Package 'MetaboDynamics'

February 23, 2025

Title Bayesian analysis of longitudinal metabolomics data

Version 0.99.20

URL https://github.com/KatjaDanielzik/MetaboDynamics

BugReports https://github.com/KatjaDanielzik/MetaboDynamics/issues

Description MetaboDynamics is an R-package that provides a framework of probabilistic models to analyze longitudinal metabolomics data. It enables robust estimation of mean concentrations despite varying spread between timepoints and reports differences between timepoints as well as metabolite specific dynamics profiles that can be used for identifying ``dynamics clusters'' of metabolites of similar dynamics. Provides probabilistic over-representation analysis of KEGG functional modules and pathways as well as comparison between clusters of different experimental conditions.

License GPL (>= 3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Suggests knitr, rmarkdown, BiocStyle, testthat (>= 3.0.0)

Config/testthat/edition 3

Imports dplyr, ggplot2, KEGGREST, methods, Rcpp (>= 0.12.0), RcppParallel (>= 5.0.1), rstan (>= 2.18.1), rstantools (>= 2.4.0), S4Vectors, stringr, SummarizedExperiment, tidyr

VignetteBuilder knitr

Depends R (>= 4.4.0)

LazyData false

LinkingTo BH (>= 1.66.0), Rcpp (>= 0.12.0), RcppEigen (>= 0.3.3.3.0), RcppParallel (>= 5.0.1), rstan (>= 2.18.1), StanHeaders (>= 2.18.0)

SystemRequirements GNU make

biocViews Software,Metabolomics,Bayesian,FunctionalPrediction,MultipleComparison,KEGG

Biarch true

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

git_branch devel

git_last_commit cddeae6

git_last_commit_date 2025-02-14

Repository Bioconductor 3.21

Date/Publication 2025-02-23

Author Katja Danielzik [aut, cre] (ORCID:
<https://orcid.org/0009-0007-5021-6212>),
Simo Kitanovski [ctb] (ORCID: <https://orcid.org/0000-0003-2909-5376>),
Johann Matschke [ctb] (ORCID: <https://orcid.org/0000-0003-4878-8741>),
Daniel Hoffmann [ctb] (ORCID: <https://orcid.org/0000-0003-2973-7869>)

Maintainer Katja Danielzik <katja.danielzik@uni-due.de>

Contents

`MetaboDynamics-package` *MetaboDynamics: Bayesian analysis of longitudinal metabolomics data*

Description

MetaboDynamics is an R-package that provides a framework of probabilistic models to analyze longitudinal metabolomics data. It enables robust estimation of mean concentrations despite varying spread between timepoints and reports differences between timepoints as well as metabolite specific dynamics profiles that can be used for identifying "dynamics clusters" of metabolites of similar dynamics. Provides probabilistic over-representation analysis of KEGG functional modules and pathways as well as comparison between clusters of different experimental conditions.

Author(s)

Maintainer: Katja Danielzik <katja.danielzik@uni-due.de> (ORCID)

Other contributors:

- Simo Kitanovski (ORCID) [contributor]
- Johann Matschke (ORCID) [contributor]
- Daniel Hoffmann (ORCID) [contributor]

See Also

Useful links:

- https://github.com/KatjaDanielzik/MetaboDynamics
- Report bugs at https://github.com/KatjaDanielzik/MetaboDynamics/issues

---

`.calculate_distances` *compare_dynamics()*

Description

compare_dynamics()

Usage

```
.calculate_distances(group_a, group_b, dynamics)
```

Arguments

`group_a`	dataframe of one cluster of one condition
`group_b`	dataframe of one cluster of a differenct condition than group_a
`dynamics`	character vector specifying the columns that hold dynamic estimates in data

Value

matrix of pairwise euclidean distances between two groups of vectors

---

`.calculate_jaccard` *Function to calculate Jaccard index on two character vectors of metabolite names*

Description

Function to calculate Jaccard index on two character vectors of metabolite names

Usage

```
.calculate_jaccard(group_a, group_b)
```

Arguments

`group_a`	group of clusters of metabolites
`group_b`	group of clusters of metabolites

Value

the Jaccard index

`.eu` *euclidean distance compare_dynamics()*

Description

euclidean distance compare_dynamics()

Usage

```
.eu(a, b)
```

Arguments

`a`	a numeric vector
`b`	a numeric vector of same length as a

Value

euclidean distance between vectors

`.similarity` *Jaccard Index: intersection/union compare_metabolites()*

Description

Jaccard Index: intersection/union compare_metabolites()

Usage

```
.similarity(a, b)
```

Arguments

a	a vector
b	a vector

Value

Jaccard Index of a and b

`compare_dynamics` *Comparison of metabolite dynamics clusters under different experimental conditions*

Description

Employs a Bayesian model that assumes a normal distribution of Euclidean distances between dynamics vectors (metabolite concentrations at different time points) of two clusters that come from different experimental conditions to estimate the mean distance between clusters.

