

# Package ‘weित्रix’

April 11, 2024

**Title** Tools for matrices with precision weights, test and explore weighted or sparse data

**Version** 1.14.0

**Description** Data type and tools for working with matrices having precision weights and missing data. This package provides a common representation and tools that can be used with many types of high-throughput data. The meaning of the weights is compatible with usage in the base R function `lm` and the package `limma`. Calibrate weights to account for known predictors of precision. Find rows with excess variability. Perform differential testing and find rows with the largest confident differences. Find PCA-like components of variation even with many missing values, rotated so that individual components may be meaningfully interpreted. DelayedArray matrices and BiocParallel are supported.

**License** LGPL-2.1 | file LICENSE

**Encoding** UTF-8

**LazyData** true

**Depends** R (>= 3.6), SummarizedExperiment

**Imports** methods, utils, stats, grDevices, assertthat, S4Vectors, DelayedArray, DelayedMatrixStats, BiocParallel, BiocGenerics, limma, topconfects, dplyr, purrr, ggplot2, rlang, scales, reshape2, splines, Ckmeans.1d.dp, glm2, RnpcBLASctl

**Suggests** knitr, rmarkdown, BiocStyle, tidyverse, airway, edgeR, EnsDb.Hsapiens.v86, org.Sc.sgd.db, AnnotationDbi, ComplexHeatmap, patchwork, testthat (>= 2.1.0)

**RoxygenNote** 7.2.3

**VignetteBuilder** knitr

**biocViews** Software, DataRepresentation, DimensionReduction, GeneExpression, Transcriptomics, RNASeq, SingleCell, Regression

**git\_url** <https://git.bioconductor.org/packages/weित्रix>

**git\_branch** RELEASE\_3\_18

**git\_last\_commit** 2633120

**git\_last\_commit\_date** 2023-10-24

**Repository** Bioconductor 3.18

**Date/Publication** 2024-04-10

**Author** Paul Harrison [aut, cre] (<<https://orcid.org/0000-0002-3980-268X>>)

**Maintainer** Paul Harrison <paul.harrison@monash.edu>

## R topics documented:

as_weitrix . . . . .	2
bless_weitrix . . . . .	3
components_seq_scre . . . . .	4
counts_proportions . . . . .	5
counts_shift . . . . .	6
matrix_long . . . . .	7
simwei . . . . .	8
weitrix_calibrate . . . . .	8
weitrix_calibrate_all . . . . .	9
weitrix_calibrate_trend . . . . .	11
weitrix_calplot . . . . .	12
weitrix_components . . . . .	13
weitrix_confects . . . . .	16
weitrix_dispersions . . . . .	18
weitrix_elist . . . . .	18
weitrix_hill . . . . .	19
weitrix_randomize . . . . .	20
weitrix_sd_confects . . . . .	20
weitrix_weights . . . . .	22
weitrix_x . . . . .	23
well_knotted_spline . . . . .	23
<b>Index</b>	<b>25</b>

---

as_weitrix	<i>Convert data to a weitrix</i>
------------	----------------------------------

---

### Description

Ensure data is a weighted matrix or "weitrix". A weitrix is a SummarizedExperiment or subclass thereof with some metadata fields set. If it is ambiguous how to do this, produce an error.

### Usage

```
as_weitrix(object, weights = NULL)
```

### Arguments

object	Object to convert.
weights	Optional, weights matrix if not present in object.

## Details

Input can be a matrix or DelayedArray.

Input can be anything the limma package recognizes, notably the limma EList class (for example as output by voom or vooma).

If weights are not present in "object" and not given with "weights", they default for 0 for NA values and 1 for everything else.

## Value

A SummarizedExperiment object with metadata fields marking it as a weitrix.

## Examples

```
mat <- matrix(c(1,2,NA,3,NA,4), ncol=2)
weitrix <- as_weitrix(mat)

metadata(weitrix)
weitrix_x(weitrix)
weitrix_weights(weitrix)
```

---

bless\_weitrix

*Bless a SummarizedExperiment as a weitrix*

---

## Description

Set metadata entries in a SummarizedExperiment object so that it can be used as a weitrix.

## Usage

```
bless_weitrix(object, x_name, weights_name)
```

## Arguments

object	A SummarizedExperiment object.
x_name	Name of the assay containing the observations.
weights_name	Name of the assay containing the weights.

## Value

A SummarizedExperiment object with metadata fields marking it as a weitrix.

**Examples**

```

mat <- matrix(c(1,2,NA,3,NA,4), ncol=2)
weights <- matrix(c(1,0.5,0,2,0,1), ncol=2)
se <- SummarizedExperiment(assays=list(foo=mat, bar=weights))

weitrax <- bless_weitrax(se, "foo", "bar")

metadata(weitrax)
weitrax_x(weitrax)
weitrax_weights(weitrax)

```

---

components\_seq\_scee *Proportion more variance explained by adding components one at a time*

---

**Description**

Based on the output of components\_seq, work out how much further variance is explained by adding further components.

