maanova

March 24, 2012

`PairContrast` *Pairwise comparison matrix*

Description

It returns all possible pairwise comparison.

Usage

```
PairContrast(n)
```

Arguments

`n`	Number of levels in test term.

Value

`res`	All pairwise comparison matrix.

Author(s)

Hyuna Yang

Examples

```
# load in abf1 data
data(abf1)
## Not run:
fit.full.mix <- fitmaanova(abf1, formula = ~Strain+Sample,
   random = ~Sample)
ftest.all = matest(abf1, fit.full.mix, test.method=c(1,1),
    shuffle.method="sample", term="Strain", n.perm= 100)
C = PairContrast(3)
ftest.pair = matest(abf1, fit.full.mix, Contrast = C,
   term="Strain", n.perm=100)
## End(Not run)
```

Rmaanova.version *Display the current version of the package*

Description

This is the function to display the current verion number of R/maanova pakcage.

Usage

```
Rmaanova.version()
```

Author(s)

Hao Wu

Examples

```
Rmaanova.version()
```

abf1 *Data for a 18-array Affymetrix experiment*

Description

This is the data set for a 18-array affymetrix experiment. There are three mouse strains, AJ, B6 and their F1 offspring. Three biological replicates each and two technical replicates for each individual.

Usage

```
data(abf1)
```

Format

An object of class `madata`.

Examples

```
data(abf1)
```

adjPval *Generate FDR adjusted P values for F test result.*

Description

This function takes a result object from matest and calculate the FDR adjusted P values. The new P values will be appended to the input object as additional fields. It has four options; "stepup" (Hochberg and Benjamini, 1990), "adaptive" (Benjamini and Hochberg, 2000), "stepdown" (Westfall and Young, 1993) and "jsFDR" (Storey, 2002). "jsFDR" option uses 'qvalue' package by John Storey and user suppose to install 'qvalue' package before using this option. There is no default option, thus you need to specify one option.

Usage

```
adjPval(matestobj, method=c("stepup","adaptive", "stepdown", "jsFDR"))
```

Arguments

matestobj	An object of class matest, which is the result from matest.
method	The method for FDR control.

Value

An object of class matest with the following fields added for each F test:

adjPtab	FDR adjusted tabulated P values.
adjPvalperm	FDR adjusted permutation P values.

Author(s)

Hao Wu

Examples

```
data(abf1)
## Not run:
fit.full.simple = fitmaanova(abf1,formula = ~Strain)
# F-test strain effect
ftest.all = matest(abf1, fit.full.simple, term="Strain",n.perm= 1000)
# make FDR adjusted P values
ftest.all = adjPval(ftest.all, 'jsFDR')
# there will be new fields in test.strain.fix after this

## End(Not run)
```

`arrayview` *View the layout of input data*

Description

This function reconstructs the input data according to the Microarray grid location structure and plots the data according to the user specified color map.

By default, it will plot the log ratios for 2-dye array and raw intensity for 1-dye array. It does not work for N-dye (N>2) array at this time.

Note that if user collapsed the replicates by using 'avgrep' in `read.madata`, and then arrayview will not be available.

Usage

```
arrayview(object, ratio, array, colormap, onScreen=TRUE, ...)
```

Arguments

`object`	An object of class `madata`.
`ratio`	The data to be plotted. The length of it must be equal to the length of the grid locations, .e.g, madata$row and madata$col. If ratio is a vector, there will be one plot. If ratio is a matrix, there will be one plot for each column. If ratio is not provided, `link[maanova]{make.ratio}` will be called to calculate the ratios from the original data.
`array`	A list of arrays to be plotted. This variable is only valid when `ratio` is not provided. Whenever `ratio` is provided, all columns in ratio will be plotted.
`colormap`	User specified color map. See `colors` for more detail.
`...`	Other parameters to be passed to `image`.
`onScreen`	A logical value to represent whether to display the plots on screen or not. If TRUE, x11() (in Unix/Windows) or macintosh (in Mac) will be called inside the function. Otherwise, it will plot the figure on the current device. Default is TRUE.

Author(s)

Hao Wu

Examples

```
## Not run:
data(kidney)
# arrayview data on screen
arrayview(kidney.raw, array=1)
graphics.off()
# arrayview raw data array 1 and 3 and output to postscript file
postscript(file="kidneyArrayview.ps")
arrayview(kidney.raw, array=c(1,3), onScreen=FALSE)

## End(Not run)
```

consensus *Build consensus tree out of bootstrap cluster result*

Description

This is the function to build the consensus tree from the bootstrap clustering analysis. If the clustering algorithm is hierarchical clustering, the majority rule consensus tree will be built based on the given significance level. If the clustering algorithm is K-means, a consensus K-means group will be built.

Usage

```
consensus(macluster, level = 0.8, draw=TRUE)
```

Arguments

macluster	An object of class macluster, which is the output of macluster.
level	The significance level for the consensus tree. This is a numeric number between 0.5 and 1.
draw	A logical value to indicate whether to draw the consensus tree on screen or not.

Value

An object of class consensus.hc or consensus.kmean according to the clustering method.

Author(s)

Hao Wu

See Also

macluster

Examples

```
# load data
data(abf1)
## Not run:
# fit the anova model
fit.fix = fitmaanova(abf1,formula = ~Strain)
# test Strain effect
test.fix = matest(abf1, fit.fix, term="Strain",n.perm= 1000)
# pick significant genes - pick the genes selected by Fs test
idx <- volcano(test.fix)$idx.Fs
# do k-means cluster on genes
gene.cluster <- macluster(fit.fix, term="Strain", idx, what="gene",
   method="kmean", kmean.ngroups=5, n.perm=100)
# get the consensus group
genegroup = consensus(gene.cluster, 0.5)
# get the gene names belonging to each group
genegroupname = genegroup$groupname

# HC cluster on samples
```

```
sample.cluster <- macluster(fit.fix, term="Strain", idx, what="sample",method="hc")
# get the consensus group
consensus(sample.cluster, 0.5)

## End(Not run)
```

`dyeswapfilter` *Gene filter for dye-swap experiment*

Description

This function is used to flag the questionable spot in any kind of dye-swap experiment.

This function only works for 2-dye arrays.

Usage

```
dyeswapfilter(dataobj, r=4)
```

Arguments

`dataobj`	An object of class `madata`.
`r`	A cut-off value for bad spot. The genes with log-ratio difference larger than r times standard deviation will be flagged.

