description

This produces plots of probesets of interest.

Usage

```
plotErrorBars(
 eset
, probesets = if(dim(exprs(eset))[1] <= 12) 1:dim(exprs(eset))[1] else 1
, arrays = 1:dim(pData(eset))[1] # default is to use all
, xlab = paste(colnames(pData(eset))[1:numOfFactorsToUse(eset)], collapse=":")
, ylab = "Expression Estimate"
, xLabels = apply(
                  as.matrix(pData(eset)[arrays,1:numOfFactorsToUse(eset)])
                , 1
                , function(mat){paste(mat, collapse=":")}
                )
, ylim = NA
, numOfSEs = qnorm(0.975)
, globalYlim = FALSE # Not yet implemented!
, plot_cols = NA
, plot_rows = NA
, featureNames = NA
, showGeneNames = FALSE
, showErrorBars = if(
     length(assayDataElement(eset,"se.exprs"))==0 ||
   length(assayDataElement(eset,"se.exprs")) == sum(is.na(assayDataElement(eset,"se.exprs")))
     ) FALSE else TRUE
, plotColours = FALSE
, log.it = if(max(exprs(eset)) > 32) TRUE else FALSE
, eset_comb = NULL
, jitterWidth = NA
, qtpcrData = NULL
, ...
)
```

Arguments

eset	An object of class ExpressionSet. This is the main object being plotted.
probesets	A vector of integers indicating the probesets to be plotted. These integers refer to the row numbers of the eset.
arrays	A vector of integers indicating the arrays to be shown on plots.
xlab	Character string of title to appear on x-axis
ylab	Character string of title to appear on y-axis
xLabels	Vector of strings for labels of individual points on x-axis.
ylim	2-element numeric vector showing minimum and maximum values for y-axis.
numOfSEs	Numeric indicating the scaling for the error bars. The default value give error bars that include 95% of expected values.
globalYlim	Not yet implemented!
plot_cols	Integer specifying number of columns for multi-figure plot.
plot_rows	Integer specifying number of rows for multi-figure plot.
featureNames	A vector of strings for featureNames (Affy IDs). This is an alternative (to the probesets argument) way of specifying probe sets.
showGeneNames	Boolean indicating whether to use Affy IDs as titles for each plot.
showErrorBars	Boolean indicating whether error bars should be shown on plots.
plotColours	A vector of colours to plot.
log.it	Boolean indicating whether expression values should be logged.
eset_comb	An object of class ExpressionSet. This is a secondary object to be plotted on the same charts as eset. This should be an object created using pumaComb and pumaCombImproved which holds the values created by combining information from the replicates of each condition.
jitterWidth	Numeric indicating the x-axis distance between replicates of the same condition.
qtpcrData	A 2-column matrix of qRT-PCR values (or other data to be plotted on the same charts).
...	Additional arguments to be passed to plot.

Value

This function has no return value. The output is the plot created.

Author(s)

Richard D. Pearson

Examples

```
# Next 4 lines commented out to save time in package checks, and saved version used
   # if (require(affydata)) {
# data(Dilution)
# eset_mmgmos <- mmgmos(Dilution)
# }
data(eset_mmgmos)
plotErrorBars(eset_mmgmos)
plotErrorBars(eset_mmgmos,1:6)
```

`plotHistTwoClasses` *Stacked histogram plot of two different classes*

Description

Stacked histogram plot of two different classes

Usage

```
plotHistTwoClasses(
 scores
, class1Elements
, class2Elements
, space=0
, col=c("white", "grey40")
, xlab="PPLR"
, ylab="Number of genes"
, ylim=NULL
, las=0 # axis labels all perpendicular to axes
, legend=c("non-spike-in genes", "spike-in genes")
, inset=0.05
, minScore=0
, maxScore=1
, numOfBars=20
, main=NULL
)
```

Arguments

`scores`	A numeric vector of scores (e.g. from the output of `pumaDE`)
`class1Elements`	Boolean vector, TRUE if element is in first class
`class2Elements`	Boolean vector, TRUE if element is in second class
`space`	Numeric. x-axis distance between bars
`col`	Colours for the two different classes
`xlab`	Title for the x-axis
`ylab`	Title for the y-axis
`ylim`	2-element numeric vector showing minimum and maximum values for y-axis.
`las`	See `par`. Default of 0 means axis labels all perpendicular to axes.
`legend`	2-element string vector giving text to appear in legend for the two classes.
`inset`	See `legend`
`minScore`	Numeric. Minimum score to plot.
`maxScore`	Numeric. Maximum score to plot.
`numOfBars`	Integer. Number of bars to plot.
`main`	String. Main title for the plot.

Value

This function has no return value. The output is the plot created.