**Usage**

```

components_seq_scee(comp_seq, rand_comp = NULL)

components_seq_sceepplot(comp_seq, rand_comp = NULL)

```

**Arguments**

**comp\_seq** A list of Components objects, as produced by components\_seq.

**rand\_comp** Optional. A Components object with a single component. This should be based on a randomized version of the weitrax, for example as produced by weitrax\_components(weitrax\_randomize(my\_weitrax), p=1).

**Details**

If rand\_comp is given, some possible threshold levels for including further components are also calculated.

The "Parallel analysis" threshold is chosen based on variance explained by a single component in a randomized weitrax.

The "Optimistic" thresholds are chosen starting from the "Parallel Analysis" threshold. We view the Parallel Analysis threshold as indicating random variance is split amongst an effective number of samples, which will be somewhat smaller than the real number of samples. As each component is accepted, the pool of remaining variance is reduced by its contribution, and also the number of effective samples is reduced by one. The threshold is then the size of the remaining variance pool divided by the effective remaining number of samples. This is a somewhat ad-hoc method, but may indicate more components are justified than by criteria based on a flat threshold.

**Value**

components\_seq\_scree returns a data frame listing the variance explained by each further component.

components\_seq\_screeplot returns a ggplot2 plot object.

**Examples**

```
comp_seq <- weitrix_components_seq(simwei, 4, verbose=FALSE)
```

```
components_seq_scree(comp_seq)
components_seq_screeplot(comp_seq)
```

---

counts\_proportions      *Produce a weitrix of proportions within groups*

---

**Description**

Produce a weitrix of proportions between 0 and 1. The input is read counts at a collection of features in a collection of samples. The features need to be grouped, for example by gene. The proportions will add to 1 within each group.

**Usage**

```
counts_proportions(counts, grouping, verbose = TRUE, typecast = identity)
```

**Arguments**

counts	A matrix of read counts. Rows are peaks and columns are samples.
grouping	A data frame defining the grouping of features. Should have a column "group" naming the group and a column "name" naming the feature (corresponding to rownames(counts)).
verbose	If TRUE, output some debugging and progress information.
typecast	A function to convert a matrix to a matrix-like type. Applied at the chunk level, before all chunks are rbinded. Allows use of memory-efficient matrix representations.

**Value**

A SummarizedExperiment object with metadata fields marking it as a weitrix.

**Examples**

```
grouping <- data.frame(
  group=c("A","A","A","B","B"),
  name=c("p1","p2","p3","p4","p5"))

counts <- rbind(
  p1=c(1,2,0),
  p2=c(0,1,0),
  p3=c(1,0,0),
  p4=c(0,0,1),
  p5=c(0,2,1))

wei <- counts_proportions(counts, grouping)

weitrrix_x(wei)
weitrrix_weights(wei)
rowData(wei)
```

---

counts\_shift

*Produce a weitrrix of shift scores*


---

**Description**

Produce a weitrrix of shift scores between -1 and 1. The input is read counts at a collection of peaks (or other features) in a collection of samples. The peaks can be grouped by gene, and are ordered within each gene.

**Usage**

```
counts_shift(counts, grouping, verbose = TRUE, typecast = identity)
```

**Arguments**

counts	A matrix of read counts. Rows are peaks and columns are samples.
grouping	A data frame defining the grouping of peaks into genes. Should have a column "group" naming the gene and a column "name" naming the peak (corresponding to rownames(counts)). Within each group, peak names should be ordered from 5' to 3' position.
verbose	If TRUE, output some debugging and progress information.
typecast	A function to convert a matrix to a matrix-like type. Applied at the chunk level, before all chunks are rbinded. Allows use of memory-efficient matrix representations.

**Details**

For a particular gene, a shift score measures the tendency of reads to be upstrand (negative) or downstrand (positive) of the average over all samples. Shift scores range between -1 and 1.

**Value**

A SummarizedExperiment object with metadata fields marking it as a weitrrix.

**Examples**

```
grouping <- data.frame(
  group=c("A", "A", "A", "B", "B"),
  name=c("p1", "p2", "p3", "p4", "p5"))

counts <- rbind(
  p1=c(1,2,0),
  p2=c(0,1,0),
  p3=c(1,0,0),
  p4=c(0,0,1),
  p5=c(0,2,1))

wei <- counts_shift(counts, grouping)

weitrrix_x(wei)
weitrrix_weights(wei)
rowData(wei)
```

---

matrix\_long

---

*Convert a matrix to long form for ggplotting*


---

**Description**

A convenience function which melts the matrix and then joins row and column information.

