Package: limma (via r-universe)

November 1, 2026

Version 3.99.4

Date 2026-11-01

Title Linear Models for Microarray and Omics Data

Description Data analysis, linear models and differential expression for omics data.

License GPL (>=2)

Depends R (>= 3.6.0)

Imports grDevices, graphics, stats, utils, methods, statmod

Suggests BiasedUrn, ellipse, gplots, knitr, locfit, MASS, splines, affy, AnnotationDbi, Biobase, BiocStyle, GO.db, illuminaio, org.Hs.eg.db, SummarizedExperiment, vsn

VignetteBuilder knitr

URL https://bioinf.wehi.edu.au/limma/

biocViews ExonArray, GeneExpression, Transcription, AlternativeSplicing, DifferentialExpression, DifferentialSplicing, GeneSetEnrichment, DataImport, Bayesian, Clustering, Regression, TimeCourse, Microarray, MicroRNAArray, mRNAMicroarray, OneChannel, ProprietaryPlatforms, TwoChannel, Sequencing, RNASeq, BatchEffect, MultipleComparison, Normalization, Preprocessing, QualityControl, BiomedicalInformatics, CellBiology, Cheminformatics, Epigenetics, FunctionalGenomics, Genetics, ImmunoOncology, Metabolomics, Proteomics, SystemsBiology, Transcriptomics

Repository https://bioc.r-universe.dev

Date/Publication 2026-11-01 07:12:50 UTC

RemoteUrl https://github.com/bioc/limma

RemoteRef HEAD

RemoteSha 498f86398f0cb3ad2f74fa3d3588155c0d1f51c9

Contents

01.Introduction . . . 5
02.Classes . . . 7
03.ReadingData . . . 8
04.Background . . . 9
05.Normalization . . . 10
06.LinearModels . . . 11
07.SingleChannel . . . 13
08.Tests . . . 14
09.Diagnostics . . . 15
10.GeneSetTests . . . 16
11.RNAseq . . . 17
alias2Symbol . . . 18
anova.MAList-method . . . 20
arrayWeights . . . 21
arrayWeightsQuick . . . 23
as.data.frame . . . 24
as.MAList . . . 25
as.matrix . . . 26
asMatrixWeights . . . 27
auROC . . . 28
avearrays . . . 29
avedups . . . 30
avereps . . . 31
backgroundCorrect . . . 32
barcodeplot . . . 34
beadCountWeights . . . 38
blockDiag . . . 40
bwss . . . 41
bwss.matrix . . . 42
camera . . . 43
cbind . . . 46
changeLog . . . 47
chooseLowessSpan . . . 48
classifyTestsF . . . 50
contrastAsCoef . . . 52
contrasts.fit . . . 53
controlStatus . . . 55
coolmap . . . 56
cumOverlap . . . 58
decideTests . . . 59
designI2M . . . 61
detectionPValues . . . 62
diffSplice . . . 64
dim . . . 66
dimnames . . . 67
dupcor . . . 68

eBayes . . . 70
EList-class . . . 75
exprs.MA . . . 76
fitFDist . . . 77
fitGammaIntercept . . . 80
fitmixture . . . 81
fitted.MArrayLM . . . 82
genas . . . 83
geneSetTest . . . 85
getEAWP . . . 88
getLayout . . . 89
getSpacing . . . 90
gls.series . . . 91
goana . . . 93
goanaTrend . . . 98
gridr . . . 100
head . . . 100
heatdiagram . . . 101
helpMethods . . . 103
ids2indices . . . 104
imageplot . . . 105
imageplot3by2 . . . 106
intraspotCorrelation . . . 107
is.fullrank . . . 109
isNumeric . . . 110
kooperberg . . . 111
LargeDataObject-class . . . 112
limmaUsersGuide . . . 113
lm.series . . . 114
lmFit . . . 115
lmscFit . . . 119
loessFit . . . 120
logcosh . . . 122
logsumexp . . . 123
ma3x3 . . . 124
makeContrasts . . . 125
makeUnique . . . 126
MAList-class . . . 127
MArrayLM-class . . . 128
mdplot . . . 129
merge . . . 130
mergeScans . . . 132
modelMatrix . . . 133
modifyWeights . . . 135
mrlm . . . 136
nec . . . 137
normalizeBetweenArrays . . . 139
normalizeCyclicLoess . . . 142

normalizeForPrintorder . . . 144
normalizeMedianAbsValues . . . 146
normalizeQuantiles . . . 147
normalizeRobustSpline . . . 148
normalizeVSN . . . 149
normalizeWithinArrays . . . 150
normexp.fit . . . 153
normexp.fit.control . . . 154
normexp.fit.detection.p . . . 156
normexp.signal . . . 158
plotDensities . . . 159
plotExonJunc . . . 160
plotExons . . . 162
plotFB . . . 163
plotlines . . . 164
plotMA . . . 165
plotMA3by2 . . . 168
plotMD . . . 169
plotMDS . . . 172
plotPrintTipLoess . . . 174
plotRLDF . . . 175
plotSA . . . 177
plotSplice . . . 179
plotWithHighlights . . . 180
poolVar . . . 182
predFCm . . . 183
printHead . . . 185
PrintLayout . . . 186
printorder . . . 187
printtipWeights . . . 188
propexpr . . . 190
propTrueNull . . . 191
protectMetachar . . . 193
qqt . . . 194
QualityWeights . . . 195
rankSumTestWithCorrelation . . . 196
read.columns . . . 198
read.idat . . . 199
read.ilmn . . . 201
read.ilmn.targets . . . 203
read.maimages . . . 204
readGAL . . . 208
readHeader . . . 209
readImaGeneHeader . . . 210
readSpotTypes . . . 211
readTargets . . . 212
removeBatchEffect . . . 213
removeExt . . . 215

residuals.MArrayLM . . . 216
RGList-class . . . 217
roast . . . 218
romer . . . 223
sampleInfoFromGEO . . . 225
selectModel . . . 227
squeezeVar . . . 228
strsplit2 . . . 230
subsetting . . . 231
summary . . . 233
targetsA2C . . . 234
TestResults-class . . . 235
tmixture . . . 236
topGO . . . 237
topRomer . . . 238
topSplice . . . 239
topTable . . . 241
tricubeMovingAverage . . . 244
trigammaInverse . . . 245
trimWhiteSpace . . . 246
uniquegenelist . . . 247
unwrapdups . . . 248
venn . . . 249
volcanoplot . . . 250
voom . . . 252
vooma . . . 255
voomaLmFit . . . 258
voomLmFit . . . 261
voomWithQualityWeights . . . 265
weighted.median . . . 267
weightedLowess . . . 268
write.fit . . . 271
wsva . . . 273
zscore . . . 274
zscoreT . . . 275

Index **278**

`01.Introduction` *Introduction to the LIMMA Package*

Description

LIMMA is a package for the analysis of gene expression microarray data, especially the use of linear models for analyzing designed experiments and the assessment of differential expression. LIMMA provides the ability to analyze comparisons between many RNA targets simultaneously in arbitrary complicated designed experiments. Empirical Bayesian methods are used to provide stable results

even when the number of arrays is small. The linear model and differential expression functions apply to all gene expression technologies, including microarrays, RNA-seq and quantitative PCR.

Details

There are three types of documentation available:

1. The *LIMMA User's Guide* can be reached through the "User Guides and Package Vignettes" links at the top of the LIMMA contents page. The function `limmaUsersGuide` gives the file location of the User's Guide.

2. An overview of limma functions grouped by purpose is contained in the numbered chapters at the foot of the LIMMA package index page, of which this page is the first.

3. The LIMMA contents page gives an alphabetical index of detailed help topics.

The function `changeLog` displays the record of changes to the package.

Author(s)

Gordon Smyth, with contributions from many colleagues

References

Law CW, Chen Y, Shi W, Smyth GK (2014). Voom: precision weights unlock linear model analysis tools for RNA-seq read counts. *Genome Biology* 15, R29. doi:10.1186/gb2014152r29. See also the preprint version at `https://gksmyth.github.io/pubs/VoomPreprint.pdf` incorporating some notational corrections.

Phipson B, Lee S, Majewski IJ, Alexander WS, Smyth GK (2016). Robust hyperparameter estimation protects against hypervariable genes and improves power to detect differential expression. *Annals of Applied Statistics* 10(2), 946-963. doi:10.1214/16AOAS920.

Ritchie ME, Phipson B, Wu D, Hu Y, Law CW, Shi W, Smyth GK (2015). limma powers differential expression analyses for RNA-sequencing and microarray studies. *Nucleic Acids Research* 43(7), e47. doi:10.1093/nar/gkv007.

Smyth GK (2004). Linear models and empirical Bayes methods for assessing differential expression in microarray experiments. *Statistical Applications in Genetics and Molecular Biology* 3(1), Article 3. doi:10.2202/15446115.1027. See also the preprint version `https://gksmyth.github.io/pubs/ebayes.pdf` incorporating corrections to 30 June 2009.

See Also

02.Classes, 03.ReadingData, 04.Background, 05.Normalization, 06.LinearModels, 07.SingleChannel, 08.Tests, 09.Diagnostics, 10.GeneSetTests, 11.RNAseq

02.Classes *Topic: Classes Defined by this Package*

Description

This package defines the following data classes.

RGList A class used to store raw intensities as they are read in from an image analysis output file, usually by read.maimages.

MAList Intensities converted to M-values and A-values, i.e., to with-spot and whole-spot contrasts on the log-scale. Usually created from an RGList using MA.RG or normalizeWithinArrays. Objects of this class contain one row for each spot. There may be more than one spot and therefore more than one row for each probe.

EListRaw A class to store raw intensities for one-channel microarray data. May or may not be background corrected. Usually created by read.maimages.

EList A class to store normalized log2 expression values for one-channel microarray data. Usually created by normalizeBetweenArrays.

MArrayLM Store the result of fitting gene-wise linear models to the normalized intensities or log-ratios. Usually created by lmFit. Objects of this class normally contain only one row for each unique probe.

TestResults Store the results of testing a set of contrasts equal to zero for each probe. Usually created by decideTests. Objects of this class normally contain one row for each unique probe.

All these data classes obey many analogies with matrices. In the case of RGList, MAList, EListRaw and EList, rows correspond to spots or probes and columns to arrays. In the case of MArrayLM, rows correspond to unique probes and the columns to parameters or contrasts. The functions summary, dim, length, ncol, nrow, dimnames, rownames, colnames have methods for these classes. Objects of any of these classes may be subsetted. Multiple data objects may be combined by rows (to add extra probes) or by columns (to add extra arrays).

Furthermore all of these classes may be coerced to actually be of class matrix using as.matrix, although this entails loss of information. Fitted model objects of class MArrayLM can be coerced to class data.frame using as.data.frame.

The first three classes belong to the virtual class LargeDataObject. A show method is defined for LargeDataObjects which uses the utility function printHead.

Author(s)

Gordon Smyth

See Also

01.Introduction, 02.Classes, 03.ReadingData, 04.Background, 05.Normalization, 06.LinearModels, 07.SingleChannel, 08.Tests, 09.Diagnostics, 10.GeneSetTests, 11.RNAseq

03.ReadingData *Topic: Reading Microarray Data from Files*

Description

This help page gives an overview of LIMMA functions used to read data from files.

Reading Target Information

The function readTargets is designed to help with organizing information about which RNA sample is hybridized to each channel on each array and which files store information for each array.

Reading Intensity Data

The first step in a microarray data analysis is to read into R the intensity data for each array provided by an image analysis program. This is done using the function read.maimages.

read.maimages optionally constructs quality weights for each spot using quality functions listed in QualityWeights.

If the data is two-color, then read.maimages produces an RGList object. If the data is one-color (single channel), then an EListRaw object is produced. In either case, read.maimages stores only the information required from each image analysis output file. read.maimages uses utility functions removeExt, read.imagene and read.columns. There are also a series of utility functions which read the header information from image output files including readGPRHeader, readImaGeneHeader and readGenericHeader.

read.ilmn reads probe or gene summary profile files from Illumina BeadChips, and produces an EListRaw object.

read.idat reads Illumina files in IDAT format, and produces an EListRaw object. detectionPValues can be used to add detection p-values.

The function as.MAList can be used to convert a marrayNorm object to an MAList object if the data was read and normalized using the marray and marrayNorm packages.

Reading the Gene List

Most image analysis software programs provide gene IDs as part of the intensity output files, for example GenePix, Imagene and the Stanford Microarray Database do this. In other cases the probe ID and annotation information may be in a separate file. The most common format for the probe annotation file is the GenePix Array List (GAL) file format. The function readGAL reads information from a GAL file and produces a data frame with standard column names.

The function getLayout extracts from the GAL-file data frame the print layout information for a spotted array. The functions gridr, gridc, spotr and spotc use the extracted layout to compute grid positions and spot positions within each grid for each spot. The function printorder calculates the printorder, plate number and plate row and column position for each spot given information about the printing process. The utility function getSpacing converts character strings specifying spacings of duplicate spots to numeric values.

The Australian Genome Research Facility often produces GAL files with composite probe IDs or names consisting of multiple strings separated by a delimiter. These can be separated into name and annotation information using strsplit2.

If each probe is printed more than once on the arrays in a regular pattern, then uniquegenelist will remove duplicate names from the gal-file or gene list.

Identifying Control Spots

The functions readSpotTypes and controlStatus assist with separating control spots from ordinary genes in the analysis and data exploration.

Manipulating Data Objects

cbind, rbind, merge allow different RGList or MAList objects to be combined. cbind combines data from different arrays assuming the layout of the arrays to be the same. merge can combine data even when the order of the probes on the arrays has changed. merge uses utility function makeUnique.

Author(s)

Gordon Smyth

See Also

01.Introduction, 02.Classes, 03.ReadingData, 04.Background, 05.Normalization, 06.LinearModels, 07.SingleChannel, 08.Tests, 09.Diagnostics, 10.GeneSetTests, 11.RNAseq

---

04.Background *Topic: Background Correction*

Description

This page deals with background correction methods provided by the backgroundCorrect, kooperberg or neqc functions. Microarray data is typically background corrected by one of these functions before normalization and other downstream analysis.

backgroundCorrect works on matrices, EListRaw or RGList objects, and calls backgroundCorrect.matrix.

The movingmin method of backgroundCorrect uses utility functions ma3x3.matrix and ma3x3.spottedarray.

The normexp method of backgroundCorrect uses utility functions normexp.fit and normexp.signal.

kooperberg is a Bayesian background correction tool designed specifically for two-color GenePix data. It is computationally intensive and requires several additional columns from the GenePix data files. These can be read in using read.maimages and specifying the other.columns argument.

neqc is for single-color data. It performs normexp background correction and quantile normalization using control probes. It uses utility functions normexp.fit.control and normexp.signal. If robust=TRUE, then normexp.fit.control uses the function huber in the MASS package.

Author(s)

Gordon Smyth

See Also

01.Introduction, 02.Classes, 03.ReadingData, 04.Background, 05.Normalization, 06.LinearModels, 07.SingleChannel, 08.Tests, 09.Diagnostics, 10.GeneSetTests, 11.RNAseq

---

`05.Normalization` *Topic: Normalization of Microarray Data*

Description

This page gives an overview of the LIMMA functions available to normalize data from single-channel or two-color microarrays. Smyth and Speed (2003) give an overview of the normalization techniques implemented in the functions for two-color arrays.

Usually data from spotted microarrays will be normalized using `normalizeWithinArrays`. A minority of data will also be normalized using `normalizeBetweenArrays` if diagnostic plots suggest a difference in scale between the arrays.

In rare circumstances, data might be normalized using `normalizeForPrintorder` before using `normalizeWithinArrays`.

All the normalization routines take account of spot quality weights which might be set in the data objects. The weights can be temporarily modified using `modifyWeights` to, for example, remove ratio control spots from the normalization process.

If one is planning analysis of single-channel information from the microarrays rather than analysis of differential expression based on log-ratios, then the data should be normalized using a single channel-normalization technique. Single channel normalization uses further options of the `normalizeBetweenArrays` function. For more details see the LIMMA User's Guide which includes a section on single-channel normalization.

`normalizeWithinArrays` uses utility functions `MA.RG`, `loessFit` and `normalizeRobustSpline`.

`normalizeBetweenArrays` is the main normalization function for one-channel arrays, as well as an optional function for two-color arrays. `normalizeBetweenArrays` uses utility functions `normalizeMedianValues`, `normalizeMedianAbsValues`, `normalizeQuantiles` and `normalizeCyclicLoess`, none of which need to be called directly by users. `normalizeCyclicLoess` calls `chooseLowessSpan` when `adaptive.span=TRUE`.

`neqc` is a between array normalization function customized for Illumina BeadChips.

The function `normalizeVSN` is also provided as an interface to the vsn package. It performs variance stabilizing normalization, an algorithm which includes background correction, within and between normalization together, and therefore doesn't fit into the paradigm of the other methods.

`removeBatchEffect` can be used to remove a batch effect, associated with hybridization time or some other technical variable, prior to unsupervised analysis.

Author(s)

Gordon Smyth

References

Smyth GK, Speed TP (2003). Normalization of cDNA microarray data. *Methods* 31, 265-273. doi:10.1016/S10462023(03)001555. See also the preprint at https://gksmyth.github.io/pubs/normalize.pdf.

See Also

01.Introduction, 02.Classes, 03.ReadingData, 04.Background, 05.Normalization, 06.LinearModels, 07.SingleChannel, 08.Tests, 09.Diagnostics, 10.GeneSetTests, 11.RNAseq

---

`06.LinearModels` *Topic: Linear Models for Microarrays*

Description

This page gives an overview of the LIMMA functions available to fit linear models and to interpret the results. This page covers models for two color arrays in terms of log-ratios or for single-channel arrays in terms of log-intensities. If you wish to fit models to the individual channel log-intensities from two colour arrays, see 07.SingleChannel.

The core of this package is the fitting of gene-wise linear models to microarray data. The basic idea is to estimate log-ratios between two or more target RNA samples simultaneously. See the LIMMA User's Guide for several case studies.

Fitting Models

The main function for model fitting is `lmFit`. This is the recommended interface for most users. `lmFit` produces a fitted model object of class `MArrayLM` containing coefficients, standard errors and residual standard errors for each gene. `lmFit` calls one of the following three functions to do the actual computations:

`lm.series` Straightforward least squares fitting of a linear model for each gene.

`mrlm` An alternative to `lm.series` using robust regression as implemented by the `rlm` function in the MASS package.

`gls.series` Generalized least squares taking into account correlations between duplicate spots (i.e., replicate spots on the same array) or related arrays. The function `duplicateCorrelation` is used to estimate the inter-duplicate or inter-block correlation before using `gls.series`.

All the functions which fit linear models use `getEAWP` to extract data from microarray data objects, and `unwrapdups` which provides a unified method for handling duplicate spots.

Forming the Design Matrix

`lmFit` has two main arguments, the expression data and the design matrix. The design matrix is essentially an indicator matrix which specifies which target RNA samples were applied to each channel on each array. There is considerable freedom in choosing the design matrix - there is always more than one choice which is correct provided it is interpreted correctly.

Design matrices for Affymetrix or single-color arrays can be created using the function `model.matrix` which is part of the R base package. The function `modelMatrix` is provided to assist with creation of an appropriate design matrix for two-color microarray experiments. For direct two-color designs, without a common reference, the design matrix often needs to be created by hand.

Making Comparisons of Interest

Once a linear model has been fit using an appropriate design matrix, the command `makeContrasts` may be used to form a contrast matrix to make comparisons of interest. The fit and the contrast matrix are used by `contrasts.fit` to compute fold changes and t-statistics for the contrasts of interest. This is a way to compute all possible pairwise comparisons between treatments, for example, in an experiment which compares many treatments to a common reference.

Assessing Differential Expression

After fitting a linear model, the standard errors are moderated using a simple empirical Bayes model using `eBayes` or `treat`. A moderated t-statistic and a log-odds of differential expression is computed for each contrast for each gene. `treat` tests whether log-fold-changes are greater than a threshold rather than merely different to zero.

`eBayes` and `treat` use internal functions `squeezeVar`, `fitFDist`, `fitFDistRobustly`, `fitFDistUnequalDF1`, `tmixture.matrix` and `tmixture.vector`.

Summarizing Model Fits

After the above steps the results may be displayed or further processed using:

`topTable` Presents a list of the genes most likely to be differentially expressed for a given contrast or set of contrasts.

`topTreat` Presents a list of the genes most likely to be differentially expressed for one specified coefficient or contrast, following `treat`.

`volcanoplot` Volcano plot of fold change versus the B-statistic for any fitted coefficient.

`plotlines` Plots fitted coefficients or log-intensity values for time-course data.

`genas` Estimates and plots biological correlation between two coefficients.

`write.fit` Writes an `MArrayLM` object to a file. Note that if `fit` is an `MArrayLM` object, either `write.fit` or `write.table` can be used to write the results to a delimited text file.

For multiple testing functions which operate on linear model fits, see 08.Tests.

Model Selection

`selectModel` provides a means to choose between alternative linear models using AIC or BIC information criteria.

Author(s)

Gordon Smyth

References

Phipson B, Lee S, Majewski IJ, Alexander WS, Smyth GK (2016). Robust hyperparameter estimation protects against hypervariable genes and improves power to detect differential expression. *Annals of Applied Statistics* 10(2), 946-963. doi:10.1214/16AOAS920.

Smyth GK (2004). Linear models and empirical Bayes methods for assessing differential expression in microarray experiments. *Statistical Applications in Genetics and Molecular Biology*, 3(1), Article 3. doi:10.2202/15446115.1027. See also the preprint version `https://gksmyth.github.io/pubs/ebayes.pdf` with corrections.

Smyth GK, Michaud J, Scott H (2005). The use of within-array replicate spots for assessing differential expression in microarray experiments. *Bioinformatics* 21(9), 2067-2075. doi:10.1093/bioinformatics/bti270.

See Also

01.Introduction, 02.Classes, 03.ReadingData, 04.Background, 05.Normalization, 06.LinearModels, 07.SingleChannel, 08.Tests, 09.Diagnostics, 10.GeneSetTests, 11.RNAseq

---

`07.SingleChannel` *Topic: Individual Channel Analysis of Two-Color Microarrays*

Description

This page gives an overview of the LIMMA functions used to fit linear models to two-color microarray data in terms of the log-intensities rather than log-ratios.

The function `intraspotCorrelation` estimates the intra-spot correlation between the two channels. The regression function `lmscFit` takes the correlation as an argument and fits linear models to the two-color data in terms of the individual log-intensities. The output of `lmscFit` is an `MArrayLM` object just the same as from `lmFit`, so inference proceeds in the same way as for log-ratios once the linear model is fitted. See 06.LinearModels.

The function `targetsA2C` converts two-color format target data frames to single channel format, i.e., converts from array-per-line to channel-per-line, to facilitate the formulation of the design matrix.

Author(s)

Gordon Smyth

See Also

01.Introduction, 02.Classes, 03.ReadingData, 04.Background, 05.Normalization, 06.LinearModels, 07.SingleChannel, 08.Tests, 09.Diagnostics, 10.GeneSetTests, 11.RNAseq

08.Tests *Topic: Hypothesis Testing for Linear Models*

Description

LIMMA provides a number of functions for multiple testing across both contrasts and genes. The starting point is an MArrayLM object, called fit say, resulting from fitting a linear model and running eBayes and, optionally, contrasts.fit. See 06.LinearModels or 07.SingleChannel for details.

Multiple testing across genes and contrasts

The key function is decideTests. This function writes an object of class TestResults, which is basically a matrix of -1, 0 or 1 elements, of the same dimension as fit$coefficients, indicating whether each coefficient is significantly different from zero. A number of different multiple testing strategies are provided. decideTests calls classifyTestsF to implement the nested F-test strategy.

selectModel chooses between linear models for each probe using AIC or BIC criteria. This is an alternative to hypothesis testing and can choose between non-nested models.

A number of other functions are provided to display the results of decideTests. The function heatDiagram (or the older version heatdiagram) displays the results in a heat-map style display. This allows visual comparison of the results across many different conditions in the linear model.

The functions vennCounts and vennDiagram provide Venn diagram style summaries of the results.

Summary and show methods exist for objects of class TestResults.

The results from decideTests can also be included when the results of a linear model fit are written to a file using write.fit.

Gene Set Tests

Competitive gene set testing for an individual gene set is provided by wilcoxGST or geneSetTest, which permute genes. The gene set can be displayed using barcodeplot.

Self-contained gene set testing for an individual set is provided by roast, which uses rotation technology, analogous to permuting arrays.

Gene set enrichment analysis for a large database of gene sets is provided by romer. topRomer is used to rank results from romer.

The functions alias2Symbol, alias2SymbolTable and alias2SymbolUsingNCBI are provided to help match gene sets with microarray probes by way of official gene symbols.

Global Tests

The function genas can test for associations between two contrasts in a linear model.

Given a set of p-values, the function propTrueNull can be used to estimate the proportion of true null hypotheses.

When evaluating test procedures with simulated or known results, the utility function auROC can be used to compute the area under the Receiver Operating Curve for the test results for a given probe.

Author(s)

Gordon Smyth

See Also

01.Introduction, 02.Classes, 03.ReadingData, 04.Background, 05.Normalization, 06.LinearModels, 07.SingleChannel, 08.Tests, 09.Diagnostics, 10.GeneSetTests, 11.RNAseq

---

`09.Diagnostics` *Topic: Diagnostics and Quality Assessment*

Description

This page gives an overview of the LIMMA functions available for microarray quality assessment and diagnostic plots.

This package provides an `anova` method which is designed for assessing the quality of an array series or of a normalization method. It is not designed to assess differential expression of individual genes. `anova` uses utility functions `bwss` and `bwss.matrix`.

The function `arrayWeights` estimates the empirical reliability of each array following a linear model fit.

Diagnostic plots can be produced by

`imageplot` Produces a spatial picture of any spot-specific measure from an array image. If the log-ratios are plotted, then this produces an in-silico representation of the well known false-color TIFF image of an array. `imageplot3by2` will write imageplots to files, six plots to a page.

`plotFB` Plots foreground versus background log-intensities.

`plotMD` Mean-difference plots. Very versatile plot. For two color arrays, this plots the M-values vs A-values. For single channel technologies, this plots one column of log-expression values vs the average of the other columns. For fitted model objects, this plots a log-fold-change versus average log-expression. `mdplot` can also be useful for comparing two one-channel microarrays.

`plotMA` MA-plots, essentially the same as mean-difference plots. `plotMA3by2` will write MA-plots to files, six plots to a page.

`plotWithHighlights` Scatterplots with highlights. This is the underlying engine for `plotMD` and `plotMA`.

`plotPrintTipLoess` Produces a grid of MA-plots, one for each print-tip group on an array, together with the corresponding lowess curve. Intended to help visualize print-tip loess normalization.

`plotPrintorder` For an array, produces a scatter plot of log-ratios or log-intensities by print order.

`plotDensities` Individual channel densities for one or more arrays. An essential plot to accompany between array normalization, especially quantile normalization.

plotMDS Multidimensional scaling plot for a set of arrays. Useful for visualizing the relationship between the set of samples.

plotSA Sigma vs A plot. After a linear model is fitted, this checks constancy of the variance with respect to intensity level.

plotPrintTipLoess uses utility functions gridr and gridc. plotDensities uses utility function RG.MA.

Author(s)

Gordon Smyth

See Also

01.Introduction, 02.Classes, 03.ReadingData, 04.Background, 05.Normalization, 06.LinearModels, 07.SingleChannel, 08.Tests, 09.Diagnostics, 10.GeneSetTests, 11.RNAseq

---

10.GeneSetTests *Topic: Gene Set Tests*

Description

This page gives an overview of the LIMMA functions for gene set testing and pathway analysis.

roast Self-contained gene set testing for one set. Uses zscoreT to normalize t-statistics.

mroast Self-contained gene set testing for many sets. Uses zscoreT to normalize t-statistics.

fry Fast approximation to mroast, especially useful when heteroscedasticity of genes can be ignored.

camera Competitive gene set testing.

cameraPR Competitive gene set testing with a pre-ranked gene set.

romer **and** topRomer Gene set enrichment analysis.

ids2indices Convert gene sets consisting of vectors of gene identifiers into a list of indices suitable for use in the above functions.

alias2Symbol**,** alias2SymbolTable **and** alias2SymbolUsingNCBI Convert gene symbols or aliases to current official symbols.

geneSetTest **or** wilcoxGST Simple gene set testing based on gene or probe permutation.

barcodeplot Enrichment plot of a gene set.

goana **and** topGO Gene ontology over-representation analysis of gene lists using Entrez Gene IDs. goana can work directly on a fitted model object or on one or more lists of genes.

kegga **and** topKEGG KEGG pathway over-representation analysis of gene lists using Entrez Gene IDs. kegga can work directly on a fitted model object or on one or more lists of genes.

Author(s)

Gordon Smyth

See Also

01.Introduction, 02.Classes, 03.ReadingData, 04.Background, 05.Normalization, 06.LinearModels, 07.SingleChannel, 08.Tests, 09.Diagnostics, 10.GeneSetTests, 11.RNAseq

---

`11.RNAseq` *Topic: Analysis of RNA-seq Data*

Description

This page gives an overview of LIMMA functions to analyze RNA-seq data.

`voom` Transform RNA-seq or ChIP-seq counts to log counts per million (log-cpm) with associated precision weights. After this tranformation, RNA-seq or ChIP-seq data can be analyzed using the same functions as would be used for microarray data.

`voomWithQualityWeights` Combines the functionality of `voom` and `arrayWeights`.

`diffSplice` Test for differential transcript or exon usage between experimental conditions.

`topSplice` Show a data.frame of top results from `diffSplice`.

`plotSplice` Plot results from `diffSplice`.

`plotExons` Plot logFC for individual exons for a given gene.

References

Law CW, Chen Y, Shi W, Smyth GK (2014). Voom: precision weights unlock linear model analysis tools for RNA-seq read counts. *Genome Biology* 15, R29. doi:10.1186/gb2014152r29.

Ritchie ME, Phipson B, Wu D, Hu Y, Law CW, Shi W, Smyth GK (2015). limma powers differential expression analyses for RNA-sequencing and microarray studies. *Nucleic Acids Research* 43, e47. doi:10.1093/nar/gkv007.

See Also

See also the edgeR package for normalization and data summaries of RNA-seq data, as well as for alternative differential expression methods based on the negative binomial distribution. `voom` accepts DGEList objects and normalization factors from edgeR.

The limma function `voomLmFit` is a drop-in replacement for either `voom` or `voomWithQualityWeights`.

01.Introduction, 02.Classes, 03.ReadingData, 04.Background, 05.Normalization, 06.LinearModels, 07.SingleChannel, 08.Tests, 09.Diagnostics, 10.GeneSetTests, 11.RNAseq

alias2Symbol *Convert Gene Aliases to Official Gene Symbols*

Description

Maps gene alias names to official gene symbols.

Usage

```
alias2Symbol(alias, species = "Hs", expand.symbols = FALSE)
alias2SymbolTable(alias, species = "Hs")
alias2SymbolUsingNCBI(alias, gene.info.file,
                      required.columns = c("GeneID","Symbol","description"))
```

Arguments

alias	character vector of gene aliases
species	character string specifying the species. Possible values include "Hs" (human), "Mm" (mouse), "Rn" (rat), "Dm" (fly) or "Pt" (chimpanzee), but other values are possible if the corresponding organism package is available.
expand.symbols	logical. This affects those elements of alias that are the official gene symbol for one gene and also an alias for another gene. If FALSE, then these elements will just return themselves. If TRUE, then all the genes for which they are aliases will also be returned.
gene.info.file	either the name of a gene information file downloaded from the NCBI or a data.frame resulting from reading such a file.
required.columns	character vector of columns from the gene information file that are required in the output.

Details

Aliases are mapped via NCBI Entrez Gene ID numbers using Bioconductor organism packages.

alias2Symbol maps a set of aliases to a set of symbols, without necessarily preserving order. The output vector may be longer or shorter than the original vector, because some aliases might not be found and some aliases may map to more than one symbol.

alias2SymbolTable returns a vector of the same length as the vector of aliases. If an alias maps to more than one symbol, then the one with the lowest Entrez ID number is returned. If an alias can't be mapped, then NA is returned.

species can be any character string XX for which an organism package org.XX.eg.db exists and is installed. The only requirement of the organism package is that it contains objects org.XX.egALIAS2EG and org.XX.egSYMBOL linking the aliases and symbols to Entrez Gene Ids. At the time of writing, the following organism packages are available from Bioconductor 3.16:

Package	Species
org.Ag.eg.db	Anopheles
org.Bt.eg.db	Bovine
org.Ce.eg.db	Worm
org.Cf.eg.db	Canine
org.Dm.eg.db	Fly
org.Dr.eg.db	Zebrafish
org.EcK12.eg.db	E coli strain K12
org.EcSakai.eg.db	E coli strain Sakai
org.Gg.eg.db	Chicken
org.Hs.eg.db	Human
org.Mm.eg.db	Mouse
org.Mmu.eg.db	Rhesus
org.Pt.eg.db	Chimp
org.Rn.eg.db	Rat
org.Ss.eg.db	Pig
org.Xl.eg.db	Xenopus

`alias2SymbolUsingNCBI` is analogous to `alias2SymbolTable` but uses a gene-info file from NCBI instead of a Bioconductor organism package. It also gives the option of returning multiple columns from the gene-info file. NCBI gene-info files can be downloaded from https://ftp.ncbi.nlm.nih.gov/gene/DATA/GENE_INFO/. For example, the human file is https://ftp.ncbi.nlm.nih.gov/gene/DATA/GENE_INFO/Mammalia/Homo_sapiens.gene_info.gz and the mouse file is ftp://ftp.ncbi.nlm.nih.gov/gene/DATA/GENE_INFO/Mammalia/Mus_musculus.gene_info.gz.

Value

`alias2Symbol` and `alias2SymbolTable` produce a character vector of gene symbols. `alias2SymbolTable` returns a vector of the same length and order as `alias`, including `NA` values where no gene symbol was found. `alias2Symbol` returns an unordered vector that may be longer or shorter than `alias`.

`alias2SymbolUsingNCBI` returns a data.frame with rows corresponding to the entries of `alias` and columns as specified by `required.columns`.

Author(s)

Gordon Smyth and Yifang Hu

See Also

This function is often used to assist gene set testing, see 10.GeneSetTests.

Examples

```
alias2Symbol(c("PUMA","NOXA","BIM"), species="Hs")
alias2Symbol("RS1", expand=TRUE)
```

anova.MAList-method *ANOVA Table - method*

Description

Analysis of variance method for objects of class MAList. Produces an ANOVA table useful for quality assessment by decomposing between and within gene sums of squares for a series of replicate arrays. This method produces a single ANOVA Table rather than one for each gene and is not used to identify differentially expressed genes.

Usage

```
anova(object,design=NULL,ndups=2,...)
```

Arguments

object	object of class MAList. Missing values in the M-values are not allowed.
design	numeric vector or single-column matrix containing the design matrix for linear model. The length of the vector or the number of rows of the matrix should agree with the number of columns of M.
ndups	number of duplicate spots. Each gene is printed ndups times in adjacent spots on each array.
...	other arguments are not used

Details

This function aids in quality assessment of microarray data and in the comparison of normalization methodologies. It applies only to replicated two-color experiments in which all the arrays are hybridized with the same RNA targets, possibly with dye-swaps, so the design matrix should have only one column. The function has not been heavily used and is somewhat experimental.

Value

An object of class anova containing rows for between genes, between arrays, gene x array interaction, and between duplicates within arrays sums of squares. Variance components are estimated for each source of variation.

Note

This function does not give valid results in the presence of missing M-values.

Author(s)

Gordon Smyth

See Also

MAList-class, bwss.matrix, anova.

An overview of quality assessment and diagnostic functions in LIMMA is given by 09.Diagnostics.

arrayWeights *Array Quality Weights*

Description

Estimate relative quality weights for each array or group in a multi-array experiment.

Usage

```
arrayWeights(object, design = NULL, weights = NULL,
     var.design = NULL, var.group = NULL, prior.n = 10,
     method = "auto", maxiter = 50L, tol = 1e-5, trace = FALSE,
     nthreads = 1L)
```

Arguments

object	any matrix-like object containing log-expression values or log-ratio expression values, for example an EList or ExpressionSet object. See help("getEAWP") for a list of possible classes.
design	the design matrix of the microarray experiment, with rows corresponding to arrays and columns to coefficients to be estimated. Defaults to the unit vector meaning that the arrays are treated as replicates.
weights	numeric matrix containing prior weights for each expression value.
var.design	design matrix for the variance model. Defaults to the sample-specific model whereby each sample has a distinct variance.
var.group	vector or factor indicating groups to have different array weights. This is another way to specify var.design for groupwise variance models.
prior.n	prior number of genes. Larger values squeeze the array weights more strongly towards equality.
method	character string specifying the estimating algorithm to be used. Choices are "genebygene", "reml" or "auto".
maxiter	maximum number of iterations allowed when method="reml".
tol	convergence tolerance when method="reml".
trace	logical. If TRUE then progress information is printed at each iteration of the "reml" algorithm or at every 1000th gene for the "genebygene" algorithm.
nthreads	positive integer giving the number of OpenMP threads used by the REML algorithm. Defaults to 1.

Details

The relative reliability of each array is estimated by measuring how well the expression values for that array follow the linear model. Arrays that tend to have larger residuals are assigned lower weights.

The basic method is described by Ritchie et al (2006) and the extension to custom variance models by Liu et al (2015). A weighted linear model is fitted to the expression values for each gene. The variance model is fitted to the squared residuals from the linear model fit and is updated either by full REML scoring iterations (`method="reml"`) or using an efficient gene-by-gene update algorithm (`method="genebygene"`). The final estimates of these array variances are converted to weights. The gene-by-gene algorithm is described by Ritchie et al (2006) while the REML algorithm is an adaptation of the algorithm of Smyth (2002).

For stability, the array weights are squeezed slightly towards equality. This is done by adding a prior likelihood corresponding to unit array weights equivalent to `prior.n` genes. The gene-by-gene algorithm is started from the prior genes while the REML algorithm adds the prior to the log-likelihood derivatives.

By default, `arrayWeights` chooses between the REML and gene-by-gene algorithms automatically (`method="auto"`). REML is chosen if there are no prior weights or missing values and otherwise the gene-by-gene algorithm is used.

The input `object` is interpreted as for `lmFit` and `getEAWP`. In particular, the arguments `design` and `weights` will be extracted from the data `object` if available and do not normally need to be set explicitly in the call; if any of these are set in the call then they will over-ride the slots or components in the data `object`.

Value

A numeric vector of array weights, which multiply to 1.

Author(s)

Matthew Ritchie and Gordon Smyth

References

Liu R, Holik AZ, Su S, Jansz N, Chen K, Leong HS, Blewitt ME, Asselin-Labat ML, Smyth GK, Ritchie ME (2015). Why weight? Combining voom with estimates of sample quality improves power in RNA-seq analyses. *Nucleic Acids Research* 43, e97. doi:10.1093/nar/gkv412.

Ritchie ME, Diyagama D, Neilson J, van Laar R, Dobrovic A, Holloway A, Smyth GK (2006). Empirical array quality weights in the analysis of microarray data. *BMC Bioinformatics* 7, 261. doi:10.1186/147121057261.

Smyth GK (2002). An efficient algorithm for REML in heteroscedastic regression. *Journal of Computational and Graphical Statistics* 11, 836-847. https://gksmyth.github.io/pubs/remlalgo.pdf

See Also

`arrayWeightsQuick`, `voomWithQualityWeights`

An overview of linear model functions in limma is given by 06.LinearModels.

Examples

```
ngenes <- 1000
narrays <- 6
y <- matrix(rnorm(ngenes*narrays), ngenes, narrays)
var.group <- c(1,1,1,2,2,2)
y[,var.group==1] <- 2*y[,var.group==1]
arrayWeights(y, var.group=var.group)
```

---

arrayWeightsQuick *Array Quality Weights*

Description

Estimates relative quality weights for each array in a multi-array experiment with replication.

Usage

```
arrayWeightsQuick(y, fit)
```

Arguments

y	the data object used to estimate fit. Can be of any class which can be coerced to matrix, including matrix, MAList, marrayNorm or ExpressionSet.
fit	MArrayLM fitted model object

Details

Estimates the relative reliability of each array by measuring how well the expression values for that array follow the linear model.

This is a quick and dirty version of arrayWeights.

Value

Numeric vector of weights of length ncol(fit).

Author(s)

Gordon Smyth

References

Ritchie ME, Diyagama D, Neilson J, van Laar R, Dobrovic A, Holloway A, Smyth GK (2006). Empirical array quality weights in the analysis of microarray data. BMC Bioinformatics 7, 261. doi:10.1186/147121057261.

See Also

See arrayWeights. An overview of linear model functions in limma is given by 06.LinearModels.

Examples

```
## Not run:
fit <- lmFit(y, design)
arrayWeightsQuick(y, fit)

## End(Not run)
```

`as.data.frame` *Turn a Microarray Linear Model Object into a Dataframe*

Description

Turn a `MArrayLM` object into a `data.frame`.

Usage

```
## S3 method for class 'MArrayLM'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`	an object of class `MArrayLM`
`row.names`	`NULL` or a character vector giving the row names for the data frame. Missing values are not allowed.
`optional`	logical. If `TRUE`, setting row names and converting column names (to syntactic names) is optional.
`...`	additional arguments to be passed to or from methods.

Details

This method combines all the components of `x` which have a row for each probe on the array into a `data.frame`.

Value

A data.frame.

Author(s)

Gordon Smyth

See Also

`as.data.frame` in the base package.

02.Classes gives an overview of data classes used in LIMMA. 06.LinearModels gives an overview of linear model functions in LIMMA.

`as.MAList` *Convert marrayNorm Object to an MAList Object*

Description

Convert marrayNorm Object to an MAList Object

Usage

```
as.MAList(object)
```

Arguments

`object`	an `marrayNorm` object

Details

The `marrayNorm` class is defined in the `marray` package. This function converts a normalized two color microarray data object created by the `marray` package into the corresponding limma data object.

Note that such conversion is not necessary to access the limma linear modeling functions, because `lmFit` will operate on a `marrayNorm` data object directly.

Value

Object of class `MAList`

Author(s)

Gordon Smyth

See Also

02.Classes gives an overview of all the classes defined by this package.

The marrayNorm class is defined in the marray package.

as.matrix *Turn a Microarray Data Object into a Matrix*

Description

Turn a microarray data object into a numeric matrix by extracting the expression values.

Usage

```
## S3 method for class 'MAList'
as.matrix(x,...)
```

Arguments

x	an object of class RGList, MAList, EList, MArrayLM, marrayNorm, PLMset, ExpressionSet, LumiBatch or vsn.
...	additional arguments, not used for these methods.

Details

These methods extract the matrix of log-ratios, for MAList or marrayNorm objects, or the matrix of expression values for other expression objects such as EList or ExpressionSet. For MArrayLM objects, the matrix of fitted coefficients is extracted.

These methods involve loss of information, so the original data object is not recoverable.

Value

A numeric matrix.

Author(s)

Gordon Smyth

See Also

as.matrix in the base package or exprs in the Biobase package.

02.Classes gives an overview of data classes used in LIMMA.

asMatrixWeights *asMatrixWeights*

Description

Convert probe-weights or array-weights to a matrix of weights.

Usage

```
asMatrixWeights(weights, dim)
```

Arguments

weights	numeric matrix of weights, rows corresponding to probes and columns to arrays. Or vector of probe weights. Or vector of array weights.
dim	numeric dimension vector of length 2, i.e., the number of probes and the number of arrays.

Details

This function converts a vector of probe-weights or a vector of array-weights to a matrix of the correct size. Probe-weights are repeated across rows while array-weights are repeated down the columns. If weights has length equal to the number of probes, it is assumed to contain probe-weights. If it has length equal to the number of arrays, it is assumed to contain array-weights. If the number of probes is equal to the number of arrays, then weights is assumed to contain array-weights if it is a row-vector of the correct size, i.e., if it is a matrix with one row.

This function is used internally by the linear model fitting functions in limma.

Value

Numeric matrix of dimension dim.

Author(s)

Gordon Smyth

See Also

modifyWeights.

An overview of functions in LIMMA used for fitting linear models is given in 06.LinearModels.

Examples

```
asMatrixWeights(1:3,c(4,3))
asMatrixWeights(1:4,c(4,3))
```

auROC *Area Under Receiver Operating Curve*

Description

Compute exact area under the ROC for empirical data.

Usage

```
auROC(truth, stat=NULL)
```

Arguments

truth	logical vector, or numeric vector of 0s and 1s, indicating whether each case is a true positive.
stat	numeric vector containing test statistics used to rank cases, from largest to smallest. If NULL, then truth is assumed to be already sorted in decreasing test statistic order.

Details

A receiver operating curve (ROC) is a plot of sensitivity (true positive rate) versus 1-specificity (false positive rate) for a statistical test or binary classifier. The area under the ROC is a well accepted measure of test performance. It is equivalent to the probability that a randomly chosen pair of cases is correctly ranked.

Here we consider a test statistic stat, with larger values being more significant, and a vector truth indicating whether the alternative hypothesis is in fact true. truth==TRUE or truth==1 indicates a true discovery and truth=FALSE or truth=0 indicates a false discovery. Correct ranking here means that truth[i] is greater than or equal to truth[j] when stat[i] is greater than stat[j]. The function computes the exact area under the empirical ROC curve defined by truth when ordered by stat.

If stat contains ties, then auROC returns the average area under the ROC for all possible orderings of truth for tied stat values.

The area under the curve is undefined if truth is all TRUE or all FALSE or if truth or stat contain missing values.

Value

Numeric value between 0 and 1 giving area under the curve, 1 being perfect and 0 being the minimum.

Author(s)

Gordon Smyth

Examples

```
auROC(c(1,1,0,0,0))
truth <- rbinom(30,size=1,prob=0.2)
stat <- rchisq(30,df=2)
auROC(truth,stat)
```

---

avearrays *Average Over Replicate Arrays*

Description

Condense a microarray data object so that technical replicate arrays are replaced with (weighted) averages.

Usage

```
## Default S3 method:
avearrays(x, ID=colnames(x), weights=NULL)
## S3 method for class 'MAList'
avearrays(x, ID=colnames(x), weights=x$weights)
## S3 method for class 'EList'
avearrays(x, ID=colnames(x), weights=x$weights)
```

Arguments

x	a matrix-like object, usually a matrix, MAList or EList object.
ID	sample identifier.
weights	numeric matrix of non-negative weights

Details

A new data object is computed in which technical replicate arrays are replaced by their (weighted) averages.

For an MAList object, the components M and A are both averaged in this way, as are any matrices found in object$other.

EList objects are similar, except that the E component is averaged instead of M and A.

The weights matrix, if present, will be used to compute weighted averages of all the other matrices over replicate arrays. For both MAList and EList objects, the weights themselves are summed instead of averaged.

The default method will work for character as well as numeric matrices. If x is of mode "character", then the first column for each unique ID is taken as the average.

Value

A data object of the same class as x with a column for each unique value of ID.

Author(s)

Gordon Smyth

See Also

avereps.

02.Classes gives an overview of data classes used in LIMMA.

Examples

```
x <- matrix(rnorm(8*3),8,3)
colnames(x) <- c("a","a","b")
avearrays(x)
```

---

avedups *Average Over Duplicate Spots*

Description

Condense a microarray data object so that values for within-array replicate spots are replaced with their average.

Usage

```
## Default S3 method:
avedups(x, ndups=2, spacing=1, weights=NULL)
## S3 method for class 'MAList'
avedups(x, ndups=x$printer$ndups, spacing=x$printer$spacing, weights=x$weights)
## S3 method for class 'EList'
avedups(x, ndups=x$printer$ndups, spacing=x$printer$spacing, weights=x$weights)
```

Arguments

x	a matrix-like object, usually a matrix, MAList or EList object.
ndups	number of within-array replicates for each probe.
spacing	number of spots to step from a probe to its duplicate.
weights	numeric matrix of spot weights.

Details

A new data object is computed in which each probe is represented by the (weighted) average of its duplicate spots. For an MAList object, the components M and A are both averaged in this way. For an EList object, the component E is averaged in this way.

If x is of mode "character", then the duplicate values are assumed to be equal and the first is taken as the average.

Value

A data object of the same class as x with 1/ndups as many rows.

Author(s)

Gordon Smyth

See Also

avereps.

02.Classes gives an overview of data classes used in LIMMA.

---

avereps *Average Over Irregular Replicate Probes*

Description

Condense a microarray data object so that values for within-array replicate probes are replaced with their average.

Usage

```
## Default S3 method:
avereps(x, ID=rownames(x), ...)
## S3 method for class 'MAList'
avereps(x, ID=NULL, ...)
## S3 method for class 'EList'
avereps(x, ID=NULL, ...)
```

Arguments

x	a matrix-like object, usually a matrix, MAList or EList object.
ID	probe identifier.
...	other arguments are not currently used.

Details

A new data object is computed in which each probe ID is represented by the average of its replicate spots or features.

For an MAList object, the components M and A are both averaged in this way, as are weights and any matrices found in object$other. For an MAList object, ID defaults to MA$genes$ID if that exists, otherwise to rownames(MA$M).

EList objects are similar, except that the E component is averaged instead of M and A.

If x is of mode "character", then the replicate values are assumed to be equal and the first is taken as the average.

Value

A data object of the same class as x with a row for each unique value of ID.

Note

This function should only be applied to normalized log-expression values, and not to raw unlogged expression values. It will generate an error message if applied to RGList or EListRaw objects.

Author(s)

Gordon Smyth

See Also

avedups, avearrays. Also rowsum in the base package.

02.Classes gives an overview of data classes used in LIMMA.

Examples

```
x <- matrix(rnorm(8*3),8,3)
colnames(x) <- c("S1","S2","S3")
rownames(x) <- c("b","a","a","c","c","b","b","b")
avereps(x)
```

---

backgroundCorrect *Correct Intensities for Background*

Description

Background correct microarray expression intensities.

Usage

```
backgroundCorrect(RG, method="auto", offset=0, printer=RG$printer,
                  normexp.method="saddle", verbose=TRUE)
backgroundCorrect.matrix(E, Eb=NULL, method="auto", offset=0, printer=NULL,
                  normexp.method="saddle", verbose=TRUE)
```

Arguments

RG	a numeric matrix, EListRaw or RGList object.
E	numeric matrix containing foreground intensities.
Eb	numeric matrix containing background intensities.
method	character string specifying correction method. Possible values are "auto", "none", "subtract", "half", "minimum", "movingmin", "edwards" or "normexp". If RG is a matrix, possible values are restricted to "none" or "normexp". The default "auto" is interpreted as "subtract" if background intensities are available or "normexp" if they are not.

offset	numeric value to add to intensities
printer	a list containing printer layout information, see PrintLayout-class. Ignored if RG is a matrix.
normexp.method	character string specifying parameter estimation strategy used by normexp, ignored for other methods. Possible values are "saddle", "mle", "rma" or "rma75".
verbose	logical. If TRUE, progress messages are sent to standard output

Details

This function implements the background correction methods reviewed or developed in Ritchie et al (2007) and Silver et al (2009). Ritchie et al (2007) recommend method="normexp" whenever RG contains local background estimates. Silver et al (2009) shows that either normexp.method="mle" or normexp.method="saddle" are excellent options for normexp. If RG contains morphological background estimates instead (available from SPOT or GenePix image analysis software), then method="subtract" performs well.

If method="none" then no correction is done, i.e., the background intensities are treated as zero. If method="subtract" then the background intensities are subtracted from the foreground intensities. This is the traditional background correction method, but is not necessarily recommended. If method="movingmin" then the background estimates are replaced with the minimums of the backgrounds of the spot and its eight neighbors, i.e., the background is replaced by a moving minimum of 3x3 grids of spots.

The remaining methods are all designed to produce positive corrected intensities. If method="half" then any intensity which is less than 0.5 after background subtraction is reset to be equal to 0.5. If method="minimum" then any intensity which is zero or negative after background subtraction is set equal to half the minimum of the positive corrected intensities for that array. If method="edwards" a log-linear interpolation method is used to adjust lower intensities as in Edwards (2003). If method="normexp" a convolution of normal and exponential distributions is fitted to the foreground intensities using the background intensities as a covariate, and the expected signal given the observed foreground becomes the corrected intensity. This results in a smooth monotonic transformation of the background subtracted intensities such that all the corrected intensities are positive.

The normexp method is available in a number of variants depending on how the model parameters are estimated, and these are selected by normexp.method. Here "saddle" gives the saddle-point approximation to maximum likelihood from Ritchie et al (2007) and improved by Silver et al (2009), "mle" gives exact maximum likelihood from Silver et al (2009), "rma" gives the background correction algorithm from the RMA-algorithm for Affymetrix microarray data as implemented in the affy package, and "rma75" gives the RMA-75 method from McGee and Chen (2006). In practice "mle" performs well and is nearly as fast as "saddle", but "saddle" is the default for backward compatibility. See normexp.fit for more details.

The offset can be used to add a constant to the intensities before log-transforming, so that the log-ratios are shrunk towards zero at the lower intensities. This may eliminate or reverse the usual 'fanning' of log-ratios at low intensities associated with local background subtraction.

Background correction (background subtraction) is also performed by the normalizeWithinArrays method for RGList objects, so it is not necessary to call backgroundCorrect directly unless one wants to use a method other than simple subtraction. Calling backgroundCorrect before normalizeWithinArrays will over-ride the default background correction.

Value

A matrix, EListRaw or RGList object in which foreground intensities have been background corrected and any components containing background intensities have been removed.

Author(s)

Gordon Smyth

References

Edwards DE (2003). Non-linear normalization and background correction in one-channel cDNA microarray studies *Bioinformatics* 19, 825-833.

McGee M, Chen Z (2006). Parameter estimation for the exponential-normal convolution model for background correction of Affymetrix GeneChip data. *Statistical Applications in Genetics and Molecular Biology* 5(1), Article 24.

Ritchie ME, Silver J, Oshlack A, Holmes M, Diyagama D, Holloway A, Smyth GK (2007). A comparison of background correction methods for two-colour microarrays. *Bioinformatics* 23, 2700-2707. doi:10.1093/bioinformatics/btm412.

Silver JD, Ritchie ME, Smyth GK (2009). Microarray background correction: maximum likelihood estimation for the normal-exponential convolution. *Biostatistics* 10, 352-363. doi:10.1093/biostatistics/kxn042.

See Also

kooperberg, neqc.

An overview of background correction functions is given in 04.Background.

Examples

```
RG <- new("RGList", list(R=c(1,2,3,4),G=c(1,2,3,4),Rb=c(2,2,2,2),Gb=c(2,2,2,2)))
backgroundCorrect(RG)
backgroundCorrect(RG, method="half")
backgroundCorrect(RG, method="minimum")
backgroundCorrect(RG, offset=5)
```

---

barcodeplot *Barcode Enrichment Plot*

Description

Display the enrichment of one or two gene sets in a ranked gene list.

Usage

```
barcodeplot(statistics, index = NULL, index2 = NULL, gene.weights = NULL,
            weights.label = "Weight", labels = c("Down","Up"),
            quantiles = c(-1,1)*sqrt(2), col.bars = NULL, alpha = 0.4,
            worm = TRUE, span.worm = 0.45, xlab = "Statistic", ...)
```

Arguments

statistics	numeric vector giving the values of statistics to rank genes by.
index	index vector for the gene set. This can be a vector of indices, or a logical vector of the same length as statistics or, in general, any vector such that statistic[index] gives a subset of the statistic values. Can be omitted if gene.weights has same length as statistics, in which case positive values of gene.weights indicate members of the positive set and negative weights correspond to members of the negative set.
index2	optional index vector for a second (negative) gene set. If specified, then index and index2 specify positive and negative genes respectively. Usually used to distinguish down-regulated genes from up-regulated genes.
gene.weights	numeric vector giving directional weights for the genes in the (first) set. Positive and negative weights correspond to positive and negative genes. Ignored if index2 is non-null.
weights.label	label describing the entries in gene.weights.
labels	character vector of labels for low and high statistics. First label is associated with low statistics or negative statistics and is displayed at the left end of the plot. Second label is associated with high or positive statistics and is displayed at the right end of the plot.
quantiles	numeric vector of length 2, giving cutoff values for statistics considered small or large respectively. Used to color the rectangle of the barcodeplot.
col.bars	character vector of colors for the vertical bars of the barcodeplot showing the ranks of the gene set members. Defaults to "black" for one set or c("red","blue") for two sets.
alpha	transparency for vertical bars. When gene.weights are not NULL, values 0<alpha<1 give semitransparent colors for the vertical bars inside the rectangle. This helps distinguish position bars from the weighted bars and also helps to show the density of the bars when there are many bars. Ignored if gene.weights=NULL.
worm	logical, should enrichment worms be plotted?
span.worm	loess span for enrichment worms. Larger spans give smoother worms.
xlab	x-axis label for statistics.
...	other arguments are passed to plot.

Details

The function displays the enrichment of a specified gene set signature in a ranked list of genes. The vector statistics defines the ranking of the population of genes. This vector can represent any useful ranking but often it provides t-statistics or log-fold-changes arising from a differential analysis. The gene set signature is defined either by index and index2 or by gene.weights.

The signature can be either unidirectional or bidirectional. A unidirectional signature is a simple set of genes (defined by index), optionally accompanied by a set of positive magnitude scores (specified by gene.weights). Typically this is a set of genes representing a pathway or biological process that are expected to be co-regulated in the same direction. A bidirectional signature consists of a set of up-genes and a set of down-genes (specified by index and index2 respectively) or, more

generally, a set of genes with accompanying magnitude scores that are either positive or negative (specified by `gene.weights`).

Technically, this function plots the positions of one or two gene sets in a ranked list of statistics. If there are two sets, then one is considered to be the positive set and the other the negative set. For example, the first set and second sets often correspond to genes that are expected to be up- or down-regulated respectively. The function can optionally display varying weights for different genes, for example log-fold-changes from a previous experiment.

The statistics are ranked left to right from smallest to largest. The ranked statistics are represented by a shaded bar or bed, and the positions of the specified subsets are marked by vertical bars, forming a pattern like a barcode. An enrichment worm optionally shows the relative enrichment of the vertical bars in each part of the plot. The worm is computed by the `tricubeMovingAverage` function.

Barcode plots are often used in conjunction with gene set tests, and show the enrichment of gene sets amongst high or low ranked genes. They were inspired by the set location plot of Subramanian et al (2005), with a number of enhancements, especially the ability to plot positive and negative sets simultaneously. Barcode plots first appeared in the literature in Lim et al (2009). More recent examples can be seen in Liu et al (2014), Sheikh et al (2015), Witkowski et al (2015) and Ng et al (2015).

The function can be used with any of four different calling sequences:

- `index` is specified, but not `index2` or `gene.weights`. Single direction plot.
- `index` and `index2` are specified. Two directional plot.
- `index` and `gene.weights` are specified. `gene.weights` must have same length as `statistics[index]`. Plot will be two-directional if `gene.weights` contains positive and negative values.
- `gene.weights` is specified but not `index` or `index2`. `gene.weights` must have same length as `statistics`. Plot will be two-directional if `gene.weights` contains positive and negative values.

Value

No value is returned but a plot is produced as a side effect.

Author(s)

Yifang Hu, Gordon Smyth and Di Wu

References

Ng AP, Hu Y, Metcalf D, Hyland CD, Ierino H, Phipson B, Wu D, Baldwin TM, Kauppi M, Kiu H, Di Rago L, Hilton DJ, Smyth GK, Alexander WS (2015). Early lineage priming by trisomy of Erg leads to myeloproliferation in a down syndrome model. *PLOS Genetics* 11, e1005211. doi:10.1371/journal.pgen.1005211.

Lim E, Vaillant F, Wu D, Forrest NC, Pal B, Hart AH, Asselin-Labat ML, Gyorki DE, Ward T, Partanen A, Feleppa F, Huschtscha LI, Thorne HJ; kConFab; Fox SB, Yan M, French JD, Brown MA, Smyth GK, Visvader JE, Lindeman GJ (2009). Aberrant luminal progenitors as the candidate target population for basal tumor development in BRCA1 mutation carriers. *Nature Medicine* 15, 907-913. doi:10.1038/nm.2000.

Liu GJ, Cimmino L, Jude JG, Hu Y, Witkowski MT, McKenzie MD, Kartal-Kaess M, Best SA, Tuohey L, Liao Y, Shi W, Mullighan CG, Farrar MA, Nutt SL, Smyth GK, Zuber J, Dickins RA (2014). Pax5 loss imposes a reversible differentiation block in B progenitor acute lymphoblastic leukemia. *Genes & Development* 28, 1337-1350. doi:10.1101/gad.240416.114.

Sheikh B, Lee S, El-saafin F, Vanyai H, Hu Y, Pang SHM, Grabow S, Strasser A, Nutt SL, Alexander WS, Smyth GK, Voss AK, Thomas T (2015). MOZ regulates B cell progenitors in mice, consequently, Moz haploinsufficiency dramatically retards MYC-induced lymphoma development. *Blood* 125, 1910-1921. doi:10.1182/blood201408594655.

Subramanian A, Tamayo P, Mootha VK, Mukherjee S, Ebert BL, Gillette MA, Paulovich A, Pomeroy SL, Golub TR, Lander ES, Mesirov JP (2005). Gene set enrichment analysis: a knowledge-based approach for interpreting genome-wide expression profiles. *Proc Natl Acad Sci USA* 102, 15545-15550.

Witkowski MT, Cimmino L, Hu Y, Trimarchi T, Tagoh H, McKenzie MD, Best SA, Tuohey L, Willson TA, Nutt SL, Busslinger M, Aifantis I, Smyth GK, Dickins RA (2015). Activated Notch counteracts Ikaros tumor suppression in mouse and human T cell acute lymphoblastic leukemia. *Leukemia* 29, 1301-1311. doi:10.1038/leu.2015.27.

See Also

tricubeMovingAverage, roast, camera, romer, geneSetTest

There is a topic page on 10.GeneSetTests.

Examples

