Package ‘preprocessCore’

March 10, 2026

Version 1.72.0

Title A collection of pre-processing functions

Author Ben Bolstad <bmb@bmbolstad.com>

Maintainer Ben Bolstad <bmb@bmbolstad.com>

Imports stats

Description A library of core preprocessing routines.

License LGPL (>= 2)

URL https://github.com/bmbolstad/preprocessCore

Collate normalize.quantiles.R quantile_extensions.R rma.background.correct.R rcModel.R colSummarize.R subColSummarize.R plmr.R plmd.R

LazyLoad yes

biocViews Infrastructure

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

git_branch RELEASE_3_22

git_last_commit f8fc99a

git_last_commit_date 2025-10-29

Repository Bioconductor 3.22

Date/Publication 2026-03-09

Contents

colSumamrize 2
normalize.quantiles 3
normalize.quantiles.in.blocks 4
normalize.quantiles.robust 5
normalize.quantiles.target 7
rcModelPLMd 8
rcModelPLMr 9
rcModels 11
rma.background.correct 13
subColSummarize 14
subrcModels 15

Index **18**

`colSumamrize` *Summarize the column of matrices*

Description

Compute column wise summary values of a matrix.

Usage

```
colSummarizeAvg(y)
colSummarizeAvgLog(y)
colSummarizeBiweight(y)
colSummarizeBiweightLog(y)
colSummarizeLogAvg(y)
colSummarizeLogMedian(y)
colSummarizeMedian(y)
colSummarizeMedianLog(y)
colSummarizeMedianpolish(y)
colSummarizeMedianpolishLog(y)
```

Arguments

`y`	A numeric matrix

Details

This groups of functions summarize the columns of a given matrices.

- `colSummarizeAvg`Take means in column-wise manner
- `colSummarizeAvgLoglog2` transform the data and then take means in column-wise manner
- `colSummarizeBiweight`Summarize each column using a one step Tukey Biweight procedure
- `colSummarizeBiweightLoglog2` transform the data and then summarize each column using a one step Tuke Biweight procedure
- `colSummarizeLogAvg`Compute the mean of each column and then `log2` transform it
- `colSummarizeLogMedian`Compute the median of each column and then `log2` transform it
- `colSummarizeMedian`Compute the median of each column
- `colSummarizeMedianLoglog2` transform the data and then summarize each column using the median
- `colSummarizeMedianpolish`Use the median polish to summarize each column, by also using a row effect (not returned)
- `colSummarizeMedianpolishLoglog2` transform the data and then use the median polish to summarize each column, by also using a row effect (not returned)

Value

A list with following items:

`Estimates`	Summary values for each column.
`StdErrors`	Standard error estimates.

Author(s)

B. M. Bolstad <bmb@bmbolstad.com>

Examples

```
y <- matrix(10+rnorm(100),20,5)

colSummarizeAvg(y)
colSummarizeAvgLog(y)
colSummarizeBiweight(y)
colSummarizeBiweightLog(y)
colSummarizeLogAvg(y)
colSummarizeLogMedian(y)
colSummarizeMedian(y)
colSummarizeMedianLog(y)
colSummarizeMedianpolish(y)
colSummarizeMedianpolishLog(y)
```

---

normalize.quantiles *Quantile Normalization*

Description

Using a normalization based upon quantiles, this function normalizes a matrix of probe level intensities.

Usage

```
normalize.quantiles(x,copy=TRUE, keep.names=FALSE)
```

Arguments

x	A matrix of intensities where each column corresponds to a chip and each row is a probe.
copy	Make a copy of matrix before normalizing. Usually safer to work with a copy, but in certain situations not making a copy of the matrix, but instead normalizing it in place will be more memory friendly.
keep.names	Boolean option to preserve matrix row and column names in output.

Details

This method is based upon the concept of a quantile-quantile plot extended to n dimensions. No special allowances are made for outliers. If you make use of quantile normalization please cite Bolstad et al, Bioinformatics (2003).

This functions will handle missing data (ie NA values), based on the assumption that the data is missing at random.

Note that the current implementation optimizes for better memory usage at the cost of some additional run-time.

Value

A normalized `matrix`.