**Usage**

```
matrix_long(
  matrix,
  row_info = NULL,
  col_info = NULL,
  varnames = c("name", "col")
)
```

**Arguments**

matrix	A matrix, or object that can be converted to a matrix.
row_info	Information about rows of the matrix. A data frame, or object that can be converted to a data frame.
col_info	Information about columns of the matrix. A data frame, or object that can be converted to a data frame.
varnames	Vector of two column names in the output, the first for the row and the second for the column.

**Value**

A data frame containing the matrix and associated information in long format.

**Examples**

```
matrix_long(weitrx(simwei), rowData(simwei), colData(simwei))
```

---

simwei	<i>Simulated weitr<sub>x</sub> dataset.</i>
--------	---

---

**Description**

This is a small simulated weitr<sub>x</sub> used in examples. There is one component of variation to be found, plus Gaussian noise with variance inversely proportional to the weights.

**Usage**

```
simwei
```

**Format**

A weitr<sub>x</sub> object.

---

weitr <sub>x</sub> _calibrate	<i>Adjust weights row-wise based on given row dispersions</i>
-------------------------------	---

---

**Description**

Based on estimated row dispersions, adjust weights in each row.

**Usage**

```
weitrx_calibrate(weitrx, dispersions)
```

**Arguments**

weitr <sub>x</sub>	A weitr <sub>x</sub> object, or an object that can be converted to a weitr <sub>x</sub> with <code>as_weitr<sub>x</sub></code> .
dispersions	A dispersion for each row.

**Details**

For large numbers of samples this can be based directly on `weitrx_dispersions`. For small numbers of samples, when using `limma`, it should be based on a trend-line fitted to known co-variates of the dispersions. This can be done using `weitrx_calibrate_trend`.



**Value**

A SummarizedExperiment object with metadata fields marking it as a weित्रix.

**Examples**

```
# Adjust weights so dispersion for each row is exactly 1. This is dubious
# for a small dataset, but would be fine for a dataset with many columns.
comp <- weित्रix_components(simwei, p=1, verbose=FALSE)
disp <- weित्रix_dispersions(simwei, comp)
cal <- weित्रix_calibrate(simwei, disp)
weित्रix_dispersions(cal, comp)
```

---

`weित्रix_calibrate_all` *Adjust weights element-wise by fitting a trend to squared residuals*

---

**Description**

This is a very flexible method of calibrating weights. It should be especially useful if your existing weights account for technical variation, but there is also biological variation. In this case large weights will tend to be overly optimistic, and a non-linear transformation of weights is needed.

**Usage**

```
weित्रix_calibrate_all(
  weित्रix,
  design = ~1,
  trend_formula = NULL,
  mu_min = NA,
  mu_max = NA,
  keep_fit = FALSE
)
```

**Arguments**

<code>weित्रix</code>	A weित्रix object, or an object that can be converted to a weित्रix with <code>as_weित्रix</code> .
<code>design</code>	A formula in terms of <code>colData(weित्रix)</code> or a design matrix, which will be fitted to the weित्रix on each row. Can also be a pre-existing Components object, in which case the existing fits ( <code>design\$row</code> ) are used.
<code>trend_formula</code>	A formula specification for predicting squared residuals. See below. If absent, <code>metadata(weित्रix)\$weित्रix\$calibrate_all_formula</code> is used.
<code>mu_min</code>	When fitting the GLM, omit observations where the estimated $\mu$ is less than this value. When calculating weights from the fitted GLM, clip $\mu$ to be at least this value.
<code>mu_max</code>	When fitting the GLM, omit observations where the estimated $\mu$ is greater than this value. When calculating weights from the fitted GLM, clip $\mu$ to be at most this value.

`keep_fit` Keep glm fit and the data used to create it. This can be large! If TRUE, these will be stored in `metadata(weitrrix)$weitrrix$all_fit` and `metadata(weitrrix)$weitrrix$all_data`.

### Details

Residuals are found relative to a fitted model. A trend model is then fitted to the squared residuals using a gamma GLM with log link function. Weitrrix weights are set based on the inverse of the fitted trend.

Residuals from a fitted model are generally smaller than residuals from the true model. A simple adjustment to the weights is made to account for this. Weights are reduced by a factor of  $(n - ncol(design) * nrow(weitrrix)) / n$  where  $n$  is the number of non-missing values in the weitrrix.

`trend_formula` may reference any row or column variables, or `mu` for the predicted value, or `weight` for the existing weights, or special factors `row` and `col`. Keep in mind also that a log link function is used.

Unlike in `weitrrix_calibrate_trend`, existing weights must be explicitly included in the formula if they are to be retained (see examples).

This function is currently not memory efficient, it should be fine for bulk experiments but may struggle for single cell. To reduce memory usage somewhat, when constructing the data frame on which to fit the glm, only columns referenced in `trend_formula` are included.

Example formulas:

`trend_formula=~1+offset(-log(weight))` Apply a global scaling, otherwise keeping weights the same.

