Package 'dominatR'

November 11, 2026

Title Feature Dominance-based R Package for Genomic Data

Version 1.0.0

Date 2025-09-10

Description
dominatR is an R package for quantifying and visualizing feature dominance in datasets. dominatR applies concepts drawn from physics such as center of mass and shannon's entropy to effectively visualize features (e.g. genes) that are present within a specific context or condition. The package integrates, dataframes, matrices and SummerizedExperiment objects and is able to perform common genomic normalization methods. The key aspect is the generation of plots that serve to highlight context-relevant feature dominance.

License MIT + file LICENSE

URL https://github.com/VanBortleLab/dominatR, https://vanbortlelab.github.io/dominatR/

BugReports https://github.com/VanBortleLab/dominatR/issues

Encoding UTF-8

LazyData false

Depends R (>= 4.5.0)

Imports scales, ggnewscale, SummarizedExperiment, dplyr, rlang, ggforce, geomtextpath, ggplot2

Suggests BiocStyle, airway, tidyverse, knitr, rmarkdown, testthat (>= 3.0.0), dominatRData

VignetteBuilder knitr

biocViews Visualization, Normalization, Classification, GeneExpression

Config/testthat/edition 3

RoxygenNote 7.3.3

Config/Needs/website rmarkdown

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

git_branch RELEASE_3_23

git_last_commit 5264c80

git_last_commit_date 2026-04-28

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Author Simon Lizarazo [aut, cre] (ORCID:
<https://orcid.org/0009-0001-8974-6225>),
Ethan Chen [aut],
Rajendra K C [aut],
Kevin Van Bortle [aut, cph]

Maintainer Simon Lizarazo <silizarazoch@gmail.com>

Contents

centmass 2
cpm_normalization 4
entropy 6
minmax_normalization 7
plot_circle 10
plot_circle_frequency 17
plot_rope 21
plot_triangle 25
Qentropy 29
quantile_normalization 31
rpkm_normalization 33
tpm_normalization 35

Index **38**

`centmass` *Compute the "center of mass" for rows of a data frame or SummarizedExperiment*

Description

For each row of the numeric data, `centmass()` computes a 2D center of mass with coordinates (`comx`, `comy`). The `x_coord` and `y_coord` vectors specify the location for each column's "mass."

The original usage assumes a ternary coordinate system by default, but this can be generalized to any scenario where columns represent discrete "masses" at known (x,y) positions.

By default, `x_coord = c(0, 1, 0.5)` and `y_coord = c(0, 0, sqrt(3)/2)`, which correspond to the corners of an equilateral triangle (often used in ternary plots).

Usage

```
centmass(
  x,
  x_coord = c(0, 1, 0.5),
  y_coord = c(0, 0, sqrt(3)/2),
  assay_name = NULL
)
```

Arguments

x	A data.frame (with numeric columns) or a SummarizedExperiment.
x_coord	Numeric vector of length equal to the number of columns in x, specifying the x-coordinates of each column's mass.
y_coord	Numeric vector of length equal to the number of columns in x, specifying the y-coordinates of each column's mass.
assay_name	If x is a SummarizedExperiment, the name of the assay to use. Defaults to the first assay if not specified.

Value

- If x is a data.frame, returns a new `data.frame` with columns `comx` and `comy`.
- If x is a SummarizedExperiment, returns the same object but with two new columns `comx` and `comy` in `rowData(x)`.

Examples

```
library(SummarizedExperiment)
library(airway)
data('airway')

se = airway

# Only use a random subset of 1000 rows
set.seed(123)
idx <- sample(seq_len(nrow(se)), size = min(1000, nrow(se)))
se <- se[idx, ]

# Let's subset for the first 3 columns for this example
se = se[,1:3]
# ------------------------------
# 1) Using a data.frame
# ------------------------------


df = assay(se) |> as.data.frame()

df = centmass(df)
head(df)
```

```
# ------------------------------
# 2) Using a SummarizedExperiment
# ------------------------------

se2 = centmass(se)

## X and Y coordinates are stored in rowData(se2)
head(rowData(se2))
```

`cpm_normalization` *Counts Per Million normalization*

Description

Normalizes a count matrix (or a SummarizedExperiment assay) by the counts-per-million (CPM) method. Specifically:

1. If `log_trans = TRUE`, a `log2(x + 1)` transform is applied afterward.

Usage

```
cpm_normalization(
  x,
  log_trans = FALSE,
  assay_name = NULL,
  new_assay_name = NULL
)
```

Arguments

`x`	A `matrix`, `data.frame`, or a `SummarizedExperiment` object.
`log_trans`	Logical. If `TRUE`, apply `log2(... + 1)` transform to the CPM-normalized values.
`assay_name`	If `x` is a `SummarizedExperiment`, name of the assay to normalize (defaults to the first assay). Ignored otherwise.
`new_assay_name`	If `x` is a `SummarizedExperiment`, name of a new assay where results should be stored (defaults to `NULL`, meaning the existing assay is overwritten).

Value

- If `x` is a `matrix` or `data.frame`, returns a **matrix** of CPM-normalized (and optionally `log2`-transformed) counts.
- If `x` is a `SummarizedExperiment`, returns the same `SummarizedExperiment` object with the specified assay replaced or a new assay created containing the CPM-normalized data.

Examples

```
library(SummarizedExperiment)
library(airway)
data('airway')

se = airway

# Only use a random subset of 1000 rows
set.seed(123)
idx <- sample(seq_len(nrow(se)), size = min(1000, nrow(se)))
se <- se[idx, ]

# --------------------------------
# 1) Using a data.frame
# --------------------------------

df = assay(se)

## Without log transformation
df1 = cpm_normalization(df, log_trans = FALSE)

df1[1:5,1:5]

## With log transformation
df1 = cpm_normalization(df, log_trans = TRUE)

df1[1:5,1:5]

# --------------------------------
# 2) Using a SummarizedExperiment
# --------------------------------

# If now new_assay_name is provided, then overwrites existing assay
se2 = cpm_normalization(se, log_trans = FALSE)

se2
head(assay(se2))

# If new new_assay_name, normalization stored in a new object
se2 = cpm_normalization(se, log_trans = FALSE, new_assay_name = 'cpm_counts')

se2
head(assay(se2, 'cpm_counts'))

# A specific assay can also be selected
new_matrix =  matrix(data = sample(x = seq(1, 100000),
                                   size = nrow(se) * ncol(se),
                                   replace = TRUE),
                     nrow = nrow(se),
                     ncol = ncol(se))
rownames(new_matrix) = rownames(se)
colnames(new_matrix) = colnames(se)
```

```
## Creating a new assay called new counts
assay(se, 'new_counts') = new_matrix

se2 = cpm_normalization(se, new_assay_name = 'cpm_counts_new', assay_name =
'new_counts')

se2
head(assay(se2, 'cpm_counts_new'))
```

`entropy` *Compute Shannon Entropy on row-normalized data*

Description

Compute Shannon Entropy on row-normalized data

Usage

```
entropy(x, assay_name = NULL, new_assay_name = "Entropy")
```

Arguments

`x`	A data.frame (with numeric columns) or a SummarizedExperiment (with an assay of numeric data).
`assay_name`	(SummarizedExperiment only) The name of the assay to transform and compute Entropy on. If NULL, uses the first assay.
`new_assay_name`	If you prefer to store Q-values in a *new* assay, provide a name. By default 'Entropy'

Value

- If `x` is a data.frame: returns the same data.frame in which numeric columns have been replaced by their row-wise proportions, and an `Entropy` column is appended.
- If `x` is a SummarizedExperiment: returns the same SummarizedExperiment in with a new assay (Default name is `Entropy`) and `rowData(x)$Entropy` is added.