Author(s)

Richard D. Pearson

Examples

```
class1 <- rnorm(1000,0.2,0.1)
class2 <- rnorm(1000,0.6,0.2)
class1[which(class1<0)] <- 0
class1[which(class1>1)] <- 1
class2[which(class2<0)] <- 0
class2[which(class2>1)] <- 1
scores <- c(class1, class2)
class1elts <- c(rep(TRUE,1000), rep(FALSE,1000))
class2elts <- c(rep(FALSE,1000), rep(TRUE,1000))
plotHistTwoClasses(scores, class1elts, class2elts, ylim=c(0,300))
```

---

plotROC *Receiver Operator Characteristic (ROC) plot*

Description

Plots a Receiver Operator Characteristic (ROC) curve.

Usage

```
plotROC(
 scoresList
, truthValues
, includedProbesets=1:length(truthValues)
, legendTitles=1:length(scoresList)
, main = "PUMA ROC plot"
, lty = 1:length(scoresList)
, col = rep(1,length(scoresList))
, lwd = rep(1,length(scoresList))
, yaxisStat = "tpr"
, xaxisStat = "fpr"
, downsampling = 100
, showLegend = TRUE
, showAUC = TRUE
, ...
)
```

Arguments

scoresList	A list, each element of which is a numeric vector of scores.
truthValues	A boolean vector indicating which scores are True Positives.
includedProbesets	A vector of indices indicating which scores (and truthValues) are to be used in the calculation. The default is to use all, but a subset can be used if, for example, you only want a subset of the probesets which are not True Positives to be treated as False Positives.

legendTitles	Vector of names to appear in legend.
main	Main plot title
lty	Line types.
col	Colours.
lwd	Line widths.
yaxisStat	Character string identifying what is to be plotted on the y-axis. The default is "tpr" for True Positive Rate. See performance function from **ROCR** package.
xaxisStat	Character string identifying what is to be plotted on the x-axis. The default is "fpr" for False Positive Rate. See performance function from **ROCR** package.
downsampling	See details for plot.performance from the **ROCR** package.
showLegend	Boolean. Should legend be displayed?
showAUC	Boolean. Should AUC values be included in legend?
...	Other parameters to be passed to plot.

Value

This function has no return value. The output is the plot created.

Author(s)

Richard D. Pearson

See Also

Related method calcAUC

Examples

```
if(FALSE){
 class1a <- rnorm(1000,0.2,0.1)
 class2a <- rnorm(1000,0.6,0.2)
 class1b <- rnorm(1000,0.3,0.1)
 class2b <- rnorm(1000,0.5,0.2)
 scores_a <- c(class1a, class2a)
 scores_b <- c(class1b, class2b)
 scores <- list(scores_a, scores_b)
 classElts <- c(rep(FALSE,1000), rep(TRUE,1000))
 plotROC(scores, classElts)
}
```

---

plotWhiskers *Standard errors whiskers plot*

Description

A plot showing error bars for genes of interest.

Usage

```
plotWhiskers(
 eset
, comparisons=c(1,2)
, sortMethod = c("logRatio", "PPLR")
, numGenes=50
, xlim
, main = "PUMA Whiskers plot"
, highlightedGenes=NULL
)
```

Arguments

eset	An object of class ExpressionSet.
comparisons	A 2-element integer vector specifying the columns of data to be compared.
sortMethod	The method used to sort the genes. "logRatio" is fold change. PPLR is Probability of Positive Log Ratio (as determined by the pumaDE method).
numGenes	Integer. Number of probesets to plot.
xlim	The x limits of the plot. See plot.default.
main	A main title for the plot. See plot.default.
highlightedGenes	Row numbers of probesets to highlight with an asterisk.

Value

This function has no return value. The output is the plot created.

Author(s)

Richard D. Pearson

See Also

Related method pumaDE

PMmmgmos *Multi-chip modified gamma Model for Oligonucleotide Signal using only PM probe intensities*

Description

This function converts an object of class FeatureSet into an object of class exprReslt using the Multi-chip modified gamma Model for Oligonucleotide Signal (PMmulti-mgMOS). This method uses only PM probe intensites. This function obtains confidence of measures, standard deviation and 5, 25, 50, 75 and 95 percentiles, as well as the estimated expression levels.

Usage

```
PMmmgmos(
 object
, background=TRUE
, replaceZeroIntensities=TRUE
, gsnorm=c("median", "none", "mean", "meanlog")
, savepar=FALSE
, eps=1.0e-6
, addConstant = 0
)
```

Arguments

object	an object of FeatureSet
background	Logical value. If TRUE, perform background correction before applying PMmmgmos.
replaceZeroIntensities	Logical value. If TRUE, replace 0 intensities with 1 before applying PMmmgmos.
gsnorm	character. specifying the algorithm of global scaling normalisation.
savepar	Logical value. If TRUE the estimated parameters of the model are saved in file par_pmmmgmos.txt
eps	Optimisation termination criteria.
addConstant	numeric. This is an experimental feature and should not generally be changed from the default value.

Details

The obtained expression measures are in log base 2 scale.

The algorithms of global scaling normalisation can be one of "median", "none", "mean", "meanlog". "mean" and "meanlog" are mean-centered normalisation on raw scale and log scale respectively, and "median" is median-centered normalisation. "none" will result in no global scaling normalisation being applied.