```
stat <- rnorm(100)
sel <- 1:10
sel2 <- 11:20
stat[sel] <- stat[sel]+1
stat[sel2] <- stat[sel2]-1

# One directional
barcodeplot(stat, index = sel)

# Two directional
barcodeplot(stat, index = sel, index2 = sel2)

# Second set can be indicated by negative weights
barcodeplot(stat, index = c(sel,sel2), gene.weights = c(rep(1,10), rep(-1,10)))

# Two directional with unequal weights
w <- rep(0,100)
w[sel] <- runif(10)
w[sel2] <- -runif(10)
barcodeplot(stat, gene.weights = w, weights.label = "logFC")

# One directional with unequal weights
w <- rep(0,100)
w[sel2] <- -runif(10)
barcodeplot(stat, gene.weights = w, weights.label = "logFC", col.bars = "dodgerblue")
```

beadCountWeights *Bead Count Weights for Illumina BeadChips*

Description

Estimates weights which account for biological variation and technical variation resulting from varying bead numbers.

Usage

```
beadCountWeights(y, x, design = NULL, bead.stdev = NULL, bead.stderr = NULL,
                 nbeads = NULL, array.cv = TRUE, scale = FALSE)
```

Arguments

y	an "EList" object or a numeric matrix containing normalized log2-expression values.
x	an "EListRaw" object or a numeric matrix of raw expression values, with same dimensions as y.
design	the design matrix of the microarray experiment, with rows corresponding to arrays and columns to coefficients to be estimated. Defaults to y$design or, if that is NULL, then to a column of ones meaning that the arrays are treated as replicates.
bead.stdev	numeric matrix of bead-level standard deviations.
bead.stderr	numeric matrix of bead-level standard errors. Not required if bead.stdev is set.
nbeads	numeric matrix containing number of beads.
array.cv	logical, should technical variation for each observation be calculated from a constant or array-specific coefficient of variation? The default is to use array-specific coefficients of variation.
scale	logical, should weights be scaled so that the average weight size is the mean of the inverse technical variance along a probe? By default, weights are scaled so that the average weight size along a probe is 1.

Details

This function estimates optimum weights using the bead statistics for each probe for an Illumina expression BeadChip. It can be used with any Illumina expression BeadChip, but is most likely to be useful with HumanHT-12 BeadChips.

Arguments x and y are both required. x contains the raw expression values and y contains the corresponding log2 values for the same probes and the same arrays after background correction and normalization. x and y can be any type of object that can be coerced to a matrix, with rows corresponding to probes and columns to arrays. x and y must contain the same rows and columns in the same order.

The reliability of the normalized expression value for each probe on each array is measured by estimating its technical and biological variability. The bead number weights are the inverse sum of the technical and biological variances.

The technical variance for each probe on each array is inversely proportional to the number of beads and is estimated using array-specific bead-level coefficients of variation.

Coefficients of variation are calculated using raw expression values.

The biological variance for each probe across the arrays is estimated using a Newton iteration, with the assumption that the total residual deviance for each probe from `lmFit` is inversely proportional to the sum of the technical variance and biological variance.

Only one of `bead.stdev` or `bead.stderr` needs to be set. If `bead.stdev` is not provided, then it will be computed as `bead.stderr * sqrt(nbeads)`.

If arguments `bead.stdev` and `nbeads` are not set explicitly in the call, then they will be extracted from `y$other$BEAD_STDEV` and `y$other$Avg_NBEADS`. An `EList` object containing these components can be created by `read.idat` or `read.ilmn`, see the example code below.

Value

A list object with the following components:

`weights`	numeric matrix of bead number weights
`cv.constant`	numeric value of constant bead-level coefficient of variation
`cv.array`	numeric vector of array-specific bead-level coefficients of variation
`var.technical`	numeric matrix of technical variances
`var.biological`	numeric vector of biological variances

Author(s)

Charity Law and Gordon Smyth

References

Law CW (2013). *Precision weights for gene expression analysis*. PhD Thesis. University of Melbourne, Australia. http://hdl.handle.net/11343/38150

See Also

`read.ilmn`, `read.idat`, `neqc`.

An overview of linear model functions in limma is given by 06.LinearModels.

Examples

```
## Not run:
z <- read.ilmn(files="probesummaryprofile.txt",
     ctrfiles="controlprobesummary.txt",
     other.columns=c("BEAD_STDEV","Avg_NBEADS"))
y <- neqc(z)
x <- z[z$genes$Status=="regular",]
```

```
bcw <- beadCountWeights(y,x,design)
fit <- lmFit(y,design,weights=bcw$weights)
fit <- eBayes(fit)

## End(Not run)
```

---

blockDiag *Block Diagonal Matrix*

Description

Form a block diagonal matrix from the given blocks.

Usage

```
blockDiag(...)
```

Arguments

...	numeric matrices

Details

This function is sometimes useful for constructing a design matrix for a disconnected two-color microarray experiment in conjunction with `modelMatrix`.

Value

A block diagonal matrix with dimensions equal to the sum of the input dimensions

Author(s)

Gordon Smyth

See Also

`modelMatrix`

Examples

```
a <- matrix(1,3,2)
b <- matrix(2,2,2)
blockDiag(a,b)
```

bwss *Between and within sums of squares*

Description

Sums of squares between and within groups. Allows for missing values.

Usage

```
bwss(x,group)
```

Arguments

x	a numeric vector giving the responses.
group	a vector or factor giving the grouping variable.

Details

This is equivalent to one-way analysis of variance.

Value

A list with components

bss	sums of squares between the group means.
wss	sums of squares within the groups.
bdf	degrees of freedom corresponding to bss.
wdf	degrees of freedom corresponding to wss.

Author(s)

Gordon Smyth

See Also

bwss.matrix

bwss.matrix *Between and within sums of squares for matrix*

Description

Sums of squares between and within the columns of a matrix. Allows for missing values. This function is called by the anova method for MAList objects.

Usage

```
bwss.matrix(x)
```

Arguments

x	a numeric matrix.

Details

This is equivalent to a one-way analysis of variance where the columns of the matrix are the groups. If x is a matrix then bwss.matrix(x) is the same as bwss(x,col(x)) except for speed of execution.

Value

A list with components

bss	sums of squares between the column means.
wss	sums of squares within the column means.
bdf	degrees of freedom corresponding to bss.
wdf	degrees of freedom corresponding to wss.

Author(s)

Gordon Smyth

See Also

bwss, anova.MAList

camera *Competitive Gene Set Test Accounting for Inter-gene Correlation*

Description

Test whether a set of genes is highly ranked relative to other genes in terms of differential expression, accounting for inter-gene correlation.

Usage

```
## Default S3 method:
camera(y, index, design, contrast = ncol(design), weights = NULL,
     use.ranks = FALSE, allow.neg.cor=FALSE, inter.gene.cor=0.01, trend.var = FALSE,
       sort = TRUE, directional = TRUE, ...)
## Default S3 method:
cameraPR(statistic, index, use.ranks = FALSE, inter.gene.cor=0.01,
       sort = TRUE, directional = TRUE, ...)
interGeneCorrelation(y, design)
```

Arguments

y	a numeric matrix of log-expression values or log-ratios of expression values, or any data object containing such a matrix. Rows correspond to probes and columns to samples. Any type of object that can be processed by getEAWP is acceptable. NA or infinite values are not allowed.
statistic	a numeric vector of genewise statistics. If index contains gene IDs, then statistic should be a named vector with the gene IDs as names.
index	a list of gene sets. Each gene set is either an index vector i or a data.frame where the first column is the index vector and the second column gives the positive or negative change magnitude associated with each gene in the gene set. The index vector can contain gene IDs or row numbers, provided that y[i,] of statistic[i] will subset to rows corresponding to genes in the gene set.
design	design matrix.
contrast	contrast of the linear model coefficients for which the test is required. Can be an integer specifying a column of design, or else a numeric vector of same length as the number of columns of design.
weights	numeric matrix of precision weights. Can be a matrix of the same size as y, or a numeric vector of array weights with length equal to ncol(y), or a numeric vector of gene weights with length equal to nrow(y).
use.ranks	do a rank-based test (TRUE) or a parametric test (FALSE)?
allow.neg.cor	should reduced variance inflation factors be allowed for negative correlations?
inter.gene.cor	numeric, optional preset value for the inter-gene correlation within tested sets. If NA or NULL, then an inter-gene correlation will be estimated for each tested set.

trend.var	logical, should an empirical Bayes trend be estimated? See eBayes for details.
sort	logical, should the results be sorted by p-value?
directional	are the gene sets directional? If TRUE, will test for genes changing in the same direction within each set. If FALSE, will test for large effects without regard to direction in each set.
...	other arguments are not currently used

Details

camera and interGeneCorrelation implement methods proposed by Wu and Smyth (2012). camera performs a *competitive* test in the sense defined by Goeman and Buhlmann (2007). It tests whether the genes in the set are highly ranked in terms of differential expression relative to genes not in the set. It has similar aims to geneSetTest but accounts for inter-gene correlation. See roast for an analogous *self-contained* gene set test.

The function can be used for any microarray experiment which can be represented by a linear model. The design matrix for the experiment is specified as for the lmFit function, and the contrast of interest is specified as for the contrasts.fit function. This allows users to focus on differential expression for any coefficient or contrast in a linear model by giving the vector of test statistic values.

The argument index specifies one or more gene sets. The gene sets can be simple lists of genes, as indicated by a vector of gene IDs. For such gene sets, all the genes in the set are expected to change in the same direction and by the same amount when the molecular pathway is active. Alternatively, the gene sets can be provided as expression signatures, whereby an expected directional change is provided for each gene in the gene set. For example, the directional changes could be the log-fold-changes observed in previous experiments when a particular biological pathway was perturbed. Expression signature gene sets are provided as data.frames, with the gene indices in the first column and the directional changes in the second column.

One convenient way to construct the index argument is by using ids2indices.

camera estimates p-values after adjusting the variance of test statistics by an estimated variance inflation factor. The inflation factor depends on estimated genewise correlation and the number of genes in the gene set.

By default, camera uses interGeneCorrelation to estimate the mean pair-wise correlation within each set of genes. camera can alternatively be used with a preset correlation specified by inter.gene.cor that is shared by all sets. This usually works best with a small value, say inter.gene.cor=0.01.

If inter.gene.cor=NA, then camera will estimate the inter-gene correlation for each set. In this mode, camera gives rigorous error rate control for all sample sizes and all gene sets. However, in this mode, highly co-regulated gene sets that are biologically interpretable may not always be ranked at the top of the list.

With the default value inter.gene.cor=0.01, camera will rank biologically interpretable sets more highly. This gives a useful compromise between strict error rate control and interpretable gene set rankings.

cameraPR is a "pre-ranked" version of camera where the genes are pre-ranked according to a pre-computed statistic.

If directional=TRUE, then the gene sets are assumed to contain genes changing in the same direction and the functions will do two-sided directional tests for each gene set. If directional=FALSE,

then the gene sets are assumed to be non-directional, so that significant genes in each set could be changing in different directions. In this case, the functions will test for large changes in each set without regard to direction of change. If `directional=FALSE`, then `use.ranks` should be `TRUE` and `statistic` should be non-negative.

Value

`camera` and `cameraPR` return a data.frame with a row for each set and the following columns:

`NGenes`	number of genes in set.
`Correlation`	inter-gene correlation (only included if the `inter.gene.cor` was not preset).
`Direction`	direction of change (`"Up"` or `"Down"`).
`PValue`	two-tailed p-value.
`FDR`	Benjamini and Hochberg FDR adjusted p-value.

`interGeneCorrelation` returns a list with components:

`vif`	variance inflation factor.
`correlation`	inter-gene correlation.

Note

The default settings for `inter.gene.cor` and `allow.neg.cor` were changed to the current values in limma 3.29.6. Previously, the default was to estimate an inter-gene correlation for each set. To reproduce the earlier default, use `allow.neg.cor=TRUE` and `inter.gene.cor=NA`.

Author(s)

Di Wu and Gordon Smyth

References

Wu D, Smyth GK (2012). Camera: a competitive gene set test accounting for inter-gene correlation. *Nucleic Acids Research* 40, e133. doi:10.1093/nar/gks461.

Goeman JJ, Buhlmann P (2007). Analyzing gene expression data in terms of gene sets: methodological issues. *Bioinformatics* 23, 980-987.

See Also

`getEAWP`

`rankSumTestWithCorrelation`, `geneSetTest`, `roast`, `fry`, `romer`, `ids2indices`.

There is a topic page on 10.GeneSetTests.

Examples

```
y <- matrix(rnorm(1000*6),1000,6)
rownames(y) <- paste0("G",1:1000)
colnames(y) <- paste0("S",1:6)
design <- cbind(Intercept=1,Group=c(0,0,0,1,1,1))

# First set of 20 genes are genuinely up-regulated
index1 <- 1:20
y[index1,4:6] <- y[index1,4:6] + 1

# Second set of 20 genes are genuinely down-regulated
index2 <- 21:40
y[index2,4:6] <- y[index1,4:6] - 1

# Other genes are not DE
index3 <- 41:60

# Directional gene set containing both up and down genes
index4 <- data.frame(
  ID = c("G1","G2","G3","G21","G22","G23"),
  logFC = c(1,1,1,-1,-1,-1)
)

index <- list(set1=index1, set2=index2, set3=index3, set4=index4)
camera(y, index, design, inter.gene.cor=NA, sort=FALSE)
camera(y, index, design, inter.gene.cor=0.01, sort=FALSE)

# Pre-ranked version
fit <- eBayes(lmFit(y, design))
logFC <- fit$coefficient[,2]
cameraPR(logFC, index, sort=FALSE)
```

---

cbind *Combine RGList, MAList, EList or EListRaw Objects*

Description

Combine a set of RGList, MAList, EList or EListRaw objects.

Usage

```
## S3 method for class 'RGList'
cbind(..., deparse.level=1)
## S3 method for class 'RGList'
rbind(..., deparse.level=1)
```

Arguments

...	RGList, MAList, EList or EListRaw objects.
deparse.level	not currently used, see cbind in the base package

Details

cbind combines data objects assuming the same probes in the same order but different arrays. rbind combines data objects assuming equivalent arrays, i.e., the same RNA targets, but different probes.

For cbind, the matrices of expression data from the individual objects are cbinded. The data.frames of target information, if they exist, are rbinded. The combined data object will preserve any additional components or attributes found in the first object to be combined. For rbind, the matrices of expression data are rbinded while the target information, if any, is unchanged.

Value

An RGList, MAList, EList or EListRaw object holding data from all the arrays and all genes from the individual objects.

Author(s)

Gordon Smyth

See Also

cbind in the base package.

03.ReadingData gives an overview of data input and manipulation functions in LIMMA.

Examples

```
M <- A <- matrix(11:14,4,2)
rownames(M) <- rownames(A) <- c("a","b","c","d")
colnames(M) <- colnames(A) <- c("A1","A2")
MA1 <- new("MAList",list(M=M,A=A))

M <- A <- matrix(21:24,4,2)
rownames(M) <- rownames(A) <- c("a","b","c","d")
colnames(M) <- colnames(A) <- c("B1","B2")
MA2 <- new("MAList",list(M=M,A=A))

cbind(MA1,MA2)
```

---

changeLog *Change Log*

Description

Show the most recent changes from a package change log or NEWS file.

Usage

```
changeLog(n = 30, package = "limma")
```

Arguments

n	integer, number of lines to write of change log.
package	character string giving name of package.

Details

The function will look for a file changelog.txt or ChangeLog in the top-level or doc directories of the installed package. Failing that, it will look for NEWS or NEWS.md in the top-level directory.

Note that changeLog does not write the content of NEWS.Rd, which is a structured file. Use news(package="limma") for that instead.

Value

No value is produced, but a number of lines of text are written to standard output.

Author(s)

Gordon Smyth

See Also

01.Introduction, news.

Examples

```
changeLog()
changeLog(package="statmod")
```

---

chooseLowessSpan *Choose Span for Local-Weighted Regression Smoothing*

Description

Choose an optimal span, depending on the number of points, for lowess smoothing of variance trends.

Usage

```
chooseLowessSpan(n=1000, small.n=50, min.span=0.3, power=1/3)
```

Arguments

n	the number of points the lowess curve will be applied to.
small.n	the span will be set to 1 for any n less than or equal to this value.
min.span	the minimum span for large n.
power	numeric power between 0 and 1 that determines how fast the chosen span decreases with n.

Details

The span is the proportion of points used for each of the local regressions. When there are a few points, a large span should be used to ensure a smooth curve. When there are a large number of points, smaller spans can be used because each span window still contains good coverage. By default, the chosen span decreases as the cube-root of the number of points, a rule that is motivated by analogous rules to choose the number of bins for a histogram (Scott, 1979; Freedman & Diaconis, 1981; Hyndman, 1995).

The span returned is `min.span + (1-min.span) * (small.n/n)^power` except that the span is set to 1 for any `n` less than `small.n`. Note that the fitted lowess curve will still estimate a trend (i.e., will not be constant) even if `span=1`.

The function is tuned for smoothing of mean-variance trends, for which the trend is usually monotonic, so preference is given to moderately large spans. Even for very large datasets, the span is always greater than `min.span`.

This function is used to create adaptive spans for `voom`, `vooma` and `voomaLmFit` where `n` is the number of genes in the analysis.

Value

A numeric vector of length 1 containing the span value.

Author(s)

Gordon Smyth

References

Freedman D, Diaconis P (1981). On the histogram as a density estimator: L_2 theory. *Zeitschrift fur Wahrscheinlichkeitstheorie und verwandte Gebiete* 57, 453-476.

Hyndman RJ (1995). The problem with Sturges' rule for constructing histograms. https://robjhyndman.com/papers/sturges.pdf.

Scott DW (1979). On optimal and data-based histograms. *Biometrika* 66, 605-610.

See Also

`loessFit`, `weightedLowess`, `lowess`, `loess`.

`vooma`, `eBayes`, `squeezeVar`, `fitFDistRobustly`.

Examples

```
chooseLowessSpan(100)
chooseLowessSpan(1e6)

n <- 10:5000
span <- chooseLowessSpan(n)
plot(n,span,type="l",log="x")
```

`classifyTestsF` *Genewise Nested F-Tests*

Description

For each gene, classify a series of related t-statistics as significantly up or down using nested F-tests.

Usage

```
classifyTestsF(object, cor.matrix = NULL, df = Inf, p.value = 0.01, fstat.only = FALSE)
```

Arguments

`object`	numeric matrix of t-statistics or an `MArrayLM` object from which the t-statistics may be extracted.
`cor.matrix`	covariance matrix of each row of t-statistics. Will be extracted automatically from an `MArrayLM` object but otherwise defaults to the identity matrix.
`df`	numeric vector giving the degrees of freedom for the t-statistics. May have length 1 or length equal to the number of rows of `tstat`. Will be extracted automatically from an `MArrayLM` object but otherwise default to `Inf`.
`p.value`	numeric value between 0 and 1 giving the desired size of the test.
`fstat.only`	logical, if `TRUE` then return the overall F-statistic as for `FStat` instead of classifying the test results.

Details

`classifyTestsF` implements the `"nestedF"` multiple testing option offered by `decideTests`. Users should generally use `decideTests` rather than calling `classifyTestsF` directly because, by itself, `classifyTestsF` does not incorporate any multiple testing adjustment across genes. Instead it simply tests across contrasts for each gene individually.

`classifyTestsF` uses a nested F-test approach giving particular attention to correctly classifying genes that have two or more significant t-statistics, i.e., which are differentially expressed in two or more conditions. For each row of `tstat`, the overall F-statistic is constructed from the t-statistics as for `FStat`. At least one contrast will be classified as significant if and only if the overall F-statistic is significant. If the overall F-statistic is significant, then the function makes a best choice as to which t-statistics contributed to this result. The methodology is based on the principle that any t-statistic should be called significant if the F-test is still significant for that row when all the larger t-statistics are set to the same absolute size as the t-statistic in question.

Compared to conventional multiple testing methods, the nested F-test approach achieves better consistency between related contrasts. (For example, if B is judged to be different from C, then at least one of B or C should be different to A.) The approach was first used by Michaud et al (2008). The nested F-test approach provides *weak* control of the family-wise error rate, i.e., it correctly controls the type I error rate of calling any contrast as significant if all the null hypotheses are true. In other words, it provides error rate control at the overall F-test level but does not provide strict error rate control at the individual contrast level.

Usually `object` is a limma linear model fitted object, from which a matrix of t-statistics can be extracted, but it can also be a numeric matrix of t-statistics. In either case, rows correspond to genes and columns to coefficients or contrasts. If `object` is a matrix, then it may be necessary to supply values for `cor.matrix` and `df`. The `cor.matrix` is the same as the correlation matrix of the coefficients from which the t-statistics were calculated and `df` is the degrees of freedom of the t-statistics. All statistics for the same gene must have the same degrees of freedom.

If `fstat.only=TRUE`, then `classifyTestsF` just returns the vector of overall F-statistics for each gene.

Value

If `fstat.only=FALSE`, then an object of class `TestResults` is returned. This is essentially a numeric matrix with elements `-1`, `0` or `1` depending on whether each t-statistic is classified as significantly negative, not significant or significantly positive respectively.

If `fstat.only=TRUE`, then a numeric vector of F-statistics is returned with attributes `df1` and `df2` giving the corresponding degrees of freedom.

Author(s)

Gordon Smyth

References

Michaud J, Simpson KM, Escher R, Buchet-Poyau K, Beissbarth T, Carmichael C, Ritchie ME, Schutz F, Cannon P, Liu M, Shen X, Ito Y, Raskind WH, Horwitz MS, Osato M, Turner DR, Speed TP, Kavallaris M, Smyth GK, Scott HS (2008). Integrative analysis of RUNX1 downstream pathways and target genes. *BMC Genomics* 9, 363.

See Also

An overview of multiple testing functions is given in 08.Tests.

Examples

```
TStat <- matrix(c(0,10,0, 0,5,0, -4,-4,4, 2,2,2), 4, 3, byrow=TRUE)
colnames(TStat) <- paste0("Contrast",1:3)
rownames(TStat) <- paste0("Gene",1:4)
classifyTestsF(TStat, df=20)
FStat <- classifyTestsF(TStat, df=20, fstat.only=TRUE)
P <- pf(FStat, df1=attr(FStat,"df1"), df2=attr(FStat,"df2"), lower.tail=FALSE)
data.frame(F.Statistic=FStat,P.Value=P)
```

`contrastAsCoef` *Reform a Design Matrix so that Contrasts Become Coefficients*

Description

Reform a design matrix so that one or more coefficients from the new matrix correspond to specified contrasts of coefficients from the old matrix.

Usage

```
contrastAsCoef(design, contrast=NULL, first=TRUE)
```

Arguments

`design`	numeric design matrix.
`contrast`	numeric matrix with rows corresponding to columns of the design matrix (coefficients) and columns containing contrasts. May be a vector if there is only one contrast.
`first`	logical, should coefficients corresponding to contrasts be the first columns (`TRUE`) or last columns (`FALSE`) of the output design matrix.

Details

If the contrasts contained in the columns of `contrast` are not linearly dependent, then superfluous columns are dropped until the remaining matrix has full column rank. The number of retained contrasts is stored in `qr$rank` and the retained columns are given by `qr$pivot`.

Value

A list with components

`design`	reformed design matrix
`coef`	columns of design matrix which hold the meaningful coefficients
`qr`	QR-decomposition of contrast matrix

Author(s)

Gordon Smyth

See Also

`model.matrix` in the stats package.

An overview of linear model functions in limma is given by 06.LinearModels.

Examples

```
design <- cbind(1,c(0,0,1,1,0,0),c(0,0,0,0,1,1))
cont <- c(0,-1,1)
design2 <- contrastAsCoef(design, cont)$design

#  Original coef[3]-coef[2] becomes coef[1]
y <- rnorm(6)
fit1 <- lm(y~0+design)
fit2 <- lm(y~0+design2)
coef(fit1)
coef(fit1)
coef(fit2)
```

`contrasts.fit` *Compute Contrasts from Linear Model Fit*

Description

Given a linear model fit to microarray data, compute estimated coefficients and standard errors for a given set of contrasts.

Usage

```
contrasts.fit(fit, contrasts=NULL, coefficients=NULL)
```

Arguments

`fit`	an MArrayLM object or a list object produced by the function `lm.series` or equivalent. Must contain components `coefficients` and `stdev.unscaled`.
`contrasts`	numeric matrix with rows corresponding to coefficients in `fit` and columns containing contrasts. May be a vector if there is only one contrast. `NA`s are not allowed.
`coefficients`	vector indicating which coefficients are to be kept in the revised fit object. An alternative way to specify the `contrasts`.

Details

This function accepts input from any of the functions `lmFit`, `lm.series`, `mrlm`, `gls.series` or `lmscFit`. The function re-orientates the fitted model object from the coefficients of the original design matrix to any set of contrasts of the original coefficients. The coefficients, unscaled standard deviations and correlation matrix are re-calculated in terms of the contrasts.

The idea of this function is to fit a full-rank model using `lmFit` or equivalent, then use `contrasts.fit` to obtain coefficients and standard errors for any number of contrasts of the coefficients of the original model. Unlike the design matrix input to `lmFit`, which normally has one column for each treatment in the experiment, the matrix `contrasts` may have any number of columns and these are not required to be linearly independent. Methods of assessing differential expression, such as `eBayes` or `classifyTestsF`, can then be applied to the fitted model object.

The coefficients argument provides a simpler way to specify the contrasts matrix when the desired contrasts are just a subset of the original coefficients.

Value

A list object of the same class as fit, usually MArrayLM. This is a list with components

coefficients numeric matrix containing the estimated coefficients for each contrast for each probe.

stdev.unscaled numeric matrix conformal with coef containing the unscaled standard deviations for the coefficient estimators.

cov.coefficients numeric matrix giving the unscaled covariance matrix of the estimable coefficients.

Most other components found in fit are passed through unchanged, but t, p.value, lods, F and F.p.value will all be removed.

Note

For efficiency reasons, this function does not re-factorize the design matrix for each probe. A consequence is that, if the design matrix is non-orthogonal and the original fit included precision weights or missing values, then the unscaled standard deviations produced by this function are approximate rather than exact. The approximation is usually acceptable. If not, then the issue can be avoided by redefining the design matrix to fit the contrasts directly.

Even with precision weights or missing values, the results from contrasts.fit are always exact if the coefficients being compared are statistically independent. This will be true, for example, if the original fit was a oneway model without blocking and the group-means (no-intercept) parametrization was used for the design matrix.

Author(s)

Gordon Smyth

See Also

An overview of linear model functions in limma is given by 06.LinearModels.

Examples

```
#  Simulate gene expression data: 6 microarrays and 100 genes
#  with one gene differentially expressed in first 3 arrays
M <- matrix(rnorm(100*6,sd=0.3),100,6)
M[1,1:3] <- M[1,1:3] + 2
#  Design matrix corresponds to oneway layout, columns are orthogonal
design <- cbind(First3Arrays=c(1,1,1,0,0,0),Last3Arrays=c(0,0,0,1,1,1))
fit <- lmFit(M,design=design)
# Would like to consider original two estimates plus difference between first 3 and last 3 arrays
contrast.matrix <- cbind(First3=c(1,0),Last3=c(0,1),"Last3-First3"=c(-1,1))
fit2 <- contrasts.fit(fit,contrast.matrix)
```

```
fit2 <- eBayes(fit2)
#  Large values of eb$t indicate differential expression
results <- decideTests(fit2, method="nestedF")
vennCounts(results)
```

---

controlStatus *Set Status of each Spot from List of Spot Types*

Description

Determine the type (or status) of each spot in the gene list.

Usage

```
controlStatus(types, genes, spottypecol="SpotType", regexpcol, verbose=TRUE)
```

Arguments

types	dataframe containing spot type specifiers, usually input using readSpotTypes.
genes	dataframe containing gene annotation, or an object of class RGList, MAList, EListRaw, EList or MArrayLM from which the gene annotation can be extracted.
spottypecol	integer or name specifying column of types containing spot type names.
regexpcol	vector of integers or column names specifying columns of types containing regular expressions. Defaults to any column names in common between types and genes.
verbose	logical, if TRUE then progress on pattern matching is reported to the standard output channel.

Details

This function constructs a vector of status codes by searching for patterns in the gene list. The data frame genes contains gene IDs and should have as many rows as there are spots on the microarrays. Such a data frame is often read using readGAL. The data frame types has as many rows as you want to distinguish types of spots in the gene list. This data frame should contain a column or columns, the regexpcol columns, which have the same names as columns in genes and which contain patterns to match in the gene list. Another column, the spottypecol, contains the names of the spot types. Any other columns are assumed to contain plotting parameters, such as colors or symbols, to be associated with the spot types.

The patterns in the regexpcol columns are simplified regular expressions. For example, AA* means any string starting with AA, *AA means any string ending with AA, AA means exactly these two letters, *AA* means any string containing AA, AA. means AA followed by exactly one other character and AA\. means exactly AA followed by a period and no other characters. Any other regular expressions are allowed but the codes ^ for beginning of string and $ for end of string should not be included.

Note that the patterns are matched sequentially from first to last, so more general patterns should be included first. For example, it is often a good idea to include a default spot-type as the first line in types with pattern * for all regexpcol columns and default plotting parameters.

Value

Character vector specifying the type (or status) of each spot on the array. Attributes contain plotting parameters associated with each spot type.

Author(s)

Gordon Smyth

See Also

An overview of LIMMA functions for reading data is given in 03.ReadingData.

Examples

```
genes <- data.frame(
      ID=c("Control","Control","Control","Control","AA1","AA2","AA3","AA4"),
      Name=c("Ratio 1","Ratio 2","House keeping 1","House keeping 2",
             "Gene 1","Gene 2","Gene 3","Gene 4"))
types <- data.frame(
      SpotType=c("Gene","Ratio","Housekeeping"),
      ID=c("*","Control","Control"),
      Name=c("*","Ratio*","House keeping*"),
      col=c("black","red","blue"))
status <- controlStatus(types,genes)
```

---

coolmap *Heatmap of gene expression values*

Description

Create a heatmap of a matrix of log-expression values.

Usage

```
coolmap(x, cluster.by="de pattern", col=NULL,
     linkage.row="complete", linkage.col="complete", show.dendrogram="both", ...)
```

Arguments

x	any data object that can be coerced to a matrix of log-expression values, for example an ExpressionSet or EList. Rows represent genes and columns represent RNA samples.
cluster.by	choices are "de pattern" or "expression level". In the former case, the intention is to cluster by relative changes in expression, so genes are clustered by Pearson correlation and log-expression values are mean-corrected by rows for the plot. In the latter case, the intention is to cluster by absolute expression, so genes are clustered by Euclidean distance and log-expression values are not mean-corrected.

col — character vector specifying the color panel. Can be either the name of the panel or a vector of R colors that can be passed directly to the heatmap.2 function. Possible panel names are "redblue", "redgreen", "yellowblue" or "whitered". Defaults to "redblue" if cluster.by="de pattern" or "yellowblue" if cluster.by="expression level".

linkage.row — linkage criterion used to cluster the rows. Choices are "none", "ward", "ward.D", "ward.D2", "single", "complete", "average", "mcquitty", "median" or "centroid", with "ward" treated as "ward.D2".

linkage.col — linkage criterion used to cluster the columns. Choices are the same as for linkage.row.

show.dendrogram — choices are "row", "column", "both" or "none".

... — any other arguments are passed to heatmap.2. See details for which arguments are reserved.

Details

This function calls the heatmap.2 function in the gplots package with sensible argument settings for genomic log-expression data. The default settings for heatmap.2 are often not ideal for expression data, and overriding the defaults requires explicit calls to hclust and as.dendrogram as well as prior standardization of the data values. The coolmap function implements our preferred defaults for the two most common types of heatmaps. When clustering by relative expression (cluster.by="de pattern"), it implements a row standardization that takes account of NA values and standard deviations that might be zero.

coolmap sets the following heatmap.2 arguments internally: Rowv, Colv, scale, density.info, trace, col, symbreaks, symkey, dendrogram, key.title and key.xlab. These arguments are therefore reserved and cannot be varied. Other than these reserved arguments, any other heatmap.2 argument can be included in the coolmap call, thereby giving full access to heatmap.2 functionality.

Value

A plot is created on the current graphics device. A list is also invisibly returned, see heatmap.2 for details.

Author(s)

Gordon Smyth

See Also

heatmap.2, hclust, dist.

An overview of diagnostic functions available in LIMMA is given in 09.Diagnostics.

Examples

```
# Simulate gene expression data for 50 genes and 6 microarrays.
# Samples are in two groups
# First 50 probes are differentially expressed in second group
```

```
ngenes <- 50
sd <- 0.3*sqrt(4/rchisq(ngenes,df=4))
x <- matrix(rnorm(ngenes*6,sd=sd),ngenes,6)
rownames(x) <- paste("Gene",1:ngenes)
x <- x + seq(from=0, to=16, length=ngenes)
x[,4:6] <- x[,4:6] + 2
coolmap(x)
```

`cumOverlap` *Cumulative Overlap Analysis of Ordered Lists*

Description

Test whether the leading members of ordered lists significantly overlap.

Usage

```
cumOverlap(ol1, ol2)
```

Arguments

`ol1`	vector containing first ordered list. Duplicate values not allowed.
`ol2`	vector containing second ordered list. Should contain the same values as found in `ol1` but in a possibly different order. Duplicate values not allowed.

Details

The function compares the top `n` members of each list, for every possible `n`, and conducts a hypergeometric test for overlap. The function returns the value of `n` giving the smallest p-value.

The p-values are adjusted for multiple testing in a similar way to Bonferroni's method, but starting from the top of the ranked list instead of from the smallest p-values. This approach is designed to be sensitive to contexts where the number of IDs involved in the significant overlap is a small proportion of the total.

The vectors `ol1` and `ol2` do not need to be of the same length, but only values in common between the two vectors will be used in the calculation.

This method was described in Chapter 4 of Wu (2011).

Value

List containing the following components:

`n.total`	integer, total number of values in common between `ol1` and `ol2`.
`n.min`	integer, top table length leading to smallest adjusted p-value.
`p.min`	smallest adjusted p-value.
`n.overlap`	integer, number of overlapping IDs in first `n.min`.
`id.overlap`	vector giving the overlapping IDs in first `n.min`.

p.value	numeric, vector of p-values for each possible top table length.
adj.p.value	numeric, vector of Bonferroni adjusted p-values for each possible top table length.

Author(s)

Gordon Smyth and Di Wu

References

Wu D (2011). Finding hidden relationships between gene expression profiles with application to breast cancer biology. PhD thesis, University of Melbourne. http://hdl.handle.net/11343/36278

Examples

```
GeneIds <- paste0("Gene",1:50)
ol1 <- GeneIds
ol2 <- c(sample(GeneIds[1:5]), sample(GeneIds[6:50]))
coa <- cumOverlap(ol1, ol2)
coa$p.min
coa$id.overlap
```

---

decideTests *Multiple Testing Across Genes and Contrasts*

Description

Identify which genes are significantly differentially expressed for each contrast from a fit object containing p-values and test statistics. A number of different multiple testing strategies are offered that adjust for multiple testing down the genes as well as across contrasts for each gene.

Usage

```
## S3 method for class 'MArrayLM'
decideTests(object, method = "separate", adjust.method = "BH", p.value = 0.05,
            lfc = 0, ...)
## Default S3 method:
decideTests(object, method = "separate", adjust.method = "BH", p.value = 0.05,
          lfc = 0, coefficients = NULL, cor.matrix = NULL, tstat = NULL, df = Inf,
            genewise.p.value = NULL, ...)
```

Arguments

object	a numeric matrix of p-values or an MArrayLM object from which p-values and t-statistics can be extracted.

method	character string specifying how genes and contrasts are to be combined in the multiple testing scheme. Choices are "separate", "global", "hierarchical" or "nestedF".
adjust.method	character string specifying p-value adjustment method. Possible values are "none", "BH", "fdr" (equivalent to "BH"), "BY" and "holm". See p.adjust for details.
p.value	numeric value between 0 and 1 giving the required family-wise error rate or false discovery rate.
lfc	numeric, minimum absolute log2-fold-change required.
coefficients	numeric matrix of coefficients or log2-fold-changes. Of same dimensions as object.
cor.matrix	correlation matrix of coefficients. Square matrix of dimension ncol(object).
tstat	numeric matrix of t-statistics. Of same dimensions as object.
df	numeric vector of length nrow(object) giving degrees of freedom for the t-statistics.
genewise.p.value	numeric vector of length nrow(object) containing summary gene-level p-values for use with method="hierarchical".
...	other arguments are not used.

Details

This function can be applied to a matrix of p-values but is more often applied to an MArrayLM fit object produced by eBayes or treat. In either case, rows of object correspond to genes and columns to coefficients or contrasts.

This function applies a multiple testing procedure and a significance level cutoff to the statistics contained in object. It implements a number of multiple testing procedures for determining whether each statistic should be considered significantly different from zero.

method="separate" will apply multiple testing adjustments to each column of p-values separately. Setting method="separate" is equivalent to using topTable separately for each coefficient in the linear model fit and will identify the same probes as significantly differentially expressed if adjust.method is the same. method="global" will treat the entire matrix of t-statistics as a single vector of unrelated tests. method="hierarchical" adjusts down genes and then across contrasts. method="nestedF" adjusts down genes according to overall F-tests and then uses classifyTestsF to classify contrasts as significant or not for the selected genes.

The default method="separate" and adjust.method="BH" settings are appropriate for most analyses. method="global" is useful when it is important that the same t-statistic cutoff should correspond to statistical significance for all the contrasts. The "nestedF" method was proposed by Michaud et al (2008) and achieves better consistency between contrasts than the other methods. It provides formal error rate control at the gene level but not for individual contrasts. See the classifyTestsF help page for more detail about the "nestedF" method.

If object is an MArrayLM linear model fit, then the "hierarchical" method conducts row-wise F-tests and then proceeds to t-tests for those rows with significant F-tests. The multiple testing adjustment is applied initially to the F-tests and then, with an adjusted level, to the t-tests for each significant row.

Also see the limma User's Guide for a discussion of the statistical properties of the various adjustment methods.

Value

An object of class TestResults. This is essentially a numeric matrix with elements -1, 0 or 1 depending on whether each t-statistic is classified as significantly negative, not significant or significantly positive.

If lfc>0 then contrasts are judged significant only when the log2-fold change is at least this large in absolute value. For example, one might choose lfc=log2(1.5) to restrict to 50% changes or lfc=1 for 2-fold changes. In this case, contrasts must satisfy both the p-value and the fold-change cutoff to be judged significant.

Note

Although this function enables users to set p-value and lfc cutoffs simultaneously, this combination criterion is not recommended. logFC cutoffs tend to favor low expressed genes and thereby reduce rather than increase biological significance. Unless the fold changes and p-values are very highly correlated, the addition of a fold change cutoff can increase the family-wise error rate or false discovery rate above the nominal level. Users wanting to use fold change thresholding are recommended to use treat instead of eBayes and to leave lfc at the default value when using decideTests.

Author(s)

Gordon Smyth

References

Michaud J, Simpson KM, Escher R, Buchet-Poyau K, Beissbarth T, Carmichael C, Ritchie ME, Schutz F, Cannon P, Liu M, Shen X, Ito Y, Raskind WH, Horwitz MS, Osato M, Turner DR, Speed TP, Kavallaris M, Smyth GK, Scott HS (2008). Integrative analysis of RUNX1 downstream pathways and target genes. *BMC Genomics* 9, 363. doi:10.1186/147121649363.

See Also

An overview of multiple testing functions is given in 08.Tests.

---

designI2M *Convert Individual Channel Design Matrix to M-A Format*

Description

Convert a design matrix in terms of individual channels to ones in terms of M-values or A-values for two-color microarray data.

Usage

```
designI2M(design)
designI2A(design)
```

Arguments

design	numeric model matrix with one row for each channel observation, i.e., twice as many rows as arrays

Details

If design is a model matrix suitable for modeling individual log-intensities for two color microarray data, then designI2M computes the corresponding model matrix for modeling M-values (log-ratios) and designI2A computes the model matrix for modeling A-values (average log-intensities).

Note that the matrices designI2M(design) or designI2A(design) may be singular if not all of the coefficients are estimable from the M or A-values. In that case there will be columns containing entirely zeros.

Value

numeric model matrix with half as many rows as design

Author(s)

Gordon Smyth

See Also

model.matrix in the stats package.

An overview of individual channel linear model functions in limma is given by 07.SingleChannel.

Examples

```
X <- cbind(1,c(1,1,1,1,0,0,0,0),c(0,0,0,0,1,1,1,1))
designI2M(X)
designI2A(X)
```

---

detectionPValues *Detection P-Values from Negative Controls*

Description

Compute the proportion of negative controls greater than each observed expression value. Particularly useful for Illumina BeadChips.

Usage

```
## S3 method for class 'EListRaw'
detectionPValues(x, status = NULL, ...)
## Default S3 method:
detectionPValues(x, status, negctrl = "negative", ...)
```

Arguments

x	object of class EListRaw or a numeric matrix containing raw intensities for regular and control probes from a series of microarrays.
status	character vector giving probe types. Defaults to x$genes$Status if x is an EListRaw object.
negctrl	character string identifier for negative control probes.
...	other arguments are not currently used.

Details

The rows of x for which status == negctrl are assumed to correspond to negative control probes.

For each column of x, the detection p-values are defined as (N.eq/2 + N.gt) / N.neg, where N.gt is the number of negative controls with expression greater than the observed value, N.eq is the number of negative controls with expression equal to the observed value, and N.neg is the total number of negative controls.

When used on Illumina BeadChip data, this function produces essentially the same detection p-values as returned by Illumina's GenomeStudio software.

Value

numeric matrix of same dimensions as x containing detection p-values.

Author(s)

Gordon Smyth

References

Shi W, de Graaf C, Kinkel S, Achtman A, Baldwin T, Schofield L, Scott H, Hilton D, Smyth GK (2010). Estimating the proportion of microarray probes expressed in an RNA sample. *Nucleic Acids Research* 38(7), 2168-2176. doi:10.1093/nar/gkp1204.

See Also

An overview of LIMMA functions to read expression data is given in 03.ReadingData.

read.idat reads Illumina BeadChip expression data from binary IDAT files.

neqc performs normexp background correction and quantile normalization aided by control probes.

Examples

```
## Not run:
# Read Illumina binary IDAT files
x <- read.idat(idat, bgx)
x$other$Detection <- detectionPValues(x)
y <- neqc(x)

## End(Not run)
```

diffSplice *Differential Transcript or Exon Usage*

Description

Given a limma fit object with multiple features per gene, test for differential usage of those features within genes between experimental conditions. The features may be any set of isoform-identifying features such as transcripts, exons or exon-junctions.

Usage

```
## S3 method for class 'MArrayLM'
diffSplice(fit, geneid, exonid = NULL,
     robust = FALSE, legacy = FALSE, verbose = TRUE, ...)
```

Arguments

fit	an MArrayLM fitted model object produced by voomLmFit, lmFit or contrasts.fit. Rows should correspond to transcripts, for a DTU analysis, or to exons and exon-exon junctions for a DEU analysis.
geneid	gene identifiers. Either a vector of length nrow(fit) or the name of the column of fit$genes containing the gene identifiers. Rows with the same ID are assumed to belong to the same gene.
exonid	feature or exon identifiers. Either a vector of length nrow(fit) or the name of the column of fit$genes containing the feature identifiers.
robust	logical, should the estimation of the empirical Bayes prior parameters be robustified against outlier sample variances?
legacy	logical. If FALSE then the new empirical Bayes hyperparameter estimation (introduced in limma 3.61.8) will be used, if TRUE the earlier hyperparameter estimation will be used. The new method is particularly appropriate when the residual degrees of freedom are not all equal, which is likely to be the case for diffSplice.
verbose	logical, if TRUE some diagnostic information about the number of genes and exons is output.
...	other arguments are not currently used.

Details

Given a limma fit object with multiple features per gene, test for differential usage of those features within genes between experimental conditions. The features may be any set of isoform-identifying features such as transcripts, exons or exon-junctions. The features can be transcripts for a differential transcript usage (DTU) analysis (Baldoni et al 2025), or can be a combination of exons and exon-exon junctions for a differential exon usage (DEU) analysis. Another possibility is for the features to be transcript-overlap equivalence classes (Cmero et al 2019).

Testing for differential usage is equivalent to testing whether the log-fold-changes in the `fit` differ between features for the same gene. Two different tests are provided. The first is a F-test for differences between the log-fold-changes for each gene. This is equivalent to testing for interaction between the features for that gene and the coefficient of the linear model. The other is a series of t-tests in which each transcript is compared to the weighted average of all other features for the same gene. The feature-level t-tests are converted into genewise tests by adjusting the p-values for the same gene by Simes' method. As an alternative, the feature-level t-tests are also converted into genewise tests by adjusting the smallest p-value for each gene by Bonferroni's method.

This function can be used on transcript level RNA-seq counts from RSEM, kallisto, or Salmon, after pre-processing by the edgeR functions `catchSalmon()` (or `catchKallisto` or `catchRSEM`) and `voomLmFit()`, as described by Baldoni et al (2025). It can also be used on equivalence-class counts from Salmon or kallisto, after pre-processing by voomLmFit(), as described by Cmero et al (2019). It can also be used on exon-level read counts or on data from an exon microarray.

Value

An object of class `MArrayLM` containing both exon level and gene level tests. Results are sorted by geneid and by exonid within gene.

`coefficients`	numeric matrix of coefficients of same dimensions as `fit`. Each coefficient is the difference between the log-fold-change for that exon versus the average log-fold-change for all other exons for the same gene.
`t`	numeric matrix of moderated t-statistics, of same dimensions as `fit`.
`p.value`	numeric vector of p-values corresponding to the t-statistics
`genes`	data.frame of exon annotation
`genecolname`	character string giving the name of the column of `genes` containing gene IDs
`gene.F`	numeric matrix of moderated F-statistics, one row for each gene.
`gene.F.p.value`	numeric matrix of p-values corresponding to `gene.F`
`gene.simes.p.value`	numeric matrix of Simes adjusted p-values, one row for each gene.
`gene.bonferroni.p.value`	numeric matrix of Bonferroni adjusted p-values, one row for each gene.
`gene.genes`	data.frame of gene annotation.

Note

This function is not designed for situations with a very high level of multi-counting of RNA-seq reads that overlap two or more exons. In particular, it is not designed for use with "chopped" exons, where overlapping exons belonging to different transcripts of the same gene are chopped up into unique sub-exons, because artificial exons of this sort lead to high levels of multi-counting.

Author(s)

Gordon Smyth and Charity Law

References

Baldoni PL, Chen L, Li M, Chen Y, Smyth GK (2025). Dividing out quantification uncertainty enables assessment of differential transcript usage with limma and edgeR. *Nucleic Acids Research* 53(22), gkaf1305. doi:10.1093/nar/gkaf1305.

Cmero M, Davidson NM, Oshlack A (2019). Using equivalence class counts for fast and accurate testing of differential transcript usage. *F1000Research* 8, 265. doi:10.12688/f1000research.18276.2.

See Also

`topSplice` and `plotSplice` are downstream functions that operate on the output from `diffSplice`.

Also see `diffSplice.DGEGLM` in the edgeR package, which has comparable functionality but for edgeR fit objects.

A summary of functions available in LIMMA for RNA-seq analysis is given in 11.RNAseq.

Examples

```
## Not run:
fit <- voomLmFit(dge, design)
ex <- diffSplice(fit, geneid="GeneID")
topSplice(ex)
plotSplice(ex, xlab="Transcript")

## End(Not run)
```

`dim` *Retrieve the Dimensions of an RGList, MAList or MArrayLM Object*

Description

Retrieve the number of rows (genes) and columns (arrays) for an RGList, MAList or MArrayLM object.

Usage

```
## S3 method for class 'RGList'
dim(x)
```

Arguments

`x`	an object of class `RGList`, `MAList` or `MArrayLM`

Details

Microarray data objects share many analogies with ordinary matrices in which the rows correspond to spots or genes and the columns to arrays. These methods allow one to extract the size of microarray data objects in the same way that one would do for ordinary matrices.

A consequence is that row and column commands `nrow(x)`, `ncol(x)` and so on also work.

Value

Numeric vector of length 2. The first element is the number of rows (genes) and the second is the number of columns (arrays).

Author(s)

Gordon Smyth

See Also

dim in the base package.

02.Classes gives an overview of data classes used in LIMMA.

Examples

```
M <- A <- matrix(11:14,4,2)
rownames(M) <- rownames(A) <- c("a","b","c","d")
colnames(M) <- colnames(A) <- c("A1","A2")
MA <- new("MAList",list(M=M,A=A))
dim(M)
ncol(M)
nrow(M)
```

---

dimnames *Retrieve the Dimension Names of an RGList, MAList, EList, EListRaw or MArrayLM Object*

Description

Retrieve the dimension names of a microarray data object.

Usage

```
## S3 method for class 'RGList'
dimnames(x)
## S3 replacement method for class 'RGList'
dimnames(x) <- value
```

Arguments

x	an object of class RGList, MAList, EList, EListRaw or (not for assignment) MArrayLM
value	a possible value for dimnames(x): see dimnames

Details

The dimension names of a microarray object are the same as those of the most important matrix component of that object.

A consequence is that `rownames` and `colnames` will work as expected.

Value

Either `NULL` or a list of length 2. If a list, its components are either `NULL` or a character vector the length of the appropriate dimension of `x`.

Author(s)

Gordon Smyth

See Also

`dimnames` in the base package.

02.Classes gives an overview of data classes used in LIMMA.

---

dupcor *Correlation Between Duplicates or Within Blocks*

Description

Estimate the intra-block correlation given a block structure for the arrays or samples.

Usage

```
duplicateCorrelation(object, design=NULL, ndups=2L, spacing=1L, block=NULL,
                     trim=0.15, weights=NULL, nthreads=1L)
```

Arguments

`object`	A matrix-like data object containing log-ratios or log-expression values for a series of samples, with rows corresponding to genes and columns to samples. Any type of data object that can be processed by `getEAWP` is acceptable.
`design`	the design matrix of the microarray experiment, with rows corresponding to arrays and columns to comparisons to be estimated. The number of rows must match the number of columns of `object`. Defaults to the unit vector meaning that the arrays are treated as replicates.
`ndups`	a positive integer giving the number of times each gene is printed on an array. `nrow(object)` must be divisible by `ndups`. Ignored if `block` is specified.
`spacing`	the spacing between the rows of `object` corresponding to duplicate spots, `spacing=1` for consecutive spots
`block`	vector or factor specifying a blocking variable

`trim` the fraction of observations to be trimmed from each end of `tanh(all.correlations)` when computing the trimmed mean.

`weights` an optional numeric matrix of the same dimension as `object` containing weights for each spot. If smaller than `object` then it will be filled out to the same size.

`nthreads` positive integer giving the number of OpenMP threads used for genewise correlation estimation. Defaults to 1. Builds without OpenMP run serially.

Details

When `block=NULL`, this function estimates the correlation between duplicate spots (regularly spaced within-array replicate spots). If `block` is not null, this function estimates the correlation between repeated observations on the blocking variable. Typically the blocks are biological replicates and repeated observations on the same block may be correlated. In either case, the correlation is estimated by fitting a mixed linear model by REML individually for each gene. The function also returns a consensus correlation, which is a robust average of the individual correlations, intended for input to functions such as `lmFit`, `gls.series` or `voom`.

It is not possible to estimate correlations between duplicate spots and with sample blocks simultaneously. If `block` is not null, then the function will set `ndups=1`, which is equivalent to ignoring duplicate spots.

For this function to return statistically useful results, there must be at least two more arrays than the number of coefficients to be estimated, i.e., two more than the column rank of `design`.

The function may take a long time to execute as it fits a mixed linear model for each gene using an iterative algorithm.

If present, `ndups` and `spacing` will be extracted from `object$printer$ndups` and `object$printer$spacing`.

Value

A list with components

`consensus.correlation` the average estimated inter-duplicate correlation. The average is the trimmed mean of the individual correlations on the atanh-transformed scale.

`cor` same as `consensus.correlation`, for compatibility with earlier versions of the software

`atanh.correlations` numeric vector of length `nrow(object)/ndups` giving the individual genewise atanh-transformed correlations.

Author(s)

Gordon Smyth

References

Smyth GK, Michaud J, Scott H (2005). The use of within-array replicate spots for assessing differential expression in microarray experiments. *Bioinformatics* 21(9), 2067-2075. doi:10.1093/bioinformatics/bti270. See also the preprint with corrections: https://gksmyth.github.io/pubs/dupcor.pdf.

See Also

The two-component mixed linear models are fitted by native C code.

An overview of linear model functions in limma is given by 06.LinearModels.

Examples

```
# Simulate a paired experiment with incomplete blocks
Block <- c(1,1,2,2,3,3,4,4,5,6,7,8)
Treat <- factor(c(1,2,1,2,1,2,1,2,1,2,1,2))
design <- model.matrix(~Treat)

ngenes <- 50
nsamples <- 12
y <- matrix(rnorm(ngenes*nsamples),ngenes,nsamples)
rownames(y) <- paste0("Gene",1:ngenes)

# Estimate the within-block correlation
dupcor <- duplicateCorrelation(y,design,block=Block)
dupcor$consensus.correlation

# Estimate the treatment effect using both complete and incomplete blocks
fit <- lmFit(y,design,block=Block,correlation=dupcor$consensus)
fit <- eBayes(fit)
topTable(fit,coef=2)
```

---

eBayes *Empirical Bayes Statistics for Differential Expression*

Description

Given a linear model fit from `lmFit`, compute moderated t-statistics, moderated F-statistic, and log-odds of differential expression by empirical Bayes moderation of the standard errors towards a global value.

Usage