`trend_formula=~log(weight)` Moderate weights by raising them to some power and applying some overall scaling factor. This will allow for biological variation.

`trend_formula=~poly(log(weight),2)` Apply a more complex quadratic curve-based moderation of weights.

`trend_formula=~col+offset(-log(weight))` Calibrate each sample's weights by a scaling factor. Note that due to the simplistic adjustment for using a fitted model rather than the true model, this may give misleading results when the design is unbalanced and there are few samples, i.e. when there are some samples with much higher leverage than others.

`trend_formula=~col*poly(log(weight),2)` Quadratic curve moderation of weights, applied to each sample individually.

### Value

A `SummarizedExperiment` object with metadata fields marking it as a weitrrix.

`metadata(weitrrix)$weitrrix` will contain the fitted trend model, and if requested the data frame used to fit the model.

### Examples

```
simcal <- weitrrix_calibrate_all(simwei, ~1, ~log(weight), keep_fit=TRUE)
```

```
metadata(simcal)$weitrrix$all_fit
```

---

weित्रix\_calibrate\_trend

*Adjust weights row-wise by fitting a trend to estimated dispersions*


---

### Description

Dispersions are estimated using `weित्रix_dispersions`. A trend line is then fitted to the dispersions using a gamma GLM with log link function. Weित्रix weights are calibrated based on this trend line.

### Usage

```
weित्रix_calibrate_trend(weित्रix, design = ~1, trend_formula = NULL)
```

### Arguments

<code>weित्रix</code>	A weित्रix object, or an object that can be converted to a weित्रix with <code>as_weित्रix</code> .
<code>design</code>	A formula in terms of <code>colData(weित्रix)</code> or a design matrix, which will be fitted to the weित्रix on each row. Can also be a pre-existing Components object, in which case the existing fits ( <code>design\$row</code> ) are used.
<code>trend_formula</code>	A formula specification for predicting log dispersion from columns of <code>rowData(weित्रix)</code> . If absent, <code>metadata(weित्रix)\$weित्रix\$calibrate_trend_formula</code> is used.

### Value

A SummarizedExperiment object with metadata fields marking it as a weित्रix.

Several columns are added to the `rowData`:

- `deg_free` Degrees of freedom for dispersion calculation.
- `dispersion_before` Dispersion before calibration.
- `dispersion_trend` Fitted dispersion trend.
- `dispersion_after` Dispersion for these new weights.

### Examples

```
rowData(simwei)$total_weight <- rowSums(weित्रix_weights(simwei))

# To estimate dispersions, use a simple model containing only an intercept
# term. Model log dispersion as a straight line relationship with log total
# weight and adjust weights to remove any trend.
cal <- weित्रix_calibrate_trend(simwei, ~1, trend_formula=~log(total_weight))

# This dataset has few rows, so calibration like this is dubious.
# Predictors in the fitted model are not significant.
summary( metadata(cal)$weित्रix$trend_fit )

# Information about the calibration is added to rowData
```

```

rowData(cal)

# A Components object may also be used as the design argument.
comp <- weitrrix_components(simwei, p=1, verbose=FALSE)
cal2 <- weitrrix_calibrate_trend(simwei, comp, trend_formula=~log(total_weight))

rowData(cal2)

```

---

weitrrix\_calplot

*Weight calibration plots, optionally versus a covariate*


---

## Description

Various plots based on weighted squared residuals of each element in the weitrrix.  $\text{weight} \times \text{residual}^2$  is the Pearson residual for a gamma GLM plus one, as used by `weitrrix_calibrate_all`.

## Usage

```

weitrrix_calplot(
  weitrrix,
  design = ~1,
  covar,
  cat,
  funnel = FALSE,
  guides = TRUE
)

```

## Arguments

<code>weitrrix</code>	A weitrrix object, or an object that can be converted to a weitrrix with <code>as_weitrrix</code> .
<code>design</code>	A formula in terms of <code>colData(weitrrix)</code> or a design matrix, which will be fitted to the weitrrix on each row. Can also be a Components object.
<code>covar</code>	Optional. A covariate. Specify as you would with <code>ggplot2::aes</code> . Can be a matrix of the same size as <code>weitrrix</code> .
<code>cat</code>	Optional. A categorical variable to break down the data by. Specify as you would with <code>ggplot2::aes</code> .
<code>funnel</code>	Flag. Produce a funnel plot? Note: <code>covar</code> can not be used for funnel plots.
<code>guides</code>	Show blue guide lines.

**Details**

This function is not memory efficient. It is suitable for typical bulk data, but generally not for single-cell.

Defaults to a boxplot of  $\sqrt{\text{weight}}$  weighted residuals. Blue guide bars are shown for the expected quartiles, these will ideally line up with the boxplot.

If `cat` is given, it will be used to break the elements down into categories.

