Package: MetaboDynamics (via r-universe)

October 31, 2026

Title Bayesian analysis of longitudinal metabolomics data

Version 2.3.15

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)

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, dynamicTreeCut, rlang, ape, ggtree, patchwork

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,Pathways,TimeCourse,
Clustering

Biarch true

Config/roxygen2/version 8.1.0

Config/pak/sysreqs libcairo2-dev cmake libfontconfig1-dev libfreetype6-dev make libicu-dev libpng-dev libuv1-dev libssl-dev zlib1g-dev

Repository https://bioc.r-universe.dev

Date/Publication 2026-10-30 17:10:57 UTC

RemoteUrl https://github.com/bioc/MetaboDynamics

RemoteRef HEAD

RemoteSha 6ad6c3abc1f82e35a9fcef21f480beb7f4b4cdc6

Contents

cluster_dynamics . . . 3
compare_dynamics . . . 4
compare_metabolites . . . 5
diagnostics_dynamics . . . 6
estimates_dynamics . . . 7
fit_dynamics_model . . . 9
get_ORA_annotations . . . 11
heatmap_dynamics . . . 12
heatmap_metabolites . . . 13
IDs . . . 14
longitudinalMetabolomics . . . 15
longitudinalMetabolomics_df . . . 18
metabolite_modules . . . 21
modules_compounds . . . 22
ORA_hypergeometric . . . 22
plot_cluster . . . 24
plot_diagnostics . . . 25
plot_estimates . . . 26
plot_ORA . . . 27
plot_PPC . . . 28

Index **30**

`cluster_dynamics` *cluster dynamics profiles of metabolites*

Description

Convenient wrapper function for clustering of metabolite dynamics employing the "hybrid" method of the dynamicTreeCut package for clustering and hclust for computing of distance matrix and hierarchical clustering needed as input for dynamicTreeCut. Provides bootstrapping of clustering solution from posterior estimates of the model.

Usage

```
cluster_dynamics(
  data = NULL,
  fit,
  estimates = NULL,
  distance = "euclidean",
  agglomeration = "ward.D2",
  minClusterSize = 1,
  deepSplit = 2,
  B = 1000
)
```

Arguments

`data`	data frame or colData of a SummarizedExperiment used to fit dynamics model
`fit`	model fit obtained by fit_dynamics_model(). Needed if data is not a SummarizedExperiment object for which the model fit is saved in "dynamic_fit" of metadata
`estimates`	output of estimates_dynamics function, needed if data is not a SummarizedExperiment object for which the model estimates are saved in "estimates_dynamics" of metadata
`distance`	distance method to be used as input for hierarchical clustering dist can be "euclidean", "maximum", "manhattan", "canberra", "binary" or "minkowski"
`agglomeration`	agglomerative method to be used for hierarchical clustering hclust can be "ward.D", "ward.D2", "single", "complete", "average", "mcquitty", "median" or "centroid"
`minClusterSize`	minimum number of metabolites per of cluster cutreeDynamic
`deepSplit`	rough control over sensitivity of cluster analysis. Possible values are 0:4, the higher the value, the more and smaller clusters will be produced by cutreeDynamic
`B`	number of bootstraps

Value

a list with one list per condition. The elements per condition are 'data' (mean estimates of mu plus the clustering solution), mean_dendro' the dendrogram of the mean estimates, and mean_phylo' the phylogram of the mean estimates. if data is a SummarizedExperiment object clustering results are stored in metadata under "cluster" Element 'dynamics' contains column names of time points

See Also

`fit_dynamics_model()`, `estimates_dynamics()`, `plot_cluster()`

Examples

```
data("longitudinalMetabolomics")
data <- longitudinalMetabolomics[, longitudinalMetabolomics$condition == "A" &
  longitudinalMetabolomics$metabolite %in% c("ATP", "L-Alanine", "GDP")]
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
)
data <- cluster_dynamics(data, B = 1000)
S4Vectors::metadata(data)[["cluster"]][["A"]]
plot(S4Vectors::metadata(data)[["cluster"]][["A"]][["mean_dendro"]])
```

`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 abundances at different time points) of two clusters that come from different experimental conditions to estimate the mean distance between clusters.