Examples

```
library(SummarizedExperiment)
library(airway)
data('airway')

se = airway

# Only use a random subset of 1000 rows
set.seed(123)
idx <- sample(seq_len(nrow(se)), size = min(1000, nrow(se)))
```

```
se <- se[idx, ]

# ------------------------------
# 1) Using a data.frame
# ------------------------------
df = assay(se) |> as.data.frame()
df = entropy(df)

## The function adds a new column called Entropy and transform all
## the counts accordingly
head(df)

# ------------------------------
# 2) Using a SummarizedExperiment
# ------------------------------

## The function adds a new assay called 'Entropy' with the transformed
## counts.
## This name can be modified with the 'new_assay_name' parameter
## In the rowData dataframe a new column called Entropy is added.
se2 <- entropy(se, new_assay_name = 'Entropy')
se2

## In case the experiment has multiple assays, the function allows you to
## choose which assay to use.
new_matrix =  matrix(data = sample(x = seq(1, 100000),
                                   size = nrow(se) * ncol(se),
                                   replace = TRUE),
                     nrow = nrow(se),
                     ncol = ncol(se))
rownames(new_matrix) = rownames(se)
colnames(new_matrix) = colnames(se)

## Creating a new assay called new counts
assay(se, 'new_counts') = new_matrix


## Saving the entropy values as Entropy_newmatrix using the assay 'new
## counts'
se2 = entropy(se,
              new_assay_name = 'Entropy_newmatrix',
              assay_name = 'new_counts')

se2
```

---

minmax_normalization *Min-Max Normalization*

Description

Scales each column of a matrix (or SummarizedExperiment assay) so that the minimum value in that column is mapped to `new_min` and the maximum value is mapped to `new_max`

Usage

```
minmax_normalization(
  x,
  new_min = 0,
  new_max = 1,
  assay_name = NULL,
  new_assay_name = NULL
)
```

Arguments

`x`	A numeric `matrix`, `data.frame`, or `SummarizedExperiment`.
`new_min`	The lower bound of the new range (default 0).
`new_max`	The upper bound of the new range (default 1).
`assay_name`	If `x` is a SummarizedExperiment, name of the assay to normalize. Defaults to the first assay if none is specified.
`new_assay_name`	If `x` is a SummarizedExperiment, name of a new assay to store the normalized data. If `NULL`, overwrites the assay specified by `assay_name`.

Value

- If `x` is a data.frame or matrix, returns a matrix of column-wise scaled values (same dimensions as `x`).
- If `x` is a SummarizedExperiment, returns the same SummarizedExperiment object with the chosen or new assay replaced by the scaled values.

Examples

```
library(SummarizedExperiment)
library(airway)
data('airway')

se <- airway

# Only use a random subset of 1000 rows
set.seed(123)
idx <- sample(seq_len(nrow(se)), size = min(1000, nrow(se)))
se <- se[idx, ]

# ------------------------------
# 1) Using a data.frame
# ------------------------------

df <- assay(se)
```

```
df1 <- minmax_normalization(df)

apply(df1, 2, range)

## Using a new range
df1 <- minmax_normalization(df, new_min = 5, new_max = 10)

apply(df1, 2, range)

# ------------------------------
# 2) Using a SummarizedExperiment
# ------------------------------

# If now new_assay_name is provided, then overwrites existing assay
se2 <- minmax_normalization(se)

apply(assay(se2), 2, range)


# If new new_assay_name, normalization stored in a new object
se2 <- minmax_normalization(se, new_assay_name = 'minmax_counts')

apply(assay(se2, 'minmax_counts'), 2, range)

# A specific assay can also be selected
new_matrix <-  matrix(data = sample(x = seq(1, 100000),
                                    size = nrow(se) * ncol(se),
                                    replace = TRUE),
                      nrow = nrow(se),
                      ncol = ncol(se))
rownames(new_matrix) <- rownames(se)
colnames(new_matrix) <- colnames(se)

## Creating a new assay called new counts
assay(se, 'new_counts') <- new_matrix

se2 <- minmax_normalization(se,
                            new_assay_name = 'minmax_counts_new',
                            assay_name = 'new_counts')

apply(assay(se2, 'minmax_counts_new'), 2, range)

## Using a different range
se2 <- minmax_normalization(se,
                            new_assay_name = 'minmax_counts_new',
                            assay_name = 'new_counts',
                            new_min = 10,
                            new_max = 20)

apply(assay(se2, 'minmax_counts_new'), 2, range)
```

plot_circle *Circular Dominance Plot (More than 3 variables)*

Description

Produces a radial dominance plot in which each observation is located by:

- **Angle (t)** – the variable with the greatest value (ties broken at random).
- **Radius (r)** – a monotone mapping of the row-wise Shannon entropy: points with low entropy (one variable dominates) are near the edge; points with high entropy lie toward the centre.

The circle is partitioned into n coloured slices; optional factor information can colour/jitter points independently. Labels for each slice may be drawn as curved text on the circle or shown in a legend.

Usage

```
plot_circle(
  x,
  n,
  column_variable_factor = NULL,
  variables_highlight = NULL,
  entropyrange = c(0, Inf),
  magnituderange = c(0, Inf),
  background_alpha_polygon = 0.05,
  background_polygon = NULL,
  background_na_polygon = "whitesmoke",
  point_size = 1,
  point_fill_colors = NULL,
  point_fill_na_colors = "whitesmoke",
  point_line_colors = NULL,
  point_line_na_colors = "whitesmoke",
  straight_points = TRUE,
  line_col = "gray90",
  out_line = "black",
  label = "legend",
  text_label_curve_size = 3,
  assay_name = NULL,
  output_table = TRUE
)
```

Arguments

x	A numeric `data.frame`, `matrix`, or a `SummarizedExperiment`.
n	Integer (≥ 3). How many numeric variables to visualise. Must match `length(column_variable_factor` when supplied.

`column_variable_factor`
: Character. Name of a column (or rowData column in a SummarizedExperiment) holding a categorical variable whose levels will colour the points. If `NULL` (default) points are coloured by their dominant variable.

`variables_highlight`
: Character vector naming which variables should receive curved text labels when `label = "curve"`. Defaults to all variables.

`entropyrange`, `magnituderange`
: Numeric length-2 vectors. Rows falling outside either interval are excluded from the plot/data.