There are n+2 columns in file par_pmmmgmos.txt, n is the number of chips. The first n columns are 'alpha' values for n chips, column n+1 is 'c' values and the final column is values for 'd'.

Value

An object of class exprReslt.

Author(s)

Xuejun Liu, Zhenzhu Gao, Magnus Rattray, Marta Milo, Neil D. Lawrence

References

Liu,X., Milo,M., Lawrence,N.D. and Rattray,M. (2005) A tractable probabilistic model for Affymetrix probe-level analysis across multiple chips, Bioinformatics 21: 3637-3644.

Milo,M., Niranjan,M., Holley,M.C., Rattray,M. and Lawrence,N.D. (2004) A probabilistic approach for summarising oligonucleotide gene expression data, technical report available upon request.

Milo,M., Fazeli,A., Niranjan,M. and Lawrence,N.D. (2003) A probabilistic model for the extractioin of expression levels from oligonucleotide arrays, Biochemical Society Transactions, 31: 1510-1512.

Peter Spellucci. DONLP2 code and accompanying documentation. Electronically available via http://plato.la.asu.edu/donlp2.html

See Also

Related class exprReslt-class and related method mgmos

Examples

```
## Code commented out to speed up checks
## load example data from package pumadata
 #if (require(pumadata)&&require(puma)){
   # data(oligo.estrogen)
## use method PMmmgMOS to calculate the expression levels and related confidence
##of the measures for the example data
 #   eset<-PMmmgmos(oligo.estrogen,gsnorm="none")
#}
```

---

pplr *Probability of positive log-ratio*

Description

WARNING - this function is generally not expected to be used, but is intended as an internal function. It is included for backwards compatibility with the **pplr** package, but may be deprecated and then hidden in future. Users should generally use pumaDE instead.

This function calculates the probability of positive log-ratio (PPLR) between any two specified conditions in the input data, mean and standard deviation of gene expression level for each condition.

Usage

```
pplr(e, control, experiment, sorted=TRUE)
```

Arguments

e	a data frame containing the mean and standard deviation of gene expression levels for each condition.
control	an integer denoting the control condition.
experiment	an integer denoting the experiment condition.
sorted	Boolean. Should PPLR values be sorted by value? If FALSE, PPLR values are returned in same order as supplied.

Details

The input of 'e' should be a data frame comprising of 2*n components, where n is the number of conditions. The first 1,2,...,n components include the mean of gene expression values for conditions 1,2,...,n, and the n+1, n+2,...,2*n components contain the standard deviation of expression levels for condition 1,2,...,n.

Value

The return is a data frame. The description of the components are below.

index	The original row number of genes.
cM	The mean expression levels under control condition.
sM	The mean expression levels under experiment condition.
cStd	The standard deviation of gene expression levels under control condition.
sStd	The standard deviation of gene expression levels under experiment condition.
LRM	The mean log-ratio between control and experiment genes.
LRStd	The standard deviation of log-ratio between control and experiment genes.
stat	A statistic value which is -mean/(sqrt(2)*standard deviation).
PPLR	Probability of positive log-ratio.

Author(s)

Xuejun Liu, Marta Milo, Neil D. Lawrence, Magnus Rattray

References

Liu,X., Milo,M., Lawrence,N.D. and Rattray,M. (2006) Probe-level variances improve accuracy in detecting differential gene expression, Bioinformatics, 22(17):2107-13.

See Also

Related methods pumaDE, bcomb and hcomb

Examples

```
data(exampleE)
data(exampleStd)
r<-bcomb(exampleE,exampleStd,replicates=c(1,1,1,2,2,2),method="map")
p<-pplr(r,1,2)
```

---

pplrUnsorted *Return an unsorted matrix of PPLR values*

Description

Returns the output from pplr as an unsorted matrix (i.e. sorted according to the original sorting in the original matrix)

Usage

```
pplrUnsorted(p)
```

Arguments

p	A matrix as output by pplr.

Value

A matrix of PPLR values

Author(s)

Richard D. Pearson

See Also

Related method pplr

---

pumaClust *Propagate probe-level uncertainty in model-based clustering on gene expression data*

Description

This function clusters gene expression using a Gaussian mixture model including probe-level measurement error. The inputs are gene expression levels and the probe-level standard deviation associated with expression measurement for each gene on each chip. The outputs is the clustering results.

Usage

```
pumaClust(e=NULL, se=NULL, efile=NULL, sefile=NULL,
      subset=NULL, gsnorm=FALSE, clusters,
      iter.max=100, nstart=10, eps=1.0e-6, del0=0.01)
```

Arguments

e	either a valid ExpressionSet object, or a data frame containing the expression level for each gene on each chip.
se	data frame containing the standard deviation of gene expression levels.
efile	character, the name of the file which contains gene expression measurements.
sefile	character, the name of the file which contains the standard deviation of gene expression measurements.
subset	vector specifying the row number of genes which are clustered on.
gsnorm	logical specifying whether do global scaling normalisation or not.
clusters	integer, the number of clusters.
iter.max	integer, the maximum number of iterations allowed in the parameter initialisation.
nstart	integer, the number of random sets chosen in the parameter initialisation.
eps	numeric, optimisation parameter.
del0	numeric, optimisation parameter.