Details

For each pair of dye-swap, the difference in log ratios (d) are computed. Then compute the IQR (interquartile range) of d and convert that to Standard Deviation by SD = IQR/1.35. Any gene with d larger than r times SD will be flagged.

Note that I assume in the input data object, the adjacent arrays is a dye-swap pair.

Value

An object of class `rawdata` or `madata` with the `flag` field created or updated.

Author(s)

Hao Wu

Examples

```
## Not run:
data(kidney)
# riplot before filtering
riplot(kidney.raw, array=1)
# filter the gene
rawdata <- dyeswapfilter(kidney.raw)
# riplot again - some genes are highlighted
riplot(rawdata, array=1)
## End(Not run)
```

`fill.missing` *Fill in missing data*

Description

This is the function to do missing data imputation.

Usage

```
fill.missing(data, method="knn", k=20, dist.method="euclidean")
```

Arguments

`data`	An object of class `madata`, which should be the result from `read.madata`.
`method`	The method to do missing data imputation. Currently only "knn" (K nearest neighbour) is implemented.
`k`	Number of neighbours used in imputation. Default is 20.
`dist.method`	The distance measure to be used. See `dist` for detail.

Details

This function will take an object of class `madata` and fill in the missing data. Currently only KNN (K nearest neighbour) algorithm is implemented. The memory usage is quadratic in the number of genes.

Value

An object of class `madata` with missing data filled in.

Author(s)

Hao Wu

References

O.Troyanskaya, M. Cantor, G. Sherlock, P. Brown, T. Hastie, R. Tibshirani, D. Botstein, & R. B. Altman. Missing Value estimation methods for DNA microarrays. Bioinformatics 17(6):520-525, 2001.

Examples

```
data(abf1)
# randomly generate some missing data
rawdata <- abf1
ndata <- length(abf1$data)
pct.missing <- 0.05 # 5% missing
idx.missing <- sample(ndata, floor(ndata*pct.missing))
rawdata$data[idx.missing] <- NA
rawdata <- fill.missing(rawdata)
# plot impute data versus original data
plot(rawdata$data[idx.missing], abf1$data[idx.missing])
abline(0,1)
```

`fitmaanova` *Fit ANOVA model for Micro Array experiment*

Description

This is the function to fit the ANOVA model for Microarray experiment. Given the data and formula, this function fits the regression model for each gene and calculates the ANOVA estimates, variance components for random terms, fitted values, etc. For a mixed effect models, the output estimates will be BLUE and BLUP.

All terms used in the formula should be corresponding to the factor names in designfile except "Spot" and "Label". "Spot" represents the spotting effect and "Label" represents the labeling effects. They are from the within slide technical replicates. If there is no replicated spots, These two terms cannot be fitted. Also these two terms cannot be fitted for one-dye system (e.g., Affymetrix arrays). (Note that Dye effect should not be fitted in one-dye system).

A typical formula will be like "~Array+Dye+Sample", which means you want to fit Array, Dye and Sample effect in the ANOVA model. In this case, you need to have Array, Dye and Sample columns in your input design file. Make sure you have enough degree of freedom when making a model. Also you need to be careful about confounding problem.

If you have multiple factors in your experiment, you can specify the main and interaction effect in the formula. At this time, only two-way interactions are allowed.

When you have random or covariate effect they should be specified in the 'random' and 'covariate', and also in the formula.

For most mixed effect models, Array should be treated as random factor. Sample should be treated as random if you have biological replicates. Note that the reference sample (0's in Sample) will always be treated as fixed even if you specify Sample as random.

Note that the calculation could be very slow for mixed effect models. The computational time depends on the number of genes, number of arrays and the size of the random variables (dimension of Z matrix).

Array specific covariate should be included in the design matrix, and gene specific covariate should be read by 'covM' in `read.madata()`, and need to be specified in covariate term.

Usage

```
fitmaanova(madata, formula, random= ~1, covariate = ~1, mamodel,
           inits20,method=c("REML","ML","MINQE-I","MINQE-UI", "noest"),
           verbose=TRUE, subCol=FALSE)
```

Arguments

`madata`	An object of class `madata`.
`formula`	The ANOVA model formula.
`random`	The formula for random terms. ~1 means only the residual is random (fixed model). Note that all random terms should be in the ANOVA model formula.
`covariate`	The formula for covariates. ~1 means no covariates. The array specific covariates should be numeric values in the design matrix, and the gene specific covariates should be read by `covM` in `read.madata`.
`mamodel`	Inside arguments to save the calculation time.

inits20	The initial value for variance components. This should be a matrix with number of rows equals to the number of genes and number of columns equals to the number of random terms in the model. Good initial values will greatly speed up the calculation. If it is not given, it will be calculated based on the corresponding fixed model.
method	The method used to solve the Mixed Model Equation. Available options includes: "ML" for maximum liklihood; "REML" for restricted maximum liklihood; "MINQE-I" and "MINQE-UI" are for minimum norm and "noest" for no estimate for variance component (use the initial value). Both "ML" and "REML" use method of scoring algorithm to solve MME iteratively. "noest" skips the iteration and will be significantly faster (but accurate). Default method is "REML". For details about fitting mixed effects models, read the "Fitting mixed Effects model" section.
verbose	A logical value to indicate whether to display some message for calculation progress.
subCol	A logical value to indicate whether subtracting column mean from the raw data or not. Default is not subtracting column mean but for two color array it automatically subtracts the column mean.

Value

It returns `anova` and `anova.subcol`. Depending on 'subCol' option, one field may not contain any information. Still it needs two fields to calculate Fss test statistics. `anova` and `anova.subcol` contains the same following fields.

yhat	Fitted intensity value which has the same dimension as the input intensity data
S2	Variance components for the random terms. It is a matrix with number of rows equals to the number of genes and number of columns equals to the number of random terms. Note that for fixed effect model, S2 is a one column vector for error's variance.
G	Gene effects. A vector with the same length as the number of genes.
reference	The estimates for reference sample. If there is no reference sample specified in the design, this field will be absent in the output object.
S2.level	A list of strings to indicate the order of the S2 field. Note that the last column of S2 is always the error's variance. S2.level is only for the non-error terms. For example, if there are three columns in S2 and S2.level is c("Strain", "Diet"), then the three columns of S2 correspond to the variances of Strain, Diet and error respectively for each gene.
Others	Estimates (or BLUE/BLUP for mixed effect model) for the terms in model. There will be XXX.level field for each term representing the order of the estimates (similar to S2.level).
flag	A vector to indicate whether there is bad spot for this gene. 0 means no bad spot and 1 means has bad spot. If there is no flag information in input data, this field will not be available.
model	The model object used for this fitting.