```
eBayes(fit, proportion = 0.01, stdev.coef.lim = c(0.1,4),
       trend = FALSE, span = NULL,
       robust = FALSE, winsor.tail.p = c(0.05,0.1), legacy = NULL)
treat(fit, fc = 1.2, lfc = NULL, trend = FALSE,
     robust = FALSE, winsor.tail.p = c(0.05,0.1), legacy = NULL, upshot = FALSE)
```

Arguments

`fit`	an `MArrayLM` fitted model object produced by `lmFit` or `contrasts.fit`. For `ebayes` only, `fit` can alternatively be an unclassed list produced by `lm.series`, `gls.series` or `mrlm` containing components `coefficients`, `stdev.unscaled`, `sigma` and `df.residual`.

proportion	numeric value between 0 and 1, assumed proportion of genes which are differentially expressed
stdev.coef.lim	numeric vector of length 2, assumed lower and upper limits for the standard deviation of log2-fold-changes for differentially expressed genes
trend	logical, should an intensity-dependent trend be allowed for the prior variance? If FALSE then the prior variance is constant. Alternatively, trend can be a row-wise numeric vector, which will be used as the covariate for the prior variance.
span	lowess span used for prior variance trend.
robust	logical, should the estimation of df.prior and var.prior be robustified against outlier sample variances?
winsor.tail.p	numeric vector of length 1 or 2, giving left and right tail proportions of x to Winsorize (if robust=TRUE and legacy=TRUE).
legacy	logical. If FALSE then the new hyperparameter estimation (introduced in limma 3.61.8) will be used, if TRUE the earlier hyperparameter estimation will be used. If NULL, then the new method will be used when the residual degrees of freedom are not all equal and the old method will be used otherwise. legacy=FALSE will also be assumed if span is not NULL.
fc	minimum fold-change below which changes are not considered scientifically meaningful.
lfc	minimum log2-fold-change below which changes not considered scientifically meaningful. Defaults to log2(fc). If specified then takes precedence over fc.
upshot	logical. If TRUE, then a more powerful test is conducted assuming that the true logFC are uniformly distributed over the null interval from -lfc to lfc. If FALSE, then the standard TREAT p-values are computed assuming the worst-case scenario of true logFC equal to lfc.

Details

These functions are used to rank genes in order of evidence for differential expression. They use an empirical Bayes method to squeeze the genewise residual variances towards a common value (or towards a global trend) (Smyth, 2004; Phipson et al, 2016). The degrees of freedom for the individual variances are increased to reflect the extra information gained from the empirical Bayes moderation, resulting in increased statistical power to detect differential expression.

These functions accept as input an MArrayLM fitted model object fit produced by lmFit. The columns of fit define a set of contrasts which are to be tested equal to zero. The fitted model object may have been processed by contrasts.fit before being passed to eBayes to convert the coefficients of the original design matrix into an arbitrary number of contrasts.

The empirical Bayes moderated t-statistics test each individual contrast equal to zero. For each gene (row), the moderated F-statistic tests whether all the contrasts are zero. The F-statistic is an overall test computed from the set of t-statistics for that probe. This is exactly analogous to the relationship between t-tests and F-statistics in conventional anova, except that the residual mean squares have been moderated between genes.

The estimates s2.prior and df.prior are computed by one of fitFDist, fitFDistRobustly or fitFDistUnequalDF1 (depending on settings for robust and legacy). s2.post is the weighted

average of s2.prior and sigma^2 with weights proportional to df.prior and df.residual respectively. The log-odds of differential expression lods was called the *B-statistic* by Loennstedt and Speed (2002). The F-statistics F are computed by classifyTestsF with fstat.only=TRUE.

eBayes does not compute ordinary t-statistics because they always have worse performance than the moderated versions. The ordinary (unmoderated) t-statistics can, however, be easily extracted from the linear model output for comparison purposes—see the example code below.

treat computes empirical Bayes moderated-t p-values relative to a minimum fold-change threshold. Instead of testing for genes that have true log-fold-changes different from zero, it tests whether the true log2-fold-change is greater than lfc in absolute value (McCarthy and Smyth, 2009). In other words, it uses an interval null hypothesis, where the interval is [-lfc,lfc]. When the number of DE genes is large, treat is often useful for giving preference to larger fold-changes and for prioritizing genes that are biologically important. treat is concerned with p-values rather than posterior odds, so it does not compute the B-statistic lods. The idea of thresholding doesn't apply to F-statistics in a straightforward way, so moderated F-statistics are also not computed. When fc=1 and lfc=0, treat is identical to eBayes, except that F-statistics and B-statistics are not computed. The fc threshold is usually chosen relatively small, because genes need to have fold changes substantially greater than the testing threshold in order to be considered statistically significant. Typical values for fc are 1.1, 1.2 or 1.5. The top genes chosen by treat can be examined using topTreat.

The treat threshold can be specified either as a fold-change via fc or as a log2-fold-change via lfc, with lfc = log2(fc). Note that the treat testing procedure is considerably more rigorous and conservative than simply applying the same fc values as a fold-change cutoff to the list of differentially expressed genes. Indeed, the observed log2-fold-change needs to be substantially larger than lfc for a gene to be called as statistically significant by treat. The threshold should be chosen as a small value below which results should be ignored rather than as a target fold-change. In practice, modest values for fc such as 1.1, 1.2 or 1.5 are usually the most useful. Setting fc=1.2 or fc=1.5 will usually cause most differentially expressed genes to have estimated fold-changes of 2-fold or greater, depending on the sample size and precision of the experiment. Larger thresholds are usually overly conservative and counter productive. In general, the fc threshold should be chosen sufficiently small so that a worthwhile number of DE genes remain, otherwise the purpose of prioritizing genes with larger fold-changes will be defeated.

The use of eBayes or treat with trend=TRUE is known as the *limma-trend* method (Law et al, 2014; Phipson et al, 2016). With this option, an intensity-dependent trend is fitted to the prior variances s2.prior. Specifically, squeezeVar is called with the covariate equal to Amean, the average log2-intensity for each gene. The trend that is fitted can be examined by plotSA. limma-trend is useful for processing expression values that show a mean-variance relationship. This is often useful for microarray data, and it can also be applied to RNA-seq counts that have been converted to log2-counts per million (logCPM) values (Law et al, 2014). When applied to RNA-seq logCPM values, limma-trend give similar results to the voom method. The voom method incorporates the mean-variance trend into the precision weights, whereas limma-trend incorporates the trend into the empirical Bayes moderation. limma-trend is somewhat simpler than voom because it assumes that the sequencing depths (library sizes) are not wildly different between the samples and it applies the mean-variance trend on a genewise basis rather than to individual observations. limma-trend is recommended for RNA-seq analysis when the library sizes are reasonably consistent (less than 3-fold difference from smallest to largest) because of its simplicity and speed.

If robust=TRUE then the robust empirical Bayes procedure of Phipson et al (2016) is used. This is frequently useful to protect the empirical Bayes procedure against hyper-variable or hypo-variable genes, especially when analyzing RNA-seq data. See squeezeVar for more details.

In limma 3.61.8 (August 2024), the new function fitFDistUnequalDF1 was introduced to improve estimation of the hyperparameters s2.prior and df.prior, especially when not all genes have the same residual degrees of freedom. fitFDistUnequalDF1 is a potential replacement for the original functions fitFDist and fitFDistRobustly and the argument legacy is provided to control backward compatibility. The new hyperparameter estimation will be used if legacy=FALSE and the original methods will be used if legacy=TRUE. If legacy=NULL, then the new method will be used if the residual degrees of freedom are unequal and the original methods otherwise. Unequal residual degrees of freedom arise in limma pipelines when the expression matrix includes missing values or from the quasi-likelihood pipeline in edgeR v4.

If upshot=FALSE, then the original TREAT method is used, which computes p-values relative to the "worst case" scenario in which the true logFCs are on the boundary of the null hypothesis interval closest to the observed logFC. If upshot=TRUE, then a new more powerful UPSHOT approach suggested by Nikos Ignatiadis (University of Chicago) is used. The UPSHOT test assumes that the true logFC are uniformly distributed over the null interval from -lfc to lfc. The UPSHOT test gives correct error rate control over any unimodal distribution of true logFCs over the null interval.

Value

eBayes produces an object of class MArrayLM (see MArrayLM-class) containing everything found in fit plus the following added components:

t	numeric matrix of moderated t-statistics.
p.value	numeric matrix of two-sided p-values corresponding to the t-statistics.
lods	numeric matrix giving the log-odds of differential expression (on the natural log scale).
s2.prior	estimated prior value for sigma^2. A row-wise vector if covariate is non-NULL, otherwise a single value.
df.prior	degrees of freedom associated with s2.prior. A row-wise vector if robust=TRUE, otherwise a single value.
df.total	row-wise numeric vector giving the total degrees of freedom associated with the t-statistics for each gene. Equal to df.prior+df.residual or sum(df.residual), whichever is smaller.
s2.post	row-wise numeric vector giving the posterior values for sigma^2.
var.prior	column-wise numeric vector giving estimated prior values for the variance of the log2-fold-changes for differentially expressed genes for each contrast. Used for evaluating lods.
F	row-wise numeric vector of moderated F-statistics for testing all contrasts defined by the columns of fit simultaneously equal to zero.
F.p.value	row-wise numeric vector giving p-values corresponding to F.

The matrices t, p.value and lods have the same dimensions as the input object fit, with rows corresponding to genes and columns to coefficients or contrasts. The vectors s2.prior, df.prior, df.total, F and F.p.value correspond to rows, with length equal to the number of genes. The vector var.prior corresponds to columns, with length equal to the number of contrasts. If s2.prior or df.prior have length 1, then the same value applies to all genes.

s2.prior, df.prior and var.prior contain empirical Bayes hyperparameters used to obtain df.total, s2.post and lods.

`treat` produces an `MArrayLM` object similar to that from `eBayes` but without `lods`, `var.prior`, `F` or `F.p.value`.

Note

The algorithm used by `eBayes` and `treat` with `robust=TRUE` was revised slightly in limma 3.27.6. The minimum `df.prior` returned may be slightly smaller than previously.

Author(s)

Gordon Smyth and Davis McCarthy

References

Law CW, Chen Y, Shi W, Smyth GK (2014). Voom: precision weights unlock linear model analysis tools for RNA-seq read counts. *Genome Biology* 15, R29. doi:10.1186/gb2014152r29. See also the preprint version at https://gksmyth.github.io/pubs/VoomPreprint.pdf incorporating some notational corrections.

Loennstedt I, Speed TP (2002). Replicated microarray data. *Statistica Sinica* 12, 31-46.

McCarthy DJ, Smyth GK (2009). Testing significance relative to a fold-change threshold is a TREAT. *Bioinformatics* 25(6), 765-771. doi:10.1093/bioinformatics/btp053.

Phipson B, Lee S, Majewski IJ, Alexander WS, Smyth GK (2016). Robust hyperparameter estimation protects against hypervariable genes and improves power to detect differential expression. *Annals of Applied Statistics* 10(2), 946-963. doi:10.1214/16AOAS920.

Smyth GK (2004). Linear models and empirical Bayes methods for assessing differential expression in microarray experiments. *Statistical Applications in Genetics and Molecular Biology* 3(1), Article 3. doi:10.2202/15446115.1027. See also the preprint version https://gksmyth.github.io/pubs/ebayes.pdf incorporating corrections to 30 June 2009.

See Also

`squeezeVar`, `fitFDist`, `tmixture.matrix`, `plotSA`.

An overview of linear model functions in limma is given by 06.LinearModels.

Examples

```
#  See also lmFit examples

#  Simulate gene expression data,
#  6 microarrays and 100 genes with one gene differentially expressed
set.seed(2016)
sigma2 <- 0.05 / rchisq(100, df=10) * 10
y <- matrix(rnorm(100*6,sd=sqrt(sigma2)),100,6)
design <- cbind(Intercept=1,Group=c(0,0,0,1,1,1))
y[1,4:6] <- y[1,4:6] + 1
fit <- lmFit(y,design)

#  Moderated t-statistic
fit <- eBayes(fit)
```

```
topTable(fit,coef=2)

#  Ordinary t-statistic
ordinary.t <- fit$coef[,2] / fit$stdev.unscaled[,2] / fit$sigma

# Treat relative to a 10% fold-change
tfit <- treat(fit, fc=1.1)
topTreat(tfit,coef=2)
```

EList-class *Expression List (EList) class*

Description

A list-based S4 class for storing expression values (E-values), for example for a set of one-channel microarrays or a set of RNA-seq samples. EListRaw holds expression values on the raw scale. EList holds expression values on the log scale, usually after background correction and normalization.

EListRaw objects are often created by read.maimages, while EList objects are often created by normalizeBetweenArrays or by voom. Alternatively, an EList object can be created directly by new("EList",x), where x is a list.

Required Components

These classes contain no slots (other than .Data), but objects should contain a list component E:

E numeric matrix containing expression values. In an EListRaw object, the expression values are unlogged, while in an EList object, they are log2 values. Rows correspond to probes and columns to samples.

Optional Components

Optional components include:

Eb numeric matrix containing unlogged background expression values, of same dimensions as E. For an EListRaw object only.

weights numeric matrix of same dimensions as E containing relative spot quality weights. Elements should be non-negative.

other list containing other matrices, all of the same dimensions as E.

genes data.frame containing probe information. Should have one row for each probe. May have any number of columns.

targets data.frame containing information on the target RNA samples. Rows correspond to samples. May have any number of columns.

Valid EList or EListRaw objects may contain other optional components, but all probe or sample information should be contained in the above components.

Methods

These classes inherit directly from class `list` so any operation appropriate for lists will work on objects of this class. In addition, `EList` objects can be subsetted and combined. `EList` objects will return dimensions and hence functions such as `dim`, `nrow` and `ncol` are defined. `ELists` also inherit a `show` method from the virtual class `LargeDataObject`, which means that `ELists` will print in a compact way.

Author(s)

Gordon Smyth

See Also

02.Classes gives an overview of all the classes defined by this package.

`ExpressionSet` is a more formal class in the Biobase package used for the same purpose.

Examples

```
# Two ways to make an EList object:

y <- matrix(rnorm(10,5),10,5)
rownames(y) <- paste0("Gene",1:10)
colnames(y) <- LETTERS[1:5]
Genes <- data.frame(Chr=sample(1:21,10))
row.names(Genes) <- row.names(y)

# Create the object, then add components:
E <- new("EList")
E$E <- y
E$genes <- Genes

# Create with components:
E <- new("EList", list(E=y, genes=Genes))
```

---

exprs.MA *Extract Log-Expression Matrix from MAList*

Description

Extract the matrix of log-expression values from an `MAList` object.

Usage

```
exprs.MA(MA)
```

Arguments

`MA`	an `MAList` object.

Details

Converts M and A-values to log-expression values. The output matrix will have two columns for each array, in the order green, red for each array.

This contrasts with `as.matrix.MAList` which extracts the M-values only, or `RG.MA` which converts to expression values in `RGList` form.

Value

A numeric matrix with twice the columns of the input.

Author(s)

Gordon Smyth

See Also

02.Classes gives an overview of data classes used in LIMMA.

---

fitFDist *Moment Estimation of Scaled F-Distribution*

Description

Moment estimation of the parameters of a scaled F-distribution given one of the degrees of freedom. These functions are called internally by `eBayes` and `squeezeVar` and are not usually called directly by a user.

Usage

```
fitFDist(x, df1, covariate = NULL)
fitFDistRobustly(x, df1, covariate = NULL,
                 winsor.tail.p = c(0.05,0.1), trace = FALSE)
fitFDistUnequalDF1(x, df1, covariate = NULL, span = NULL,
                   robust = FALSE, prior.weights = NULL)
```

Arguments

`x`	numeric vector or array of positive values representing a sample from a scaled F-distribution.
`df1`	the first degrees of freedom of the F-distribution. Can be a single value, or else a vector of the same length as `x`.
`covariate`	if non-`NULL`, the estimated scale value will depend on this numeric covariate.
`winsor.tail.p`	numeric vector of length 1 or 2, giving left and right tail proportions of `x` to Winsorize.
`trace`	logical value indicating whether a trace of the iteration progress should be printed.

span	lowess span parameter. Defaults to chooseLowessSpan(length(x),small.n=500).
robust	logical. Should outlier values of x be down-weighted with results similar to fitFDistRobustly?
prior.weights	numeric vector of (non-negative) prior weights.

Details

fitFDist() implements an algorithm proposed by Smyth (2004) and Phipson et al (2016). It estimates scale and df2 under the assumption that x is distributed as scale times an F-distributed random variable on df1 and df2 degrees of freedom. The parameters are estimated using the method of moments, specifically from the mean and variance of the x values on the log-scale.

When covariate is supplied, a spline curve trend will be estimated for the x values and the estimation will be adjusted for this trend (Phipson et al, 2016).

fitFDistRobustly is similar to fitFDist except that it computes the moments of the Winsorized values of x, making it robust against left and right outliers. Larger values for winsor.tail.p produce more robustness but less efficiency. When covariate is supplied, a loess trend is estimated for the x values. The robust method is described by Phipson et al (2016).

As well as estimating the F-distribution for the bulk of the cases, i.e., with outliers discounted, fitFDistRobustly also returns an estimated F-distribution with reduced df2 that might be appropriate for each outlier case.

fitFDistUnequalDF1 was introduced in limma 3.61.8 and gives special attention to the possibility that the degrees of freedom df1 might be unequal and might include non-integer values. The most important innovation of fitFDistUnequalDF1 is downweighting of observations with lower degrees of freedom, to give more precise estimation overall. It also allows the possibility of prior weights, which can be used to downweight unreliable x values for reasons other than small df1. fitFDistUnequalDF1 implements a different robust estimation strategy to fitFDistRobustly. Instead of Winsorizing the x values, potential outliers are instead downweighted using the prior weights. Whereas fitFDist and fitFDistRobustly use unweighted moment estimation for both scale and df2, fitFDistUnequalDF1 uses weighted moment estimation for scale and profile maximum likelihood for df2.

fitFDistUnequalDF1 gives improved performance over fitFDist and fitFDistRobustly, especially when the degrees of freedom are unequal but also to a lesser extent when the degrees of freedom are equal. Unequal residual degrees of freedom arise in limma pipelines when the expression matrix includes missing values, or from edgeR::voomLmFit or from the quasi-likelihood pipeline in edgeR v4 (Chen et al 2024). The edgeR v4 pipeline produces fractional degrees of freedom including, potentially, degrees of freedom less than 1.

Value

fitFDist or fitFDistUnequalDF1 with robust=FALSE produces a list with the following components:

scale	scale factor for F-distribution. A vector if covariate is non-NULL, otherwise a scalar.
df2	the second degrees of freedom of the fitted F-distribution.

fitFDistRobustly returns the following components as well:

tail.p.value	right tail probability of the scaled F-distribution for each x value.
prob.outlier	posterior probability that each case is an outlier relative to the scaled F-distribution with degrees of freedom df1 and df2.
df2.outlier	the second degrees of freedom associated with extreme outlier cases.
df2.shrunk	numeric vector of values for the second degrees of freedom, with shrunk values for outliers. Most values are equal to df2, but outliers have reduced values depending on how extreme each case is. All values lie between df2.outlier and df2.

Note

The algorithm used by fitFDistRobustly was revised slightly in limma 3.27.6. The prob.outlier value, which is the lower bound for df2.shrunk, may be slightly smaller than previously.

Author(s)

Gordon Smyth, Belinda Phipson (fitFDistRobustly) and Lizhong Chen (fitFDistUnequalDF1).

References

Smyth GK (2004). Linear models and empirical Bayes methods for assessing differential expression in microarray experiments. *Statistical Applications in Genetics and Molecular Biology* 3(1), Article 3. doi:10.2202/15446115.1027. See also the preprint version at https://gksmyth.github.io/pubs/ebayes.pdf with corrections.

Phipson B, Lee S, Majewski IJ, Alexander WS, Smyth GK (2016). Robust hyperparameter estimation protects against hypervariable genes and improves power to detect differential expression. *Annals of Applied Statistics* 10(2), 946-963. doi:10.1214/16AOAS920.

Chen Y, Chen L, Lun ATL, Baldoni PL, Smyth GK (2025). edgeR v4: powerful differential analysis of sequencing data with expanded functionality and improved support for small counts and larger datasets. *Nucleic Acids Research* 53(2), gkaf018. doi:10.1093/nar/gkaf018.

See Also

This function is called by squeezeVar, which in turn is called by eBayes and treat.

This function calls trigammaInverse.

Examples

```
x <- rf(100,df1=8,df2=16)
fitFDist(x,df1=8)
```

`fitGammaIntercept` *Fit Intercept to Vector of Gamma Distributed Variates*

Description

Fit Intercept to Vector of Gamma Distributed Variates

Usage

```
fitGammaIntercept(y,offset=0,maxit=1000)
```

Arguments

`y`	numeric vector of positive response values.
`offset`	numeric vector giving known part of the expected value of `y`. Can be a single value, or else a vector of the same length as `y`.
`maxit`	maximum number of Newton iterations to be done.

Details

The values `y` are assumed to follow a gamma distribution with common shape parameter and with expected values given by `x+offset`. The function implements a globally convergent Newton iteration to estimate `x`.

Value

Numeric value giving intercept.

Author(s)

Gordon Smyth and Belinda Phipson

References

Phipson B (2013). *Empirical Bayes modelling of expression profiles and their associations*. PhD Thesis. University of Melbourne, Australia.

See Also

This function is called by genas.

Examples

```
offset <- runif(10)
x <- 9
mu <- x+offset
y <- rgamma(10,shape=20,scale=mu/20)
fitGammaIntercept(y,offset=offset)
```

`fitmixture` *Fit Mixture Model by Non-Linear Least Squares*

Description

Fit Mixture Model by Non-Linear Least Squares

Usage

```
fitmixture(log2e, mixprop, niter = 4, trace = FALSE)
```

Arguments

`log2e`	a numeric matrix containing log2 expression values. Rows correspond to probes for genes and columns to RNA samples.
`mixprop`	a vector of length `ncol(log2e)` giving the mixing proportion (between 0 and 1) for each sample.
`niter`	integer number of iterations.
`trace`	logical. If `TRUE`, summary working estimates are output from each iteration.

Details

A mixture experiment is one in which two reference RNA sources are mixed in different proportions to create experimental samples. Mixture experiments have been used to evaluate genomic technologies and analysis methods (Holloway et al, 2006). This function uses all the data for each gene to estimate the expression level of the gene in each of two pure samples.

The function fits a nonlinear mixture model to the log2 expression values for each gene. The expected values of `log2e` for each gene are assumed to be of the form `log2( mixprop*Y1 + (1-mixprop)*Y2 )` where `Y1` and `Y2` are the expression levels of the gene in the two reference samples being mixed. The `mixprop` values are the same for each gene but `Y1` and `Y2` are specific to the gene. The function returns the estimated values `A=0.5*log2(Y1*Y2)` and `M=log2(Y2/Y1)` for each gene.

The nonlinear estimation algorithm implemented in `fitmixture` uses a nested Gauss-Newton iteration (Smyth, 1996). It is fully vectorized so that the estimation is done for all genes simultaneously.

Value

List with three components:

`A`	numeric vector giving the estimated average log2 expression of the two reference samples for each gene
`M`	numeric vector giving estimated log-ratio of expression between the two reference samples for each gene
`stdev`	standard deviation of the residual term in the mixture model for each gene

Author(s)

Gordon K Smyth

References

Holloway AJ, Oshlack A, Diyagama DS, Bowtell DDL, Smyth GK (2006). Statistical analysis of an RNA titration series evaluates microarray precision and sensitivity on a whole-array basis. *BMC Bioinformatics* 7, Article 511. doi:10.1186/147121057511.

Smyth GK (1996). Partitioned algorithms for maximum likelihood and other nonlinear estimation. *Statistics and Computing*, 6, 201-216. https://gksmyth.github.io/pubs/partitio.pdf

Examples

```
ngenes <- 100
TrueY1 <- rexp(ngenes)
TrueY2 <- rexp(ngenes)
mixprop <- matrix(c(0,0.25,0.75,1),1,4)
TrueExpr <- TrueY1

log2e <- log2(TrueExpr) + matrix(rnorm(ngenes*4),ngenes,4)*0.1
out <- fitmixture(log2e,mixprop)

# Plot true vs estimated log-ratios
plot(log2(TrueY1/TrueY2), out$M)
```

---

fitted.MArrayLM *Fitted Values Method for MArrayLM Fits*

Description

Obtain fitted values from a fitted linear model object.

Usage

```
## S3 method for class 'MArrayLM'
fitted(object, ...)
```

Arguments

object	a fitted object of class inheriting from "MArrayLM".
...	other arguments are not currently used.

Value

A numeric matrix of fitted values.

Author(s)

Gordon Smyth

See Also

residuals.MArrayLM. See also fitted in the stats package.

---

genas *Genuine Association of Gene Expression Profiles*

Description

Calculates biological correlation between two gene expression profiles.

Usage

```
genas(fit, coef=c(1,2), subset="all", plot=FALSE, alpha=0.4)
```

Arguments

fit	an MArrayLM fitted model object produced by lmFit or contrasts.fit and followed by eBayes.
coef	numeric vector of length 2 indicating which columns in the fit object are to be correlated.
subset	character string indicating which subset of genes to include in the correlation analysis. Choices are "all", "Fpval", "p.union", "p.int", "logFC" or "predFC".
plot	logical, should a scatterplot be produced summarizing the correlation analysis?
alpha	numeric value between 0 and 1 determining the transparency of the technical and biological ellipses if a plot is produced. alpha=0 indicates fully transparent and alpha=1 indicates fully opaque.

Details

The function estimates the biological correlation between two different contrasts in a linear model. By biological correlation, we mean the correlation that would exist between the log2-fold changes (logFC) for the two contrasts, if measurement error could be eliminated and the true log-fold-changes were known. This function is motivated by the fact that different contrasts for a linear model are often strongly correlated in a technical sense. For example, the estimated logFC for multiple treatment conditions compared back to the same control group will be positively correlated even in the absence of any biological effect. This function aims to separate the biological from the technical components of the correlation. The method is explained briefly in Majewski et al (2010) and in full detail in Phipson (2013).

The subset argument specifies whether and how the fit object should be subsetted. Ideally, only genes that are truly differentially expressed for one or both of the contrasts should be used to estimate the biological correlation. The default is "all", which uses all genes in the fit object to

estimate the biological correlation. The option "Fpval" chooses genes based on how many F-test p-values are estimated to be truly significant using the function propTrueNull. This should capture genes that display any evidence of differential expression in either of the two contrasts. The options "p.union" and "p.int" are based on the moderated t p-values from both contrasts. From the propTrueNull function an estimate of the number of p-values truly significant in either of the two contrasts can be obtained. "p.union" takes the union of these genes and "p.int" takes the intersection of these genes. The other options, "logFC" and "predFC", subset on genes that attain a logFC or predFC at least as large as the 90th percentile of the log fold changes or predictive log fold changes on the absolute scale.

The plot option is a logical argument that specifies whether or not to plot a scatter plot of log-fold-changes for the two contrasts. The biological and technical correlations are overlaid on the scatterplot using semi-transparent ellipses. library(ellipse) is required to enable the plotting of ellipses.

Value

genas produces a list with the following components:

technical.correlation
: estimate of the technical correlation

biological.correlation
: estimate of the biological correlation

covariance.matrix
: estimate of the covariance matrix from which the biological correlation is obtained

deviance
: the likelihood ratio test statistic used to test whether the biological correlation is equal to 0

p.value
: the p.value associated with deviance

n
: the number of genes used to estimate the biological correlation

Note

At present, genas assumes that technical correlations between coefficients are the same for all genes, and hence it only works with fit objects that were created without observation weights or missing values. It does not work with voom pipelines, because these involve observation weights.

Author(s)

Belinda Phipson and Gordon Smyth

References

Majewski IJ, Ritchie ME, Phipson B, Corbin J, Pakusch M, Ebert A, Busslinger M, Koseki H, Hu Y, Smyth GK, Alexander WS, Hilton DJ, Blewitt ME (2010). Opposing roles of polycomb repressive complexes in hematopoietic stem and progenitor cells. *Blood* 116, 731-739. doi:10.1182/blood-200912260760.

Phipson B (2013). *Empirical Bayes modelling of expression profiles and their associations*. PhD Thesis. University of Melbourne, Australia. http://hdl.handle.net/11343/38162

Ritchie ME, Phipson B, Wu D, Hu Y, Law CW, Shi W, Smyth GK (2015). limma powers differential expression analyses for RNA-sequencing and microarray studies. *Nucleic Acids Research* 43, e47. doi:10.1093/nar/gkv007.

See Also

lmFit, eBayes, contrasts.fit

Examples

```
#  Simulate gene expression data

#  Three conditions (Control, A and B) and 1000 genes
ngene <- 1000
mu.A <- mu.B <- mu.ctrl <- rep(5,ngene)

#  200 genes are differentially expressed.
#  All are up in condition A and down in B
#  so the biological correlation is negative.
mu.A[1:200] <- mu.ctrl[1:200]+2
mu.B[1:200] <- mu.ctrl[1:200]-2

#  Two microarrays for each condition
mu <- cbind(mu.ctrl,mu.ctrl,mu.A,mu.A,mu.B,mu.B)
y <- matrix(rnorm(6000,mean=mu,sd=1),ngene,6)

# two experimental groups and one control group with two replicates each
group <- factor(c("Ctrl","Ctrl","A","A","B","B"), levels=c("Ctrl","A","B"))
design <- model.matrix(~group)

# fit a linear model
fit <- lmFit(y,design)
fit <- eBayes(fit)

#  Estimate biological correlation between the logFC profiles
#  for A-vs-Ctrl and B-vs-Ctrl
genas(fit, coef=c(2,3), plot=TRUE, subset="F")
```

---

geneSetTest *Mean-rank Gene Set Test*

Description

Test whether a set of genes is highly ranked relative to other genes in terms of a given statistic. Genes are assumed to be independent.

Usage

```
geneSetTest(index, statistics, alternative = "mixed", type= "auto",
            ranks.only = TRUE, nsim=9999)
wilcoxGST(index, statistics, ...)
```

Arguments

index	index vector for the gene set. This can be a vector of indices, or a logical vector of the same length as statistics or, in general, any vector such that statistic[index] gives the statistic values for the gene set to be tested.
statistics	vector, any genewise statistic by which genes can be ranked.
alternative	character string specifying the alternative hypothesis, must be one of "mixed", "either", "up" or "down". "two.sided", "greater" and "less" are also permitted as synonyms for "either", "up" and "down" respectively.
type	character string specifying whether the statistics are signed (t-like, "t") or unsigned (F-like, "f") or whether the function should make an educated guess ("auto"). If the statistic is unsigned, then it is assumed that larger statistics are more significant.
ranks.only	logical, if TRUE only the ranks of the statistics are used.
nsim	number of random samples to take in computing the p-value. Not used if ranks.only=TRUE.
...	other arguments are passed to geneSetTest.

Details

These functions compute a p-value to test the hypothesis that the indexed test set of genes tends to be more highly ranked in terms of some test statistic compared to randomly chosen genes. The statistic might be any statistic of interest, for example a t-statistic or F-statistic for differential expression. Like all gene set tests, these functions can be used to detect differential expression for a group of genes, even when the effects are too small or there is too little data to detect the genes individually.

wilcoxGST is a synonym for geneSetTest with ranks.only=TRUE. This version of the test procedure was developed by Michaud et al (2008), who called it *mean-rank gene-set enrichment*.

geneSetTest performs a *competitive* test in the sense that genes in the test set are compared to other genes (Goeman and Buhlmann, 2007). If the statistic is a genewise test statistic for differential expression, then geneSetTest tests whether genes in the set are more differentially expressed than genes not in the set. By contrast, a *self-contained* gene set test such as roast tests whether genes in the test set are differentially expressed, in an absolute sense, without regard to any other genes on the array.

Because it is based on permuting genes, geneSetTest assumes that the different genes (or probes) are statistically independent. (Strictly speaking, it assumes that the genes in the set are no more correlated on average than randomly chosen genes.) If inter-gene correlations are present, then a statistically significant result from geneSetTest indicates either that the set is highly ranked or that the genes in the set are positively correlated on average (Wu and Smyth, 2012). Unless gene sets with positive correlations are particularly of interest, it may be advisable to use camera or cameraPR instead to adjust the test for inter-gene correlations. Inter-gene correlations are likely to be present in differential expression experiments with biologically heterogeneous experimental units. On the other hand, the assumption of independence between genes should hold when the replicates are purely technical, i.e., when there is no biological variability between the replicate arrays in each experimental condition.

The statistics are usually a set of probe-wise statistics arising for some comparison from a microarray experiment. They may be t-statistics, meaning that the genewise null hypotheses would be rejected for large positive or negative values, or they may be F-statistics, meaning that only

large values are significant. Any set of signed statistics, such as log-ratios, M-values or moderated t-statistics, are treated as t-like. Any set of unsigned statistics, such as F-statistics, posterior probabilities or chi-square tests are treated as F-like. If `type="auto"` then the statistics will be taken to be t-like if they take both positive and negative values and will be taken to be F-like if they are all of the same sign.

There are four possible alternatives to test for. `alternative=="up"` means the genes in the set tend to be up-regulated, with positive t-statistics. `alternative=="down"` means the genes in the set tend to be down-regulated, with negative t-statistics. `alternative=="either"` means the set is either up or down-regulated as a whole. `alternative=="mixed"` tests whether the genes in the set tend to be differentially expressed, without regard for direction. In this case, the test will be significant if the set contains mostly large test statistics, even if some are positive and some are negative.

The latter three alternatives are appropriate when there is a prior expectation that all the genes in the set will react in the same direction. The `"mixed"` alternative is appropriate if you know only that the genes are involved in the relevant pathways, possibly in different directions. The `"mixed"` alternative is the only meaningful alternative with F-like statistics.

The test statistic used for the gene-set-test is the mean of the statistics in the set. If `ranks.only` is `TRUE` then only the ranks of the statistics are used. In this case the p-value is obtained from a Wilcoxon test. If `ranks.only` is `FALSE`, then the p-value is obtained by simulation using `nsim` random sets of genes.

Value

numeric value giving the estimated p-value.

Note

Wu and Smyth (2012) show that `geneSetTest` does not correct for inter-gene correlations and is more likely to assign small p-values to sets containing positive correlated genes. The function `cameraPR` is recommended as an alternative.

Author(s)

Gordon Smyth and Di Wu

References

Wu D, Smyth GK (2012). Camera: a competitive gene set test accounting for inter-gene correlation. *Nucleic Acids Research* 40(17), e133. doi:10.1093/nar/gks461.

Goeman JJ, Buhlmann P (2007). Analyzing gene expression data in terms of gene sets: methodological issues. *Bioinformatics* 23, 980-987.

Michaud J, Simpson KM, Escher R, Buchet-Poyau K, Beissbarth T, Carmichael C, Ritchie ME, Schutz F, Cannon P, Liu M, Shen X, Ito Y, Raskind WH, Horwitz MS, Osato M, Turner DR, Speed TP, Kavallaris M, Smyth GK, Scott HS (2008). Integrative analysis of RUNX1 downstream pathways and target genes. *BMC Genomics* 9, 363. doi:10.1186/147121649363.

See Also

cameraPR, camera, roast, barcodeplot, wilcox.test.

There is a topic page on 10.GeneSetTests.

Examples

```
stat <- rnorm(100)
sel <- 1:10; stat[sel] <- stat[sel]+1
wilcoxGST(sel,stat)
```

---

getEAWP *Extract Basic Data from Expression Data Objects*

Description

Given an expression data object of any known class, get the expression values, weights, probe annotation and A-values that are needed for linear modeling. This function is called by the linear modeling functions in LIMMA.

Usage

```
getEAWP(object)
```

Arguments

object	any matrix-like object containing log-expression values. Can be an object of class MAList, EList, marrayNorm, PLMset, vsn, or any class inheriting from ExpressionSet, or any object that can be coerced to a numeric matrix.

Details

Rows correspond to probes and columns to RNA samples.

In the case of two-color microarray data objects (MAList or marrayNorm), Amean is the vector of row means of the matrix of A-values. For other data objects, Amean is the vector of row means of the matrix of expression values.

From April 2013, the rownames of the output exprs matrix are required to be unique. If object has no row names, then the output rownames of exprs are 1:nrow(object). If object has row names but with duplicated names, then the rownames of exprs are set to 1:nrow(object) and the original row names are preserved in the ID column of probes.

object should be a normalized data object. getEAWP will return an error if object is a non-normalized data object such as RGList or EListRaw, because these do not contain log-expression values.

Value

A list with components

exprs	numeric matrix of log-ratios, log-intensities or log-expression values
weights	numeric matrix of weights
probes	data.frame of probe-annotation
Amean	numeric vector of average log-expression for each probe

exprs is the only required component. The other components will be NULL if not found in the input object.

Author(s)

Gordon Smyth

See Also

02.Classes gives an overview of data classes used in LIMMA.

---

getLayout *Extract the Print Layout of an Array from the GAL File*

Description

From the Block, Row and Column information in a genelist, determine the number of grid rows and columns on the array and the number of spot rows and columns within each grid.

Usage

```
getLayout(gal, guessdups=FALSE)
getLayout2(galfile)
getDupSpacing(ID)
```

Arguments

gal	data.frame containing the GAL, i.e., giving the position and gene identifier of each spot
galfile	name or path of GAL file
guessdups	logical, if TRUE then try to determine number and spacing of duplicate spots, i.e., within-array replicates
ID	vector or factor of gene IDs

Details

A GenePix Array List (GAL) file is a list of genes and associated information produced by an Axon microarray scanner. The function `getLayout` determines the print layout from a data frame created from a GAL file or gene list. The data.frame must contain columns `Block`, `Column` and `Row`. (The number of tip columns is assumed to be either one or four.)

On some arrays, each probe may be duplicated a number of times (`ndups`) at regular intervals (`spacing`) in the GAL file. `getDupSpacing` determines valid values for `ndups` and `spacing` from a vector of IDs. If `guessdups=TRUE`, then `getLayout` calls `getDupSpacing`.

The function `getLayout2` attempts to determine the print layout from the header information of an actual GAL file.

Value

A `printlayout` object, which is a list with the following components. The last two components are present only if `guessdups=TRUE`.

`ngrid.r`	integer, number of grid rows on the arrays
`ngrid.c`	integer, number of grid columns on the arrays
`nspot.r`	integer, number of rows of spots in each grid
`nspot.c`	integer, number of columns of spots in each grid
`ndups`	integer, number of times each probe is printed on the array
`spacing`	integer, spacing between multiple printings of each probe

Author(s)

Gordon Smyth and James Wettenhall

See Also

An overview of LIMMA functions for reading data is given in 03.ReadingData.

Examples

```
# gal <- readGAL()
# layout <- getLayout(gal)
```

`getSpacing` *Get Numerical Spacing*

Description

Convert character to numerical spacing measure for within-array replicate spots.

Usage

```
getSpacing(spacing, layout)
```

Arguments

spacing	character string or integer. Acceptable character strings are "columns", "rows", "subarrays" or "topbottom". Integer values are simply passed through.
layout	list containing printer layout information

Details

"rows" means that duplicate spots are printed side-by-side by rows. These will be recorded in consecutive rows in the data object.

"columns" means that duplicate spots are printed side-by-side by columns. These will be separated in the data object by layout$nspot.r rows.

"subarrays" means that a number of sub-arrays, with identical probes in the same arrangement, are printed on each array. The spacing therefore will be the size of a sub-array.

"topbottom" is the same as "subarrays" when there are two sub-arrays.

Value

Integer giving spacing between replicate spots in the gene list.

Author(s)

Gordon Smyth

See Also

An overview of LIMMA functions for reading data is given in 03.ReadingData.

Examples

```
getSpacing("columns",list(ngrid.r=2,ngrid.c=2,nspot.r=20,nspot.c=19))
getSpacing("rows",list(ngrid.r=2,ngrid.c=2,nspot.r=20,nspot.c=19))
getSpacing("topbottom",list(ngrid.r=2,ngrid.c=2,nspot.r=20,nspot.c=19))
```

---

gls.series *Fit Linear Model to Microarray Data by Generalized Least Squares*

Description

Fit a linear model genewise to expression data from a series of microarrays. The fit is by generalized least squares allowing for correlation between duplicate spots or related arrays. This is a utility function for lmFit.

Usage

```
gls.series(M, design = NULL, ndups = 2, spacing = 1, block = NULL,
     correlation = NULL, weights = NULL, nthreads = 1L, contrasts = NULL, ...)
```

Arguments

M	numeric matrix containing log-ratio or log-expression values for a series of microarrays, rows correspond to genes and columns to arrays.
design	numeric design matrix defining the linear model, with rows corresponding to arrays and columns to comparisons to be estimated. The number of rows must match the number of columns of M. Defaults to the unit vector meaning that the arrays are treated as replicates.
ndups	positive integer giving the number of times each gene is printed on an array. nrow(M) must be divisible by ndups. Ignored if block is not NULL.
spacing	the spacing between the rows of M corresponding to duplicate spots, spacing=1 for consecutive spots. Ignored if block is not NULL.
block	vector or factor specifying a blocking variable on the arrays. Same length as ncol(M).
correlation	numeric value specifying the inter-duplicate or inter-block correlation.
weights	an optional numeric matrix of the same dimension as M containing weights for each spot. If it is of different dimension to M, it will be filled out to the same size.
nthreads	positive integer giving the number of OpenMP threads used for genewise native-code fits. Defaults to 1.
contrasts	optional numeric contrast matrix with rows corresponding to coefficients. If non-NULL, a contrast-space fit is returned. On the genewise (weighted or missing-value) path the contrasts are applied within each gene's fit, giving exact contrast stdev.unscaled; on the no-weights fast path the exact contrasts.fit is applied (all genes share the same unscaled covariance).
...	other optional arguments to be passed to dupcor.series.

Details

This is a utility function used by the higher level function lmFit. Most users should not use this function directly but should use lmFit instead.

This function is for fitting gene-wise linear models when some of the expression values are correlated. The correlated groups may arise from replicate spots on the same array (duplicate spots) or from a biological or technical replicate grouping of the arrays. This function is normally called by lmFit and is not normally called directly by users.

Note that the correlation is assumed to be constant across genes. If correlation=NULL then a call is made to duplicateCorrelation to estimate the correlation. Parallel threads are used only when genewise native-code fits are required. Complete-data fast paths are unchanged. Builds without OpenMP run serially.

Value

A list with components

coefficients	numeric matrix containing the estimated coefficients for each linear model. Same number of rows as M, same number of columns as design.

stdev.unscaled	numeric matrix conformal with coef containing the unscaled standard deviations for the coefficient estimators. The standard errors are given by stdev.unscaled * sigma.
sigma	numeric vector containing the residual standard deviation for each gene.
df.residual	numeric vector giving the degrees of freedom corresponding to sigma
correlation	inter-duplicate or inter-block correlation
qr	QR decomposition of the generalized linear squares problem, i.e., the decomposition of design standardized by the Choleski-root of the correlation matrix defined by correlation

Author(s)

Gordon Smyth

See Also

duplicateCorrelation.

An overview of linear model functions in limma is given by 06.LinearModels.

---

goana *Gene Ontology or KEGG Pathway Analysis*

Description

Test for over-representation of gene ontology (GO) terms or KEGG pathways in one or more sets of genes, optionally adjusting for abundance or gene length bias.

Usage

```
## S3 method for class 'MArrayLM'
goana(de, coef = ncol(de), geneid = rownames(de), FDR = 0.05, trend = FALSE, ...)
## S3 method for class 'MArrayLM'
kegga(de, coef = ncol(de), geneid = rownames(de), FDR = 0.05, trend = FALSE, ...)
## Default S3 method:
goana(de, universe = NULL, species = "Hs", null.prob = NULL, covariate=NULL,
      plot=FALSE, ...)
## Default S3 method:
kegga(de, universe = NULL, restrict.universe = FALSE, species = "Hs", species.KEGG = NULL,
      convert = FALSE, gene.pathway = NULL, pathway.names = NULL,
      null.prob = NULL, covariate=NULL, plot=FALSE, ...)
getGeneKEGGLinks(species.KEGG = "hsa", convert = FALSE)
getKEGGPathwayNames(species.KEGG = NULL, remove.qualifier = FALSE)
```

Arguments

de	a character vector of Entrez Gene IDs, or a list of such vectors, or an MArrayLM fit object.
coef	column number or column name specifying for which coefficient or contrast differential expression should be assessed.
geneid	Entrez Gene identifiers. Either a vector of length nrow(de) or the name of the column of de$genes containing the Entrez Gene IDs.
FDR	false discovery rate cutoff for differentially expressed genes. Numeric value between 0 and 1.
species	character string specifying the species. Possible values include "Hs" (human), "Mm" (mouse), "Rn" (rat), "Dm" (fly) or "Pt" (chimpanzee), but other values are possible if the corresponding organism package is available. See alias2Symbol for other possible values. Ignored if species.KEGG is not NULL or if gene.pathway and pathway.names are not NULL.
species.KEGG	three-letter KEGG species identifier. See https://www.kegg.jp/kegg/catalog/org_list.html or https://rest.kegg.jp/list/organism for possible values. Alternatively, if de contains KEGG ortholog Ids ("k00001" etc) instead of gene Ids, then set species.KEGG="ko". This argument is ignored if gene.pathway and pathway.names are both not NULL.
convert	if TRUE then KEGG gene identifiers will be converted to NCBI Entrez Gene identifiers. Note that KEGG IDs are the same as Entrez Gene IDs for most species anyway.
gene.pathway	data.frame linking genes to pathways. First column gives gene IDs, second column gives pathway IDs. By default this is obtained automatically by getGeneKEGGLinks(species.KEGG)
remove.qualifier	if TRUE, the species qualifier will be removed from the pathway names.
pathway.names	data.frame giving full names of pathways. First column gives pathway IDs, second column gives pathway names. By default this is obtained automatically using getKEGGPathwayNames(species.KEGG, remove=TRUE).
trend	adjust analysis for gene length or abundance? Can be logical, or a numeric vector of covariate values, or the name of the column of de$genes containing the covariate values. If TRUE, then de$Amean is used as the covariate.
universe	vector specifying the set of Entrez Gene identifiers to be the background universe. If NULL then all Entrez Gene IDs associated with any gene ontology term will be used as the universe.
restrict.universe	logical, should the universe be restricted to gene identifiers found in at least one pathway in gene.pathway?
null.prob	optional numeric vector of the same length as universe giving the null probability that each gene in the universe will appear in a gene set without enrichment. Will be computed from covariate if the latter is provided. Ignored if universe is NULL.
covariate	optional numeric vector of the same length as universe giving a covariate against which null.prob should be computed. Ignored if universe is NULL.

plot	logical, should the null.prob vs covariate trend be plotted?
...	any other arguments in a call to the MArrayLM methods are passed to the corresponding default method.

Details

These functions perform over-representation analyses for Gene Ontology terms or KEGG pathways. The default methods accept a gene set as a vector of Entrez Gene IDs or multiple gene sets as a list of such vectors. An over-represention analysis is then done for each set. The MArrayLM method extracts the gene sets automatically from a linear model fit object.

The p-values returned by goana and kegga are unadjusted for multiple testing. The authors have chosen not to correct automatically for multiple testing because GO terms and KEGG pathways are often overlapping, so standard methods of p-value adjustment may be very conservative. Users should be aware though that p-values are unadjusted, meaning that only very small p-values should be used for published results.

goana uses annotation from the appropriate Bioconductor organism package. The species can be any character string XX for which an organism package org.XX.eg.db is installed. Examples are "Hs" for human or "Mm" for mouse. See alias2Symbol for other possible values for species.

kegga reads KEGG pathway annotation from the KEGG website. For kegga, the species name can be provided in either Bioconductor or KEGG format. Examples of KEGG format are "hsa" for human, "mmu" for mouse or "dme" for fly. kegga can be used for any species supported by KEGG, of which there are more than 14,000 possibilities. By default, kegga obtains the KEGG annotation for the specified species from the https://rest.kegg.jp website using getGeneKEGGLinks and getKEGGPathwayNames. Alternatively one can supply the required pathway annotation to kegga in the form of two data.frames. If this is done, then an internet connection is not required.

The gene ID system used by kegga for each species is determined by KEGG. For human and mouse, the default (and only choice) is Entrez Gene ID. For Drosophila, the default is FlyBase CG annotation symbol. The format of the IDs can be seen by typing head(getGeneKEGGLinks(species)), for examplehead(getGeneKEGGLinks("hsa")) or head(getGeneKEGGLinks("dme")). Entrez Gene IDs can always be used. If Entrez Gene IDs are not the default, then conversion can be done by specifying "convert=TRUE".

Another possibility is to use KEGG orthology IDs as the gene IDs, and these can be used for any species. In that case, set species.KEGG="ko".

The ability to supply data.frame annotation to kegga means that kegga can in principle be used in conjunction with any user-supplied set of annotation terms.

The default goana and kegga methods accept a vector null.prob giving the prior probability that each gene in the universe appears in a gene set. This vector can be used to correct for unwanted trends in the differential expression analysis associated with gene length, gene abundance or any other covariate (Young et al, 2010). The MArrayLM object computes the null.prob vector automatically when trend is non-NULL.

If null.prob=NULL, the function computes one-sided hypergeometric tests equivalent to Fisher's exact test. If prior probabilities are specified, then a test based on the Wallenius' noncentral hypergeometric distribution is used to adjust for the relative probability that each gene will appear in a gene set, following the approach of Young et al (2010).

The MArrayLM methods performs over-representation analyses for the up and down differentially expressed genes from a linear model analysis. In this case, the universe is all the genes found in the fit object.

trend=FALSE is equivalent to null.prob=NULL. If trend=TRUE or a covariate is supplied, then a trend is fitted to the differential expression results and this is used to set null.prob.

The statistical approach provided here is the same as that provided by the goseq package, with one methodological difference and a few restrictions. Unlike the goseq package, the gene identifiers here must be Entrez Gene IDs and the user is assumed to be able to supply gene lengths if necessary. The goseq package has additional functionality to convert gene identifiers and to provide gene lengths. The only methodological difference is that goana and kegga computes gene length or abundance bias using tricubeMovingAverage instead of monotonic regression. While tricubeMovingAverage does not enforce monotonicity, it has the advantage of numerical stability when de contains only a small number of genes. The trend is estimated by the goanaTrend function.

Value

The goana default method produces a data frame with a row for each GO term and the following columns:

Term	GO term.
Ont	ontology that the GO term belongs to. Possible values are "BP", "CC" and "MF".
N	number of genes in the GO term.
DE	number of genes in the DE set.
P.DE	p-value for over-representation of the GO term in the set.

The last two column names above assume one gene set with the name DE. In general, there will be a pair of such columns for each gene set and the name of the set will appear in place of "DE".

The goana method for MArrayLM objects produces a data frame with a row for each GO term and the following columns:

Term	GO term.
Ont	ontology that the GO term belongs to. Possible values are "BP", "CC" and "MF".
N	number of genes in the GO term.
Up	number of up-regulated differentially expressed genes.
Down	number of down-regulated differentially expressed genes.
P.Up	p-value for over-representation of GO term in up-regulated genes.
P.Down	p-value for over-representation of GO term in down-regulated genes.

The row names of the data frame give the GO term IDs.

The output from kegga is the same except that row names become KEGG pathway IDs, Term becomes Pathway and there is no Ont column.

Note

kegga requires an internet connection unless gene.pathway and pathway.names are both supplied.

The default for kegga with species="Dm" changed from convert=TRUE to convert=FALSE in limma 3.27.8. Users wanting to use Entrez Gene IDs for Drosophila should set convert=TRUE, otherwise FlyBase CG annotation symbol IDs are assumed (for example "Dme1_CG4637").

The default for restrict.universe=TRUE in kegga changed from TRUE to FALSE in limma 3.33.4.

Bug fix: results from kegga with trend=TRUE or with non-NULL covariate were incorrect prior to limma 3.32.3. The results were biased towards significant Down p-values and against significant Up p-values.

Author(s)

Gordon Smyth and Yifang Hu

References

Young MD, Wakefield MJ, Smyth GK, Oshlack A (2010). Gene ontology analysis for RNA-seq: accounting for selection bias. *Genome Biology* 11, R14. doi:10.1186/gb2010112r14

See Also

topGO, topKEGG, goanaTrend

The goseq package provides an alternative implementation of methods from Young et al (2010). Unlike the limma functions documented here, goseq will work with a variety of gene identifiers and includes a database of gene length information for various species.

The gostats package also does GO analyses without adjustment for bias but with some other options.

See 10.GeneSetTests for a description of other functions used for gene set testing.

Examples

```
## Not run:
## Linear model usage:

fit <- lmFit(y, design)
fit <- eBayes(fit)

# Standard GO analysis

go.fisher <- goana(fit, species="Hs")
topGO(go.fisher, sort = "up")
topGO(go.fisher, sort = "down")

# GO analysis adjusting for gene abundance

go.abund <- goana(fit, geneid = "GeneID", trend = TRUE)
topGO(go.abund, sort = "up")
topGO(go.abund, sort = "down")
```

```
# GO analysis adjusting for gene length bias
# (assuming that y$genes$Length contains gene lengths)

go.len <- goana(fit, geneid = "GeneID", trend = "Length")
topGO(go.len, sort = "up")
topGO(go.len, sort = "down")

## Default usage with a list of gene sets:

go.de <- goana(list(DE1 = EG.DE1, DE2 = EG.DE2, DE3 = EG.DE3))
topGO(go.de, sort = "DE1")
topGO(go.de, sort = "DE2")
topGO(go.de, ontology = "BP", sort = "DE3")
topGO(go.de, ontology = "CC", sort = "DE3")
topGO(go.de, ontology = "MF", sort = "DE3")

## Standard KEGG analysis

k <- kegga(fit, species="Hs")
k <- kegga(fit, species.KEGG="hsa") # equivalent to previous
topKEGG(k, sort = "up")
topKEGG(k, sort = "down")


## End(Not run)
```

---

goanaTrend *Estimate DE Trend for Gene Ontology or KEGG Pathway Analysis*

Description

Given a list of differentially expressed (DE) genes and a covariate, estimate the probability of a gene being called significant as a function of the covariate. This function is typically used to estimate the gene length or gene abundance bias for a pathway analysis.

Usage

```
goanaTrend(index.de, covariate, n.prior = 10, plot = FALSE,
           xlab = "Covariate Rank", ylab = "Probability gene is DE",
           main="DE status vs covariate")
```

Arguments

index.de	an index vector specifying which genes are significantly DE. Can be a vector of integer indices, or a logical vector of length nrow(covariate), or any vector such as covariate[index] selects the DE genes.
covariate	numeric vector, length equal to the number of genes in the analysis. Usually equal to gene length or average log-expression but can be any meaningful gene-newise covariate.

n.prior	prior number of genes using for moderating the trend towards constancy, for stability when the number of DE genes is small.
plot	if TRUE, plot the estimated trend.
xlab	label for x-axis of plot.
ylab	label for y-axis of plot.
main	main title for the plot.

Details

goanaTrend is called by goana and kegga when the trend argument is used to correct for unwanted trends in the differential expression analysis associated with gene length, gene abundance or any other covariate (Young et al, 2010).

This function is analogous to the nullp function of the goseq package but the trend is estimated using tricubeMovingAverage instead of by monotonic regression. While tricubeMovingAverage does not enforce strict monotonicity, it has the advantage of numerical stability and statistical robustness when there are only a small number of DE genes.

This function also moderates the estimated trend slightly towards constancy to provide more stability. The degree of moderation is determined by the n.prior argument relative to the number of DE genes.

Value

Numeric vector of same length as covariate giving estimated probabilities.

Author(s)

Gordon Smyth and Yifang Hu

References

Young MD, Wakefield MJ, Smyth GK, Oshlack A (2010). Gene ontology analysis for RNA-seq: accounting for selection bias. *Genome Biology* 11, R14. doi:10.1186/gb2010112r14

See Also

goana, kegga

See 10.GeneSetTests for a description of other functions used for gene set testing.

Examples

```
x <- runif(100)
i <- 1:10
goanaTrend(i, x, plot=TRUE)
```

gridr *Row and Column Positions on Microarray*

Description

Grid and spot row and column positions.

Usage

```
gridr(layout)
gridc(layout)
spotr(layout)
spotc(layout)
```

Arguments

layout	list with the components ngrid.r, ngrid.c, nspot.r and nspot.c

Value

Vector of length prod(unlist(layout)) giving the grid rows (gridr), grid columns (gridc), spot rows (spotr) or spot columns (spotc).

Author(s)

Gordon Smyth

head *Return the First to Last Part of a Data Object*

Description

Retrieve the first or last parts of an RGList, MAList, EListRaw, EList, MArrayLM or TestResults object.

Usage

```
## S3 method for class 'EList'
head(x, n = 6L, ...)
## S3 method for class 'EList'
tail(x, n = 6L, ...)
```

Arguments

x	an object of class RGList, MAList, EListRaw, EList, MArrayLM or TestResults.
n	a single integer. If positive or zero, number of rows of resulting object. If negative, all but the n last/first rows of x.
...	other arguments are not currently used.

Details

head (tail) returns the first (last) n rows when n >= 0 or all but the last (first) n rows when n < 0.

Value

An object like x but generally with fewer rows.

Author(s)

Gordon Smyth

See Also

head in the utils package.

02.Classes gives an overview of data classes used in LIMMA.

Examples

```
E <- matrix(rnorm(40),20,2)
rownames(E) <- paste0("Gene",1:20)
colnames(E) <- c("A","B")
y <- new("EList",list(E=E))
head(y)
tail(y)
```

---

heatdiagram *Stemmed Heat Diagram*

Description

Creates a heat diagram showing the co-regulation of genes under one condition with a range of other conditions.

Usage

```
heatDiagram(results, coef, primary=1, names=NULL, treatments=colnames(coef), limit=NULL,
            orientation="landscape", low="green", high="red", cex=1, mar=NULL,
            ncolors=123, ...)
heatdiagram(stat, coef, primary=1, names=NULL, treatments=colnames(stat),
        critical.primary=4, critical.other=3, limit=NULL, orientation="landscape",
            low="green", high="red", cex=1, mar=NULL, ncolors=123, ...)
```

Arguments

results	TestResults matrix, containing elements -1, 0 or 1, from decideTests
stat	numeric matrix of test statistics. Rows correspond to genes and columns to treatments or contrasts between treatments.
coef	numeric matrix of the same size as stat. Holds the coefficients to be displayed in the plot.
primary	number or name of the column to be compared to the others. Genes are included in the diagram according to this column of stat and are sorted according to this column of coef. If primary is a name, then stat and coef must have the same column names.
names	optional character vector of gene names
treatments	optional character vector of treatment names
critical.primary	critical value above which the test statistics for the primary column are considered significant and included in the plot
critical.other	critical value above which the other test statistics are considered significant. Should usually be no larger than critical.primary although larger values are permitted.
limit	optional value for coef above which values will be plotted in extreme color. Defaults to max(abs(coef)).
orientation	"portrait" for upright plot or "landscape" for plot oriented to be wider than high. "portrait" is likely to be appropriate for inclusion in a printed document while "landscape" may be appropriate for a presentation on a computer screen.
low	color associated with repressed gene regulation
high	color associated with induced gene regulation
ncolors	number of distinct colors used for each of up and down regulation
cex	factor to increase or decrease size of column and row text
mar	numeric vector of length four giving the size of the margin widths. Default is cex*c(5,6,1,1) for landscape and cex*c(1,1,4,3) for portrait.
...	any other arguments will be passed to the image function

Details

Users are encouraged to use heatDiagram rather than heatdiagram as the latter function may be removed in future versions of limma.

This function plots an image of gene expression profiles in which rows (or columns for portrait orientation) correspond to treatment conditions and columns (or rows) correspond to genes. Only genes which are significantly differentially expressed in the primary condition are included. Genes are sorted by differential expression under the primary condition.

Note: the plot produced by this function is unique to the limma package. It should not be confused with "heatmaps" often used to display results from cluster analyses.

Value

An image is created on the current graphics device. A matrix with named rows containing the coefficients used in the plot is also invisibly returned.

Author(s)

Gordon Smyth

See Also

image.

Examples

```
## Not run:
MA <- normalizeWithinArrays(RG)
design <- cbind(c(1,1,1,0,0,0),c(0,0,0,1,1,1))
fit <- lmFit(MA,design=design)
contrasts.mouse <- cbind(Control=c(1,0),Mutant=c(0,1),Difference=c(-1,1))
fit <- eBayes(contrasts.fit(fit,contrasts=contrasts.mouse))
results <- decideTests(fit,method="global",p=0.1)
heatDiagram(results,fit$coef,primary="Difference")

## End(Not run)
```

---

helpMethods *Prompt for Method Help Topics*

Description

For any S4 generic function, find all methods defined in currently loaded packages. Prompt the user to choose one of these to display the help document.

Usage

```
helpMethods(genericFunction)
```

Arguments

genericFunction	a generic function or a character string giving the name of a generic function

Author(s)

Gordon Smyth

See Also

showMethods

Examples

```
## Not run: helpMethods(show)
```

`ids2indices` *Convert Gene Identifiers to Indices for Gene Sets*

Description

Make a list of gene identifiers into a list of indices for gene sets.

Usage

```
ids2indices(gene.sets, identifiers, remove.empty=TRUE)
```

Arguments

`gene.sets`	list of character vectors, each vector containing the gene identifiers for a set of genes.
`identifiers`	character vector of gene identifiers.
`remove.empty`	logical, should sets of size zero be removed from the output?

Details

This function is used to create input for the `romer`, `mroast` and `camera` functions. Typically, `identifiers` is the vector of Entrez Gene IDs, and `gene.sets` is obtained from a database of gene sets, for example a representation of the Molecular Signatures Database (MSigDB) downloaded from https://bioinf.wehi.edu.au/software/MSigDB/.

Value

list of integer vectors, each vector containing the indices of a gene set in the vector `identifiers`.

Author(s)

Gordon Smyth and Yifang Hu

See Also

`romer`, `mroast`, `camera`

There is a topic page on 10.GeneSetTests.

Examples

```
## Not run:

download.file("https://bioinf.wehi.edu.au/software/MSigDB/human_c2_v5p2.rdata",
       "human_c2_v5p2.rdata", mode = "wb")

load("human_c2_v5p2.rdata")
c2.indices <- ids2indices(Hs.c2, y$genes$GeneID)
camera(y, c2.indices, design)


## End(Not run)
```

`imageplot` *Image Plot of Microarray Statistics*

Description

Creates an image of colors or shades of gray that represent the values of a statistic for each spot on a spotted microarray. This function can be used to explore any spatial effects across the microarray.

Usage

```
imageplot(z, layout, low = NULL, high = NULL, ncolors = 123, zerocenter = NULL,
zlim = NULL, mar=c(2,1,1,1), legend=TRUE, ...)
```

Arguments

`z`	numeric vector or array. This vector can contain any spot statistics, such as log intensity ratios, spot sizes or shapes, or t-statistics. Missing values are allowed and will result in blank spots on the image. Infinite values are not allowed.
`layout`	a list specifying the dimensions of the spot matrix and the grid matrix.
`low`	color associated with low values of `z`. May be specified as a character string such as `"green"`, `"white"` etc, or as a rgb vector in which `c(1,0,0)` is red, `c(0,1,0)` is green and `c(0,0,1)` is blue. The default value is `"green"` if `zerocenter=T` or `"white"` if `zerocenter=F`.
`high`	color associated with high values of `z`. The default value is `"red"` if `zerocenter=T` or `"blue"` if `zerocenter=F`.
`ncolors`	number of color shades used in the image including low and high.
`zerocenter`	should zero values of `z` correspond to a shade exactly halfway between the colors low and high? The default is TRUE if `z` takes positive and negative values, otherwise FALSE.
`zlim`	numerical vector of length 2 giving the extreme values of `z` to associate with colors `low` and `high`. By default `zlim` is the range of `z`. Any values of `z` outside the interval `zlim` will be truncated to the relevant limit.

mar	numeric vector of length 4 specifying the width of the margin around the plot. This argument is passed to par.
legend	logical, if TRUE the range of z and zlim is shown in the bottom margin
...	any other arguments will be passed to the function image

Details

This function may be used to plot the values of any spot-specific statistic, such as the log intensity ratio, background intensity or a quality measure such as spot size or shape. The image follows the layout of an actual microarray slide with the bottom left corner representing the spot (1,1,1,1). The color range is used to represent the range of values for the statistic. When this function is used to plot the red/green log-ratios, it is intended to be an in silico version of the classic false-colored red-yellow-green image of a scanned two-color microarray.

This function is related to the earlier plot.spatial function in the sma package and to the later maImage function in the marray package. It differs from plot.spatial most noticeably in that all the spots are plotted and the image is plotted from bottom left rather than from top left. It is intended to display spatial patterns and artifacts rather than to highlight only the extreme values as does plot.spatial. It differs from maImage in that any statistic may be plotted and in its use of a red-yellow-green color scheme for log-ratios, similar to the classic false-colored jpeg image, rather than the red-black-green color scheme associated with heat maps.

Value

A plot is created on the current graphics device.

Author(s)

Gordon Smyth

See Also

maImage in the marray package, image in the graphics package.

An overview of diagnostic functions available in LIMMA is given in 09.Diagnostics.

Examples

```
M <- rnorm(8*4*16*16)
imageplot(M,layout=list(ngrid.r=8,ngrid.c=4,nspot.r=16,nspot.c=16))
```

---

imageplot3by2 *Write Imageplots to Files*

Description

Write imageplots to files in PNG format, six plots to a file in a 3 by 2 grid arrangement.

Usage