Details

The input data is specified either as an ExpressionSet object (in which case se, efile and sefile will be ignored), or by e and se, or by efile and sefile.

Value

The result is a list with components

cluster: vector, containing the membership of clusters for each gene; centers: matrix, the center of each cluster; centersigs: matrix, the center variance of each cluster; likelipergene: matrix, the likelihood of belonging to each cluster for each gene; bic: numeric, the Bayesian Information Criterion score.

Author(s)

Xuejun Liu, Magnus Rattray

References

Liu,X., Lin,K.K., Andersen,B., and Rattray,M. (2006) Propagating probe-level uncertainty in model-based gene expression clustering, BMC Bioinformatics, 8(98).

Liu,X., Milo,M., Lawrence,N.D. and Rattray,M. (2005) A tractable probabilistic model for Affymetrix probe-level analysis across multiple chips, Bioinformatics, 21(18):3637-3644.

See Also

Related method `mmgmos` and `pumaClustii`

Examples

```
data(Clust.exampleE)
data(Clust.exampleStd)
pumaClust.example<-pumaClust(Clust.exampleE,Clust.exampleStd,clusters=7)
```

`pumaClustii` *Propagate probe-level uncertainty in robust t mixture clustering on replicated gene expression data*

Description

This function clusters gene expression by including uncertainties of gene expression measurements from probe-level analysis models and replicate information into a robust t mixture clustering model. The inputs are gene expression levels and the probe-level standard deviation associated with expression measurement for each gene on each chip. The outputs is the clustering results.

Usage

```
pumaClustii(e=NULL, se=NULL, efile=NULL, sefile=NULL,
      subset=NULL, gsnorm=FALSE, mincls, maxcls, conds, reps, verbose=FALSE,
      eps=1.0e-5, del0=0.01)
```

Arguments

e	data frame containing the expression level for each gene on each chip.
se	data frame containing the standard deviation of gene expression levels.
efile	character, the name of the file which contains gene expression measurements.
sefile	character, the name of the file which contains the standard deviation of gene expression measurements.
subset	vector specifying the row number of genes which are clustered on.
gsnorm	logical specifying whether do global scaling normalisation or not.
mincls	integer, the minimum number of clusters.
maxcls	integer, the maximum number of clusters.
conds	integer, the number of conditions.
reps	vector, specifying which condition each column of the input data matrix belongs to.
verbose	logical value. If 'TRUE' messages about the progress of the function is printed.
eps	numeric, optimisation parameter.
del0	numeric, optimisation parameter.

Details

The input data is specified either by e and se, or by efile and sefile.

Value

The result is a list with components

cluster: vector, containing the membership of clusters for each gene; centers: matrix, the center of each cluster; centersigs: matrix, the center variance of each cluster; likelipergene: matrix, the likelihood of belonging to each cluster for each gene; optK: numeric, the optimal number of clusters. optF: numeric, the maximised value of target function.

Author(s)

Xuejun Liu

References

Liu,X. and Rattray,M. (2009) Including probe-level measurement error in robust mixture clustering of replicated microarray gene expression, Statistical Application in Genetics and Molecular Biology, 9(1), Article 42.

Liu,X., Lin,K.K., Andersen,B., and Rattray,M. (2007) Propagating probe-level uncertainty in model-based gene expression clustering, BMC Bioinformatics, 8:98.

Liu,X., Milo,M., Lawrence,N.D. and Rattray,M. (2005) A tractable probabilistic model for Affymetrix probe-level analysis across multiple chips, Bioinformatics, 21(18):3637-3644.

See Also

Related method mmgmos and pumaclust

Examples

```
data(Clustii.exampleE)
data(Clustii.exampleStd)
r<-vector(mode="integer",0)
for (i in c(1:20))
  for (j in c(1:4))
    r<-c(r,i)
cl<-pumaClustii(Clustii.exampleE,Clustii.exampleStd,mincls=6,maxcls=6,conds=20,reps=r,eps=1e-3)
```

---

pumaComb *Combining replicates for each condition*

Description

This function calculates the combined (from replicates) signal for each condition using Bayesian models. The inputs are gene expression levels and the probe-level standard deviations associated with expression measurements for each gene on each chip. The outputs include gene expression levels and standard deviation for each condition.

Usage

```
pumaComb(
 eset
, design.matrix=NULL
, method="em"
, numOfChunks=1000
, save_r=FALSE
, cl=NULL
, parallelCompute=FALSE
)
```

Arguments

eset	An object of class ExpressionSet.
design.matrix	A design matrix.
method	Method "map" uses MAP of a hierarchical Bayesion model with Gamma prior on the between-replicate variance (Gelman et.al. p.285) and shares the same variance across conditions. This method is fast and suitable for the case where there are many conditions. Method "em" uses variational inference of the same hierarchical Bayesian model as in method "map" but with conjugate prior on between-replicate variance and shares the variance across conditions. This is generaly much slower than "map", but is recommended where there are few conditions (as is usually the case).
numOfChunks	An integer defining how many chunks the data is divided into before processing. There is generally no need to change the default value.
save_r	Will save an internal variable r to a file. Used for debugging purposes.
cl	A "cluster" object. See makeCluster function from **snow** package for more details (if available).
parallelCompute	Boolean identifying whether processing in parallel should occur.