Author(s)

Ben Bolstad, <bmbolstad.com>

References

Bolstad, B (2001) *Probe Level Quantile Normalization of High Density Oligonucleotide Array Data*. Unpublished manuscript http://bmbolstad.com/stuff/qnorm.pdf

Bolstad, B. M., Irizarry R. A., Astrand, M, and Speed, T. P. (2003) *A Comparison of Normalization Methods for High Density Oligonucleotide Array Data Based on Bias and Variance.* Bioinformatics 19(2) ,pp 185-193. http://bmbolstad.com/misc/normalize/normalize.html

See Also

normalize.quantiles.robust

---

normalize.quantiles.in.blocks

Quantile Normalization carried out separately within blocks of rows

---

Description

Using a normalization based upon quantiles this function normalizes the columns of a matrix such that different subsets of rows get normalized together.

Usage

```
normalize.quantiles.in.blocks(x,blocks,copy=TRUE)
```

Arguments

x	A matrix of intensities where each column corresponds to a chip and each row is a probe.
copy	Make a copy of matrix before normalizing. Usually safer to work with a copy
blocks	A vector giving block membership for each each row

Details

This method is based upon the concept of a quantile-quantile plot extended to n dimensions. No special allowances are made for outliers. If you make use of quantile normalization either through rma or expresso please cite Bolstad et al, Bioinformatics (2003).

Value

From normalize.quantiles.use.target a normalized matrix.

Author(s)

Ben Bolstad, <bmb@bmbolstad.com>

References

Bolstad, B (2001) *Probe Level Quantile Normalization of High Density Oligonucleotide Array Data*. Unpublished manuscript http://bmbolstad.com/stuff/qnorm.pdf

Bolstad, B. M., Irizarry R. A., Astrand, M, and Speed, T. P. (2003) *A Comparison of Normalization Methods for High Density Oligonucleotide Array Data Based on Bias and Variance.* Bioinformatics 19(2) ,pp 185-193. http://bmbolstad.com/misc/normalize/normalize.html

See Also

normalize.quantiles

Examples

```
### setup the data
blocks <- c(rep(1,5),rep(2,5),rep(3,5))
par(mfrow=c(3,2))
x <- matrix(c(rexp(5,0.05),rnorm(5),rnorm(5,10)))
boxplot(x ~ blocks)
y <- matrix(c(-rexp(5,0.05),rnorm(5,10),rnorm(5)))
boxplot(y ~ blocks)
pre.norm <- cbind(x,y)

### the in.blocks version
post.norm <- normalize.quantiles.in.blocks(pre.norm,blocks)
boxplot(post.norm[,1] ~ blocks)
boxplot(post.norm[,2] ~ blocks)

### the usual version
post.norm  <- normalize.quantiles(pre.norm)
boxplot(post.norm[,1] ~ blocks)
boxplot(post.norm[,2] ~ blocks)
```

---

normalize.quantiles.robust

Robust Quantile Normalization

Description

Using a normalization based upon quantiles, this function normalizes a matrix of probe level intensities. Allows weighting of chips

Usage

```
normalize.quantiles.robust(x,copy=TRUE,weights=NULL,
          remove.extreme=c("variance","mean","both","none"),
          n.remove=1,use.median=FALSE,use.log2=FALSE, keep.names=FALSE)
```

Arguments

`x`	A matrix of intensities, columns are chips, rows are probes
`copy`	Make a copy of matrix before normalizing. Usually safer to work with a copy
`weights`	A vector of weights, one for each chip
`remove.extreme`	If weights is null, then this will be used for determining which chips to remove from the calculation of the normalization distribution, See details for more info
`n.remove`	number of chips to remove
`use.median`	if TRUE use the median to compute normalization chip, otherwise uses a weighted mean
`use.log2`	work on log2 scale. This means we will be using the geometric mean rather than ordinary mean
`keep.names`	Boolean option to preserve matrix row and column names in output.

Details

This method is based upon the concept of a quantile-quantile plot extended to n dimensions. Note that the matrix is of intensities not log intensities. The function performs better with raw intensities.

Choosing **variance** will remove chips with variances much higher or lower than the other chips, **mean** removes chips with the mean most different from all the other means, **both** removes first extreme variance and then an extreme mean. The option **none** does not remove any chips, but will assign equal weights to all chips.

Note that this function does not handle missing values (ie NA). Unexpected results might occur in this situation.