Fitting mixed Effects model

Fitting mixed effects models needs a lot of computation. A good starting value for the variances is very important. This function first treats all random factors as fixed and fits a fixed effects model.

Then variances for random factors are calculated and used as the initial values for mixed effects model fitting.

There are several methods available for fitting the mixed effects model. "noest" does not really fit the mixed effects model. It takes the initial variance and solve mixed model equations to get the estimates (BLUE and BLUP). "MINQE-I" and "MINQE-UI" are based on minimum norm unbiased estimators. It is can be thought as a first iterate solution of "ML" and "REML", respectively. "ML" and "REML" are based on maximum likelihood and restricted maximum likelihood. Both of them need to be solved iteratively so they are very slow to compute. For "ML" and "REML", a MINQUE estimates is used as the starting value. "Method of scoring" is used as the iteratively algorithm to solve ML and REML. "Method of scoring" algorithm is similar to New-Raphson method except that it uses the expected value of Hessian (second derivative matrix of the objective function) instead of Hessian itself. Method of scoring is more robust to poor starting values and the Hessian is easier to calculate than Newton-Raphson.

For more mathematical details please read Searle et al.

Author(s)

Hao Wu

References

Kerr and Churchill(2001), Statistical design and the analysis of gene expression microarrays, *Genetical Research*, **77**:123-128.

Kerr, Martin and Churchill(2000), Analysis of variance for gene expression microarray data, *Journal of Computational Biology*, **7**:819-837.

Searle, Casella and McCulloch, *Variance Components*, John Wiley and sons, Inc.

See Also

makeModel, matest

Examples

```
###################################
# fixed model fitting
###################################
# load in abf1 data
data(abf1)
## Not run:

# fit model with random effect
fit.full.mix <- fitmaanova(abf1, formula = ~Strain+Sample,
   random = ~Sample)

# this is to explain the usage of including covariate variable.
# .CEL file is not included in the package, thus use can not use this.
# array specific covariate : add it to the design matrix
beforeRma <- ReadAffy() # suppose there are 18 arrays.
rmaData <- rma(beforeRma)
datafile <- exprs(rmaData)
design.table=data.frame(Array=row.names(pData(beforeRma)))
Strain = rep(c('Aj', 'B6', 'B6xAJ'), each=6)
Sample = rep(c(1:9), each=2)
Cov1 = sample(1:100,18) # this is artificial example
```

```
designfile.cov1 = cbind(design.table, Strain, Sample,Cov1)
data.cov1=read.madata(datafile, designfile=designfile.cov1)
fit.cov1 = fitmaanova(data.cov1,formula = ~Strain+Sample+Cov1, covariate = ~ Cov1)

# gene specific covariate - make artificial 'covM' matrix
covm = matrix(rnorm(length(datafile)), nrow=nrow(datafile))
designfile.cov2 = cbind(design.table, Strain, Sample)
data.cov2=read.madata(datafile, designfile=designfile.cov2, covM=covm)
fit.cov2 = fitmaanova(data.cov2,formula = ~Strain+Sample+covM, covariate = ~ covM)
## End(Not run)
```

`fom` *Figure of Merit*

Description

K-means clustering needs a given number of groups, which is difficult to guess in most of the cases. This function calculates the Figure of Merit values for different number of groups and generates the FOM plot (FOM value versus number of groups). Lower FOM value means better grouping. User can decide the number of groups in kmeans cluster based on that result.

Usage

```
fom(anovaobj, idx.gene, term, ngroups)
```

Arguments

`anovaobj`	An object of class `maanova`.
`idx.gene`	The index of genes to be clustered.
`term`	The factor (in formula) used in clustering. The expression level for this term will be used in clustering. This term has to correspond to the gene list, e.g, idx.gene in this function. The gene list should be the significant hits in testing this term.
`ngroups`	The number of groups for K-means cluster. This could be a vector or an integer.

Value

A vector of FOM values for the given number of groups

Author(s)

Hao Wu

References

Yeung, K.Y., D.R. Haynor, and W.L.Ruzzo (2001). Validating clustering for gene expression data. *Bioinformatics*, **17:309-318**.

See Also

`macluster`, `consensus`, `kmeans`

Examples

```
# load in data
data(abf1)
# fit the anova model
## Not run:
fit.fix = fitmaanova(abf1,formula = ~Strain)
# test Strain effect
test.fix = matest(abf1, fit.fix, term="Strain",n.perm= 1000)
# pick significant genes - pick the genes selected by Fs test
idx <- volcano(test.fix)$idx.Fs
# generate FOM
m <- fom(fit.fix, idx, "Strain", 10)
## End(Not run)
```

`geneprofile` *Expression plot for selected genes*

Description

This function generate a plot with many lines. Each line represents a gene. The y-axis is the estimated expression level for the given factor from ANOVA model. The x-axis is for the levels of the give factor, e.g., different strains.

Usage

```
geneprofile(anovaobj, term, geneidx,
            col="blue", type="b", ylim, xlab, ylab, ...)
```

Arguments

`anovaobj`	An object of class `maanova`. It should be the result from `fitmaanova`.
`term`	The terms to be plotted.
`geneidx`	The index of genes to be plotted.
`col`	The color to be used in plot.
`type`	The line type.
`ylim`	Y-axis limit.
`xlab`	X-axis label.
`ylab`	Y-axis label.
`...`	Other parameters to be passed to `plot`.

Author(s)

Hao Wu

Examples

```
# load in data
data(abf1)
# fit the anova model
## Not run:
fit.fix = fitmaanova(abf1,formula = ~Strain)
# test Strain effect
test.fix = matest(abf1, fit.fix, term="Strain",n.perm= 1000)
# pick significant genes - pick the genes selected by Fs test
idx <- volcano(test.fix)$idx.Fs
geneprofile(fit.fix, "Strain", idx)
## End(Not run)
```

---

gridcheck *Plot grid-by-grid data comparison for arrays*

Description

This function is used to check microarray data quality. It can check the data within the same array or cross different arrays.

Normally, on one array, the intensity data for both channels (Cy5 and Cy3) should be highly correlated (also apparent on the RI plot). The intensity data for the same sample on different arrays should be highly correlated too. Normally if an error happened in gridding, only a few blocks will be gridded. This function does the scatter plot on a grid basis to check the quality of hybridization and gridding.