If `covar` is given,  $\sqrt{\text{weight}}$  weighted residuals are plotted versus the covariate, with red trend lines for the mean and for the mean  $\pm$  one standard deviation. If the `weirix` is calibrated, the trend lines should be horizontal lines with y intercept close to -1, 0 and 1. Blue guide lines are shown for this ideal outcome.

Any of the variables available with `weirix_calibrate_all` can be used for `covar` or `cat`.

**Value**

A `ggplot2` plot.

**Examples**

```
weirix_calplot(simwei, ~1)
weirix_calplot(simwei, ~1, covar=mu)
weirix_calplot(simwei, ~1, cat=col)

# weirix_calplot should generally be used after calibration
cal <- weirix_calibrate_all(simwei, ~1, ~col+log(weight))
weirix_calplot(cal, ~1, cat=col)

# You can use a matrix of the same size as the weirix as a covariate.
# It will often be useful to assess vs the original weighting.
weirix_calplot(cal, ~1, covar=weirix_weights(simwei))
```

---

`weirix_components`      *Principal components of a weirix*

---

**Description**

Finds principal components of a `weirix`. If varimax rotation is enabled, these are then rotated to enhance interpretability.

**Usage**

```
weirix_components(
  weirix,
  p = 0,
  design = ~1,
  n_restarts = 3,
```

```

    max_iter = 1000,
    tol = 1e-05,
    use_varimax = TRUE,
    initial = NULL,
    verbose = TRUE
  )

  weitrix_components_seq(
    weitrix,
    p,
    design = ~1,
    n_restarts = 3,
    max_iter = 1000,
    tol = 1e-05,
    use_varimax = TRUE,
    verbose = TRUE
  )

```

### Arguments

<code>weitrix</code>	A <code>weitrix</code> object, or an object that can be converted to a <code>weitrix</code> with <code>as_weitrix</code> .
<code>p</code>	Number of components to find.
<code>design</code>	A formula referring to <code>colData(weitrix)</code> or a matrix, giving predictors of a linear model for the experimental design. By default only an intercept term is used, i.e. rows are centered before finding components. A more complex formula might be used to account for batch effects. <code>~0</code> can be used if rows are already centered.
<code>n_restarts</code>	Number of restarts of the iteration to use.
<code>max_iter</code>	Maximum iterations.
<code>tol</code>	Stop iterating if R-squared increased by less than this amount in an iteration.
<code>use_varimax</code>	Use varimax rotation to enhance interpretability of components.
<code>initial</code>	Optional, an initial guess for column components (scores). Can have fewer columns than <code>p</code> , in which remaining components are initialized randomly. Can have more columns than <code>p</code> , in which case a randomly chosen subspace is used in each restart.
<code>verbose</code>	Show messages about the progress of the iterations.

### Details

Note that this is a slow numerical method to solve a gnarly problem, for the case where weights are not uniform. The case of uniform weights or weights that can be written as an outer product of row and column weights is somewhat faster, however there are much faster algorithms for this available elsewhere.

An iterative method is used, starting from a random initial set of components. It is possible for this to get stuck at a local minimum. To ameliorate this, the iteration is initially run `n_restarts` times and the best result used. This is then iterated further. Examine `all_R2s` in the output to see if this

is happening – if the values are not all nearly identical, the iteration is sometimes getting stuck at local minima. Increase `n_restarts` to increase the odds of finding the global minimum.

## Value

A "Components" object with the following elements accessible using `$`.

- `row` Row matrix, aka loadings. Rows are rows in the `weitrrix`, and columns contain the experimental design (usually just an intercept term), and components.
- `col` Column matrix, aka scores. Rows are columns in the `weitrrix`, and columns contain fitted coefficients for the experimental design, and components.
- `R2` Weighted R squared statistic. The proportion of total variance explained by the components.
- `all_R2s` R2 statistics from all restarts. This can be used to check how consistently the iteration finds optimal components.
- `ind_designColumn` indices associated with experimental design.
- `ind_componentsColumn` indices associated with components.

For a result `comp`, the original measurements are approximated by `comp$row %*% t(comp$col)`.

`weitrrix_components_seq` returns a list of Components objects, with increasing numbers of components from 1 up to `p`.

## Functions

- `weitrrix_components()`: Find a matrix decomposition with the specified number of components.
- `weitrrix_components_seq()`: Produce a sequence of `weitrrix` decompositions with 1 to `p` components.

## Examples

```
# Variables in rows, observations in columns, as per Bioconductor convention
dat <- t(iris[,1:4])

# Find two components
comp <- weitrrix_components(dat, p=2, max_iter=5, n_restart=1)

# Examine row and col matrices
pairs(comp$row, panel=function(x,y) text(x,y,rownames(comp$row)))
pairs(comp$col)
```

---

weitr <sub>ix</sub> _confects	<i>Top confident effects based on one or more contrasts of a linear model for each row</i>
-------------------------------	--

---

### Description

This function provides topconfects-style testing of a linear model contrast, as well as a multi-contrast extension of this method for F-tests with effect sizes.