Usage

```
compare_dynamics(data, dynamics, cores = 4)
```

Arguments

data	a dataframe or containing a column specifying the metabolite names to be compared and cluster IDs (column named "cluster") of clusters of similar dynamics, as well as a column "condition" specifying the experimental conditions. to be compared or a SummarizedExperiment storing the same information in metadata(data) under "cluster"
dynamics	vector specifying the column names of dataframe clusters that hold the dynamics information
cores	how many cores should be used for model fitting; this parallelizes the model fitting and therefore speeds it up; default=4

Value

a list holding a 1) the model fit 2) dataframe of estimates of the mean distance between #' clusters of different experimental conditions ("mean") and the standard deviation ("sigma"). If data input was a SummarizedExperiment results are stored in metadata(data) under "comparison_dynamics"

See Also

Visualization of estimates `heatmap_dynamics()`/ compare metabolite composition of clusters `compare_metabolites()`

Examples

```
data("longitudinalMetabolomics")
longitudinalMetabolomics <- compare_dynamics(
  data = longitudinalMetabolomics,
  dynamics = c("mu1_mean", "mu2_mean", "mu3_mean", "mu4_mean"),
  cores = 1
)
S4Vectors::metadata(longitudinalMetabolomics)[["comparison_dynamics"]]
```

---

compare_metabolites *Comparison of metabolite sets between dynamics clusters of different experimental conditions*

Description

Uses the Jaccard Index to compare metabolite names between dynamics clusters of different experimental conditions

Usage

```
compare_metabolites(data, metabolite = "metabolite")
```

Arguments

`data`	a dataframe or containing a column specifying the metabolite names to be compared and cluster IDs (column named "cluster") of clusters of similar dynamics, as well as a column "condition" specifying the experimental conditions. to be compared or a SummarizedExperimentstoring the same information in metadata(data) under "cluster"
`metabolite`	column in "data" that specifies either metabolite name or KEGG ID or some other identifier

Value

a dataframe of Jaccard indices between data or if data input was a SummarizedExperiment results are stored in metadata(data) under "comparison_metabolites"

See Also

Visualization of metabolite similarity `heatmap_metabolites()`/ compare dynamics of clusters `compare_dynamics()`

Examples

```
data("longitudinalMetabolomics")
longitudinalMetabolomics <- compare_metabolites(
  data = longitudinalMetabolomics
)
S4Vectors::metadata(longitudinalMetabolomics)[["comparison_metabolites"]]
```

---

diagnostics_dynamics *Extracts diagnostic criteria from numeric fit of Bayesian model of dynamics*

Description

gathers number of divergences, rhat values, number of effective samples (n_eff) and provides plots for diagnostics criteria as well as posterior predictive checks. Dataframe "model_diagnostics" contains information about experimental condition, number of divergent transitions and rhat and neff values for all timepoints.

Usage

```
diagnostics_dynamics(
  data,
  assay = "scaled_log",
  iter = 2000,
  warmup = iter/4,
  chains = 4,
  fits = metadata(data)[["dynamic_fits"]]
)
```

Arguments

`data`	dataframe or a SummarizedExperiment used to fit dynamics model column of "time" that contains time as numeric
`assay`	of the SummarizedExperiment object that was used to fit the dynamics model
`iter`	number of iterations used for model fit
`warmup`	number of warmup iterations used for model fit
`chains`	number of chains used for model fit
`fits`	list of model fits for which diagnostics should be extracted, is the object that gets returned by fit_dynamics_model(), or if a summarizedExperiment object the results of fit_dynamics_model() are stored in metadata(data) under "dynamic_fits"

Value

a list which contains diagnostics criteria of all conditions in a dataframe (named "model_diagnostics") and one dataframe per condition that contains necessary information for Posterior predictive check (named "PPC_condition"). If data is a summarizedExperiment object the diagnostics are stored in metadata(data) "diagnostics_dynamics"

See Also

estimates_dynamics() parent function fit_dynamics_model() visualization functions: plot_diagnostics()/plot_PPC

Examples

```
data("longitudinalMetabolomics")
data <- longitudinalMetabolomics[, longitudinalMetabolomics$condition == "A" &
  longitudinalMetabolomics$metabolite == "ATP"]
data <- fit_dynamics_model(
  data = data,
  scaled_measurement = "m_scaled", assay = "scaled_log",
  max_treedepth = 14, adapt_delta = 0.95, iter = 2000, cores = 1, chains = 1
)
data <- diagnostics_dynamics(
  data = data, assay = "scaled_log",
  iter = 2000, chains = 1,
  fits = metadata(data)[["dynamic_fits"]]
)
S4Vectors::metadata(data)[["diagnostics_dynamics"]][["model_diagnostics"]]
S4Vectors::metadata(data)[["diagnostics_dynamics"]][["posterior_A"]]
```

---

estimates_dynamics *Extracts parameter estimates from numeric fit of Bayesian model of dynamics*

Description

Extracts the mean concentrations (mu) at every timepoint from the dynamics model fit, the 95% highest density interval (HDI), the estimated standard deviation of metabolite concentrations at every time point (sigma), and the pooled standard deviation of every metabolite over all timepoints (lambda). Additionally samples from the posterior of mu can be drawn. This can be helpful if p.e. one wants to estimate the clustering precision. Lambda can be used for clustering algorithms such as VSClust that also take the variance into account.