Value

a matrix of normalized intensites

Note

This function is still experimental.

Author(s)

Ben Bolstad, <bmb@bmbolstad.com>

See Also

`normalize.quantiles`

normalize.quantiles.target

Quantile Normalization using a specified target distribution vector

Description

Using a normalization based upon quantiles, these function normalizes the columns of a matrix based upon a specified normalization distribution

Usage

```
normalize.quantiles.use.target(x,target,copy=TRUE,subset=NULL)
normalize.quantiles.determine.target(x,target.length=NULL,subset=NULL)
```

Arguments

x	A matrix of intensities where each column corresponds to a chip and each row is a probe.
copy	Make a copy of matrix before normalizing. Usually safer to work with a copy
target	A vector containing datapoints from the distribution to be normalized to
target.length	number of datapoints to return in target distribution vector. If NULL then this will be taken to be equal to the number of rows in the matrix.
subset	A logical variable indexing whether corresponding row should be used in reference distribution determination

Details

This method is based upon the concept of a quantile-quantile plot extended to n dimensions. No special allowances are made for outliers. If you make use of quantile normalization either through rma or expresso please cite Bolstad et al, Bioinformatics (2003).

These functions will handle missing data (ie NA values), based on the assumption that the data is missing at random.

Value

From normalize.quantiles.use.target a normalized matrix.

Author(s)

Ben Bolstad, <bmb@bmbolstad.com>

References

Bolstad, B (2001) *Probe Level Quantile Normalization of High Density Oligonucleotide Array Data*. Unpublished manuscript http://bmbolstad.com/stuff/qnorm.pdf

Bolstad, B. M., Irizarry R. A., Astrand, M, and Speed, T. P. (2003) *A Comparison of Normalization Methods for High Density Oligonucleotide Array Data Based on Bias and Variance.* Bioinformatics 19(2) ,pp 185-193. http://bmbolstad.com/misc/normalize/normalize.html

See Also

normalize.quantiles

---

rcModelPLMd *Fit robust row-column models to a matrix*

Description

These functions fit row-column effect models to matrices using PLM-d

Usage

```
rcModelPLMd(y,group.labels)
```

Arguments

y	A numeric matrix
group.labels	A vector of group labels. Of length ncol(y)

Details

This functions first tries to fit row-column models to the specified input matrix. Specifically the model

$$y_{ij} = r_i + c_j + \epsilon_{ij}$$

with r_i and c_j as row and column effects respectively. Note that these functions treat the row effect as the parameter to be constrained using sum to zero.

Next the residuals for each row are compared to the group variable. In cases where there appears to be a significant relationship, the row-effect is "split" and separate row-effect parameters, one for each group, replace the single row effect.

Value

A list with following items:

Estimates	The parameter estimates. Stored in column effect then row effect order
Weights	The final weights used
Residuals	The residuals
StdErrors	Standard error estimates. Stored in column effect then row effect order
WasSplit	An indicator variable indicating whether or not a row was split with separate row effects for each group

Author(s)

B. M. Bolstad <bmb@bmbolstad.com>

See Also

rcModelPLM,rcModelPLMr

Examples

```
col.effects <- c(10,11,10.5,12,9.5)
row.effects <- c(seq(-0.5,-0.1,by=0.1),seq(0.1,0.5,by=0.1))


y <- outer(row.effects, col.effects,"+")
y <- y + rnorm(50,sd=0.1)

rcModelPLMd(y,group.labels=c(1,1,2,2,2))


row.effects <- c(4,3,2,1,-1,-2,-3,-4)
col.effects  <- c(8,9,10,11,12,10)

y <- outer(row.effects, col.effects,"+") + rnorm(48,0,0.25)

y[8,4:6] <- c(11,12,10)+ 2.5 + rnorm(3,0,0.25)
y[5,4:6] <- c(11,12,10)+-2.5 + rnorm(3,0,0.25)


rcModelPLMd(y,group.labels=c(1,1,1,2,2,2))

par(mfrow=c(2,2))
matplot(y,type="l",col=c(rep("red",3),rep("blue",3)),ylab="residuals",xlab="probe",main="Observed Data")
matplot(rcModelPLM(y)$Residuals,col=c(rep("red",3),rep("blue",3)),ylab="residuals",xlab="probe",main="Resid
matplot(rcModelPLMd(y,group.labels=c(1,1,1,2,2,2))$Residuals,col=c(rep("red",3),rep("blue",3)),xlab="probe"
```