### Usage

```
weitrix_confects(
  weitrix,
  design,
  coef = NULL,
  contrasts = NULL,
  effect = c("auto", "contrast", "sd", "cohen_f"),
  dispersion_est = c("ebayes_limma", "row", "none"),
  fdr = 0.05,
  step = NULL,
  full = FALSE
)
```

### Arguments

weitr <sub>ix</sub>	A weitr <sub>ix</sub> object, or an object that can be converted to a weitr <sub>ix</sub> with <code>as_weitr<sub>ix</sub></code> .
design	A formula in terms of <code>colData</code> (weitr <sub>ix</sub> or a design matrix, which will be fitted to the weitr <sub>ix</sub> on each row. Can also be a pre-existing Components object, in which case the existing fits ( <code>design\$row</code> ) are used.
coef	Give either <code>coef</code> or <code>contrasts</code> but not both. If <code>coef</code> is given, it is converted into a set of contrasts that simply test each given coefficient. Coefficients can be specified by number or name.
contrasts	Give either <code>coef</code> or <code>contrasts</code> but not both. One or more contrasts of interest, i.e. specifications of linear combination of coefficients. Each contrast should be placed in a column. The number of rows should match the number of coefficients.
effect	Effect to estimate and provide confidence bounds on. See description.
dispersion_est	Method of estimating per-row dispersion. See description.
fdr	False Discovery Rate to control for.
step	Granularity of effect sizes to test.
full	If TRUE, output some further columns related to the calculations.



## Details

Based on the `effect` argument, the estimated effect may be:

- "auto" Choose "contrast" or "sd" as appropriate.
- "contrast" The estimated contrast. This should produce results identical to a `limma-topconfects` analysis.
- "sd" Standard deviation explained (i.e. square root of the variance explained) by the part of the model captured by the contrasts provided.
- "cohen\_f" Cohen's  $f$ , i.e. the signal to noise ratio. Ranking is similar to traditional ranking of results by p-value.

Based on the `dispersion_est` argument, the estimated residual dispersion is estimated as:

- "none" Weित्रix is assumed to be fully calibrated already. Dispersion is assumed to be 1. If the assumption is correct, this is most powerful, as there is no uncertainty to the dispersion.
- "row" Dispersion is estimated based on the residuals for each row. With a limited number of columns, this estimate is uncertain (low residual degrees of freedom), so may lack power.
- "ebayes\_limma" Default, recommended. Perform Empirical Bayes squeezing of dispersions, using `limma::squeezeVar`. This also reduces the uncertainty about the dispersion (manifesting as extra "prior" degrees of freedom), increasing the power of the test.

In results from this function, whenever we talk about the mean, standard deviation explained, or typical observation error, this should be understood to be weighted. Standard deviation explained is in the same units as the observations, but its estimation is weighted by the weights, so in a row with some high weight observations and other low weight observations, estimated standard deviation explained will mostly be driven by the high weight observations.

## Value

A `topconfects` result. The `$table` data frame contains columns:

- `effect` Estimated effect (as requested using the `effect` parameter).
- `confect` An inner confidence bound on effect.
- `fdr_zero` FDR-adjusted p-value for the null hypothesis that effect is zero.
- `row_mean` Weighted row mean.
- `typical_obs_err` Typical residual standard deviation (square root of variance) associated with observations in this row. Note that each observation has its own associated variance, based on its weight and the row dispersion estimate used. This column is calculated from the weighted average variance of observations.

## Examples

```
# Simplest possible test
# Which rows have an average different from zero?
weित्रix_confects(simwei, ~1, coef="(Intercept)")

# See vignettes for more substantial examples
```

---

weitr<sub>x</sub>\_dispersions     *Calculate row dispersions*

---

### Description

Calculate the dispersion of each row. For each observation, this value divided by the weight gives the observation's variance.

### Usage

```
weitrx_dispersions(weitrx, design = ~1)
```

### Arguments

weitr <sub>x</sub>	A weitr <sub>x</sub> object, or an object that can be converted to a weitr <sub>x</sub> with <code>as_weitr<sub>x</sub></code> .
design	A formula in terms of <code>colData(weitr<sub>x</sub>)</code> or a design matrix, which will be fitted to the weitr <sub>x</sub> on each row. Can also be a pre-existing Components object, in which case the existing fits ( <code>design\$row</code> ) are used.

### Value

A numeric vector.

### Examples

```
# Using a model just containing an intercept
weitrx_dispersions(simwei, ~1)

# Allowing for one component of variation, the dispersions are lower
comp <- weitrx_components(simwei, p=1, verbose=FALSE)
weitrx_dispersions(simwei, comp)
```

---

weitr<sub>x</sub>\_elist     *Convert a weitr<sub>x</sub> object to a limma EList object*

---

### Description

The resulting object can be used as input to `limma::lmFit` for a limma analysis.