If you only provide array1 (either an integer or a vector), it will do grid check within the same array, that is, for each slide, there will be one scatter plot for log2(Red) versus log2(Green) for each grid. If you provide array1 and array2 (both need to be one integer), it will check the data for the same sample (sample ID information is in experimental design) for these two arrays. If there's no common sample on these two arrays, the function will report an error.

In either case, you should see a nearly linear curve in all plots. If there were errors in hybridization and/or gridding, some of the plots will look messy. Then you have to check if something wrong happened, e.g., miss labeling, wrong gridding, etc.

If you don't have grid information for the data, this function will be unavailable.

Note that this function only works for 2-dye array.

Usage

```
gridcheck(rawdata, array1, array2, highlight.flag = TRUE, flag.color = "Red",
          margin = c(3.1, 3.1, 3.1, 1.1))
```

Arguments

`rawdata`	An object of class `madata`.
`array1`	A list of array numbers for which you want to do grid checking. All arrays will be checked by default. If you want to compare the same sample across arrays, this parameter must be an integer to indicate the first array number.
`array2`	The second array number if you want to do cross array comparisons.

`highlight.flag`
: A logical parameter to indicate whether to highlight the bad spot or not.

`flag.color`
: The color for bad spot; default is red.

`margin`
: A numerical vector of the form c(bottom, left, top, right) which gives the lines of the margin to be specified on the four sides of the plot. Read `par` for details.

Note

This function will plot one figure for each array. So if you have many arrays, there will be many figures generated.

Author(s)

Hao Wu

Examples

```
## Not run:
# load in data
data(kidney)
# grid check on the first arrays
gridcheck(kidney.raw, array1=1, margin=c(1,1,1,1))
graphics.off()
# grid check array 1 versus array 2
gridcheck(kidney.raw, array1=1, array2=2)
graphics.off()

## End(Not run)
```

---

kidney.raw *Kidney Data from CAMDA*

Description

This is a 24-array double reference design. Six samples are compared to a reference with dye swapped and all arrays are duplicated. Flag for bad spots is included in the data.

Usage

```
data(kidney)
```

Format

An object of class `madata`.

Source

http://www.camda.duke.edu

References

Prichard CC, Hsu L, Delrow J and Nelson PS (2001), Project normal: defining normal variance in mouse gene expression, *PNAS*, 98:13266

Examples

```
data(kidney)
```

`macluster` *Clustering analysis for Microarray experiment*

Description

This function bootstraps K-means or hierarchical clusters and builds a consensus tree (consensus group for K-means) from the bootstrap result.

Usage

```
macluster(anovaobj, term, idx.gene, what = c("gene", "sample"),
    method = c("hc", "kmean"), dist.method = "correlation",
    hc.method = "ward", kmean.ngroups, n.perm = 100)
```

Arguments

`anovaobj`	The result object for fitting ANOVA model.
`term`	The factor (in formula) used in clustering. The expression level for this term will be used in clustering. This term has to correspond to the gene list, e.g, idx.gene in this function. The gene list should be the significant hits in testing this term.
`idx.gene`	A vector indicating the list of differentially expressed genes. The expression level of these genes will be used to construct the cluster.
`what`	What to be clustered, either gene or sample.
`method`	The clustering method. Right now hierarchical clustering ("hc") and K-means ("kmean") are available.
`dist.method`	Distance measure to be used in hierarchical clustering. Besides the methods listed in `dist`, there is a new method "correlation" (default). The "correlation" distance equals to (1 - r^2), where r is the sample correlation between observations.
`hc.method`	The agglomeration method to be used in hierarchical clustering. See `hclust` for detail.
`kmean.ngroups`	The number of groups for K-means cluster.
`n.perm`	Number of bootstraps. If it is 1, this function will cluster the observed data. If it is bigger than 1, a bootstrap will be performed.

Details

Normally after the F test, user can select a list of differentially expressed genes. The next step is to investigate the relationship among these genes. Using the expression levels of these genes, the user can cluster the genes or the samples using either hierarchical or K-means clustering algorithm. In order to evaluate the stability of the relationship, this function bootstraps the data, re-fits the model and recluster the genes/samples. Then for a certain number of bootstrap iterations, say, 1000, we have 1000 cluster results. We can use `consensus` to build the consensus tree from these 1000 trees.

Note that if you have a large number (say, more than 100) of genes/samples to cluster, hierarchical clustering could be very unstable. A slight change in the data can result in a big change in the tree structure. In that case, K-means will give better results.

Value

An object of class `macluster`.

Author(s)

Hao Wu

See Also

`hclust`, `kmeans`, `consensus`

Examples

```
# load in data
data(abf1)
# fit the anova model
## Not run:
fit.fix = fitmaanova(abf1,formula = ~Strain)
# test Strain effect
test.fix = matest(abf1, fit.fix, term="Strain",n.perm= 1000)
# pick significant genes - pick the genes selected by Fs test
idx <- volcano(test.fix)$idx.Fs
# do k-means cluster on genes
gene.cluster <- macluster(fit.fix, term="Strain", idx, what="gene",
   method="kmean", kmean.ngroups=5, n.perm=100)
# get the consensus group
consensus(gene.cluster, 0.5)

# HC cluster on samples
sample.cluster <- macluster(fit.fix, term="Strain", idx, what="sample",method="hc")
# get the consensus group
consensus(sample.cluster, 0.5)
## End(Not run)
```

---

matest *Statistical test for Microarray experiment*

Description

This is the function to perform F or T test on one or multiple experimental factor(s). Permutation test will be carried upon request.

Usage

```
matest(data, anovaobj, term, Contrast, n.perm=1000, nnodes=1,
       critical=.9, test.type = c("ttest", "ftest"),
       shuffle.method=c("sample", "resid"),
       MME.method=c("REML","noest","ML"),
       test.method=c(1,1),pval.pool=TRUE, verbose=TRUE)
```