Usage

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

Arguments

`data`	result of `cluster_dynamics()` function: either a list of data frames or a SummarizedExperiment object
`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")
data <- longitudinalMetabolomics[, longitudinalMetabolomics$condition %in% c("A", "B") &
  longitudinalMetabolomics$metabolite %in% c("ATP", "L-Alanine", "GDP")]
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
)
data <- cluster_dynamics(data, B = 1)
data <- compare_dynamics(
  data = data,
  cores = 1
)
S4Vectors::metadata(data)[["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)
```

Arguments

data	result of `cluster_dynamics()` function: either a list of data frames or a SummarizedExperiment object

Value

a data frame 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. Output 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,
  fit = metadata(data)[["dynamic_fit"]]
)
```

Arguments

data	data frame or a SummarizedExperiment used to fit dynamics model column of "time" that contains time must be numeric, has to contain columns specifying the metabolite named "metabolite", and column specifiying the time point named "time", a column named "condition" must specify the experimental condition.
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
fit	model fit 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_fit"

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$condition %in% c("A", "B") &
  longitudinalMetabolomics$metabolite == "ATP"]
data <- fit_dynamics_model(
  model = "scaled_log",
  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,
  fit = metadata(data)[["dynamic_fit"]]
)
S4Vectors::metadata(data)[["diagnostics_dynamics"]][["model_diagnostics"]]
S4Vectors::metadata(data)[["diagnostics_dynamics"]][["posterior"]]
```

---

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

Description

Extracts the mean concentrations (mu) at every time point from the dynamics model fit, the 95% highest density interval (HDI) (i.e. metabolite dynamic vectors (MDVs)), differences between subsequent time points of the same condition and Euclidean distances between MDVs of the same metabolite under different conditions.

Usage

```
estimates_dynamics(
  data,
  assay = "scaled_log",
  fit = metadata(data)[["dynamic_fit"]],
  model_option = "sd_per_condition"
)
```

Arguments

data	data frame or colData of a SummarizedExperiment used to fit dynamics model, must contain a 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 stored in metadata(data) under "dynamic_fits" (automatically done by `fit_dynamics_model()`
assay	of the SummarizedExperiment object that was used to fit the dynamics model
fit	model fit for which estimates should be extracted
model_option	model option that was used for fitting with `fit_dynamics_model()`. If data is a SummarizedExperiment this is automatically provided by the metadata written by `fit_dynamics_model()`

Value

a list of dataframes (one per parameters mu, sigma, lambda, delta_mu and euclidean distance) that contains the estimates: mu: is the estimated mean metabolite abundance sigma: the estimated standard deviation of metabolite abundance lambda: pooled sigma per condition delta_mu: differences of mu between time points euclidean_distances: estimated euclidean distance of time vectors of one metabolite between conditions 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
)
S4Vectors::metadata(data)[["estimates_dynamics"]]
```

`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 = normalized metabolite abundance) values for robust estimation of mean concentrations over time of single metabolites at single experimental conditions. At least three replicates for metabolite concentrations per time point and condition are needed. If cell counts are provided at least one replicate per time point and condition is needed.