`rcModelPLMr` *Fit robust row-column models to a matrix*

Description

These functions fit row-column effect models to matrices using PLM-r and variants

Usage

```
rcModelPLMr(y)
rcModelPLMrr(y)
rcModelPLMrc(y)
rcModelWPLMr(y, w)
rcModelWPLMrr(y, w)
rcModelWPLMrc(y, w)
```

Arguments

`y`	A numeric matrix
`w`	A matrix or vector of weights. These should be non-negative.

Details

These functions fit row-column models to the specified input matrix. Specifically the model

$$y_{ij} = r_i + c_j + \epsilon_{ij}$$

with r_i and c_j as row and column effects respectively. Note that these functions treat the row effect as the parameter to be constrained using sum to zero.

The `rcModelPLMr` and `rcModelWPLMr` functions use the PLM-r fitting procedure. This adds column and row robustness to single element robustness.

The `rcModelPLMrc` and `rcModelWPLMrc` functions use the PLM-rc fitting procedure. This adds column robustness to single element robustness.

The `rcModelPLMrr` and `rcModelWPLMrr` functions use the PLM-rr fitting procedure. This adds row robustness to single element robustness.

Value

A list with following items:

`Estimates`	The parameter estimates. Stored in column effect then row effect order
`Weights`	The final weights used
`Residuals`	The residuals
`StdErrors`	Standard error estimates. Stored in column effect then row effect order

Author(s)

B. M. Bolstad <bmb@bmbolstad.com>

See Also

`rcModelPLM`,`rcModelPLMd`

Examples

```
col.effects <- c(10,11,10.5,12,9.5)
row.effects <- c(seq(-0.5,-0.1,by=0.1),seq(0.1,0.5,by=0.1))


y <- outer(row.effects, col.effects,"+")
w <- runif(50)

rcModelPLMr(y)
rcModelWPLMr(y, w)


### An example where there no or only occasional outliers
y <- y + rnorm(50,sd=0.1)
par(mfrow=c(2,2))
image(1:10,1:5,rcModelPLMr(y)$Weights,xlab="row",ylab="col",main="PLM-r",zlim=c(0,1))
image(1:10,1:5,rcModelPLMrc(y)$Weights,xlab="row",ylab="col",main="PLM-rc",zlim=c(0,1))
image(1:10,1:5,rcModelPLMrr(y)$Weights,xlab="row",ylab="col",main="PLM-rr",zlim=c(0,1))
matplot(y,type="l")
```

```
### An example where there is a row outlier
y <- outer(row.effects, col.effects,"+")
y[1,] <- 11+ rnorm(5)

y <- y + rnorm(50,sd=0.1)

par(mfrow=c(2,2))
image(1:10,1:5,rcModelPLMr(y)$Weights,xlab="row",ylab="col",main="PLM-r",zlim=c(0,1))
image(1:10,1:5,rcModelPLMrc(y)$Weights,xlab="row",ylab="col",main="PLM-rc",zlim=c(0,1))
image(1:10,1:5,rcModelPLMrr(y)$Weights,xlab="row",ylab="col",main="PLM-rr",zlim=c(0,1))
matplot(y,type="l")

### An example where there is a column outlier
y <- outer(row.effects, col.effects,"+")
w <- rep(1,50)

y[,4] <- 12 + rnorm(10)
y <- y + rnorm(50,sd=0.1)

par(mfrow=c(2,2))
image(1:10,1:5,rcModelWPLMr(y,w)$Weights,xlab="row",ylab="col",main="PLM-r",zlim=c(0,1))
image(1:10,1:5,rcModelWPLMrc(y,w)$Weights,xlab="row",ylab="col",main="PLM-rc",zlim=c(0,1))
image(1:10,1:5,rcModelWPLMrr(y,w)$Weights,xlab="row",ylab="col",main="PLM-rr",zlim=c(0,1))
matplot(y,type="l")


### An example where there is both column and row outliers
y <- outer(row.effects, col.effects,"+")
w <- rep(1,50)

y[,4] <- 12 + rnorm(10)
y[1,] <- 11+ rnorm(5)

y <- y + rnorm(50,sd=0.1)

par(mfrow=c(2,2))
image(1:10,1:5,rcModelWPLMr(y,w)$Weights,xlab="row",ylab="col",main="PLM-r",zlim=c(0,1))
image(1:10,1:5,rcModelWPLMrc(y,w)$Weights,xlab="row",ylab="col",main="PLM-rc",zlim=c(0,1))
image(1:10,1:5,rcModelWPLMrr(y,w)$Weights,xlab="row",ylab="col",main="PLM-rr",zlim=c(0,1))
matplot(y,type="l")
```