Details

It is generally recommended that data is normalised prior to using this function. Note that the default behaviour of mmgmos is to normalise data so this shouldn't generally be an issue. See the function pumaNormalize for more details on normalisation.

Value

The result is an ExpressionSet object.

Author(s)

Xuejun Liu, Marta Milo, Neil D. Lawrence, Magnus Rattray

References

Gelman,A., Carlin,J.B., Stern,H.S., Rubin,D.B., Bayesian data analysis. London: Chapman & Hall; 1995.

Liu,X., Milo,M., Lawrence,N.D. and Rattray,M. (2006) Probe-level variances improve accuracy in detecting differential gene expression, Bioinformatics, 22:2107-2113.

See Also

Related methods pumaNormalize, bcomb, mmgmos and pumaDE

Examples

```
# Next 4 lines commented out to save time in package checks, and saved version used
   # if (require(affydata)) {
# data(Dilution)
# eset_mmgmos <- mmgmos(Dilution)
# }
data(eset_mmgmos)

# Next line shows that eset_mmgmos has 4 arrays, each of which is a different
#   condition (the experimental design is a 2x2 factorial, with both liver and
# scanner factors)
pData(eset_mmgmos)

# Next line shows expression levels of first 3 probe sets
exprs(eset_mmgmos)[1:3,]

# Next line used so eset_mmgmos only has information about the liver factor
# The scanner factor will thus be ignored, and the two arrays of each level
# of the liver factor will be treated as replicates
pData(eset_mmgmos) <- pData(eset_mmgmos)[,1,drop=FALSE]

# To save time we'll just use 100 probe sets for the example
eset_mmgmos_100 <- eset_mmgmos[1:100,]
eset_comb <- pumaComb(eset_mmgmos_100)

# We can see that the resulting ExpressionSet object has just two conditions
# and 1 expression level for each condition
pData(eset_comb)
exprs(eset_comb)[1:3,]
```

pumaCombImproved *Combining replicates for each condition with the true gene expression*

Description

This function calculates the combined (from replicates) signal for each condition using Bayesian models, which are added a hidden variable to represent the true expression for each gene on each chips. The inputs are gene expression levels and the probe-level standard deviations associated with expression measurements for each gene on each chip. The outputs include gene expression levels and standard deviation for each condition.

Usage

```
pumaCombImproved(
 eset
, design.matrix=NULL
, numOfChunks=1000
,       maxOfIterations=200
, save_r=FALSE
, cl=NULL
, parallelCompute=FALSE
)
```

Arguments

eset	An object of class ExpressionSet.
design.matrix	A design matrix.
numOfChunks	An integer defining how many chunks the data is divided into before processing. There is generally no need to change the default value.
maxOfIterations	The maximum number of iterations controls the convergence.
save_r	Will save an internal variable r to a file. Used for debugging purposes.
cl	A "cluster" object. See makeCluster function from **snow** package for more details (if available).
parallelCompute	Boolean identifying whether processing in parallel should occur.

Details

It is generally recommended that data is normalised prior to using this function. Note that the default behaviour of mmgmos is to normalise data so this shouldn't generally be an issue. See the function pumaNormalize for more details on normalisation.

The maxOfIterations is used to control the maximum number of the iterations in the EM algorithm. You can change the number of maxOfIterations, but the best value of the maxOfIterations is from 200 to 1000, and should be set 200 at least. The default value is 200.

Value

The result is an ExpressionSet object.

Author(s)

Li Zhang, Xuejun Liu

References

Gelman,A., Carlin,J.B., Stern,H.S., Rubin,D.B., Bayesian data analysis. London: Chapman & Hall; 1995.

Zhang,L. and Liu,X. (2009) An improved probabilistic model for finding differential gene expression, technical report available request. the 2nd BMEI 17-19 oct. 2009. Tianjin. China.

Liu,X., Milo,M., Lawrence,N.D. and Rattray,M. (2006) Probe-level variances improve accuracy in detecting differential gene expression, Bioinformatics, 22(17):2107-13.