Usage

```
fit_dynamics_model(
  model = "scaled_log",
  model_option = "sd_per_condition",
  data,
  scaled_measurement = "m_scaled",
  counts = NULL,
  assay = "scaled_log",
  chains = 4,
  cores = 4,
  adapt_delta = 0.95,
  max_treedepth = 10,
  iter = 2000,
  warmup = iter/4,
  prior_mean_abundance = c(12, 5),
  prior_sd_abundance = 2,
  prior_counts = 1e+07
)
```

Arguments

`model`	which model to fit. Two options are available: "scaled_log": taking in normalized and scaled metabolite concentrations (see scaled measurement) "raw_plus_cell_counts": tailored for in vitro untargeted LC-MS experiments, taking in "raw" (i.e. not normalized and not scaled) metabolite concentrations and cell counts. This model assumes independent measurement (i.e. different wells) of cell counts and metabolite concentrations. Additionally it assumes that cell counts were estimated e.g. by cell counters (i.e. that cells were not counted under the microscope) leading to a small uncertainty of the true cell count.
`model_option`	for each model ("scaled_log" and "raw_plus_cell_counts") two versions are available: "sd_per_time_point" and "sd_per_condition". "sd_per_time_point" is suited for more replicates (at least triplicates) and estimates standard deviations per time point and therefore allows for more sensitive discrimination of changing abundances. high deviations between replicates) or for many metabolites, time

points or conditions as it reduces run time due to less parameters. For model details see vignette (browseVignettes("MetaboDynamics")).

`data` concentration table with at least three (model_option "sd_per_time_point) or two (model_option "sd_per_condition") replicate measurements per metabolite. Must contain columns named "metabolite" (containing names or IDs), "time" (categorical, the same for all conditions), and "condition" or colData of a SummarizedExperiment object Time column needs to be sorted in ascending order

`scaled_measurement` column of "data" that contains the abundances per cell (model "raw_plus_cell_counts"), or in case of model "scaled_log" the log-transformed metabolite abundances centered and normalized per metabolite and experimental condition to mean=0 and sd=1, must be numeric

`counts` only model "raw_plus_cell_counts", data frame with at least one replicate per time point and condition specifying the cell counts, must contain columns "time", and "condition" equivalent to the specifications of "data". Must contain a column named "counts" that specifies the cell counts. Model assumes that the replicates of the cell counts and metabolite concentrations are independent of each other (i.e. cell counts were measured in in different wells than metabolite concentrations)

`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, in case of the model "scaled_log" assay needs to hold scaled log-transformed metabolite concentrations (mean=0,sd=1 per metabolite and experimental condition), if model "raw_plus_cell_counts" is chosen must hold the non-transformed and non-scaled metabolite concentrations

`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

`prior_mean_abundance` only for model 'raw_plus_cell_counts', has to be >0, mu and sigma of normal distribution covering the range for log-transformed metabolite abundances, for details see Vignette "Prior setting".

`prior_sd_abundance` only for model 'raw_plus_cell_counts', has to be >0, expected value (most abundant) value of metabolite specific standard deviation of log-transformed metabolite abundances, for details see Vignette "Prior setting".

prior_counts only for model 'raw_plus_cell_counts', has to be >0, expected value of cell counts, for details see Vignette "Prior setting".

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". 'model' and 'model_option' are also stored in metadata(data)

See Also

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

Examples

```
## on scaled log-transformed metabolite concentrations
data("longitudinalMetabolomics")
data <- longitudinalMetabolomics[, longitudinalMetabolomics$condition %in% c("A", "B") &
  longitudinalMetabolomics$metabolite == "ATP"]
data <- fit_dynamics_model(
  model = "scaled_log",
  data = data,
  assay = "scaled_log",
  max_treedepth = 14, adapt_delta = 0.95, iter = 2000, cores = 1, chains = 1
)
S4Vectors::metadata(data)[["dynamic_fit"]]
```

---

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 = TRUE
)
```

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? Should be set to TRUE of this is the first time using this function. 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

Do 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")
data <- longitudinalMetabolomics[, longitudinalMetabolomics$condition %in% c("A", "B") &
  longitudinalMetabolomics$metabolite %in% c("ATP", "L-Alanine", "GDP")]
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
)
data <- cluster_dynamics(data, B = 1)
data <- compare_dynamics(
  data = data,
  cores = 1
)
S4Vectors::metadata(data)[["comparison_dynamics"]]
heatmap_dynamics(data = data)
```