Arguments

data	An object of class madata.
anovaobj	An object of class fitmaanova.
term	The term(s) to be tested. It can be multiple terms. Note that the tested term must be fixed. If the term to be tested is a random term, it will be converted to a fixed term than do test.
Contrast	The contrast matrix for the term. The number of columns equals to the number of levels in the term. The number of rows is the number of T-test you want to carry. Note that it must be a matrix. Use PairContrast to make all possible pairwise comparison or matrix command to make it manually. Note that the the hypothesis test can be formulated as H0: Lb=0 versus alternative. This contrast matrix is L. For testing a covariate, use a one by one contrast matrix of 1.
n.perm	An integer for number of permutations.
nnodes	Number of nodes in the MPI cluster. If 1, the permutation test will be running on the local computer.
critical	percentile of F-distribution used to get a subset to calculate p-value. Default is 90th percentile of F-distribution, and permutation analysis is conducted based on genes whose test statistics are smaller than 90th percentile of the F-distribution.
test.type	Test type. It could be F-test or T-test. If the Contrast matrix is missing, this should be a "ftest" and the contrast matrix is generated automatically to cover the whole linear space except for testing covariates. If the Contrast matrix is given, this could be "ftest" or "ttest". The default is "ttest" (for backward compatability). For T-test, the code will do a series of T-test, where each T-test corresponds to a row in the contrast matrix.
shuffle.method	Data shuffling method. "sample" for sample shuffling and "resid" for residual shuffling. Read "Data Shuffling" section for detail.
MME.method	The method used to solve the Mixed Model Equations. See fitmaanova for detail. This parameter only applies for mixed effects model permutation test. Default method is "REML". The variance components for observed data will be used for permuted data. It will greatly increase the speed but you may lose power in statistical test in some cases.
test.method	An integer vector of two elements to indicate which F test to carry. Default is c(1,1), which means do F1 and Fs test.
pval.pool	A logical value to indicate whether to use pooled permutation F values to calculate the P values.
verbose	A logical value to indicate whether to display some message for calculation progress.

Details

If user provide a comparison matrix, this function will perform T-test on the given comparison(s). Otherwise, this function will perform F-test for the given term.

There are three types of tests available. All three tests are based on the gene-specific ANOVA model. F1 is the usual F statistic, Fs is based on the James-Stein shrinkage estimates of the error variance.

Permutation tests can run on MPI cluster. This feature is only available for Unix/Linux system. Several other R packages (such like SNOW, Rmpi, etc.) are needed for using cluster. You may need help from your system administrator to setup LAM-MPI correctly. For detailed information on LAM-MPI cluster setup and the cluster usage in R, read "MPI_README" distributed with the package.

Value

An object of class `matest`, which is a list of the following components:

`model`	Input model object.
`term`	The input term(s) to be tested.
`dfde`	Denominator's degree of freedom for the test.
`dfnu`	Numerator's degree of freedom for the test. Note that this is always 1 for T-test.
`obsAnova`	An object of `fitmaanova`, which is the ANOVA model fitting result on the original data.
`Contrast`	The contrast matrix used in the test.
`n.perm`	Number of permutations.
`shuffle`	Shuffle style
`pval.pool`	Use pooled P value or not.
`F1, Fs`	Objects of four different F tests results. All or any of them could be there according to the requested F test method. Each of them contains the following fields: • FobsF value for the observed data. • PtabTabulated P values for the observed data. • PvalpermNominal permutation P values for each gene. This field will be unavailable if user do not do permutation test. • PvalmaxFWER one-step adjusted P values from the permutation test. All the F values and P values are matrices. The number of rows in the matrices equals to the number of genes. For F-test, the number of columns will be one. For T-test, the number of columns equals to the number of tests carried.

Data Shuffling

Data shuffling method is a crucial part in the permutation test. Currently there are two shuffling method available, residual shuffling and sample shuffling.

Residual shuffling is to shuffle the null model residuals within gene without replacement.

Sample shuffling is to shuffle the samples based on the nesting relationship among the experimental factors in the model. For sample shuffling, you need to make sure you have a good sample size. Otherwise the result may be biased.

Author(s)

Hao Wu

References

Cui, X. and Churchill,GA (2003), Statistical tests for differential expression in cDNA Microarray experiments, *Genome Biology* 4:210.

Cui, X., Hwang, J.T.G., Blades N., Qiu J. and Churchill GA (2003), Improved statistical tests for differential gene expression by shrinking variance components, to be submitted.

See Also

makeModel, fitmaanova

Examples

```
# load in abf1 data
data(abf1)
## Not run:
fit.full.mix <- fitmaanova(abf1, formula = ~Strain+Sample,
   random = ~Sample)
ftest.all = matest(abf1, fit.full.mix, test.method=c(1,1),
    shuffle.method="sample", term="Strain", n.perm= 100)
C = matrix(c(1,-1,0,1,0,-1), ncol=3, byrow=T)
ftest.pair = matest(abf1, fit.full.mix, Contrast = C,
   term="Strain", n.perm=100)
## End(Not run)
```

---

read.madata *Read Microarray data*

Description

This is the function to read Microarray experiment data from a TAB delimited text file or matrix object.

Usage

```
read.madata(datafile=datafile, designfile=designfile, covM = covM,
arrayType=c("oneColor", "twoColor"),header=TRUE, spotflag=FALSE, n.rep=1, avgrep
  log.trans=FALSE, metarow, metacol, row, col, probeid, intensity, matchDataToDe
```

Arguments

datafile	Matrix R object or data file name with path name as a string.
designfile	Matrix or data.frame R object or design file name with path as a string.
covM	Gene specific covariate matrix. Specify this only if you have gene specific covariate matrix.
arrayType	Specify if it is one or two color array. Default is one color.
header	A logical value indicating when input files (data file, design file or covariate matrix) are TAB delimited file, whether they have column header.
spotflag	A flag to indicate whether the input file contains the flag for bad spot or not.
n.rep	An integer to represent the number of replicates.
avgreps	An integer to indicate whether to average or collapse the replicates or not. 0 means no average; 1 means to take the mean of the replicates; 2 means to take the median of the replicates.
log.trans	A logical value to indicate whether to take log2 transformation on the raw data or not. It is FALSE by default.If this is TRUE, TransformMethod field will be set to "log2".
metarow	For 2-dye array. The column number for meta row. Default values are 1s.

metacol	For 2-day array. The column number for meta column. Default values are 1s.
row	For 2-day array. The column number for row. Default value is NA.
col	For 2-day array. The column number for column. Default value is NA.
probeid	The column number storing probe (clone) id. When datafile is matrix R object, it assumes rowname of the data is probe id. If data does not have row name, then 1,2,... is used as a probe id. For TAB delimited file, if probeid is not provided, it assumes that the first column stores the probe id. If you do not have probe id then set probeid = 0.
intensity	The start column number of intensity. For the matrix R object, it assumes intensity starts from the first column and for TAB delimited file, it assumes intensity stars from the second column, as a default.
matchDataToDesign	Defaults to false. If set to TRUE then the datafile column headers (or colnames(datafile) in the case of a matrix) will be matched up to the design file's Array column. This allows you to ignore the input order of array data as long as the datafile's header values can be matched exactly to the designfile's Array values
...	Other gene information in the data file.