`background_alpha_polygon`
: Alpha level (0–1) for the coloured background slices.

`background_polygon`
: Character vector of slice fill colours; defaults to `scales::hue_pal()(n)`. `background_na_polygon` sets the colour for missing values.

`background_na_polygon`, `point_fill_na_colors`, `point_line_na_colors`
: Sets the colour for missing values.

`point_size`
: Numeric; plotted point size.

`point_fill_colors`, `point_line_colors`
: Optional colour vectors for point fill / outline.

`straight_points`
: Logical. If TRUE points are plotted in a straight line.

`line_col`
: Colour for the inner grid / slice borders.

`out_line`
: Colour for the outermost circle.

`label`
: Either `"legend"` (default) to list variables in a legend or `"curve"` to print them around the rim.

`text_label_curve_size`
: Numeric font size for curved labels.

`assay_name`
: (SummarizedExperiment only) Which assay to use. Defaults to the first assay.

`output_table`
: Logical. Also return the underlying data frame?

Details

Radius mapping: A linear map is used

$$r \;=\; 100\,\frac{n - 2^{H}}{n - 1}$$

where H is the Shannon entropy of the row after log base 2, so $H \in [0, \log_2 n]$.

Value

If `output_table = TRUE` a list with:

- `circle_plot` — a ggplot object;
- `data` — the augmented data frame containing entropy, radius, (x,y) coordinates, dominant variable and optional factor.

Otherwise only the `ggplot` object is returned.

Examples

```
library(SummarizedExperiment)
library(airway)
library(tidyverse)
data('airway')
se = airway

# Only use a random subset of 1000 rows
set.seed(123)
idx <- sample(seq_len(nrow(se)), size = min(500, nrow(se)))
se <- se[idx, ]

## Normalize the data first using tpm_normalization
rowData(se)$gene_length = rowData(se)$gene_seq_end
- rowData(se)$gene_seq_start

se = tpm_normalization(se, log_trans = TRUE, new_assay_name = 'tpm_norm')

# --------------------------------
# 1) Using a data.frame
# --------------------------------

df <- assay(se, 'tpm_norm') |> as.data.frame()

## For simplicity let's rename the columns
colnames(df) <- paste('Column_', 1:8, sep ='')

# Default
plot_circle(
  x = df,
  n = 8,
  entropyrange     = c(0, 3),
  magnituderange   = c(0, Inf),
  label  = 'legend', output_table = FALSE
)

# Filtering by entropy, 8 variables, max entropy value is log2(8)
plot_circle(
  x = df,
  n = 8,
  entropyrange     = c(2, 3),
  magnituderange   = c(0, Inf),
  label  = 'legend', output_table = FALSE
)

plot_circle(
  x = df,
  n = 8,
  entropyrange     = c(0, 2),
  magnituderange   = c(0, Inf),
  label  = 'legend', output_table = FALSE
)
```

```
# Aesthetics modification
plot_circle(
  x = df,
  n = 8,
  entropyrange     = c(0, 2),
  magnituderange   = c(0, Inf),
  label  = 'curve',
  output_table = FALSE
)

# It is possible to highlight only a specific variable
plot_circle(
  x = df,
  n = 8,
  entropyrange     = c(0, 2),
  magnituderange   = c(0, Inf),
  label  = 'legend',
  output_table = FALSE,
  background_alpha_polygon = 0.2,
  background_na_polygon = 'transparent',
  background_polygon = c('Column_1'  = 'indianred',
                         'Column_3' = 'lightblue',
                         'Column_5' = 'lightgreen'),
  point_fill_colors = c('Column_1'  = 'darkred',
                        'Column_3' = 'darkblue',
                        'Column_5' = 'darkgreen'),
  point_line_colors = c('Column_1'  = 'black',
                        'Column_3' = 'black',
                        'Column_5' = 'black')
)

# Let's create a factor column in our df
df$factor <- sample(c('A', 'B', 'C', 'D'), size = nrow(df), replace = TRUE)

# It is possible to visualize things by this specific factor column using
# column_variable_factor
plot_circle(
  x = df,
  n = 8,
  column_variable_factor = 'factor',
  entropyrange     = c(0, 2),
  magnituderange   = c(0, Inf),
  label  = 'legend',
  output_table = FALSE,
  background_alpha_polygon = 0.2,
  background_na_polygon = 'transparent',
  background_polygon = c('Column_1'  = 'indianred',
                         'Column_3' = 'lightblue',
                         'Column_5' = 'lightgreen')
)

# Colors can be modified
```

```
plot_circle(
  x = df,
  n = 8,
  column_variable_factor = 'factor',
  entropyrange     = c(0, 2),
  magnituderange   = c(0, Inf),
  label  = 'curve',
  output_table = FALSE,
  background_alpha_polygon = 0.02,
  background_na_polygon = 'transparent',
  point_fill_colors = c('A' = 'black',
                        'B' = 'gray',
                        'C' = 'white',
                        'D' = 'orange'),
  point_line_colors = c('A' = 'black',
                        'B' = 'gray',
                        'C' = 'white',
                        'D' = 'orange')
)

# Size of the points can be modified too
plot_circle(
  x = df,
  n = 8,
  point_size =  2,
  column_variable_factor = 'factor',
  entropyrange     = c(0, 2),
  magnituderange   = c(0, Inf),
  label  = 'curve',
  output_table = FALSE,
  background_alpha_polygon = 0.02,
  background_na_polygon = 'transparent',
  point_fill_colors = c('A' = 'black',
                        'B' = 'gray',
                        'C' = 'white',
                        'D' = 'orange'),
  point_line_colors = c('A' = 'black',
                        'B' = 'gray',
                        'C' = 'white',
                        'D' = 'orange')
)

# Retrieving a dataframe with the results used for plotting,
# set output_table <- TRUE
plot <- plot_circle(
  x = df,
  n = 8,
  point_size =  2,
  column_variable_factor = 'factor',
  entropyrange     = c(0, 2),
  magnituderange   = c(0, Inf),
  label  = 'curve',
  output_table = TRUE,
```

```
  background_alpha_polygon = 0.02,
  background_na_polygon = 'transparent',
  point_fill_colors = c('A' = 'black',
                        'B' = 'gray',
                        'C' = 'white',
                        'D' = 'orange'),
  point_line_colors = c('A' = 'black',
                        'B' = 'gray',
                        'C' = 'white',
                        'D' = 'orange')
)


# The first object is the plot
plot[[1]]

# The second the dataframe with information for each row, including
# Entropy and the variable that dominates that particular observation.


head(plot[[2]])



# --------------------------------
# 1) Using a SummarizedExperiment
# --------------------------------
# Changing column names
colnames(se) <- paste('Column_', 1:8, sep ='')

# Default
plot_circle(
  x = se,
  n = 8,
  entropyrange     = c(0, 3),
  magnituderange   = c(0, Inf),
  label  = 'legend',
  output_table = FALSE,
  assay_name = 'tpm_norm'
)

# Filtering High Entropy genes
plot_circle(
  x = se,
  n = 8,
  entropyrange     = c(0, 1.5),
  magnituderange   = c(0, Inf),
  label  = 'legend',
  output_table = FALSE,
  assay_name = 'tpm_norm'
)

# Filtering Low Entropy genes
```

```
plot_circle(
  x = se,
  n = 8,
  entropyrange     = c(2, 3),
  magnituderange   = c(0, Inf),
  label  = 'legend',
  output_table = FALSE,
  assay_name = 'tpm_norm'
)


# Using a character column from rowData

plot_circle(
  x = se,
  n = 8,
  column_variable_factor = 'gene_biotype',
  entropyrange     = c(2,3),
  magnituderange   = c(0, Inf),
  label  = 'legend',
  output_table = FALSE,
  assay_name = 'tpm_norm'
)

plot_circle(
  x = se,
  n = 8,
  column_variable_factor = 'gene_biotype',
  point_size = 3,
  entropyrange     = c(0,1.5),
  magnituderange   = c(2, Inf),
  label  = 'legend',
  output_table = FALSE,
  assay_name = 'tpm_norm',
)

# Highlighting only a class of interest

plot_circle(
  x = se,
  n = 8,
  column_variable_factor = 'gene_biotype',
  point_size = 3,
  entropyrange     = c(0,1.5),
  magnituderange   = c(2, Inf),
  label  = 'legend',
  output_table = FALSE,
  assay_name = 'tpm_norm',
  point_fill_colors = c('miRNA' = 'orange'),
  point_line_colors = c('miRNA' = 'orange')
)
```

```
# Retrieving a dataframe with the results used for plotting,
# set output_table <- TRUE

plot <- plot_circle(
  x = se,
  n = 8,
  column_variable_factor = 'gene_biotype',
  point_size = 3,
  entropyrange     = c(0,1.5),
  magnituderange   = c(2, Inf),
  label  = 'legend',
  output_table = TRUE,
  assay_name = 'tpm_norm',
  point_fill_colors = c('miRNA' = 'orange'),
  point_line_colors = c('miRNA' = 'orange')
)

# It returns a list.
# The first object is the plot
plot[[1]]

# The second the dataframe with information for each row, including
# Entropy and the variable that dominates that particular observation.
head(plot[[2]])
```