---

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 SummarizedExperiment than 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)
```

IDs *KEGG ID mapping of metabolites in data set longitudinalMetabolomics*

Description

Example data set of KEGG ID annotations of metabolites as needed for `ORA_hypergeometric()`

Usage

```
data("IDs")
```

Format

A dataframe

`KEGG` KEGG ID of metabolites

`replicate` column that specifies the measurement replicate

`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 2 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 set.seed(1234) # load KEGG database for assignment of metabolite names: data("metabolite_modules")

metabolite_db <- metabolite_modules # Group <- middle_hierarchy

library(dplyr) library(SummarizedExperiment) library(tidyverse) # 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 <- 2 # Number of experimental conditions

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

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

# 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 = 0.1, max = 1e5)

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 condition = 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, meanlog = 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(simulated_data_df,name_map_HMDB_CAS[,c("Query","KEGG")],join_by("

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 facilitate access ## with colData -> se[,se$condition="A"] concentrations <- t(as.matrix(concentrations)) row.names(concentrations) <- NULL # prepare log_transformed data temp <- longitudinalMetabolomics temp <- temp select(condition,metabolite,KEGG,time,log_m,replicate) pivot_wider(id_cols=c(condition,metabolite,KEGG,r 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 <- longitudinalMetabolomics temp <- temp select(condition,metabolite,KEGG,time,m_scaled,replicate) pivot_wider(id_cols=c(condition,metabolite,KEG names_from = time, values_from = m_scaled) scaled_log <- temp scaled_log <- t(as.matrix(scaled_log)) row.names(scaled_log) <- NULL

simulate cell counts ## I am assuming that all cell counts are from a poisson distribution with the same ## location parameter (not the case for real life applications). One parameter is just used to show application of the dynamics model including cell counts

counts <- rbind(rpois(ncol(concentrations),1e5), rpois(ncol(concentrations),1e5), rpois(ncol(concentrations),1e5), rpois(ncol(concentrations),1e5)) # 882 columns, three entries for same metabolite, four time points

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, cell_counts = counts), 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"

data <- se data <- fit_dynamics_model(data = data, scaled_measurement = "m_scaled", assay = "scaled_log", max_treedepth = 14, adapt_delta = 0.999, iter = 5000, cores = 2, chains = 2)

data <- estimates_dynamics(data = data) data <- cluster_dynamics(data,B=10) metadata(data)[["dynamic_fit"]] <- NULL

data("modules_compounds") data("metabolite_modules") data("IDs") data <- ORA_hypergeometric(data = data, background = modules_compounds, IDs = IDs, annotations = metabolite_modules, tested_column = "middle_hierarchy")

save shortened analysis in metadata of se metadata(se)[["estimates_dynamics"]] <- metadata(data)[["estimates_dynamics"]] metadata(se)[["dynamics"]] <- metadata(data)[["dynamics"]] metadata(se)[["cluster"]] <- metadata(data)[["cluster"]] metadata(se)[["ORA_middle_hierarchy"]] <- metadata(data)[["ORA_middle_hierarchy"]]

longitudinalMetabolomics <- se usethis::use_data(longitudinalMetabolomics,overwrite = TRUE)

longitudinalMetabolomics_df <- dplyr::left_join(simulated_data_df,name_map_HMDB_CAS[,c("Query","KEGG")],join_b usethis::use_data(longitudinalMetabolomics_df,overwrite = TRUE)

`longitudinalMetabolomics_df`

A simulated data set of longitudinal concentration tables of metabolites. In contrast to "longitudinalMetabolomics" this dataset is in data frame format. It was simulated with a different seed compared to longitudinalMetabolomics so the results can deviate.

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 <- 2 # Number of experimental conditions

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

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

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 condition = 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, meanlog = 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_df <- dplyr::left_join(simulated_data_df,name_map_HMDB_CAS[,c("Query","KEGG")],join_b
usethis::use_data(longitudinalMetabolomics_df,overwrite = TRUE)
```

