Package 'preprocessCore'

November 25, 2026

Version 1.76.0

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_24

git_last_commit e8acdbb

git_last_commit_date 2026-11-23

Repository Bioconductor 3.24

Date/Publication 2026-11-24

Contents

colSummarize . . . 2
normalize.quantiles . . . 4
normalize.quantiles.in.blocks . . . 5
normalize.quantiles.robust . . . 6
normalize.quantiles.target . . . 9
rcModelPLMd . . . 10
rcModelPLMr . . . 12
rcModels . . . 15
rma.background.correct . . . 19
subColSummarize . . . 20
subrcModels . . . 22

Index **25**

colSummarize *Summarize the columns 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. Each column represents an independent group or sample and each row represents an observation or measurement within that group. The matrix should contain at least 1 row and 1 column. Values must be numeric; NA and NaN values will be propagated to the results.

Details

This group of functions summarize the columns of a given matrix. Note that colSummarizeLogMedian and colSummarizeMedianLog return NA for standard errors, as SE is not computed for these methods.

colSummarizeAvg Take means in a column-wise manner.

colSummarizeAvgLog log2 transform the data and then take means in a column-wise manner.

colSummarizeBiweight Summarize each column using a one-step Tukey Biweight procedure.

colSummarizeBiweightLog log2 transform the data and then summarize each column using a one-step Tukey 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.

colSummarizeMedianLog log2 transform the data and then summarize each column using the median.

colSummarizeMedianpolish Use median polish to summarize each column while also accounting for a row effect (not returned).

colSummarizeMedianpolishLog log2 transform the data and then use median polish to summarize each column while also accounting for a row effect (not returned).

Value

A named list with two elements:

`Estimates`	A numeric vector of length `ncol(y)` containing the summary value for each column.
`StdErrors`	A numeric vector of length `ncol(y)` containing the standard error estimate for each column. Note that `colSummarizeLogMedian` and `colSummarizeMedianLog` return `NA` for all elements of this vector.

Choosing a Summarization Method

The choice of method depends on your data characteristics and analysis goals:

Simple methods (Avg, Median) Use for clean data without significant outliers. `colSummarizeAvg` is suitable for approximately normally distributed data, while `colSummarizeMedian` is more robust to skewed distributions and outliers. Both return results on the original data scale.

Log-transformed methods (AvgLog, LogAvg, MedianLog, LogMedian) Use when data exhibits multiplicative noise or exponential behavior (common in biological measurements). `colSummarizeAvgLog` and `colSummarizeLogAvg` are mathematically related through logarithm properties; `colSummarizeAvgLog` is often preferred for right-skewed data. Note that `colSummarizeLogMedian` and `colSummarizeMedianLog` do not return standard error estimates.

Robust methods (Biweight, Biweight with log) Use when you expect outliers or want resistance to extreme values. The Tukey Biweight method provides robust estimation with less influence from outliers compared to means. Choose `colSummarizeBiweightLog` if your data is best modeled on a log scale.

Median polish methods (Medianpolish, Medianpolish with log) Use when you have a two-way structure (both row and column effects) and want to account for systematic differences between rows. This method simultaneously estimates column effects (returned as Estimates) while accounting for row effects.

Author(s)

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

See Also

`subColSummarize` for summarizing subsets of columns; see also `subColSummarizeAvg` and `subColSummarizeMedian` for grouped analogues of the main summarization functions.

Examples

```
# Set random seed for reproducible examples
set.seed(1234)

# Create sample data: 20 observations per column, 5 columns
y <- matrix(10 + rnorm(100), 20, 5)

# Simple methods - returns mean and standard error for each column
colSummarizeAvg(y)
colSummarizeMedian(y)

# Log-transformed average methods (useful for right-skewed data)
colSummarizeAvgLog(y)
```

```
colSummarizeLogAvg(y)

# Log-transformed median methods (no standard errors returned)
colSummarizeLogMedian(y)
colSummarizeMedianLog(y)

# Robust methods - resistant to outliers using Tukey Biweight
colSummarizeBiweight(y)
colSummarizeBiweightLog(y)

# Median polish methods - accounts for both column and row effects
colSummarizeMedianpolish(y)
colSummarizeMedianpolishLog(y)

# Example of extracting results
result <- colSummarizeAvg(y)
head(result$Estimates)    # Column summaries
head(result$StdErrors)    # Standard errors
```

`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, <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.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

Normalize columns of a numeric matrix by robustly estimating a common quantile distribution. Unlike standard quantile normalization, columns can be assigned weights or excluded from computation of the normalization distribution.