`rcModels` *Fit row-column model to a matrix*

Description

These functions fit row-column effect models to matrices

Usage

```
rcModelPLM(y,row.effects=NULL,input.scale=NULL)
rcModelWPLM(y, w,row.effects=NULL,input.scale=NULL)
rcModelMedianPolish(y)
```

Arguments

`y`	A numeric matrix
`w`	A matrix or vector of weights. These should be non-negative.
`row.effects`	If these are supplied then the fitting procedure uses these (and analyzes individual columns separately)
`input.scale`	If supplied will be used rather than estimating the scale from the data

Details

These functions fit row-column models to the specified input matrix. Specifically the model

$$y_{ij} = r_i + c_j + \epsilon_{ij}$$

with r_i and c_j as row and column effects respectively. Note that this functions treat the row effect as the parameter to be constrained using sum to zero (for `rcModelPLM` and `rcModelWPLM`) or median of zero (for `rcModelMedianPolish`).

The `rcModelPLM` and `rcModelWPLM` functions use a robust linear model procedure for fitting the model.

The function `rcModelMedianPolish` uses the median polish algorithm.

Value

A list with following items:

`Estimates`	The parameter estimates. Stored in column effect then row effect order
`Weights`	The final weights used
`Residuals`	The residuals
`StdErrors`	Standard error estimates. Stored in column effect then row effect order
`Scale`	Scale Estimates

Author(s)

B. M. Bolstad <bmb@bmbolstad.com>

See Also

`rcModelPLMr`,`rcModelPLMd`

Examples

```
col.effects <- c(10,11,10.5,12,9.5)
row.effects <- c(seq(-0.5,-0.1,by=0.1),seq(0.1,0.5,by=0.1))


y <- outer(row.effects, col.effects,"+")
w <- runif(50)

rcModelPLM(y)
rcModelWPLM(y, w)
rcModelMedianPolish(y)

y <- y + rnorm(50)
```

```
rcModelPLM(y)
rcModelWPLM(y, w)
rcModelMedianPolish(y)



rcModelPLM(y,row.effects=row.effects)
rcModelWPLM(y,w,row.effects=row.effects)

rcModelPLM(y,input.scale=1.0)
rcModelWPLM(y, w,input.scale=1.0)
rcModelPLM(y,row.effects=row.effects,input.scale=1.0)
rcModelWPLM(y,w,row.effects=row.effects,input.scale=1.0)
```

`rma.background.correct` *RMA Background Correction*

Description

Background correct each column of a matrix

Usage

```
rma.background.correct(x,copy=TRUE)
```

Arguments

`x`	A matrix of intensities where each column corresponds to a chip and each row is a probe.
`copy`	Make a copy of matrix before background correctiong. Usually safer to work with a copy, but in certain situations not making a copy of the matrix, but instead background correcting it in place will be more memory friendly.

Details

Assumes PMs are a convolution of normal and exponentional. So we observe X+Y where X is backround and Y is signal. `bg.adjust` returns E[Y|X+Y, Y>0] as our backround corrected PM.

Value

A RMA background corrected `matrix`.