Usage

```
estimates_dynamics(
  data,
  assay = "scaled_log",
  kegg = "KEGG",
  condition = "condition",
```

```
  time = "time",
  fits = metadata(data)[["dynamic_fits"]],
  iter = 2000,
  warmup = iter/4,
  chains = 4,
  samples = 1
)
```

Arguments

data	dataframe or colData of a SummarizedExperiment used used to fit dynamics model, must contain a column specifying KEGG IDs, column named "condition" specifiyng the experimental condition and a column named "time" specifying the timepoints. If it is a SummarizedExperiment object the dynamic fits must be stores in metadata(data) under "dynamic_fits"
assay	of the SummarizedExperiment object that was used to fit the dynamics model
kegg	column in "data" that contains the KEGG IDs or other identifier of metabolites
condition	name of column in dataframe data that specifies the experimental condition
time	column in "data" that contains the time point identifiers
fits	list of model fits for which estimates should be extracted
iter	how many iterations were used to fit the dynamics model
warmup	how many warm-up iterations were used to fit the dynamics model
chains	how many chains were used to fit the dynamics model
samples	how many posterior samples should be drawn (p.e. for check of clustering precision)

Value

a list of dataframes (one per experimental condition) that contains the estimates at the timepoints and samples from the posterior (number as specified in samples), delta_mu specifies the difference between time point specified in column "time.ID" and subsequent time point (delta_mu in row time.ID=1: mu(time point 2)- mu(time point 1)) if number of time points in dataset is >1 If data is a summarizedExperiment object the estimates are stored in metadata(data) under "estimates_dynamics"

See Also

Fit the dynamic model `fit_dynamics_model()`. Diagnostics of the dynamic model `diagnostics_dynamics()` Visualization of estimates with `plot_estimates()`

Examples

```
data("longitudinalMetabolomics")
data <- longitudinalMetabolomics[, longitudinalMetabolomics$condition == "A" &
  longitudinalMetabolomics$metabolite == "ATP"]
data <- fit_dynamics_model(
  data = data,
```

```
  scaled_measurement = "m_scaled", assay = "scaled_log",
  max_treedepth = 14, adapt_delta = 0.95, iter = 2000, cores = 1, chains = 1
)
data <- estimates_dynamics(
  data = data, iter = 2000,
  chains = 1, condition = "condition"
)
```

`fit_dynamics_model` *Fits dynamics model*

Description

Employs a hierarchical model that assumes a normal distribution of standardized (mean=0, sd=1) log(cpc) (cpc = concentration per cell) values for robust estimation of mean concentrations over time of single metabolites at single experimental conditions.

Usage

```
fit_dynamics_model(
  data,
  metabolite = "metabolite",
  time = "time",
  condition = "condition",
  scaled_measurement = "m_scaled",
  assay = "scaled_log",
  chains = 4,
  cores = 4,
  adapt_delta = 0.95,
  max_treedepth = 10,
  iter = 2000,
  warmup = iter/4
)
```

Arguments

`data`	concentration table with at least three replicate measurements per metabolites containing the columns "metabolite", "condition", and "m_scaled" by default or colData of a SummarizedExperiment object
`metabolite`	column of "data" that contains the metabolite names or IDs
`time`	column of "time" that contains time as numeric, make sure your time column is ordered from lowest to highest for the model to work
`condition`	column of "data" that contains the experimental conditions
`scaled_measurement`	column of "data" that contains the concentrations per cell, centered and normalized per metabolite and experimental condition (mean=0, sd=1)

assay	if input is a summarizedExperiment specify the assay that should be used for input, colData has to hold the columns, "condition" and "metabolite", rowData the timepoint specifications
chains	how many Markov-Chains should be used for model fitting, use at least two, default=4
cores	how many cores should be used for model fitting; this parallelizes the model fitting and therefore speeds it up; default=4
adapt_delta	target average acceptance probability, can be adapted if divergent transitions are reported, default is 0.95
max_treedepth	can be adapted if model throws warnings about hitting max_treedepth, warnings are mostly efficiency not validity concerns and treedepth can be raised, default=10
iter	how many iterations are run, increasing might help with effective sample size being to low, default=2000
warmup	how many iterations the model warms up for, increasing might facilitate efficiency, must be at least 25% of ITER, default=iter/4

Value

returns a list of model fits. One model fit named "condition" per experimental condition. If input is a summarizedExperiment object the dynamic fits are stored metadata(data) under "dynamic_fits"

See Also

Example data setlongitudinalMetabolomics. Get model diagnostics `diagnostics_dynamics()` Get model estimates `estimates_dynamics()`