`plot_circle_frequency` *Dominance–Entropy Frequency Plot*

Description

Visualises how often each categorical level ('Factor') is dominant at a given entropy score. The function expects the **second** element of the list returned by `plot_circle()`.

Visualises how often each categorical level ('Factor') is dominant at a given entropy score. The function expects the **second** element of the list returned by `plot_circle()`.

Usage

```
plot_circle_frequency(
  n,
  circle,
  single = FALSE,
  legend = TRUE,
  numb_columns = 1,
  filter_class = NULL,
  point_size = 2
)

plot_circle_frequency(
```

```
  n,
  circle,
  single = FALSE,
  legend = TRUE,
  numb_columns = 1,
  filter_class = NULL,
  point_size = 2
)
```

Arguments

n	Integer. Number of numeric variables used in plot_circle().
circle	The list returned by plot_circle().
single	Logical. If TRUE draw one combined panel; otherwise facet by Factor.
legend	Logical. Show a legend for the plot
numb_columns	Faceting columns when single = FALSE.
filter_class	Character vector of levels to keep; NULL keeps all.
point_size	Numeric. Size of jitter points.

Value

A list with

- plot_stat — a ggplot object.
- data — the aggregated frequency table.

#' Dominance–Entropy Frequency Plot

See Also

plot_circle

plot_circle

Examples

```
library(SummarizedExperiment)
library(airway)
data('airway')
se = airway

# Only use a random subset of 1000 rows
set.seed(123)
idx <- sample(seq_len(nrow(se)), size = min(1000, nrow(se)))
se <- se[idx, ]

## Normalize the data first using tpm_normalization
rowData(se)$gene_length = rowData(se)$gene_seq_end
- rowData(se)$gene_seq_start
```

```
se = tpm_normalization(se, log_trans = TRUE, new_assay_name = 'tpm_norm')

## Creating a plot_circle list using the 'gene_biotype' column as factor
plot_test <- plot_circle(
 x = se,
 n = 8,
 column_variable_factor = 'gene_biotype',
 entropyrange     = c(0,Inf),
 magnituderange   = c(0, Inf),
 label  = 'legend',
 output_table = TRUE,
 assay_name = 'tpm_norm'
)

## Using the plot_test object created above
## Default
plot <- plot_circle_frequency(n = 8,
                     circle = plot_test,
                     single = TRUE,
                     legend = TRUE,
                     numb_columns = 1,
                     filter_class = NULL,
                     point_size = 2)

plot[[1]]

## Facetting by factor is possible, adjusting the number of columns
plot <- plot_circle_frequency(n = 8,
                     circle = plot_test,
                     single = FALSE,
                     legend = TRUE,
                     numb_columns = 3,
                     filter_class = NULL,
                     point_size = 2)

plot[[1]]

## Subsetting by a specific class present in Factor
plot_circle_frequency(n = 8,
                     circle = plot_test,
                     single = FALSE,
                     legend = TRUE,
                     numb_columns = 1,
                     filter_class = c('protein_coding', 'snoRNA', 'miRNA'),
                     point_size = 2)

plot[[1]]

library(SummarizedExperiment)
library(airway)
data('airway')
se = airway
```

```
# Only use a random subset of 1000 rows
set.seed(123)
idx <- sample(seq_len(nrow(se)), size = min(1000, nrow(se)))
se <- se[idx, ]

## Normalize the data first using tpm_normalization
rowData(se)$gene_length = rowData(se)$gene_seq_end
- rowData(se)$gene_seq_start

se = tpm_normalization(se, log_trans = TRUE, new_assay_name = 'tpm_norm')

## Creating a plot_circle list using the 'gene_biotype' column as factor
plot_test <- plot_circle(
 x = se,
 n = 8,
 column_variable_factor = 'gene_biotype',
 entropyrange     = c(0,Inf),
 magnituderange   = c(0, Inf),
 label  = 'legend',
 output_table = TRUE,
 assay_name = 'tpm_norm'
)

## Using the plot_test object created above
## Default
plot <- plot_circle_frequency(n = 8,
                     circle = plot_test,
                     single = TRUE,
                     legend = TRUE,
                     numb_columns = 1,
                     filter_class = NULL,
                     point_size = 2)

plot[[1]]

## Facetting by factor is possible, adjusting the number of columns
plot <- plot_circle_frequency(n = 8,
                     circle = plot_test,
                     single = FALSE,
                     legend = TRUE,
                     numb_columns = 3,
                     filter_class = NULL,
                     point_size = 2)

plot[[1]]

## Subsetting by a specific class present in Factor
plot_circle_frequency(n = 8,
                     circle = plot_test,
                     single = FALSE,
                     legend = TRUE,
                     numb_columns = 1,
                     filter_class = c('protein_coding', 'snoRNA', 'miRNA'),
```

```
                    point_size = 2)

plot[[1]]
```

---

plot_rope *Rope (binary) dominance plot*

Description

Creates a rope-like visualization comparing two numeric columns (e.g., "a" vs. "b"), with optional color filtering based on maximum value range and entropy range.