```
imageplot3by2(RG, z="Gb", prefix=paste("image",z,sep="-"), path=NULL,
              zlim=NULL, common.lim=TRUE, ...)
```

Arguments

RG	an RGList or MAList object, or any list with component named by z
z	character string giving name of component of RG to plot
prefix	character string giving prefix to attach to file names
path	character string specifying directory for output files
zlim	numeric vector of length 2, giving limits of response vector to be associated with saturated colors
common.lim	logical, should all plots on a page use the same axis limits
...	any other arguments are passed to imageplot

Details

At the time of writing, this function writes plots in PNG format in an arrangement optimized for A4-sized paper.

Value

No value is returned, but one or more files are written to the working directory. The number of files is determined by the number of columns of RG.

Author(s)

Gordon Smyth

See Also

An overview of diagnostic functions available in LIMMA is given in 09.Diagnostics.

---

intraspotCorrelation *Intra-Spot Correlation for Two Color Data*

Description

Estimate the within-block correlation associated with spots for spotted two color microarray data.

Usage

```
intraspotCorrelation(object, design, trim=0.15)
```

Arguments

object	an MAList object or a list from which M and A values may be extracted
design	a numeric matrix containing the design matrix for linear model in terms of the individual channels. The number of rows should be twice the number of arrays. The number of columns will determine the number of coefficients estimated for each gene.
trim	the fraction of observations to be trimmed from each end of the atanh-correlations when computing the consensus correlation. See mean.

Details

This function estimates the correlation between two channels observed on each spot. The correlation is estimated by fitting a heteroscedastic regression model to the M and A-values of each gene. The function also returns a consensus correlation, which is a robust average of the individual correlations, which can be used as input for the function lmscFit.

The function may take a long time to execute.

Value

A list with components

consensus.correlation	robust average of the estimated inter-duplicate correlations. The average is the trimmed mean of the correlations for individual genes on the atanh-transformed scale.
atanh.correlations	a numeric vector giving the individual genewise correlations on the atanh scale
df	numeric matrix of degrees of freedom associated with the correlations. The first column gives the degrees of freedom for estimating the within-spot or M-value mean square while the second gives the degrees of freedom for estimating the between spot or A-value mean square.

Author(s)

Gordon Smyth

References

Smyth GK (2005). Individual channel analysis of two-colour microarray data. *Proceedings of the 55th Session of the International Statistics Institute*, 5-12 April 2005, Sydney, Australia, Paper 116. https://gksmyth.github.io/pubs/ISI2005-116.pdf

See Also

This function uses remlscore from the statmod package.

An overview of methods for single channel analysis in limma is given by 07.SingleChannel.

Examples

```
#  See lmscFit
## Not run:
corfit <- intraspotCorrelation(MA, design)
all.correlations <- tanh(corfit$atanh.correlations)
boxplot(all.correlations)

## End(Not run)
```

`is.fullrank` *Check for Full Column Rank*

Description

Test whether a numeric matrix has full column rank.

Usage

```
is.fullrank(x)
nonEstimable(x)
```

Arguments

x	a numeric matrix or vector

Details

`is.fullrank` is used to check the integrity of design matrices in limma, for example after subsetting operations.

`nonEstimable` is used by `lmFit` to report which coefficients in a linear model cannot be estimated.

Value

`is.fullrank` returns `TRUE` or `FALSE`.

`nonEstimable` returns a character vector of names for the columns of `x` which are linearly dependent on previous columns. If `x` has full column rank, then the value is `NULL`.

Author(s)

Gordon Smyth

Examples

```
# TRUE
is.fullrank(1)
is.fullrank(cbind(1,0:1))

# FALSE
is.fullrank(0)
is.fullrank(matrix(1,2,2))
nonEstimable(matrix(1,2,2))
```

---

isNumeric *Test for Numeric Argument*

Description

Test whether argument is numeric or a data.frame with numeric columns.

Usage

```
isNumeric(x)
```

Arguments

x	any object

Details

This function is used to check the validity of arguments for numeric functions. It is an attempt to emulate the behavior of internal generic math functions.

`isNumeric` differs from `is.numeric` in that data.frames with all columns numeric are accepted as numeric.

Value

`TRUE` or `FALSE`

Author(s)

Gordon Smyth

See Also

`is.numeric`, `Math`

Examples

```
isNumeric(3)
isNumeric("a")
x <- data.frame(a=c(1,1),b=c(0,1))
isNumeric(x)   # TRUE
is.numeric(x)  # FALSE
```

---

kooperberg *Kooperberg Model-Based Background Correction for GenePix data*

Description

This function uses a Bayesian model to background correct GenePix microarray data.

Usage

```
kooperberg(RG, a = TRUE, layout = RG$printer, verbose = TRUE)
```

Arguments

RG	an RGList of GenePix data, read in using read.maimages, with other.columns=c("F635 SD","B635 SD","F532 SD","B532 SD","B532 Mean","B635 Mean","F Pixels","B Pixels").
a	logical. If TRUE, the 'a' parameters in the model (equations 3 and 4) are estimated for each slide. If FALSE the 'a' parameters are set to unity.
layout	list containing print layout with components ngrid.r, ngrid.c, nspot.r and nspot.c. Defaults to RG$printer.
verbose	logical. If TRUE, progress is reported to standard output.

Details

This function is for use with GenePix data and is designed to cope with the problem of large numbers of negative intensities and hence missing values on the log-intensity scale. It avoids missing values in most cases and at the same time dampens down the variability of log-ratios for low intensity spots. See Kooperberg et al (2002) for more details.

kooperberg uses the foreground and background intensities, standard deviations and number of pixels to compute empirical estimates of the model parameters as described in equation 2 of Kooperberg et al (2002).

Value

An RGList containing the components

R	matrix containing the background adjusted intensities for the red channel for each spot for each array
G	matrix containing the background adjusted intensities for the green channel for each spot for each array
printer	list containing print layout

Author(s)

Matthew Ritchie

References

Kooperberg C, Fazzio TG, Delrow JJ, Tsukiyama T (2002) Improved background correction for spotted DNA microarrays. *Journal of Computational Biology* 9, 55-66.

Ritchie ME, Silver J, Oshlack A, Holmes M, Diyagama D, Holloway A, Smyth GK (2007). A comparison of background correction methods for two-colour microarrays. *Bioinformatics* 23, 2700-2707. doi:10.1093/bioinformatics/btm412.

See Also

04.Background gives an overview of background correction functions defined in the LIMMA package.

Examples

```
#  This is example code for reading and background correcting GenePix data
#  given GenePix Results (gpr) files in the working directory (data not
#  provided).
## Not run:
# get the names of the GenePix image analysis output files in the current directory
genepixFiles <- dir(pattern="*\\.gpr$")
RG <- read.maimages(genepixFiles, source="genepix", other.columns=c("F635 SD","B635 SD",
                    "F532 SD","B532 SD","B532 Mean","B635 Mean","F Pixels","B Pixels"))
RGmodel <- kooperberg(RG)
MA <- normalizeWithinArrays(RGmodel)

## End(Not run)
```

LargeDataObject-class *Large Data Object - class*

Description

A virtual class including the data classes `RGList`, `MAList` and `MArrayLM`, all of which typically contain large quantities of numerical data in vector, matrices and data.frames.

Methods

A `show` method is defined for objects of class `LargeDataObject` which uses `printHead` to print only the leading elements or rows of components or slots which contain large quantities of data.

Author(s)

Gordon Smyth

See Also

02.Classes gives an overview of all the classes defined by this package.

Examples

```
#  see normalizeBetweenArrays
```

---

limmaUsersGuide *View Limma User's Guide*

Description

Finds the location of the Limma User's Guide and optionally opens it.

Usage

```
limmaUsersGuide(view=TRUE)
```

Arguments

view	logical, should the document be opened using the default PDF document reader?

Details

The function `vignette("limma")` will find the short limma Vignette which describes how to obtain the Limma User's Guide. The User's Guide is not itself a true vignette because it is not automatically generated using Sweave during the package build process. This means that it cannot be found using `vignette`, hence the need for this special function.

If the operating system is other than Windows, then the PDF viewer used is that given by `Sys.getenv("R_PDFVIEWER")`. The PDF viewer can be changed using `Sys.putenv(R_PDFVIEWER=)`.

This function is used by the drop-down Vignettes menu when the Rgui interface for Windows is used.

Value

Character string giving the file location.

Author(s)

Gordon Smyth

See Also

vignette, openPDF, openVignette, Sys.getenv, Sys.putenv

Examples

```
limmaUsersGuide(view=FALSE)
```

lm.series *Fit Linear Model to Microarray Data by Ordinary Least Squares*

Description

Fit a linear model genewise to expression data from a series of arrays. This function uses ordinary least squares and is a utility function for lmFit.

Usage

```
lm.series(M,design=NULL,ndups=1,spacing=1,weights=NULL,nthreads=1L,contrasts=NULL)
```

Arguments

M	numeric matrix containing log-ratio or log-expression values for a series of microarrays, rows correspond to genes and columns to arrays
design	numeric design matrix defining the linear model. The number of rows should agree with the number of columns of M. The number of columns will determine the number of coefficients estimated for each gene.
ndups	number of duplicate spots. Each gene is printed ndups times in adjacent spots on each array.
spacing	the spacing between the rows of M corresponding to duplicate spots, spacing=1 for consecutive spots
weights	an optional numeric matrix of the same dimension as M containing weights for each spot. If it is of different dimension to M, it will be filled out to the same size.
nthreads	positive integer giving the number of OpenMP threads used for genewise native-code fits. Defaults to 1.
contrasts	optional numeric contrast matrix with rows corresponding to coefficients. If non-NULL, a contrast-space fit is returned. On the genewise (weighted or missing-value) path the contrasts are applied within each gene's fit, giving exact contrast stdev.unscaled; on the no-weights fast path the exact contrasts.fit is applied (all genes share the same unscaled covariance).

Details

This is a utility function used by the higher level function lmFit. Most users should not use this function directly but should use lmFit instead.

The linear model is fit for each gene by calling the function lm.fit or lm.wfit from the base library. Parallel threads are used only when genewise native-code fits are required. Complete-data fast paths are unchanged. Builds without OpenMP run serially.

Value

A list with components

coefficients	numeric matrix containing the estimated coefficients for each linear model. Same number of rows as M, same number of columns as design.
stdev.unscaled	numeric matrix conformal with coef containing the unscaled standard deviations for the coefficient estimators. The standard errors are given by stdev.unscaled * sigma.
sigma	numeric vector containing the residual standard deviation for each gene.
df.residual	numeric vector giving the degrees of freedom corresponding to sigma.
qr	QR-decomposition of design

Author(s)

Gordon Smyth

See Also

lm.fit.

An overview of linear model functions in limma is given by 06.LinearModels.

Examples

```
# See lmFit for examples
```

---

lmFit *Linear Model for Series of Arrays*

Description

Fit linear model for each gene given a series of arrays

Usage

```
lmFit(object, design = NULL, ndups = NULL, spacing = NULL, block = NULL, correlation,
      weights = NULL, method = "ls", nthreads = 1L, contrasts = NULL, ...)
```

Arguments

object	A matrix-like data object containing log-ratios or log-expression values for a series of arrays, with rows corresponding to genes and columns to samples. Any type of data object that can be processed by getEAWP is acceptable.
design	the design matrix of the microarray experiment, with rows corresponding to samples and columns to coefficients to be estimated. Defaults to object$design if that is non-NULL, otherwise to the unit vector meaning that all samples will be treated as replicates of a single treatment group.

ndups	positive integer giving the number of times each distinct probe is printed on each array.
spacing	positive integer giving the spacing between duplicate occurrences of the same probe, spacing=1 for consecutive rows.
block	vector or factor specifying a blocking variable on the arrays. Has length equal to the number of arrays. Must be NULL if ndups>2.
correlation	the inter-duplicate or inter-technical replicate correlation
weights	non-negative precision weights. Can be a numeric matrix of individual weights of same size as the object expression matrix, or a numeric vector of array weights with length equal to ncol of the expression matrix, or a numeric vector of gene weights with length equal to nrow of the expression matrix.
method	fitting method; "ls" for least squares or "robust" for robust regression. Note that the block and correlation arguments will be ignored if method="robust".
nthreads	positive integer giving the number of OpenMP threads used for genewise least-squares fits. Defaults to 1.
contrasts	optional numeric matrix with rows corresponding to coefficients (columns of design) and columns corresponding to contrasts. If non-NULL, the returned fit is in contrast space, equivalent to running contrasts.fit on the coefficient fit. When probe-wise weights or missing values are present, the contrast stdev.unscaled is computed exactly for each gene from its own weighted model, rather than the shared approximation used by contrasts.fit. Not supported when method="robust".
...	other optional arguments to be passed to lm.series, gls.series or mrlm

Details

This function fits multiple linear models by weighted or generalized least squares. It accepts data from an experiment involving a series of microarrays with the same set of probes. A linear model is fitted to the expression data for each probe. The expression data should be log-ratios for two-color array platforms or log-expression values for one-channel platforms. (To fit linear models to the individual channels of two-color array data, see lmscFit.) The coefficients of the fitted models describe the differences between the RNA sources hybridized to the arrays. The probe-wise fitted model results are stored in a compact form suitable for further processing by other functions in the limma package.

The function allows for missing values and accepts quantitative precision weights through the weights argument. It also supports two different correlation structures. If block is non-NULL, then different arrays are assumed to be correlated. If block is NULL and ndups is greater than one, then replicate spots on the same array are assumed to be correlated. It is not possible at this time to fit models with both a block structure and a duplicate-spot correlation structure simultaneously. If the argument block is used, then it is assumed that ndups=1.

If either correlation structure is used, with block non-NULL or ndups greater than 1, then a consensus correlation should be estimated before running lmFit and entered as the correlation argument. The correlation estimation is usually done by running duplicateCorrelation.

If object is a matrix then it should contain log-ratios or log-expression data with rows corresponding to probes and columns to arrays. (A numeric vector is treated the same as a matrix with one

column.) For objects of other classes, a matrix of expression values is taken from the appropriate component or slot of the object. If object is of class MAList or marrayNorm, then the matrix of log-ratios (M-values) is extracted. If object is of class ExpressionSet, then the expression matrix is extracted. (This may contain log-expression or log-ratio values, depending on the platform.) If object is of class PLMset then the matrix of chip coefficients chip.coefs is extracted.

The arguments design, ndups, spacing and weights will be extracted from the data object if available. On the other hand, if any of these are set to a non-NULL value in the function call, then the argument value will override the value found in object. If object is an PLMset, then weights are computed as 1/pmax(object@se.chip.coefs, 1e-05)^2. If object is an ExpressionSet object, then weights are not computed.

The actual linear model computations are done by passing the data to one of the lower-level functions lm.series, gls.series or mrlm. The function mrlm is used if method="robust". If method="ls", then gls.series is used if a correlation structure has been specified, i.e., if ndups>1 or block is non-NULL and correlation is different from zero. If method="ls" and there is no correlation structure, lm.series is used. If method="robust" then any correlation structure will be ignored.

If contrasts is non-NULL, the fit is returned in contrast space, so that lmFit(y, design, contrasts=C) is a one-step equivalent of contrasts.fit(lmFit(y, design), C). The returned object has the same format in both cases. The only difference arises when probe-wise weights or missing values are present: lmFit then refits each gene's weighted model and so reports the exact contrast standard deviations stdev.unscaled, whereas contrasts.fit applied afterwards uses a shared-correlation approximation.

Value

An MArrayLM object containing the result of the fits.

The rownames of object are preserved in the fit object and can be retrieved by rownames(fit) where fit is output from lmFit. The column names of design are preserved as column names and can be retrieved by colnames(fit).

Author(s)

Gordon Smyth

See Also

lmFit uses getEAWP to extract expression values, gene annotation and so on from the data object.

An overview of linear model functions in limma is given by 06.LinearModels.

Examples

```
# Simulate gene expression data for 100 probes and 6 microarrays
# Microarray are in two groups
# First two probes are differentially expressed in second group
# Std deviations vary between genes with prior df=4
sd <- 0.3*sqrt(4/rchisq(100,df=4))
y <- matrix(rnorm(100*6,sd=sd),100,6)
rownames(y) <- paste("Gene",1:100)
y[1:2,4:6] <- y[1:2,4:6] + 2
```

```
design <- cbind(Grp1=1,Grp2vs1=c(0,0,0,1,1,1))
options(digits=3)

# Ordinary fit
fit <- lmFit(y,design)
fit <- eBayes(fit)
topTable(fit,coef=2)
dim(fit)
colnames(fit)
rownames(fit)[1:10]
names(fit)

# Fold-change thresholding
fit2 <- treat(fit,lfc=0.1)
topTreat(fit2,coef=2)

# Fit a contrast directly (equivalent to contrasts.fit)
contrast.matrix <- makeContrasts(Grp2vs1, levels=design)
fitc <- lmFit(y,design,contrasts=contrast.matrix)
fitc <- eBayes(fitc)
topTable(fitc)

# Volcano plot
volcanoplot(fit,coef=2,highlight=2)

# Mean-difference plot
plotMD(fit,column=2)

# Q-Q plot of moderated t-statistics
qqt(fit$t[,2],df=fit$df.residual+fit$df.prior)
abline(0,1)

# Various ways of writing results to file
## Not run: write.fit(fit,file="exampleresults.txt")
## Not run: write.table(fit,file="exampleresults2.txt")

# Fit with correlated arrays
# Suppose each pair of arrays is a block
block <- c(1,1,2,2,3,3)
dupcor <- duplicateCorrelation(y,design,block=block)
dupcor$consensus.correlation
fit3 <- lmFit(y,design,block=block,correlation=dupcor$consensus)

# Fit with duplicate probes
# Suppose two side-by-side duplicates of each gene
rownames(y) <- paste("Gene",rep(1:50,each=2))
dupcor <- duplicateCorrelation(y,design,ndups=2)
dupcor$consensus.correlation
fit4 <- lmFit(y,design,ndups=2,correlation=dupcor$consensus)
dim(fit4)
fit4 <- eBayes(fit4)
topTable(fit4,coef=2)
```

lmscFit *Fit Linear Model to Individual Channels of Two-Color Data*

Description

Fit a linear model to the individual log-intensities for each gene given a series of two-color arrays

Usage

```
lmscFit(object, design, correlation)
```

Arguments

object	an MAList object or a list from which M and A values may be extracted
design	a numeric matrix containing the design matrix for linear model in terms of the individual channels. The number of rows should be twice the number of arrays. The number of columns will determine the number of coefficients estimated for each gene.
correlation	numeric value giving the intra-spot correlation

Details

For two color arrays, the channels measured on the same set of arrays are correlated. The M and A however are uncorrelated for each gene. This function fits a linear model to the set of M and A-values for each gene after re-scaling the M and A-values to have equal variances. The input correlation determines the scaling required. The input correlation is usually estimated using intraspotCorrelation before using lmscFit.

Missing values in M or A are not allowed.

Value

An object of class MArrayLM

Author(s)

Gordon Smyth

References

Smyth GK (2005). Individual channel analysis of two-colour microarray data. *Proceedings of the 55th Session of the International Statistics Institute*, 5-12 April 2005, Sydney, Australia; International Statistics Institute; Paper 116. https://gksmyth.github.io/pubs/ISI2005-116.pdf

Smyth GK, Altman NS (2013). Separate-channel analysis of two-channel microarrays: recovering inter-spot information. *BMC Bioinformatics* 14, 165. doi:10.1186/1471210514165.

See Also

lm.fit.

An overview of methods for single channel analysis in limma is given by 07.SingleChannel.

Examples

```
## Not run:
# Subset of data from ApoAI case study in Limma User's Guide
# Avoid non-positive intensities
RG <- backgroundCorrect(RG,method="normexp")
MA <- normalizeWithinArrays(RG)
MA <- normalizeBetweenArrays(MA,method="Aq")
targets <- data.frame(Cy3=I(rep("Pool",6)),Cy5=I(c("WT","WT","WT","KO","KO","KO")))
targets.sc <- targetsA2C(targets)
targets.sc$Target <- factor(targets.sc$Target,levels=c("Pool","WT","KO"))
design <- model.matrix(~Target,data=targets.sc)
corfit <- intraspotCorrelation(MA,design)
fit <- lmscFit(MA,design,correlation=corfit$consensus)
cont.matrix <- cbind(KOvsWT=c(0,-1,1))
fit2 <- contrasts.fit(fit,cont.matrix)
fit2 <- eBayes(fit2)
topTable(fit2,adjust="fdr")

## End(Not run)
```

---

loessFit *Univariate Lowess With Prior Weights*

Description

Univariate locally weighted linear regression allowing for prior weights. Returns fitted values and residuals.

Usage

```
loessFit(y, x, weights=NULL, span=0.3, iterations=4L, min.weight=1e-5, max.weight=1e5,
         equal.weights.as.null=TRUE, method="weightedLowess")
```

Arguments

y	numeric vector of response values. Missing values are allowed.
x	numeric vector of predictor values. Missing values are allowed.
weights	numeric vector of non-negative prior weights. Missing values are treated as zero.
span	positive numeric value between 0 and 1 specifying proportion of data to be used in the local regression moving window. Larger numbers give smoother fits.
iterations	number of local regression fits. Values greater than 1 produce robust fits.

`min.weight` minimum weight. Any lower weights will be reset.

`max.weight` maximum weight. Any higher weights will be reset.

`equal.weights.as.null`
should equal weights be treated as if weights were `NULL`, so that `lowess` is called? Applies even if all weights are zero.

`method` method used for weighted lowess. Possibilities are `"weightedLowess"`, `"loess"` or `"locfit"`.

Details

This function is essentially a wrapper function for `lowess` and `weightedLowess` with added error checking. The idea is to provide the classic univariate lowess algorithm of Cleveland (1979) but allowing for prior weights and missing values.

The venerable `lowess` code is fast, uses little memory and has an accurate interpolation scheme, so it is an advantage to use it when prior weights are not needed. This function calls `lowess` when `weights=NULL`, but returns values in original rather than sorted order and allows missing values. The treatment of missing values is analogous to `na.exclude`.

By default, `weights` that are all equal (even all zero) are treated as if they were `NULL`, so `lowess` is called in this case as well.

When unequal `weights` are provided, this function calls `weightedLowess` by default, although two other possibilities are also provided. `weightedLowess` implements a similar algorithm to `lowess` except that it uses the prior weights both in the local regressions and in determining which other observations to include in the local neighborhood of each observation.

Two alternative algorithms for weighted lowess curve fitting are provided as options. If `method="loess"`, then a call is made to `loess(y~x,weights=weights,span=span,degree=1,family="symmetric",...)`. This method differs from `weightedLowess` in that the prior weights are ignored when determining the neighbourhood of each observation.

If `method="locfit"`, then repeated calls are made to `locfit:::locfit.raw` with `deg=1`. In principle, this is similar to `"loess"`, but `"locfit"` makes some approximations and is very much faster and uses much less memory than `"loess"` for long data vectors.

The arguments `span` and `iterations` here have the same meaning as for `weightedLowess` and `loess`. `span` is equivalent to the argument `f` of `lowess` while `iterations` is equivalent to `iter+1` for `lowess`. It gives the total number of fits rather than the number of robustifying fits.

When there are insufficient observations to estimate the loess curve, `loessFit` returns a linear regression fit. This mimics the behavior of `lowess` but not that of `loess` or `locfit.raw`.

Value

A list with components

`fitted` numeric vector of same length as `y` giving the loess fit

`residuals` numeric vector of same length as `x` giving residuals from the fit

Note

With unequal weights, "loess" was the default method prior to limma version 3.17.25. The default was changed to "locfit" in limma 3.17.25, and then to "weightedLowess" in limma 3.19.16. "weightedLowess" will potentially give somewhat different results to the older algorithms because the local neighborhood of each observation is determined differently (more carefully).

Author(s)

Gordon Smyth

References

Cleveland WS (1979). Robust locally weighted regression and smoothing scatterplots. *Journal of the American Statistical Association* 74, 829-836.

See Also

If weights=NULL, this function calls lowess. Otherwise it calls weightedLowess, locfit.raw or loess. See the help pages of those functions for references and credits.

Compare with loess in the stats package.

See 05.Normalization for an outline of the limma package normalization functions.

Examples

```
x <- (1:100)/101
y <- sin(2*pi*x)+rnorm(100,sd=0.4)
out <- loessFit(y,x)
plot(x,y)
lines(x,out$fitted,col="red")

# Example using weights

y <- x-0.5
w <- rep(c(0,1),50)
y[w==0] <- rnorm(50,sd=0.1)
pch <- ifelse(w>0,16,1)
plot(x,y,pch=pch)
out <- loessFit(y,x,weights=w)
lines(x,out$fitted,col="red")
```

---

logcosh *Logarithm of cosh*

Description

Compute log(cosh(x)) without floating overflow or underflow

Usage

```
logcosh(x)
```

Arguments

x	a numeric vector or matrix.

Details

The computation uses asymptotic expressions for very large or very small arguments. For intermediate arguments, `log(cosh(x))` is returned.

Value

Numeric vector or matrix of same dimensions as `x`.

Author(s)

Gordon K Smyth

See Also

`logsumexp`

Examples

```
x <- c(1e-8,1e-7,1e-6,1e-5,1e-4,1,3,50,800)
logcosh(x)
log(cosh(x))
```

---

logsumexp *Log Sum of Exponentials*

Description

Compute `log( exp(x)+exp(y) )` without floating overflow or underflow

Usage

```
logsumexp(x, y)
```

Arguments

x	a numeric vector or matrix.
y	a numeric vector or matrix of same size as `x`.

Details

The computation uses `logcosh()`.

Value

Numeric vector or matrix of same dimensions as x.

Author(s)

Gordon K Smyth

See Also

logcosh

Examples

```
x <- y <- c(1e-8,1e-7,1e-6,1e-5,1e-4,1,3,50,800)
logsumexp(x,y)
log( exp(x)+exp(y) )
```

---

ma3x3 *Two dimensional Moving Averages with 3x3 Window*

Description

Apply a specified function to each value of a matrix and its immediate neighbors.

Usage

```
ma3x3.matrix(x,FUN=mean,na.rm=TRUE,...)
ma3x3.spottedarray(x,printer,FUN=mean,na.rm=TRUE,...)
```

Arguments

x	numeric matrix
FUN	function to apply to each window of values
na.rm	logical value, should missing values be removed when applying FUN
...	other arguments are passed to FUN
printer	list giving the printer layout, see PrintLayout-class

Details

For ma3x3.matrix, x is a numeric matrix. For ma3x3.spottedarray, each column of x is assumed to contain the expression values of a spotted array in standard order. The printer layout information is used to re-arrange the values of each column as a spatial matrix before applying ma3x3.matrix.

Value

Numeric matrix of same dimension as x containing smoothed values

Author(s)

Gordon Smyth

See Also

An overview of functions for background correction are given in 04.Background.

Examples

```
x <- matrix(c(2,5,3,1,6,3,10,12,4,6,4,8,2,1,9,0),4,4)
ma3x3.matrix(x,FUN="mean")
ma3x3.matrix(x,FUN="min")
```

---

makeContrasts *Construct Matrix of Custom Contrasts*

Description

Construct the contrast matrix corresponding to specified contrasts of a set of parameters.

Usage

```
makeContrasts(..., contrasts=NULL, levels)
```

Arguments

...	expressions, or character strings which can be parsed to expressions, specifying contrasts
contrasts	character vector specifying contrasts
levels	character vector or factor giving the names of the parameters of which contrasts are desired, or a design matrix or other object with the parameter names as column names.

Details

This function expresses contrasts between a set of parameters as a numeric matrix. The parameters are usually the coefficients from a linear model fit, so the matrix specifies which comparisons between the coefficients are to be extracted from the fit. The output from this function is usually used as input to contrasts.fit. The contrasts can be specified either as expressions using ... or as a character vector through contrasts. (Trying to specify contrasts both ways will cause an error.)

The parameter names must be syntactically valid variable names in R and so, for example, must begin with a letter rather than a numeral. See make.names for a complete specification of what is a valid name.

Value

Matrix whose columns correspond to contrasts.

Author(s)

Gordon Smyth

See Also

An overview of linear model functions in limma is given by the help page 06.LinearModels.

Examples

```
# Specify contrasts as a list of expressions:
makeContrasts(B-A, C-B, C-A, levels=c("A","B","C"))

# Can be any numerical expression:
makeContrasts(C-(A+B)/2, levels=c("A","B","C"))

# Optionally include names for the contrasts:
makeContrasts(BvsA=B-A, CvsB=C-B, CvsA=C-A, levels=c("A","B","C"))

# Supply the contrasts as a character vector:
x <- c("B-A", "C-B", "C-A")
makeContrasts(contrasts=x, levels=c("A","B","C"))

# Include names for the contrasts:
x <- c(BvsA="B-A", CvsB="C-B", CvsA="C-A")
makeContrasts(contrasts=x, levels=c("A","B","C"))
```

---

makeUnique *Make Values of Character Vector Unique*

Description

Paste characters on to values of a character vector to make them unique.

Usage

```
makeUnique(x)
```

Arguments

x	object to be coerced to a character vector

Details

Repeated values of x are labeled with suffixes "1", "2" etc.

Value

A character vector of the same length as x

Author(s)

Gordon Smyth

See Also

makeUnique is called by merge.RGList. Compare with make.unique in the base package.

Examples

```
x <- c("a","a","b")
makeUnique(x)
```

MAList-class *M-value, A-value Expression List - class*

Description

A simple list-based class for storing M-values and A-values for a batch of spotted microarrays. MAList objects are usually created during normalization by the functions normalizeWithinArrays or MA.RG.

Slots/List Components

MAList objects can be created by new("MAList",MA) where MA is a list. This class contains no slots (other than .Data), but objects should contain the following components:

M: numeric matrix containing the M-values (log-2 expression ratios). Rows correspond to spots and columns to arrays.
A: numeric matrix containing the A-values (average log-2 expression values).

Optional components include:

weights: numeric matrix of same dimensions as M containing relative spot quality weights. Elements should be non-negativ
other: list containing other matrices, all of the same dimensions as M.
genes: data.frame containing probe information. Should have one row for each spot. May have any number of columns.
targets: data.frame containing information on the target RNA samples. Rows correspond to arrays. May have any number
printer: list containing information on the process used to print the spots on the arrays. See PrintLayout.

Valid MAList objects may contain other optional components, but all probe or array information should be contained in the above components.

Methods

This class inherits directly from class list so any operation appropriate for lists will work on objects of this class. In addition, MAList objects can be subsetted and combined. MAList objects will return dimensions and hence functions such as dim, nrow and ncol are defined. MALists also inherit a show method from the virtual class LargeDataObject, which means that MALists will print in a compact way.

Other functions in LIMMA which operate on MAList objects include normalizeWithinArrays, normalizeBetweenArrays, normalizeForPrintorder, plotMA and plotPrintTipLoess.

Author(s)

Gordon Smyth

See Also

02.Classes gives an overview of all the classes defined by this package.

marrayNorm is the corresponding class in the marray package.

MArrayLM-class *Microarray Linear Model Fit - class*

Description

A list-based S4 class for storing the results of fitting gene-wise linear models to a set of microarrays. Objects are normally created by lmFit, and additional components are added by eBayes.

Components

MArrayLM objects do not contain any slots (apart from .Data) but they should contain the following list components:

coefficients	matrix containing fitted coefficients or contrasts
stdev.unscaled	matrix containing unscaled standard deviations of the coefficients or contrasts
sigma	numeric vector containing residual standard deviations for each gene
df.residual	numeric vector containing residual degrees of freedom for each gene

The following additional components may be created by lmFit:

Amean	numeric vector containing the average log-intensity for each probe over all the arrays in the original lin
genes	data.frame containing probe annotation.
design	design matrix.
cov.coefficients	numeric matrix giving the unscaled covariance matrix of the estimable coefficients
pivot	integer vector giving the order of coefficients in cov.coefficients. Is computed by the QR-decompos
qr	QR-decomposition of the design matrix (if the fit involved no weights or missing values).
...	other components returned by lm.fit (if the fit involved no weights or missing values).

The following component may be added by contrasts.fit:

contrasts	numeric matrix defining contrasts of coefficients for which results are desired.

The following components may be added by eBayes:

s2.prior	numeric value or vector giving empirical Bayes estimated prior value for residual variances
df.prior	numeric value or vector giving empirical Bayes estimated degrees of freedom associated with s2.prior for eac
df.total	numeric vector giving total degrees of freedom used for each gene, usually equal to df.prior + df.residual.

s2.post	numeric vector giving posterior residual variances
var.prior	numeric vector giving empirical Bayes estimated prior variance for each true coefficient
F	numeric vector giving moderated F-statistics for testing all contrasts equal to zero
F.p.value	numeric vector giving p-value corresponding to F
t	numeric matrix containing empirical Bayes t-statistics

Methods

MArrayLM objects will return dimensions and hence functions such as dim, nrow and ncol are defined. MArrayLM objects inherit a show method from the virtual class LargeDataObject.

fitted will return a matrix of fitted values and residuals will return a matrix of residuals.

The functions eBayes, decideTests and classifyTestsF accept MArrayLM objects as arguments.

Author(s)

Gordon Smyth

See Also

02.Classes gives an overview of all the classes defined by this package.

---

mdplot *Mean-Difference Plot*

Description

Creates a mean-difference plot of two columns of a matrix.

Usage

```
mdplot(x, columns=c(1,2), xlab="Mean", ylab="Difference", main=NULL, ...)
```

Arguments

x	numeric matrix with at least two columns.
columns	which columns of x to compare. Plot will display second minus first.
xlab	label for the x-axis.
ylab	label for the y-axis.
main	title of the plot.
...	any other arguments are passed to plotWithHighlights.

Details

Plots differences vs means for a set of bivariate values. This is a generally useful approach for comparing two correlated measures of the same underlying phenomenon. Bland and Altman (1986) argue that it is more informative than a simple scatterplot of the two variables. The bivariate values are stored as columns of `x`.

Value

A plot is created on the current graphics device.

Author(s)

Gordon Smyth

References

Cleveland WS (1993). *Visualizing Data*. Hobart Press.

Bland JM, Altman DG (1986). Statistical methods for assessing agreement between two methods of clinical measurement. *Lancet* 327, 307-310.

See Also

`plotWithHighlights`

`plotMD` is an object-oriented implementation of mean-difference plots for expression data.

An overview of diagnostic functions available in LIMMA is given in 09.Diagnostics.

Examples

```
x1 <- runif(100)
x2 <- (x1 + rnorm(100,sd=0.01))^1.2
oldpar <- par(mfrow=c(1,2))
plot(x1,x2)
mdplot(cbind(x1,x2),bg.pch=1,bg.cex=1)
par(oldpar)
```

---

merge *Merge RGList or MAList Data Objects*

Description

Merge two microarray data sets represented by RGLists in possibly irregular order.

Usage

```
## S3 method for class 'RGList'
merge(x,y,...)
```

Arguments

x	data object of class RGList, MAList, EList or EListRaw.
y	data object of same class as x, corresponding to the same genes as for x, possibly in a different order, but with different arrays.
...	other arguments are accepted but not used at present

Details

RGList, MAList, EListRaw and EList data objects are lists containing numeric matrices all of the same dimensions. The data objects are merged by merging each of the components by row names or, if there are no row names, by IDs in the genes component. Unlike when using cbind, row names are not required to be in the same order or to be unique. In the case of repeated row names, the order of the rows with repeated names is preserved. This means that the first occurrence of each name in x is matched with the first occurrence of the same name in y, the second with the second, and so on. The final vector of row names is the same as in x.

Note: if the objects contain the same number of genes in the same order then the appropriate function to combine them is cbind rather than merge.

Value

A merged object of the same class as x and y with the same components as x. Component matrices have the same rows names as in x but columns from y as well as from x.

Author(s)

Gordon Smyth

See Also

R base provides a merge method for merging data.frames.

An overview of limma commands for reading, subsetting and merging data is given in 03.Reading-Data.

Examples

```
M <- A <- matrix(11:14,4,2)
rownames(M) <- rownames(A) <- c("a","a","b","c")
MA1 <- new("MAList",list(M=M,A=A))

M <- A <- matrix(21:24,4,2)
rownames(M) <- rownames(A) <- c("b","a","a","c")
MA2 <- new("MAList",list(M=M,A=A))

merge(MA1,MA2)
merge(MA2,MA1)
```

mergeScans *Merge two scans of two-color arrays*

Description

Merge two sets of intensities of two-color arrays that are scanned twice at two different scanner settings, one at a lower gain setting with no saturated spot intensities and the other at a higher gain setting with a higher signal-to-noise ratio and some saturated spot intensities.

Usage

```
mergeScansRG(RGlow, RGhigh, AboveNoiseLowG=NULL, AboveNoiseLowR=NULL, outlierp=0.01)
```

Arguments

RGlow	object of class RGList containing red and green intensities constituting two-color microarray data scanned at a lower gain setting.
RGhigh	object of class RGList containing red and green intensities constituting two-color microarray data scanned at a higher gain setting.
AboveNoiseLowG	matrix of 1 or 0 for low scan intensities of green color, 1 for spots above noise level or 0 otherwise. One column per array.
AboveNoiseLowR	matrix of 1 or 0 for low scan intensities of red color, 1 for spots above noise level or 0 otherwise. One column per array.
outlierp	p-value for outliers. 0 for no outlier detection or any value between 0 and 1. Default p-value is 0.01.

Details

This function merges two separate scans of each fluorescent label on a two-color array scanned at two different scanner settings by using a nonlinear regression model consisting of two linear regression lines and a quadratic function connecting the two, which looks like a hockey stick. The changing point, i.e. the saturation point, in the high scan is also estimated as part of the model. Signals produced for certain spots can sometimes be very low (below noise) or too high (saturated) to be accurately read by the scanner. The proportions of spots that are below noise or above saturation are affected by the settings of the laser scanner used to read the arrays, with low scans minimizing saturation effects and high scans maximizing signal-to-noise ratios. Saturated spots can cause bias in intensity ratios that cannot be corrected for using conventional normalization methods.

Each fluorescent label on a two-color array can be scanned twice: for example, a high scan targeted at reaching saturation level for the brightest 1 percent of the spots on the array, and a low scan targeted at the lowest level of intensity which still allowed accurate grid placement on the arrays. By merging data from two separate laser scans of each fluorescent label on an array, we can avoid the potential bias in signal intensities due to below noise or above saturation and thus provide better estimates of true differential expression as well as increase usable spots.

The merging process is designed to retain signal intensities from the high scan except when scanner saturation causes the high scan signal to be under-measured. The saturated spots are predicted from the corresponding low scans by the fitted regression model. It also checks for any inconsistency between low and high scans.

Value

An object of class RGList-class with the following components:

G	numeric matrix containing the merged green (cy3) foreground intensities. Rows correspond to spots and columns to arrays.
R	numeric matrix containing the merged red (cy5) foreground intensities. Rows correspond to spots and columns to arrays.
Gb	numeric matrix containing the green (cy3) background intensities from high scan.
Rb	numeric matrix containing the red (cy5) background intensities from high scan.
other	list of numeric matrices Gsaturated, Rsaturated, Goutlier and Routlier. The first two contain saturation flags (1=saturated, 0=otherwise) for the green (cy3) and red (cy5) channels of the high scan. The second two contain outlier flags (1=outlier, 0=otherwise) for the green (cy3) and red (Cy5) channels.

Author(s)

Dongseok Choi <choid@ohsu.edu>.

References

Choi D, O'Malley JP, Lasarev MR, Lapidus J, Lu X, Pattee P, Nagalla SR (2006). Extending the Dynamic Range of Signal Intensities in DNA Microarrays. *Online Journal of Bioinformatics* 7, 46-56.

Examples

```
## Not run:
#RG1: An RGList from low scan
#RG2: An RGList from high scan
RGmerged <- mergeScansRG(RG1,RG2,AboveNoiseLowG=ANc3,AboveNoiseLowR=ANc5)

#merge two scans when all spots are above noise in low scan and no outlier detection.
RGmerged <- mergeScansRG(RG1,RG2,outlierp=0)

## End(Not run)
```

---

modelMatrix *Construct Design Matrix*

Description

Construct design matrix from RNA target information for a two colour microarray experiment.

Usage

```
modelMatrix(targets, parameters, ref, verbose=TRUE)
uniqueTargets(targets)
```

Arguments

targets	matrix or data.frame with columns Cy3 and Cy5 specifying which RNA was hybridized to each array
parameters	matrix specifying contrasts between RNA samples which should correspond to regression coefficients. Row names should correspond to unique RNA sample names found in targets.
ref	character string giving name of one of the RNA sources to be treated as reference. Exactly one argument of parameters or ref should be specified.
verbose	logical, if TRUE then unique names found in targets will be printed to standard output

Details

This function computes a design matrix for input to lmFit when analyzing two-color microarray experiments in terms of log-ratios.

If the argument ref is used, then the experiment is treated as a one-way layout and the coefficients measure expression changes relative to the RNA source specified by ref. The RNA source ref is often a common reference which appears on every array or is a control sample to which all the others are compared. There is no restriction however. One can choose ref to be any of the RNA sources appearing in the Cy3 or Cy5 columns of targets.

If the parameters argument is set, then the columns of this matrix specify the comparisons between the RNA sources which are of interest. This matrix must be of size n by (n-1), where n is the number of unique RNA sources found in Cy3 and Cy5, and must have row names which correspond to the RNA sources.

Value

modelMatrix produces a numeric design matrix with row names as in targets and column names as in parameters.

uniqueTargets produces a character vector of unique target names from the columns Cy3 and Cy5 of targets.

Author(s)

Gordon Smyth

See Also

model.matrix in the stats package.

An overview of linear model functions in limma is given by 06.LinearModels.

Examples

```
targets <- cbind(Cy3=c("Ref","Control","Ref","Treatment"),Cy5=c("Control","Ref","Treatment","Ref"))
rownames(targets) <- paste("Array",1:4)

parameters <- cbind(C=c(-1,1,0),T=c(-1,0,1))
```

```
rownames(parameters) <- c("Ref","Control","Treatment")

modelMatrix(targets, parameters)
modelMatrix(targets, ref="Ref")
```

`modifyWeights` *Modify Matrix of Weights By Control Status of Rows*

Description

Modify weights matrix for given gene status values.

Usage

```
modifyWeights(weights=rep(1,length(status)), status, values, multipliers)
```

Arguments

`weights`	numeric matrix of relative weights, rows corresponding to genes and columns to arrays
`status`	character vector giving the control status of each spot on the array, of same length as the number of rows of `weights`
`values`	character vector giving subset of the unique values of `status`
`multipliers`	numeric vector of same length as `values` giving factor by which weights will be modified

Details

The function is usually used to temporarily modify the weights matrix during normalization of data. The function can be used for example to give zero weight to spike-in ratio control spots during normalization.

Value

Numeric matrix of same dimensions as `weights` with rows corresponding to `values` in `status` modified by the specified multipliers.

Author(s)

Gordon Smyth

See Also

An overview of normalization functions available in LIMMA is given in 05.Normalization.

Examples

```
w <- matrix(runif(6*3),6,3)
status <- c("Gene","Gene","Ratio_Control","Ratio_Control","Gene","Gene")
modifyWeights(w,status,values="Ratio_Control",multipliers=0)
```

`mrlm` *Fit Linear Model to Microarray Data by Robust Regression*

Description

Fit a linear model genewise to expression data from a series of arrays. The fit is by robust M-estimation allowing for a small proportion of outliers. This is a utility function for `lmFit`.

Usage

```
mrlm(M, design = NULL, ndups = 1, spacing = 1, weights = NULL, ...)
```

Arguments

`M`	numeric matrix containing log-ratio or log-expression values for a series of microarrays, rows correspond to genes and columns to arrays.
`design`	numeric design matrix defining the linear model, with rows corresponding to arrays and columns to comparisons to be estimated. The number of rows must match the number of columns of `M`. Defaults to the unit vector meaning that the arrays are treated as replicates.
`ndups`	a positive integer giving the number of times each gene is printed on an array. `nrow(M)` must be divisible by `ndups`.
`spacing`	the spacing between the rows of `M` corresponding to duplicate spots, `spacing=1` for consecutive spots.
`weights`	numeric matrix of the same dimension as `M` containing weights. If it is of different dimension to `M`, it will be filled out to the same size. `NULL` is equivalent to equal weights.
`...`	any other arguments are passed to `rlm.default`.

Details

This is a utility function used by the higher level function `lmFit`. Most users should not use this function directly but should use `lmFit` instead.

This function fits a linear model for each gene by calling the function `rlm` from the MASS library.

Warning: don't use weights with this function unless you understand how `rlm` treats weights. The treatment of weights is somewhat different from that of `lm.series` and `gls.series`.

Value

A list with components

coefficients	numeric matrix containing the estimated coefficients for each linear model. Same number of rows as M, same number of columns as design.
stdev.unscaled	numeric matrix conformal with coef containing the unscaled standard deviations for the coefficient estimators. The standard errors are given by stdev.unscaled * sigma.
sigma	numeric vector containing the residual standard deviation for each gene.
df.residual	numeric vector giving the degrees of freedom corresponding to sigma.
qr	QR decomposition of design.

Author(s)

Gordon Smyth

See Also

rlm.

An overview of linear model functions in limma is given by 06.LinearModels.

---

nec *NormExp Background Correction and Normalization Using Control Probes*

Description

Perform normexp background correction using negative control probes and quantile normalization using negative and positive control probes. Particularly useful for Illumina BeadChips.

Usage

```
nec(x, status=NULL, negctrl="negative", regular="regular", offset=16,
    robust=FALSE, detection.p="Detection")
neqc(x, status=NULL, negctrl="negative", regular="regular", offset=16,
    robust=FALSE, detection.p="Detection", ...)
```

Arguments

x	object of class EListRaw or matrix containing raw intensities for regular and control probes from a series of microarrays.
status	character vector giving probe types. Defaults to x$genes$Status if x is an EListRaw object.
negctrl	character string identifier for negative control probes.

regular	character string identifier for regular probes, i.e., all probes other than control probes.
offset	numeric value added to the intensities after background correction.
robust	logical. Should robust estimators be used for the background mean and standard deviation?
detection.p	detection p-values. Only used when no negative control probes can be found in the data. Can be a numeric matrix or a character string giving the name of the component of x$other containing the matrix.
...	any other arguments are passed to normalizeBetweenArrays.

Details

neqc performs background correction followed by quantile normalization, using negative control probes for background correction and both negative and positive controls for normalization (Shi et al, 2010). nec is similar but performs background correction only. These methods are particularly designed for Illumina BeadChip microarrays, but could be useful for other platforms for which good quality negative control probes or detection p-values are available.

When control data are available, these functions call normexp.fit.control to estimate the parameters required by normal+exponential(normexp) convolution model with the help of negative control probes, followed by normexp.signal to perform the background correction. If x contains background intensities x$Eb, then these are first subtracted from the foreground intensities, prior to normexp background correction. After background correction, an offset is added to the data.

When expression values for negative controls are not available, the detection.p argument is used instead. In that case, these functions call normexp.fit.detection.p, which infers the negative control probe intensities from the detection p-values associated with the regular probes. The function outputs a message if this is done.

For more detailed descriptions of the arguments x, status, negctrl, regular and detection.p, please refer to functions normexp.fit.control, normexp.fit.detection.p and read.ilmn.

Both nec and neqc perform the above steps. neqc continues on to quantile normalize the background-corrected intensities, including control probes. After normalization, the intensities are log2 transformed and the control probes are removed.

Value

nec produces a EListRaw-class or matrix object of the same dimensions as x containing background-corrected intensities, on the raw scale. neqc produces a EList-class or matrix object containing normalized log2 intensities, with rows corresponding to control probes removed.

Author(s)

Wei Shi and Gordon Smyth

References

Shi W, Oshlack A, Smyth GK (2010). Optimizing the noise versus bias trade-off for Illumina Whole Genome Expression BeadChips. *Nucleic Acids Research* 38(22), e204. doi:10.1093/nar/gkq871.

See Also

An overview of background correction functions is given in 04.Background.

An overview of LIMMA functions for normalization is given in 05.Normalization.

`normexp.fit.control` estimates the parameters in the normal+exponential convolution model using the negative control probes.

`normexp.fit.detection.p` estimates the parameters in the normal+exponential convolution model using negative control probe intensities inferred from regular probes by using their detection p-value information.

`normexp.fit` estimates parameters in the normal+exponential convolution model using a saddle-point approximation or other methods.

`neqc` performs normexp background correction and quantile normalization aided by control probes.

Examples

```
## Not run:
# neqc normalization for data which include control probes
x <- read.ilmn(files="sample probe profile.txt", ctrlfiles="control probe profile.txt")
y <- neqc(x)
fit <- lmFit(y,design)

# Same thing but in separate steps:
x.b <- nec(x)
y <- normalizeBetweenArrays(x.b,method="quantile")
y <- y[y$genes$Status=="regular",]

# neqc normalization for data without control probes
# neqc can process detection p-values in lieu of control probes
xr <- read.ilmn(files="sample probe profile.txt")
yr <- neqc(xr)

## End(Not run)
```

`normalizeBetweenArrays`

Normalize Between Arrays

Description

Normalizes expression intensities so that the intensities or log-ratios have similar distributions across a set of arrays.

Usage

```
normalizeBetweenArrays(object, method=NULL, targets=NULL, cyclic.method="fast", ...)
normalizeBetweenSamples(object, method="quantile", cyclic.method="fast", ...)
```

Arguments

object	For normalizeBetweenArrays, a numeric matrix, EListRaw, RGList or MAList object containing un-normalized expression data. For normalizeBetweenSamples, a numeric matrix or EList object containing log-transformed un-normalized expression data. If a matrix, then it is assumed to contain log-transformed single-channel data.
method	character string specifying the normalization method to be used. Choices for single-channel data are "none", "scale", "quantile" or "cyclicloess". Choices for two-color data are those previously mentioned plus "Aquantile", "Gquantile", "Rquantile" or "Tquantile". A partial string sufficient to uniquely identify the choice is permitted. The default is "Aquantile" for two-color data objects or "quantile" for single-channel objects.
targets	vector, factor or matrix of length twice the number of arrays, used to indicate target groups if method="Tquantile"
cyclic.method	character string indicating the variant of normalizeCyclicLoess to be used if method=="cyclicloess", see normalizeCyclicLoess for possible values.
...	other arguments are passed to normalizeQuantiles or normalizeCyclicLoess

Details

normalizeBetweenArrays normalizes expression values to achieve consistency between arrays. For two-color arrays, normalization between arrays is usually a follow-up step after normalization within arrays using normalizeWithinArrays. For single-channel arrays, within array normalization is not usually relevant and so normalizeBetweenArrays is the sole normalization step.

For single-channel data, the scale, quantile or cyclic loess normalization methods can be applied to the columns of data. Trying to apply other normalization methods when object is a matrix or EListRaw object will produce an error. If object is an EListRaw object, then normalization will be applied to the matrix object$E of expression values, which will then be log2-transformed. Scale (method="scale") scales the columns to have the same median. Quantile and cyclic loess normalization was originally proposed by Bolstad et al (2003) for Affymetrix-style single-channel arrays. Quantile normalization forces the entire empirical distribution of each column to be identical. Cyclic loess normalization applies loess normalization to all possible pairs of arrays, usually cycling through all pairs several times. Cyclic loess is slower than quantile, but allows probe-wise weights and is more robust to unbalanced differential expression.

The other normalization methods are for two-color arrays. Scale normalization was proposed by Yang et al (2001, 2002) and is further explained by Smyth and Speed (2003). The idea is simply to scale the log-ratios to have the same median-absolute-deviation (MAD) across arrays. This idea has also been implemented by the maNormScale function in the marray package. The implementation here is slightly different in that the MAD scale estimator is replaced with the median-absolute-value and the A-values are normalized as well as the M-values.

Quantile normalization was explored by Yang and Thorne (2003) for two-color cDNA arrays. method="quantile" ensures that the intensities have the same empirical distribution across arrays and across channels. method="Aquantile" ensures that the A-values (average intensities) have the same empirical distribution across arrays leaving the M-values (log-ratios) unchanged. These two methods are called "q" and "Aq" respectively in Yang and Thorne (2003).

method="Tquantile" performs quantile normalization separately for the groups indicated by targets. targets may be a target frame such as read by readTargets or can be a vector indicating green channel groups followed by red channel groups.

method="Gquantile" ensures that the green (first) channel has the same empirical distribution across arrays, leaving the M-values (log-ratios) unchanged. This method might be used when the green channel is a common reference throughout the experiment. In such a case the green channel represents the same target throughout, so it makes compelling sense to force the distribution of intensities to be same for the green channel on all the arrays, and to adjust to the red channel accordingly. method="Rquantile" ensures that the red (second) channel has the same empirical distribution across arrays, leaving the M-values (log-ratios) unchanged. Both Gquantile and Rquantile normalization have the implicit effect of changing the red and green log-intensities by equal amounts.

See the limma User's Guide for more examples of use of this function.

normalizeBetweenSamples is designed for more general technologies, not necessarily microarrays. Instead of transforming an EListRaw object to an EList, it directly accepts an EList.

Value

If object is a matrix then normalizeBetweenArrays produces a matrix of the same size. If object is an EListRaw object, then an EList object with expression values on the log2 scale is produced. For two-color data, normalizeBetweenArrays produces an MAList object with M and A-values on the log2 scale.

normalizeBetweenSamples returns a matrix or EList, depending on the class of object. If an EList, then the E log-expression matrix is normalized.

Author(s)

Gordon Smyth

References

Bolstad BM, Irizarry RA, Astrand M, Speed TP (2003), A comparison of normalization methods for high density oligonucleotide array data based on bias and variance. *Bioinformatics* 19, 185-193.

Smyth GK, Speed TP (2003). Normalization of cDNA microarray data. *Methods* 31, 265-273.

Yang YH, Dudoit S, Luu P, Speed TP (2001). Normalization for cDNA microarray data. In *Microarrays: Optical Technologies and Informatics*, M. L. Bittner, Y. Chen, A. N. Dorsel, and E. R. Dougherty (eds), Proceedings of SPIE, Volume 4266, pp. 141-152.

Yang YH, Dudoit S, Luu P, Lin DM, Peng V, Ngai J, Speed TP (2002). Normalization for cDNA microarray data: a robust composite method addressing single and multiple slide systematic variation. *Nucleic Acids Research* 30(4):e15.

Yang YH, Thorne NP (2003). Normalization for two-color cDNA microarray data. In: D. R. Goldstein (ed.), *Science and Statistics: A Festschrift for Terry Speed*, IMS Lecture Notes - Monograph Series, Volume 40, pp. 403-418.

See Also

An overview of LIMMA functions for normalization is given in 05.Normalization.

The neqc function provides a variation of quantile normalization that is customized for Illumina BeadChips. This method uses control probes to refine the background correction and normalization steps.

Note that vsn normalization, previously offered as a method of this function, is now performed by the normalizeVSN function.

See also maNormScale in the marray package and normalize-methods in the affy package.

Examples

```
ngenes <- 100
narrays <- 4
x <- matrix(rnorm(ngenes*narrays),100,4)
y <- normalizeBetweenArrays(x)
```

---

normalizeCyclicLoess *Normalize Columns of a Matrix by Cyclic Loess*

Description

Normalize the columns of a matrix, cyclically applying loess normalization to normalize each pair of columns to each other.

Usage

```
normalizeCyclicLoess(x, weights = NULL, span = 0.7, adaptive.span = TRUE,
                     iterations = 3, method = "fast")
```

Arguments

x	numeric matrix, or object which can be coerced to a numeric matrix, containing log-expression values.
weights	numeric vector of probe weights. Must be non-negative.
span	span of loess smoothing window, between 0 and 1.
adaptive.span	logical. If TRUE, then an optimal value for span will be chosen depending on the number of rows of x.
iterations	number of times to cycle through all pairs of columns.
method	character string specifying which variant of the cyclic loess method to use. Options are "fast", "affy" or "pairs".

Details

This function is intended to normalize single channel or A-value microarray intensities between arrays. Cyclic loess normalization is similar in effect and intention to quantile normalization, but with some advantages, in particular the ability to incorporate probe weights.

A number of variants of cyclic loess have been suggested. `method="pairs"` implements the intuitive idea that each pair of arrays is subjected to loess normalization as for two-color arrays. This process is simply cycled through all possible pairs of arrays, then repeated for several `iterations`. This is the method described by Ballman et al (2004) as ordinary cyclic loess normalization.

`method="affy"` implements a method similar to `normalize.loess` in the affy package, except that here we call `lowess` instead of `loess` and avoid the use of probe subsets and the `predict` function. In this approach, no array is modified until a complete cycle of all pairs has been completed. The adjustments are stored for a complete iteration, then averaged, and finally used to modify the arrays. The `"affy"` method is invariant to the order of the columns of `x`, whereas the `"pairs"` method is not. The affy approach is presumably that used by Bolstad et al (2003), although the algorithm was not explicitly described in that article.

`method="fast"` implements the "fast linear loess" method of Ballman et al (2004), whereby each array is simply normalized to a reference array, the reference array being the average of all the arrays. This method is relatively fast because computational time is linear in the number of arrays, whereas `"pairs"` and `"affy"` are quadratic in the number of arrays. `"fast"` requires n lowess fits per iteration, where n is the number of arrays, whereas `"pairs"` and `"affy"` require n*(n-1)/2 lowess fits per iteration.

If `adaptive.span` is `TRUE`, then `span` is set to `chooseLowessSpan(nrow(x), small.n=50, min.span=0.3, power=1/3)`.

Value

A matrix of the same dimensions as `x` containing the normalized values.

Author(s)

Yunshun (Andy) Chen and Gordon Smyth

References

Bolstad BM, Irizarry RA, Astrand M, Speed TP (2003). A comparison of normalization methods for high density oligonucleotide array data based on bias and variance. *Bioinformatics* 19, 185-193.

Ballman KV, Grill DE, Oberg AL, Therneau TM (2004). Faster cyclic loess: normalizing RNA arrays via linear models. *Bioinformatics* 20, 2778-2786.

See Also

An overview of LIMMA functions for normalization is given in 05.Normalization.

normalize.loess in the affy package also implements cyclic loess normalization, without weights.

normalizeForPrintorder *Print-Order Normalization*

Description

Normalize intensity values on one or more spotted microarrays to adjust for print-order effects.

Usage

```
normalizeForPrintorder(object, layout, start="topleft", method = "loess",
                       separate.channels = FALSE, span = 0.1, plate.size = 32)
normalizeForPrintorder.rg(R, G, printorder, method = "loess", separate.channels = FALSE,
                          span = 0.1, plate.size = 32, plot = FALSE)
plotPrintorder(object, layout, start="topleft", slide = 1, method = "loess",
               separate.channels = FALSE, span = 0.1, plate.size = 32)
```

Arguments

object	an RGList or list object containing components R and G which are matrices containing the red and green channel intensities for a series of arrays
R	numeric vector containing red channel intensities for a single microarray
G	numeric vector containing the green channel intensities for a single microarray
layout	list specifying the printer layout, see PrintLayout-class
start	character string specifying where printing starts in each pin group. Choices are "topleft" or "topright".
printorder	numeric vector specifying order in which spots are printed. Can be computed from printorder(layout,start=start).
slide	positive integer giving the column number of the array for which a plot is required
method	character string, "loess" if a smooth loess curve should be fitted through the print-order trend or "plate" if plate effects are to be estimated
separate.channels	logical, TRUE if normalization should be done separately for the red and green channel and FALSE if the normalization should be proportional for the two channels
span	numerical constant between 0 and 1 giving the smoothing span for the loess the curve. Ignored if method="plate".
plate.size	positive integer giving the number of consecutive spots corresponding to one plate or plate pack. Ignored if method="loess".
plot	logical. If TRUE then a scatter plot of the print order effect is sent to the current graphics device.

Details

Print-order is associated with the 384-well plates used in the printing of spotted microarrays. There may be variations in DNA concentration or quality between the different plates. There may be variations in ambient conditions during the time the array is printed.

This function is intended to pre-process the intensities before other normalization methods are applied to adjust for variations in DNA quality or concentration and other print-order effects.

Printorder means the order in which spots are printed on a microarray. Spotted arrays are printed using a print head with an array of print-tips. Spots in the various tip-groups are printed in parallel. Printing is assumed to start in the top left hand corner of each tip-group and to proceed right and down by rows, or else to start in the top right hand corner and to proceed left and down by rows. See `printorder` for more details. (WARNING: this is not always the case.) This is true for microarrays printed at the Australian Genome Research Facility but might not be true for arrays from other sources.

If `object` is an `RGList` then printorder is performed for each intensity in each array.

`plotPrintorder` is a non-generic function which calls `normalizeForPrintorder` with `plot=TRUE`.

Value

`normalizeForPrintorder` produces an `RGList` containing normalized intensities.

The function `plotPrintorder` or `normalizeForPrintorder.rg` with `plot=TRUE` returns no value but produces a plot as a side-effect.

`normalizeForPrintorder.rg` with `plot=FALSE` returns a list with the following components:

`R`	numeric vector containing the normalized red channel intensities
`G`	numeric vector containing the normalized green channel intensities
`R.trend`	numeric vector containing the fitted printorder trend for the red channel
`G.trend`	numeric vector containing the fitted printorder trend for the green channel

Author(s)

Gordon Smyth

References

Smyth GK. Print-order normalization of cDNA microarrays. March 2002. https://gksmyth.github.io/pubs/porder/porder.html

See Also

`printorder`.

An overview of LIMMA functions for normalization is given in 05.Normalization.

Examples