### Usage

```
weitrx_elist(weitrx)
```

### Arguments

weitr <sub>x</sub>	A weitr <sub>x</sub> object.
--------------------	------------------------------

**Value**

A limma EList object.

**Examples**

```
library(limma)

elist <- weित्रix_elist(simwei)
design <- model.matrix(~true_score, data=colData(simwei))
fit <- lmFit(elist, design)
# ...perform further limma analysis steps as desired...
```

---

weित्रix_hill	<i>Calculate Hill numbers (effective number of observations) for rows or columns</i>
---------------	--

---

**Description**

Effective numbers of observations. order=0 produces count of non-zero weights. order=1 produces exp(entropy). order=2 produces the inverse Simpson index.

**Usage**

```
weित्रix_hill(weित्रix, what = c("row", "col"), order = 2)
```

**Arguments**

weित्रix	A weित्रix object.
what	Calculate for rows ("row") (default) or columns ("col")?
order	Order of the Hill numbers.

**Value**

A numeric vector of effective numbers of observations.

**Examples**

```
weित्रix_weights(simwei)

weित्रix_hill(simwei, what="row", order=0)
weित्रix_hill(simwei, what="row", order=1)
weित्रix_hill(simwei, what="row", order=2)

weित्रix_hill(simwei, what="col", order=0)
weित्रix_hill(simwei, what="col", order=1)
weित्रix_hill(simwei, what="col", order=2)
```

---

`weitrax_randomize`      *Generate a random normally distributed version of a weitrax*

---

### Description

Values are generated with variance equal to  $1/\text{weight}$ . This can be used to see what R-squared would be achieved with purely random data, and therefore an appropriate number of components to use. This is known as Parallel Analysis.

### Usage

```
weitrax_randomize(weitrax)
```

### Arguments

`weitrax`      A weitrax object, or an object that can be converted to a weitrax with `as_weitrax`.

### Value

A SummarizedExperiment object with metadata fields marking it as a weitrax.

### See Also

[components\\_seq\\_screplot](#)

### Examples

```
weitrax_randomize(simwei)
```

---

`weitrax_sd_confects`      *Find rows with confidently excessive variability in a calibrated weitrax*

---

### Description

Find rows with confident excess standard deviation beyond what is expected based on the weights of a calibrated weitrax. This may be used, for example, to find potential marker genes.

### Usage

```
weitrax_sd_confects(  
  weitrax,  
  design = ~1,  
  fdr = 0.05,  
  step = 0.001,  
  assume_normal = TRUE  
)
```

**Arguments**

<code>weitr<sub>x</sub></code>	A weitr <sub>x</sub> object, or an object that can be converted to a weitr <sub>x</sub> with <code>as_weitr<sub>x</sub></code> .
<code>design</code>	A formula in terms of <code>colData(weitr<sub>x</sub>)</code> or a design matrix, which will be fitted to the weitr <sub>x</sub> on each row. Can also be a pre-existing Components object, in which case the existing fits ( <code>design\$row</code> ) are used.
<code>fdr</code>	False Discovery Rate to control for.
<code>step</code>	Granularity of effect sizes to test.
<code>assume_normal</code>	Assume weighted residuals are normally distributed? Assumption of normality is quite a strong assumption here. If TRUE, tests are based on the weighted squared residuals following a chi-squared distribution. If FALSE, tests are based on assuming the dispersion follows an asymptotically normal distribution, with variance estimated from the weighted squared residuals. If FALSE, a reasonably large number of columns is required. Defaults to TRUE.

**Details**

Important note: With the default setting of `assume_normal=TRUE`, the "confect" values produced by this method are only valid if the weighted residuals are close to normally distributed. If you have a reasonably large number of columns (eg single cell data), you can and should relax this assumption by specifying `assume_normal=FALSE`.

This is a conversion of the "dispersion" statistic for each row into units that are more readily interpretable, accompanied by confidence bounds with a multiple testing correction.

We are looking for further perturbation of observed values beyond what is accounted for by a linear model and, further, beyond what is expected based on the observation weights (assumed to be calibrated and so interpreted as  $1/\text{variance}$ ). We are seeking to estimate the standard deviation of this further perturbation.

The weitr<sub>x</sub> must have been calibrated for results to make sense.

Top confident effect sizes are found using the `topconfects` method, based on the model that the observed weighted sum of squared residuals being non-central chi-square distributed.

Note that all calculations are based on weighted residuals, with a rescaling to place results on the original scale. When a row has highly variable weights, this is an approximation that is only sensible if the weights are unrelated to the values themselves.

**Value**

A `topconfects` result. The `$table` data frame contains columns:

- `effect` Estimated excess standard deviation, in the same units as the observations themselves. 0 if the dispersion is less than 1.
- `confect` A lower confidence bound on effect.
- `row_mean` Weighted mean of observations in this row.
- `typical_obs_err` Typical accuracy of each observation.
- `dispersion` Dispersion. Weighted sum of squared residuals divided by residual degrees of freedom.