Examples

```
data("longitudinalMetabolomics")
data <- longitudinalMetabolomics[, longitudinalMetabolomics$condition == "A" &
  longitudinalMetabolomics$metabolite == "ATP"]
data <- fit_dynamics_model(
  data = data,
  scaled_measurement = "m_scaled", assay = "scaled_log",
  max_treedepth = 14, adapt_delta = 0.95, iter = 2000, cores = 1, chains = 1
)
S4Vectors::metadata(data)[["dynamic_fits"]]
```

---

get_ORA_annotations *Retrieve background and annotation information for over-representation analysis (ORA)*

Description

Uses the package KEGGREST to retrieve background and annotation information needed for over-representation analysis. As KEGGREST only allows 10 queries per second this might take some time to run, depending on the size of the dataset and organism. The user should check afterwards if all functional modules are applicable for the analysis of the dataset (p.e. organism, tissue).

Usage

```
get_ORA_annotations(
  data,
  kegg = "KEGG",
  metabolite_name = "metabolite",
  update_background = FALSE
)
```

Arguments

data	data frame to be analyzed with ORA. Must at least contain a column with KEGG IDs or a SummarizedExperiment where the metabolite names or IDs are stored in colData
kegg	column name of "data" that holds KEGG IDs
metabolite_name	column name of "data" that holds metabolite names
update_background	logical. Should the background information be updated? If TRUE this may take some time.

Value

a list with dataframes "background" and "annotation" needed for ORA, if data is a SummarizedExperiment SummarizedExperiment object annotations are stored in metadata(data) under "KEGG_annotations"

See Also

Dor over-representation analysis of KEGG functional modules `ORA_hypergeometric()`

Examples

```
data("longitudinalMetabolomics")
data <- longitudinalMetabolomics[, longitudinalMetabolomics$condition == "A" &
  longitudinalMetabolomics$metabolite == "ATP"]
data <- get_ORA_annotations(
  data = data,
  kegg = "KEGG",
  metabolite_name = "metabolite",
  update_background = FALSE
)
S4Vectors::metadata(data)[["KEGG_annotations"]]
```

heatmap_dynamics *plot bubble heatmap from the numerical fit of compare_dynamics()*

Description

plot bubble heatmap from the numerical fit of compare_dynamics()

Usage

```
heatmap_dynamics(
  estimates = metadata(data)[["comparison_dynamics"]][["estimates"]],
  data
)
```

Arguments

estimates	dataframe of estimates of the mean distance between clusters of different experimental conditions ("mean") and the standard deviation ("sigma") obtain by function compare_dynamics()
data	a dataframe or containing a column specifying the metabolite names to be compared and cluster IDs (column named "cluster") of clusters of similar dynamics, as well as a column "condition" specifying the experimental conditions. to be compared or a SummarizedExperiment storing the same information in metadata(data) under "cluster"

Value

a bubble heat map where the color of the bubble represents the similarity of two clusters in regards to their dynamics in the color and the size the uncertainty of the similarity. Big bright bubbles mean high similarity with low uncertainty.

See Also

Do calculations for comparison of dynamics between clusters `compare_dynamics()`

Examples

```
data("longitudinalMetabolomics")
longitudinalMetabolomics <- compare_dynamics(
  data = longitudinalMetabolomics,
  dynamics = c("mu1_mean", "mu2_mean", "mu3_mean", "mu4_mean"),
  cores = 1
)
heatmap_dynamics(data = longitudinalMetabolomics)
```

`heatmap_metabolites` *plot heatmap from comparison of metabolite composition compare_metabolites()*

Description

plot heatmap from comparison of metabolite composition compare_metabolites()

Usage

```
heatmap_metabolites(
  distances = metadata(data)[["comparison_metabolites"]],
  data
)
```

Arguments

`distances`	dataframe of Jaccard indices between clusters obtained by function compare_metabolites(). If compare_metabolites() was executed on as SummarizedExperiment or a SummarizedExperimentthan this is stored in metadata(data) under "comparison_metabolites"
`data`	a dataframe containing the columns "metabolite" specifying the metabolite names to be compared and cluster IDs(column named "cluster") of clusters of similar dynamics, as well as a column "condition" specifying the experimental conditions to be compared

Value

a heatmap where the color of the tile represents the similarity of two clusters in regards to their metabolite composition. The brighter the color the more similar the metabolite compositions.

See Also

Do calculations for comparison of metabolites between clusters `compare_metabolites()`

Examples

```
data("longitudinalMetabolomics")
longitudinalMetabolomics <- compare_metabolites(
  data = longitudinalMetabolomics
)
heatmap_metabolites(data = longitudinalMetabolomics)
```

`longitudinalMetabolomics` *A simulated data set of longitudinal concentration tables of metabolites.*

Description

A simulated data set of 98 metabolites. 3 replicate measurements of 4 time points and at 3 experimental conditions. Metabolites are in 8 dynamics groups per experimental condition. 4 groups have varying dynamics between conditions. Is represented as a SummarizedExperiment object where concentration tables of each experimental condition are stored in assays (raw concentrations in "concentrations", log-transformed transformations in "log_con" and scaled log-transformed concentrations" in "scaled_log") and metabolite names, KEGG IDs, experimental conditions and clustering solutions per experimental condition are stored in colData and timepoint specifications in rowData. (SummarizedExperiment).