Author(s)

Ben Bolstad, <bmbolstad.com>

References

Bolstad, BM (2004) *Low Level Analysis of High-density Oligonucleotide Array Data: Background, Normalization and Summarization*. PhD Dissertation. University of California, Berkeley. pp 17-21

`subColSummarize` *Summarize columns when divided into groups of rows*

Description

These functions summarize columns of a matrix when the rows of the matrix are classified into different groups

Usage

```
subColSummarizeAvg(y, group.labels)
       subColSummarizeAvgLog(y, group.labels)
       subColSummarizeBiweight(y, group.labels)
       subColSummarizeBiweightLog(y, group.labels)
       subColSummarizeLogAvg(y, group.labels)
       subColSummarizeLogMedian(y, group.labels)
       subColSummarizeMedian(y, group.labels)
       subColSummarizeMedianLog(y, group.labels)
       subColSummarizeMedianpolish(y, group.labels)
       subColSummarizeMedianpolishLog(y, group.labels)
       convert.group.labels(group.labels)
```

Arguments

`y`	A numeric `matrix`
`group.labels`	A vector to be treated as a factor variable. This is used to assign each row to a group. NA values should be used to exclude rows from consideration

Details

These functions are designed to summarize the columns of a matrix where the rows of the matrix are assigned to groups. The summarization is by column across all rows in each group.

- subColSummarizeAvgSummarize by taking mean
- subColSummarizeAvgLog`log2` transform the data and then take means in column-wise manner
- subColSummarizeBiweightUse a one-step Tukey Biweight to summarize columns
- subColSummarizeBiweightLog`log2` transform the data and then use a one-step Tukey Biweight to summarize columns
- subColSummarizeLogAvgSummarize by taking mean and then taking `log2`
- subColSummarizeLogMedianSummarize by taking median and then taking `log2`
- subColSummarizeMedianSummarize by taking median
- subColSummarizeMedianLog`log2` transform the data and then summarize by taking median
- subColSummarizeMedianpolishUse the median polish to summarize each column, by also using a row effect (not returned)
- subColSummarizeMedianpolishLog`log2` transform the data and then use the median polish to summarize each column, by also using a row effect (not returned)

Value

A matrix containing column summarized data. Each row corresponds to data column summarized over a group of rows.

Author(s)

B. M. Bolstad <bmb@bmbolstad.com>

Examples

```
### Assign the first 10 rows to one group and
### the second 10 rows to the second group
###
y <- matrix(c(10+rnorm(50),20+rnorm(50)),20,5,byrow=TRUE)

subColSummarizeAvgLog(y,c(rep(1,10),rep(2,10)))
subColSummarizeLogAvg(y,c(rep(1,10),rep(2,10)))
subColSummarizeAvg(y,c(rep(1,10),rep(2,10)))

subColSummarizeBiweight(y,c(rep(1,10),rep(2,10)))
subColSummarizeBiweightLog(y,c(rep(1,10),rep(2,10)))

subColSummarizeMedianLog(y,c(rep(1,10),rep(2,10)))
subColSummarizeLogMedian(y,c(rep(1,10),rep(2,10)))
subColSummarizeMedian(y,c(rep(1,10),rep(2,10)))



subColSummarizeMedianpolishLog(y,c(rep(1,10),rep(2,10)))
subColSummarizeMedianpolish(y,c(rep(1,10),rep(2,10)))
```

---

subrcModels — *Fit row-column model to a matrix*

Description

These functions fit row-column effect models to matrices

Usage

```
subrcModelPLM(y, group.labels,row.effects=NULL,input.scale=NULL)
subrcModelMedianPolish(y, group.labels)
```

Arguments

y	A numeric matrix
group.labels	A vector to be treated as a factor variable. This is used to assign each row to a group. NA values should be used to exclude rows from consideration

`row.effects`	If these are supplied then the fitting procedure uses these (and analyzes individual columns separately)
`input.scale`	If supplied will be used rather than estimating the scale from the data

Details

These functions fit row-column models to the specified input matrix. Specifically the model

$$y_{ij} = r_i + c_j + \epsilon_{ij}$$

with r_i and c_j as row and column effects respectively. Note that this functions treat the row effect as the parameter to be constrained using sum to zero (for `rcModelPLM` and `rcModelWPLM`) or median of zero (for `rcModelMedianPolish`).

The `rcModelPLM` and `rcModelWPLM` functions use a robust linear model procedure for fitting the model.

The function `rcModelMedianPolish` uses the median polish algorithm.