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 numeric matrix, with rows representing probes or features and columns representing chips or samples. Missing values are not supported; the matrix should contain no NA or infinite values.
copy	Logical. If TRUE, normalize a copy of x and leave the input matrix unchanged. If FALSE, normalize the matrix in place to reduce memory use.
weights	Controls construction of the normalization distribution. It can be a numeric vector of length ncol(x) containing non-negative weights for the columns, or "huber" to use Huber M-estimation at each quantile with tuning constant 1.345. If NULL, weights are generated from remove.extreme and n.remove.
remove.extreme	Character string specifying which columns are assigned zero weight when weights is NULL. "variance" removes columns with the most extreme variance, "mean" removes columns whose mean is most different from the other columns, "both" applies both checks, and "none" assigns equal weight to every column. Ignored when weights is supplied.
n.remove	Number of extreme columns to remove for each selected criterion. Must be a non-negative integer.
use.median	Logical. If TRUE, use the median of the sorted values at each rank as the normalization distribution. If FALSE, construct a weighted mean or Huber estimate, depending on weights.
use.log2	Logical. If TRUE, estimate the normalization distribution on the log2 scale and transform it back to the original scale. This produces a geometric rather than arithmetic mean for the weighted normalization. Has no effect when use.median = TRUE.
keep.names	Logical. If TRUE, assign the row and column names of x to the normalized matrix.

Details

Robust quantile normalization generalizes standard quantile normalization by reducing the influence of unusually variable or unusually shifted columns when estimating the common reference distribution. For each rank, sorted values from the columns are combined according to the requested weighting scheme. Every normalized column is then replaced with values from this common distribution in rank order.

By default, remove.extreme is used to identify columns that should not contribute to the reference distribution. With numeric weights, positive values give columns greater influence, and zero values exclude columns from computation of the reference distribution. Supplying "huber" uses robust Huber estimation at each rank instead of fixed column weights.

With use.median = TRUE, the median target is computed independently of weights; this option takes precedence over the weighted mean and Huber options. Log2 processing estimates either the weighted mean or Huber location on the log2 scale before transforming the target distribution back to the intensity scale.

The implementation does not support missing values. In particular, NA handling differs from normalize.quantiles; do not use this routine unless all entries of x are observed and finite.

If copy = FALSE, the input matrix is normalized in place and the returned matrix may be the same R object as x. Use the default copy = TRUE when the original data must be preserved.

Value

A numeric matrix with the same dimensions as `x`. When `keep.names = TRUE`, row and column names are preserved from `x`.

Note

This function is experimental. The extreme-column selection and robust target estimates can be sensitive to small sample sizes.

Author(s)

Ben Bolstad, <bmb@bmbolstad.com>

References

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), 185–193.

See Also

`normalize.quantiles`

Examples

```
set.seed(1234)

# Create positive intensity-like data with one clearly unusual column.
x <- matrix(rgamma(100, shape = 2, rate = 1), nrow = 20, ncol = 5)
x[, 5] <- x[, 5] * 3

# Assign names before demonstrating the keep.names option.
rownames(x) <- paste0("feature", seq_len(nrow(x)))
colnames(x) <- paste0("sample", seq_len(ncol(x)))

# By default, the function automatically gives zero weight to an extreme
# column selected using the requested remove.extreme criterion.
y <- normalize.quantiles.robust(x, remove.extreme = "variance")
stopifnot(all(dim(y) == dim(x)))

# Supply fixed non-negative column weights. This gives the fifth column only
# half as much influence as the other columns in the reference distribution.
weights <- c(1, 1, 1, 1, 0.5)
y.weights <- normalize.quantiles.robust(x, weights = weights)

# Huber weighting applies robust weights row by row when forming the target
# distribution. The tuning constant is fixed internally at 1.345.
y.huber <- normalize.quantiles.robust(x, weights = "huber")

# Median normalization uses the median value at each rank and does not use
# the fixed weights supplied through the weights argument.
y.median <- normalize.quantiles.robust(x, weights = weights,
                                       use.median = TRUE)

# Fit the robust target distribution on the log2 scale and transform it back
# to the intensity scale.
```

```
y.log2 <- normalize.quantiles.robust(x, weights = "huber", use.log2 = TRUE)

# Preserve the input dimnames in the returned matrix.
y.named <- normalize.quantiles.robust(x, keep.names = TRUE)
identical(colnames(y.named), colnames(x))

# copy = FALSE normalizes x in place. This option is useful for memory savings
# when the original matrix is not needed.
x.in.place <- x
y.in.place <- normalize.quantiles.robust(x.in.place, copy = FALSE)
identical(y.in.place, x.in.place)
```