See Also

Related methods pumaNormalize, hcomb, mmgmos and pumaDE

Examples

```
# Next 4 lines commented out to save time in package checks, and saved version used
   # if (require(affydata)) {
# data(Dilution)
# eset_mmgmos <- mmgmos(Dilution)
# }
data(eset_mmgmos)

# Next line shows that eset_mmgmos has 4 arrays, each of which is a different
#   condition (the experimental design is a 2x2 factorial, with both liver and
# scanner factors)
pData(eset_mmgmos)

# Next line shows expression levels of first 3 probe sets
exprs(eset_mmgmos)[1:3,]

# Next line used so eset_mmgmos only has information about the liver factor
# The scanner factor will thus be ignored, and the two arrays of each level
# of the liver factor will be treated as replicates
pData(eset_mmgmos) <- pData(eset_mmgmos)[,1,drop=FALSE]

# To save time we'll just use 100 probe sets for the example
eset_mmgmos_100 <- eset_mmgmos[1:100,]
eset_combimproved <- pumaCombImproved(eset_mmgmos_100)

# We can see that the resulting ExpressionSet object has just two conditions
# and 1 expression level for each condition
pData(eset_combimproved)
exprs(eset_combimproved)[1:3,]
```

---

pumaDE *Calculate differential expression between conditions*

Description

The function generates lists of genes ranked by probability of differential expression (DE). This uses the PPLR method.

Usage

```
pumaDE(
 eset
, design.matrix = createDesignMatrix(eset)
, contrast.matrix = createContrastMatrix(eset)
)
```

Arguments

eset	An object of class ExpressionSet.
design.matrix	A design matrix
contrast.matrix	A contrast matrix

Details

A separate list of genes will be created for each contrast of interest.

Note that this class returns a DEResult-class object. This object contains information on both the PPLR statistic values (which should generally be used to rank genes for differential expression), as well as fold change values (which are generally not recommended for ranking genes, but which might be useful, for example, to use as a filter). To understand more about the object returned see DEResult-class, noting that when created a DEResult object with the pumaDE function, the statistic method should be used to return PPLR values. Also note that the pLikeValues method can be used on the returned object to create values which can more readily be compared with p-values returned by other methods such as variants of t-tests (limma, etc.).

While it is possible to run this function on data from individual arrays, it is generally recommended that this function is run on the output of the function pumaComb (which combines information from replicates).

Value

An object of class DEResult-class.

Author(s)

Richard D. Pearson

See Also

Related methods calculateLimma, calculateFC, calculateTtest, pumaComb, pumaCombImproved, mmgmos, pplr, createDesignMatrix and createContrastMatrix

Examples

```
# Next 4 lines commented out to save time in package checks, and saved version used
   # if (require(affydata)) {
# data(Dilution)
# eset_mmgmos <- mmgmos(Dilution)
# }
data(eset_mmgmos)

# Next line shows that eset_mmgmos has 4 arrays, each of which is a different
#   condition (the experimental design is a 2x2 factorial, with both liver and
```

```
# scanner factors)
pData(eset_mmgmos)

# Next line shows expression levels of first 3 probe sets
exprs(eset_mmgmos)[1:3,]

# Next line used so eset_mmgmos only has information about the liver factor
# The scanner factor will thus be ignored, and the two arrays of each level
# of the liver factor will be treated as replicates
pData(eset_mmgmos) <- pData(eset_mmgmos)[,1,drop=FALSE]

# To save time we'll just use 100 probe sets for the example
eset_mmgmos_100 <- eset_mmgmos[1:100,]
eset_comb <- pumaComb(eset_mmgmos_100)
       eset_combimproved <- pumaCombImproved(eset_mmgmos_100)

pumaDEResults <- pumaDE(eset_comb)
       pumaDEResults_improved <- pumaDE(eset_combimproved)

topGeneIDs(pumaDEResults,6) # Gives probeset identifiers
topGeneIDs(pumaDEResults_improved,6)
topGenes(pumaDEResults,6) # Gives row numbers
       topGenes(pumaDEResults_improved,6)
statistic(pumaDEResults)[topGenes(pumaDEResults,6),] # PPLR scores of top six genes
statistic(pumaDEResults_improved)[topGenes(pumaDEResults_improved,6),]
FC(pumaDEResults)[topGenes(pumaDEResults,6),] # Fold-change of top six genes
FC(pumaDEResults_improved)[topGenes(pumaDEResults_improved,6),]
```

---

pumaDEUnsorted *Return an unsorted matrix of PPLR values*

Description

Returns the output from pumaDE as an unsorted matrix (i.e. sorted according to the original sorting in the ExpressionSet)

Usage

```
pumaDEUnsorted(pp)
```

Arguments

pp	A list as output by pumaDE.

Value

A matrix of PPLR values

Author(s)

Richard D. Pearson

See Also

Related method pumaDE

pumaFull *Perform a full PUMA analysis*

Description

Full analysis including pumaPCA and mmgmos/pumaDE vs rma/limma comparison

Usage

```
pumaFull (
 ExpressionFeatureSet = NULL
, data_dir = getwd()
, load_ExpressionFeatureSet = FALSE
, calculate_eset = TRUE
, calculate_pumaPCAs = TRUE
, calculate_bcomb = TRUE
, mmgmosComparisons = FALSE
)
```

Arguments

ExpressionFeatureSet	An object of class FeatureSet.
data_dir	A character string specifying where data files are stored.
load_ExpressionFeatureSet	Boolean. Load a pre-existing ExpressionFeatureSet object? Note that this has to be named "ExpressionFeatureSet.rda" and be in the data_dir directory.
calculate_eset	Boolean. Calculate ExpressionSet from ExpressionFeatureSet object? If FALSE, files named "eset_mmgmos.rda" and "eset_rma.rda" must be available in the data_dir directory.
calculate_pumaPCAs	Boolean. Calculate pumaPCA from eset_mmgmos object? If FALSE, a file named "pumaPCA_results.rda" must be available in the data_dir directory.
calculate_bcomb	Boolean. Calculate pumaComb from eset_mmgmos object? If FALSE, files named "eset_comb.rda" and "eset_normd_comb.rda" must be available in the data_dir directory.
mmgmosComparisons	Boolean. If TRUE, will compare mmgmos with default settings, with mmgmos used with background correction.