The plot is useful for visualising “winner-takes-all” behaviour in two-way comparisons, e.g. gene expression in *A* and *B* conditions.

Usage

```
plot_rope(
  x,
  column_name = NULL,
  push_text = 1,
  rope_width = 1,
  rope_color = "#CCCCCCCC",
  rope_border = TRUE,
  col = c("red", "blue"),
  col_bg = "whitesmoke",
  pch = c(21, 21),
  pch_bg = 19,
  cex = 1,
  entropyrange = c(0, Inf),
  maxvaluerange = c(0, Inf),
  plotAll = TRUE,
  label = TRUE,
  output_table = TRUE,
  assay_name = NULL
)
```

Arguments

x	A data.frame or matrix with numeric columns, or a SummarizedExperiment containing such data in one of its assays.
column_name	Character. The name of the two variables that will be used for the analysis. By default it is NULL.
push_text	Numeric. Expands or contracts text label positions along the x-axis.
rope_width	Numeric. Thickness of the "rope" drawn in the center.
rope_color	Character. Color for the rope’s fill.

rope_border	Logical or a color. Whether or how to draw the rope border.
col	Character vector of length 2. Colors assigned when a > b or b > a, respectively.
col_bg	Background color (used when a row is filtered out by entropy or max value).
pch	Integer or vector specifying point types for the main two categories.
pch_bg	Integer specifying the point type for the "gray" points (if plotAll=TRUE).
cex	Numeric. Expansion factor for point size.
entropyrange	Numeric vector of length 2. Rows with entropy outside this range become background color.
maxvaluerange	Numeric vector of length 2. Rows with max(a,b) outside this range become background color.
plotAll	Logical. If TRUE, also draw "filtered" points in col_bg color. If FALSE, only highlight active points.
label	Logical. If TRUE, label the two columns near the rope ends.
output_table	Logical. If TRUE, return the processed data frame with added columns.
assay_name	(SummarizedExperiment only) Name of the assay containing the 2-column data. If not specified, the first assay is used.

Details

The function expects two numeric columns. If the experiment has more than two columns, the name of the columns of interest can be specified by using the parameter column_name. If x is a SummarizedExperiment, it extracts the indicated assay and extracts the columns of interest

It also uses: - centmass() for computing comx. - entropy() for computing Shannon entropy, stored in the entropy column. Between two variables, entropy rangeS between 0 and 1.

The rope is drawn in the middle of the plot (the x-axis from -1 to 1, y = 0), with thickness rope_width. Points are scattered in comy direction for a bit of jitter within the rope.

Value

- If output_table=TRUE, returns a data frame with extra columns (comx, comy, color, maxvalue, entropy) used in the plot.
- If output_table=FALSE, invisibly returns NULL.

Examples

```
library(SummarizedExperiment)
library(airway)
data('airway')
se <- airway

# Only use a random subset of 1000 rows
set.seed(123)
idx <- sample(seq_len(nrow(se)), size = min(1000, nrow(se)))
se <- se[idx, ]

## Normalize the data first using tpm_normalization
```

```
rowData(se)$gene_length = rowData(se)$gene_seq_end
- rowData(se)$gene_seq_start

se <- tpm_normalization(se, log_trans = TRUE, new_assay_name = 'tpm_norm')

# ------------------------------
# 1) Using a data.frame
# ------------------------------

df <- assay(se, 'tpm_norm')
df <- as.data.frame(df)

# Choose two columns of interest, in this case 'SRR1039508'
# and SRR1039516'

# Default Behaviour
plot_rope(df,
          column_name = c("SRR1039508", "SRR1039516"),
          output_table = FALSE)

# Colors can be modified
plot_rope(df,
          column_name = c("SRR1039508", "SRR1039516"),
          output_table = FALSE,
          col = c('darkgreen', 'darkred'))

# Emphasis can be applied to highly dominant variables by controling
# entropy parameter,
# values outside of that range will be colored smokewhite.
plot_rope(df,
          column_name = c("SRR1039508", "SRR1039516"),
          output_table = FALSE,
          col = c('darkgreen', 'darkred'),
          entropyrange = c(0,0.1))

# Points in the center are a reflection of genes with expression levels = 0.
# This can be modified by adjusting the maxvalue range

plot_rope(df,
          column_name = c("SRR1039508", "SRR1039516"),
          output_table = FALSE,
          col = c('darkgreen', 'darkred'),
          entropyrange = c(0,0.1),
          maxvaluerange = c(2, Inf))

# By controling entropy range, you can observe different types of genes.
# Values closer to 0 represent dominance and closer to 1 shareness.

# Exploring genes with high normalized expression values across different
#' entropy ranges
```

```
# Looking for genes with a Log2(TPM) score between 4 and 8
plot_rope(df,
          column_name = c("SRR1039508", "SRR1039516"),
          output_table = FALSE,
          col = c('darkgreen', 'darkred'),
          entropyrange = c(0,0.1),
          maxvaluerange = c(4, 8))


plot_rope(df,
          column_name = c("SRR1039508", "SRR1039516"),
          output_table = FALSE,
          col = c('darkgreen', 'darkred'),
          entropyrange = c(0.1,0.8),
          maxvaluerange = c(4, 8))


plot_rope(df,
          column_name = c("SRR1039508", "SRR1039516"),
          output_table = FALSE,
          col = c('darkgreen', 'darkred'),
          entropyrange = c(0.8,1),
          maxvaluerange = c(4, 8))

# ------------------------------
# 1) Using a SummarizedExperiment
# ------------------------------

plot_rope(se,
          column_name = c("SRR1039508", "SRR1039516"),
          output_table = FALSE,
          col = c('lightgreen', 'indianred'),
          entropyrange = c(0,0.1),
          maxvaluerange = c(4, 8))


plot_rope(se,
          column_name = c("SRR1039508", "SRR1039516"),
          output_table = FALSE,
          col =c('lightgreen', 'indianred'),
          entropyrange = c(0.1,0.8),
          maxvaluerange = c(4, 8))


plot_rope(se,
          column_name = c("SRR1039508", "SRR1039516"),
          output_table = FALSE,
          col = c('lightgreen', 'indianred'),
          entropyrange = c(0.8,1),
          maxvaluerange = c(4, 8))

### Obtaining the DF output for the analysis
```

```
object <- plot_rope(se,
                    column_name = c("SRR1039508", "SRR1039516"),
                    output_table = TRUE,
                    col = c('lightgreen', 'indianred'),
                    entropyrange = c(0.8,1),
                    maxvaluerange = c(4, 8))

head(object)
```

---

plot_triangle *Triangle (ternary) dominance plot*

Description

Creates a triangular (ternary) scatter plot for **three** numeric variables Each point is coloured by the variable with the largest value and can be filtered by (i) Entropy score ranging from (0 to 1.585) and (ii) overall score

The plot is useful for visualising "winner-takes-all" behaviour in three-way comparisons, e.g. gene expression in *A*, *B*, *C* conditions.