`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 row-column models with group-specific row effects*

Description

These functions fit additive row-column decomposition models in which the row effects are allowed to vary by predefined groups of arrays. This is useful when a subset of probes behaves differently across blocks, batches, or other sample groups, and a common row effect is not adequate for all arrays. The procedure automatically checks whether selected rows show evidence of group- dependent behavior and, if so, splits the row effect into separate values for each group.

Usage

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

Arguments

`y`	A numeric matrix with rows representing probes/features and columns representing arrays/samples. The model is $$y_{ij} = r_i + c_j + \epsilon_{ij}$$, where r_i is the row effect and c_j is the column effect.
`group.labels`	A vector or factor specifying the group label for each column of `y`. Its length must be `ncol(y)`. Each group must contain at least two arrays. The procedure tests whether row effects differ across these groups and, where appropriate, splits a row effect into group- specific effects.

Details

The ordinary PLM model fits a single shared row effect for all arrays in the matrix, as in

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

. The PLM-d procedure generalizes this model by allowing some row effects to vary across group-defined blocks. Thus, for a row that appears to behave differently in one group relative to another, the model estimates separate row effects for each group instead of a single common value.

The implementation in the C code follows the idea described in the PLM extension manuscript: the method first fits a standard robust row-column model, then examines the residuals for each row to

see whether they are associated with the grouping variable. If a row shows evidence of a group-specific departure, that row is split and fitted with separate row effects for the different groups. The process repeats until no further group-dependent row effects are detected.

This differs from the robust row/column weighting methods in `rcModelPLMr`. The PLM-r family addresses outlying rows or columns by downweighting them during the fit. The PLM-d family instead asks whether the underlying row effect itself differs across groups and, if so, estimates a separate effect within each group.

Value

A list with the following components:

Estimates The fitted parameter estimates. These include the column effects and the row effects, with additional group-specific row effects for any rows whose effects are split across the groups. The exact order is the same as the fitted design matrix: column effects first, followed by the row effects and any split terms needed for the group-specific rows.

Weights The final observation weights used during the robust fit.

Residuals A matrix of residuals after subtracting the fitted row and column effects.

StdErrors The standard errors associated with the fitted parameters, in the same order as `Estimates`.

WasSplit An integer vector of length `nrow(y)` indicating which rows were split into separate effects across groups. A value of 1 means that the row effect was allowed to differ across the specified groups; 0 means it remained shared across all groups.

Model Overview

The PLM-d model retains the additive form

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

but allows selected rows to be split into group-specific effects. In practice, this is useful when a subset of probes has a different systematic offset in one biological or experimental group than in another. Rows that do not show such a difference keep a single shared effect across all groups.

Choosing a Method

`rcModelPLM` Use this for a standard robust decomposition with one shared row effect across all arrays. This is the simplest baseline model when no grouping structure is expected.

`rcModelPLMr` Use this when the main issue is rows or columns that are unusually noisy or outlying. These methods downweight such rows or columns rather than splitting the row effect by group.

`rcModelPLMd` Use this when you have a known grouping variable and suspect that only some rows behave differently between groups. This method tests for and fits group-specific row effects only where the data support them.

Comparison to ordinary PLM and PLM-r models

`rcModelPLM` Fits a common row effect across all columns. There is no notion of group-specific row effects.

`rcModelPLMr` Adds row and/or column robustness by downweighting outlying rows or columns. It does not split row effects by group labels.

`rcModelPLMd` Retains the additive structure but allows selected probes to have separate effects in different groups when the data support that structure. This is a group-wise model extension rather than a weighting-based robustification.