```
## Not run:
plotPrintorder(RG,layout,slide=1,separate=TRUE)
RG <- normalizeForPrintorder(mouse.data,mouse.setup)

## End(Not run)
```

`normalizeMedianAbsValues` *Normalize Columns of a Matrix to have the Median Absolute Value*

Description

Performs scale normalization of an M-value matrix or an A-value matrix across a series of arrays. Users do not normally need to call these functions directly - use `normalizeBetweenArrays` instead.

Usage

```
normalizeMedianValues(x)
normalizeMedianAbsValues(x)
```

Arguments

`x`	numeric matrix

Details

If `x` is a matrix of log-ratios of expression (M-values) then `normalizeMedianAbsValues` is very similar to scaling to equalize the median absolute deviation (MAD) as in Yang et al (2001, 2002). Here the median-absolute value is used in preference so as not to re-center the M-values.

`normalizeMedianAbsValues` is also used to scale the A-values when scale-normalization is applied to an `MAList` object.

Value

A numeric matrix of the same size as that input which has been scaled so that each column has the same median value (for `normalizeMedianValues`) or median-absolute value (for `normalizeMedianAbsValues`).

Author(s)

Gordon Smyth

See Also

An overview of LIMMA functions for normalization is given in 05.Normalization.

Examples

```
M <- cbind(Array1=rnorm(10),Array2=2*rnorm(10))
normalizeMedianAbsValues(M)
```

`normalizeQuantiles` *Normalize Columns of a Matrix to have the same Quantiles*

Description

Normalize the columns of a matrix to have the same quantiles, allowing for missing values. Users do not normally need to call this function directly - use `normalizeBetweenArrays` instead.

Usage

```
normalizeQuantiles(A, ties=TRUE)
```

Arguments

`A`	numeric matrix. Missing values are allowed.
`ties`	logical. If `TRUE`, ties in each column of `A` are treated in a careful way. Tied values will be normalized to the mean of the corresponding pooled quantiles.

Details

This function is intended to normalize single channel or A-value microarray intensities between arrays. Each quantile of each column is set to the mean of that quantile across arrays. The intention is to make all the normalized columns have the same empirical distribution. This will be exactly true if there are no missing values and no ties within the columns: the normalized columns are then simply permutations of one another.

If there are ties amongst the intensities for a particular array, then with `ties=FALSE` the ties are broken in an unpredictable order. If `ties=TRUE`, all the tied values for that array will be normalized to the same value, the average of the quantiles for the tied values.

Value

A matrix of the same dimensions as `A` containing the normalized values.

Author(s)

Gordon Smyth

References

Bolstad BM, Irizarry RA, Astrand M, Speed TP (2003), A comparison of normalization methods for high density oligonucleotide array data based on bias and variance. *Bioinformatics* 19, 185-193.

See Also

An overview of LIMMA functions for normalization is given in 05.Normalization.

`normalizeRobustSpline` *Normalize Single Microarray Using Shrunk Robust Splines*

Description

Normalize the M-values for a single microarray using robustly fitted regression splines and empirical Bayes shrinkage.

Usage

```
normalizeRobustSpline(M,A,layout=NULL,df=5,method="M")
```

Arguments

`M`	numeric vector of M-values
`A`	numeric vector of A-values
`layout`	list specifying the dimensions of the spot matrix and the grid matrix. Defaults to a single group for the whole array.
`df`	degrees of freedom for regression spline, i.e., the number of regression coefficients and the number of knots
`method`	choices are `"M"` for M-estimation or `"MM"` for high breakdown point regression

Details

This function implements an idea similar to print-tip loess normalization but uses regression splines in place of the loess curves and uses empirical Bayes ideas to shrink the individual print-tip curves towards a common value. This allows the technique to introduce less noise into good quality arrays with little spatial variation while still giving good results on arrays with strong spatial variation.

The original motivation for the robustspline method was to use whole-array information to moderate the normalization curves used for the individual print-tip groups. This was an important issue for academically printed spotted two-color microarrays, especially when some of the print-tip groups contained relatively few spots. In these situations, robust spline normalization ensures stable results even for print-tip groups with few spots.

Modern commercial two colour arrays do not usually have print tips, so in effect the whole array is a single print-tip group, and so the need for moderating individual curves is gone. Robustspline normalization can still be used for data from these arrays, in which case a single normalization curve is estimated. In this situation, the method is closely analogous to global loess, with a regression spline replacing the loess curve and with robust regression replacing the loess robustifying weights. Robust spline normalization with `method="MM"` has potential advantages over global loess normalization when there is a lot of differential expression or the differential expression is asymmetric, because of the increased level of robustness. The potential advantages of this approach have not been fully explored in a refereed publication however.

Value

Numeric vector containing normalized M-values.

Author(s)

Gordon Smyth

References

Ritchie ME, Phipson B, Wu D, Hu Y, Law CW, Shi W, Smyth GK (2015). limma powers differential expression analyses for RNA-sequencing and microarray studies. *Nucleic Acids Research* 43, e47. doi:10.1093/nar/gkv007.

See Also

`normalizeRobustSpline` uses `ns` in the splines package to specify regression splines and `rlm` in the MASS package for robust regression.

This function is usually accessed through `normalizeWithinArrays`. An overview of LIMMA functions for normalization is given in 05.Normalization.

Examples

```
A <- 1:100
M <- rnorm(100)
normalized.M <- normalizeRobustSpline(M,A)

# Usual usage
## Not run: MA <- normalizeWithinArrays(RG, method="robustspline")
```

---

normalizeVSN *Variance Stabilizing Normalization (vsn)*

Description

Apply variance stabilizing normalization (vsn) to limma data objects.

Usage

```
normalizeVSN(x, ...)
```

Arguments

`x`	a numeric `matrix`, `EListRaw` or `RGList` object.
`...`	other arguments are passed to `vsn`

Details

This is an interface to the vsnMatrix function from the vsn package. The input x should contain raw intensities. If x contains background as well as foreground intensities, these will be subtracted from the foreground intensities before vsnMatrix is called.

Note that the vsn algorithm performs background correction and normalization simultaneously. If the data are from two-color microarrays, then the red and green intensities are treated as if they were single channel data, i.e., red and green channels from the same array are treated as unpaired. This algorithm is therefore separate from the backgroundCorrection, normalizeWithinArrays, then normalizeBetweenArrays paradigm used elsewhere in the limma package.

Value

The class of the output depends on the input. If x is a matrix, then the result is a matrix of the same size. If x is an EListRaw object, then an EList object with expression values on the log2 scale is produced. If x is an RGList, then an MAList object with M and A-values on the log2 scale is produced.

Author(s)

Gordon Smyth

References

Huber W, von Heydebreck A, Sueltmann H, Poustka A, Vingron M (2002). Variance stabilization applied to microarray data calibration and to the quantification of differential expression. Bioinformatics 18 Supplement 1, S96-S104.

See Also

An overview of LIMMA functions for normalization is given in 05.Normalization.

See also vsnMatrix in the vsn package.

Examples

```
ngenes <- 100
narrays <- 4
x <- matrix(rnorm(ngenes*narrays),100,4)
y <- normalizeVSN(x)
```

---

normalizeWithinArrays *Normalize Within Arrays*

Description

Normalize the expression log-ratios for one or more two-colour spotted microarray experiments so that the log-ratios average to zero within each array or sub-array.

Usage

```
normalizeWithinArrays(object, layout, method="printtiploess", weights=object$weights,
                    span=0.3, iterations=4, controlspots=NULL, df=5, robust="M",
                      bc.method="subtract", offset=0)
MA.RG(object, bc.method="subtract", offset=0)
RG.MA(object)
```

Arguments

object	object of class list, RGList or MAList containing red and green intensities constituting two-color microarray data.
layout	list specifying the dimensions of the spot matrix and the grid matrix. For details see PrintLayout-class.
method	character string specifying the normalization method. Choices are "none", "median", "loess", "printtiploess", "composite", "control" and "robustspline". A partial string sufficient to uniquely identify the choice is permitted.
weights	numeric matrix or vector of the same size and shape as the components of object containing spot quality weights.
span	numeric scalar giving the smoothing parameter for the loess fit
iterations	number of iterations used in loess fitting. More iterations give a more robust fit.
controlspots	numeric or logical vector specifying the subset of spots which are non-differentially-expressed control spots, for use with method="composite" or method="control".
df	degrees of freedom for spline if method="robustspline".
robust	robust regression method if method="robustspline". Choices are "M" or "MM".
bc.method	character string specifying background correct method, see backgroundCorrect for options.
offset	numeric value, intensity offset used when computing log-ratios, see backgroundCorrect.

Details

Normalization is intended to remove from the expression measures any systematic trends which arise from the microarray technology rather than from differences between the probes or between the target RNA samples hybridized to the arrays.

This function normalizes M-values (log-ratios) for dye-bias within each array. Apart from method="none" and method="median", all the normalization methods make use of the relationship between dye-bias and intensity. Method "none" computes M-values and A-values but does no normalization. Method "median" subtracts the weighted median from the M-values for each array.

The loess normalization methods ("loess", "printtiploess" and "composite") were proposed by Yang et al (2001, 2002). Smyth and Speed (2003) review these methods and describe how the methods are implemented in the limma package, including choices of tuning parameters. More information on the loess control parameters span and iterations can be found under loessFit. The default values used here are equivalent to those for the older function stat.ma in the sma package.

Oshlack et al (2004) consider the special issues that arise when a large proportion of probes are differentially expressed. They propose an improved version of composite loess normalization, which

is implemented in the "control" method. This fits a global loess curve through a set of control spots, such as a whole-library titration series, and applies that curve to all the other spots.

The "robustspline" method calls normalizeRobustSpline. See that function for more documentation.

MA.RG converts an unlogged RGList object into an MAList object. MA.RG(object) is equivalent to normalizeWithinArrays(object,method="none").

RG.MA(object) converts back from an MAList object to a RGList object with unlogged intensities.

weights is normally a matrix giving a quality weight for every spot on every array. If weights is instead a vector or a matrix with only one column, then the weights will be assumed to be the same for every array, i.e., the weights will be probe-specific rather than spot-specific.

Value

An object of class MAList. Any components found in object will be preserved except for R, G, Rb, Gb and other.

Author(s)

Gordon Smyth

References

Oshlack A, Emslie D, Corcoran L, Smyth GK (2007). Normalization of boutique two-color microarrays with a high proportion of differentially expressed probes. *Genome Biology* 8, R2. doi:10.1186/gb200781r2.

Smyth GK, Speed TP (2003). Normalization of cDNA microarray data. *Methods* 31, 265-273. doi:10.1016/S10462023(03)001555. See also the preprint at https://gksmyth.github.io/pubs/normalize.pdf.

Yang YH, Dudoit S, Luu P, Speed TP (2001). Normalization for cDNA microarray data. In *Microarrays: Optical Technologies and Informatics*, M. L. Bittner, Y. Chen, A. N. Dorsel, and E. R. Dougherty (eds), Proceedings of SPIE, Vol. 4266, pp. 141-152.

Yang YH, Dudoit S, Luu P, Lin DM, Peng V, Ngai J, Speed TP (2002). Normalization for cDNA microarray data: a robust composite method addressing single and multiple slide systematic variation. *Nucleic Acids Research* 30(4), e15.

See Also

An overview of limma functions for normalization is given in 05.Normalization. In particular, see normalizeBetweenArrays for between-array normalization.

The original loess normalization function was the statma function in the sma package. normalizeWithinArrays is a direct generalization of that function, with more options and with support for quantitative spot quality weights.

A different implementation of loess normalization methods, with potentially different behavior, is provided by the maNorm function in the marray package.

normexp.fit *Fit Normal+Exp Convolution Model to Observed Intensities*

Description

Fit the normal+exponential convolution model to a vector of observed intensities. The normal part represents the background and the exponential part represents the signal intensities. This function is called by backgroundCorrect and is not normally called directly by users.

Usage

```
normexp.fit(x, method="saddle", n.pts=NULL, trace=FALSE)
```

Arguments

x	numeric vector of (background corrected) intensities
method	method used to estimate the three parameters. Choices for normexp.fit are "mle", "saddle", "rma" and "rma75".
n.pts	number of quantiles of x to use for the fit. If NULL then all values of x will be used.
trace	logical, if TRUE, tracing information on the progress of the optimization is given.

Details

The Normal+Exp (normexp) convolution model is a mathematical model representing microarray intensity data for the purposes of background correction. It was proposed originally as part of the RMA algorithm for Affymetrix microarray data. For two-color microarray data, the normexp background correction method was introduced and compared with other methods by Ritchie et al (2007).

This function uses maximum likelihood estimation to fit the normexp model to background-corrected intensities. The model assumes that the observed intensities are the sum of background and signal components, the background being normal and the signal being exponentially distributed.

The likelihood may be computed exactly (method="mle") or approximated using a saddle-point approximation (method="saddle"). The saddle-point approximation was proposed by Ritchie et al (2007). Silver et al (2008) added some computational refinements to the saddle-point approximation, making it more reliable in practice, and developed the exact likelihood maximization algorithm. The "mle" method uses the best performing algorithm from Silver et al (2008), which calls the optimization function nlminb with analytic first and second derivatives. Derivatives are computed with respect to the normal-mean, the log-normal-variance and the log-exponential-mean.

Two ad-hoc estimators are also available which do not require iterative estimation. "rma" results in a call to the bg.parameters function of the affy package. This provides the kernel estimation method that is part of the RMA algorithm for Affymetrix data. "rma75" uses the similar but less biased RMA-75 method from McGee and Chen (2006).

If the length x is very large, it may be worth saving computation time by setting n.pts to a value less than the total number of probes, for example n.pts=2^14.

Value

A list containing the components

par	numeric vector giving estimated values of the mean and log-standard-deviation of the background-normal part and the log-mean of the signal-exponential part.
m2loglik	numeric scalar giving minus twice the maximized log-likelihood
convergence	integer code indicating successful convergence or otherwise of the optimization.

Author(s)

Gordon Smyth and Jeremy Silver

References

McGee M, Chen Z (2006). Parameter estimation for the exponential-normal convolution model for background correction of Affymetrix GeneChip data. *Statistical Applications in Genetics and Molecular Biology* 5(1), Article 24.

Ritchie ME, Silver J, Oshlack A, Holmes M, Diyagama D, Holloway A, Smyth GK (2007). A comparison of background correction methods for two-colour microarrays. *Bioinformatics* 23, 2700-2707. doi:10.1093/bioinformatics/btm412.

Silver JD, Ritchie ME, Smyth GK (2009). Microarray background correction: maximum likelihood estimation for the normal-exponential convolution. *Biostatistics* 10, 352-363. doi:10.1093/biostatistics/kxn042.

See Also

normexp.signal, normexp.fit.control. Also bg.parameters in the affy package.

An overview of background correction functions is given in 04.Background.

Examples

```
x <- c(2,3,1,10,3,20,5,6)
out <- normexp.fit(x)
normexp.signal(out$par, x=x)
```

---

normexp.fit.control *Normexp Model Parameter Estimation Aided by Negative Controls*

Description

The mean and log-standard-deviation of the background-normal part of the normal+exponential convolution model are estimated as the mean and log-standard deviation of intensities from negative control probes. The log-mean of the signal-exponential part is estimated as the log of the difference between signal mean and background mean.

Usage

```
normexp.fit.control(x, status=NULL, negctrl="negative", regular="regular", robust=FALSE)
```

Arguments

x	object of class EListRaw-class or matrix containing raw intensities for regular and control probes for a series of microarrays
status	character vector giving probe types.
negctrl	character string identifier for negative control probes.
regular	character string identifier for regular probes.
robust	logical. Should robust estimators be used for the background mean and standard deviation?

Details

x has to contain raw expression intensities from both regular probes and negative control probes.

The probe type information for an object of EListRaw-class is normally saved in the Status column of its genes component. However, it will be overriden by the status parameter if it is explicitly provided to this function. If x is a matrix object, the probe type information has to be provided through the status parameter of this function. Regular probes have the status regular. Negative control probes have the status indicated by negctrl, which is negative by default.

This function estimates parameters of the normal+exponential convolution model with the help of negative control probes. The mean and log-standard-deviation of the background-normal part of the normal+exponential(normexp) convolution model are estimated as the mean and log-standard deviation of intensities from negative control probes respectively. The log-mean of the signal-exponential part is estimated as the log of the difference between signal mean and background mean. The signal mean is simply the mean of intensities from regular probes.

When negative control probes are not available, the normexp.fit.detection.p function can be used to estimate the normexp model parameters which infers the negative control probe intensities from regular probes by taking advantage of their detection p value information.

Value

A matrix containing estimated parameters with rows being arrays and with columns being parameters. Column names are mu, logsigma and logalpha.

Author(s)

Wei Shi and Gordon Smyth

References

Shi W, Oshlack A, Smyth GK (2010). Optimizing the noise versus bias trade-off for Illumina Whole Genome Expression BeadChips. *Nucleic Acids Research* 38(22), e204. doi:10.1093/nar/gkq871.

See Also

nec calls this function to get the parameters of the normal+exponential convolution model and then calls normexp.signal to perform the background correction.

normexp.fit.detection.p estimates the parameters in the normal+exponential convolution model using negative control probe intensities inferred from regular probes by using their detection p-value information.

normexp.fit estimates normexp parameters using a saddle-point approximation or other methods.

An overview of background correction functions is given in 04.Background.

Examples

```
## Not run:
# read in BeadChip probe profile file and control profile file
x <- read.ilmn(files="sample probe profile", ctrlfiles="control probe profile")
# estimated normexp parameters
normexp.fit.control(x)
# normalization using control data
y <- neqc(x)

## End(Not run)
```

---

normexp.fit.detection.p

Estimate Normexp Model Parameter Using Negative Controls Inferred from Regular Probes

---

Description

Detection p values from Illumina BeadChip microarray data can be used to infer negative control probe intensities from regular probe intensities by using detection p value information when negative control data are not available. The inferred negative control intensities can then be used in the background correction in the same way as those control data outputted from BeadChip used in the normexp.fit.control function.

Usage

```
normexp.fit.detection.p(x, detection.p="Detection")
```

Arguments

x	object of class EListRaw-class or matrix containing raw intensities of regular probes for a series of microarrays
detection.p	a character string giving the name of the component which contains detection p value information in x or a numeric matrix giving detection p values, Detection by default

Details

This function estimates the normexp parameters in the same way as normexp.fit.control does, except that negative control probe intensities are inferred from regular probes by taking advantage of detection p value information rather than from the control probe profile outputted by BeadStudio.

Calculation of detection p values in Illumina BeadChip data is based on the rank of probe intensities in the list of negative control probe intensities. Therefore, the detection p values can be used to find regular probes which have expression intensities falling into the range of negative control probe intensities. These probes give a good approximation to the real negative control data and thus can be used to estimate the mean and standard deviation of background intensities when negative control data is not available.

If x is an EListRaw-class object, this function will try to look for the component which includes detection p value matrix in x when detection.p is a character string. This function assumes that this component is located within the other component in x. The component name specified by detection.p should be exactly the same as the name of the detection p value component in x. If detection.p is a matrix, then this matrix will be used as the detection p value data used in this function.

If x is a matrix object, then detection.p has to be a data matrix which includes detection p values.

When detection.p is a matrix, it has to have the same dimension as that of x.

This function will replace the detection p values with 1 subtracted by these values if high intensity probes have detection p values less than those from low intensity probes.

Note that when control data are available, the normexp.fit.control function should be used instead.

Value

A matrix containing estimated parameters with rows being arrays and with columns being parameters. Column names are mu, logsigma and logalpha.

Author(s)

Wei Shi and Gordon Smyth

References

Shi W, Oshlack A, Smyth GK (2010). Optimizing the noise versus bias trade-off for Illumina Whole Genome Expression BeadChips. *Nucleic Acids Research* 38(22), e204. doi:10.1093/nar/gkq871.

See Also

nec calls this function to get the parameters of the normal+exponential convolution model when control probe profile file is not available and then calls normexp.signal to perform the background correction.

normexp.fit.control estimates normexp parameters using control data outputted by BeadStudio.

normexp.fit estimates normexp parameters using a saddle-point approximation or other methods.

An overview of background correction functions is given in 04.Background.

Examples

```
## Not run:
# read in BeadChip data which do not have control data available
x <- read.ilmn(files="sample probe profile")
# estimated normexp parameters
normexp.fit.detection.p(x)
# normalization using inferred negative controls
y <- neqc(x)

## End(Not run)
```

`normexp.signal` *Expected Signal Given Observed Foreground Under Normal+Exp Model*

Description

Adjust foreground intensities for observed background using Normal+Exp Model. This function is called by `backgroundCorrect` and is not normally called directly by the user.

Usage

```
normexp.signal(par, x)
```

Arguments

`par`	numeric vector containing the parameters of the Normal+Exp distribution, see `normexp.fit` for details.
`x`	numeric vector of (background corrected) intensities

Details

In general the vector `normmean` is computed conditional on background at each spot.

Value

Numeric vector containing adjusted intensities.

Author(s)

Gordon Smyth

References

Ritchie ME, Silver J, Oshlack A, Holmes M, Diyagama D, Holloway A, Smyth GK (2007). A comparison of background correction methods for two-colour microarrays. *Bioinformatics* 23, 2700-2707. doi:10.1093/bioinformatics/btm412.

Silver JD, Ritchie ME, Smyth GK (2009). Microarray background correction: maximum likelihood estimation for the normal-exponential convolution. *Biostatistics* 10, 352-363. doi:10.1093/biostatistics/kxn042.

See Also

normexp.fit

An overview of background correction functions is given in 04.Background.

Examples

```
# See normexp.fit
```

---

plotDensities *Plot Expression Densities*

Description

Plot the density of expression values for multiple arrays on the same plot.

Usage

```
## S3 method for class 'RGList'
plotDensities(object, log=TRUE, group=NULL, col=NULL, main="RG Densities",
              bc.method="subtract", ...)
## S3 method for class 'MAList'
plotDensities(object, log=TRUE, group=NULL, col=NULL, main="RG Densities", ...)
## S3 method for class 'EListRaw'
plotDensities(object, log=TRUE, bc.method="subtract", ...)
## S3 method for class 'EList'
plotDensities(object, log=TRUE, ...)
## Default S3 method:
plotDensities(object, group=NULL, col=NULL, main=NULL, legend="topleft", ...)
```

Arguments

object	an RGList, MAList, EListRaw or EList object containing expression data. Or any data object that can be coerced to a matrix.
log	logical, should densities be plotted on the log2 scale?
group	optional vector or factor classifying the arrays into groups. Should be same length as ncol(object).
col	optional vector of colors of the same length as the number of groups.
main	the main title for the plot.
bc.method	background subtraction method passed to backgroundCorrect.
legend	character string giving position to place legend. See legend for possible values. Can also be logical, with FALSE meaning no legend.
...	other arguments are passed to plotDensities.default or density.

Details

This function is useful to display and contrast the distribution of expression values on different arrays. It can for example be used to display the effects of between-array normalization. See the section on between-array normalization in the LIMMA User's Guide.

Value

A plot is created on the current graphics device.

Author(s)

Natalie Thorne and Gordon Smyth

See Also

An overview of diagnostic plots in LIMMA is given in 09.Diagnostics. There is a section using `plotDensities` in conjunction with between-array normalization in the LIMMA User's Guide.

This function uses `density` and `matplot`.

Examples

```
## Not run:
#  Default is to plot red channels in red and green channels in green
plotDensities(MA)

#  Alternatively colors
plotDensities(MA, col=c("red","blue"))

#  Color by group, with three groups:
plotDensities(MA,group=group,col=c("blue","orange","green"))

## End(Not run)
```

---

plotExonJunc *Differential splicing plot with junctions*

Description

Plot differential usage results by exons and junctions for the specified gene and highlight the significantly spliced exons and junctions.

Usage

```
plotExonJunc(fit, coef=ncol(fit), geneid, genecolname=NULL, FDR=0.05, annotation=NULL)
```

Arguments

fit	MArrayLM fit object produced by diffSplice. Must have the Entrez gene ids for all the exons and junctions stored in fit$genes$GeneID, length information for all the exons and junctions stored in fit$genes$Length and the strand information stored in fit$genes$Strand. To distinguish between exons and junctions, fit$genes$Length has to be set to 1 for all the junctions.
coef	the coefficient (column) of fit for which differential splicing is assessed.
geneid	character string, ID of the gene to plot.
genecolname	column name of fit$genes containing geneid.
FDR	numeric, highlight exons and junctions with false discovery rate less than this cutoff. Red indicates up-regulation whereas blue indicates down-regulation. The FDR of the individual exon/junction is calculated based on the exon-level t-statistics test for differences between each exon/junction and all other exons/junctions for the same gene.
annotation	data frame containing the full exon annotation of the corresponding species. Must have the Entrez gene ids for all the exons stored in the GeneID column, start and end positions for all the exons stored in the Start and End columns, respectively.

Details

Plot differential usage results by exons and junctions for the specified gene. The significantly spliced individual exons are highlighted as red blocks if up-regulated and blue blocks if down-regulated. All other exons are displayed as black blocks. The significantly spliced individual junctions are highlighted as red lines if up-regulated and blue lines if down-regulated. All other junctions are displayed as black lines.

Since the diffSplice analysis is usually performed after filtering, the full annotation (e.g. the in-built annotation in featureCounts) is highly recommended for producing the plot. When annotation is provided, the filtered exons are displayed as grey blocks.

Value

A plot is created on the current graphics device.

Author(s)

Yunshun Chen and Gordon Smyth

See Also

diffSplice, topSplice

Examples

```
## Not run:
# diffSplice analysis
v <- voom(dge, design)
```

```
fit <- lmFit(v, design)
ex <- diffSplice(fit, geneid="GeneID")

# Get full annotation from Rsubread
library(Rsubread)
annotation.full <- getInBuiltAnnotation("mm10")

# Make a plot
plotExonJunc(ex, geneid="Foxp1", genecolname="Symbol", annotation=annotation.full)

## End(Not run)
```

---

plotExons *Plot exons of differentially expressed gene*

Description

Plot exons of differentially expressed gene and mark the differentially expressed exons.

Usage

```
plotExons(fit, coef = ncol(fit), geneid = NULL, genecolname = "GeneID",
          exoncolname = NULL, rank = 1L, FDR = 0.05)
```

Arguments

fit	MArrayLM fit object produced by eBayes.
coef	the coefficient (column) of fit for which differential expression is assessed.
geneid	character string, ID of the gene to plot.
genecolname	character string for the column name of fit$genes containing gene IDs. Defaults to "GeneID" for Entrez Gene ID.
exoncolname	character string for the column name of fit$genes containing exon IDs.
rank	integer, if geneid=NULL then this ranked gene will be plotted.
FDR	numeric, mark differentially expressed exons with false discovery rate less than this cutoff.

Details

Plots log2-fold-change by exon for the specified gene and highlights the differentially expressed exons. Shows annotations such as GeneID, Symbol and Strand, if available, as the plot title. The significantly differentially expressed individual exons are highlighted as red dots for up-regulation and as blue dots for down-regulation. The size of the dots is weighted by their significance.

Value

A plot is created on the current graphics device.

Author(s)

Yifang Hu and Gordon Smyth

See Also

lmFit, eBayes, plotSplice

A summary of functions available in LIMMA for RNA-seq analysis is given in 11.RNAseq.

Examples

```
## Not run:
fit <- lmFit(y,design)
fit <- eBayes(fit)
plotExons(fit)
plotExons(fit, exoncolname = "Start", rank = 1)
plotExons(fit, geneid = "ps", genecolname = "Symbol", exoncolname = "Start")

## End(Not run)
```

---

plotFB *FB-Plot*

Description

Creates foreground-background plots.

Usage

```
## S3 method for class 'RGList'
plotFB(x, array = 1, lim = "separate", pch = 16, cex = 0.3,
   xlab = "log2 Background", ylab = "log2 Foreground", main = colnames(x)[array], ...)
## S3 method for class 'EListRaw'
plotFB(x, array = 1, pch = 16, cex=0.3,
   xlab = "log2 Background", ylab = "log2 Foreground", main = colnames(x)[array], ...)
```

Arguments

x	an RGList or EListRaw object.
array	integer giving the array to be plotted.
lim	character string indicating whether the red and green plots should have "separate" or "common" x- and y- co-ordinate limits.
pch	vector or list of plotting characters. Defaults to integer code 16.
cex	numeric vector of plot symbol expansions.
xlab	character string, label for x-axis.
ylab	character string, label for y-axis.
main	character string, title for plot.
...	any other arguments are passed to plot.

Details

A foreground-background plot is a plot of log2-foreground vs log2-background for a particular array. For two-color arrays, this function produces a pair of plots, one for the green channel and one for the red.

See `points` for possible values for `pch`, `col` and `cex`.

Value

A plot is created on the current graphics device.

Author(s)

Gordon Smyth

See Also

An overview of diagnostic functions available in LIMMA is given in 09.Diagnostics.

`plotlines` *plotlines*

Description

Time course style plot of expression data.

Usage

```
plotlines(x,first.column.origin=FALSE,xlab="Column",ylab="x",col="black",lwd=1,...)
```

Arguments

`x`	numeric matrix or object containing expression data.
`first.column.origin`	logical, should the lines be started from zero?
`xlab`	x-axis label
`ylab`	y-axis label
`col`	vector of colors for lines
`lwd`	line width multiplier
`...`	any other arguments are passed to `plot`

Details

Plots a line for each probe.

Value

A plot is created on the current graphics device.

Author(s)

Gordon Smyth

See Also

An overview of modeling functions and associated plots available in LIMMA is given in 06.Linear-Models.

---

plotMA *MA-Plot of Expression Data*

Description

Creates an MA-plot with color coding for control spots.

Usage

```
## Default S3 method:
plotMA(object, array = 1, xlab = "Average log-expression",
       ylab = "Expression log-ratio (this sample vs others)",
       main = colnames(object)[array], status=NULL, ...)
## S3 method for class 'EList'
plotMA(object, array = 1, xlab = "Average log-expression",
       ylab = "Expression log-ratio (this sample vs others)",
       main = colnames(object)[array], status=object$genes$Status,
       zero.weights = FALSE, ...)
## S3 method for class 'RGList'
plotMA(object, array = 1, xlab = "A", ylab = "M",
       main = colnames(object)[array], status=object$genes$Status,
       zero.weights = FALSE, ...)
## S3 method for class 'MAList'
plotMA(object, array = 1, xlab = "A", ylab = "M",
       main = colnames(object)[array], status=object$genes$Status,
       zero.weights = FALSE, ...)
## S3 method for class 'MArrayLM'
plotMA(object, coef = ncol(object), xlab = "Average log-expression",
       ylab = "log-fold-change", main = colnames(object)[coef],
       status=object$genes$Status, ...)
```

Arguments

object	an RGList, MAList, EList, ExpressionSet or MArrayLM object. Alternatively a numeric matrix.
array	integer giving the array to be plotted.
coef	integer giving the linear model coefficient to be plotted.
xlab	character string, label for x-axis
ylab	character string, label for y-axis
main	character string, title for plot
status	vector giving the control status of each spot on the array, of same length as the number of rows of object. If NULL, then all points are plotted in the default color, symbol and size.
zero.weights	logical, should spots with zero or negative weights be plotted?
...	other arguments are passed to plotWithHighlights.

Details

An MA-plot is a plot of log-intensity ratios (M-values) versus log-intensity averages (A-values). See Ritchie et al (2015) for a brief historical review.

For two color data objects, a within-array MA-plot is produced with the M and A values computed from the two channels for the specified array. This is the same as a mean-difference plot (mdplot) with the red and green log2-intensities of the array providing the two columns.

For single channel data objects, a between-array MA-plot is produced. An artificial array is produced by averaging all the arrays other than the array specified. A mean-difference plot is then produced from the specified array and the artificial array. Note that this procedure reduces to an ordinary mean-difference plot when there are just two arrays total.

If object is an MArrayLM object, then the plot is a fitted model MA-plot in which the estimated coefficient is on the y-axis and the average A-value is on the x-axis.

The status vector can correspond to any grouping of the probes that is of interest. If object is a fitted model object, then the status vector is often used to indicate statistical significance, so that differentially expressed points are highlighted. If object is a microarray data object, then status might distinguish control probes from regular probes so that different types of controls are highlighted.

The status can be included as the component object$genes$Status instead of being passed as an argument to plotMA.

See plotWithHighlights for how to set colors and graphics parameters for the highlighted and non-highlighted points.

Value

A plot is created on the current graphics device.

Note

The plotMD function provides the same functionality as plotMA with slightly different arguments.

Author(s)

Gordon Smyth

References

Ritchie ME, Phipson B, Wu D, Hu Y, Law CW, Shi W, Smyth GK (2015). limma powers differential expression analyses for RNA-sequencing and microarray studies. *Nucleic Acids Research* Volume 43, e47. doi:10.1093/nar/gkv007.

See Also

The driver function for `plotMA` is `plotWithHighlights`.

An overview of plot functions available in LIMMA is given in 09.Diagnostics.

Examples

```
A <- runif(1000,4,16)
y <- A + matrix(rnorm(1000*3,sd=0.2),1000,3)
status <- rep(c(0,-1,1),c(950,40,10))
y[,1] <- y[,1] + status
plotMA(y, array=1, status=status, values=c(-1,1), hl.col=c("blue","red"))

MA <- new("MAList")
MA$A <- runif(300,4,16)
MA$M <- rt(300,df=3)

# Spike-in values
MA$M[1:3] <- 0
MA$M[4:6] <- 3
MA$M[7:9] <- -3

status <- rep("Gene",300)
status[1:3] <- "M=0"
status[4:6] <- "M=3"
status[7:9] <- "M=-3"
values <- c("M=0","M=3","M=-3")
col <- c("blue","red","green")

plotMA(MA,main="MA-Plot with 12 spiked-in points",
       status=status, values=values, hl.col=col)

#  Same as above but setting graphical parameters as attributes
attr(status,"values") <- values
attr(status,"col") <- col
plotMA(MA, main="MA-Plot with 12 spiked-in points", status=status)

#  Same as above but passing status as part of object
MA$genes$Status <- status
plotMA(MA, main="MA-Plot with 12 spiked-in points")

#  Change settings for background points
```

```
MA$genes$Status <- status
plotMA(MA, bg.pch=1, bg.cex=0.5)
```

---

plotMA3by2 *Write MA-Plots to Files*

Description

Write MA-plots to files in PNG format, six plots to a file in a 3 by 2 grid arrangement.

Usage

```
plotMA3by2(object, prefix="MA", path=NULL, main=colnames(object),
           zero.weights=FALSE, common.lim=TRUE, device="png", ...)
```

Arguments

object	an MAList, RGList, EListRaw or EList object, or a matrix containing log-intensities.
prefix	character string giving prefix to attach to file names
path	character string specifying directory for output files
main	character vector giving titles for plots
zero.weights	logical, should points with non-positive weights be plotted
common.lim	logical, should all plots on a page use the same axis limits
device	device driver for the plot. Choices are "png", "jpeg", "pdf", "postscript".
...	any other arguments are passed to plotMA

Details

This function writes a series of graphic files to disk. Each file contains six MA-plots in three rows and two columns. The layout is optimized for A4-sized paper.

The graph format can be "png" or "jpeg", which are screen-resolution formats, or "pdf" or "postscript", which are loss-less formats. "png" is not available on every R platform. Note that "pdf" or "postscript" may produce very large files.

Value

No value is returned, but one or more files are written to the working directory. The number of files is determined by the number of columns of object.

Author(s)

Gordon Smyth

See Also

plotMA

An overview of diagnostic functions available in LIMMA is given in 09.Diagnostics.

---

plotMD *Mean-Difference Plot of Expression Data*

Description

Creates a mean-difference plot (aka MA plot) with color coding for highlighted points.

Usage

```
## Default S3 method:
plotMD(object, column = 1, xlab = "Average log-expression",
       ylab = "Expression log-ratio (this sample vs others)",
       main = colnames(object)[column], status=NULL, ...)
## S3 method for class 'EList'
plotMD(object, column = 1, array = NULL, xlab = "Average log-expression",
       ylab = "Expression log-ratio (this sample vs others)",
       main = colnames(object)[column], status=object$genes$Status,
       zero.weights = FALSE, ...)
## S3 method for class 'RGList'
plotMD(object, column = 1, array = NULL, xlab = "A", ylab = "M",
       main = colnames(object)[column], status=object$genes$Status,
       zero.weights = FALSE, ...)
## S3 method for class 'MAList'
plotMD(object, column = 1, array = NULL, xlab = "A", ylab = "M",
       main = colnames(object)[column], status=object$genes$Status,
       zero.weights = FALSE, ...)
## S3 method for class 'MArrayLM'
plotMD(object, column = ncol(object), coef = NULL, xlab = "Average log-expression",
       ylab = "log-fold-change", main = colnames(object)[column],
       status=object$genes$Status, ...)
```

Arguments

object	an RGList, MAList, EList, ExpressionSet or MArrayLM object. Alternatively a numeric matrix.
column	integer, column of object to be plotted.
array	alternative to column for microarray data objects. If specified, then column is ignored.
coef	alternative to column for fitted model objects. If specified, then column is ignored.
xlab	character string, label for x-axis.

ylab	character string, label for y-axis.
main	character string, title for plot.
status	vector giving the control status of each spot on the array, of same length as the number of rows of object. If NULL, then all points are plotted in the default color, symbol and size.
zero.weights	logical, should spots with zero or negative weights be plotted?
...	other arguments are passed to plotWithHighlights.

Details

A mean-difference plot (MD-plot) is a plot of log-intensity ratios (differences) versus log-intensity averages (means). For two color data objects, a within-array MD-plot is produced with the M and A values computed from the two channels for the specified array. This is the same as a mean-difference plot (mdplot) with the red and green log2-intensities of the array providing the two columns.

For single channel data objects, a between-array MD-plot is produced. An artificial array is produced by averaging all the arrays other than the array specified. A mean-difference plot is then produced from the specified array and the artificial array. Note that this procedure reduces to an ordinary mean-difference plot when there are just two arrays total.

If object is an MArrayLM object, then the plot is a fitted model MD-plot in which the estimated coefficient is on the y-axis and the average A-value is on the x-axis.

The status vector can correspond to any grouping of the probes that is of interest. If object is a fitted model object, then the status vector is often used to indicate statistical significance, so that differentially expressed points are highlighted. If object is a microarray data object, then status might distinguish control probes from regular probes so that different types of controls are highlighted.

The status can be included as the component object$genes$Status instead of being passed as an argument to plotMD.

See plotWithHighlights for how to set colors and graphics parameters for the highlighted and non-highlighted points.

Value

A plot is created on the current graphics device.

Note

This function is an alternative to plotMA, which was one of the original functions of the limma package in 2002. The history of mean-difference plots and MA-plots is reviewed in Ritchie et al (2015).

Author(s)

Gordon Smyth

References

Ritchie ME, Phipson B, Wu D, Hu Y, Law CW, Shi W, Smyth GK (2015). limma powers differential expression analyses for RNA-sequencing and microarray studies. *Nucleic Acids Research* Volume 43, e47. doi:10.1093/nar/gkv007.

See Also

The driver function for `plotMD` is `plotWithHighlights`. See also `mdplot` for a very basic mean-difference plot function.

An overview of plot functions available in LIMMA is given in 09.Diagnostics.

Examples

```
A <- runif(1000,4,16)
y <- A + matrix(rnorm(1000*3,sd=0.2),1000,3)
status <- rep(c(0,-1,1),c(950,40,10))
y[,1] <- y[,1] + status
plotMD(y, column=1, status=status, values=c(-1,1), hl.col=c("blue","red"))

MA <- new("MAList")
MA$A <- runif(300,4,16)
MA$M <- rt(300,df=3)

# Spike-in values
MA$M[1:3] <- 0
MA$M[4:6] <- 3
MA$M[7:9] <- -3

status <- rep("Gene",300)
status[1:3] <- "M=0"
status[4:6] <- "M=3"
status[7:9] <- "M=-3"
values <- c("M=0","M=3","M=-3")
hl.col <- c("blue","red","green3")

plotMD(MA,main="MA-Plot with 12 spiked-in points",
       status=status, values=values, hl.col=hl.col)

#  Same as above but setting graphical parameters as attributes
attr(status,"values") <- values
attr(status,"col") <- hl.col
plotMD(MA, main="Mean-Difference Plot with 12 spiked-in points", status=status)

#  Same as above but passing status as part of object
MA$genes$Status <- status
plotMD(MA, main="Mean-Difference Plot with 12 spiked-in points")

#  Change settings for background points
MA$genes$Status <- status
plotMD(MA, bg.pch=1, bg.cex=0.5)
```

plotMDS *Multidimensional Scaling Plot of Distances Between Expression Profiles*

Description

Plot samples on a two-dimensional scatterplot so that distances on the plot approximate the typical log2 fold changes between the samples.

Usage

```
## Default S3 method:
plotMDS(x, top = 500, labels = NULL, pch = NULL, cex = 1,
     dim.plot = c(1,2), gene.selection = "pairwise",
     xlab = NULL, ylab = NULL, plot = TRUE, var.explained = TRUE, ...)
## S3 method for class 'MDS'
plotMDS(x, labels = NULL, pch = NULL, cex = 1, dim.plot = NULL,
     xlab = NULL, ylab = NULL, var.explained = TRUE, ...)
```

Arguments

x	any data object that can be coerced to a matrix of log-expression values, for example an ExpressionSet or an EList. Rows represent genes or genomic features while columns represent samples.
top	number of top genes used to calculate pairwise distances.
labels	character vector of sample names or labels. Defaults to colnames(x).
pch	plotting symbol or symbols. See points for possible values. Ignored if labels is non-NULL.
cex	numeric vector of plot symbol expansions.
dim.plot	integer vector of length two specifying which principal components should be plotted.
gene.selection	character, "pairwise" to choose the top genes separately for each pairwise comparison between the samples or "common" to select the same genes for all comparisons.
xlab	title for the x-axis.
ylab	title for the y-axis.
plot	logical. If TRUE then a plot is created on the current graphics device.
var.explained	logical. If TRUE then the percentage variation explained is included in the axis labels.
...	any other arguments are passed to plot, and also to text (if pch is NULL).

Details

This function uses multidimensional scaling (MDS) to produce a principal coordinate (PCoA) or principal component (PCA) plot showing the relationships between the expression profiles represented by the columns of `x`. If `gene.selection = "common"`, or if the `top` is equal to or greater than the number of rows of `x`, then a PCA plot is constructed from the `top` genes with largest standard deviations across the samples.

If `gene.selection = "pairwise"` and `top` is less than `nrow(x)` then a PCoA plot is produced and distances on the plot represent the *leading log2-fold-changes*. The leading log-fold-change between a pair of samples is defined as the root-mean-square average of the `top` largest log2-fold-changes between those two samples. The PCA and PCoA plots produced by `gene.selection="common"` and `gene.selection="pairwise"`, respectively, use similar distance measures but the PCA plot uses the same genes throughout whereas the PCoA plot potentially selects different genes to distinguish each pair of samples. The pairwise choice is the default. It potentially gives better resolution than a PCA plot if different molecular pathways are relevant for distinguishing different pairs of samples.

If `pch=NULL`, then each sample is represented by a text label, defaulting to the column names of `x`. If `pch` is not `NULL`, then plotting symbols are used.

See `text` for possible values for `col` and `cex`.

Value

If `plot=TRUE` or if `x` is an object of class `"MDS"`, then a plot is created on the current graphics device.

An object of class `"MDS"` is also invisibly returned. This is a list containing the following components:

`eigen.values`	eigen values
`eigen.vectors`	eigen vectors
`var.explained`	proportion of variance explained by each dimension
`distance.matrix.squared`	numeric matrix of squared pairwise distances between columns of `x`
`dim.plot`	dimensions plotted
`x`	x-coordinates of plotted points
`y`	y-coordinates of plotted points
`gene.selection`	gene selection method

Author(s)

Di Wu and Gordon Smyth

References

Ritchie ME, Phipson B, Wu D, Hu Y, Law CW, Shi W, Smyth GK (2015). limma powers differential expression analyses for RNA-sequencing and microarray studies. *Nucleic Acids Research* 43, e47. doi:10.1093/nar/gkv007.

See Also

An overview of diagnostic functions available in LIMMA is given in 09.Diagnostics.

Examples

```
# Simulate gene expression data for 1000 probes and 6 microarrays.
# Samples are in two groups
# First 50 probes are differentially expressed in second group
sd <- 0.3*sqrt(4/rchisq(1000,df=4))
ExprMatrix <- matrix(rnorm(1000*6,sd=sd),1000,6)
rownames(ExprMatrix) <- paste("Gene",1:1000)
ExprMatrix[1:50,4:6] <- ExprMatrix[1:50,4:6] + 2
# without labels, indexes of samples are plotted.
mds <- plotMDS(ExprMatrix,  col=c(rep("black",3), rep("red",3)) )
# or labels can be provided, here group indicators:
plotMDS(mds,  col=c(rep("black",3), rep("red",3)), labels= c(rep("Grp1",3), rep("Grp2",3)))
```

---

plotPrintTipLoess *MA Plots by Print-Tip Group*

Description

Creates a coplot giving MA-plots with loess curves by print-tip groups.

Usage

```
plotPrintTipLoess(object,layout,array=1,span=0.4,...)
```

Arguments

object	MAList or RGList object or list with components M containing log-ratios and A containing average intensities
layout	a list specifying the number of tip rows and columns and the number of spot rows and columns printed by each tip. Defaults to MA$printer if that is non-null.
array	integer giving the array to be plotted. Corresponds to columns of M and A.
span	span of window for lowess curve
...	other arguments passed to panel.smooth

Details

Note that spot quality weights in object are not used for computing the loess curves for this plot even though such weights would be used for loess normalization using normalizeWithinArrays.

Value

A plot is created on the current graphics device. If there are missing values in the data, then the vector of row numbers for spots with missing values is invisibly returned, as for coplot.

Author(s)

Gordon Smyth

See Also

An overview of diagnostic functions available in LIMMA is given in 09.Diagnostics.

`plotRLDF` *Plot of regularized linear discriminant functions for microarray data*

Description

Plot regularized linear discriminant functions for classifying samples based on expression data.

Usage

```
plotRLDF(y, design = NULL, z = NULL, nprobes = 100, plot = TRUE,
         labels.y = NULL, labels.z = NULL, pch.y = NULL, pch.z = NULL,
         col.y = "black", col.z = "black",
         show.dimensions = c(1,2), ndim = max(show.dimensions),
         var.prior = NULL, df.prior = NULL, trend = FALSE, robust = FALSE, ...)
```

Arguments

`y`	the training dataset. Can be any data object which can be coerced to a matrix, such as `ExpressionSet` or `EList`.
`design`	design matrix defining the training groups to be distinguished. The first column is assumed to represent the intercept. Defaults to `model.matrix(~factor(labels.y))`.
`z`	the dataset to be classified. Can be any data object which can be coerced to a matrix, such as `ExpressionSet` or `EList`. Rows must correspond to rows of `y`.
`nprobes`	number of probes to be used for the calculations. The probes will be selected by moderated F statistic.
`plot`	logical, should a plot be created?
`labels.y`	character vector of sample names or labels in `y`. Defaults to `colnames(y)` or failing that to `1:n`.
`labels.z`	character vector of sample names or labels in `z`. Defaults to `colnames(z)` or failing that to `letters[1:n]`.
`pch.y`	plotting symbol or symbols for `y`. See `points` for possible values. Takes precedence over `labels.y` if both are specified.
`pch.z`	plotting symbol or symbols for `z`. See `points` for possible values. Takes precedence over `labels.z` if both are specified.
`col.y`	colors for the plotting `labels.y`.
`col.z`	colors for the plotting `labels.z`.

show.dimensions
: integer vector of length two indicating which two discriminant functions to plot. Functions are in decreasing order of discriminatory power.

ndim
: number of discriminant functions to compute

var.prior
: prior variances, for regularizing the within-group covariance matrix. By default is estimated by squeezeVar.

df.prior
: prior degrees of freedom for regularizing the within-group covariance matrix. By default is estimated by squeezeVar.

trend
: logical, should a trend be estimated for var.prior? See eBayes for details. Only used if var.prior or df.prior are NULL.

robust
: logical, should var.prior and df.prior be estimated robustly? See eBayes for details. Only used if var.prior or df.prior are NULL.

...
: any other arguments are passed to plot.

Details

The function builds discriminant functions from the training data (y) and applies them to the test data (z). The method is a variation on classical linear discriminant functions (LDFs), in that the within-group covariance matrix is regularized to ensure that it is invertible, with eigenvalues bounded away from zero. The within-group covariance matrix is squeezed towards a diagonal matrix with empirical Bayes posterior variances as diagonal elements.

The calculations are based on a filtered list of probes. The nprobes probes with largest moderated F statistics are used to discriminate.

The ndim argument allows all required LDFs to be computed even though only two are plotted.

Value

If plot=TRUE a plot is created on the current graphics device. A list containing the following components is (invisibly) returned:

training
: numeric matrix with ncol(y) rows and ndim columns containing discriminant functions evaluated for the training data.

predicting
: numeric matrix with ncol(z) rows and ndim columns containing discriminant functions evaluated on the classification data.

top
: integer vector of length nprobes giving indices of probes used.

metagenes
: numeric matrix with nprobes rows and ndim columns containing probe weights defining each discriminant function.

singular.values
: numeric vector of singular values showing the predictive power of each discriminant function.

rank
: maximum number of discriminant functions with singular.values greater than zero.

var.prior
: numeric vector of prior variances.

df.prior
: numeric vector of prior degrees of freedom.

Note

The default values for `df.prior` and `var.prior` were changed in limma 3.27.10. Previously these were preset values. Now the default is to estimate them using `squeezeVar`.

Author(s)

Gordon Smyth, Di Wu and Yifang Hu

See Also

`lda` in package `MASS`

Examples

```
# Simulate gene expression data for 1000 probes and 6 microarrays.
# Samples are in two groups
# First 50 probes are differentially expressed in second group
sd <- 0.3*sqrt(4/rchisq(1000,df=4))
y <- matrix(rnorm(1000*6,sd=sd),1000,6)
rownames(y) <- paste("Gene",1:1000)
y[1:50,4:6] <- y[1:50,4:6] + 2

z <- matrix(rnorm(1000*6,sd=sd),1000,6)
rownames(z) <- paste("Gene",1:1000)
z[1:50,4:6] <- z[1:50,4:6] + 1.8
z[1:50,1:3] <- z[1:50,1:3] - 0.2

design <- cbind(Grp1=1,Grp2vs1=c(0,0,0,1,1,1))
options(digit=3)

# Samples 1-6 are training set, samples a-f are test set:
plotRLDF(y, design, z=z, col.y="black", col.z="red")
legend("top", pch=16, col=c("black","red"), legend=c("Training","Predicted"))
```

---

plotSA *Sigma vs A plot for microarray linear model*

Description

Plot residual standard deviation versus average log expression for a fitted microarray linear model.

Usage

```
plotSA(fit, xlab = "Average log-expression", ylab = "sqrt(sigma)", zero.weights = FALSE,
       pch = 16, cex = 0.3, col = c("black","red"), ...)
```

Arguments

fit	an MArrayLM object.
xlab	label for x-axis
ylab	label for y-axis
zero.weights	logical, should genes with all zero weights be plotted?
pch	vector of codes for plotting characters.
cex	numeric, vector of expansion factors for plotting characters.
col	plotting colors for regular and outlier variances respectively.
...	any other arguments are passed to plot

Details

This plot is used to check the mean-variance relationship of the expression data, after fitting a linear model. A scatterplot of residual-variances vs average log-expression is created. The plot is especially useful for examining the mean-variance trend estimated by eBayes or treat with trend=TRUE. It can be considered as a routine diagnostic plot in the limma-trend pipeline.

If robust empirical Bayes was used to create fit, then outlier variances are highlighted in the color given by col[2].

The y-axis is square-root fit$sigma, where sigma is the estimated residual standard deviation. The y-axis therefore corresponds to quarter-root variances. The y-axis was changed from log2-variance to quarter-root variance in limma version 3.31.21. The quarter-root scale matches the similar plot produced by the voom function and gives a better plot when some of the variances are close to zero.

See points for possible values for pch and cex.

Value

A plot is created on the current graphics device.

Author(s)

Gordon Smyth

See Also

eBayes

An overview of diagnostic functions available in LIMMA is given in 09.Diagnostics.

plotSplice *Differential usage plot*

Description

Plot relative log-fold changes by exons (or transcripts) for the specified gene and highlight differentially used exons (or transcripts).

Usage

```
plotSplice(fit, coef = ncol(fit), geneid = NULL, genecolname = NULL,
           rank = 1L, FDR = 0.05, xlab = "Exon")
```

Arguments

fit	MArrayLM fit object produced by diffSplice.
coef	the coefficient (column) of fit for which differential splicing is assessed.
geneid	character string, ID of the gene to plot.
genecolname	column name of fit$genes containing gene IDs. Defaults to fit$genecolname.
rank	integer, if geneid=NULL then this ranked gene will be plotted.
FDR	numeric, highlight exons as red dots with false discovery rate less than this cut-off. The FDR of the individual exon is calculated based on the exon-level t-statistics test for differences between each exon and all other exons for the same gene.
xlab	x-axis lab for the plot. Typical choices are "Exon" for DEU analyses or "Transcript" for DTU analyses.

Details

Plot relative log2-fold-changes by exon (or transcript) for the specified gene. The relative logFC is the difference between that exon (or transcript)'s logFC and the overall logFC for the gene, as computed by diffSplice. Individual exons (or transcripts) that are significantly differentially used are highlighted as red dots. The size of the red dots increases with the statistical significance of the exon (or transcript).

Value

A plot is created on the current graphics device.

Author(s)

Gordon Smyth and Yifang Hu

References

Baldoni PL, Chen L, Li M, Chen Y, Smyth GK (2025). Dividing out quantification uncertainty enables assessment of differential transcript usage with limma and edgeR. *bioRxiv* doi:10.1101/2025.04.07.647659.

Ritchie ME, Phipson B, Wu D, Hu Y, Law CW, Shi W, Smyth GK (2015). limma powers differential expression analyses for RNA-sequencing and microarray studies. *Nucleic Acids Research* 43, e47. doi:10.1093/nar/gkv007.

See Also

diffSplice, topSplice

A summary of functions available in LIMMA for RNA-seq analysis is given in 11.RNAseq.

Examples

```
# See diffSplice
```

---

plotWithHighlights *Scatterplot With Highlighting of Special Points*

Description

Creates scatterplot, with optional size and color coding for points of special interest. This is the engine for plotMD and plotMA.

Usage

```
plotWithHighlights(x, y, status = NULL, values = NULL,
                   hl.pch = 16, hl.col = NULL, hl.cex = 1, legend = "topright",
                   bg.pch = 16, bg.col = "black", bg.cex = 0.3,
                   pch = NULL, col = NULL, cex = NULL, ...)
```

Arguments

x	numeric vector.
y	numeric vector of same length as x.
status	character vector giving the control status of each point, of same length as x and y, or else a TestResults object with one column and number of rows matching the length of x. If NULL, then all points are plotted in the background color, symbol and size.
values	character vector giving values of status to be highlighted on the plot. Set automatically if status contains TestResults. Defaults to unique values of status in decreasing order of frequency, with the most frequent value set as the background value. Ignored if there is no status vector.
hl.pch	vector of plotting characters for highlighted points, either of unit length or of same length as values. Ignored if there is no status vector.

hl.col	vector of colors for highlighted points, either of unit length or of same length as values. Set automatically if status contains TestResults. Defaults to 1+1:length(values). Ignored if there is no status vector.
hl.cex	numeric vector of plot symbol expansions for highlighted points, either of unit length or of same length as values. Ignored if there is no status vector.
legend	character string giving position to place legend. See legend for possible values. Can also be logical, with FALSE meaning no legend. Ignored if there is no status vector.
bg.pch	plotting character for background (non-highlighted) points.
bg.col	color for background (non-highlighted) points.
bg.cex	plot symbol expansion for background (non-highlighted) points.
pch	synonym for hl.pch allowed for backward compatibility.
col	synonym for hl.col allowed for backward compatibility.
cex	synonym for hl.cex allowed for backward compatibility.
...	other arguments are passed to plot.

Details

This function produces a scatterplot in which the highlighted points are, by default, larger and colored compared to background points.

The status vector establishes the status of each point and values indicates which values of status should be highlighted. If values=NULL, then the most common value of status is assumed to correspond to background points and all other values are highlighted.

The arguments hl.pch, hl.col and hl.cex give graphics settings for highlighted points. By default, highlighted points are larger than background points and a different color is used for each distinct highlighted value.

The arguments bg.pch, bg.col and bg.cex give the graphics settings for non-highlighted (background) points. The same settings are used for all background points.

The arguments values, pch, col and cex can be included as attributes to status instead of being passed as arguments to plotWithHighlights. This is for compatibility with controlStatus.

See points for possible values for the graphics parameters.

Value

A plot is created on the current graphics device.

Author(s)

Gordon Smyth

References

Ritchie ME, Phipson B, Wu D, Hu Y, Law CW, Shi W, Smyth GK (2015). limma powers differential expression analyses for RNA-sequencing and microarray studies. *Nucleic Acids Research* 43, e47. doi:10.1093/nar/gkv007.

See Also

plotMD, plotMA, mdplot

An overview of diagnostic plots available in LIMMA is given in 09.Diagnostics.

Examples

```
x <- runif(1000, min=4, max=16)
status <- rep(c(0,-1,1), c(950,40,10))
y <- status + rnorm(1000, sd=0.2)
plotWithHighlights(x, y, status=status)
```

---

poolVar *Pool Sample Variances with Unequal Variances*

Description

Compute the Satterthwaite (1946) approximation to the distribution of a weighted sum of sample variances.

Usage

```
poolVar(var, df=n-1, multiplier=1/n, n)
```

Arguments

var	numeric vector of independent sample variances.
df	numeric vector of degrees of freedom for the sample variances.
multiplier	numeric vector of multipliers (weights) for the sample variances in the summation.
n	numeric vector of sample sizes.

Details

The sample variances var are assumed to follow scaled chi-square distributions, with specified df but with unknown and generally different scale factors (i.e., the true sample variances). A scaled chi-square approximation is found for the distribution of sum(multiplier * var) by equating first and second moments. On output the sum to be approximated is equal to multiplier * var, which follows approximately a scaled chi-square distribution on df degrees of freedom. The approximation was proposed by Satterthwaite (1946).

The df and multipliers are related to the sample sizes in a common use scenario, but the approximation works for any df and multipliers, which do not necessarily need to be related to sample sizes.

If there are only two groups and the degrees of freedom are one less than the sample sizes then this approximation gives the denominator of Welch's t-test for unequal variances.

Value

A list with components

`var`	effective pooled sample variance.
`df`	effective pooled degrees of freedom.
`multiplier`	pooled multiplier.

Author(s)

Gordon Smyth

References

Welch BL (1938). The significance of the difference between two means when the population variances are unequal. *Biometrika* 29, 350-362.

Satterthwaite FE (1946). An approximate distribution of estimates of variance components. *Biometrics Bulletin* 2, 110-114.

Welch BL (1947). The generalization of 'Student's' problem when several different population variances are involved. *Biometrika* 34, 28-35.

Welch BL (1949). Further note on Mrs Aspin's tables and on certain approximations to the tabled function. *Biometrika* 36, 293-296.

Examples

```
#  Welch's t-test with unequal variances
x <- rnorm(10,mean=1,sd=2)
y <- rnorm(20,mean=2,sd=1)
s2 <- c(var(x),var(y))
n <- c(10,20)
out <- poolVar(var=s2,n=n)
tstat <- (mean(x)-mean(y)) / sqrt(out$var*out$multiplier)
pvalue <- 2*pt(-abs(tstat),df=out$df)
#  Equivalent to t.test(x,y)
```

---

predFCm *Predictive log fold change for microarrays*

Description

Calculate the predictive log fold change for a particular coefficient from a fit object.

Usage