Value

No return values. Various objects are saved as .rda files during the execution of this function, and various PDF files are created.

Author(s)

Richard D. Pearson

See Also

Related methods pumaDE, createDesignMatrix and createContrastMatrix

Examples

```
## Code commented out to ensure checks run quickly
# if (require(pumadata)) data(oligo.estrogen)
# pumaFull(oligo.estrogen)
```

---

pumaNormalize *Normalize an ExpressionSet*

Description

This is used to apply a scaling normalization to set of arrays. This normalization can be at the array scale (thus giving all arrays the same mean or median), or at the probeset scale (thus giving all probesets the same mean or median).

It is generally recommended that the default option (median array scaling) is used after running mmgmos and before running pumaComb and/or pumaDE. There are however, situations where this might not be the recommended, for example in time series experiments where it is expected than there will be general up-regulation or down-regulation in overall gene expression levels between time points.

Usage

```
pumaNormalize(
 eset
, arrayScale = c("median", "none", "mean", "meanlog")
, probesetScale = c("none", "mean", "median")
, probesetNormalisation = NULL
, replicates = list(1:dim(exprs(eset))[2])
)
```

Arguments

eset	An object of class ExpressionSet.
arrayScale	A method of scale normalisation at the array level.
probesetScale	A method of scale normalisation at the probe set level.
probesetNormalisation	If not NULL normalises the expression levels to have zero mean and adjusts the variance of the gene expression according to the zero-centered normalisation.
replicates	List of integer vectors indicating which arrays are replicates.

Value

An object of class ExpressionSet holding the normalised data.

Author(s)

Richard D. Pearson

See Also

Methods mmgmos, pumaComb and pumaDE

Examples

```
# Next 4 lines commented out to save time in package checks, and saved version used
   # if (require(affydata)) {
# data(Dilution)
# eset_mmgmos <- mmgmos(Dilution)
# }
data(eset_mmgmos)
apply(exprs(eset_mmgmos),2,median)
eset_mmgmos_normd <- pumaNormalize(eset_mmgmos)
apply(exprs(eset_mmgmos_normd),2,median)
```

---

pumaPCA *PUMA Principal Components Analysis*

Description

This function carries out principal components analysis (PCA), taking into account not only the expression levels of genes, but also the variability in these expression levels.

The various other pumaPCA... functions are called during the execution of pumaPCA

Usage

```
pumaPCA(
    eset
,   latentDim           =  if(dim(exprs(eset))[2] <= 3)
         dim(exprs(eset))[[2]]-1
       else
        3
,   sampleSize          = if(dim(exprs(eset))[1] <= 1000)
         dim(exprs(eset))[[1]]
       else
        1000 ## Set to integer or FALSE for all
,   initPCA             = TRUE ## Initialise parameters with PCA
,   randomOrder         = FALSE ## Update parameters in random order
,   optimMethod         = "BFGS"  ## ?optim for details of methods
,   stoppingCriterion   = "deltaW"## can also be "deltaL"
,   tol                 = 1e-3 ## Stop when delta update < this
,   stepChecks          = FALSE ## Check likelihood after each update?
,   iterationNumbers    = TRUE ## Show iteration numbers?
,   showUpdates         = FALSE ## Show values after each update?
,   showTimings         = FALSE ## Show timings after each update?
,   showPlot            = FALSE ## Show projection plot after each update?
,   maxIters            = 500  ## Number of EM iterations.
,   transposeData       = FALSE ## Transpose eset matrices?
,   returnExpectations  = FALSE
,   returnData          = FALSE
,   returnFeedback      = FALSE
```

```
, pumaNormalize  = TRUE
)
```

Arguments

eset	An object of class ExpressionSet.
latentDim	An integer specifying the number of latent dimensions (kind of like the number of principal components).
sampleSize	An integer specifying the number of probesets to sample (default is 1000), or FALSE, meaning use all the data.
initPCA	A boolean indicating whether to initialise using standard PCA (the default, and generally quicker and recommended).
randomOrder	A boolean indicating whether the parameters should be updated in a random order (this is generally not recommended, and the default is FALSE).
optimMethod	See ?optim for details of methods.
stoppingCriterion	If set to "deltaW" will stop when W changes by less than tol. If "deltaL" will stop when L (lambda) changes by less than tol.
tol	Tolerance value for stoppingCriterion.
stepChecks	Boolean. Check likelihood after each update?
iterationNumbers	Boolean. Show iteration numbers?
showUpdates	Boolean. Show values after each update?
showTimings	Boolean. Show timings after each update?
showPlot	Boolean. Show projection plot after each update?
maxIters	Integer. Maximum number of EM iterations.
transposeData	Boolean. Transpose eset matrices?
returnExpectations	Boolean. Return expectation values?
returnData	Boolean. Return expectation data?
returnFeedback	Boolean. Return feedback on progress of optimisation?
pumaNormalize	Boolean. Normalise data prior to running algorithm (recommended)?