Author(s)

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

References

Bolstad, B. M. PLM Extensions Manuscript. https://bmbolstad.com/stuff/PLM-Extensions-Manuscript.pdf

See Also

rcModelPLM,rcModelPLMr,rcModelPLMrr,rcModelPLMrc

Examples

```
set.seed(1234)

# Example 1: two groups with a row-level effect that differs between them.
# This matches the intended PLM-d use case: the algorithm should detect a
# group-specific row effect and split it.
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, sd = 0.25)

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

fit <- rcModelPLMd(y, group.labels = factor(c(1, 1, 1, 2, 2, 2)))
fit$WasSplit
fit$Estimates

# Example 2: the same design without any group-specific row shift.
# This checks the non-split case, where all row effects remain common across groups.
row.effects <- c(seq(-0.5, -0.1, by = 0.1), seq(0.1, 0.5, by = 0.1))
y.clean <- outer(row.effects, col.effects, "+") + rnorm(60, sd = 0.1)
fit.clean <- rcModelPLMd(y.clean, group.labels = factor(c(1, 1, 1, 2, 2, 2)))
fit.clean$WasSplit

# Compare the residual structure with ordinary PLM.
par(mfrow = c(2, 2))
matplot(y, type = "l", col = c(rep("red", 3), rep("blue", 3)),
        ylab = "signal", xlab = "probe", main = "Observed data")
matplot(rcModelPLM(y)$Residuals, col = c(rep("red", 3), rep("blue", 3)),
        ylab = "residuals", xlab = "probe", main = "Residuals (PLM)")
matplot(fit$Residuals, col = c(rep("red", 3), rep("blue", 3)),
        ylab = "residuals", xlab = "probe", main = "Residuals (PLM-d)")
```

---

rcModelPLMr *Fit row-column models with row and/or column robustness*

Description

These functions fit additive row-column decomposition models while adding an additional layer of robustness beyond the standard `rcModelPLM` procedure. They are useful when systematic problems affect entire rows or columns, rather than only isolated elements in the matrix. The methods differ in which direction of robustification is used: both row and column effects, row effects only, or column effects only.

Usage

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

Arguments

`y`	A numeric matrix with rows representing features or probes and columns representing arrays or samples. The model is $$y_{ij} = r_i + c_j + \epsilon_{ij}$$, where r_i denotes the row effect and c_j the column effect.
`w`	For the weighted variants `rcModelWPLMr`, `rcModelWPLMrr`, and `rcModelWPLMrc`, a matrix or vector of non-negative observation weights. If a vector is supplied, it must have length `prod(dim(y))`; if a matrix is supplied, it must have the same dimensions as `y`. Zero weights remove an observation from the fit, while larger weights give that observation more influence.

Details

These functions fit the row-column model

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

with row and column effects constrained so that the row effects are treated as the identifiable component. The robust fitting is based on the same Huber M-estimation framework as `rcModelPLM`, with tuning constant $k = 1.345$, but in addition the methods estimate row and/or column robustness weights to downweight systematic row-level or column-level departures from the additive structure.

The three variants differ in how this extra robustness is applied:

`rcModelPLMr` / `rcModelWPLMr` Fit the PLM-r procedure, which adds both row and column robustness to the standard element-wise robust fit. This is appropriate when outlying behavior may occur in either direction, affecting entire rows or columns of the matrix.

`rcModelPLMrr` / `rcModelWPLMrr` Fit the PLM-rr procedure, which adds row robustness only. This is useful when a small number of rows are systematically discrepant but the columns are not expected to require special handling.

`rcModelPLMrc` / `rcModelWPLMrc` Fit the PLM-rc procedure, which adds column robustness only. This is useful when a small number of columns are systematically discrepant but row-specific problems are not the main concern.

In practice, these methods are most helpful when the data contain structured outliers or patterns of contamination that cannot be well represented by the ordinary element-wise robust fit alone. The robust weights are designed to identify and reduce the influence of rows or columns that deviate from the common additive pattern.

Value

A list with the following components:

Estimates A numeric vector of length `nrow(y) + ncol(y)` containing the fitted row and column effects. The values are stored in column-effect then row-effect order: [column effect 1, ..., column effect p, row effect 1, ..., row effect n].

Weights A numeric matrix with the same dimensions as `y`. Its entries are the final observation weights used in the fit; values near 0 indicate downweighting of outlying observations.

Residuals A numeric matrix of residuals after subtracting the fitted additive row and column effects.

StdErrors A numeric vector of standard errors for the fitted effects, ordered in the same way as `Estimates`.

Comparison to the ordinary PLM functions

The functions `rcModelPLM` and `rcModelWPLM` use a robust linear model fit with a single robust weighting step at the element level. The PLM-r family extends this idea by adding an additional row/column weighting layer. As a result, the PLM-r variants are generally more appropriate when the problem is not just a few isolated outlying cells but broader row-wise or column-wise departures from the additive form.