```
predFCm(fit, coef=2, var.indep.of.fc=TRUE, all.de=TRUE, prop.true.null.method="lfdr")
```

Arguments

fit	an MArrayLM fitted model object produced by lmFit and eBayes
coef	integer vector indicating which columns in the fit object are to be shrunk
var.indep.of.fc	assume the genewise variances are independent of genewise fold changes?
all.de	assume all genes have a non-zero true fold change (TRUE)? If FALSE, then the proportion of truly non-differentially expressed (non-DE) genes will be estimated.
prop.true.null.method	method used to estimate proportion of truly non-DE genes. See propTrueNull for possible values.

Details

The predictive log fold changes are calculated as the posterior mean log fold changes in the empirical Bayes hierarchical model. We call them predictive log fold changes because they are the best prediction of what the log fold change will be for each gene in a comparable future experiment.

The log fold changes are shrunk towards zero depending on how variable they are. The var.indep.of.fc argument specifies whether the prior belief is that the log fold changes are independent of the variability of the genes or whether the log fold changes increase with increasing variability of the genes.

If all.de=TRUE, then all genes are assumed to have a non-zero log fold change, even if quite small. If all.de=FALSE, then some genes are assumed to have log fold changes exactly zero. The proportion of non-DE genes is estimated and taken into account in the calculation.

Value

numeric vector of predictive (shrunk) log fold changes

Author(s)

Belinda Phipson and Gordon Smyth

References

Phipson B (2013). *Empirical Bayes modelling of expression profiles and their associations*. PhD Thesis. University of Melbourne, Australia. http://hdl.handle.net/11343/38162

See Also

lmFit, eBayes, contrasts.fit

Examples

```
#  Simulate gene expression data,
#  6 microarrays with 1000 genes on each array
set.seed(2004)
y <- matrix(rnorm(6000),ncol=4)
```

```
# two experimental groups and one control group with two replicates each
group <- factor(c("A","A","B","B"))
design <- model.matrix(~group)

# fit a linear model
fit <- lmFit(y,design)
fit <- eBayes(fit)

# output predictive log fold changes for first 5 genes
pfc <- predFCm(fit,coef=2)
```

---

printHead *Print Leading Rows of Large Objects*

Description

Print the leading rows of a large vector, matrix or data.frame. This function is used by show methods for data classes defined in LIMMA.

Usage

```
printHead(x)
```

Arguments

x	any object

Details

If x is a vector with more than 20 elements, then printHead(x) prints only the first 5 elements. If x is a matrix or data.frame with more than 10 rows, then printHead(x) prints only the first 5 rows. Any other type of object is printed normally.

Author(s)

Gordon Smyth

See Also

An overview of classes defined in LIMMA is given in 02.Classes

PrintLayout *Print Layout - class*

Description

A list-based class for storing information about the process used to print spots on a microarray.

PrintLayout objects can be created using getLayout. The printer component of an RGList or MAList object is of this class.

Slots/List Components

Objects of this class contain no slots but should contain the following list components:

ngrid.r:	number of grid rows on the arrays
ngrid.c:	number of grid columns on the arrays
nspot.r:	number of rows of spots in each grid
nspot.c:	number of columns of spots in each grid
ndups:	number of duplicates of each DNA clone, i.e., number of times print-head dips into each well of DNA
spacing:	number of spots between duplicate spots. Only applicable if ndups>1. spacing=1 for side-by-side spots by rows,
npins:	actual number of pins or tips on the print-head
start:	character string giving position of the spot printed first in each grid. Choices are "topleft" or "topright" and p

Author(s)

Gordon Smyth

See Also

02.Classes gives an overview of all the classes defined by this package.

Examples

```
#  Settings for Swirl and ApoAI example data sets in User's Guide

printer <- list(ngrid.r=4, ngrid.c=4, nspot.r=22, nspot.c=24,
                ndups=1, spacing=1, npins=16, start="topleft")

#  Typical settings at the Australian Genome Research Facility

#  Full pin set, duplicates side-by-side on same row
printer <- list(ngrid.r=12, ngrid.c=4, nspot.r=20, nspot.c=20,
                ndups=2, spacing=1, npins=48, start="topright")

#  Half pin set, duplicates in top and lower half of slide
printer <- list(ngrid.r=12, ngrid.c=4, nspot.r=20, nspot.c=20,
                ndups=2, spacing=9600, npins=24, start="topright")
```

printorder *Identify Order in which Spots were Printed*

Description

Identify order in which spots were printed and the 384-well plate from which they were printed.

Usage

```
printorder(layout, ndups=1, spacing="columns", npins, start="topleft")
```

Arguments

layout	list with the components ngrid.r, ngrid.c, nspot.r and nspot.c, or an RGList or MAList object from which the printer layout may be extracted.
ndups	number of duplicate spots, i.e., number of times print-head dips into each well
spacing	character string indicating layout of duplicate spots. Choices are "columns", "rows" or "topbottom".
npins	actual number of pins or tips on the print-head
start	character string giving position of the spot printed first in each grid. Choices are "topleft" or "topright" and partial matches are accepted.

Details

In most cases the printer-head contains the layout$ngrid.r times layout$ngrid.c pins or tips and the array is printed using layout$nspot.r times layout$nspot.c dips of the head. The plate holding the DNA to be printed is assumed to have 384 wells in 16 rows and 24 columns.

ndups indicates the number of spots printed from each well. The replicate spots from multiple dips into the same wells are assumed to be side-by-side by columns (spacing="columns"), by rows (spacing="rows") or in the top and bottom halves of the array (spacing="topbottom").

In some cases a smaller number of physical pins is used and the total number of grids is built up by effectively printing two or more sub-arrays on the same slide. In this case the number of grids should be a multiple of the number of pins.

Printing is assumed to proceed by rows within each grid starting either from the top-left or the top-right.

Value

List with components

printorder	numeric vector giving printorder of each spot, i.e., which dip of the print-head was used to print it
plate	numeric vector giving plate number from which each spot was printed
plate.r	numeric vector giving plate-row number of the well from which each spot was printed

plate.c	numeric vector giving plate-column number of the well from which each spot was printed
plateposition	character vector summarizing plate number and plate position of the well from which each spot was printed with letters for plate rows and number for columns. For example 02B13 is second row, 13th column, of the second plate.

Author(s)

Gordon Smyth

See Also

normalizeForPrintorder.

An overview of LIMMA functions for reading data is given in 03.ReadingData.

Examples

```
printorder(list(ngrid.r=2,ngrid.c=2,nspot.r=12,nspot.c=8))
```

---

printtipWeights *Sub-array Quality Weights*

Description

Estimates relative quality weights for each sub-array in a multi-array experiment.

Usage

```
printtipWeights(object, design = NULL, weights = NULL, method = "genebygene", layout,
                maxiter = 50, tol = 1e-10, trace=FALSE)
```

Arguments

object	object of class numeric, matrix, MAList, marrayNorm, or ExpressionSet containing log-ratios or log-values of expression for a series of spotted microarrays.
design	the design matrix of the microarray experiment, with rows corresponding to arrays and columns to coefficients to be estimated. Defaults to the unit vector meaning that the arrays are treated as replicates.
weights	optional numeric matrix containing prior weights for each spot.
method	character string specifying the estimating algorithm to be used. Choices are "genebygene" and "reml".
layout	list specifying the dimensions of the spot matrix and the grid matrix. For details see PrintLayout-class.
maxiter	maximum number of iterations allowed.
tol	convergence tolerance.
trace	logical variable. If true then output diagnostic information at each iteration of the "reml" algorithm.

Details

The relative reliability of each sub-array (print-tip group) is estimated by measuring how well the expression values for that sub-array follow the linear model.

The method described in Ritchie et al (2006) and implemented in the `arrayWeights` function is adapted for this purpose. A heteroscedastic model is fitted to the expression values for each gene by calling the function `lm.wfit`. The dispersion model is fitted to the squared residuals from the mean fit, and is set up to have sub-array specific coefficients, which are updated in either full REML scoring iterations, or using an efficient gene-by-gene update algorithm. The final estimates of the sub-array variances are converted to weights.

The data object `object` is interpreted as for `lmFit`. In particular, the arguments `design`, `weights` and `layout` will be extracted from the data `object` if available and do not normally need to be set explicitly in the call; if any of these are set in the call then they will override the slots or components in the data `object`.

Value

A matrix of sub-array weights.

Author(s)

Matthew Ritchie and Gordon Smyth

References

Ritchie ME, Diyagama D, Neilson J, van Laar R, Dobrovic A, Holloway A, Smyth GK (2006). Empirical array quality weights in the analysis of microarray data. *BMC Bioinformatics* 7, 261. doi:10.1186/147121057261.

See Also

An overview of linear model functions in limma is given by 06.LinearModels.

Examples

```
## Not run:
# This example is designed for work on a subset of the data
# from ApoAI case study in Limma User's Guide

RG <- backgroundCorrect(RG, method="normexp")
MA <- normalizeWithinArrays(RG)
targets <- data.frame(Cy3=I(rep("Pool",6)),Cy5=I(c("WT","WT","WT","KO","KO","KO")))
design <- modelMatrix(targets, ref="Pool")
subarrayw <- printtipWeights(MA, design, layout=mouse.setup)
fit <- lmFit(MA, design, weights=subarrayw)
fit2 <- contrasts.fit(fit, contrasts=c(-1,1))
fit2 <- eBayes(fit2)
# Use of sub-array weights increases the significance of the top genes
topTable(fit2)
# Create an image plot of sub-array weights from each array
zlim <- c(min(subarrayw), max(subarrayw))
```

```
par(mfrow=c(3,2), mai=c(0.1,0.1,0.3,0.1))
for(i in 1:6)
 imageplot(subarrayw[,i], layout=mouse.setup, zlim=zlim, main=paste("Array", i))

## End(Not run)
```

---

propexpr *Estimate Proportion of Expressed Probes*

Description

Estimate the proportion of microarray probes which are expressed in each array.

Usage

```
propexpr(x, neg.x=NULL, status=x$genes$Status, labels=c("negative","regular"))
```

Arguments

x	matrix or similar object containing raw intensities for a set of arrays.
neg.x	matrix or similar object containing raw intensities for negative control probes for the same arrays. If NULL, then negative controls must be provided in x.
status	character vector specifying control type of each probe. Only used if neg.x is NULL.
labels	character vector giving the status values for negative control probes and regular (non-control) probes respectively. If of length 1, then all probes other than the negative controls are assumed to be regular. Only used if neg.x is NULL.

Details

This function estimates the overall proportion of probes on each microarray that correspond to expressed genes using the method of Shi et al (2010). The function is especially useful for Illumina BeadChips arrays, although it can in principle be applied to any platform with good quality negative controls.

The negative controls can be supplied either as rows of x or as a separate matrix. If supplied as rows of x, then the negative controls are identified by the status vector. x might also include other types of control probes, but these will be ignored in the calculation.

Illumina BeadChip arrays contain 750-1600 negative control probes. If read.idat is used to read Illumina expression IDAT files, then the control probes will be populated as rows of the output EListRaw object, and the vector x$genes$Status will be set to identify control probes.

Alternatively, expression values can be exported from Illumina's GenomeStudio software as tab-delimited text files. In this case, the control probes are usually written to a separate file from the regular probes.

Value

Numeric vector giving the proportions of expressed probes in each array.

Author(s)

Wei Shi and Gordon Smyth

References

Shi W, de Graaf C, Kinkel S, Achtman A, Baldwin T, Schofield L, Scott H, Hilton D, Smyth GK (2010). Estimating the proportion of microarray probes expressed in an RNA sample. *Nucleic Acids Research* 38(7), 2168-2176. doi:10.1093/nar/gkp1204.

See Also

Description of the control probes in Illumina BeadChips can be found in `read.ilmn`.

Examples

```
## Not run:
# Read Illumina binary IDAT files
x <- read.idat(idat, bgx)
propexpr(x)

# Read text files exported from GenomeStudio
x <- read.ilmn(files = "sample probe profile.txt",
               ctrlfiles = "control probe profile.txt")
propexpr(x)

## End(Not run)
```

---

propTrueNull *Estimate Proportion of True Null Hypotheses*

Description

Estimate the proportion of true null hypotheses from a vector of p-values.

Usage

```
propTrueNull(p, method="lfdr", nbins=20, ...)
convest(p, niter=100, plot=FALSE, report=FALSE, file="", tol=1e-6)
```

Arguments

`p`	numeric vector of p-values.
`method`	estimation method. Choices are `"lfdr"`, `"mean"`, `"hist"` or `"convest"`.
`nbins`	number of histogram bins (if `method="hist"`).
`niter`	number of iterations to be used in fitting the convex, decreasing density for the p-values.

plot	logical, should updated plots of fitted convex decreasing p-value density be produced at each iteration?
report	logical, should the estimated proportion be printed at each iteration?
file	name of file to which to write the report. Defaults to standard output.
tol	accuracy of the bisectional search for finding a new convex combination of the current iterate and the mixing density
...	other arguments are passed to convest if method="convest".

Details

The proportion of true null hypotheses in a collection of hypothesis tests is often denoted pi0. This function estimates pi0 from a vector of p-values.

method="lfdr" implements the method of Phipson (2013) based on averaging local false discovery rates across the p-values.

method="mean" is a very simple method based on averaging the p-values. It gives a slightly smaller estimate than 2*mean(p).

method="hist" implements the histogram method of Mosig et al (2001) and Nettleton et al (2006).

method="convest" calls convest, which implements the method of Langaas et al (2005) based on a convex decreasing density estimate.

Value

Numeric value in the interval [0,1] representing the estimated proportion of true null hypotheses.

Author(s)

Belinda Phipson and Gordon Smyth for propTrueNull. Egil Ferkingstad, Mette Langaas and Marcus Davy for convest.

References

Langaas M, Ferkingstad E, Lindqvist B (2005). Estimating the proportion of true null hypotheses, with application to DNA microarray data. *Journal of the Royal Statistical Society Series* B 67, 555-572.

Mosig MO, Lipkin E, Khutoreskaya G, Tchourzyna E, Soller M, Friedmann A (2001). A whole genome scan for quantitative trait loci affecting milk protein percentage in Israeli-Holstein cattle, by means of selective milk DNA pooling in a daughter design, using an adjusted false discovery rate criterion. *Genetics* 157, 1683-1698.

Nettleton D, Hwang JTG, Caldo RA, Wise RP (2006). Estimating the number of true null hypotheses from a histogram of p values. *Journal of Agricultural, Biological, and Environmental Statistics* 11, 337-356.

Phipson B (2013). Empirical Bayes Modelling of Expression Profiles and Their Associations. PhD Thesis, University of Melbourne, Australia. http://hdl.handle.net/11343/38162

Ritchie ME, Phipson B, Wu D, Hu Y, Law CW, Shi W, Smyth GK (2015). limma powers differential expression analyses for RNA-sequencing and microarray studies. *Nucleic Acids Research* 43, e47. doi:10.1093/nar/gkv007.

See Also

See 08.Tests for other functions for producing or interpreting p-values.

Examples

```
# Test statistics
z <- rnorm(200)

# First 40 have non-zero means
z[1:40] <- z[1:40]+2

# True pi0
160/200

# Two-sided p-values
p <- 2*pnorm(-abs(z))

# Estimate pi0
propTrueNull(p, method="lfdr")
propTrueNull(p, method="hist")
```

---

protectMetachar *Protect Metacharacters*

Description

Add backslashes before any metacharacters found in a string.

Usage

```
protectMetachar(x)
```

Arguments

x	character vector

Details

This function is used to protect strings containing metacharacters so that the metacharacters can be treated as ordinary characters in string matching operations.

Value

A character vector of the same length as x in which two backslashes have been inserted before any metacharacter.

Author(s)

Gordon Smyth

See Also

An overview of LIMMA functions for reading data is given in 03.ReadingData.

Examples

```
# without protectMetachar, this would be no match
grep(protectMetachar("Ch1 (mean)"),"Ch1 (mean)")
```

---

qqt *Student's t or Fisher's F Quantile-Quantile Plot*

Description

Plots the quantiles of a data sample against the theoretical quantiles of a Student's t distribution.

Usage

```
qqt(y, df = Inf, ylim = range(y), main = "Student's t Q-Q Plot",
   xlab = "Theoretical Quantiles", ylab = "Sample Quantiles", plot.it = TRUE, ...)
qqf(y, df1, df2, ylim=range(y), main= "F Distribution Q-Q Plot",
   xlab = "Theoretical Quantiles", ylab = "Sample Quantiles", plot.it = TRUE, ...)
```

Arguments

y	a numeric vector or array containing the data sample
df	degrees of freedom for the t-distribution. The default df=Inf represents the normal distribution.
df1	numerator degrees of freedom for the F-distribution.
df2	denominator degrees of freedom for the F-distribution.
ylim	plotting range for y
main	main title for the plot
xlab	x-axis title for the plot
ylab	y-axis title for the plot
plot.it	whether or not to produce a plot
...	other arguments to be passed to plot

Details

This function is analogous to qqnorm for normal probability plots. In fact qqt(y,df=Inf) is identical to qqnorm(y) in all respects except the default title on the plot.

Value

A list is invisibly returned containing the values plotted in the QQ-plot:

x	theoretical quantiles of the t-distribution or F-distribution
y	the data sample, same as input y

Author(s)

Gordon Smyth

See Also

qqnorm

Examples

```
#  See also the lmFit examples

y <- rt(50,df=4)
qqt(y,df=4)
abline(0,1)
```

QualityWeights *Spot Quality Weights for Spotted Microarrays*

Description

Functions to calculate quality weights for individual spots based on the image analysis output file for a spotted microarray.

Usage

```
wtarea(ideal = c(160,170))
wtflags(weight = 0, cutoff = 0)
wtIgnore.Filter
```

Arguments

ideal	numeric vector giving the ideal range of areas for good quality spots (in pixels). The minimum and maximum values are used to specify the range of ideal values. All values should be positive.
weight	non-negative weight to be given to flagged spots.
cutoff	cutoff value for Flags below which spots will be downweighted.

Details

These functions can be passed as an argument to read.maimages to construct quality weights as the microarray data is read in.

wtarea downweights unusually small or large spots and is designed for SPOT output. It gives weight 1 to spots that have areas in the ideal range, given in pixels, and linearly downweights spots that are smaller or larger than this range.

wtflags is designed for GenePix output and gives the specified weight to spots with Flags value less than the cutoff value. Choose cutoff=0 to downweight all flagged spots. Choose cutoff=-50

to downweight bad or absent spots or `cutoff=-75` to downweight only spots which have been manually flagged as bad.

`wtIgnore.Filter` is designed for QuantArray output and sets the weights equal to the column `Ignore Filter` produced by QuantArray. These weights are 0 for spots to be ignored and 1 otherwise.

Value

A function that takes a dataframe or matrix as argument and produces a numeric vector of weights between 0 and 1.

Author(s)

Gordon Smyth

See Also

An overview of LIMMA functions for reading data is given in 03.ReadingData.

Examples

```
#  Read in spot output files from current directory and give full weight to 165
#  pixel spots.  Note: for this example to run you must set fnames to the names
#  of actual spot output files (data not provided).
## Not run:
RG <- read.maimages(fnames,source="spot",wt.fun=wtarea(165))
#  Spot will be downweighted according to weights found in RG
MA <- normalizeWithinArrays(RG,layout)

## End(Not run)
```

`rankSumTestWithCorrelation` *Two Sample Wilcoxon-Mann-Whitney Rank Sum Test Allowing For Correlation*

Description

An extension of the well-known rank-based test, but allowing for correlations between cases.

Usage

```
rankSumTestWithCorrelation(index, statistics, correlation=0, df=Inf)
```

Arguments

index	any index vector such that statistics[index] contains the values of the statistic for the test group.
statistics	numeric vector giving values of the test statistic.
correlation	numeric scalar, average correlation between cases in the test group. Cases in the second group are assumed independent of each other and of the first group.
df	degrees of freedom on which the correlation has been estimated.

Details

This function implements a correlation-adjusted version of the Wilcoxon-Mann-Whitney test proposed by Wu and Smyth (2012). It tests whether the mean rank of statistics in the test group is greater or less than the mean rank of the remaining statistic values.

When the correlation (or variance inflation factor) is zero, the function performs the usual two-sample Wilcoxon-Mann-Whitney rank sum test. The Wilcoxon-Mann-Whitney test is implemented following the formulas given in Zar (1999) Section 8.10, including corrections for ties and for continuity.

The test allows for the possibility that cases in the test group may be more highly correlated on average than cases not in the group. When the correlation is non-zero, the variance of the rank-sum statistic is computed using a formula derived from equation (4.5) of Barry et al (2008). When the correlation is positive, the variance is increased and the test will become more conservative.

Value

Numeric vector of length 2 containing the left.tail and right.tail p-values.

Author(s)

Gordon Smyth and Di Wu

References

Barry WT, Nobel AB, Wright FA (2008). A statistical framework for testing functional categories in microarray data. *Annals of Applied Statistics* 2, 286-315.

Wu D, Smyth GK (2012). Camera: a competitive gene set test accounting for inter-gene correlation. *Nucleic Acids Research* 40, e133. doi:10.1093/nar/gks461.

Zar JH (1999). *Biostatistical Analysis 4th Edition*. Prentice-Hall International, Upper Saddle River, New Jersey.

See Also

wilcox.test performs the usual Wilcoxon-Mann-Whitney test assuming independence.

An overview of tests in limma is given in 08.Tests.

Examples

```
stat <- rnorm(100)
index <- 1:10
stat[index] <- stat[1:10]+1

rankSumTestWithCorrelation(index, stat)
rankSumTestWithCorrelation(index, stat, correlation=0.1)

group <- rep(1,100)
group[index] <- 2
group <- factor(group)
wilcox.test(stat ~ group)
```

---

read.columns *Read specified columns from a file*

Description

Reads specified columns from a file in table format and creates a data frame from it, with cases corresponding to lines and variables to fields in the file.

Usage

```
read.columns(file, required.col=NULL, text.to.search="", sep="\t", quote="\"", skip=0,
        fill=TRUE, blank.lines.skip=TRUE, comment.char="", allowEscapes=FALSE, ...)
```

Arguments

file	the name of the file which the data are to be read from.
required.col	character vector of names of the required columns
text.to.search	character string. If any column names can be found in this string, those columns will also be read.
sep	the field separator character
quote	character string of characters to be treated as quote marks
skip	the number of lines of the data file to skip before beginning to read data.
fill	logical. If TRUE then in case the rows have unequal length, blank fields are implicitly added.
blank.lines.skip	logical: if TRUE blank lines in the input are ignored.
comment.char	character: a character vector of length one containing a single character or an empty string.
allowEscapes	logical. Should C-style escapes such as '\n' be processed or read verbatim (the default)?
...	other arguments are passed to read.table, excluding the following which are reserved and cannot be set by the user: header, col.names, check.names and colClasses.

Details

This function is an interface to `read.table` in the base package. It uses `required.col` and `text.to.search` to set up the `colClasses` argument of `read.table`.

Note the following arguments of `read.table` are used by `read.columns` and therefore cannot be set by the user: `header`, `col.names`, `check.names` and `colClasses`.

This function is used by `read.maimages`.

Value

A data frame (data.frame) containing a representation of the data in the file.

Author(s)

Gordon Smyth

See Also

`read.maimages`, `read.table`.

An overview of LIMMA functions for reading data is given in 03.ReadingData.

`read.idat` *Read Illumina expression data from IDAT files*

Description

Read Illumina BeadArray data from IDAT and manifest (.bgx) files for gene expression platforms.

Usage

```
read.idat(idatfiles, bgxfile, path = NULL, bgxpath = path,
          dateinfo = FALSE, annotation = "Symbol", tolerance = 0, verbose = TRUE)
```

Arguments

`idatfiles`	character vector specifying the IDAT files to be read in. Gzipped files are not accepted.
`bgxfile`	character string specifying bead manifest file (.bgx) to be read in. May be gzipped.
`path`	character string giving the directory containing the IDAT files. The default is the current working directory.
`bgxpath`	character string giving the directory containing the bgx manifest file. Defaults to the same directory as for IDAT files.
`dateinfo`	logical. Should date and software version information be read in?
`annotation`	character vector of annotation columns to be read from the manifest file.
`tolerance`	integer. The number of probe ID discrepancies allowed between the manifest and any of the IDAT files.
`verbose`	logical. Should progress messages be sent to standard output?

Details

Illumina's BeadScan/iScan software outputs probe intensities in IDAT format (encrypted XML files) and uses probe information stored in a platform specific manifest file (.bgx). These files can be processed using the low-level functions `readIDAT` and `readBGX` from the `illuminaio` package (Smith et al, 2013).

The `read.idat` function provides a convenient way to read these files into R and to store them in an `EListRaw-class` object. The function serves a similar purpose to `read.ilmn`, which reads text files exported by Illumina's GenomeStudio software, but it reads the IDAT files directly without any need to convert them first to text.

The function reads information on control probes as well as for regular probes. Probe types are indicated in the `Status` column of the `genes` component of the `EListRaw` object.

The `annotation` argument specifies probe annotation columns to be extracted from the manifest file. The manifest typically contains the following columns: `Species`, `Source`, `Search_Key`, `Transcript`, `ILMN_Gene`, `Source_Reference_ID`, `RefSeq_ID`, `Unigene_ID`, `Entrez_Gene_ID`, `GI`, `Accession`, `Symbol`, `Protein_Product`, `Probe_Id`, `Array_Address_Id`, `Probe_Type`, `Probe_Start`, `Probe_Sequence`, `Chromosome`, `Probe_Chr_Orientation`, `Probe_Coordinates`, `Cytoband`, `Definition`, `Ontology_Component`, `Ontology_Process`, `Ontology_Function`, `Synonyms`, `Obsolete_Probe_Id`. Note that the `Probe_Id` and `Array_Address_Id` columns are always read and do not need to be included in the `annotation` argument.

If more than `tolerance` probes in the manifest cannot be found in an IDAT file then the function will return an error.

Value

An `EListRaw` object with the following components:

`E`	numeric matrix of raw intensities.
`other$NumBeads`	numeric matrix of same dimensions as `E` giving number of beads used for each intensity value.
`other$STDEV`	numeric matrix of same dimensions as `E` giving bead-level standard deviation or standard error for each intensity value.
`genes`	data.frame of probe annotation. This includes the `Probe_Id` and `Array_Address_Id` columns extracted from the manifest file, plus a `Status` column identifying control probes, plus any other columns specified by `annotation`.
`targets`	data.frame of sample information. This includes the IDAT file names plus other columns if `dateinfo=TRUE`.

Author(s)

Matt Ritchie and Gordon Smyth

References

Smith ML, Baggerly KA, Bengtsson H, Ritchie ME, Hansen KD (2013). illuminaio: An open source IDAT parsing tool. *F1000 Research* 2, 264. doi:10.12688/f1000research.2264.v1.

See Also

read.ilmn imports gene expression data as a text file exported from GenomeStudio.

neqc performs normexp by control background correction, log transformation and quantile between-array normalization for Illumina expression data.

propexpr estimates the proportion of expressed probes in a microarray.

detectionPValues computes detection p-values from the negative controls.

Examples

```
## Not run:
idatfiles <- dir(pattern="idat")
bgxfile <- dir(pattern="bgx")
x <- read.idat(idatfiles, bgxfile)
x$other$Detection <- detectionPValues(x)
propexpr(x)
y <- neqc(x)

## End(Not run)
```

---

read.ilmn *Read Illumina Expression Data*

Description

Read Illumina summary probe profile files and summary control probe profile files

Usage

```
read.ilmn(files=NULL, ctrlfiles=NULL, path=NULL, ctrlpath=NULL, probeid="Probe",
          annotation=c("TargetID", "SYMBOL"), expr="AVG_Signal",
          other.columns="Detection", sep="\t", quote="\"", verbose=TRUE, ...)
```

Arguments

files	character vector giving the names of the summary probe profile files.
ctrlfiles	character vector giving the names of the summary control probe profile files.
path	character string giving the directory containing the summary probe profile files. Default is the current working directory.
ctrlpath	character string giving the directory containing the summary control probe profile files. Default is the same directory as for the probe profile files.
probeid	character string giving the name of the probe identifier column.
annotation	character vector giving possible column names for probe annotation.
expr	character string giving a keyword identifying the expression intensity columns. Any input column with column name containing this key will be read as containing intensity values.

other.columns	character vector giving keywords sufficient to identify any extra data columns that should be read in, such as "Detection", "Avg_NBEADS", "BEAD_STDEV" etc. The default of Detection is usually sufficient to identify the columns containing detection p-values.
sep	the field separator character.
quote	character string of characters to be treated as quote marks.
verbose	logical, TRUE to report names of profile files being read.
...	any other parameters are passed on to read.columns.

Details

Illumina BeadStudio outputs probe intensities (regular probe intensities) and control probe intensities to summary probe profile files (containing regular probes) and summary control probe profile files, respectively. If both files and ctrlfiles are not NULL, this function will combine the data read from the two file types and save them to an EListRaw-class object. If one of them is NULL, then only the required data are read in.

Probe types are indicated in the Status column of genes, a component of the returned EListRaw-class object. There are seven types of control probes in total, including negative, biotin, labeling, cy3_hyb, housekeeping, high_stringency_hyb and low_stringency_hyb. Regular probes have the probe type regular. The Status column will not be created if ctrlfiles is NULL.

To read in columns other than probeid, annotation and expr, users need to specify keywords in other.columns. One keyword corresponds to one type of columns. Examples of keywords are "Detection", "Avg_NBEADS", "BEAD_STDEV" etc.

Value

An EListRaw-class object with the following components:

E	numeric matrix of intensities.
genes	data.frame of probe annotation. Contains any columns specified by annotation that are found in the input files.
other	a list of matrices corresponding to any other.columns found in the input files.

Author(s)

Wei Shi and Gordon K Smyth

See Also

read.ilmn.targets reads in Illumina expression data using the file information extracted from a target data frame which is often created by the readTargets function.

neqc performs normexp by control background correction, log transformation and quantile between-array normalization for Illumina expression data.

normexp.fit.control estimates the parameters of the normal+exponential convolution model with the help of negative control probes.

propexpr estimates the proportion of expressed probes in a microarray.

Examples

```
## Not run:
x <- read.ilmn(files="sample probe profile.txt",
               ctrlfiles="control probe profile.txt")

## End(Not run)
# See neqc and beadCountWeights for other examples using read.ilmn
```

`read.ilmn.targets` *Read Illumina Data from a Target Dataframe*

Description

Read Illumina data from a target dataframe

Usage

```
read.ilmn.targets(targets, ...)
```

Arguments

`targets`	data frame including names of profile files.
`...`	any other parameters are passed on to `read.ilmn`.

Details

`targets` is often created by calling the function `readTargets`. Rows in `targets` are arrays and columns contain related array or RNA sample information.

At least one of the two columns called `files` and/or `ctrlfiles` should be present in `targets`, which includes names of summary probe profile files and names of summary control probe profile files respectively. This function calls `read.ilmn` to read in the data.

Value

An `EListRaw-class` object. See return value of the function `read.ilmn` for details.

Author(s)

Wei Shi

See Also

`read.ilmn`

read.maimages *Read RGList or EListRaw from Image Analysis Output Files*

Description

Reads an RGList from a set of two-color microarray image analysis output files, or an EListRaw from a set of one-color files.

Usage

```
read.maimages(files=NULL, source="generic", path=NULL, ext=NULL, names=NULL,
            columns=NULL, other.columns=NULL, annotation=NULL, green.only=FALSE,
              wt.fun=NULL, verbose=TRUE, sep="\t", quote=NULL, ...)
read.imagene(files, path=NULL, ext=NULL, names=NULL, columns=NULL, other.columns=NULL,
             wt.fun=NULL, verbose=TRUE, sep="\t", quote="\"", ...)
```

Arguments

files	character vector giving the names of the files containing image analysis output or, for Imagene data, a character matrix of names of files. Alternatively, it can be a data.frame containing a column called FileName. If omitted, then all files with extension ext in the specified directory will be read in alphabetical order.
source	character string specifying the image analysis program which produced the output files. Choices are "generic", "agilent", "agilent.median", "agilent.mean", "arrayvision", "arrayvision.ARM", "arrayvision.MTM", "bluefuse", "genepix", "genepix.custom", "genepix.median", "imagene", "imagene9", "quantarray", "scanarrayexpress", "smd.old", "smd", "spot" or "spot.close.open".
path	character string giving the directory containing the files. The default is the current working directory.
ext	character string giving optional extension to be added to each file name
names	character vector of unique names to be associated with each array as column name. Can be supplied as files$Label if files is a data.frame. Defaults to removeExt(files).
columns	list, or named character vector. For two color data, this should have fields R, G, Rb and Gb giving the column names to be used for red and green foreground and background or, in the case of Imagene data, a list with fields f and b. For single channel data, the fields are usually E and Eb. This argument is optional if source is specified, otherwise it is required.
other.columns	character vector of names of other columns to be read containing spot-specific information
annotation	character vector of names of columns containing annotation information about the probes
green.only	logical, for use with source, should the green (Cy3) channel only be read, or are both red and green required?

wt.fun	function to calculate spot quality weights
verbose	logical, TRUE to report each time a file is read
sep	the field separator character
quote	character string of characters to be treated as quote marks
...	any other arguments are passed to read.table

Details

These are the main data input functions for the LIMMA package. read.maimages reads either single channel or two-color microarray intensity data from text files. read.imagene is specifically for two-color ImaGene intensity data created by ImaGene versions 1 through 8, and is called by read.maimages to read such data.

read.maimages is designed to read data from any microarray platform except for Illumina BeadChips, which are read by read.ilmn, and Affymetrix GeneChip data, which is best read and preprocessed by specialist packages designed for that platform.

read.maimages extracts the foreground and background intensities from a series of files, produced by an image analysis program, and assembles them into the components of one list. The image analysis programs Agilent Feature Extraction, ArrayVision, BlueFuse, GenePix, ImaGene, QuantArray (Version 3 or later), Stanford Microarray Database (SMD) and SPOT are supported explicitly. Almost all these programs write the intensity data for each microarray to one file. The exception is ImaGene, early versions of which wrote the red and green channels of each microarray to different files. Data from some other image analysis programs not mentioned above can be read if the appropriate column names containing the foreground and background intensities are specified using the columns argument. (Reading custom columns will work provided the column names are unique and there are no rows in the file after the last line of data. Header lines are ok.)

For Agilent files, two possible foreground estimators are supported: source="agilent.median" use median foreground while source="agilent.mean" uses mean foreground. Background estimates are always medians. The use of source="agilent" defaults to "agilent.median". Note that this behavior is new from 9 March 2012. Previously, in limma 3.11.16 or earlier, "agilent" had the same meaning as "agilent.mean".

For GenePix files, two possible foreground estimators are supported as well as custom background: source="genepix.median" uses the median foreground estimates while source="genepix.mean" uses mean foreground estimates. The use of source="genepix" defaults to "genepix.mean". Background estimates are always medians unless source="genepix.custom" is specified. GenePix 6.0 and later supply some custom background options, notably morphological background. If the GPR files have been written using a custom background, then source="genepix.custom" will cause it to be read and used.

For SPOT files, two possible background estimators are supported: source="spot" uses background intensities estimated from the morphological opening algorithm. If source="spot.close.open" then background intensities are estimated from morphological closing followed by opening.

ArrayVision reports spot intensities in a number of different ways. read.maimages caters for ArrayVision's Artifact-removed (ARM) density values using source="arrayvision.ARM" or for Median-based Trimmed Mean (MTM) density values with "arrayvision.MTM". ArrayVision users may find it useful to read the top two lines of their data file to check which version of density values they have.

SMD data should consist of raw data files from the database, in tab-delimited text form. There are two possible sets of column names depending on whether the data was entered into the database before or after September 2003. `source="smd.old"` indicates that column headings in use prior to September 2003 should be used.

Intensity data from ImaGene versions 1 to 8 (`source="imagene"`) is different from other image analysis programs in that the red and green channels were written to separate files. `read.maimages` handles the special behavior of the early ImaGene versions by requiring that the argument `files` should be a matrix with two columns instead of a vector. The first column should contain the names of the files containing green channel (cy3) data and the second column should contain names of files containing red channel (cy5) data. Alternately, `files` can be entered as a vector of even length instead of a matrix. In that case, each consecutive pair of file names is assumed to contain the green (cy3) and red (cy5) intensities respectively from the same array. The function `read.imagene` is called by `read.maimages` when `source="imagene"`, so `read.imagene` does not need to be called directly by users.

ImaGene version~9 (`source="imagene9"`) reverts to the same behavior as the other image analysis programs. For ImaGene~9, `files` is a vector of length equal to the number of microarrays, same as for other image analysis programs.

Spot quality weights may be extracted from the image analysis files using a weight function wt.fun. `wt.fun` may be any user-supplied function which accepts a data.frame argument and returns a vector of non-negative weights. The columns of the data.frame are as in the image analysis output files. There is one restriction, which is that the column names should be referred to in full form in the weight function, i.e., do not rely on name expansion for partial matches when referring to the names of the columns. See `QualityWeights` for suggested weight functions.

The argument `other.columns` allows arbitrary columns of the image analysis output files to be preserved in the data object. These become matrices in the component `other` component. For ImaGene data, the other column headings should be prefixed with `"R "` or `"G "` as appropriate.

Value

For one-color data, an `EListRaw` object. For two-color data, an `RGList` object containing the components

`R`	matrix containing the red channel foreground intensities for each spot for each array.
`Rb`	matrix containing the red channel background intensities for each spot for each array.
`G`	matrix containing the green channel foreground intensities for each spot for each array.
`Gb`	matrix containing the green channel background intensities for each spot for each array.
`weights`	spot quality weights, if `wt.fun` is given
`other`	list containing matrices corresponding to `other.columns` if given
`genes`	data frame containing annotation information about the probes, for example gene names and IDs and spatial positions on the array, currently set only if `source` is `"agilent"`, `"genepix"` or `source="imagene"` or if the `annotation` argument is set

targets	data frame with column FileName giving the names of the files read. If files was a data.frame on input, then the whole data.frame is stored here on output.
source	character string giving the image analysis program name
printer	list of class PrintLayout, currently set only if source="imagene"

Warnings

All image analysis files being read are assumed to contain data for the same genelist in the same order. No checking is done to confirm that this is true. Probe annotation information is read from the first file only.

Author(s)

Gordon Smyth, with speed improvements suggested by Marcus Davy

References

Ritchie ME, Phipson B, Wu D, Hu Y, Law CW, Shi W, Smyth GK (2015). limma powers differential expression analyses for RNA-sequencing and microarray studies. *Nucleic Acids Research* 43, e47. doi:10.1093/nar/gkv007.

See Also

read.maimages uses read.columns for efficient reading of text files. As far as possible, it has similar behavior to read.table in the base package.

read.ilmn reads probe or gene summary profile files from Illumina BeadChips.

An overview of LIMMA functions for reading data is given in 03.ReadingData.

Examples

```
#  Read all .gpr files from current working directory
#  and give weight 0.1 to spots with negative flags

## Not run: files <- dir(pattern="*\\.gpr$")
RG <- read.maimages(files,"genepix",wt.fun=wtflags(0.1))
## End(Not run)

#  Read all .spot files from current working director and down-weight
#  spots smaller or larger than 150 pixels

## Not run: files <- dir(pattern="*\\.spot$")
RG <- read.maimages(files,"spot",wt.fun=wtarea(150))
## End(Not run)
```

readGAL *Read a GAL file*

Description

Read a GenePix Array List (GAL) file into a dataframe.

Usage

```
readGAL(galfile=NULL,path=NULL,header=TRUE,sep="\t",quote="\"",skip=NULL,as.is=TRUE,...)
```

Arguments

galfile	character string giving the name of the GAL file. If NULL then a file with extension .gal is found in the directory specified by path.
path	character string giving the directory containing the files. If NULL then assumed to be the current working directory.
header	logical variable, if TRUE then the first line after skip is assumed to contain column headings. If FALSE then a value should specified for skip.
sep	the field separator character
quote	the set of quoting characters
skip	number of lines of the GAL file to skip before reading data. If NULL then this number is determined by searching the file for column headings.
as.is	logical variable, if TRUE then read in character columns as vectors rather than factors.
...	any other arguments are passed to read.table

Details

A GAL file is a list of genes IDs and associated information produced by an Axon microarray scanner. Apart from header information, the file must contain data columns labeled Block, Column, Row and ID. A Name column is usually included as well. Other columns are optional. See the Axon URL below for a detailed description of the GAL file format.

This function reads in the data columns with a minimum of user information. In most cases the function can be used without specifying any of the arguments.

Value

A data frame with columns

Block	numeric vector containing the print tip indices
Column	numeric vector containing the spot columns
Row	numeric vector containing the spot rows
ID	character vector, as factor if as.is=FALSE, containing gene library identifiers
Name	character vector, as factor if as.is=FALSE, containing gene names

The data frame will be sorted so that Column is the fastest moving index, then Row, then Block.

Author(s)

Gordon Smyth

References

https://www.cryer.co.uk/file-types/a/atf/genepix_file_formats.htm

See Also

`read.Galfile` in the marray package.

An overview of LIMMA functions for reading data is given in 03.ReadingData.

Examples

```
# readGAL()
# will read in the first GAL file (with suffix ".gal")
# found in the current working directory
```

`readHeader` *Read Header Information from Microarray Raw Data File*

Description

Read the header information from a microarray raw data file, as output from an image analysis software program such as GenePix. These functions are used internally by `read.maimages` and are not usually called directly by users.

Usage

```
readGenericHeader(file, columns, sep="\t")
readGPRHeader(file)
readSMDHeader(file)
```

Arguments

`file`	character string giving file name. If it does not contain an absolute path, the file name is relative to the current working directory.
`columns`	character vector specifying data column headings expected to be in file
`sep`	the character string separating column names

Details

Raw data files exported by image analysis programs include a number of header lines which contain information about the scanning process. This function extracts that information and locates the line where the intensity data begins. `readGPRHeader` is for GenePix output and `readSMDHeader` is for files from the Stanford Microarray Database (SMD). `readGenericHeader` finds the line in the file on which the data begins by searching for specified column headings.

Value

A list with components corresponding to lines of header information. A key component is `NHeaderRecords` which gives the number of lines in the file before the intensity data begins. All other components are character vectors.

Author(s)

Gordon Smyth

References

See https://www.cryer.co.uk/file-types/a/atf/genepix_file_formats.htm for GenePix formats.

The Stanford Microarray Database was at http://smd.princeton.edu but has now been retired.

See Also

`read.maimages`

An overview of LIMMA functions to read data is given in 03.ReadingData.

---

readImaGeneHeader *Read ImaGene Header Information*

Description

Read the header information from an ImaGene image analysis output file. This function is used internally by `read.maimages` and is not usually called directly by users.

Usage

```
readImaGeneHeader(file)
```

Arguments

`file`	character string giving file name or path

Details

The raw data files exported by the microarray image analysis software ImaGene include a number of header lines which contain information about the printing and scanning processes. This function extracts that information and locates the line where the intensity data begins.

Value

A list containing information read from the header of the ImaGene file. Each Begin-End environment found in the file header will become a recursive list in the output object, with components corresponding to fields in the file. See the ImaGene documentation for further information. The output object will also contain a component `NHeaderRecords` giving the number of lines in the file before the intensity data begins.

Author(s)

Gordon Smyth

See Also

read.imagene

An overview of LIMMA functions to read data is given in 03.ReadingData.

Examples

```
## Not run:
h <- readImaGeneHeader("myImaGenefile.txt")
names(h)
h$NHeaderRecords
h[["Field Dimensions"]]

## End(Not run)
```

---

readSpotTypes *Read Spot Types File*

Description

Read a table giving regular expressions to identify different types of spots in the gene-dataframe.

Usage

```
readSpotTypes(file="SpotTypes.txt",path=NULL,sep="\t",check.names=FALSE,...)
```

Arguments

file	character string giving the name of the file specifying the spot types.
path	character string giving the directory containing the file. Can be omitted if the file is in the current working directory.
sep	the field separator character
check.names	logical, if FALSE column names will not be converted to valid variable names, for example spaces in column names will not be left as is
...	any other arguments are passed to read.table

Details

The file is a text file with rows corresponding to types of spots and the following columns: SpotType gives the name for the spot type, ID is a regular expression matching the ID column, Name is a regular expression matching the Name column, and Color is the R name for the color to be associated with this type.

Value

A data frame with columns

SpotType	character vector giving names of the spot types
ID	character vector giving regular expressions
Name	character vector giving regular expressions
Color	character vector giving names of colors

Author(s)

Gordon Smyth following idea of James Wettenhall

See Also

An overview of LIMMA functions for reading data is given in 03.ReadingData.

---

readTargets *Read Targets File*

Description

Read targets file for a microarray experiment into a dataframe.

Usage

```
readTargets(file="Targets.txt", path=NULL, sep="\t", row.names=NULL, quote="\"",...)
```

Arguments

file	character string giving the name of the targets file.
path	character string giving the directory containing the file. Can be omitted if the file is in the current working directory.
sep	field separator character
row.names	character string giving the name of a column from which to obtain row names
quote	the set of quoting characters
...	other arguments are passed to read.table

Details

The targets file is a text file containing information about the RNA samples used as targets in the microarray experiment. Rows correspond to arrays and columns to covariates associated with the targets. For a two-color experiment, the targets file will normally include columns labeled `Cy3` and `Cy5` or similar specifying which RNA samples are hybridized to each channel of each array. Other columns may contain any other covariate information associated with the arrays or targets used in the experiment.

If `row.names` is non-null and there is a column by that name with unique values, then those values will be used as row names for the dataframe. If `row.names` is null, then the column `Label` will be used if such exists or, failing that, the column `FileName`.

See the Limma User's Guide for examples of this function.

Value

A dataframe. Character columns are not converted into factors.

Author(s)

Gordon Smyth

See Also

An overview of LIMMA functions for reading data is given in 03.ReadingData.

`removeBatchEffect` *Remove Batch Effect*

Description

Remove batch effects from expression data.

Usage

```
removeBatchEffect(x, batch = NULL, batch2 = NULL, covariates = NULL,
                  design = NULL, group = NULL, ...)
```

Arguments

`x`	numeric matrix, or any data object that can be processed by `getEAWP` containing log-expression values for a series of samples. Rows correspond to probes and columns to samples.
`batch`	factor or vector indicating batches.
`batch2`	factor or vector indicating a second series of batches.
`covariates`	matrix or vector of numeric covariates to be adjusted for.

design	design matrix relating to experimental conditions to be preserved, usually the design matrix with all experimental factors and covariates other than the batch effects. Ignored if group is not NULL. Defaults to an intercept column, implying that the experiment is one group.
group	factor defining the experimental conditions to be preserved. An alternative way to specify the design matrix design.
...	other arguments are passed to lmFit.

Details

This function is useful for removing unwanted batch effects, associated with hybridization time or other technical variables, ready for plotting or unsupervised analyses such as PCA, MDS or heatmaps. The design matrix or group factor is used to define comparisons between the samples, for example treatment effects, that should not be removed. The function fits a linear model to the data, including both batches and regular treatments, then removes the component due to the batch effects.

There are no restrictions on the generality of the design matrix, which can contain any combination of factors or continuous covariates. It is usually the same design matrix that would be used for a differential expression analysis but with the unwanted batch factors removed.

In most applications, only the first batch argument will be needed. This case covers the situation where the data has been collected in a series of separate batches.

The batch2 argument is used when there is a second series of batch effects, independent of the first series. For example, batch might correspond to time of data collection while batch2 might correspond to operator or some other change in operating characteristics. If batch2 is included, then the effects of batch and batch2 are assumed to be additive.

The covariates argument allows correction for one or more continuous numeric effects, similar to the analysis of covariance method in statistics. If covariates contains more than one column, then the columns are assumed to have additive effects. Setting covariates to be a design matrix constructed from batch effects and technical effects allows very general batch effects to be accounted for.

The covariates argument allows a completely general approach, which could proceed as follows. If design.full is the full design matrix containing both predictors of interest and unwanted nuisance effects (batches or covariates), and j is a vector of column numbers selecting the predictors of interest, then one could set design.interest <- design[,j], design.nuisance <- design[,-j] and y <- removeBatchEffect(x, design=design.interest, covariates=design.nuisance). The intercept should be included with the predictors of interest, rather than as a nuisance column.

The data object x can be of any class for which lmFit works. If x contains weights, then these will be used in estimating the batch effects.

Value

A numeric matrix of log-expression values with batch and covariate effects removed.

Note

This function is intended for plotting and data exploration purposes. This function is not intended to be used to prepare data for linear modeling by lmFit. For linear modeling, it is better to include

the batch factors in the linear model so that lmFit can correctly assess the standard errors of the linear model parameters.

Author(s)

Gordon Smyth and Carolyn de Graaf

See Also

05.Normalization

Examples

```
ngenes <- 10
nsamples <- 8
y <- matrix(rnorm(ngenes*nsamples),ngenes,nsamples)
group <- factor(c("A","A","A","A","B","B","B","B"))
batch <- factor(c(1,1,2,2,1,1,2,2))
colnames(y) <- paste(group,batch,sep=".")
y[,batch==2] <- y[,batch==2] + 5
y[,group=="B"] <- y[,group=="B"] + 1
y.corrected <- removeBatchEffect(y, batch=batch, group=group)
oldpar <- par(mfrow=c(1,2))
plotMDS(y,main="Original")
plotMDS(y.corrected,main="Batch corrected")
par(oldpar)
devAskNewPage(FALSE)
```

---

removeExt *Remove Common Extension from File Names*

Description

Finds and removes any common extension from a vector of file names.

Usage

```
removeExt(x, sep=".")
```

Arguments

x	character vector
sep	character string that separates the body of each character string from the extension.

Details

This function is used for simplifying file names, or any vector of character strings, when the strings all finish with the same suffix or extension. If the same extension is not shared by every element of x, then it is not removed from any element.

Note that sep is interpreted as a literal character string: it is not a regular expression.

Value

A character vector of the same length as x in which any common extension has been stripped off.

Author(s)

Gordon Smyth

See Also

An overview of LIMMA functions for reading data is given in 03.ReadingData.

Examples

```
x <- c("slide1.spot","slide2.spot","slide3.spot")
removeExt(x)

x <- c("Harry - a name from Harry Potter","Hermione - a name from Harry Potter")
removeExt(x, sep=" - ")
```

---

residuals.MArrayLM *Extract Residuals from MArrayLM Fit*

Description

This method extracts the residuals from all the probewise linear model fits and returns them in a matrix.

Usage

```
## S3 method for class 'MArrayLM'
residuals(object, y, ...)
```

Arguments

object	a fitted model object inheriting from class MArrayLM.
y	a data object containing the response data used to compute the fit. This can be of any class for which as.matrix is defined, including MAList, ExpressionSet, marrayNorm etc.
...	other arguments are not used.

Value

Numeric matrix of residuals.

See Also

residuals.

RGList-class *Red, Green Intensity List - class*

Description

A list-based S4 class for storing red and green channel foreground and background intensities for a batch of spotted microarrays. RGList objects are normally created by read.maimages.

Slots/List Components

RGList objects can be created by new("RGList",RG) where RG is a list. Objects of this class contain no slots (other than .Data), but objects should contain the following list components:

R	numeric matrix containing the red (cy5) foreground intensities. Rows correspond to spots and columns to arrays.
G	numeric matrix containing the green (cy3) foreground intensities. Rows correspond to spots and columns to arrays.

Optional components include

Rb	numeric matrix containing the red (cy5) background intensities
Gb	numeric matrix containing the green (cy3) background intensities
weights	numeric matrix of same dimension as R containing relative spot quality weights. Elements should be non-negative.
other	list containing other matrices, all of the same dimensions as R and G.
genes	data.frame containing probe information. Should have one row for each spot. May have any number of columns.
targets	data.frame containing information on the target RNA samples. Rows correspond to arrays. May have any number
printer	list containing information on the process used to print the spots on the arrays. See PrintLayout.

Valid RGList objects may contain other optional components, but all probe or array information should be contained in the above components.

Methods

This class inherits directly from class list so any operation appropriate for lists will work on objects of this class. In addition, RGList objects can be subsetted, combined and merged. RGList objects will return dimensions and hence functions such as dim, nrow and ncol are defined. RGLists also inherit a show method from the virtual class LargeDataObject, which means that RGLists will print in a compact way.

RGList objects can be converted to exprSet2 objects by as(RG,"exprSet2").

Other functions in LIMMA which operate on RGList objects include normalizeBetweenArrays, normalizeForPrintorder, normalizeWithinArrays.

Author(s)

Gordon Smyth

See Also

02.Classes gives an overview of all the classes defined by this package.

marrayRaw is the corresponding class in the marray package.

---

roast *Rotation Gene Set Tests*

Description

Rotation gene set testing for linear models.

Usage

```
## Default S3 method:
roast(y, index = NULL, design = NULL, contrast = ncol(design), geneid = NULL,
    set.statistic = "mean", gene.weights = NULL, var.prior = NULL, df.prior = NULL,
      nrot = 1999, approx.zscore = TRUE, legacy = FALSE, ...)
## Default S3 method:
mroast(y, index = NULL, design = NULL, contrast = ncol(design), geneid = NULL,
     set.statistic = "mean", gene.weights = NULL, var.prior = NULL, df.prior = NULL,
       nrot = 1999, approx.zscore = TRUE, legacy = FALSE, adjust.method = "BH",
       midp = TRUE, sort = "directional", ...)
## Default S3 method:
fry(y, index = NULL, design = NULL, contrast = ncol(design), geneid = NULL,
     gene.weights = NULL, standardize = "posterior.sd", sort = "directional", ...)
```

Arguments

y	numeric matrix giving log-expression or log-ratio values for a series of microarrays, or any object that can coerced to a matrix including ExpressionSet, MAList, EList or PLMSet objects. Rows correspond to probes and columns to samples. NA or infinite values are not allowed. If either var.prior or df.prior are NULL, then y should contain values for all genes on the arrays. If both prior parameters are given, then only y values for the test set are required.
index	index vector specifying which rows (probes) of y are in the test set. Can be a vector of integer indices, or a logical vector of length nrow(y), or a vector of gene IDs corresponding to entries in geneid. Alternatively it can be a data.frame with the first column containing the index vector and the second column containing directional gene contribution weights. For mroast or fry, index is a list of index vectors or a list of data.frames.
design	design matrix

contrast	contrast for which the test is required. Can be an integer specifying a column of design, or the name of a column of design, or a numeric contrast vector of length equal to the number of columns of design.
geneid	gene identifiers corresponding to the rows of y. Can be either a vector of length nrow(y) or the name of the column of y$genes containing the gene identifiers. Defaults to rownames(y).
set.statistic	summary set statistic. Possibilities are "mean","floormean","mean50" or "msq".
gene.weights	numeric vector of directional (positive or negative) contribution weights specifying the size and direction of the contribution of each probe to the gene set statistics. For mroast or fry, this vector must have length equal to nrow(y). For roast, can be of length nrow(y) or of length equal to the number of genes in the test set.
var.prior	prior value for residual variances. If not provided, this is estimated from all the data using squeezeVar.
df.prior	prior degrees of freedom for residual variances. If not provided, this is estimated using squeezeVar.
nrot	number of rotations used to compute the p-values. Low values like 999 are suitable for testing but higher values such as 9999 or more are recommended for publication purposes.
approx.zscore	logical, if TRUE then a fast approximation is used to convert t-statistics into z-scores prior to computing set statistics. If FALSE, z-scores will be exact.
legacy	logical. See Note below for usage.
adjust.method	method used to adjust the p-values for multiple testing. See p.adjust for possible values.
midp	logical, should mid-p-values be used instead of ordinary p-values when adjusting for multiple testing?
sort	character, whether to sort output table by directional p-value ("directional"), non-directional p-value ("mixed"), or not at all ("none").
standardize	how to standardize for unequal probewise variances. Possibilities are "residual.sd", "posterior.sd" or "none".
...	any argument that would be suitable for lmFit or eBayes can be included.

Details

These functions implement rotation gene set tests proposed by Wu et al (2010). They perform *self-contained* gene set tests in the sense defined by Goeman and Buhlmann (2007). For *competitive* gene set tests, see camera. For a gene set enrichment analysis (GSEA) style analysis using a database of gene sets, see romer.

roast and mroast test whether any of the genes in the set are differentially expressed. They can be used for any microarray experiment that can be represented by a linear model. The design matrix for the experiment is specified as for the lmFit function, and the contrast of interest is specified as for the contrasts.fit function. This allows users to focus on differential expression for any coefficient or contrast in a linear model. If contrast is not specified, then the last coefficient in the linear model will be tested.

The argument index is often made using ids2indices but does not have to be. Each set to be tested is represented by a vector of row numbers or a vector of gene IDs. Gene IDs should correspond to either the rownames of y or the entries of geneid.

All three functions support directional contribution gene weights, which can be entered either through the gene.weights argument or via index. Directional gene weights allow each gene to be flagged as to its direction and magnitude of change based on prior experimentation. A typical use is to make the gene.weights 1 or -1 depending on whether the gene is up or down-regulated in the pathway under consideration. Probes with directional weights of opposite signs are expected to have expression changes in opposite directions. Genes with larger gene weights in absolute size will have more weight in the set statistic calculation.

Gene weights can be either genome-wide or set-specific. Genome-wide weights can be entered via the gene.weights argument. Set specific weights can be input by including the gene weights as part of the set's entry in index. If any of the components of index are data.frames, then the second column will be assumed to be gene contribution weights for that set. All three functions (roast, mroast and fry) support set-specific gene contribution weights as part of an index data.frame.

Set-specific directional gene weights are used to represent *expression signatures* assembled from previous experiments, from gene annotation or from prior hypotheses. In the output from roast, mroast or fry, a significant "Up" p-value means that the differential expression results found in y are positively correlated with the expression signature coded by the gene weights. Conversely, a significant "Down" p-value means that the differential expression log-fold-changes are negatively correlated with the expression signature.

Note that the contribution weights set by gene.weights are different in nature and purpose to the precision weights set by the weights argument of lmFit. gene.weights control the contribution of each gene to the formation of the gene set statistics and are directional, i.e., can be positive or negative. weights indicate the precision of the expression measurements and should be positive. The weights are used to construct genewise test statistics whereas gene.weights are used to combine the genewise test statistics.

The arguments df.prior and var.prior have the same meaning as in the output of the eBayes function. If these arguments are not supplied, then they are estimated exactly as is done by eBayes.

The gene set statistics "mean", "floormean", "mean50" and msq are defined by Wu et al (2010). The different gene set statistics have different sensitivities when only some of the genes in a set are differentially expressed. If set.statistic="mean" then the set will be statistically significant only when the majority of the genes are differentially expressed. "floormean" and "mean50" will detect as few as 25% differentially expressed in a set. "msq" is sensitive to even smaller proportions of differentially expressed genes, if the effects are reasonably large. Overall, the "msq" statistic gives the best power for rejecting the null hypothesis of no differentially expressed genes, but the significance can be driven by a small number of genes. In many genomic applications it is appropriate to limit results to gene sets for which most of the genes respond in a concordant direction, so the relatively conservative "mean" statistic is the default choice.

The output gives p-values for three possible alternative hypotheses, "Up" to test whether the genes in the set tend to be up-regulated, with positive t-statistics, "Down" to test whether the genes in the set tend to be down-regulated, with negative t-statistics, and "Mixed" to test whether the genes in the set tend to be differentially expressed, without regard for direction.

roast estimates p-values by simulation, specifically by random rotations of the orthogonalized residuals (Langsrud, 2005), so p-values will vary slightly from run to run. The p-value is computed as (b+1)/(nrot+1) where b is the number of rotations giving a more extreme statistic than that

observed (Phipson and Smyth, 2010). This means that the smallest possible mixed or two-sided p-values are `1/(nrot+1)`. The function uses a symmetry argument to double the effective number of rotations for the one-sided tests, so the smallest possible `"Up"` or `"Down"` p-value is `1/(2*nrot+1)`.

The number of rotations `nrot` can (and should) be increased to get more precise p-values from `roast` or `mroast`. The default `nrot` is set fairly low to facilitate quick testing and experimentation but the smallest possible two-sided p-value is `1/(nrot+1)`. To get definitive p-values for publication, at least `nrot=9999` or higher is recommended.

`mroast` does roast tests for multiple sets, including adjustment for multiple testing. By default, `mroast` reports ordinary p-values but uses mid-p-values (Routledge, 1994) at the multiple testing stage. Mid-p-values are probably a good choice when using false discovery rates (`adjust.method="BH"`) but not when controlling the family-wise type I error rate (`adjust.method="holm"`).

To improve the performance of the gene set statistics, `roast` and `mroast` transform the genewise moderated t-statistics to normality using `zscoreT`. By default, an approximate closed-form transformation is used (`approx.zscore=TRUE`), which is very much faster than the exact transformation and performs just as well. In Bioconductor 2.10, the transformation used has been changed from Hill's (1970) approximation to Bailey's (1980) formula because the latter is faster and gives more even accuracy; see `zscoreT` for more details.

`fry` is a fast alternative designed to approximate what `mroast` with `set.stat="mean"` would give for a very large (infinite) number of rotations. In the special case that `df.prior` is very large and `set.statistic="mean"`, `fry` gives the same directional p-values that `mroast` would give if an infinite number of rotations could be performed. In other circumstances, when genes have different variances, `fry` uses a standardization strategy to approximate the `mroast` results. Using `fry` is recommended when performing tests for a large number of sets because it is fast and because it returns higher resolution p-values that are not limited by the number of rotations performed. Note, the close approximation of `fry` to `mroast` is only for the directional p-values. The `fry` mixed p-values are computed by a different method and will not necessarily be very close to those from `mroast`.

Value

`roast` produces an object of class `"Roast"`. This consists of a list with the following components:

`p.value`	data.frame with columns `Active.Prop` and `P.Value`, giving the proportion of genes in the set contributing materially to significance and estimated p-values, respectively. Rows correspond to the alternative hypotheses Down, Up, UpOrDown (two-sided) and Mixed.
`var.prior`	prior value for residual variances.
`df.prior`	prior degrees of freedom for residual variances.

`mroast` produces a data.frame with a row for each set and the following columns:

`NGenes`	number of genes in set
`PropDown`	proportion of genes in set with `z < -sqrt(2)`
`PropUp`	proportion of genes in set with `z > sqrt(2)`
`Direction`	direction of change, `"Up"` or `"Down"`
`PValue`	two-sided directional p-value

FDR	two-sided directional false discovery rate
PValue.Mixed	non-directional p-value
FDR.Mixed	non-directional false discovery rate

fry produces the same output format as mroast but without the columns PropDown and PropUp.

Note

For Bioconductor 3.10, roast and mroast have been revised to use much less memory by conducting the rotations in chunks and to be about twice as fast by updating the normalizing transformation used when approx.zscore=TRUE. For a limited time, users wishing to reproduce Bioconductor 3.9 results exactly can set legacy=TRUE to turn these revisions off.

approx.zscore=TRUE became the default in Bioconductor 3.0 (October 2014).

The default set statistic was changed from "msq" to "mean" in Bioconductor 2.7 (October 2010).

Author(s)

Gordon Smyth and Di Wu

References

Goeman JJ, Buhlmann P (2007). Analyzing gene expression data in terms of gene sets: methodological issues. *Bioinformatics* 23, 980-987.

Langsrud O (2005). Rotation tests. *Statistics and Computing* 15, 53-60.

Phipson B, Smyth GK (2010). Permutation P-values should never be zero: calculating exact P-values when permutations are randomly drawn. *Statistical Applications in Genetics and Molecular Biology* 9(1), Article 39. doi:10.2202/15446115.1585. See also the preprint version https://gksmyth.github.io/pubs/PermPValuesPreprint.pdf with corrections.

Routledge, RD (1994). Practicing safe statistics with the mid-p. *Canadian Journal of Statistics* 22, 103-110.

Wu D, Lim E, Vaillant F, Asselin-Labat M-L, Visvader JE, Smyth GK (2010). ROAST: rotation gene set tests for complex microarray experiments. *Bioinformatics* 26, 2176-2182. doi:10.1093/bioinformatics/btq401.

See Also

See 10.GeneSetTests for a description of other functions used for gene set testing.

Examples