Value

An object of class madata, which is a list of following components:

n.gene	Total number of genes in the experiment.
n.rep	Number of replicates in the experiment, if .
n.spot	Number of spots for each gene.
data	data field. It is either the log2 transformed data (if log.trans=TRUE), or just the original data (if log.trans=FALSE).
n.array	Number of arrays in the experiment.
n.dye	Number of dyes.
flag	A matrix for spot flag. Each element corresponding to one spot. 0 means normal spot, all other values mean bad spot.
metarow	Meta row for each spot.
metacol	Meta column for each spot.
row	Row for each spot.
col	Column for each spot.
ArrayName	A list of strings to represent the names of intensity data.
design	An object to represent the experimental design.
Others	Other experiment information listed in the data file and specified by user.

Preparing data file

Before using the package, user need to prepare the input data file.

1) The data file can be a matrix type R object, such as the output of exprs() from array or beadarray package. It is assumed that the intensity is started from the first column and row name is probe ID. Otherwise, column number containing probe ID and intensity should be specified.

2) The data file can be a TAB delimited text file. In this file, each row corresponds to a gene. In the columns, you can put some gene specific information, e.g., the Probe ID, Gene Bank ID, etc. and the grid location of the spot. But most importantly you need to put the intensity data after that. Most of the Microarray gridding software generate one file for each slide. At this point, you need to manually combine them into the data file. You need to decide which data you want to use in analysis, e.g., mean versus median, background subtracted or not, etc. For N-dye array, your intensity data should have N columns for each array. These N columns need to be adjacent to each other. You can put the spot flag as a column after intensity data for each array. (Note that if you have flag, you will have N+1 columns data for each array.) If you have replicates, replicated measurements of the same probe (clone) on the same array should appear in adjacent rows.

For example, for a 2-dye cDNA array, you have four slides scanned by Gene Pix and you get four files. First you open your favorite Spread Sheet editor, e.g., MS Excel. Copy your probe ID and Cluster ID to the first 2 columns. Then open one of the files generated by Gene Pix, copy the grid location into next 4 columns (you only need to do this once because they are all the same for four slides). Then for all four files, copy the two columns of foreground median value (if you want to use it) and one column of flag to the file in the order of Cy5, Cy3, flag. Then select the whole file and row sort it according to probe ID. Save the file as tab delimited text file and you are done.

The data file must be "full", that is, all rows have to have the same number of fields. When you have missing data in your datafile, you need to check the data or use `fill.missing` to fill in missing variable.

Sometimes leading and trailing TAB in the text file will bring problems, depends on the operating system. So user need to be careful about that.

Preparing design file

Design file can be data.frame or matrix R object or TAB delimited text file. Number of rows of this file equals number of arrays times N (the number of dyes) (plus one for column header, if design file is a TAB delimited file and header = T). The row of design file *MUST* be organized by the order of datafile unless the matchDataToDesign parameter is set to TRUE. For example, if the datafile stores the intensity from array1, array11, array2,..., then the row of designfile must follow this order. Number of columns of this file depends on the experimental design. For example, you can have "Strain", "Diet", "Sex", etc. in your design file. You *MUST* have a column named "Array" in the design file. For two-color array, in addition to the "Array" column, you must have "Sample" and "Dye" columns (case sensitive) in the design file. "Sample" should be integers representing biological individuals. Reference samples should have Sample number to be zero(0). Reference sample will always be treated as fixed factor in mixed model and it will not be involved in any test.

You must NOT have "Spot", "Label" and "covM" columns. They are reserved for spotting, labeling and covariance effects.

Note that you DO NOT have to use all factors in design file. You can put all factors in design file but turn them on/off in formula in `fitmaanova`.

Preparing covariate file

If you have array specific covariate, it should be included in the design matrix. If you have gene specific covariate, you need to prepare matrix type R object or TAB delimited text file, "covM". The size of "covM" equals to the size of intensity data (and TAB delimited text file must have column header if header = T, but NO row name). Specify covM only if you have gene specific covariate variable. Covariate variable must be a numeric value and need to be specified in the `fitmaanova`.

Author(s)

Hao Wu

Examples

```
# note that .CEL files are not distributed with the package, thus following
# code does not work. This shows how to read data from affy (or beadarray)
# package, when TAB delimited design file is ready.

## Not run:
library(affy)
beforeRma <- ReadAffy()
rmaData <- rma(beforeRma)
datafile <- exprs(rmaData)
abf1 <- read.madata(datafile=datafile,designfile="design.txt")

# make and read designfile (data.frame type R object) from R
design.table <- data.frame(Array=row.names(pData(beforeRma)));
Strain <- rep(c('Aj', 'B6', 'B6xAJ'), each=6)
Sample <- rep(c(1:9), each=2)
designfile <- cbind(design.table, Strain, Sample)
abf1 <- read.madata(datafile, designfile=designfile)

# read in a TAB delimited file with spot flag - for two color array
# HAVE TO SPECIFY that the data is from two color array
 kidney.raw <- read.madata("kidney.txt", designfile="kidneydesign.txt",
metarow=1, metacol=2, col=3, row=4, probeid=6,
intensity=7, arrayType='twoColor',log.trans=T, spotflag=T)

## End(Not run)
```

`resiplot` *Residual plot for Microarray Experiment*

Description

This is the function to plot the residuals versus fitted value figure. Two channels, e.g., red and green, are drawn in seperate figures.

Usage

```
resiplot(madata, anovaobj, header, xlab, ylab)
```

Arguments

`madata`	An object of class `madata`.
`anovaobj`	An object of class `maanova`, which is the output from `fitmaanova`.
`header`	Optional. The title of the figure. The default figure title will be "Residual vs. Yhat plot".
`xlab`	Optional. The xlab of the figure. The default will be "Yhat".
`ylab`	Optional. The ylab of the figure. The default will be "Residual".

Author(s)

Hao Wu

Examples

```
# load abf1 data
data(abf1)
## Not run:
fit.full.mix <- fitmaanova(abf1, formula = ~Strain+Sample,
   random = ~Sample)
resiplot(abf1, fit.full.mix)

## End(Not run)
```

`riplot` *Ratio intensity plot for 2-dye Microarray experiment*

Description

This function only works for 2-dye array at this time. It will plot the log-ratio (log2(R/G)) versus log-intensity (log2(R*G)/2) figure for Micro Array experiment. Ideal RI plot will be points scattered around the y=0 horizontal line.

This function works for `madata`. This function and `arrayview` assume the data is on log2 based scale. So if your rawdata is not pre-transformed, you should not do riplot on the raw data.