Comparison to `rcModelPLMd`

The function `rcModelPLMd` handles a different kind of structure: it tests whether row effects differ across groups and allows a row effect to be split by group labels. In contrast, the PLM-r variants do not split effects by group labels; instead they identify rows or columns whose residual behavior is inconsistent with the overall additive pattern and downweight them during the fit.

Author(s)

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

References

Bolstad, B. M. PLM Extensions Manuscript. https://bmbolstad.com/stuff/PLM-Extensions-Manuscript.pdf

See Also

`rcModelPLM`,`rcModelWPLM`,`rcModelPLMd`

Examples

```
set.seed(1234)

# Create a clean matrix with additive row and column effects.
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, "+") + rnorm(50, sd = 0.1)

# The usual row-column fit with row/column robustness.
fit <- rcModelPLMr(y)
fit$Estimates
fit$Weights[1:3, 1:3]

# A row outlier can be handled especially well by the row-robust fit.
y.row <- y
y.row[1, ] <- y.row[1, ] + 3
fit.r <- rcModelPLMr(y.row)
fit.rr <- rcModelPLMrr(y.row)
fit.rc <- rcModelPLMrc(y.row)

# The row-robust variant should reduce the impact of the anomalous row.
summary(c(fit.r$Weights))
summary(c(fit.rr$Weights))
summary(c(fit.rc$Weights))

# A column outlier can be handled especially well by the column-robust fit.
y.col <- y
y.col[, 4] <- y.col[, 4] + 3
fit.r.col <- rcModelPLMr(y.col)
fit.rr.col <- rcModelPLMrr(y.col)
fit.rc.col <- rcModelPLMrc(y.col)

# The column-robust variant should reduce the impact of the anomalous column.
summary(c(fit.r.col$Weights))
summary(c(fit.rr.col$Weights))
summary(c(fit.rc.col$Weights))

# Visualize the final weights for the row-outlier case.
par(mfrow = c(2, 2))
image(1:nrow(y.row), 1:ncol(y.row), fit.r$Weights,
      xlab = "row", ylab = "column", main = "PLM-r", zlim = c(0, 1))
image(1:nrow(y.row), 1:ncol(y.row), fit.rr$Weights,
      xlab = "row", ylab = "column", main = "PLM-rr", zlim = c(0, 1))
image(1:nrow(y.row), 1:ncol(y.row), fit.rc$Weights,
      xlab = "row", ylab = "column", main = "PLM-rc", zlim = c(0, 1))
matplot(y.row, type = "l", main = "Data with a row outlier")

# Weighted variants allow prior knowledge to be incorporated.
w <- rep(1, length(y))
fit.w <- rcModelWPLMr(y, w)
fit.w$Estimates
```

`rcModels` *Fit additive row-column decomposition models*

Description

Decompose a matrix into additive row and column effects plus residuals. These functions fit row-column models useful for extracting systematic patterns and identifying systematic biases in data (e.g., probe effects and sample effects in microarray data). Three fitting methods are provided: robust linear models with and without observation weights, and median polish for a non-parametric alternative.

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 with rows representing probes/features and columns representing arrays/samples. Used by all three functions to fit the row-column model.
w	(Only for rcModelWPLM) A matrix or vector of non-negative weights for each observation. If a vector, must have length prod(dim(y)) (i.e., total number of elements). If a matrix, must have the same dimensions as y. Zero weights completely exclude observations from the fit; larger weights increase an observation's influence on estimation.
row.effects	(Only for rcModelPLM and rcModelWPLM) A vector of pre-computed row effects to use instead of fitting them. Must have length nrow(y) and must sum to exactly zero (within machine precision). When supplied, the function fixes these row effects and estimates only column effects, analyzing each column separately. Useful for applying probe effects from a reference dataset to new data or for sensitivity analysis. If NULL (default), row effects are estimated from the data.
input.scale	(Only for rcModelPLM and rcModelWPLM) A pre-computed scale (residual standard deviation) estimate to use instead of estimating it from the data. Must be positive. Useful for consistency across multiple analyses or when external scale estimates are available. If NULL (default), scale is estimated from the residuals using a robust method.

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 the following components. Components are named and can be accessed via result$Estimates, result$Residuals, etc.

Estimates A numeric vector of length nrow(y) + ncol(y) containing the fitted row and column effects. Elements are ordered as: [column effect 1, ..., column effect p, row effect 1, ..., row effect n] where n = nrow(y) and p = ncol(y). These are the constrained parameters (sum to zero for rcModelPLM/rcModelWPLM; median = 0 for rcModelMedianPolish).