```
y <- matrix(rnorm(100*4,sd=0.3),100,4)
design <- cbind(Intercept=1,Group=c(0,0,1,1))

# First set of 5 genes are all up-regulated
index1 <- 1:5
y[index1,3:4] <- y[index1,3:4]+3
roast(y,index1,design,contrast=2)
```

```
# Second set of 5 genes contains none that are DE
index2 <- 6:10
mroast(y,list(set1=index1,set2=index2),design,contrast=2)
fry(y,list(set1=index1,set2=index2),design,contrast=2)

# Third set of 6 genes contains three down-regulated genes and three up-regulated genes
index3 <- 11:16
y[index3[1:3],3:4] <- y[index3[1:3],3:4]-3
y[index3[4:6],3:4] <- y[index3[4:6],3:4]+3

# Without gene weights
# Mixed p-value is significant for set3 but not the directional p-values
mroast(y,list(set1=index1,set2=index2,set3=index3),design,contrast=2)
fry(y,list(set1=index1,set2=index2,set3=index3),design,contrast=2)

# With gene weights
# Set3 is significantly up (i.e., positively correlated with the weights)
index3 <- data.frame(Gene=11:16,Weight=c(-1,-1,-1,1,1,1))
mroast(y,list(set1=index1,set2=index2,set3=index3),design,contrast=2)
fry(y,list(set1=index1,set2=index2,set3=index3),design,contrast=2)
```

`romer` *Rotation Gene Set Enrichment Analysis*

Description

Gene set enrichment analysis for linear models using rotation tests (ROtation testing using MEan Ranks).

Usage

```
## Default S3 method:
romer(y, index, design = NULL, contrast = ncol(design),
      array.weights = NULL, block = NULL, correlation,
      set.statistic = "mean", nrot = 9999, shrink.resid = TRUE, ...)
```

Arguments

`y`	numeric matrix giving log-expression values.
`index`	list of indices specifying the rows of `y` in the gene sets. The list can be made using `ids2indices`.
`design`	design matrix.
`contrast`	contrast for which the test is required. Can be an integer specifying a column of `design`, or else a contrast vector of length equal to the number of columns of `design`.
`array.weights`	optional numeric vector of array weights.
`block`	optional vector of blocks.

correlation	correlation between blocks.
set.statistic	statistic used to summarize the gene ranks for each set. Possible values are "mean", "floormean" or "mean50".
nrot	number of rotations used to estimate the p-values.
shrink.resid	logical, should the residuals be shrunk to remove systematic effects before rotation.
...	other arguments not currently used.

Details

This function implements the ROMER procedure described by Majewski et al (2010) and Ritchie et al (2015). romer tests a hypothesis similar to that of Gene Set Enrichment Analysis (GSEA) (Subramanian et al, 2005) but is designed for use with linear models. Like GSEA, it is designed for use with a database of gene sets. Like GSEA, it is a competitive test in that the different gene sets are pitted against one another. Instead of permutation, it uses rotation, a parametric resampling method suitable for linear models (Langsrud, 2005; Wu et al, 2010). romer can be used with any linear model with some level of replication.

In the output, p-values are given for each set for three possible alternative hypotheses. The alternative "up" means the genes in the set tend to be up-regulated, with positive t-statistics. The alternative "down" means the genes in the set tend to be down-regulated, with negative t-statistics. The alternative "mixed" tests whether the genes in the set tend to be differentially expressed, without regard for direction. In this case, the test will be significant if the set contains mostly large test statistics, even if some are positive and some are negative. The first two alternatives are appropriate if you have a prior expectation that all the genes in the set will react in the same direction. The "mixed" alternative is appropriate if you know only that the genes are involved in the relevant pathways, without knowing the direction of effect for each gene.

Note that romer estimates p-values by simulation, specifically by random rotations of the orthogonalized residuals (called effects in R). This means that the p-values will vary slightly from run to run. To get more precise p-values, increase the number of rotations nrot. By default, the orthogonalized residual corresponding to the contrast being tested is shrunk to have the same expected squared size as a null residual.

The argument set.statistic controls the way that t-statistics are summarized to form a summary test statistic for each set. In all cases, genes are ranked by moderated t-statistic. If set.statistic="mean", the mean-rank of the genes in each set is the summary statistic. If set.statistic="floormean" then negative t-statistics are put to zero before ranking for the up test, and vice versa for the down test. This improves the power for detecting genes with a subset of responding genes. If set.statistic="mean50", the mean of the top 50% ranks in each set is the summary statistic. This statistic performs well in practice but is slightly slower to compute. See Wu et al (2010) for discussion of these set statistics.

Value

Numeric matrix giving p-values and the number of matched genes in each gene set. Rows correspond to gene sets. There are four columns giving the number of genes in the set and p-values for the alternative hypotheses mixed, up or down.

Author(s)

Yifang Hu and Gordon Smyth

References

Langsrud O (2005). Rotation tests. *Statistics and Computing* 15, 53-60

Majewski IJ, Ritchie ME, Phipson B, Corbin J, Pakusch M, Ebert A, Busslinger M, Koseki H, Hu Y, Smyth GK, Alexander WS, Hilton DJ, Blewitt ME (2010). Opposing roles of polycomb repressive complexes in hematopoietic stem and progenitor cells. *Blood* 116, 731-739. doi:10.1182/blood-200912260760.

Ritchie ME, Phipson B, Wu D, Hu Y, Law CW, Shi W, Smyth GK (2015). limma powers differential expression analyses for RNA-sequencing and microarray studies. *Nucleic Acids Research* 43, e47. doi:10.1093/nar/gkv007.

Subramanian A, Tamayo P, Mootha VK, Mukherjee S, Ebert BL, Gillette MA, Paulovich A, Pomeroy SL, Golub TR, Lander ES, Mesirov JP (2005). Gene set enrichment analysis: a knowledge-based approach for interpreting genome-wide expression profiles. *PNAS* 102, 15545-15550

Wu D, Lim E, Vaillant F, Asselin-Labat M-L, Visvader JE, Smyth GK (2010). ROAST: rotation gene set tests for complex microarray experiments. *Bioinformatics* 26, 2176-2182. doi:10.1093/bioinformatics/btq401.

See Also

`topRomer`, `ids2indices`, `roast`, `camera`, `wilcoxGST`

There is a topic page on 10.GeneSetTests.

Examples

```
y <- matrix(rnorm(100*4),100,4)
design <- cbind(Intercept=1,Group=c(0,0,1,1))
index <- 1:5
y[index,3:4] <- y[index,3:4]+3

index1 <- 1:5
index2 <- 6:10
r <- romer(y=y,index=list(set1=index1,set2=index2),design=design,contrast=2,nrot=99)
r
topRomer(r,alt="up")
topRomer(r,alt="down")
```

---

sampleInfoFromGEO *Read Sample Characteristics From GEO Series Matrix File*

Description

Read sample annotation from a GEO Series Matrix File into data.frames.

Usage

```
sampleInfoFromGEO(file, remove.constant.columns = TRUE)
```

Arguments

`file`	file name or path of GEO series matrix file.
`remove.constant.columns`	logical, if `TRUE` then columns that have the same entry for every sample are removed.

Details

This function parses a GEO series matrix file. Sample characteristics associated with expression channels 1 and 2 are separated into separate character matrices. The function particularly allows for the fact that not every sample characteristic will have an entry for every sample.

Value

A list with three components:

`SampleInfo`	character matrix of sample annotation.
`CharacteristicsCh1`	character matrix of sample characteristics associated with expression channel 1.
`CharacteristicsCh2`	character matrix of sample characteristics associated with expression channel 2.

Each sample corresponds to one row.

Author(s)

Gordon Smyth

See Also

An overview of LIMMA functions for reading data is given in 03.ReadingData.

Examples

```
# This example downloads a series matrix file of about 33MB
## Not run:
url <- paste0("https://ftp.ncbi.nlm.nih.gov/geo/series/GSE40nnn/",
              "GSE40115/matrix/GSE40115-GPL15931_series_matrix.txt.gz")
download.file(url, "GSE40115.txt.gz")
a <- sampleInfoFromGEO("GSE40115.txt.gz")
colnames(a$SampleInfo)
colnames(a$CharacteristicsCh1)
colnames(a$CharacteristicsCh2)

## End(Not run)
```

`selectModel` *Select Appropriate Linear Model*

Description

Select the best fitting linear model for each gene by minimizing an information criterion.

Usage

```
selectModel(y, designlist, criterion="aic", df.prior=0, s2.prior=NULL, s2.true=NULL, ...)
```

Arguments

`y`	a matrix-like data object, containing log-ratios or log-values of expression for a series of microarrays. Any object class which can be coerced to matrix is acceptable including `numeric`, `matrix`, `MAList`, `marrayNorm`, `ExpressionSet` or `PLMset`.
`designlist`	list of design matrices
`criterion`	information criterion to be used for model selection, `"aic"`, `"bic"` or `"mallowscp"`.
`df.prior`	prior degrees of freedom for residual variances. See `squeezeVar`
`s2.prior`	prior value for residual variances, to be used if `df.prior`>0.
`s2.true`	numeric vector of true variances, to be used if `criterion="mallowscp"`.
`...`	other optional arguments to be passed to `lmFit`

Details

This function chooses, for each probe, the best fitting model out of a set of alternative models represented by a list of design matrices. Selection is by Akaike's Information Criterion (AIC), Bayesian Information Criterion (BIC) or by Mallow's Cp.

The criteria have been generalized slightly to accommodate an information prior on the variances represented by `s2.prior` and `df.prior` or by `s2.post`. Suitable values for these parameters can be estimated using `squeezeVar`.

Value

List with components

`IC`	matrix of information criterion scores, rows for probes and columns for models
`pref`	factor indicating the model with best (lowest) information criterion score

Author(s)

Alicia Oshlack and Gordon Smyth

See Also

An overview of linear model functions in limma is given by 06.LinearModels.

Examples

```
nprobes <- 100
narrays <- 5
y <- matrix(rnorm(nprobes*narrays),nprobes,narrays)
A <- c(0,0,1,1,1)
B <- c(0,1,0,1,1)
designlist <- list(
  None=cbind(Int=c(1,1,1,1,1)),
  A=cbind(Int=1,A=A),
  B=cbind(Int=1,B=B),
  Both=cbind(Int=1,AB=A*B),
  Add=cbind(Int=1,A=A,B=B),
  Full=cbind(Int=1,A=A,B=B,AB=A*B)
)
out <- selectModel(y,designlist)
table(out$pref)
```

---

squeezeVar *Squeeze Sample Variances*

Description

Squeeze a set of sample variances together by computing empirical Bayes posterior means.

Usage

```
squeezeVar(var, df, covariate = NULL, span = NULL,
           robust = FALSE, winsor.tail.p = c(0.05,0.1), legacy = NULL)
```

Arguments

var	numeric vector of independent sample variances.
df	numeric vector of degrees of freedom for the sample variances. Can be a unit vector or of same length as var.
covariate	numeric covariate of same length as var for estimating a trended prior variance. If NULL, then the prior variance var.prior is constant.
span	lowess span used for prior variance trend.
robust	logical, should the estimation of df.prior and var.prior be robustified against outlier sample variances?
winsor.tail.p	numeric vector of length 1 or 2, giving left and right tail proportions of x to Winsorize when robust=TRUE.

legacy — logical. If FALSE then the new function fitFDistUnequalDF1 will be called internally, otherwise the legacy functions fitFDist or fitFDistRobustly will be used. If NULL, then fitFDistUnequalDF1 will be used whenever the degrees of freedom df are not all equal. If span is set, then this argument will be ignored and fitFDistUnequalDF1 will be used.

Details

This function implements empirical Bayes algorithms proposed by Smyth (2004) and Phipson et al (2016).

A conjugate Bayesian hierarchical model is assumed for a set of sample variances. The hyperparameters are estimated by fitting a scaled F-distribution to the sample variances. The function returns the posterior variances and the estimated hyperparameters.

Specifically, the sample variances var are assumed to follow scaled chi-squared distributions, conditional on the true variances, and a scaled inverse chi-squared prior is assumed for the true variances. The scale and degrees of freedom of this prior distribution are estimated from the values of var.

The effect of this function is to squeeze the variances towards a common value, or to a global trend if a covariate is provided. The squeezed variances have a smaller expected mean square error to the true variances than do the sample variances themselves.

The amount of squeezing is controlled by the df.prior. Both the global trend and the prior df are estimated internally by fitting an F-distribution to the sample variances, using either fitFDist() or fitFDistRobustly() or fitFDistUnequalDF1().

If covariate is non-null, then the scale parameter of the prior distribution is assumed to depend on the covariate. If the covariate is average log-expression, then the effect is an intensity-dependent trend similar to that in Sartor et al (2006).

robust=TRUE implements the robust empirical Bayes procedure of Phipson et al (2016), which allows some of the var values to be outliers.

The legacy argument was added in limma version 3.61.8 (August 2024). If legacy=FALSE, then the new function fitFDistUnequalDF1() provides improved estimation of the global trend and prior df hyperparameters, especially when the df values are unequal. legacy=TRUE provides legacy behavior for backward compatibility.

Value

A list with components

var.post — numeric vector of posterior variances. Of same length as var.

var.prior — location or scale of prior distribution. A vector of same length as var if covariate is non-NULL, otherwise a single value.

df.prior — degrees of freedom of prior distribution. A vector of same length as var if robust=TRUE, otherwise a single value.

Note

This function is called by eBayes, but beware a possible confusion with the output from that function. The values var.prior and var.post output by squeezeVar correspond to the quantities s2.prior and s2.post output by eBayes, whereas var.prior output by eBayes relates to a different parameter.

Author(s)

Gordon Smyth

References

Phipson B, Lee S, Majewski IJ, Alexander WS, Smyth GK (2016). Robust hyperparameter estimation protects against hypervariable genes and improves power to detect differential expression. *Annals of Applied Statistics* 10, 946-963. doi:10.1214/16AOAS920.

Sartor MA, Tomlinson CR, Wesselkamper SC, Sivaganesan S, Leikauf GD, Medvedovic M (2006). Intensity-based hierarchical Bayes method improves testing for differentially expressed genes in microarray experiments. *BMC bioinformatics* 7, 538.

Smyth GK (2004). Linear models and empirical Bayes methods for assessing differential expression in microarray experiments. *Statistical Applications in Genetics and Molecular Biology* 3(1), Article 3. doi:10.2202/15446115.1027. See also the preprint version https://gksmyth.github.io/pubs/ebayes.pdf incorporating corrections to 30 June 2009.

See Also

This function is called by eBayes.

This function calls fitFDist, fitFDistRobustly or fitFDistUnequalDF1.

An overview of linear model functions in limma is given by 06.LinearModels.

Examples

```
s2 <- rchisq(20,df=5)/5
squeezeVar(s2, df=5)
```

---

strsplit2 *Split Composite Names*

Description

Split a vector of composite names into a matrix of simple names.

Usage

```
strsplit2(x, split, fixed = TRUE, ...)
```

Arguments

x	character vector
split	character to split each element of vector on, see strsplit
fixed	logical. If FALSE, then split will be treated as a regular expression, if TRUE then as a literal character string.
...	other arguments are passed to strsplit

Details

This function is the same as strsplit except that the output value is a matrix instead of a list. The elements of x are treated as delimited text with fields separated by split. The first column of the output matrix contains the first field from each element of x, the second column contains the second field etc. The number of columns is equal to the maximum number of fields for any element of x. If different rows have different numbers of fields, then trailing fields are filled out with "".

The motivation for this function in the limma package is to handle input columns which are composites of two or more annotation fields.

Value

A list containing components

Name	character vector of the same length as x containing first splits of each element
Annotation	character vector of the same length as x containing second splits of each element

Author(s)

Gordon Smyth

See Also

strsplit.

An overview of LIMMA functions for reading data is given in 03.ReadingData.

Examples

```
x <- c("AA196000;actinin, alpha 3",
"AA464163;acyl-Coenzyme A dehydrogenase, very long chain",
"3E7;W15277;No Annotation")
strsplit2(x,split=";")
```

---

subsetting *Subset RGList, MAList, EListRaw, EList, MArrayLM or TestResults Objects*

Description

Return an RGList, MAList, EListRaw, EList, MArrayLM or TestResults object with only selected rows and columns of the original object.

Usage

```
## S3 method for class 'EList'
object[i, j, ...]
subsetListOfArrays(object, i, j, IJ, IX, I, JX)
```

Arguments

object	object of class RGList, MAList, EListRaw, EList, MArrayLM or TestResults.
i, j	elements to extract. i subsets the probes or spots while j subsets the arrays.
IJ	character vector giving names of components that should be subsetted by i and j.
IX	character vector giving names of 2-dimensional components that should be subsetted by i only.
I	character vector giving names of vector components that should be subsetted by i.
JX	character vector giving names of 2-dimensional components whose row dimension corresponds to j.
...	other arguments are not currently used.

Details

All these objects can be subsetted as if they were matrices. i,j may take any values acceptable for the matrix components of object. Either or both can be missing. See the Extract help entry for more details on subsetting matrices.

object[] will return the whole object unchanged. A single index object[i] will be taken to subset rows, so object[i] and object[i,] are equivalent.

subsetListOfArrays is used internally as a utility function by the subsetting operations. It is not intended to be called directly by users. Values must be supplied for all arguments other than i and j.

Value

An object the same as object but containing data from the specified subset of rows and columns only.

Note the output object is of the same class as object and will have two dimensions attached even if i or j select a single row or column, i.e., subsetting for these objects does not drop dimensions. Subsetting is exactly analogous to subsetting of matrices in R with drop=FALSE.

Author(s)

Gordon Smyth

See Also

Extract in the base package.

02.Classes for a summary of the different data classes.

Examples

```
M <- A <- matrix(11:14,4,2)
rownames(M) <- rownames(A) <- c("a","b","c","d")
colnames(M) <- colnames(A) <- c("A","B")
MA <- new("MAList",list(M=M,A=A))
MA[1:2,]
MA[c("a","b"),]
MA[1:2,2]
MA[,2]
```

`summary` *Summaries of Microarray Data Objects*

Description

Briefly summarize microarray data objects.

Usage

```
## S3 method for class 'RGList'
summary(object, ...)
```

Arguments

`object`	an object of class `RGList`, `MAList`, `EListRaw`, `EList` or `MArrayLM`
`...`	other arguments are not used

Details

The data objects are summarized as if they were lists, i.e., brief information about the length and type of the components is given.

Value

A table.

Author(s)

Gordon Smyth

See Also

`summary` in the base package.

02.Classes gives an overview of data classes used in LIMMA.

`targetsA2C` *Convert Two-Color Targets Dataframe from One-Row-Per-Array to One-Row-Per-Channel*

Description

Convert a two-color targets dataframe with one row per array to one with one row per channel.

Usage

```
targetsA2C(targets, channel.codes = c(1,2), channel.columns = list(Target=c("Cy3","Cy5")),
           grep = FALSE)
```

Arguments

`targets`	data.frame with one row per array giving information about target samples and associated covariates.
`channel.codes`	numeric or character vector of length 2 giving codes for the channels
`channel.columns`	named list of character vectors of length 2. Each entry gives a pair of names of columns in `targets` which contain channel-specific information. This pair of columns should be assembled into one column in the output.
`grep`	logical, if `TRUE` then the channel column names are found by `grep`ing, i.e., the actual column names need only contain the names given by `channel.columns` as substrings

Details

The `targets` dataframe holds information about the RNA samples used as targets in the microarray experiment. It is often read from a file using `readTargets`. This function is used to convert the dataframe from an array-oriented format with one row for each array and two columns for the two channels into a channel-oriented format with one row for each individual channel observation. In statistical terms, the first format treats the arrays as cases and treats the channels as repeated measurements. The second format treats the individual channel observations as cases. The second format may be more appropriate if the data is to be analyzed in terms of individual log-intensities.

Value

data.frame with twice as many rows as `targets`. Any pair of columns named by `channel.columns` will now be one column.

Author(s)

Gordon Smyth

References

Smyth GK, Altman NS (2013). Separate-channel analysis of two-channel microarrays: recovering inter-spot information. *BMC Bioinformatics* 14, 165. doi:10.1186/1471210514165.

See Also

`targetsA2C` is used by the `coerce` method from `RGList` to `ExpressionSet` in the convert package.

An overview of methods for single channel analysis in limma is given by 07.SingleChannel.

Examples

```
targets <- data.frame(FileName=c("file1.gpr","file2.gpr"),Cy3=c("WT","KO"),Cy5=c("KO","WT"))
targetsA2C(targets)
```

TestResults-class *Matrix of Test Results - class*

Description

A matrix-based class for storing the results of simultaneous tests. `TestResults` objects are usually created by `decideTests`.

Usage

```
## S3 method for class 'TestResults'
summary(object, ...)
## S3 method for class 'TestResults'
labels(object, ...)
## S3 method for class 'TestResults'
levels(x)
```

Arguments

`object, x`	object of class `TestResults`
`...`	other arguments are not used

Slots/List Components

A `TestResults` object is essentially a numeric matrix with elements equal to `0`, `1` or `-1`. Zero represents acceptance of the null hypothesis, `1` indicates rejection in favor of the right tail alternative and `-1` indicates rejection in favor of the left tail alternative.

`TestResults` objects can be created by `new("TestResults",results)` where `results` is a matrix. Objects of this class contain no slots (other than `.Data`), although the attributes `dim` and `dimnames` may be treated as slots.

Methods

This class inherits directly from class `matrix` so any operation appropriate for matrices will work on objects of this class. `show` and `summary` methods are also implemented.

Functions in LIMMA which operate on `TestResults` objects include `heatDiagram`, `vennCounts`, `vennDiagram`, `write.fit`.

Author(s)

Gordon Smyth

See Also

02.Classes gives an overview of all the classes defined by this package. 08.Tests gives an overview of multiple testing.

Examples

```
## Not run:
#  Assume a data object y and a design matrix
fit <- lmFit(y, design)
fit <- eBayes(fit)
results <- decideTests(fit)
summary(results)

## End(Not run)
```

---

tmixture *Estimate Scale Factor in Mixture of t-Distributions*

Description

These functions estimate the unscaled standard deviation of the true (unobserved) log fold changes for differentially expressed genes. They are used internally by the `eBayes` function and are not intended to be called directly by users.

Usage

```
tmixture.vector(tstat, stdev.unscaled, df, proportion, v0.lim = NULL)
tmixture.matrix(tstat, stdev.unscaled, df, proportion, v0.lim = NULL)
```

Arguments

`tstat`	numeric vector or matrix of t-statistics. `tmixture.vector` assumes a vector while `tmixture.matrix` assumes a matrix.
`stdev.unscaled`	numeric vector or matrix, conformal with `tstat`, containing the unscaled standard deviations of the coefficients used to compute the t-statistics.
`df`	numeric vector giving the degrees of freedom associated with `tstat`.

proportion	assumed proportion of genes that are differentially expressed.
v0.lim	numeric vector of length 2 giving the lower and upper limits for the estimated unscaled standard deviations.

Details

The values in each column of tstat are assumed to follow a mixture of an ordinary t-distribution, with mixing proportion 1-proportion, and (v0+v1)/v1 times a t-distribution, with mixing proportion proportion. Here v1 is stdev.unscaled^2 and v0 is the value to be estimated.

Value

Numeric vector, of length equal to the number of columns of tstat, containing estimated v0 values.

Author(s)

Gordon Smyth

See Also

eBayes

---

topGO *Table of Top GO Terms or Top KEGG Pathways*

Description

Extract top GO terms from goana output or top KEGG pathways from kegga output.

Usage

```
topGO(results, ontology = c("BP", "CC", "MF"), sort = NULL, number = 20L,
      truncate.term = NULL, p.value = 1)
topKEGG(results, sort = NULL, number = 20L, truncate.path = NULL, p.value = 1)
```

Arguments

results	data frame produced by goana or kegga.
ontology	character vector of ontologies to be included in output. Elements should be one or more of "BP", "CC" or "MF".
sort	character vector of names of gene lists for which results are required. Should be one or more of the column names of results. Defaults to all gene lists.
number	maximum number of top GO terms or top KEGG pathways to list. For all terms or all pathways, set number=Inf.
truncate.term	truncate the name of the GO term at this number of characters.
truncate.path	truncate the name of the KEGG pathway at this number of characters.
p.value	p.value cutoff. Only GO terms or pathways with lower p-values are included in the output.

Details

topGO organizes the output from goana into top-tables of the most significant GO terms. topKEGG similarly extracts the most significant KEGG pathways from kegga output. In either case, rows are sorted by the minimum p-value of any of the result columns specified by sort.

Value

Same as results but with rows subsetted by Ontology and sorted by p-value.

Author(s)

Gordon Smyth and Yifang Hu

See Also

goana, kegga

See 10.GeneSetTests for a description of other functions used for gene set testing.

Examples

```
# See goana examples
```

---

topRomer *Top Gene Set Testing Results from Romer*

Description

Extract a matrix of the top gene set testing results from the romer output.

Usage

```
topRomer(x,n=10,alternative="up")
```

Arguments

x	matrix which is the output from romer.
n	number of top gene set testing results to be extracted.
alternative	character which can be one of the three possible alternative p values: "up", "down" or "mixed".

Details

This function takes the results from romer and returns a number of top gene set testing results that are sorted by the p values.

Value

matrix, which is sorted by the "up", "down" or "mixed" p values, with the rows corresponding to estimated p-values for the top number of gene sets and the columns corresponding to the number of genes for each gene set and the alternative hypotheses mixed, up, down.

Author(s)

Gordon Smyth and Yifang Hu

See Also

romer

There is a topic page on 10.GeneSetTests.

Examples

```
# See romer for examples
```

---

topSplice *Top Table of Differential Transcript Usage*

Description

Top table of results from a diffSplice differential transcript usage analysis.

Usage

```
topSplice(fit, coef = ncol(fit), test = "F", number = 10,
          FDR = 1, sort.by = "p", treat.lfc = 0)
```

Arguments

fit	MArrayLM fit object produced by diffSplice.
coef	the coefficient (column) of fit for which differential usage is assessed.
test	character string specifying which statistical test to apply. Possible values are "simes", "F" or "t". "F" gives F-tests for each gene. "t" gives t-tests for each transcript or exon. "simes" gives genewise p-values derived from the t-tests after Simes adjustment for each gene.
number	integer, maximum number of rows to output.
FDR	numeric, only show exons or genes with false discovery rate less than this cutoff.
sort.by	character string specifying which column to sort results by. Possible values are "p", "logFC", "NExons" or "none". "logFC" is only available if test="t" and "NExons" is only available if test="simes" or test="F".
treat.lfc	a treat-style log2-fold-change threshold to apply to significant testing when test="t". The null hypothesis will be that the true log2-fold-change is no larger than this threshold.

Details

This function ranks results from a differential transcript usage analysis conducted by `diffSplice`. The output object from `diffSplice` includes results at the gene level and results at the isoform level. The isoform-level results can be transcripts, if a DTU analysis was done, or can be exons and exon-junctions, if a DEU analysis was done.

Choosing `test="F"` or `test="simes"` gives gene-level results, where the null hypothesis is that all transcripts or exons are used equally. Choosing `test="t"` gives transcript or exon level results.

The F-statistic option chosen by `test="F"` conducts an ANOVA-like F-test of differential usage for each gene. The Simes method chosen by `test="simes"` processes the isoform-level p-values for each gene to give an overall call of differential usage for that gene. The Simes p-values are likely to be more powerful than F-tests when only one transcript of the gene is differentially expressed. The F-tests are likely to be more powerful than Simes when there are multiple transcripts of the same gene differentially expressed in different directions.

Value

A data.frame with annotation columns found in `fit` plus the following columns

`logFC`	log2-fold change of transcript vs other transcripts for the same gene (if `test="t"`)
`t`	moderated t-statistic (if `test="t"`)
`F`	moderated F-statistic (if `test="F"`)
`P.Value`	p-value
`FDR`	false discovery rate bound, computed by the Benjamini & Hochberg method.

Author(s)

Gordon Smyth

References

Baldoni PL, Chen L, Li M, Chen Y, Smyth GK (2025). Dividing out quantification uncertainty enables assessment of differential transcript usage with limma and edgeR. *Nucleic Acids Research* 53(22), gkaf1305. doi:10.1093/nar/gkaf1305.

See Also

`diffSplice`, `plotSplice`. See also the `diffSplice` method for `DGEGLM` objects in the edgeR package.

A summary of functions available in LIMMA for RNA-seq analysis is given in 11.RNAseq.

Examples

```
# See diffSplice
```

topTable *Table of Top Genes from Linear Model Fit*

Description

Extract a table of the top-ranked genes from a linear model fit.

Usage

```
topTable(fit, coef = NULL, number = 10, genelist = fit$genes,
         adjust.method = "BH", sort.by = "B", resort.by = NULL,
         p.value = 1, fc = NULL, lfc = NULL, confint = FALSE)
topTreat(fit, coef = 1, sort.by = "p", resort.by = NULL, ...)
```

Arguments

fit	list containing a linear model fit produced by lmFit, lm.series, gls.series or mrlm. For topTable, fit should be an object of class MArrayLM as produced by lmFit and eBayes.
coef	column number or column name specifying which coefficient or contrast of the linear model is of interest. For topTable, can also be a vector of column subscripts, in which case the gene ranking is by F-statistic for that set of contrasts.
number	maximum number of genes to list
genelist	data frame or character vector containing gene information. For topTable only, this defaults to fit$genes.
adjust.method	method used to adjust the p-values for multiple testing. Options, in increasing conservatism, include "none", "BH", "BY" and "holm". See p.adjust for the complete list of options. A NULL value will result in the default adjustment method, which is "BH".
sort.by	character string specifying which statistic to rank the genes by. Possible values for topTable are "logFC", "AveExpr", "t", "P", "p", "B" or "none". (Permitted synonyms are "M" for "logFC", "A" or "Amean" for "AveExpr", "T" for "t" and "p" for "P".) topTreat accepts the same values as topTable except for "B".
resort.by	character string specifying statistic to sort the selected genes by in the output data.frame. Possibilities are the same as for sort.by.
p.value	cutoff value for adjusted p-values. Only genes with lower p-values are listed.
fc	optional minimum fold-change required.
lfc	optional minimum log2-fold-change required, equal to log2(fc). fc and lfc are alternative ways to specify a fold-change cutoff and, if both are specified, then fc takes precedence. If specified, then the results from topTable or topTreat will include only genes with (at least one) absolute log-fold-change greater than lfc. This argument is not normally used with topTreat, which handles fold-change thresholds differently via the treat function.

confint	logical, should confidence 95% intervals be output for logFC? Alternatively, can be a numeric value between zero and one specifying the confidence level required.
...	other topTreat arguments are passed to topTable.

Details

These functions summarize the linear model fit object produced by lmFit, lm.series, gls.series or mrlm by selecting the top-ranked genes for any given contrast, or for a set of contrasts. topTable assumes that the linear model fit has already been processed by eBayes. topTreat assumes that the fit has been processed by treat.

If coef has a single value, then the moderated t-statistics and p-values for that coefficient or contrast are used. If coef takes two or more values, the moderated F-statistics for that set of coefficients or contrasts are used. If coef is left NULL, then all the coefficients or contrasts in the fitted model are used, except that any coefficient named (Intercept) will be removed.

The p-values for the coefficient/contrast of interest are adjusted for multiple testing by a call to p.adjust. The "BH" method, which controls the expected false discovery rate (FDR) below the specified value, is the default adjustment method because it is the most likely to be appropriate for microarray studies. Note that the adjusted p-values from this method are bounds on the FDR rather than p-values in the usual sense. Because they relate to FDRs rather than rejection probabilities, they are sometimes called q-values. See help("p.adjust") for more information.

Note that, if there is no good evidence for differential expression in the experiment, it is quite possible for all the adjusted p-values to be large, even for all of them to be equal to one. It is quite possible for all the adjusted p-values to be equal to one if the smallest p-value is no smaller than 1/ngenes where ngenes is the number of genes with non-missing p-values.

The sort.by argument specifies the criterion used to select the top genes. The choices are: "logFC" to sort by the (absolute) coefficient representing the log-fold-change; "A" to sort by average expression level (over all arrays) in descending order; "T" or "t" for absolute t-statistic; "P" or "p" for p-values; or "B" for the lods or B-statistic.

Normally the genes appear in order of selection in the output table. If a different order is wanted, then the resort.by argument may be useful. For example, topTable(fit, sort.by="B", resort.by="logFC") selects the top genes according to log-odds of differential expression and then orders the selected genes by log-ratio in decreasing order. Or topTable(fit, sort.by="logFC", resort.by="logFC") would select the genes by absolute log-fold-change and then sort them from most positive to most negative.

Toptable output for all probes in original (unsorted) order can be obtained by topTable(fit,sort="none",n=Inf). However write.fit or write may be preferable if the intention is to write the results to a file. A related method is as.data.frame(fit) which coerces an MArrayLM object to a data.frame.

By default number probes are listed. Alternatively, by specifying p.value and number=Inf, all genes with adjusted p-values below a specified value can be listed.

The arguments fc and lfc give the ability to filter genes by log-fold change, but see the Note below. This argument is not available for topTreat because treat already handles fold-change thresholding in a more sophisticated way.

Value

A dataframe with a row for the number top genes and the following columns:

genelist	one or more columns of probe annotation, if genelist was included as input
logFC	estimate of the log2-fold-change corresponding to the effect or contrast. There may be several columns of log-fold-changes when multiple coefficients have been tested.
CI.L	left limit of confidence interval for logFC (if confint=TRUE or confint is numeric)
CI.R	right limit of confidence interval for logFC (if confint=TRUE or confint is numeric)
AveExpr	average log2-expression for the probe over all arrays and channels, same as Amean in the MarrayLM object
t	moderated t-statistic (omitted when F-tests have been conducted).
F	moderated F-statistic (omitted when only one coefficient or contrast has been tested).
P.Value	raw p-value
adj.P.Value	adjusted p-value or q-value
B	log-odds that the gene is differentially expressed (omitted for topTreat).

If fit had unique rownames, then the row.names of the above data.frame are the same in sorted order. Otherwise, the row.names of the data.frame indicate the row number in fit. If fit had duplicated row names, then these are preserved in the ID column of the data.frame, or in ID0 if genelist already contained an ID column.

Note

Although topTable enables users to set both p-value and fold-change cutoffs, the use of fold-change cutoffs is not generally recommended. If the fold changes and p-values are not highly correlated, then the use of a fold change cutoff can increase the false discovery rate above the nominal level. Users wanting to use fold change thresholding are usually recommended to use treat and topTreat instead.

In general, the adjusted p-values returned by adjust.method="BH" remain valid as FDR bounds only when the genes remain sorted by p-value. Resorting the table by log-fold-change can increase the false discovery rate above the nominal level for genes at the top of the resorted table.

Author(s)

Gordon Smyth

See Also

An overview of linear model and testing functions is given in 06.LinearModels. See also p.adjust in the stats package.

Examples

```
#  See lmFit examples
```

`tricubeMovingAverage` *Moving Average Smoother With Tricube Weights*

Description

Apply a moving average smoother with tricube distance weights to a numeric vector.

Usage

```
tricubeMovingAverage(x, span=0.5, power=3)
```

Arguments

`x`	numeric vector
`span`	the smoother span. This gives the proportion of `x` values that contribute to each moving average. Larger values give more smoothness. Should be positive but not greater than 1.
`power`	a positive exponent used to compute the tricube weights. `power=3` gives the usual tricube weights. Smaller values give more even weighting. Should be greater than 0.

Details

This function smooths a vector (considered as a time series) using a moving average with tricube weights. Specifically, the function computes running weighted means of `w` consecutive values of `x`, where the window width `w` is equal to `2*h+1` with `h = 2*floor(span*length(x)/2)`. The window width `w` is always odd so that each window has one of the original `x` values at its center. Each weighted mean uses a set of tricube weights so that values near the ends of the window receive less weight.

The smoother returns a vector of the same length as input. At the start and end of the vector, the series is considered to be extended by missing values, and the weighted average is computed only over the observed values. In other words, the window width is reduced to `h+1` at the boundaries with asymmetric weights.

The result of this function is similar to a least squares loess curve of degree zero, with a couple of differences. First, a continuity correction is applied when computing the distance to neighboring points, so that exactly `w` points are included with positive weights in each average. Second, the span halves at the end points so that the smoother is more sensitive to trends at the ends.

The `filter` function in the stats package is called to do the low-level calculations.

This function is used by `barcodeplot` to compute enrichment worms.

Value

Numeric vector of same length as `x` containing smoothed values.

Author(s)

Gordon Smyth

See Also

filter, barcodeplot, loessByCol

Examples

```
x <- rbinom(100,size=1,prob=0.5)
plot(1:100,tricubeMovingAverage(x))
```

---

trigammaInverse *Inverse Trigamma Function*

Description

The inverse of the trigamma function.

Usage

```
trigammaInverse(x)
```

Arguments

x	numeric vector or array

Details

The function uses Newton's method with a clever starting value to ensure monotonic convergence.

Value

Numeric vector or array y satisfying `trigamma(y)==x`.

Note

This function does not accept a data.frame as argument although the base package function `trigamma` does.

Author(s)

Gordon Smyth

See Also

This function is the inverse of trigamma in the base package.

This function is called by fitFDist.

Examples

```
y <- trigammaInverse(5)
trigamma(y)
```

`trimWhiteSpace` *Trim Leading and Trailing White Space*

Description

Trims leading and trailing white space from character strings.

Usage

```
trimWhiteSpace(x)
```

Arguments

`x`	character vector

Value

A character vector of the same length as `x` in which leading and trailing white space has been stripped off each value.

Author(s)

Tim Beissbarth and Gordon Smyth

See Also

An overview of LIMMA functions for reading data is given in 03.ReadingData.

Examples

```
x <- c("a "," b ")
trimWhiteSpace(x)
```

uniquegenelist *Eliminate Duplicate Names from the Gene List*

Description

Eliminate duplicate names from the gene list. The new list is shorter than the full list by a factor of ndups.

Usage

```
uniquegenelist(genelist,ndups=2,spacing=1)
```

Arguments

genelist	vector of gene names
ndups	number of duplicate spots. The number of rows of genelist must be divisible by ndups.
spacing	the spacing between duplicate names in genelist

Value

A vector of length length(genelist)/ndups containing each gene name once only.

Author(s)

Gordon Smyth

See Also

unwrapdups

Examples

```
genelist <- c("A","A","B","B","C","C","D","D")
uniquegenelist(genelist,ndups=2)
genelist <- c("A","B","A","B","C","D","C","D")
uniquegenelist(genelist,ndups=2,spacing=2)
```

unwrapdups *Unwrap Duplicate Spot Values from Rows into Columns*

Description

Reshape a matrix so that a set of consecutive rows becomes a single row in the output.

Usage

```
unwrapdups(M,ndups=2,spacing=1)
```

Arguments

M	a matrix.
ndups	number of duplicate spots. The number of rows of M must be divisible by ndups.
spacing	the spacing between the rows of M corresponding to duplicate spots, spacing=1 for consecutive spots

Details

This function is used on matrices corresponding to a series of microarray experiments. Rows corresponding to duplicate spots are re-arranged so that all values corresponding to a single gene are on the same row. This facilitates fitting models or computing statistics for each gene.

Value

A matrix containing the same values as M but with fewer rows and more columns by a factor of ndups. Each set of ndups rows in M is strung out to a single row so that duplicate values originally in consecutive rows in the same column are in consecutive columns in the output.

Author(s)

Gordon Smyth

Examples

```
M <- matrix(1:12,6,2)
unwrapdups(M,ndups=2)
unwrapdups(M,ndups=3)
unwrapdups(M,ndups=2,spacing=3)
```

venn *Venn Diagrams*

Description

Compute classification counts and draw a Venn diagram.

Usage

```
vennCounts(x, include="both")
vennDiagram(object, include="both", names=NULL, mar=rep(1,4), cex=c(1.5,1,0.7), lwd=1,
            circle.col=NULL, counts.col=NULL, show.include=NULL, ...)
```

Arguments

x	a TestResults matrix. This is numeric matrix of 0's, 1's and -1's indicating significance of a test or membership of a set. Each row corresponds to a gene and each column to a contrast or set. Usually created by decideTests.
object	either a TestResults matrix or a VennCounts object produced by vennCounts.
include	character vector specifying whether all differentially expressed genes should be counted, or whether the counts should be restricted to genes changing in a certain direction. Choices are "both" for all differentially expressed genes, "up" for up-regulated genes only or "down" for down-regulated genes only. If include=c("up","down") then both the up and down counts will be shown. This argument is ignored if object is already a vennCounts object.
names	character vector giving names for the sets or contrasts
mar	numeric vector of length 4 specifying the width of the margins around the plot. This argument is passed to par.
cex	numerical vector of length 3 giving scaling factors for large, medium and small text on the plot.
lwd	numerical value giving the amount by which the circles should be scaled on the plot. See par.
circle.col	vector of colors for the circles. See par for possible values.
counts.col	vector of colors for the counts. Of same length as include. See par for possible values.
show.include	logical value whether the value of include should be printed on the plot. Defaults to FALSE if include is a single value and TRUE otherwise
...	any other arguments are passed to plot

Details

Each column of x corresponds to a contrast or set, and the entries of x indicate membership of each row in each set or alternatively the significance of each row for each contrast. In the latter case, the entries can be negative as well as positive to indicate the direction of change.

vennCounts can collate intersection counts for any number of sets. vennDiagram can plot up to five sets.

Value

vennCounts produces an object of class "VennCounts". This contains only one slot, which is a numerical matrix with 2^ncol{x} rows and ncol(x)+1 columns. Each row corresponds to a particular combination of set memberships. The first ncol{x} columns of output contain 1 or 0 indicating membership or not in each set. The last column called "Counts" gives the number of rows of x corresponding to that combination of memberships.

vennDiagram produces no output but causes a plot to be produced on the current graphical device.

Author(s)

Gordon Smyth, James Wettenhall, Francois Pepin, Steffen Moeller and Yifang Hu

See Also

An overview of linear model functions in limma is given by 06.LinearModels.

Examples

```
Y <- matrix(rnorm(100*6),100,6)
Y[1:10,3:4] <- Y[1:10,3:4]+3
Y[1:20,5:6] <- Y[1:20,5:6]+3
design <- cbind(1,c(0,0,1,1,0,0),c(0,0,0,0,1,1))
fit <- eBayes(lmFit(Y,design))
results <- decideTests(fit)
a <- vennCounts(results)
print(a)
mfrow.old <- par()$mfrow
par(mfrow=c(1,2))
vennDiagram(a)
vennDiagram(results,
    include=c("up", "down"),
    counts.col=c("red", "blue"),
    circle.col = c("red", "blue", "green3"))
par(mfrow=mfrow.old)
```

---

volcanoplot *Volcano Plot*

Description

Creates a volcano plot for a specified coefficient of a linear model.

Usage

```
volcanoplot(fit, coef = 1, style = "p-value",
            highlight = 0, names = fit$genes$ID, hl.col = "blue",
            xlab = "Log2 Fold Change", ylab = NULL, pch=16, cex=0.35, ...)
```

Arguments

fit	an MArrayLM fitted linear model object.
coef	index indicating which coefficient of the linear model is to be plotted.
style	character string indicating which significance statistic to plot on the y-axis. Possibilities are "p-value" or "B-statistic".
highlight	number of top genes to be highlighted by name.
names	character vector of length nrow(fit) giving gene names. Only used if highlight > 0.
hl.col	color for the gene names. Only used if highlight > 0.
xlab	character string giving label for x-axis
ylab	character string giving label for y-axis
pch	vector or list of plotting characters.
cex	numeric vector of plot symbol expansions.
...	any other arguments are passed to plot

Details

A volcano plot displays log fold changes on the x-axis versus a measure of statistical significance on the y-axis. Here the significance measure can be -log(p-value) or the B-statistics, which give the posterior log-odds of differential expression.

The plot is optionally annotated with the names of the most significant genes.

Value

No value is returned but a plot is created on the current graphics device.

Author(s)

Gordon Smyth

See Also

An overview of presentation plots following the fitting of a linear model in LIMMA is given in 06.LinearModels.

Examples

```
#  See lmFit examples
```

voom *Transform RNA-Seq Counts for Linear Modeling With Precision Weights (Deprecated)*

Description

Transform count data to log2 counts-per-million (logCPM), estimate the mean-variance relationship and use it to compute observation-level precision weights. The logCPM and associated precision weights are then ready for linear modeling.

Usage

```
voom(counts, design = NULL,
   lib.size = NULL, offset = NULL, offset.prior = NULL, normalize.method = "none",
    block = NULL, correlation = NULL, weights = NULL,
    span = 0.5, adaptive.span = TRUE, plot = FALSE, save.plot = FALSE)
```

Arguments

counts	a numeric matrix containing raw counts, or an ExpressionSet containing raw counts, or a DGEList object. Counts must be non-negative and NAs are not permitted.
design	design matrix with rows corresponding to samples and columns to coefficients to be estimated. Defaults to model.matrix(~group, data=counts$samples) if counts is a DGEList, otherwise defaults to the unit vector meaning that all samples are treated as replicates.
lib.size	numeric vector containing the library sizes for each sample. Defaults to the columnwise count totals if counts is a matrix or to normLibSizes(counts) if counts is a DGEList.
offset	an edgeR-style offset matrix of same dimensions as counts specifying observation-specific library sizes on the natural log scale. Provides a way to input observation-specific normalization schemes. If not NULL, then the library sizes will be set to exp(offset).
offset.prior	an alternative way to specify the offset matrix. offset.prior specifies observation-specific deviations from the sample-wise library sizes instead of replacing them, as offset does. If provided, then the library sizes will be set to exp(offset), where offset is obtained by adding the normalized library sizes to offset.prior. The rows of offset.prior should sum to zero on average to ensure that the combined offset is reflective of the library sizes.
normalize.method	a microarray-style normalization method to be applied to the logCPM values (if any). Choices are as for the method argument of normalizeBetweenArrays when the data is single-channel. Note that any normalization factors found in counts will still be used to form the log2CPM even if normalize.method="none".
block	vector or factor specifying a blocking variable on the samples. Has length equal to the number of samples (ncol(counts)).

correlation	the intrablock correlation. Normally a single numeric value between -1 and 1, but a vector of genewise correlations is also allowed.
weights	prior weights. Can be a numeric matrix of individual weights of same dimensions as the counts, or a numeric vector of sample weights with length equal to ncol(counts), or a numeric vector of gene weights with length equal to nrow(counts).
span	width of the smoothing window used for the lowess mean-variance trend. Expressed as a proportion between 0 and 1.
adaptive.span	logical. If TRUE, then an optimal value for span will be chosen depending on the number of genes.
plot	logical, should a plot of the mean-variance trend be displayed?
save.plot	logical, should the coordinates and line of the plot be saved in the output?

Details

This was the original function implementing the "voom" method, although we now recommend voomLmFit instead. voomLmFit is a further developed version of voom with extra capabilities and ease of use. voomLmFit automates the computation of sample weights, intrablock correlation and contrasts, and also improves variance estimation for sparse data.

voom is an acronym for "mean-variance modeling at the observational level". The voom function processes sequence count data from technologies such as RNA-seq or ChIP-seq to make it ready for linear modeling in limma. The idea is to estimate the mean-variance relationship in the data, then use this to compute an appropriate precision weight for each observation. Count data always show marked mean-variance relationships. Raw counts show increasing variance with increasing count size, while log-counts typically show a decreasing mean-variance trend. This function estimates the mean-variance trend for log-counts, then assigns a weight to each observation based on its predicted variance. The weights are then used in the linear modeling process to adjust for heteroscedasticity. The mean-variance trend is estimated from gene-level data but is extrapolated back to individual observations to obtain a precision weight (inverse variance) for each observation.

voom performs the following specific calculations. First, the counts are converted to logCPM values, adding 0.5 to all the counts to avoid taking the logarithm of zero. The logCPM calculation uses normalized library sizes if counts is a DGEList or simply the column sums if counts is a matrix. A microarray-style normalization method can also be optionally applied to the matrix of logCPM values. The lmFit function is used to fit row-wise linear models. The lowess function is then used to fit a trend to the square-root residual standard deviations as a function of an average log-count measure. The trend line is then used to predict the variance of each logCPM value as a function of its fitted value on the count scale, and the inverse variances become the estimated precision weights.

The optional arguments block, correlation and weights are passed to lmFit in the above calling sequence, so they influence the row-wise standard deviations to which the mean-variance trend is fitted. The arguments block and correlation have the same meaning as for lmFit. Most users will not need to specify the weights argument but, if it is included, then the output weights are taken to modify the input prior weights in a multiplicative fashion.

For good results, the counts matrix should be filtered to remove rows with very low counts before running voom(). The filterByExpr function in the edgeR package can be used for that purpose.

If counts is a DGEList object from the edgeR package, then voom will use the normalization factors found in the object when computing the logCPM values. In other words, the logCPM values are

computed from the effective library sizes rather than the raw library sizes. If the DGEList object has been scale-normalized in edgeR, then it is usual to leave normalize.method="none" in voom, i.e., the logCPM values should not usually be re-normalized in the voom call.

If the DGEList object contains an offset matrix, then observation-specific effective library sizes will be set to exp(offset). If an offset.prior matrix is found, then offset.prior will be added to the log library sizes to make the offset matrix. offset.prior provides a way to implement normalization observation-specific normalization strategies based on considerations such as transcript length or GC content.

The voom method is similar in purpose to the limma-trend method, which uses eBayes or treat with trend=TRUE. The voom method incorporates the mean-variance trend into the precision weights, whereas limma-trend incorporates the trend into the empirical Bayes moderation. The voom method takes into account the sequencing depths (library sizes) of the individual columns of counts and applies the mean-variance trend on an individual observation basis. limma-trend, on the other hand, assumes that the library sizes are not wildly different and applies the mean-variance trend on a genewise basis. As noted by Law et al (2014), voom should be more powerful than limma-trend if the library sizes are very different but, otherwise, the two methods should give similar results.

If adaptive.span is TRUE, then span is set to chooseLowessSpan(nrow(counts), small.n=50, min.span=0.3, power=1/3).

Value

An EList object with the following components:

E	numeric matrix of normalized expression values on the log2 scale
weights	numeric matrix of inverse variance weights
design	design matrix
lib.size	numeric vector of total normalized library sizes
genes	data-frame of gene annotation extracted from counts
span	if adaptive.span, the chosen value for span is returned
voom.xy	if save.plot, list containing x and y coordinates for points in mean-variance plot
voom.line	if save.plot, list containing coordinates of loess line in the mean-variance plot

Note

What sort of counts can voom accept? voom is designed to accept counts. Usually these will be sequence read counts, but counts of species abundance or other biological quantities might also be appropriate. Estimated counts are also acceptable provided that the column sums are representative of the total library size (total number of reads) for that sample. voom can analyze scaled counts provided that the column sums remain proportional to the total library sizes. voom is designed to take account of sample-specific library sizes and hence voom should not be used to analyze quantities that have been normalized for library size such as RPKM, transcripts per million (TPM) or counts per million (CPM). Such quantities prevent voom from inferring the correct library sizes and hence the correct precision with which each value was measured.

Author(s)

Charity Law and Gordon Smyth

References

Law CW (2013). *Precision weights for gene expression analysis*. PhD Thesis. University of Melbourne, Australia. http://hdl.handle.net/11343/38150.

Law CW, Chen Y, Shi W, Smyth GK (2014). Voom: precision weights unlock linear model analysis tools for RNA-seq read counts. *Genome Biology* 15, R29. doi:10.1186/gb2014152r29. See also the Preprint Version at https://gksmyth.github.io/pubs/VoomPreprint.pdf incorporating some notational corrections.

Law CW, Alhamdoosh M, Su S, Smyth GK, Ritchie ME (2018). RNA-seq analysis is easy as 1-2-3 with limma, Glimma and edgeR. *F1000Research* 5, 1408. doi:10.12688/f1000research.9005.3.

Law CW, Alhamdoosh M, Su S, Dong X, Tian L, Smyth GK, Ritchie ME (2018). RNA-seq analysis is easy as 1-2-3 with limma, Glimma and edgeR. *Bioconductor Workflow Package* https://www.bioconductor.org/packages/RNAseq123/.

See Also

lmFit, contrasts.fit and eBayes are downstream of voom.

voomWithQualityWeights and voomLmFit are further developed versions of voom with extra capabilities. Either can be used as a replacement for voom although voomLmFit is now recommended. voomWithQualityWeights estimates sample weights. voomLmFit estimates both sample weights and intrablock correlation and also improves variance estimation for sparse data.

vooma is analogous to voom but for continuous log-expression data instead of RNA-seq counts.

A summary of limma functions for RNA-seq analysis is given in 11.RNAseq.

Examples

```
## Not run:
keep <- filterByExpr(counts, design)
v <- voom(counts[keep,], design, plot=TRUE)
fit <- lmFit(v, design)
fit <- eBayes(fit, robust=TRUE)
## End(Not run)
```

---

vooma *Convert Variance Trend to Observation Precision Weights for Microarray-style Data*

Description

Estimate the variance trend for microarray data and use it to compute appropriate observational-level weights. The variance trend optionally depends on a second predictor as well as on average log-expression.

Usage

```
vooma(y, design = NULL, block = NULL, correlation,
    predictor = NULL, span = NULL, legacy.span = FALSE, plot = FALSE, save.plot = FALSE)

voomaByGroup(y, group, design = NULL, block = NULL, correlation,
             span = NULL, legacy.span = FALSE, plot = FALSE, col = NULL, lwd = 1,
              pch = 16, cex = 0.3, alpha = 0.5, legend = "topright")
```

Arguments

y	a numeric matrix, EList object, or any object containing log-expression data that can be coerced to a matrix. Rows correspond to genes and columns to samples.
design	design matrix with rows corresponding to samples and columns to coefficients to be estimated. Defaults to the unit vector meaning that samples are treated as replicates.
block	vector or factor specifying a blocking variable on the arrays. Has length equal to the number of arrays.
correlation	intra-block correlation
predictor	precision predictor. Numeric matrix of the same dimensions as y that predicts the precision of each log-expression value. Is used as a second covariate together with the log-intensities to predict the variances and produce the final precision weights.
span	width of the smoothing window, as a proportion of the data set. Defaults to a value that depends on the number of genes (nrow(y)). Equal to 1 if the number of genes is less than or equal to 50, then decreases slowly to 0.3 if the number of genes is very large.
legacy.span	logical. If TRUE, then the original default setting will be used for span, which is slightly smaller than the new default.
plot	logical. If TRUE, a plot of the mean-variance trend is displayed.
save.plot	logical, should the coordinates and line of the plot be saved in the output?
group	categorical vector or factor giving group membership of columns of y.
col	vector of colors for plotting group trends
lwd	line width for plotting group trends
pch	plotting character. Default is integer code 16, which gives a solid circle. If a vector, then should be of length nrow(y).
cex	numeric vector of plot symbol expansions. If a vector, then should be of length equal to number of groups.
alpha	transparency of points, on scale from 0 for fully transparent to 1 for fully opaque.
legend	character string giving position to place legend.

Details

vooma is an acronym for "mean-variance modeling at the observational level for arrays". It is analogous to voom but for continuous log-expression values rather than for sequencing counts.

vooma estimates the mean-variance relationship in the data and uses it to compute appropriate precision weights for each observation. The mean-variance trend is estimated from gene-level data but is extrapolated back to individual observations to obtain a precision weight (inverse variance) for each observation. The weights can then be used by other functions such as lmFit to adjust for heteroscedasticity.

If span=NULL, then an optimal span value is estimated depending on nrow(y). The span is chosen by chooseLowessSpan with n=nrow(y), small.n=50, min.span=0.3 and power=1.3. If legacy.span=TRUE, then the chooseLowessSpan arguments are reset to small.n=10, min.span=0.3 and power=0.5 to match the settings used by vooma in limma version 3.59.1 and earlier.

The variance trend can be modeled using a second optional predictor as well as in terms of log-expression. If predictor is not NULL, then the variance trend is modeled as a function of both the mean log-expression and the predictor using a multiple linear regression with the two predictors. In this case, the predictor is assumed to be some prior predictor of the precision or standard deviation of each log-expression value. Any predictor that is correlated with the precision of each observation should give good results.

voomaByGroup estimates precision weights separately for different groups of samples. In other words, it allows for different mean-variance curves in different groups. voomaByGroup has a quite simple implementation and simply subsets the design matrix for each group. This subsetting is equivalent to interacting the design factors with the groups and might not work well with complex design matrices. It will work fine if the design matrix corresponds to the same groups as defined by the group argument. It can work well for large datasets, for example it has been used by Ravindra et al (2023) to account for TMT groups in proteomics data.

Value

An EList object with the following components:

E	numeric matrix of log-expression values. Equal to y for vooma or log2-counts-per-million for voomaByGroup.
weights	numeric matrix of observation precision weights.
design	numeric matrix of experimental design.
genes	data-frame of gene annotation, only if counts was a DGEList object.
voom.xy	if save.plot, list containing x and y coordinates for points in mean-variance plot
voom.line	if save.plot, list containing coordinates of lowess line in the mean-variance plot

Author(s)

Charity Law, Gordon Smyth and Mengbo Li. Mengbo Li contributed the functionality associated with the predictor argument.

References

Law CW (2013). *Precision weights for gene expression analysis*. PhD Thesis. University of Melbourne, Australia. http://hdl.handle.net/11343/38150

Ravindra KC, Vaidya VS, Wang Z, Federspiel JD, Virgen-Slane R, Everley RA, Grove JI, Stephens C, Ocana MF, Robles-Diaz M, Lucena MI (2023). Tandem mass tag-based quantitative proteomic profiling identifies candidate serum biomarkers of drug-induced liver injury in humans. *Nature Communications* 14(1), 1215.

See Also

voomaLmFit, voom, arrayWeights

Examples