Value

An object of class pumaPCARes

Author(s)

Richard D. Pearson

See Also

Related methods pumaDE, createDesignMatrix and createContrastMatrix

Examples

```
# Next 4 lines commented out to save time in package checks, and saved version used
   # if (require(affydata)) {
# data(Dilution)
# eset_mmgmos <- mmgmos(Dilution)
# }
data(eset_mmgmos)

pumapca_mmgmos <- pumaPCA(eset_mmgmos)
plot(pumapca_mmgmos)
```

---

pumaPCAExpectations-class

Class pumaPCAExpectations

Description

This is a class representation for storing a set of expectations from a pumaPCA model. It is an internal representation and shouldn't normally be instantiated.

Objects from the Class

Objects can be created by calls of the form `new("pumaPCAExpectations", ...)`.

Slots

`x`: Object of class "matrix" representing x

`xxT`: Object of class "array" representing xxT

`logDetCov`: Object of class "numeric" representing logDetCov

Methods

This class has no methods defined

Author(s)

Richard D. Pearson

See Also

Related method `pumaPCA` and related class `pumaPCARes`.

pumaPCAModel-class *Class pumaPCAModel*

Description

This is a class representation for storing a pumaPCA model. It is an internal representation and shouldn't normally be instantiated.

Objects from the Class

Objects can be created by calls of the form `new("pumaPCAModel", ...)`.

Slots

`sigma`: Object of class "numeric" representing sigma

`m`: Object of class "matrix" representing m

`Cinv`: Object of class "matrix" representing Cinv

`W`: Object of class "matrix" representing W

`mu`: Object of class "matrix" representing mu

Methods

This class has no methods defined

Author(s)

Richard D. Pearson

See Also

Related method `pumaPCA` and related class `pumaPCARes`.

pumaPCARes-class *Class pumaPCARes*

Description

This is a class representation for storing the outputs of the pumaPCA function. Objects of this class should usually only be created through the `pumaPCA` function.

Objects from the Class

Objects can be created by calls of the form `new("pumaPCARes", ...)`.

Slots

`model`: Object of class "pumaPCAModel" representing the model parameters

`expectations`: Object of class "pumaPCAExpectations" representing the model expectations

`varY`: Object of class "matrix" representing the variance in the expression levels

`Y`: Object of class "matrix" representing the expression levels

`phenoData`: Object of class "AnnotatedDataFrame" representing the phenotype information

`timeToCompute`: Object of class "numeric" representing the time it took `pumaPCA` to run

`numberOfIterations`: Object of class "numeric" representing the number of iterations it took `pumaPCA` to converge

`likelihoodHistory`: Object of class "list" representing the history of likelihood values while `pumaPCA` was running

`timingHistory`: Object of class "list" representing the history of how long each iteration took while `pumaPCA` was running

`modelHistory`: Object of class "list" representing the history of how the model was changing while `pumaPCA` was running

`exitReason`: Object of class "character" representing the reason `pumaPCA` halted. Can take the values "Update of Likelihood less than tolerance x", "Update of W less than tolerance x", "Iterations exceeded", "User interrupt", "unknown exit reason"

Methods

plot `signature(x="pumaPCARes-class")`: plots two principal components on a scatter plot.

write.reslts `signature(x = "pumaPCARes-class")`: writes the principal components for each array to a file. It takes the same arguments as `write.table`. The argument "file" does not need to set any extension. The file name and extension "csv" will be added automatically. The default file name is "tmp".

Author(s)

Richard D. Pearson

See Also

Related method `pumaPCA` and related class `pumaPCARes`.

`removeUninformativeFactors` *Remove uninformative factors from the phenotype data of an ExpressionSet*

Description

This is really an internal function used to remove uninformative factors from the phenotype data. Uninformative factors here are defined as those which have the same value for all arrays in the `ExpressionSet`.

Usage

```
removeUninformativeFactors(eset)
```

Arguments

eset	An object of class ExpressionSet.

Value

An ExpressionSet object with the same data as the input, except for a new phenoData slot.

Author(s)

Richard D. Pearson

See Also

Related methods createDesignMatrix and createContrastMatrix

Examples

```
eset_test <- new("ExpressionSet", exprs=matrix(rnorm(400,8,2),100,4))
pData(eset_test) <- data.frame("informativeFactor"=c("A", "A", "B", "B"), "uninformativeFactor"=c("X","X","X
eset_test2 <- removeUninformativeFactors(eset_test)
pData(eset_test)
pData(eset_test2)
```

Index