Usage

```
plot_triangle(
  x,
  column_name = NULL,
  entropyrange = c(0, Inf),
  maxvaluerange = c(0, Inf),
  col = c("darkred", "darkgreen", "darkblue"),
  background_col = "whitesmoke",
  output_table = TRUE,
  plotAll = TRUE,
  cex = 1,
  pch = 16,
  assay_name = NULL,
  label = TRUE,
  push_text = 1
)
```

Arguments

x	A numeric data.frame/matrix **or** a SummarizedExperiment.
column_name	Character. Names (or indices) of the three columns to visualise. If NULL, the first three numeric columns are used.
entropyrange	Numeric. Keep points whose entropy lies inside this interval. Default is c(0,Inf)
maxvaluerange	Numeric. Keep points whose values lies inside this interval. Default is c(0,Inf)

col	Character. Colors for each variable.
background_col	Character. Color for the observations outside entropyrange and maxvaluerange
output_table	Logical. If TRUE returns the processed data frame.
plotAll	Logical. If TRUE, filtered points are shown in background_col; if FALSE, they are omitted.
cex, pch	Base-graphics point size / symbol.
assay_name	(SummarizedExperiment only) Which assay to use. Default: the first assay.
label	Logical. If TRUE, label the vertices of the triangle
push_text	Numeric. Expands or contracts text label positions.

Details

The function expects three numeric columns. If the experiment has more than three columns, the name of the columns of interest can be specified by using the parameter column_name. If x is a SummarizedExperiment, it extracts the indicated assay and extracts the columns of interest

It also uses: - centmass() for computing comx and comy. - entropy() for computing Shannon entropy, stored in the entropy column. Between three variables, entropy rangeS between 0 and 1.585.

The ternary vertices are fixed at $(\sin(0), \cos(0))$, $(\sin(2\pi/3), \cos(2\pi/3))$ and $(\sin(4\pi/3), \cos(4\pi/3))$.

Value

If output_table = TRUE, a data.frame with the original three columns plus:

- comx, comy — Cartesian coordinates in the triangle;
- color — final plotting colour;
- entropy — Entropy scores for each gene;
- max_counts — Maximum score across variables

Examples

```
library(SummarizedExperiment)
library(airway)
data('airway')
se <- airway

# Only use a random subset of 1000 rows
set.seed(123)
idx <- sample(seq_len(nrow(se)), size = min(1000, nrow(se)))
se <- se[idx, ]

## Normalize the data first using tpm_normalization
rowData(se)$gene_length <- rowData(se)$gene_seq_end -
                           rowData(se)$gene_seq_start

se <- tpm_normalization(se, log_trans = TRUE, new_assay_name = 'tpm_norm')
```

```
# ------------------------------
# 1) Using a data.frame
# ------------------------------

df <- assay(se, 'tpm_norm') |> as.data.frame()


# Choose three columns of interest, in this case 'SRR1039508', 'SRR1039516'
# and 'SRR1039512'

# Default Behaviour
plot_triangle(x = df,
              column_name = c("SRR1039508", "SRR1039516", 'SRR1039512'),
              output_table = FALSE)

# Colors can be modified
plot_triangle(x = df,
              column_name = c("SRR1039508", "SRR1039516", 'SRR1039512'),
              output_table = FALSE,
              col = c('indianred', 'lightgreen', 'lightblue'))

# Emphasis can be applied to highly dominant variables by controling
# entropy parameter,
# values outside of that range will be colored smokewhite.
plot_triangle(x = df,
              column_name = c("SRR1039508", "SRR1039516", 'SRR1039512'),
              output_table = FALSE,
              col = c('indianred', 'lightgreen', 'lightblue'),
              entropyrange = c(0, 0.1))

# Points in the center are a reflection of genes with expression levels = 0.
# This can be modified by adjusting the maxvalue range

plot_triangle(x = df,
              column_name = c("SRR1039508", "SRR1039516", 'SRR1039512'),
              output_table = FALSE,
              col = c('indianred', 'lightgreen', 'lightblue'),
              entropyrange = c(0, 0.1),
              maxvaluerange = c(0.1, Inf))

# By controling entropy range, you can observe different types of genes.
# Values closer to 0 represent dominance and closer to 1.6 shareness.

plot_triangle(x = df,
              column_name = c("SRR1039508", "SRR1039516", 'SRR1039512'),
              output_table = FALSE,
              col = c('indianred', 'lightgreen', 'lightblue'),
              entropyrange = c(0, 0.4),
              maxvaluerange = c(0.1, Inf))

plot_triangle(x = df,
              column_name = c("SRR1039508", "SRR1039516", 'SRR1039512'),
              output_table = FALSE,
```

```
              col = c('indianred', 'lightgreen', 'lightblue'),
              entropyrange = c(0.4, 1.3),
              maxvaluerange = c(0.1, Inf))

plot_triangle(x = df,
              column_name = c("SRR1039508", "SRR1039516", 'SRR1039512'),
              output_table = FALSE,
              col = c('indianred', 'lightgreen', 'lightblue'),
              entropyrange = c(1.3, Inf),
              maxvaluerange = c(0.1, Inf))

# Same analysis can be performed by filtering out genes with low expression
# values

plot_triangle(x = df,
              column_name = c("SRR1039508", "SRR1039516", 'SRR1039512'),
              output_table = FALSE,
              col = c('indianred', 'lightgreen', 'lightblue'),
              entropyrange = c(1.2, Inf),
              maxvaluerange = c(2, Inf))

plot_triangle(x = df,
              column_name = c("SRR1039508", "SRR1039516", 'SRR1039512'),
              output_table = FALSE,
              col = c('indianred', 'lightgreen', 'lightblue'),
              entropyrange = c(1.2, Inf),
              maxvaluerange = c(5, Inf))

plot_triangle(x = df,
              column_name = c("SRR1039508", "SRR1039516", 'SRR1039512'),
              output_table = FALSE,
              col = c('indianred', 'lightgreen', 'lightblue'),
              entropyrange = c(1.2, Inf),
              maxvaluerange = c(10, Inf))

# Background points can be removed
plot_triangle(x = df,
              column_name = c("SRR1039508", "SRR1039516", 'SRR1039512'),
              output_table = FALSE,
              col = c('indianred', 'lightgreen', 'lightblue'),
              entropyrange = c(1.2, Inf),
              maxvaluerange = c(2, Inf),
              plotAll = FALSE)
# ------------------------------
# 1) Using a SummarizedExperiment
# ------------------------------


plot_triangle(x = se,
              column_name = c("SRR1039508", "SRR1039516", 'SRR1039512'),
              output_table = FALSE,
              col = c('darkred', 'darkgreen', 'darkblue'),
              entropyrange = c(0, 0.4),
```

```
                maxvaluerange = c(0.1, Inf),
                assay_name = 'tpm_norm')

plot_triangle(x = se,
                column_name = c("SRR1039508", "SRR1039516", 'SRR1039512'),
                output_table = FALSE,
                col = c('darkred', 'darkgreen', 'darkblue'),
                entropyrange = c(0.4, 1.3),
                maxvaluerange = c(0.1, Inf),
                assay_name = 'tpm_norm')

plot_triangle(x = se,
                column_name = c("SRR1039508", "SRR1039516", 'SRR1039512'),
                output_table = FALSE,
                col = c('darkred', 'darkgreen', 'darkblue'),
                entropyrange = c(1.3, Inf),
                maxvaluerange = c(0.1, Inf),
                assay_name = 'tpm_norm')


### Obtaining the DF output for the analysis

object = plot_triangle(x = se,
                        column_name = c("SRR1039508", "SRR1039516",
                        'SRR1039512'),
                        output_table = TRUE,
                        col = c('darkred', 'darkgreen', 'darkblue'),
                        entropyrange = c(1.3, Inf),
                        maxvaluerange = c(0.1, Inf),
                        assay_name = 'tpm_norm')

head(object)
```