Usage

```
data("longitudinalMetabolomics")
```

Format

A SummarizedExperiment object with concentration tables in assays. RowData contains the time point specification. ColData as specified below.

`condition` experimental condition

`metabolite` metabolite name

`KEGG` KEGG ID of metabolites

`replicate` column that specifies the measurement replicate

`cluster` cluster ID that is condition specific for every metabolite

Source

Script used to create simulated data

load KEGG database for assignment of metabolite names: data("metabolite_modules")

metabolite_db <- metabolite_modules # Group <- middle_hierarchy

library(dplyr) library(SummarizedExperiment) # Parameters (as before)

n_features <- 98

n_groups <- 8 # Number of groups (randomly choose between 6-8)

n_time_points <- 4 # Number of time points

n_replicates <- 3 # Number of replicates for all features and time points

n_conditions <- 3 # Number of experimental conditions

x_varying_groups <- 4 # Number of groups with varying dynamics across conditions

```
condition_names <- c("A","B","C")

# Probability matrix for assigning metabolites from different database groups to dynamic groups #
For simplicity, we assume equal probability; customize as needed

group_probabilities <- matrix(c(0.8,rep(0.01,7), #amino acid metabolism rep(0.01,7),0.8, #nucleotide
metabolism 0.1,0.8,0.8,rep(0.1,5), # energy and carbohydrate metabolism runif(5 * length(unique(metabolite_modules$middl
nrow = n_groups, ncol = length(unique(metabolite_modules$middle_hierarchy)))

# Generate group dynamics (base trends over time) for each condition

group_dynamics <- list()

# Define the base group dynamics for condition 1

group_dynamics[[1]] <- lapply(1:n_groups, function(g) trend <- rnorm(n_time_points, mean = g *
2, sd = 0.5) return(trend) )

# Define varying dynamics for selected groups across other conditions

varying_groups <- sample(1:n_groups, x_varying_groups, replace = FALSE)

for (cond in 2:n_conditions) group_dynamics[[cond]] <- group_dynamics[[1]] for (g in varying_groups)
group_dynamics[[cond]][[g]] <- rnorm(n_time_points, mean = g * 2, sd = 1)

# Assign each feature to a group

feature_to_group <- sample(1:n_groups, n_features, replace = TRUE)

# Initialize a list to store the simulated data

simulated_data <- list()

# Assign metabolite names to features

available_metabolites <- metabolite_modules # Copy of metabolite database to keep track of unused
names

# Simulate data for each feature across all conditions

for (feature in 1:n_features)

# Get the group for this feature

group <- feature_to_group[feature]

# Determine probability of each metabolite database group for this dynamic group

group_probs <- group_probabilities[group, ]

# Subset the metabolite database for selection based on group probabilities

metabolite_candidates <- available_metabolites group_by(middle_hierarchy) mutate(Probability =
group_probs[match(middle_hierarchy, unique(metabolite_modules$middle_hierarchy))]) ungroup()
filter(metabolite

# Randomly sample a metabolite based on these probabilities

metabolite_name <- sample(metabolite_candidates$metabolite, 1, prob = metabolite_candidates$Probability)

# Remove this metabolite from available pool

available_metabolites <- available_metabolites[available_metabolites$metabolite != metabolite_name,
]

# Generate a random base mean for this feature between 0.001 and 1000

base_mean <- runif(1, min = 0.001, max = 1000)
```

```
# Generate feature-specific variances for each time point

feature_variances <- runif(n_time_points, min = 0.1, max = 2)

# Store data for each condition

for (cond in 1:n_conditions) trend <- group_dynamics[[cond]][[group]] feature_means <- base_mean
* trend / max(abs(trend))

feature_data <- data.frame( metabolite = metabolite_name, # Assign metabolite name here condi-
tion = paste0(condition_names[[cond]]), time = rep(1:n_time_points, each = n_replicates), replicate
= rep(1:n_replicates, times = n_time_points) )

# Generate the actual data points with strictly positive concentrations

feature_data$measurement <- unlist(lapply(1:n_time_points, function(t) rlnorm(n_replicates, mean-
log = log(feature_means[t]), sdlog = feature_variances[t]) ))

simulated_data[[length(simulated_data) + 1]] <- feature_data

rm(base_mean,cond,feature,feature_means,feature_to_group,feature_variances, g,group,group_probs,metabolite_name,n_co
n_time_points,trend,varying_groups,x_varying_groups,available_metabolites,feature_data,group_dynamics,
group_probabilities,metabolite_candidates)

# Combine all features and conditions into one data frame

simulated_data_df <- do.call(rbind, simulated_data)

simulated_data_df <- simulated_data_df group_by(metabolite, condition) mutate( log_m = log10(measurement),
m_scaled = (log_m - mean(log_m)) / sd(log_m) )

# add KEGG IDs name_map_HMDB_CAS <- readr::read_csv("name_map_HMDB_CAS.csv")
longitudinalMetabolomics <- dplyr::left_join(longitudinalMetabolomics,name_map_HMDB_CAS[,c("Query","KEGG")],jo

## concentrations temp <- longitudinalMetabolomics temp <- temp select(condition,metabolite,KEGG,time,measurement,rep
pivot_wider(id_cols=c(condition,metabolite,KEGG,replicate), names_from = time, values_from =
measurement) concentrations <- temp ## transform matrix so that conditions are in columns to facil-
itate access ## with colData -> se[,se$condition="A"] concentrations <- t(as.matrix(concentrations))
row.names(concentrations) <- NULL # prepare log_transformed data temp <- data temp <- temp se-
lect(condition,metabolite,KEGG,time,log_m,replicate) pivot_wider(id_cols=c(condition,metabolite,KEGG,replicate),
names_from = time, values_from = log_m) log_con <- temp log_con <- t(as.matrix(log_con))
row.names(log_con) <- NULL # prepare scaled log_transformed data temp <- data temp <- temp se-
lect(condition,metabolite,KEGG,time,m_scaled,replicate) pivot_wider(id_cols=c(condition,metabolite,KEGG,replicate),
names_from = time, values_from = m_scaled) scaled_log <- temp scaled_log <- t(as.matrix(scaled_log))
row.names(scaled_log) <- NULL

# prepare row and colData #### row_data <- DataFrame(time_points=c("time_point_1","time_point_2","time_point_3",
"time_point_4")) col_data <- DataFrame(condition=temp$condition,metabolite=temp$metabolite,
KEGG=temp$KEGG,replicate=temp$replicate)

se <- SummarizedExperiment(assays=SimpleList(concentrations=concentrations, log_con=log_con,
scaled_log=scaled_log), rowData = row_data, colData = col_data) # set row and colnames ####
rownames(se) <- c("time_point_1","time_point_2","time_point_3", "time_point_4") colnames(se)
<- temp$metabolite

# add metadata #### metadata(se)[["data origin"]] <- "Simulated data of 98 metabolites with three
concentration observations at four time points and at three different biological conditions. Script to
construct dataset can be seen with ?longitudinalMetabolomics"
```