Weights For rcModelPLM and rcModelWPLM: A numeric matrix (n × p) of final weights used in fitting, where weights < 1 indicate downweighting of outliers. For rcModelMedianPolish: NULL (weights not computed for median polish).

Residuals A numeric matrix (n × p) of residuals after subtracting fitted effects from the data. For a good fit, residuals should appear random with no systematic patterns.

StdErrors For rcModelPLM and rcModelWPLM: A numeric vector of length `nrow(y) + ncol(y)` containing standard errors for each parameter estimate, in the same order as Estimates. Larger values indicate less precise estimation. For rcModelMedianPolish: `NULL` (standard errors not computed; median polish is non-parametric and does not provide SEs).

Scale For rcModelPLM and rcModelWPLM: A numeric scalar (length 1) containing the estimated scale parameter (residual standard deviation). Indicates the overall noise level in the data. For rcModelMedianPolish: This component is not returned (median polish does not estimate scale).

Model Overview

These functions decompose a matrix into additive row and column effects plus residuals using the linear model:

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

This decomposition is useful for:

Detecting systematic patterns Identify whether variation in your data is systematic (captured by row/column effects) or random (residuals).

Quality assessment Row effects represent probe/feature-specific biases; column effects represent array/sample-specific effects. Unusual values can indicate quality issues.

Normalization and preprocessing Understanding and removing systematic effects before downstream analysis.

Comparative analysis Compare the magnitudes of row vs. column effects to understand whether variation is more probe-specific or sample-specific.

Choosing a Method

rcModelPLM (Robust Linear Model) Use this for standard row-column decomposition. It employs Huber's robust M-estimation procedure (tuning constant k=1.345) that downweights outliers. Good for data with occasional extreme values. Estimates row effects that sum to exactly zero (identifiability constraint). Returns scale estimate and standard errors.

rcModelWPLM (Weighted Robust Linear Model) Use this when you have prior weights for each observation (e.g., quality weights, known measurement uncertainties). Weights must be non-negative; zero weights exclude observations entirely. Uses the same Huber's M-estimation procedure as rcModelPLM but incorporates the weights. Useful for combining data from multiple sources with different reliabilities.

rcModelMedianPolish Use this for a non-parametric alternative to RLM. More resistant to outliers than even robust RLM, and makes fewer distributional assumptions. Iteratively refines row and column medians. Constraints row effects to have median equal to zero. Does not return scale estimate or standard errors. Simpler interpretation but less statistical efficiency than RLM when data is approximately normal.

Using Pre-Computed Row Effects

You can provide pre-computed row effects (e.g., from a reference dataset) using the `row.effects` parameter. When supplied:

Constraints Row effects must sum to exactly zero (within machine precision). If this constraint is violated, the function will error.

Behavior The model fits column effects given the fixed row effects, and analyzes each column separately. This is useful for ensuring consistency across multiple analyses.

Use cases Applying probe effects from a reference or training dataset to new data; sensitivity analysis; two-stage model fitting.

Interpreting the Output

The output is a list with components:

Estimates Concatenated vector of column effects followed by row effects. This order is chosen for historical reasons and model fitting efficiency.

StdErrors Standard errors for each parameter in the same order as Estimates. Indicates precision of each effect estimate. Only returned by rcModelPLM and rcModelWPLM (rcModelMedianPolish returns NULL).

Scale Estimated scale parameter (residual variability). Only returned by rcModelPLM and rcModelWPLM. Useful for understanding noise level in data.

Residuals The residuals e_ij after subtracting estimated effects. Should appear random if model fits well.

Weights The final observation weights used in fitting. For rcModelPLM, these are computed based on residuals; for rcModelWPLM, combines input weights with residual-based weights.

Related Functions

`rcModelPLMr` and `rcModelPLMd` are related variants that fit row-column models with additional row-specific constraints. These functions constrain row effects to be equal across all columns (rcModelPLMr) or use row-specific digestion effects (rcModelPLMd), respectively. See their documentation for details on when to use these variants versus the standard rcModel functions.