Value

A list with following items:

`Estimates`	The parameter estimates. Stored in column effect then row effect order
`Weights`	The final weights used
`Residuals`	The residuals
`StdErrors`	Standard error estimates. Stored in column effect then row effect order
`Scale`	Scale Estimates

Author(s)

B. M. Bolstad <bmb@bmbolstad.com>

See Also

rcModelPLM

Examples

```
y <- matrix(c(10+rnorm(50),20+rnorm(50)),20,5,byrow=TRUE)

subrcModelPLM(y,c(rep(1,10),rep(2,10)))
subrcModelMedianPolish(y,c(rep(1,10),rep(2,10)))



col.effects <- c(10,11,10.5,12,9.5)
row.effects <- c(seq(-0.5,-0.1,by=0.1),seq(0.1,0.5,by=0.1))


y <- outer(row.effects, col.effects,"+")
w <- runif(50)

rcModelPLM(y)
rcModelWPLM(y, w)
rcModelMedianPolish(y)
```

```
y <- y + rnorm(50)

rcModelPLM(y)
rcModelWPLM(y, w)
rcModelMedianPolish(y)



rcModelPLM(y,row.effects=row.effects)
rcModelWPLM(y,w,row.effects=row.effects)

rcModelPLM(y,input.scale=1.0)
rcModelWPLM(y, w,input.scale=1.0)
rcModelPLM(y,row.effects=row.effects,input.scale=1.0)
rcModelWPLM(y,w,row.effects=row.effects,input.scale=1.0)
```

Index

∗ **manip**
normalize.quantiles, 3
normalize.quantiles.in.blocks, 4
normalize.quantiles.robust, 5
normalize.quantiles.target, 7
rma.background.correct, 13
∗ **models**
rcModelPLMd, 8
rcModelPLMr, 9
rcModels, 11
subrcModels, 15
∗ **univar**
colSumamrize, 2
subColSummarize, 14

colSumamrize, 2
colSummarizeAvg *(colSumamrize)*, 2
colSummarizeAvgLog *(colSumamrize)*, 2
colSummarizeBiweight *(colSumamrize)*, 2
colSummarizeBiweightLog *(colSumamrize)*, 2
colSummarizeLogAvg *(colSumamrize)*, 2
colSummarizeLogMedian *(colSumamrize)*, 2
colSummarizeMedian *(colSumamrize)*, 2
colSummarizeMedianLog *(colSumamrize)*, 2
colSummarizeMedianpolish *(colSumamrize)*, 2
colSummarizeMedianpolishLog *(colSumamrize)*, 2
convert.group.labels *(subColSummarize)*, 14

expresso, *4*, *7*

matrix, *14*, *15*

normalize.AffyBatch.quantiles.robust *(normalize.quantiles.robust)*, 5
normalize.quantiles, 3, *5*, *6*, *8*
normalize.quantiles.determine.target *(normalize.quantiles.target)*, 7
normalize.quantiles.in.blocks, 4
normalize.quantiles.robust, *4*, 5
normalize.quantiles.target, 7
normalize.quantiles.use.target *(normalize.quantiles.target)*, 7

rcModelMedianPolish *(rcModels)*, 11
rcModelPLM, *8*, *10*, *16*
rcModelPLM *(rcModels)*, 11
rcModelPLMd, 8, *10*, *12*
rcModelPLMr, *8*, 9, *12*
rcModelPLMrc *(rcModelPLMr)*, 9
rcModelPLMrr *(rcModelPLMr)*, 9
rcModels, 11
rcModelWPLM *(rcModels)*, 11
rcModelWPLMr *(rcModelPLMr)*, 9
rcModelWPLMrc *(rcModelPLMr)*, 9
rcModelWPLMrr *(rcModelPLMr)*, 9
rma, *4*, *7*
rma.background.correct, 13

subColSummarize, 14
subColSummarizeAvg *(subColSummarize)*, 14
subColSummarizeAvgLog *(subColSummarize)*, 14
subColSummarizeBiweight *(subColSummarize)*, 14
subColSummarizeBiweightLog *(subColSummarize)*, 14
subColSummarizeLogAvg *(subColSummarize)*, 14
subColSummarizeLogMedian *(subColSummarize)*, 14
subColSummarizeMedian *(subColSummarize)*, 14
subColSummarizeMedianLog *(subColSummarize)*, 14
subColSummarizeMedianpolish *(subColSummarize)*, 14
subColSummarizeMedianpolishLog *(subColSummarize)*, 14
subrcModelMedianPolish *(subrcModels)*, 15
subrcModelPLM *(subrcModels)*, 15
subrcModels, 15
subrcModelWPLM *(subrcModels)*, 15