```
group <- gl(2,4)
design <- model.matrix(~group)
y <- matrix(rnorm(500*8),500,8)
u <- matrix(runif(length(y)),500,8)
yu <- y*u
v <- vooma(yu,design,plot=TRUE,predictor=u)
```

---

voomaLmFit *Apply vooma-lmFit Pipeline With Automatic Estimation of Sample Weights and Block Correlation*

Description

Estimate the variance trend, use it to compute observational weights and use the weights to fit a linear model. Includes automatic estimation of sample weights and block correlation. Equivalent to calling vooma(), arrayWeights(), duplicateCorrelation() and lmFit() iteratively.

Usage

```
voomaLmFit(y, design = NULL, prior.weights = NULL, block = NULL,
        sample.weights = FALSE, var.design = NULL, var.group = NULL, prior.n = 10,
           predictor = NULL, span = NULL, legacy.span = FALSE,
           plot = FALSE, save.plot = FALSE, keep.EList = TRUE)
```

Arguments

y	a numeric matrix, EList object, or any object containing log-expression data that can be coerced to a matrix. Rows correspond to genes and columns to samples.
design	design matrix with rows corresponding to samples and columns to coefficients to be estimated. Defaults to the unit vector meaning that samples are treated as replicates.

prior.weights	prior weights. Can be a numeric matrix of individual weights of same dimensions as y, or a numeric vector of sample weights with length equal to ncol(y), or a numeric vector of gene weights with length equal to nrow(y).
block	vector or factor specifying a blocking variable on the arrays. Has length equal to ncol(y).
sample.weights	logical value. If TRUE then empirical sample quality weights will be estimated.
var.design	design matrix for predicting the sample variances. Defaults to the sample-specific model whereby each sample has a different variance.
var.group	vector or factor indicating groups to have different sample weights. This is another way to specify var.design for groupwise sample weights.
prior.n	prior number of genes for squeezing the weights towards equality. Larger values squeeze the sample weights more strongly towards equality.
predictor	precision predictor. Either a column vector of length nrow(y) or a numeric matrix of the same dimensions as y that predicts the precision of each log-expression value. Is used as a second covariate together with the log-intensities to predict the variances and produce the final precision weights.
span	width of the smoothing window, as a proportion of the data set. Defaults to a value between 0.3 and 1 that depends on the number of genes (nrow(y)). Equal to 1 if the number of genes is less than or equal to 50, then decreases slowly to 0.3 if the number of genes is very large.
legacy.span	logical. If TRUE, then the original default setting will be used for span, which is slightly smaller than the new default.
plot	logical. If TRUE, a plot of the mean-variance trend is displayed.
save.plot	logical, should the coordinates and line of the plot be saved in the output?
keep.EList	logical. If TRUE, then the EList object containing log-expression values and observation weights will be saved in the component EList of the output object.

Details

This function is analogous to voomLmFit but for microarray-like data with continuous log-expression values. The function is equivalent to calling vooma() followed by lmFit(), optionally with arrayWeights() and duplicateCorrelation() as well to estimate sample weights and block correlation. The function finishes with lmFit() and returns a fitted model object.

Like vooma, voomaLmFit estimates the mean-variance relationship in the data and uses it to compute appropriate precision weights for each observation. The mean-variance trend is estimated from gene-level data but is extrapolated back to individual observations to obtain a precision weight (inverse variance) for each observation. The weights are then used by lmFit() to adjust for heteroscedasticity.

If span=NULL, then an optimal span value is estimated depending on nrow(y). The span is chosen by chooseLowessSpan with n=nrow(y), small.n=50, min.span=0.3 and power=1/3. If legacy.span=TRUE, then the chooseLowessSpan arguments are reset to small.n=10, min.span=0.3 and power=0.5 to match the settings used by vooma in limma version 3.59.1 and earlier.

If predictor is not NULL, then the variance trend is modeled as a function of both the mean log-expression and the predictor using a multiple linear regression with the two predictors. In this

case, the predictor is assumed to be some prior predictor of the precision or standard deviation of each log-expression value. Any predictor that is correlated with the precision of each observation should give good results. This ability to model the variance trend using two covariates (mean log-expression and the predictor covariate) was described for the first time by Li (2024).

Sample weights will be estimated using arrayWeights if sample.weights = TRUE or if either var.design or var.group are non-NULL. An intra-block correlation will be estimated using duplicateCorrelation if block is non-NULL. In either case, the whole estimation pipeline will be repeated twice to update the sample weights and/or block correlation.

Value

An MArrayLM object containing linear model fits for each row of data. If sample weights are estimated, then the output object will include a targets data.frame component with the sample weights as a column with heading "sample.weights".

If save.plot=TRUE then the output object will include components voom.xy and voom.line. voom.xy contains the x and y coordinates of the points in the vooma variance-trend plot and voom.line contains the estimated trend line.

If keep.EList=TRUE, then the output includes component EList with sub-components Elist$E and EList$weights. If y was an EList object, then the output EList preserves all the components of y and adds the weights.

Author(s)

Mengbo Li and Gordon Smyth

References

Li M (2024). Linear Models and Empirical Bayes Methods for Mass Spectrometry-based Proteomics Data. PhD Thesis, University of Melbourne. http://hdl.handle.net/11343/351600

See Also

vooma, lmFit, voomLmFit.

Examples

```
# Example with a precision predictor
group <- gl(2,4)
design <- model.matrix(~group)
y <- matrix(rnorm(500*8),500,8)
u <- matrix(runif(length(y)),500,8)
yu <- y*u
fit <- voomaLmFit(yu,design,plot=TRUE,predictor=u)

# Reproducing vooma plot from output object
fit <- voomaLmFit(yu,design,predictor=u,save.plot=TRUE)
do.call(plot,fit$voom.xy)
do.call(lines,fit$voom.line)
```

voomLmFit *Voom Variance Modelling with Sample Weights, Block Correlation and Sparse Data*

Description

Transform count data to log2-counts per million (logCPM), estimate voom precision weights and fit limma linear models while allowing for loss of residual degrees of freedom due to exact zeros.

Usage

```
voomLmFit(counts, design = NULL, contrasts = NULL,
 block = NULL, prior.weights = NULL,
    sample.weights = FALSE, var.design = NULL, var.group = NULL, prior.n = 10,
   lib.size = NULL, offset = NULL, offset.prior = NULL, normalize.method = "none",
    span = 0.5, adaptive.span = TRUE, plot = FALSE, save.plot = FALSE,
    keep.EList = TRUE, nthreads = 1L)
```

Arguments

counts	a numeric matrix containing raw counts, or a DGEList object, or a SummarizedExperiment object containing raw counts. Counts must be non-negative. Fractional counts are permitted but NAs are not.
design	design matrix with rows corresponding to samples and columns to coefficients to be estimated. Defaults to the unit vector meaning that samples are treated as replicates.
contrasts	optional numeric matrix specifying coefficient contrasts. Rows correspond to coefficients (columns of design) and each column is a contrast. If specified, then the output object will contain the contrasts instead of the original coefficients. May be a vector if there is only one contrast. NAs are not allowed.
block	vector or factor specifying a blocking variable on the samples. Has length equal to ncol(counts). Samples within each block are assumed to be correlated.
prior.weights	prior weights. Can be a numeric matrix of individual weights of same dimensions as the counts, or a numeric vector of sample weights with length equal to ncol(counts), or a numeric vector of gene weights with length equal to nrow(counts).
sample.weights	logical value, if TRUE then empirical sample quality weights will be estimated.
var.design	optional design matrix for the sample weights. Defaults to the sample-specific model whereby each sample has a distinct variance.
var.group	optional vector or factor indicating groups to have different array weights. This is another way to specify var.design for groupwise sample weights.
prior.n	prior number of genes for squeezing the weights towards equality. Larger values squeeze the sample weights more strongly towards equality.

lib.size	numeric vector containing total library sizes for each sample. Defaults to the normalized (effective) library sizes in counts if counts is a DGEList or to the columnwise count totals if counts is a matrix.
offset	an edgeR-style offset matrix of same dimensions as counts specifying observation-specific library sizes on the natural log scale. Provides a way to input observation-specific normalization schemes. If not NULL, then the library sizes will be set to exp(offset).
offset.prior	an alternative way to specify the offset matrix. offset.prior specifies observation-specific deviations from the sample-wise library sizes instead of replacing them, as offset does. If provided, then the library sizes will be set to exp(offset), where offset is obtained by adding the normalized library sizes to offset.prior. The rows of offset.prior should sum to zero on average to ensure that the combined offset is reflective of the library sizes.
normalize.method	a microarray-style normalization method to be applied to the logCPM values (if any). Choices are as for the method argument of normalizeBetweenArrays when the data is single-channel. Note that any normalization factors found in counts will still be used to form the log2CPM even if normalize.method="none".
span	width of the smoothing window used for the lowess mean-variance trend, expressed as a proportion between 0 and 1. Ignored if adaptive.span=TRUE.
adaptive.span	logical. If TRUE, then an optimal value for span will be chosen depending on the number of genes.
plot	logical, should a plot of the mean-variance trend be displayed?
save.plot	logical, should the coordinates and line of the plot be saved in the output?
keep.EList	logical. If TRUE, then the normalized log2-CPM values and voom weights will be saved in the component EList of the output object.
nthreads	positive integer giving the number of OpenMP threads used by native-code fitting steps. Defaults to 1.

Details

"voom" is an acronym for "mean-variance modeling at the observational level". This function processes count data from technologies such as RNA-seq or ChIP-seq to make it ready for linear modeling in limma. The idea is to estimate the mean-variance relationship in the data, then use this to compute an appropriate precision weight for each observation. Count data always show marked mean-variance relationships. Raw counts show increasing variance with increasing count size, while log-counts typically show a decreasing mean-variance trend. This function estimates the mean-variance trend for log-counts, then assigns a weight to each observation based on its predicted variance. The weights are then used in the linear modeling process to adjust for heteroscedasticity. The mean-variance trend is estimated from gene-level data but is extrapolated back to individual observations to obtain a precision weight (inverse variance) for each observation.

The original voom method was described by Law et al (2014) and was implemented in the voom function. Later, voom was combined with empirical sample quality weights by Liu et al (2015) and implemented in the voomWithQualityWeights function. voomLmFit supersedes those earlier functions with additional features and ease of use (Baldoni et al, 2025). It is particularly designed to give improved variance modeling for genes with large numbers of zero counts. It allows for loss

of residual degrees of freedom due to exact zero fitted values using the method of Lun & Smyth (2017). The loss of residual degrees of freedom occurs when all the counts in a group are zero or when there are blocking factors that can fit zero counts exactly, meaning that not all the counts contribute effectively to variance estimation. The refinement implemented in `voomLmFit` ensures that the residual variances remain approximately unbiased, even for low count genes with lots of zeros, whereas the residual variances could be underestimated by the original `voom` function for such genes.

This function is analogous to calling `voom` followed by `arrayWeights`, `duplicateCorrelation`, `lmFit` and `contrasts.fit`, except for the modified `df.residual` and `sigma` values. This function returns `df.residual` values that are less than or equal to those from `voom` and `lmFit`, and `sigma` values that are greater than or equal to those from `lmFit`. `voomLmFit` is more robust to zero counts than calling `voom` and `lmFit` separately and provides more rigorous error rate control.

`voomLmFit` replaces the original limma functions `voom`, `arrayWeights`, `duplicateCorrelation`, `lmFit` and `contrasts.fit` for the purpose of modeling of count data. Compared to the original `voom` function, `voomLmFit` adds the following functionality:

- Sample weights are estimated if `sample.weights=TRUE` or if `var.design` or `var.group` are non-NULL.
- An intra-block correlation is estimated if `block` is non-NULL.
- `residual.df` will be decreased for genes where all the counts are zero for particular experimental conditions. As a result, different genes may have different residual df, whereas the residual df from `voom` were all equal.
- The output is a `MArrayLM` linear model object rather than an `EList` data object.
- If `contrasts` is non-NULL, then the output object will contain contrasts rather than coefficients, superseding the need for downstream use of `contrasts.fit`. Importantly, the contrast standard errors from `voomLmFit` are exact whereas those from `contrasts.fit` are often approximate.

`voomLmFit` performs the following specific calculations. First, the counts are converted to logCPM values, adding 0.5 to all the counts to avoid taking the logarithm of zero. The logCPM calculation uses normalized library sizes if `counts` is a DGEList or simply the column sums if `counts` is a matrix. A microarray-style normalization method can also be optionally applied to the matrix of logCPM values. The `lmFit` function is used to fit row-wise linear models. The `lowess` function is then used to fit a trend to the square-root residual standard deviations as a function of an average log-count measure. The trend line is then used to predict the variance of each logCPM value as a function of its fitted value on the count scale, and the inverse variances become the estimated precision weights.

The smoothness of the trend line is determined by `span`. If `adaptive.span` is `TRUE`, then an optimal value for `span` is estimated by `chooseLowessSpan(nrow(counts), small.n=50, min.span=0.3, power=1/3)`.

If `sample.weights=TRUE`, then empirical sample quality weights are estimated using the `arrayWeights` function. If `var.group` is specified, then a unique quality weight will be estimated for each sample group rather than for every individual sample. Alternatively, if `var.design` is a numeric vector or design matrix, then the sample weights will be modeled as a log-linear function of that argument. This functionality allows the sample weights to be modeled, for example, as a function of prior numeric quality scores for the samples. The `lmFit` and `arrayWeights` fits will be iterated twice to ensure approximate convergence.

If block is non-NULL, then duplicateCorrelation will be called to estimate the intra-block correlation. The lmFit and duplicateCorrelation fits will be iterated twice to ensure approximate convergence.

The prior.weights argument can be used to specify *a priori* estimates of observational precision, if they exist, in which case the voom weights will modify these prior weights. Most users will not need to use this argument. If it is used, then the prior weights should arise from external annotation rather than being estimated from the data.

For good results, the counts matrix is usually filtered to remove rows with very low counts before running voomLmFit, and the filterByExpr function in the edgeR package can be used for that purpose. However, voomLmFit is relatively robust to low counts and requires much less filtering for reliable results than did the original voom function.

If counts is a DGEList object from the edgeR package, then voomLmFit will use the normalization factors found in the object when computing the logCPM values. In other words, the logCPM values are computed from the effective library sizes rather than the raw library sizes. If the DGEList object contains library normalization factors, then it is usual to leave normalize.method="none" in voomLmFit, i.e., the logCPM values should not usually be re-normalized in the voomLmFit call.

If the DGEList object contains an offset matrix, then observation-specific effective library sizes will be set to exp(offset). If an offset.prior matrix is found, then offset.prior will be added to the log library sizes to make the offset matrix. offset.prior provides a way to implement normalization observation-specific normalization strategies based on considerations such as transcript length or GC content.

The voom method is similar in purpose to the limma-trend method, which uses eBayes or treat with trend=TRUE. The voom method incorporates the mean-variance trend into the precision weights, whereas limma-trend incorporates the trend into the empirical Bayes moderation. The voom method takes into account the sequencing depths (library sizes) of the individual columns of counts and applies the mean-variance trend on an individual observation basis. limma-trend, on the other hand, assumes that the library sizes are not wildly different and applies the mean-variance trend on a genewise basis. As noted by Law et al (2014), voom should be more powerful than limma-trend if the library sizes are very different but, otherwise, the two methods should give similar results.

voomLmFit is usually followed by running eBayes or treat on the fitted model object.

Value

An MArrayLM object containing linear model fits for each row of data. The object includes a targets data.frame component containing sample annotation and library sizes. Columns of targets include lib.size and sample.weight (if sample.weights=TRUE).

If block was set, then the correlation component stores the consensus intra-block correlation.

If save.plot=TRUE then the output object will include components voom.xy and voom.line. voom.xy contains the x and y coordinates of the points in the voom mean-variance plot in the same format as produced by xy.coords and voom.line contains the estimated trend curve.

If keep.EList=TRUE then the output object includes component EList, which is an EList object in the same format as produced by voom containing the voom log2-CPM values and the voom weights.

Author(s)

Original version by Gordon Smyth as a function in the edgeR package. Self-contained limma version by Lizhong Chen, including the contrasts and nthreads arguments and C code backends

for lmFit, arrayWeights and duplicateCorrelation.

References

Baldoni PL, Chen L, Li M, Chen Y, Smyth GK (2025). Dividing out quantification uncertainty enables assessment of differential transcript usage with limma and edgeR. *Nucleic Acids Research* 53(22), gkaf1305. doi:10.1093/nar/gkaf1305.

Law CW, Chen Y, Shi W, Smyth GK (2014). Voom: precision weights unlock linear model analysis tools for RNA-seq read counts. *Genome Biology* 15, R29. doi:10.1186/gb2014152r29. See also the Preprint Version at https://gksmyth.github.io/pubs/VoomPreprint.pdf incorporating some notational corrections.

Lun ATL, Smyth GK (2017). No counts, no variance: allowing for loss of degrees of freedom when assessing biological variability from RNA-seq data. *Statistical Applications in Genetics and Molecular Biology* 16(2), 83-93. doi:10.1515/sagmb20170010.

Liu R, Holik AZ, Su S, Jansz N, Chen K, Leong HS, Blewitt ME, Asselin-Labat ML, Smyth GK, Ritchie ME (2015). Why weight? Modelling sample and observational level variability improves power in RNA-seq analyses. *Nucleic Acids Research* 43, e97. doi:10.1093/nar/gkv412.

See Also

voom, lmFit, contrasts.fit, voomWithQualityWeights, duplicateCorrelation, arrayWeights, MArrayLM-class, chooseLowessSpan.

Examples

```
ngenes <- 200
nsamples <- 6
mu <- rlnorm(ngenes, meanlog = 1, sdlog = 3)
y <- matrix(rnbinom(ngenes*nsamples, size=1/0.05, mu=mu), ngenes, nsamples)
Group <- factor(c(1,1,1,2,2,2))
design <- model.matrix(~Group)
fit <- voomLmFit(y, design, plot=TRUE)
table(fit$df.residual)
```

---

voomWithQualityWeights

Transform RNA-Seq Counts for Linear Modeling With Precision and Sample Weights (Deprecated)

Description

Combine voom observational-level precision weights with sample-specific quality weights in a designed experiment.

Usage

```
voomWithQualityWeights(counts, design = NULL,
     lib.size = NULL, normalize.method = "none",
     plot = FALSE, span = 0.5, adaptive.span = TRUE,
     var.design = NULL, var.group = NULL,
     method = "genebygene", maxiter = 50, tol = 1e-5, trace = FALSE,
     col = NULL, ...)
```

Arguments

counts	a numeric matrix containing raw counts, or an ExpressionSet containing raw counts, or a DGEList object. Counts must be non-negative and NAs are not permitted.
design	design matrix with rows corresponding to samples and columns to coefficients to be estimated. Defaults to model.matrix(~group, data=counts$samples) if counts is a DGEList, otherwise defaults to the unit vector meaning that all samples are treated as replicates.
lib.size	numeric vector containing total library sizes for each sample. If NULL and counts is a DGEList, then the normalized library sizes are taken from counts. Otherwise library sizes are calculated from the columnwise count totals.
normalize.method	the microarray-style normalization method to be applied to the logCPM values. Choices are as for the method argument of normalizeBetweenArrays when the data is single-channel.
plot	logical, should a plot of the mean-variance trend and sample-specific weights be displayed?
span	width of the smoothing window used for the lowess mean-variance trend. Expressed as a proportion between 0 and 1.
adaptive.span	logical. If TRUE, then an optimal value for span will be chosen depending on the number of genes, and the value set by the span argument will be ignored.
var.design	design matrix for the variance model. Defaults to the sample-specific model whereby each sample has a distinct quality weight.
var.group	vector or factor indicating groups to have different quality weights. This is another way to specify var.design for groupwise variance models.
method	character string specifying the method used to estimate the quality weights. Choices are "genebygene" or "reml".
maxiter	maximum number of iterations allowed for quality weight estimation when method = "reml".
tol	convergence tolerance for quality weight estimation when method = "reml".
trace	logical. If TRUE then diagnostic information is output at each iteration of the "reml" algorithm, or at every 1000th iteration of the "genebygene" algorithm.
col	colors to use in the barplot of sample-specific weights if plot=TRUE. If NULL, then bars are plotted in grey.
...	other arguments are passed to voom and hence to lmFit.

Details

This was the original function implementing the "voom" method with sample weights, although we now recommend voomLmFit instead. voomLmFit is a further developed version of voomWithQualityWeights with extra capabilities and ease of use. voomLmFit automates the computation of sample weights, intrablock correlation and contrasts, and also improves variance estimation for sparse data.

Like voom, voomWithQualityWeights is intended to process RNA-seq data prior to linear modeling in limma. It combines observational-level weights from voom with sample-specific weights estimated using the arrayWeights function. The method is described by Liu et al (2015). It consists of a call to voom, followed by arrayWeights, and then another iteration of both functions.

Value

An EList object similar to that from voom, with an extra column, sample.weights, added to the targets data.frame, containing the vector of sample quality factors. The weights component combines the sample weights and the usual voom precision weights.

Author(s)

Matthew Ritchie, Cynthia Liu, Gordon Smyth

References

Liu R, Holik AZ, Su S, Jansz N, Chen K, Leong HS, Blewitt ME, Asselin-Labat ML, Smyth GK, Ritchie ME (2015). Why weight? Modelling sample and observational level variability improves power in RNA-seq analyses. *Nucleic Acids Research* 43(15), e97. doi:10.1093/nar/gkv412.

See Also

voom, arrayWeights, voomLmFit.

A summary of limma functions for RNA-seq analysis is given in 11.RNAseq.

---

weighted.median *Weighted Median*

Description

Compute a weighted median of a numeric vector.

Usage

```
weighted.median(x, w, na.rm = FALSE)
```

Arguments

x	a numeric vector containing the values whose median is to be computed.
w	a vector of weights the same length as x giving the weights to use for each element of x.
na.rm	a logical value indicating whether NA values in x should be stripped before the computation proceeds.

Details

If w is missing then all elements of x are given the same weight.

Missing values in w are not handled.

The weighted median is the median of the discrete distribution with values given by x and probabilities given by w/sum(w).

Value

numeric value giving the weighted median

See Also

median, weighted.mean

Examples

```
## GPA from Siegel 1994
wt <- c(5,  5,  4,  1)/15
x <- c(3.7,3.3,3.5,2.8)
xm <- weighted.median(x,wt)
```

---

weightedLowess *LOWESS Smoother with Prior Weights*

Description

This function generalizes the original LOWESS smoother (locally-weighted regression) to incorporate prior weights while preserving the original algorithm design and efficiency as closely as possible.

Usage

```
weightedLowess(x, y, weights = NULL,
               delta = NULL, npts = 200, span = 0.3, iterations = 4,
               output.style = "loess")
```

Arguments

x	a numeric vector of values for the covariate or x-axis coordinates.
y	a numeric vector of response values or y-axis coordinates, of same length as x.
weights	a numeric vector containing non-negative prior weights, of same length as x. Defaults to a constant vector.
delta	a numeric scalar specifying the maximum distance between successive anchor x-values where a local regression will be computed. Roughly corresponds to diff(range(x))/npts if the x-values are equally spaced. Setting delta=0 forces every distinct x-value to be an anchor point. If NULL then a suitable delta value will be computed from npts.
npts	an integer scalar specifying the approximate number of anchor x-values at which local regressions will be computed. Ignored if delta is not NULL.
span	a numeric scalar between 0 and 1 specifying the width of the smoothing window as a proportion of the total weight.
iterations	an integer scalar specifying the number of iterations. iterations=1 corresponds to local least squares regression without robustifying weights. Each additional iteration incorporates robustifying weights.
output.style	character string indicating whether the output should be in the style of "loess" or of "lowess".

Details

This function extends the LOWESS algorithm of Cleveland (1979, 1981) to handle non-negative prior weights.

The LOWESS method consists of computing a series of local linear regressions, with each local regression restricted to a window of x-values. Smoothness is achieved by using overlapping windows and by gradually down-weighting points in each regression according to their distance from the anchor point of the window (tri-cube weighting).

To conserve running time and memory, locally-weighted regressions are computed at only a limited number of anchor x-values, either npts or the number of distinct x-values, whichever is smaller. Anchor points are defined exactly as in the original LOWESS algorithm. Any x-value within distance delta of an anchor point is considered adjacent to it. The first anchor point is min(x). With the x-values sorted in ascending order, successive anchor points are defined as follows. The next anchor point is the smallest x-value not adjacent to any previous anchor points. The last anchor point is max(x).

For each anchor point, a weighted linear regression is performed for a window of neighboring points. The neighboring points consist of the smallest set of closest neighbors such that the sum of weights is greater than or equal to span times the total weight of all points. Each local regression produces a fitted value for that anchor point. Fitted values for other x-values are then obtained by linear interpolation between anchor points.

For the first iteration, the local linear regressions use weights equal to prior weights times the tri-cube distance weights. Subsequent iterations multiply these weights by robustifying weights. Points with residuals greater than 6 times the median absolute residual are assigned weights of zero and otherwise Tukey's biweight function is applied to the residuals to obtain the robust weights. More iterations produce greater robustness.

In summary, the prior weights are used in two ways. First, the prior weights are used during the span calculations such that the points included in the window for each local regression must account for the specified proportion of the total sum of weights. Second, the weights used for the local regressions are the product of the prior weights, tri-cube local weights and biweight robustifying weights. Hence a point with prior weight equal to an integer n has the same influence as n points with unit weight and the same x and y-values.

See also loessFit, which is essentially a wrapper function for lowess and weightedLowess with added error checking.

Relationship to lowess and loess

The stats package provides two functions lowess and loess. lowess implements the original LOWESS algorithm of Cleveland (1979, 1981) designed for scatterplot smoothing with a single x-variable while loess implements the more complex algorithm by Cleveland et al (1988, 1992) designed to fit multivariate surfaces. The loess algorithm is more general than lowess in a number of ways, notably because it allows prior weights and up to four numeric predictors. On the other hand, loess is necessarily slower and uses more memory than lowess. Furthermore, it has less accurate interpolation than lowess because it uses a cruder algorithm to choose the anchor points whereby anchor points are equi-spaced in terms of numbers of points rather than in terms of x-value spacing. lowess and loess also have different defaults and input parameters. See Smyth (2003) for a detailed discussion.

Another difference between lowess and loess is that lowess returns the x and y coordinates of the fitted curve, with x in ascending order, whereas loess returns fitted values and residuals in the original data order.

The purpose of the current function is to incorporate prior weights but keep the algorithmic advantages of the original lowess code for scatterplot smoothing. The current function therefore generalizes the span and interpolation concepts of lowess differently to loess.

When output.style="loess", weightedLowess outputs results in original order similar to loessFit and loess. When output.style="lowess", weightedLowess outputs results in sorted order the same as lowess.

The span argument corresponds to the f argument of lowess and the span argument of loess. The delta argument is the same as the delta argument of lowess. The npts argument is new and amounts to a more convenient way to specify delta. The iterations argument is the same as the corresponding argument of loess and is equivalent to iter+1 where iter is the lowess argument.

Value

If output.style="loess", then a list with the following components:

fitted	numeric vector of smoothed y-values (in the same order as the input vectors).
residuals	numeric vector or residuals.
weights	numeric vector of robustifying weights used in the most recent iteration.
delta	the delta used, either the input value or the value derived from npts.

If output.style="lowess", then a list with the following components:

x	numeric vector of x-values in ascending order.
y	numeric vector or smoothed y-values.
delta	the delta used, either the input value or the value derived from npts.

Author(s)

C code and R function by Aaron Lun.

References

Cleveland WS (1979). Robust Locally Weighted Regression and Smoothing Scatterplots. *Journal of the American Statistical Association* 74(368), 829-836.

Cleveland WS (1981). LOWESS: A program for smoothing scatterplots by robust locally weighted regression. *The American Statistician* 35(1), 54.

Cleveland WS, Devlin SJ (1988). Locally-weighted regression: an approach to regression analysis by local fitting. *Journal of the American Statistical Association* 83(403), 596-610.

Cleveland WS, Grosse E, Shyu WM (1992). Local regression models. Chapter 8 In: *Statistical Models in S* edited by J.M. Chambers and T.J. Hastie, Chapman & Hall/CRC, Boca Raton.

Smyth GK (2003). lowess vs. loess. Answer on the Bioconductor Support forum https://support.bioconductor.org/p/2323/.

Smyth GK (2020). Difference between LOESS and LOWESS. Answer on the Cross Validated Stack Exchange forum https://stats.stackexchange.com/questions/161069/difference-between-loess-and-lowess.

See Also

lowess, loess, loessFit, tricubeMovingAverage.

Examples

```
y <- rt(100,df=4)
x <- runif(100)
w <- runif(100)
l <- weightedLowess(x, y, w, span=0.7, output.style="lowess")
plot(x, y, cex=w)
lines(l, col = "red")
```

---

write.fit *Write MArrayLM Object to a File*

Description

Write a microarray linear model fit to a file.

Usage

```
write.fit(fit, results = NULL, file, digits = NULL,
          adjust = "none", method = "separate", F.adjust = "none",
          quote = FALSE, sep = "\t", row.names = TRUE, ...)
```

Arguments

fit	object of class MArrayLM containing the results of a linear model fit.
results	object of class TestResults.
file	character string giving path name for the output file.
digits	integer indicating rounding precision for output values. If NULL, then no rounding is done.
adjust	character string specifying multiple-testing adjustment method for the t-statistic P-values, e.g., "BH". See p.adjust for the available options. If NULL or "none" then the P-values are not adjusted.
method	character string, should the P-value adjustment be "global" or "separate" for each contrast. Ignored if adjust = "none".
F.adjust	character string specifying adjustment method for the F-statistic P-values.
quote	logical value. If TRUE, any character or factor columns will be surrounded by double quotes.
sep	the field separator string. Values in the output file will be separated by this string.
row.names	logical value, whether to include row names in the output file.
...	other arguments are passed to write.table.

Details

This function writes a delimited text file containing for each gene (1) the average log2-intensity (AveExpr), (2) the coefficients or contrasts (log2-fold-changes, Coef), (3) moderated t-statistics, (4) t-statistic P-values, (5) F-statistic if available, (6) F-statistic P-values if available, (7) decideTests results if available and (8) gene names and annotation.

The results argument is optional. If supplied, it should be the output from decideTests for the same fit object, which indicates whether each contrast for each gene is considered statistically significant or not (coded 1 or -1 for positive or negative significant differences and 0 for non-significant values).

If fit contains row names and row.names=TRUE, then the row names will be the first column of the output file with a blank column heading. This behavior is analogous to that of write.csv or to write.table with col.names=NA.

Value

No value is produced but a file is written to the current working directory.

Author(s)

Gordon Smyth

See Also

write.table or write.csv in the base library.

An overview of linear model functions in limma is given by 06.LinearModels.

Examples

```
## Not run:
# The following three alternatives are equivalent:
write.fit(fit, file = "temp.csv", sep = ",")
write.csv(fit, file = "temp.csv")
a <- as.data.frame(fit)
write.csv(fit, file = "temp.csv")

## End(Not run)
```

---

wsva *Weighted Surrogate Variable Analysis*

Description

Calculate surrogate variables from the singular vectors of the linear model residual space.

Usage

```
wsva(y, design, n.sv = 1L, weight.by.sd = FALSE, plot = FALSE, ...)
```

Arguments

y	numeric matrix giving log-expression or log-ratio values for a series of microarrays, or any object that can be coerced to a matrix including ExpressionSet, MAList, EList or PLMSet objects. Rows correspond to genes and columns to samples.
design	design matrix
n.sv	number of surrogate variables required.
weight.by.sd	logical, should the surrogate variables be especially tuned to the more variable genes?
plot	logical. If TRUE, plots the proportion of variance explained by each surrogate variable.
...	other arguments can be included that would be suitable for lmFit.

Details

The function constructs surrogate variables that explain a high proportion of the residual variability for many of the genes. The surrogate variables can be included in the design matrix to remove unwanted variation. The surrogate variables are constructed from the singular vectors of a representation of the linear model residual space.

If weight.by.sd=FALSE, then the method is a simplification of the approach by Leek and Storey (2007).

Value

Numeric matrix with ncol(y) rows and n.sv columns containing the surrogate variables.

Author(s)

Gordon Smyth and Yifang Hu

References

Leek JT, Storey JD (2007). Capturing heterogeneity in gene expression studies by surrogate variable analysis. *PLoS Genetics* 3, 1724-1735.

`zscore` *Transform non-normal random deviates to standard normal*

Description

Compute z-score equivalents of non-normal random deviates.

Usage

```
zscore(q, distribution, ...)
zscoreGamma(q, shape, rate = 1, scale = 1/rate)
zscoreHyper(q, m, n, k)
```

Arguments

`q`	numeric vector or matrix giving deviates of a random variable
`distribution`	character name of a probability distribution for which a cumulative distribution function exists
`...`	other arguments specify distributional parameters and are passed to the cumulative distribution function
`shape`	gamma shape parameter (>0)
`rate`	gamma rate parameter (>0)
`scale`	gamma scale parameter (>0)
`m`	as for `qhyper`
`n`	as for `qhyper`
`k`	as for `qhyper`

Details

These functions compute the standard normal deviates which have the same quantiles as the given values in the specified distribution. For example, if `z <- zscoreGamma(x, shape, rate)` then `pnorm(z)` equals `pgamma(x, shape, rate)`.

`zscore` works for any distribution for which a cumulative distribution function (like `pnorm`) exists in R. The argument `distribution` is the name of the cumulative distribution function with the `"p"` removed.

zscoreGamma and zscoreHyper are specific functions for the gamma and hypergeometric distributions respectively.

The transformation to z-scores is done by converting to log tail probabilities, and then using qnorm. For numerical accuracy, the left or right tail is used, depending on which is likely to be smaller.

Value

Numeric vector or matrix of equivalent deviates from the standard normal distribution.

Author(s)

Gordon Smyth

See Also

zscoreT.

zscoreNBinom in the edgeR package.

qnorm in the stats package.

Examples

```
# These are all equivalent
zscore(c(1,2.5), dist="gamma", shape=0.5, scale=2)
zscore(c(1,2.5), dist="chisq", df=1)
zscoreGamma(c(1,2.5), shape=0.5, scale=2)
```

---

zscoreT *Transform t-statistics to standard normal*

Description

Compute z-score equivalents of t-distributed random deviates.

Usage

```
zscoreT(x, df, approx=FALSE, method = "bailey")
tZscore(z, df)
```

Arguments

x	numeric vector or matrix of values from a t-distribution.
df	degrees of freedom (>0) of the t-distribution.
approx	logical. If TRUE then a fast approximation is used otherwise exact z-scores are computed.
method	character string specifying transformation to be used when approx=TRUE, options being "bailey", "hill" or "wallace".
z	numeric vector or matrix of values from the standard normal distribution.

Details

zscoreT transforms t-distributed values to standard normal. Each value is converted to the equivalent quantile of the normal distribution so that if z <- zscoreT(x, df=df) then pnorm(z) equals pt(x, df=df).

tZscore is the inverse of zscoreT and computes t-distribution equivalents of standard normal deviates.

If approx=FALSE, the transformation is done by converting to log tail probabilities using pt or pnorm and then converting back to quantiles using qnorm or qt. For numerical accuracy, the smaller of the two tail probabilities is used for each deviate.

If approx=TRUE, then an approximate closed-form transformation is used to convert t-statistics to z-scores directly without computing tail probabilities. The method argument provides a choice of three transformations. method="bailey" is equation (5) of Bailey (1980) or equation (7) of Brophy (1987). method="hill" is from Hill (1970) as given by equation (5) of Brophy (1987). method="wallace" is from Wallace (1959) as given by equation (2) of Brophy (1987). Bailey's transformation is a modification of Wallace's approximation. The Hill approximation is generally the most accurate for df > 2 but is poor for df < 1. Bailey's approximation is faster than Hill's and gives acceptable two-figure accuracy throughout. Bailey's approximation also works for some extreme values, with very large x or df, for which Hill's approximation fails due to overflow.

Value

Numeric vector or matrix of z-scores or t-distribution deviates.

Note

The default approximation used when approx=TRUE was changed from Hill to Bailey in limma version 3.41.13.

Author(s)

Gordon Smyth

References

Bailey BJR (1980). Accurate normalizing transformations of a Student's t variate. *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 29(3), 304–306.

Hill GW (1970). Algorithm 395: Student's t-distribution. *Communications of the ACM* 13, 617–620.

Brophy AL (1987). Efficient estimation of probabilities in the t distribution. *Behavior Research Methods* 19, 462–466.

Wallace DL (1959). Bounds on normal approximations to Student's and the chi-square distributions. *The Annals of Mathematical Statistics*, 30(4), 1121–1130.

See Also

zscore.

zscoreNBinom in the edgeR package.

Examples

```
zscoreT(4, df=3)
zscoreT(4, df=3, approx=TRUE)
zscoreT(4, df=Inf)
tZscore(2.2, df=3)
```

Index

∗ **Differential transcript usage**
diffSplice, 64
∗ **Differential usage**
diffSplice, 64
∗ **IO**
controlStatus, 55
getLayout, 89
getSpacing, 90
gridr, 100
printorder, 187
readGAL, 208
readHeader, 209
readSpotTypes, 211
readTargets, 212
write.fit, 271
∗ **Model fit**
voomLmFit, 261
∗ **RNA-seq**
diffSplice, 64
∗ **Reading data**
strsplit2, 230
∗ **algebra**
is.fullrank, 109
∗ **array**
as.data.frame, 24
as.matrix, 26
avedups, 30
avereps, 31
dim, 66
dimnames, 67
exprs.MA, 76
head, 100
uniquegenelist, 247
unwrapdups, 248
∗ **background correction**
backgroundCorrect, 32
detectionPValues, 62
kooperberg, 111
ma3x3, 124
nec, 137
propexpr, 190
∗ **character**
makeUnique, 126
protectMetachar, 193
removeExt, 215
trimWhiteSpace, 246
∗ **classes**
as.MAList, 25
EList-class, 75
LargeDataObject-class, 112
MAList-class, 127
MArrayLM-class, 128
PrintLayout, 186
RGList-class, 217
TestResults-class, 235
∗ **data**
as.MAList, 25
LargeDataObject-class, 112
MAList-class, 127
PrintLayout, 186
RGList-class, 217
∗ **differential usage**
plotExonJunc, 160
plotExons, 162
plotSplice, 179
topSplice, 239
∗ **distributions**
qqt, 194
zscore, 274
zscoreT, 275
∗ **distribution**
fitGammaIntercept, 80
tmixture, 236
∗ **documentation**
01.Introduction, 5
02.Classes, 7
03.ReadingData, 8
04.Background, 9

05.Normalization, 10
06.LinearModels, 11
07.SingleChannel, 13
08.Tests, 14
09.Diagnostics, 15
10.GeneSetTests, 16
11.RNAseq, 17
changeLog, 47
limmaUsersGuide, 113
∗ **empirical Bayes**
eBayes, 70
fitFDist, 77
predFCm, 183
squeezeVar, 228
∗ **file**
readGAL, 208
readHeader, 209
readImaGeneHeader, 210
removeExt, 215
∗ **gene annotation**
alias2Symbol, 18
∗ **gene set enrichment analysis**
cumOverlap, 58
∗ **gene set tests**
10.GeneSetTests, 16
camera, 43
geneSetTest, 85
goana, 93
goanaTrend, 98
ids2indices, 104
roast, 218
romer, 223
topGO, 237
topRomer, 238
∗ **hplot**
imageplot, 105
imageplot3by2, 106
modifyWeights, 135
plotExonJunc, 160
plotExons, 162
plotMA, 165
plotMA3by2, 168
plotPrintTipLoess, 174
plotRLDF, 175
plotSplice, 179
printHead, 185
venn, 249
∗ **htest**
auROC, 28
classifyTestsF, 50
contrasts.fit, 53
decideTests, 59
propTrueNull, 191
rankSumTestWithCorrelation, 196
TestResults-class, 235
∗ **illumina beadchips**
detectionPValues, 62
nec, 137
propexpr, 190
read.idat, 199
read.ilmn, 201
read.ilmn.targets, 203
∗ **locally-weighted regression smoothing**
chooseLowessSpan, 48
loessFit, 120
normalizeCyclicLoess, 142
tricubeMovingAverage, 244
weightedLowess, 268
∗ **manip**
cbind, 46
merge, 130
∗ **mathematical functions**
logcosh, 122
trigammaInverse, 245
∗ **methods**
helpMethods, 103
summary, 233
∗ **microarray data file**
read.idat, 199
read.ilmn, 201
read.maimages, 204
readHeader, 209
∗ **models**
anova.MAList-method, 20
arrayWeights, 21
bwss, 41
bwss.matrix, 42
dupcor, 68
fitted.MArrayLM, 82
gls.series, 91
lm.series, 114
lmFit, 115
lmscFit, 119
mergeScans, 132
mrlm, 136
normalizeVSN, 149

normexp.fit, 153
normexp.fit.control, 154
normexp.fit.detection.p, 156
normexp.signal, 158
printtipWeights, 188
residuals.MArrayLM, 216
selectModel, 227
∗ **multivariate**
intraspotCorrelation, 107
normalizeVSN, 149
∗ **normalization**
normalizeBetweenArrays, 139
normalizeCyclicLoess, 142
normalizeForPrintorder, 144
normalizeMedianAbsValues, 146
normalizeQuantiles, 147
normalizeRobustSpline, 148
normalizeWithinArrays, 150
∗ **plots**
heatdiagram, 101
mdplot, 129
plotDensities, 159
plotFB, 163
plotlines, 164
plotMD, 169
plotMDS, 172
plotSA, 177
volcanoplot, 250
∗ **programming**
isNumeric, 110
∗ **reading data**
QualityWeights, 195
read.columns, 198
read.idat, 199
read.ilmn, 201
read.ilmn.targets, 203
read.maimages, 204
sampleInfoFromGEO, 225
∗ **regression**
arrayWeights, 21
contrastAsCoef, 52
dupcor, 68
fitted.MArrayLM, 82
gls.series, 91
lm.series, 114
lmFit, 115
lmscFit, 119
makeContrasts, 125
MArrayLM-class, 128
modelMatrix, 133
mrlm, 136
printtipWeights, 188
residuals.MArrayLM, 216
selectModel, 227
voomWithQualityWeights, 265
∗ **rna-seq**
plotExonJunc, 160
plotExons, 162
plotSplice, 179
topSplice, 239
voom, 252
voomWithQualityWeights, 265
∗ **separate channel analysis**
designI2M, 61
lmscFit, 119
targetsA2C, 234
∗ **univar**
weighted.median, 267
[.EList *(*subsetting*)*, 231
[.EListRaw *(*subsetting*)*, 231
[.MAList *(*subsetting*)*, 231
[.MArrayLM *(*subsetting*)*, 231
[.RGList *(*subsetting*)*, 231
[.TestResults *(*subsetting*)*, 231
01.Introduction, 5, *7*, *9–11*, *13*, *15–17*, *48*
02.Classes, *6*, 7, *7*, *9–11*, *13*, *15–17*, *24–26*, *30–32*, *67*, *68*, *76*, *77*, *89*, *101*, *113*, *128*, *129*, *185*, *186*, *218*, *232*, *233*, *236*
03.ReadingData, *6*, *7*, 8, *9–11*, *13*, *15–17*, *47*, *56*, *63*, *90*, *91*, *131*, *188*, *194*, *196*, *199*, *207*, *209–213*, *216*, *226*, *231*, *246*
04.Background, *6*, *7*, 9, *9–11*, *13*, *15–17*, *34*, *112*, *125*, *139*, *154*, *156*, *157*, *159*
05.Normalization, *6*, *7*, *9*, 10, *10*, *11*, *13*, *15–17*, *122*, *135*, *139*, *142*, *143*, *145–147*, *149*, *150*, *152*, *215*
06.LinearModels, *6*, *7*, *9*, *10*, 11, *11*, *13–17*, *22–24*, *27*, *39*, *52*, *54*, *70*, *74*, *93*, *115*, *117*, *126*, *134*, *137*, *165*, *189*, *228*, *230*, *243*, *250*, *251*, *272*
07.SingleChannel, *6*, *7*, *9–11*, 13, *13–17*, *62*, *108*, *120*, *235*
08.Tests, *6*, *7*, *9–13*, 14, *15–17*, *51*, *61*, *193*, *197*, *236*

09.Diagnostics, *6, 7*, *9–11*, *13*, 15, *15–17*, *20*, *57*, *106, 107*, *130*, *160*, *164*, *167*, *169*, *171*, *174, 175*, *178*, *182*
10.GeneSetTests, *6, 7*, *9–11*, *13*, *15*, 16, *16, 17*, *19*, *37*, *45*, *88*, *97*, *99*, *104*, *222*, *225*, *238, 239*
11.RNAseq, *6, 7*, *9–11*, *13*, *15, 16*, 17, *17*, *66*, *163*, *180*, *240*, *255*, *267*

alias2Symbol, *14*, *16*, 18, *94, 95*
alias2SymbolTable, *14*, *16*
alias2SymbolTable *(*alias2Symbol*)*, 18
alias2SymbolUsingNCBI, *14*, *16*
alias2SymbolUsingNCBI *(*alias2Symbol*)*, 18
anova, *15*, *20*, *42*
anova.MAList, *42*
anova.MAList *(*anova.MAList-method*)*, 20
anova.MAList-method, 20
array2channel *(*targetsA2C*)*, 234
arrayWeights, *15*, 21, *23*, *258*, *265*, *267*
arrayWeightsQuick, *22*, 23
as.data.frame, *7*, 24, *24*
as.MAList, *8*, 25
as.matrix, 26, *26*
asMatrixWeights, 27
auROC, *14*, 28
avearrays, 29, *32*
avedups, 30, *32*
avereps, *30*, 31, *31*

backgroundCorrect, *9*, 32, *151*, *159*
backgroundCorrect.matrix, *9*
barcodeplot, *14*, *16*, 34, *88*, *244, 245*
beadCountWeights, 38
bg.parameters, *154*
blockDiag, 40
bwss, *15*, 41, *42*
bwss.matrix, *15*, *20*, *41*, 42

camera, *16*, *37*, 43, *86*, *88*, *104*, *219*, *225*
cameraPR, *16*, *86–88*
cameraPR *(*camera*)*, 43
cbind, *9*, 46, *46, 47*, *131*
changeLog, *6*, 47
chooseLowessSpan, 48, *265*
classifyTestsF, *14*, 50, *60*, *129*
coerce,RGList,exprSet2-method *(*RGList-class*)*, 217
combined, *7*, *76*, *127*, *217*
contrastAsCoef, 52
contrasts.fit, *12*, *44*, 53, *85*, *92*, *114*, *116*, *125*, *184*, *219*, *255*, *265*
controlStatus, *9*, 55, *181*
convest *(*propTrueNull*)*, 191
coolmap, 56
cumOverlap, 58

decideTests, *14*, *50*, 59, *102*, *129*, *235*, *249*
density, *159, 160*
designI2A *(*designI2M*)*, 61
designI2M, 61
detectionPValues, *8*, 62, *201*
diffSplice, *17*, 64, *161*, *180*, *240*
dim, 66, *67*, *76*, *127*, *129*, *217*
dimnames, 67, *67, 68*
dimnames<-.EList *(*dimnames*)*, 67
dimnames<-.EListRaw *(*dimnames*)*, 67
dimnames<-.MAList *(*dimnames*)*, 67
dimnames<-.RGList *(*dimnames*)*, 67
dist, *57*
dupcor, 68
duplicateCorrelation, *11*, *93*, *116*, *265*
duplicateCorrelation *(*dupcor*)*, 68

eBayes, *12*, *44*, *49*, 70, *79*, *85*, *128, 129*, *163*, *178*, *184*, *219, 220*, *230*, *237*, *242*, *254, 255*, *264*
EList, *7*, *47*, *131*, *140*, *254*, *267*
EList-class, 75
EListRaw, *7*, *32*, *47*, *131*, *140*, *206*
EListRaw-class *(*EList-class*)*, 75
estimate.m0 *(*propTrueNull*)*, 191
ExpressionSet, *76*
exprs, *26*
exprs.MA, 76
Extract, *232*

filter, *245*
fitFDist, *12*, *74*, 77, *230*, *245*
fitFDistRobustly, *12*, *49*, *230*
fitFDistRobustly *(*fitFDist*)*, 77
fitFDistUnequalDF1, *12*, *73*, *230*
fitFDistUnequalDF1 *(*fitFDist*)*, 77
fitGammaIntercept, 80
fitmixture, 81
fitted, *83*, *129*
fitted.MArrayLM, 82
fry, *16*, *45*

fry (roast), 218

genas, *12*, *14*, *80*, 83
geneSetTest, *14*, *16*, *37*, *45*, 85
getDupSpacing (getLayout), 89
getEAWP, *11*, *43*, *45*, *68*, 88, *115*, *117*, *213*
getGeneKEGGLinks (goana), 93
getKEGGPathwayNames (goana), 93
getLayout, *8*, 89, *186*
getLayout2 (getLayout), 89
getSpacing, *8*, 90
gls.series, *11*, 91
goana, *16*, 93, *99*, *237, 238*
goanaTrend, *96, 97*, 98
gridc, *8*, *16*
gridc (gridr), 100
gridr, *8*, *16*, 100

hclust, *57*
head, 100, *101*
heatDiagram, *14*, *236*
heatDiagram (heatdiagram), 101
heatdiagram, *14*, 101
heatmap.2, *57*
helpMethods, 103

ids2indices, *16*, *44, 45*, 104, *220*, *223*, *225*
image, *103*, *106*
imageplot, *15*, 105
imageplot3by2, *15*, 106
interGeneCorrelation (camera), 43
intraspotCorrelation, *13*, 107, *119*
is.fullrank, 109
is.numeric, *110*
isNumeric, 110

kegga, *16*, *99*, *237, 238*
kegga (goana), 93
kooperberg, *9*, *34*, 111

labels.TestResults (TestResults-class), 235
LargeDataObject, *7*, *76*, *127*, *217*
LargeDataObject-class, 112
legend, *159*, *181*
levels.TestResults (TestResults-class), 235
limma (01.Introduction), 5
LIMMA User's Guide, *10*, *160*
limma-package (01.Introduction), 5
limmaUsersGuide, *6*, 113
lm.fit, *115*, *120*
lm.series, *11*, 114
lmFit, *11*, *44*, *53*, *85*, *92*, *109*, *114*, 115, *128*, *136*, *163*, *184*, *214*, *219*, *253*, *255*, *260*, *265*
lmscFit, *13*, *116*, 119
locfit.raw, *122*
loess, *49*, *122*, *271*
loessByCol, *245*
loessFit, *10*, *49*, 120, *151*, *270, 271*
logcosh, 122, *124*
logsumexp, 123, *123*
lowess, *49*, *122*, *271*

MA.RG, *10*, *127*
MA.RG (normalizeWithinArrays), 150
ma3x3, 124
ma3x3.matrix, *9*
ma3x3.spottedarray, *9*
make.names, *125*
make.unique, *127*
makeContrasts, *12*, 125
makeUnique, *9*, 126
MAList, *7*, *25*, *47*, *108*, *119*, *131*, *140, 141*, *150*, *152*
MAList-class, 127
MArrayLM, *7*, *11*, *53, 54*, *117*, *119*
MArrayLM-class, 128
Math, *110*
matplot, *160*
mdplot, *15*, 129, *166*, *170, 171*, *182*
MDS-class (plotMDS), 172
mean, *108*
median, *268*
merge, *9*, 130, *131*
merge.RGList, *127*
merged, *217*
mergeScans, 132
mergeScansRG (mergeScans), 132
model.matrix, *12*, *52*, *62*, *134*
modelMatrix, *12*, *40*, 133
modifyWeights, *10*, *27*, 135
mrlm, *11*, 136
mroast, *16*, *104*
mroast (roast), 218

ncol, *76*, *127*, *129*, *217*

nec, 137, *156, 157*
neqc, *9, 10*, *34*, *39*, *63*, *139*, *142*, *201, 202*
neqc *(*nec*)*, 137
news, *48*
nlminb, *153*
nonEstimable *(*is.fullrank*)*, 109
normalize.loess, *143*
normalizeBetweenArrays, *10*, *75*, *127*, 139, *147*, *152*, *217*
normalizeBetweenSamples *(*normalizeBetweenArrays*)*, 139
normalizeCyclicLoess, *10*, *140*, 142
normalizeForPrintorder, *10*, *127*, 144, *188*, *217*
normalizeMedianAbsValues, *10*, 146
normalizeMedianValues, *10*
normalizeMedianValues *(*normalizeMedianAbsValues*)*, 146
normalizeQuantiles, *10*, 147
normalizeRobustSpline, *10*, 148, *152*
normalizeVSN, *10*, *142*, 149
normalizeWithinArrays, *10*, *33*, *127*, *149*, 150, *217*
normexp.fit, *9*, *33*, *139*, 153, *156–159*
normexp.fit.control, *9*, *138, 139*, 154, *154*, *156, 157*, *202*
normexp.fit.detection.p, *138, 139*, *155*, 156, *156*
normexp.signal, *9*, *138*, *154*, *156, 157*, 158
nrow, *76*, *127*, *129*, *217*

openPDF, *113*
openVignette, *113*

p.adjust, *60*, *219*, *241–243*, *272*
par, *106*
plotDensities, *15, 16*, 159
plotExonJunc, 160
plotExons, *17*, 162
plotFB, *15*, 163
plotlines, *12*, 164
plotMA, *15*, *127*, 165, *169*, *182*
plotMA3by2, *15*, 168
plotMD, *15*, *130*, *166*, 169, *182*
plotMDS, *16*, 172
plotPrintorder, *15*
plotPrintorder *(*normalizeForPrintorder*)*, 144
plotPrintTipLoess, *15*, *127*, 174
plotRLDF, 175
plotSA, *16*, *72*, *74*, 177
plotSplice, *17*, *66*, *163*, 179, *240*
plotWithHighlights, *15*, *129, 130*, *166, 167*, *170, 171*, 180
points, *164*, *172*, *175*, *178*, *181*
poolVar, 182
predFCm, 183
printHead, *7*, 185
PrintLayout, *127*, 186, *207*, *217*
PrintLayout-class *(*PrintLayout*)*, 186
printorder, *8*, *145*, 187
printtipWeights, 188
printtipWeightsSimple *(*printtipWeights*)*, 188
propexpr, 190, *201, 202*
propTrueNull, *14*, *184*, 191
protectMetachar, 193

qhyper, *274*
qnorm, *275*
qqf *(*qqt*)*, 194
qqnorm, *195*
qqt, 194
QualityWeights, *8*, 195, *206*

rankSumTestWithCorrelation, *45*, 196
rbind, *9*
rbind.EList *(*cbind*)*, 46
rbind.EListRaw *(*cbind*)*, 46
rbind.MAList *(*cbind*)*, 46
rbind.RGList *(*cbind*)*, 46
read.columns, *8*, 198, *202*, *207*
read.idat, *8*, *39*, *63*, 199
read.ilmn, *8*, *39*, *138*, *191*, *200*, 201, *201*, *203*, *205*, *207*
read.ilmn.targets, *202*, 203
read.imagene, *8*, *211*
read.imagene *(*read.maimages*)*, 204
read.maimages, *8*, *75*, *199*, 204, *210*, *217*
read.table, *199*, *207*, *212*
readGAL, *8*, *55*, 208
readGenericHeader, *8*
readGenericHeader *(*readHeader*)*, 209
readGPRHeader, *8*
readGPRHeader *(*readHeader*)*, 209
readHeader, 209
readImaGeneHeader, *8*, 210
readSMDHeader *(*readHeader*)*, 209

readSpotTypes, *9*, 211
readTargets, *8*, *202, 203*, 212, *234*
remlscore, *108*
removeBatchEffect, *10*, 213
removeExt, *8*, 215
residuals, *129*, *217*
residuals.MArrayLM, *83*, 216
RG.MA, *16*
RG.MA *(*normalizeWithinArrays*)*, 150
RGList, *7*, *32*, *47*, *131*, *140*, *149*, *206*
RGList-class, 217
rlm, *137*
roast, *14*, *16*, *37*, *44, 45*, *86*, *88*, 218, *225*
Roast-class *(*roast*)*, 218
roast.default *(*roast*)*, 218
romer, *14*, *16*, *37*, *45*, *104*, *219*, 223, *238, 239*
rowsum, *32*

sampleInfoFromGEO, 225
selectModel, *12*, *14*, 227
show, *76*, *127*, *217*, *236*
show,LargeDataObject-method *(*LargeDataObject-class*)*, 112
show,MDS-method *(*plotMDS*)*, 172
show,Roast-method *(*roast*)*, 218
show,TestResults-method *(*TestResults-class*)*, 235
showMethods, *103*
spotc, *8*
spotc *(*gridr*)*, 100
spotr, *8*
spotr *(*gridr*)*, 100
squeezeVar, *12*, *49*, *72*, *74*, *79*, *227*, 228
strsplit, *231*
strsplit2, *9*, 230
subsetListOfArrays *(*subsetting*)*, 231
subsetted, *7*, *76*, *127*, *217*
subsetting, *109*, 231
summary, 233, *233*
summary.TestResults *(*TestResults-class*)*, 235
Sweave, *113*
Sys.getenv, *113*
Sys.putenv, *113*

tail.EList *(*head*)*, 100
tail.EListRaw *(*head*)*, 100
tail.MAList *(*head*)*, 100
tail.MArrayLM *(*head*)*, 100
tail.RGList *(*head*)*, 100
tail.TestResults *(*head*)*, 100
targetsA2C, *13*, 234
TestResults, *7*, *14*, *51*, *61*
TestResults-class, 235
text, *173*
tmixture, 236
tmixture.matrix, *12*, *74*
tmixture.vector, *12*
topGO, *16*, *97*, 237
topKEGG, *16*, *97*
topKEGG *(*topGO*)*, 237
topRomer, *14*, *16*, *225*, 238
topSplice, *17*, *66*, *161*, *180*, 239
topTable, *12*, 241
topTreat, *12*, *72*
topTreat *(*topTable*)*, 241
treat, *12*, *79*, *178*, *242*, *254*, *264*
treat *(*eBayes*)*, 70
tricubeMovingAverage, *37*, *96*, *99*, 244, *271*
trigamma, *245*
trigammaInverse, *79*, 245
trimWhiteSpace, 246
tZscore *(*zscoreT*)*, 275

unique.TestResults *(*TestResults-class*)*, 235
uniquegenelist, *9*, 247
uniqueTargets *(*modelMatrix*)*, 133
unwrapdups, *11*, *247*, 248

venn, 249
vennCounts, *14*, *236*
vennCounts *(*venn*)*, 249
vennDiagram, *14*, *236*
vennDiagram *(*venn*)*, 249
vignette, *113*
volcanoplot, *12*, 250
voom, *17*, *72*, *75*, 252, *258*, *262*, *265*, *267*
vooma, *49*, 255, *255*, *260*
voomaByGroup *(*vooma*)*, 255
voomaLmFit, 258, *258*
voomLmFit, *253*, *255*, *260*, 261, *267*
voomWithQualityWeights, *17*, *22*, *255*, *262*, 265, *265*
vsnMatrix, *150*

weighted.mean, *268*
weighted.median, 267

weightedLowess, *49*, *121*, *122*, 268, *270*
wilcox.test, *88*, *197*
wilcoxGST, *14*, *16*, *225*
wilcoxGST *(*geneSetTest*)*, 85
write, *242*
write.csv, *272*
write.fit, *12*, *14*, *236*, *242*, 271
write.table, *272*
wsva, 273
wtarea *(*QualityWeights*)*, 195
wtflags *(*QualityWeights*)*, 195
wtIgnore.Filter *(*QualityWeights*)*, 195

xy.coords, *264*

zscore, 274, *276*
zscoreGamma *(*zscore*)*, 274
zscoreHyper *(*zscore*)*, 274
zscoreT, *16*, *221*, 275, *275*