`Qentropy` *Compute Q-Entropy using existing row-normalized data + Entropy*

Description

#' Transform entropy scores into categorical entropy scores $Q_{ij} = \text{Entropy}_i - \log_2(x_{ij})$, or `Inf` if $x_{ij} == 0$.

@details For each row i and column j, Q_{ij} is defined as $\text{Entropy}_i - \log_2\big(x_{ij}\big)$ if x_{ij} is positive, or `Inf` otherwise.

Usage

```
Qentropy(x, assay_name = "Entropy", new_assay_name = "Qentropy")
```

Arguments

x	A data.frame (already processed by 'entropy()') or a SummarizedExperiment (already processed by 'entropy()').
assay_name	(SummarizedExperiment only) The name of the assay whose row-normalized data will be replaced by Q-values. If NULL, uses the first assay.
new_assay_name	If you prefer to store Q-values in a *new* assay, provide a name. By default 'Qentropy'

Value

- If x is a data.frame: returns the same data.frame with numeric columns replaced by Q_{ij} values and Entropy column removed.
- If x is a SummarizedExperiment: returns the same object with the specified assay replaced by Q_{ij} values (or a new assay if new_assay_name is set) and rowData(x)$Entropy removed.

Examples

```
library(SummarizedExperiment)
library(airway)
data('airway')

se = airway

# Only use a random subset of 1000 rows
set.seed(123)
idx <- sample(seq_len(nrow(se)), size = min(1000, nrow(se)))
se <- se[idx, ]

# --------------------------------
# 1) Using a data.frame
# --------------------------------
df = assay(se) |> as.data.frame()

## Entropy needs to be calculated first
df = entropy(df)

## Then you can apply the Qentropy function
df = Qentropy(df)

head(df)

# --------------------------------
# 2) Using a SummarizedExperiment
# --------------------------------

## Calculate Entropy first
se2 = entropy(se, new_assay_name = 'Entropy')

## Transform entropy into Qentropy. new_assay_name specify a new assay
## where data is going to be stored. Assay_name must have Entropy transformed
```

```
values
## By default, the function will look for an assay_name 'Entropy' and assign
## a new assay to 'Qentropy'
se2 = Qentropy(se2, new_assay_name = 'Qentropy', assay_name = 'Entropy')

se2
```

---

quantile_normalization *Quantile Normalization*

Description

Normalizes read counts by the quantile normalization method:

1. Each sample (column) is sorted, and values at each rank are averaged across columns
2. Each sample's values are replaced with the average of their respective rank
3. If `log_trans = TRUE`, applies `log2(QN + 1)` transformation

Usage

```
quantile_normalization(
  x,
  log_trans = FALSE,
  assay_name = NULL,
  new_assay_name = NULL
)
```

Arguments

`x`	A numeric `matrix` or `data.frame` of gene counts, or a `SummarizedExperiment` containing such counts. **If a** `SummarizedExperiment`**,** the function applies normalization to the specified assay (via `assay_name`). **If a** `data.frame`/`matrix`**,** the normalization is applied directly.
`log_trans`	Logical. If `TRUE`, apply `log2(... + 1)` transform to the quantile-normalized values.
`assay_name`	If `x` is a SummarizedExperiment, name of the assay to normalize. Defaults to the first assay if not specified.
`new_assay_name`	If `x` is a SummarizedExperiment, name of a new assay in which to store the quantile-normalized (or log2-transformed) values. If `NULL`, overwrites the original assay.

Details

If x is a SummarizedExperiment, the function will extract the assay using assay_name, apply quantile normalization, and return a new or updated assay. If x is a matrix or data.frame, normalization is applied directly to the input matrix.

Value

A numeric **matrix** of quantile-normalized (or log2-normalized) values if x is a data.frame or matrix. If x is a SummarizedExperiment, returns the modified SummarizedExperiment with the normalized data placed in the existing or new assay.

Examples

```
library(SummarizedExperiment)
library(airway)
data('airway')

se <- airway

# Only use a random subset of 1000 rows
set.seed(123)
idx <- sample(seq_len(nrow(se)), size = min(1000, nrow(se)))
se <- se[idx, ]

# --------------------------------
# 1) Using a data.frame
# --------------------------------
df <- assay(se)

## Without log transformation
df_qn <- quantile_normalization(df, log_trans = FALSE)
df_qn[1:5, 1:5]

## With log transformation
df_qn_log <- quantile_normalization(df, log_trans = TRUE)
df_qn_log[1:5, 1:5]

# --------------------------------
# 2) Using a SummarizedExperiment
# --------------------------------

## Overwrite existing assay
se2 <- quantile_normalization(se)
assay(se2)[1:5, 1:5]

## Store result in new assay
se3 <- quantile_normalization(se, new_assay_name = "quant_norm")
assay(se3, "quant_norm")[1:5, 1:5]

## Use specific input assay (simulate new one)
new_matrix <- matrix(
  data = sample(x = seq(1, 100000), size = nrow(se) * ncol(se),
```

```
  replace = TRUE),
  nrow = nrow(se),
  ncol = ncol(se)
)
rownames(new_matrix) <- rownames(se)
colnames(new_matrix) <- colnames(se)

## Create a new assay in the SummarizedExperiment
assay(se, "new_counts") <- new_matrix

## Normalize the new assay and store it under a new name
se4 <- quantile_normalization(se, assay_name = "new_counts",
new_assay_name = "quant_new")
assay(se4, "quant_new")[1:5, 1:5]
```

`rpkm_normalization` *RPKM Normalization*

Description

Normalizes read counts by the RPKM (Reads Per Kilobase per Million mapped reads) method:

1. Normalize counts by library size (column sums), scaled to millions.
2. Divide each gene's value by its length in kilobases.
3. If `log_trans = TRUE`, applies `log2(RPKM + 1)`.