- n\_present Number of observations with non-zero weight.
- df Degrees of freedom. n minus the number of coefficients in the model.
- fdr\_zero FDR-adjusted p-value for the null hypothesis that effect is zero.

Note that dispersion = effect<sup>2</sup>/typical\_obs\_err<sup>2</sup> + 1 for non-zero effect values.

### Examples

```
# weित्रix_sd_confects should only be used with a calibrated weित्रix
calwei <- weित्रix_calibrate_all(simwei, ~1, ~1)

weित्रix_sd_confects(calwei, ~1)
```

---

weित्रix_weights	<i>Get or set a weित्रix object's "weights" matrix</i>
------------------	--

---

### Description

Gets or sets the appropriate assay in the SummarizedExperiment object.

### Usage

```
weित्रix_weights(weित्रix)

weित्रix_weights(x) <- value
```

### Arguments

weित्रix	A weित्रix object.
x	The weित्रix to modify.
value	The new matrix.

### Value

A matrix-like object such as a matrix or a DelayedArray.

### Examples

```
weित्रix_weights(simwei)
```

---

weित्रix\_x                      *Get or set a weित्रix object's "x" matrix*

---

**Description**

Gets or sets the appropriate assay in the SummarizedExperiment object.

**Usage**

```
weित्रix_x(weित्रix)

weित्रix_x(x) <- value
```

**Arguments**

weित्रix	A weित्रix object.
x	The weित्रix to modify.
value	The new matrix.

**Value**

A matrix-like object such as a matrix or a DelayedArray.

**Examples**

```
weित्रix_x(simwei)

simwei2 <- simwei
weित्रix_x(simwei2) <- weित्रix_x(simwei2) * 2
```

---

well\_knotted\_spline                      *Natural cubic spline with good choice of knots*

---

**Description**

For use in model formulas, natural cubic spline as in `splines::ns` but with knot positions chosen using k-means rather than quantiles. Automatically uses less knots if there are insufficient distinct values.

**Usage**

```
well_knotted_spline(x, n_knots, verbose = TRUE)
```

## Arguments

x	The predictor variable. A numeric vector.
n_knots	Number of knots to use.
verbose	If TRUE, produce a message about the knots chosen.

## Details

Wong (1982, 1984) showed the asymptotic density of k-means in 1 dimension is proportional to the cube root of the density of x. Compared to using quantiles (the default for `ns`), choosing knots using k-means produces a better spread of knot locations if the distribution of values is very uneven.

k-means is computed in an optimal, deterministic way using `Ckmeans.1d.dp`.

## Value

A matrix of predictors, similar to `ns`.

This function supports "safe prediction" (see `makepredictcall`). Original knot locations will be used for prediction with `predict`.

## References

Wong, M. (1982). Asymptotic properties of univariate sample k-means clusters. Working paper #1341-82, Sloan School of Management, MIT. <https://dspace.mit.edu/handle/1721.1/46876>

Wong, M. (1984). Asymptotic properties of univariate sample k-means clusters. *Journal of Classification*, 1(1), 255–270. <https://doi.org/10.1007/BF01890126>

## See Also

`ns`, `makepredictcall`

## Examples

```
lm(mpg ~ well_knotted_spline(wt,3), data=mtcars)

# When insufficient unique values exist, less knots are used
lm(mpg ~ well_knotted_spline(gear,3), data=mtcars)

library(ggplot2)
ggplot(diamonds, aes(carat, price)) +
  geom_point() +
  geom_smooth(method="lm", formula=y~well_knotted_spline(x,10))
```



# Index

- \* **datasets**
  - simwei, 8
- as\_weitrix, 2
- bless\_weitrix, 3
- Ckmeans.1d.dp, 24
- components\_seq\_scee, 4
- components\_seq\_sceeplot, 20
- components\_seq\_sceeplot
  - (components\_seq\_scee), 4
- counts\_proportions, 5
- counts\_shift, 6
- makepredictcall, 24
- matrix\_long, 7
- ns, 24
- predict, 24
- simwei, 8
- weitrix\_calibrate, 8
- weitrix\_calibrate\_all, 9
- weitrix\_calibrate\_trend, 11
- weitrix\_calplot, 12
- weitrix\_components, 13
- weitrix\_components\_seq
  - (weitrix\_components), 13
- weitrix\_confects, 16
- weitrix\_dispersions, 18
- weitrix\_elist, 18
- weitrix\_hill, 19
- weitrix\_randomize, 20
- weitrix\_sd\_confects, 20
- weitrix\_weights, 22
- weitrix\_weights<- (weitrix\_weights), 22
- weitrix\_x, 23
- weitrix\_x<- (weitrix\_x), 23
- well\_knotted\_spline, 23