Author(s)

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

See Also

`rcModelPLMr`,`rcModelPLMd`

Examples

```
# Set random seed for reproducible examples
set.seed(1234)

# Create sample data: 10 row effects (probes) and 5 column effects (arrays)
row.effects <- c(seq(-0.5, -0.1, by = 0.1), seq(0.1, 0.5, by = 0.1))
col.effects <- c(10, 11, 10.5, 12, 9.5)

# Perfect model: no noise
y.perfect <- outer(row.effects, col.effects, "+")

# Noisy data: add random noise to model
y <- y.perfect + rnorm(50)

# Create weights for weighted model (optional)
w <- runif(50)
```

```
# Basic usage: fit all three methods without additional parameters
rcModelPLM(y)
rcModelWPLM(y, w)
rcModelMedianPolish(y)

# Using pre-computed row effects (e.g., from a reference dataset)
# When row effects are supplied, only column effects are estimated
rcModelPLM(y, row.effects = row.effects)
rcModelWPLM(y, w, row.effects = row.effects)

# Using a fixed input scale parameter
rcModelPLM(y, input.scale = 1.0)
rcModelWPLM(y, w, input.scale = 1.0)

# Combining multiple options: fixed row effects AND fixed scale
rcModelPLM(y, row.effects = row.effects, input.scale = 1.0)
rcModelWPLM(y, w, row.effects = row.effects, input.scale = 1.0)

# Examine the output structure
result <- rcModelPLM(y)
str(result)  # View the structure
result$Estimates  # Extract column and row effects (column effects first)
dim(result$Residuals)  # Residuals are same dimension as input
```

`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, <bmb@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 matrix columns by row groups*

Description

These functions partition the rows of a matrix into groups (specified via `group.labels`) and then summarize each column independently within each group. They use the same summarization methods as `colSummarize` (mean, median, Tukey Biweight, median polish, and log-transformed variants) but apply them to subsets of rows. The result is a matrix where each row represents the summarized values for one group across all columns.

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 where each column represents an independent group or sample and each row represents an observation or measurement within that group. The matrix should contain at least 1 row and 1 column.
`group.labels`	A vector of length `nrow(y)` that assigns each row to a group. Can be any atomic vector (numeric, character, logical) that will be converted to a factor. Rows with `NA` values in `group.labels` will be excluded from summarization. The names of the resulting groups will be taken from the factor level names.

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.

`subColSummarizeAvg` Summarize by taking the mean.

`subColSummarizeAvgLog` `log2` transform the data and then take the mean in a column-wise manner.

`subColSummarizeBiweight` Use a one-step Tukey Biweight to summarize columns.

`subColSummarizeBiweightLog` `log2` transform the data and then use a one-step Tukey Biweight to summarize columns.

`subColSummarizeLogAvg` Summarize by taking the mean and then applying `log2`.

`subColSummarizeLogMedian` Summarize by taking the median and then applying `log2`.

`subColSummarizeMedian` Summarize by taking the median.

`subColSummarizeMedianLog` `log2` transform the data and then summarize by taking the median.

`subColSummarizeMedianpolish` Use median polish to summarize each column while also accounting for a row effect (not returned).

`subColSummarizeMedianpolishLog` `log2` transform the data and then use median polish to summarize each column while also accounting for a row effect (not returned).

Value

A numeric matrix with dimensions: number of groups × number of columns. Each row corresponds to one group (with row names from the factor levels of `group.labels`), and each column corresponds to a column from the input matrix. The matrix contains the summarized values for each group and column combination.

When to Use subColSummarize

Use `subColSummarize*` functions when your data has a natural grouping structure within the rows, and you want to compute summary statistics separately for each group. Common use cases include:

Stratified analysis When you have multiple samples/replicates within each treatment group or experimental condition, use these functions to summarize by condition rather than across all rows at once.

Quality control grouping When rows represent different probe sets, detection methods, or measurement batches, group them to compute batch-specific summaries and identify systematic differences between groups.

Multi-level experiments When your data represents a hierarchical structure (e.g., multiple technical replicates per biological sample), group by the higher level (biological sample) to obtain summary values accounting for within-group variation.

Robust comparison Use with robust methods (Biweight, median polish) to compare groups while being resistant to outliers within each group.

Example: If you have expression data with 3 probes × 4 samples where the probes are grouped as {probe1, probe2, probe1, probe2}, you could group by probe to get summary expression values per probe per sample, effectively combining replicates from the same probe.

Author(s)

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

See Also

`colSummarize` for summarizing entire columns without grouping; see also `colSummarizeAvg` and `colSummarizeMedian` for the ungrouped versions of the same summary statistics.