add clustering solution # get distances between vectors clust_A <- metadata(data)[["estimates_dynamics"]][["A"]] clust_A <- clust_A select(metabolite.ID, condition, time.ID, mu_mean) pivot_wider(id_cols = c(metabolite.ID, condition), names_from = time.ID, values_from = mu_mean)

dist <- clust_A dd_A <- dist(dist, method = "euclidean") # hierarchical clustering clust <- hclust(dd_A, method = "ward.D2") clust_cut <- cutree(clust, k = 8) # adding cluster ID to estimates clust_A$cluster <- clust_cut

get distances between vectors clust_B <- metadata(data)[["estimates_dynamics"]][["B"]] clust_B <- clust_B select(metabolite.ID, condition, time.ID, mu_mean) pivot_wider(id_cols = c(metabolite.ID, condition), names_from = time.ID, values_from = mu_mean)

dist <- clust_B dd_B <- dist(dist, method = "euclidean") # hierarchical clustering clust <- hclust(dd_B, method = "ward.D2") clust_cut <- cutree(clust, k = 8) # adding cluster ID to estimates clust_B$cluster <- clust_cut

get distances between vectors clust_C <- metadata(data)[["estimates_dynamics"]][["C"]] clust_C <- clust_C select(metabolite.ID, condition, time.ID, mu_mean) pivot_wider(id_cols = c(metabolite.ID, condition), names_from = time.ID, values_from = mu_mean)

dist <- clust_C dd_C <- dist(dist, method = "euclidean") # hierarchical clustering clust <- hclust(dd_C, method = "ward.D2") clust_cut <- cutree(clust, k = 8) # adding cluster ID to estimates clust_C$cluster <- clust_cut

cluster <- rbind(clust_A,clust_B,clust_C) metadata(se)[["cluster"]] <- cluster

---

`metabolite_modules` *KEGG Query Results of experimental metabolites*

Description

Using the package KEGGREST (https://www.bioconductor.org/packages/release/bioc/html/KEGGREST.html) all experimental metabolites (see data("intra")) were queried with there KEGG-IDs and all functional modules recorded to which the metabolite is annotated in the KEGG-database.

Usage

```
data("metabolite_modules")
```

Format

A data frame with 348 observations on the following 8 variables.