Usage

```
rpkm_normalization(
  x,
  gene_length = NULL,
  log_trans = FALSE,
  assay_name = NULL,
  new_assay_name = NULL
)
```

Arguments

`x`	A numeric `matrix` or `data.frame` of gene counts, or a `SummarizedExperiment` containing such counts. **If a** `SummarizedExperiment`**,** the function retrieves `gene_length` from `rowData(x)$gene_length`. **If a** `data.frame`**/**`matrix`**,** the user must provide the `gene_length` argument.
`gene_length`	A numeric vector of gene lengths (one per row), used only if `x` is a data.frame or matrix. Must match the number of rows in `x`. Ignored if `x` is a SummarizedExperiment.
`log_trans`	Logical. If `TRUE`, apply `log2(... + 1)` transform to the RPKM-normalized values.

assay_name If x is a SummarizedExperiment, name of the assay to normalize. Defaults to the first assay if not specified.

new_assay_name If x is a SummarizedExperiment, name of a new assay in which to store the RPKM (or log2-RPKM). If NULL, overwrites the assay specified in assay_name.

Details

If x is a SummarizedExperiment, the function looks for a numeric column named "gene_length" in rowData(x). That column must have length equal to the number of rows in the assay being normalized.

Value

A numeric **matrix** of RPKM or log2(RPKM + 1) values if x is a data.frame or matrix. If x is a SummarizedExperiment, returns the modified SummarizedExperiment with the RPKM data placed in the existing or new assay.

Examples

```
library(SummarizedExperiment)
library(airway)
data('airway')

se = airway

# Only use a random subset of 1000 rows
set.seed(123)
idx <- sample(seq_len(nrow(se)), size = min(1000, nrow(se)))
se <- se[idx, ]

### Adding a column in rowData regarding the gene_length
rowData(se)$gene_length = rowData(se)$gene_seq_end -
rowData(se)$gene_seq_start

# ------------------------------
# 1) Using a data.frame
# ------------------------------

gene_length = rowData(se)$gene_length
df = assay(se)

## Without log transformation
df = rpkm_normalization(df, gene_length = gene_length)
df[1:5, 1:5]

## With log transformation
df = rpkm_normalization(df, gene_length = gene_length, log_trans = TRUE)
df[1:5, 1:5]

# ------------------------------
# 2) Using a SummarizedExperiment
# ------------------------------
```

```
# If no new_assay_name is provided, then overwrites existing assay
se2 = rpkm_normalization(se, log_trans = FALSE)
head(assay(se2))

# If new_assay_name is given, normalization stored in a new assay
se2 = rpkm_normalization(se, log_trans = FALSE, new_assay_name =
'rpkm_counts')
head(assay(se2, 'rpkm_counts'))

# Creating a new assay to test specific input
new_matrix = matrix(data = sample(x = seq(1, 100000),
                                  size = nrow(se) * ncol(se),
                                  replace = TRUE),
                    nrow = nrow(se),
                    ncol = ncol(se))
rownames(new_matrix) = rownames(se)
colnames(new_matrix) = colnames(se)

assay(se, 'new_counts') = new_matrix
se2 = rpkm_normalization(se, new_assay_name = 'rpkm_counts_new',
assay_name = 'new_counts')
head(assay(se2, 'rpkm_counts_new'))
```

`tpm_normalization` *TPM Normalization*

Description

Normalizes read counts by the TPM (Transcripts Per Million) method:

1. If `log_trans = TRUE`, applies `log2(TPM + 1)`.

Usage

```
tpm_normalization(
  x,
  gene_length = NULL,
  log_trans = FALSE,
  assay_name = NULL,
  new_assay_name = NULL
)
```

Arguments

`x`	A numeric `matrix` or `data.frame` of gene counts, or a `SummarizedExperiment` containing such counts. **If a** `SummarizedExperiment`**,** the function retrieves `gene_length` from `rowData(x)$gene_length`.

If a `data.frame`/`matrix`**,** the user must provide the `gene_length` argument.

`gene_length` A numeric vector of gene lengths (one per row), used only if `x` is a data.frame or matrix. Must match the number of rows in `x`. Ignored if `x` is a SummarizedExperiment.

`log_trans` Logical. If `TRUE`, apply `log2(... + 1)` transform to the TPM-normalized values.

`assay_name` If `x` is a SummarizedExperiment, name of the assay to normalize. Defaults to the first assay if not specified.

`new_assay_name` If `x` is a SummarizedExperiment, name of a new assay in which to store the TPM (or log2-TPM). If `NULL`, overwrites the assay specified in `assay_name`.

Details

If `x` is a `SummarizedExperiment`, this function looks for a numeric column named `"gene_length"` in `rowData(x)`. That column must have length equal to the number of rows in the assay being normalized.

Value

A numeric **matrix** of TPM or log2(TPM + 1) values if `x` is a data.frame or matrix. If `x` is a SummarizedExperiment, returns the modified SummarizedExperiment with the TPM data placed in the existing or new assay.

Examples

```
library(SummarizedExperiment)
library(airway)
data('airway')

se = airway

# Only use a random subset of 1000 rows
set.seed(123)
idx <- sample(seq_len(nrow(se)), size = min(1000, nrow(se)))
se <- se[idx, ]

### Adding a column in rowData regarding the gene_length
rowData(se)$gene_length = rowData(se)$gene_seq_end
- rowData(se)$gene_seq_start

# ------------------------------
# 1) Using a data.frame
# ------------------------------

gene_length = rowData(se)$gene_length

df = assay(se)

## Without log transformation
df = tpm_normalization(df, gene_length = gene_length)
```

```
df[1:5, 1:5]

## With log transformation
df = tpm_normalization(df, gene_length = gene_length, log_trans = TRUE)

df[1:5, 1:5]

# ------------------------------
# 2) Using a SummarizedExperiment
# ------------------------------

# If now new_assay_name is provided, then overwrites existing assay
se2 = tpm_normalization(se, log_trans = FALSE)

head(assay(se2))

# If new new_assay_name, normalization stored in a new object
se2 = tpm_normalization(se, log_trans = FALSE, new_assay_name = 'tpm_counts')

head(assay(se2, 'tpm_counts'))

# A specific assay can also be selected
new_matrix =  matrix(data = sample(x = seq(1, 100000),
                                   size = nrow(se) * ncol(se),
                                   replace = TRUE),
                     nrow = nrow(se),
                     ncol = ncol(se))
rownames(new_matrix) = rownames(se)
colnames(new_matrix) = colnames(se)

## Creating a new assay called new counts
assay(se, 'new_counts') = new_matrix

se2 = tpm_normalization(se, new_assay_name = 'tpm_counts_new',
assay_name = 'new_counts')

se2

head(assay(se2, 'tpm_counts_new'))
```

Index

centmass, 2
cpm_normalization, 4

entropy, 6

ggplot, *11*, *18*

minmax_normalization, 7

plot_circle, 10, *17, 18*
plot_circle_frequency, 17
plot_rope, 21
plot_triangle, 25

Qentropy, 29
quantile_normalization, 31

rpkm_normalization, 33

tpm_normalization, 35