Examples

```
# Set random seed for reproducible examples
set.seed(1234)

# Create sample data: 20 rows (10 in group 1, 10 in group 2) and 5 columns
y <- matrix(c(10 + rnorm(50), 20 + rnorm(50)), 20, 5, byrow = TRUE)
groups <- c(rep(1, 10), rep(2, 10))

# Simple methods - returns a matrix with one row per group
subColSummarizeAvg(y, groups)
subColSummarizeMedian(y, groups)

# Log-transformed average methods
subColSummarizeAvgLog(y, groups)
subColSummarizeLogAvg(y, groups)

# Log-transformed median methods
subColSummarizeLogMedian(y, groups)
subColSummarizeMedianLog(y, groups)

# Robust methods - resistant to outliers within groups
subColSummarizeBiweight(y, groups)
subColSummarizeBiweightLog(y, groups)

# Median polish methods - accounts for group and column effects
subColSummarizeMedianpolish(y, groups)
subColSummarizeMedianpolishLog(y, groups)

# Examine the output: a matrix with 2 rows (one per group) and 5 columns
result <- subColSummarizeAvg(y, groups)
dim(result)           # 2 groups × 5 columns
rownames(result)      # Group names
```

---

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, 4
normalize.quantiles.in.blocks, 5
normalize.quantiles.robust, 6
normalize.quantiles.target, 9
rma.background.correct, 19
∗ **models**
rcModelPLMd, 10
rcModelPLMr, 12
rcModels, 15
subrcModels, 22
∗ **univar**
colSummarize, 2
subColSummarize, 20

colSummarize, 2
colSummarizeAvg, *21*
colSummarizeAvg *(*colSummarize*)*, 2
colSummarizeAvgLog *(*colSummarize*)*, 2
colSummarizeBiweight *(*colSummarize*)*, 2
colSummarizeBiweightLog *(*colSummarize*)*, 2
colSummarizeLogAvg *(*colSummarize*)*, 2
colSummarizeLogMedian *(*colSummarize*)*, 2
colSummarizeMedian, *21*
colSummarizeMedian *(*colSummarize*)*, 2
colSummarizeMedianLog *(*colSummarize*)*, 2
colSummarizeMedianpolish *(*colSummarize*)*, 2
colSummarizeMedianpolishLog *(*colSummarize*)*, 2
convert.group.labels *(*subColSummarize*)*, 20

expresso, *5*, *9*

normalize.AffyBatch.quantiles.robust *(*normalize.quantiles.robust*)*, 6
normalize.quantiles, 4, *6*, *8*, *10*
normalize.quantiles.determine.target *(*normalize.quantiles.target*)*, 9
normalize.quantiles.in.blocks, 5
normalize.quantiles.robust, *5*, 6
normalize.quantiles.target, 9
normalize.quantiles.use.target *(*normalize.quantiles.target*)*, 9

rcModelMedianPolish *(*rcModels*)*, 15
rcModelPLM, *11–14*, *23*
rcModelPLM *(*rcModels*)*, 15
rcModelPLMd, 10, *14*, *18*
rcModelPLMr, *11*, *12*, 12, *18*
rcModelPLMrc, *12*
rcModelPLMrc *(*rcModelPLMr*)*, 12
rcModelPLMrr, *12*
rcModelPLMrr *(*rcModelPLMr*)*, 12
rcModels, 15
rcModelWPLM, *14*
rcModelWPLM *(*rcModels*)*, 15
rcModelWPLMr *(*rcModelPLMr*)*, 12
rcModelWPLMrc *(*rcModelPLMr*)*, 12
rcModelWPLMrr *(*rcModelPLMr*)*, 12
rma, *5*, *9*
rma.background.correct, 19

subColSummarize, 20
subColSummarizeAvg, *3*
subColSummarizeAvg *(*subColSummarize*)*, 20
subColSummarizeAvgLog *(*subColSummarize*)*, 20
subColSummarizeBiweight *(*subColSummarize*)*, 20
subColSummarizeBiweightLog *(*subColSummarize*)*, 20
subColSummarizeLogAvg *(*subColSummarize*)*, 20
subColSummarizeLogMedian *(*subColSummarize*)*, 20
subColSummarizeMedian, *3*
subColSummarizeMedian *(*subColSummarize*)*, 20
subColSummarizeMedianLog *(*subColSummarize*)*, 20
subColSummarizeMedianpolish *(*subColSummarize*)*, 20
subColSummarizeMedianpolishLog *(*subColSummarize*)*, 20
subrcModelMedianPolish *(*subrcModels*)*, 22

subrcModelPLM *(*subrcModels*)*, 22
subrcModels, 22
subrcModelWPLM *(*subrcModels*)*, 22