`...1` row number of the dataframe

`metabolite` name of the experimental metabolite

`KEGG` KEGG ID of the experimental metabolite

`module_id` ID of the KEGG module to which the metabolite is annotated

`module_name` name of the KEGG module to which the metabolite is annotated

`upper_hierarchy` name of the highest hierachy level of module organisation

`middle_hierarchy` name of the middle hierachy = functional module, p.e. "Amino acid metabolism"

`lower_hierarchy` name of the lowest level of modules, this often contain only a couple pathways p.e. "Arginine and proline metabolism"

Source

https://www.genome.jp/kegg/module.html data <- get_ORA_annotations(longitudinalMetabolomics,update_background=TR
metabolite_modules <- get_ORA_annotations["annotations"]

See Also

modules_compounds/ `get_ORA_annotations()`

`modules_compounds` *Background KEGG Query Results Of Functional Modules*

Description

Using the package KEGGREST (https://www.bioconductor.org/packages/release/bioc/html/KEGGREST.html)
a list of all KEGG-modules (KeggList("module")) including their upper, middle and lower hierachy
as given by the KEGG-database and the corresponding annotated metabolites was queried.

Usage

```
data("modules_compounds")
```

Format

A data frame with 1988 observations on the following 6 variables.

`KEGG` KEGG ID of a metabolite annotated to a functional module

`upper_hierarchy` name of the highest hierachy level of module organisation

`middle_hierarchy` name of the middle hierachy = functional module, p.e. "Amino acid metabolism"

`lower_hierarchy` name of the lowest level of modules, this often contain only a couple pathways
p.e. "Arginine and proline metabolism"

`module_id` the ID of the KEGG functional module

`module_name` name of the KEGG module

Source

https://www.genome.jp/kegg/module.html data <- get_ORA_annotations(longitudinalMetabolomics,update_background=TR
modules_compunds <- get_ORA_annotations["background"]

See Also

metabolite_modules/ `get_ORA_annotations()`

`ORA_hypergeometric` *OverRepresentationAnalysis with a hypergeometric model*

Description

Testing the hypothesis that certain KEGG modules are over-represented in clusters of metabolites. A module is considered over-represented in a cluster the number of metabolites in a cluster being annotated to a functional module (n_obs) is higher than the expected number of metabolites in a cluster of this size being annotated to a functional module (n_theo). We can calculate the OvE (Observed versus Expected = n_obs/n_theo) and show the probabilites of these ratios. log(p(OvE))>0 indicates an over-representation of the functional module in the cluster, log(p(OvE))<0 an under-representation.

Usage

```
ORA_hypergeometric(
  background = metadata(data)[["KEGG_annotations"]]$background,
  annotations = metadata(data)[["KEGG_annotations"]]$annotations,
  data,
  tested_column = "middle_hierarchy"
)
```

Arguments

`background`	dataframe that contains KEGG IDs of metabolites that are assigned to functional modules
`annotations`	dataframe tha contains information to which functional modules our experimental metabolites are annotated in KEGG
`data`	dataframe containing columns "KEGG" specifying the KEGG identifiers of metabolites, "cluster" specifying the cluster ID of metabolites and a column specifying the experimental condition called "condition" or if data is a SummarizedExperiment or a SummarizedExperiment clustering solution must be stored in metadata(data) under "cluster"
`tested_column`	column that is in background and annotations and on which the hypergeometric model will be executed

Value

a dataframe containing the ORA results or if data is SummarizedExperiment SummarizedExperiment object the output is stored in metadata(data) under "ORA_tested_column"

See Also

function to retrieve "background" and "annotation" data frames `get_ORA_annotations()`. Function to visualize ORA results `plot_ORA()`

Examples

```
data("longitudinalMetabolomics")
data("modules_compounds")
head(modules_compounds)
data("metabolite_modules")
head(metabolite_modules)
# middly hierachy
longitudinalMetabolomics <- ORA_hypergeometric(
  data = longitudinalMetabolomics,
  annotations = metabolite_modules,
  background = modules_compounds,
  tested_column = "middle_hierarchy"
)
S4Vectors::metadata(longitudinalMetabolomics)[["ORA_middle_hierarchy"]]
```

---

plot_diagnostics *Plot diagnostic criteria of numerical fit of Bayesian model of dynamics*

Description

Plot diagnostic criteria of numerical fit of Bayesian model of dynamics

Usage

```
plot_diagnostics(
  data,
  assay = "scaled_log",
 diagnostics = metadata(data)[["diagnostics_dynamics"]][["model_diagnostics"]],
  divergences = TRUE,
  max_treedepth = TRUE,
  Rhat = TRUE,
  n_eff = TRUE
)
```

Arguments

data	dataframe or colData of a SummarizedExperiment used to fit dynamics model must contain column "time"
assay	of the SummarizedExperiment object that was used to fit the dynamics model
diagnostics	dataframe containing diagnostics criteria from the numerical fit of Bayesian model of dynamics obtained by function diagnostics_dynamics()
divergences	should number of divergent transitions be visualized?
max_treedepth	should number of exeeded maximum treedepth be visualized?
Rhat	should Rhat be visualized?
n_eff	should number of effective samples be visualized?