Usage

```
riplot(object, title, xlab, ylab, array, color = "blue",highlight.flag = TRUE,
    flag.color = "Red", idx.highlight, highlight.color = "Green",
    rep.connect = FALSE, onScreen=TRUE)
```

Arguments

`object`	An object of class `madata`.
`title`	The title for figures. The default figure title is "RI plot for array number X". If the user wants to provide titles, be sure to provide a string array with the same number of elements as the number of arrays.
`xlab`	The xlab for figures. The default figure xlab is "expression(log[2](R*G))". One xlab would be used for all plot, thus unlike title, user (if one wants) need to provide only one name.
`ylab`	The ylab for figures. The default figure ylab is "expression(log[2](R/G))". One ylab would be used for all plot, thus unlike title, user (if one wants) need to provide only one name.
`array`	A list of arrays numbers for which you want to draw an RI plot.
`color`	The color for the points in scatter plot. Default is blue.
`highlight.flag`	A logical parameter to indicate whether to highlight the bad spots or not.
`flag.color`	The color for bad spots, default is red.
`idx.highlight`	A vector for highlighted spots other than bad spots.
`highlight.color`	The color for highlighted spots. Default is green.

rep.connect	A logical value to represent whether to connect the dots between the replicates or not.
onScreen	A logical value to represent whether to display the plots on screen or not. If TRUE, x11() (in Unix/Windows) or macintosh() (in Mac) will be called inside the function. Otherwise, it will plot the figure on the current device. Default is TRUE.

Note

This function will plot one figure for each array. So if you have many arrays, there will be many figures generated.

Author(s)

Hao Wu

Examples

```
## Not run:
data(kidney)
# riplot raw data on screen
riplot(kidney.raw)
graphics.off()
# riplot raw data array 1 and 3 and output to postscript file
postscript(file="kidneyRIplot.ps")
riplot(kidney.raw, array=c(1,3), onScreen=FALSE)

## End(Not run)
```

---

subset.madata *Subsetting Microarray data objects*

Description

Return subsets of an an object of class madata meeting given conditions.

Usage

```
## S3 method for class 'madata'
subset(x, arrays, genes, ...)
```

Arguments

x	An object of class madata.
arrays	A vector specifying which arrays to keep or discard.
genes	A vector specifying which genes to keep or discard.
...	Ignored at this point.

Value

An object of class madata with specified arrays and genes.

Author(s)

Hao Wu

Examples

```
data(kidney)
## Not run:
smalldata <- subset(kidney.raw, arrays=c(1,2))
# take out the all arrays except array 1
idx.array <- 1:kidney.raw$n.array
smalldata <- subset(kidney.raw,arrays=(idx.array[-1]))
# take out gene number 1 to 20
smalldata <- subset(kidney.raw,genes=1:20)

## End(Not run)
```

---

summarytable *Summarize the matest result.*

Description

This function returns list of probe ID with p-value, FDR adjusted p-value or fold change selected by given threshold.

Usage

```
summarytable(matestobj, method=c("Fold.change","Pvalperm","adjPvalperm"),
     test =c("F1","Fs"),whichTest=c("F1.Pvalperm","F1.adjPvalperm",
         "Fs.Pvalperm","Fs.adjPvalperm"),
         threshold, outfile="summarytable.csv")
```

Arguments

matestobj	An object of class matest, which is the result from matest.
method	Results that you want to include in the output among Fold.change, Pvalperm and adjPvalperm. Default is possible every field.
test	Test that you want to include in the output among F1, Fs and Fss. Default is possible every test.
whichTest	Test result used to get the subset. If whichTest is not provided, save all result.
threshold	Threshold that you want to get the subset. If threshold is not provided, save all result.
outfile	File name that you want to save the result. Default name is 'summarytable.csv'

Details

This function use

Value

result	It returns result fields (among Fold.change, P-value or adjust P-value if you selected) from test statistics (among F1, Fs or Fss statistics if you selected), subsetted by given statistics ('whichtest') is smaller than 'threshold'.
outfile	.csv file saved under the working directory.

Author(s)

Hyuna Yang

Examples

```
data(abf1)
## Not run:
fit.full.simple = fitmaanova(abf1,formula = ~ Strain)
# F-test strain effect
ftest.all = matest(abf1, fit.full.simple, term="Strain", n.perm= 1000)
# make FDR adjusted P values
ftest.all = adjPval(ftest.all, 'jsFDR')
summarytable(ftest.all, outfile='all.csv')
smallset = summarytable(ftest.all, method ='Pvalperm',
test=c('F1','Fs'), whichTest='Fs.Pvalperm', threshold = 0.1)

## End(Not run)
```

---

transform.madata *Micro Array experiment data transformation*

Description

This is the function to transform the Micro Array experiment data based on the given method.

Usage

```
## S3 method for class 'madata'
transform(`_data`,method=c("shift","glowess","rlowess","linlog","linlogshift"),
       lolim, uplim, f=0.1, iter=3, degree=1, cg=0.3, cr=0.3, n.bin=10,
       draw=c("screen", "dev", "off"), ...)
```

Arguments

_data	An object of class madata.
method	The smoothing method.
lolim	Low shift limit. If this argument is missing, the negative of the minimum element in the pmt data is used.
uplim	High shift limit. If this argument is missing, the minimum element in the pmt data is used. lolim and uplim are applicable only if the method is "shift" or "linlogshift".
f	The smoother span. This gives the proportion of points in the plot which influence the smooth at each value. Larger values give more smoothness. It is equivalent to the "span" parameter in loess.

iter	The number of robustifying iterations which should be performed. Using smaller values of iter will make lowess run faster.
degree	The degree of the polynomials to be used in loess, up to 2. This is used when method is "glowess" or "rlowess".
cg	Percentage of genes to be transformed linearly for the green channel.
cr	Percentage of genes to be transformed linearly for the red channel.
n.bin	Number of bins for calculating the variance after linlogShift.
draw	Where to plot the transformation plots. "off" means no plot. "screen" means to display the plots on screen then x11() (in Unix/Windows) or macintosh() (in Mac) will be called inside the function. "dev" means to output the plots to the current device. User can use this option to output the plot to a file. Default option is "screen".
...	Ignored at this point.