Value

plots of diagnostic criteria of numerical fit of Bayesian model of dynamics

See Also

parent function diagnostics_dynamics() visualization function for posterior predictive check plot_PPC()

Examples

```
data("longitudinalMetabolomics")
data <- longitudinalMetabolomics[, longitudinalMetabolomics$condition == "A" &
  longitudinalMetabolomics$metabolite %in% c("ATP", "ADP")]
data <- fit_dynamics_model(
  data = data,
  scaled_measurement = "m_scaled", assay = "scaled_log",
  max_treedepth = 14, adapt_delta = 0.95, iter = 2000, cores = 1, chains = 1
)
data <- diagnostics_dynamics(
  data = data, assay = "scaled_log",
  iter = 2000, chains = 1,
  fits = metadata(data)[["dynamic_fits"]]
)
plot_diagnostics(data = data, assay = "scaled_log")
```

---

plot_estimates *Visualization of parameter estimates from numeric fit of Bayesian model of dynamics*

Description

Visualization of parameter estimates from numeric fit of Bayesian model of dynamics

Usage

```
plot_estimates(
  data,
  estimates = metadata(data)[["estimates_dynamics"]],
  assay = "scaled_log",
  time = "time",
  delta_t = TRUE,
  dynamics = TRUE
)
```

Arguments

data	dataframe or SummarizedExperiment used used to fit dynamics model and extract the estimates
estimates	a list of dataframes (one per experimental condition) that contains the estimates at the timepoints and samples from the posterior generated by estimates_dynamics() or if data is a SummarizedExperiment estimates must be stored in metadata(data) under "estimates_dynamics"
assay	of the SummarizedExperiment object that was used to fit the dynamics model
time	column in "data" that contains the time point identifiers
delta_t	should differences between timepoints be plotted?
dynamics	should dynamics be plotted?

Value

Visualization of differences between timepoints(delta_t) and dynamics profiles of single metabolites

See Also

parent function `estimates_dynamics()`

Examples

```
data("longitudinalMetabolomics")
data <- longitudinalMetabolomics[, longitudinalMetabolomics$condition == "A" &
  longitudinalMetabolomics$metabolite == "ATP"]
data <- fit_dynamics_model(
  data = data,
  scaled_measurement = "m_scaled", assay = "scaled_log",
  max_treedepth = 14, adapt_delta = 0.95, iter = 2000, cores = 1, chains = 1
)
data <- estimates_dynamics(
  data = data, iter = 2000,
  chains = 1, condition = "condition"
)
plot_estimates(data = data, delta_t = FALSE)
plot_estimates(data = data, dynamics = FALSE)
```

---

plot_ORA *Plot results of over-representation analysis with ORA_hypergeometric()*

Description

Plot results of over-representation analysis with ORA_hypergeometric()

Usage

```
plot_ORA(data, tested_column = "middle_hierarchy")
```

Arguments

data	result dataframe from ORA_hypergeometric() or SummarizedExperiment object where the ORA_hypergeometric() results are stored in metadata(data) under "ORA_tested_column"
tested_column	KEGG module hierarchy level on which ORA was executed

Value

a plot of the over-representation analysis

See Also

do over-represenation analysis of KEGG functional modules ORA_hypergeometric()

Examples

```
data("longitudinalMetabolomics")
data("modules_compounds")
head(modules_compounds)
data("metabolite_modules")
head(metabolite_modules)
# middly hierachy
longitudinalMetabolomics <- ORA_hypergeometric(
  data = longitudinalMetabolomics,
  annotations = metabolite_modules,
  background = modules_compounds,
  tested_column = "middle_hierarchy"
)
plot_ORA(longitudinalMetabolomics)
```

---

plot_PPC *Plots posterior predictive check of numerical fit of Bayesian dynamics model*

Description

Plots posterior predictive check of numerical fit of Bayesian dynamics model

Usage

```
plot_PPC(
  posterior = metadata(data)[["diagnostics_dynamics"]],
  data,
  assay = "scaled_log",
  scaled_measurement = "scaled_measurement"
)
```

Arguments

posterior	a list of one dataframe per condition that contains necessary information for Posterior predictive check obtained by function diagnostics_dynamics()(named "PPC_condition")
data	dataframe or colData of a SummarizedExperiment used to fit dynamics model
assay	of the SummarizedExperiment object that was used to fit the dynamics model
scaled_measurement	column name of concentration values used to model fit, should be normalized by experimental condition and metabolite to mean of zero and standard deviation of one

Value

a list of visual posterior predictive check, one per experimental condition

See Also

parent function `diagnostics_dynamics()` visualization function for diagnostics `plot_diagnostics()`

Examples

```
data("longitudinalMetabolomics")
data <- longitudinalMetabolomics[, longitudinalMetabolomics$condition == "A" &
  longitudinalMetabolomics$metabolite %in% c("ATP", "ADP")]
data <- fit_dynamics_model(
  data = data,
  scaled_measurement = "m_scaled", assay = "scaled_log",
  max_treedepth = 14, adapt_delta = 0.95, iter = 2000, cores = 1, chains = 1
)
data <- diagnostics_dynamics(
  data = data, assay = "scaled_log",
  iter = 2000, chains = 1,
  fits = metadata(data)[["dynamic_fits"]]
)
plot_PPC(
  data = data, assay = "scaled_log"
)
```

Index