Details

The smoothing methods include:

shift – the calculation of offset is based on the minimum sum of absolute deviations (SAD). Each array will have its own offset value. The data after shift cannot be smaller than 1.

glowess – Intensity-based lowess. This method is to smooth the scatter plot of Ratio (R/G) versus Intensity (R*G). The formula in the fitting is ratio ~ intensity.

rlowess – Joint lowess. This method is to smooth the scatter plot of Ratio versus Intensity and grid locations. It is the joint of intensity-based lowess and spatial loess. You have to have the grid location for every spot in order to use this method. The formula in fitting is ratio ~ intesity + row + col.

linlog – Linear-log transformation.

linlogshift – Linear-log shift transformation.

Previously, intensity lowess was called global lowess and joint lowess was called regional lowess. So I use "glowess" and "rlowess" in the method. Although the method names doesn't make too much sense, I will keep them for the reason of backward compatibility.

If you have replicated spots and want to collapse them in `read.madata` by providing avgreps=1 or 2, you will lose grid information and joint lowess will be unavailable.

Note that this function is only working for two-dye array.

Value

The return value is an object of class `madata`. Compared with the input object, the following fields are changed:

- Field `data` is the transformed data.
- Field `TransformMethod` will be the transformation method applied.

Author(s)

Hao Wu

References

Kerr and Churchill(2001), Statistical design and the analysis of gene expression microarrays, *Genetical Research*, **77**:123-128.

Kerr, Martin and Churchill(2000), Analysis of variance for gene expression microarray data, *Journal of Computational Biology*, **7**:819-837.

Cui, Kerr and Churchill(2002), Data transformations for cDNA Microarray data, submitted, find manuscript in www.jax.org/research/churchill.

See Also

`loess`

Examples

```
# load in data
data(kidney)
# do regional loess on raw data
## Not run:
raw.lowess <- transform.madata(kidney.raw, method="rlowess")
graphics.off()
#do shift without displaying the plot
data1.shift <- transform.madata(kidney.raw, method="shift", lolim=-50,
     uplim=50,draw="off")

# do global lowess and output the plots to a postscript file
postscript(file="glowess.ps")
data1.glowess <- transform.madata(kidney.raw, method="glowess", draw="dev")
graphics.off()

# do linear-log
data1.linlog <- transform.madata(kidney.raw, method="linlog")
graphics.off()

# do linear-log shift
data1.linlogshift <- transform.madata(kidney.raw, method="linlogshift",
  lolim=-50, uplim=50)
graphics.off()

## End(Not run)
```

`varplot` *Variance component plot*

Description

This function plots the density curve of each variance component of a result from `fitmaanova`.

If the input is from fixed model ANOVA, it will plot one curve for error variance component. If the input is from mixed model ANOVA, it will plot multiple curves, one for a random term (including error).

Usage

```
varplot(anovaobj, xlab, ylab, main)
```

Arguments

anovaobj	An object of class maanova.
xlab	Figure xlab. Default is "Sigma".
ylab	Figure ylab. Default is "Density".
main	Figure title. Default is "Density plot for sqrt of variance".

Author(s)

Hao Wu

See Also

fitmaanova, density

Examples

```
# load abf1 data
data(abf1)
## Not run:
fit.full.mix <- fitmaanova(abf1, formula = ~Strain+Sample,
   random = ~Sample)
varplot(fit.full.mix)
## End(Not run)
```

---

volcano *Volcano plot for F test results*

Description

This function generates a volcano-like plot given the F test results.

Usage

```
volcano(matestobj, threshold=c(0.001,0.05),
        method=c("unadj","unadj"), title="Volcano Plot",
        highlight.flag=TRUE, onScreen=TRUE)
```

Arguments

matestobj	An object of class matest.
threshold	A vector of three double values to indicate the thresholds for three F tests. The values should be between 0 and 1. Note that you need to put three values here even if you don't have all three F tests in matestobj.
method	A flag indicates to use which P values to generate the plot and select genes. This is a vector with three elements, which corresponds to three F tests. Each element should be one of the following five selections: • "unadj"Unadjusted tabulated P values. • "nominal"Nominal permutation P values. • "fwer"FWER one-step adjusted P values. • "fdr"FDR adjusted tabulated P values.

- "fdrperm"FDR adjusted nominal permutation P values.

Default value is c("unadj", "unadj") which means to use tabulated P values for all tests.

Note that you need to put three values here even if you don't have all three F tests in matestobj.

`title` Figure title. Default is "Volcano Plot".

`highlight.flag`
A logical value to indicate whether to highlight the genes with bad spots or not.

`onScreen` A logical value to represent whether to display the plots on screen or not. If TRUE, the figure will be plotted on the screen. Otherwise, it will plot the figure on the current device. Default is TRUE.

Details

This function allows one to visualize the results from the F or T tests. The figure looks like an erupting volcano. There will be one plot For F-test result and multiple plots for T-test result, each plot corresponds to one T-test. You must have F1 test result in the input object in order to do volcano plot.

On the plot, blue dots are the genes selected by the F1 test. The y-axis value is -log10(P-value) for the F1 test and x-axis value is proportional to the fold changes. A horizontal line represents the significance threshold of the F1 test. The red dots are the genes selected by the Fs test (if there's Fs test result). If there is flag information in the data and the user wants to highlight the flagged genes, the genes with any bad spots will be circled by a black circle.

Value

For F-test volcano plot, it returns an object which is a list of the following four fields:

`idx.F1` The significant genes selected by F1 test.

`idx.Fs` The significant genes selected by Fs test.

`idx.all` The significant genes selected by all four F tests.

For T-test volcano plot, it returns an array of the above object. Each element in the array corresponds to one T-test.

Author(s)

Hao Wu

Examples

```
## Not run:
data(abf1)
fit.full.mix <- fitmaanova(abf1, formula = ~Strain+Sample,
    random = ~Sample)
ftest.all = matest(abf1, fit.full.mix, test.method=c(1,1),
    shuffle.method="sample", term="Strain", n.perm= 100)
volcano(ftest.all)

## End(Not run)
```

`write.madata` *Write Micro Array data to a TAB delimited simple text file*

Description

This function is used to write the contents of an object of class `madata` to a TAB delimited simple text file.

Usage

```
write.madata(madata, datafile="madata.txt", designfile="design.txt")
```

Arguments

`madata`	The object to be output. It must be an object of class `madata`.
`datafile`	The output file name for the data.
`designfile`	The output file name for the design file.

Author(s)

Hao Wu

Examples

```
# load abf1 data
data(abf1)
# take out first 6 arrays
## Not run:
smalldata <- subset(abf1, array=1:6)
# write to file
write.madata(smalldata, datafile="smallabf1.txt",
      designfile="smallabf1design.txt")
## End(Not run)
```

Index