`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 hierarchy level of module organization

`middle_hierarchy` name of the middle hierarchy = 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

See Also

modules_compounds

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 organization

`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

See Also

metabolite_modules

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 probabilities 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,
  annotations,
  data,
  IDs,
  tested_column = "middle_hierarchy"
)
```

Arguments

background	dataframe that contains KEGG IDs of metabolites that are assigned to functional modules, is incorporated in the package modules_compounds
annotations	dataframe tha contains information to which functional modules our experimental metabolites are annotated in KEGG, can be constructed by filtering the provided KEGG background modules_compounds for the experimental metabolites
data	result of `cluster_dynamics()` function: either a list of data frames or a SummarizedExperiment object
IDs	dataframe with two columns 'metabolite' and 'KEGG' mapping KEGG IDs to metabolites. If function `get_ORA_annotations()` is used to retrieve IDs these are stored under "KEGG_annotations" of metadata(data)
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 obtain data `cluster_dynamics()` Function to visualize ORA results `plot_ORA()`

`get_ORA_annotations()`

Examples

```
data("longitudinalMetabolomics")
data("modules_compounds")
head(modules_compounds)
data("metabolite_modules")
head(metabolite_modules)
data("IDs")
head(IDs)

longitudinalMetabolomics <- ORA_hypergeometric(
  data = longitudinalMetabolomics,
  annotations = metabolite_modules,
  background = modules_compounds,
```

```
  IDs = IDs,
  tested_column = "middle_hierarchy"
)
S4Vectors::metadata(longitudinalMetabolomics)[["ORA_middle_hierarchy"]]
```

`plot_cluster` *visualize clustering solution of cluster_dynamics()*

Description

visualize clustering solution of cluster_dynamics()

Usage

```
plot_cluster(data)
```

Arguments

`data`	result of `cluster_dynamics()` function: either a list of data frames or a SummarizedExperiment object

Value

a list of plots. Per experimental condition: 1) a 'bubbletree': a phylogram with numbers on nodes indicating in how many bootstraps of the posterior estimates the same clustering solution was generated, 2) cluster affiliation of metabolites, 3) dynamics of metabolites per cluster, 4) patchwork of 1-3, 5) order of the tips in bubbletree: needed for matching cluster plots and ORA

See Also

`cluster_dynamics()`

Examples

```
data("longitudinalMetabolomics")
plots <- plot_cluster(longitudinalMetabolomics[, longitudinalMetabolomics$condition == "A"])

plots[["trees"]][["A"]]
```

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 exceeded 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(
  model = "scaled_log",
  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,
  fit = metadata(data)[["dynamic_fit"]]
)
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 = NULL,
  estimates = metadata(data)[["estimates_dynamics"]],
  delta_t = TRUE,
  MDVs = TRUE,
  distance_conditions = TRUE
)
```

Arguments

`data`	SummarizedExperiment used to fit dynamics model and extract the estimates
`estimates`	a list of data frames (elements: mu, sigma, lambda, euclidean_distance) that contains the model estimates by estimates_dynamics() or if data is a SummarizedExperiment estimates must be stored in metadata(data) under "estimates_dynamics"
`delta_t`	should differences between time points be plotted?
`MDVs`	should metabolite dynamics vectors (MDVs) be plotted?
`distance_conditions`	should differences (i.e. Euclidean distances) of MDVs between conditions be plotted?

Value

Visualization of differences between time points(delta_t), MDVs and Euclidean distances

See Also

parent function estimates_dynamics()

Examples

```
data("longitudinalMetabolomics")
data <- longitudinalMetabolomics[, longitudinalMetabolomics$condition %in% c("A", "B") &
  longitudinalMetabolomics$metabolite %in% c("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
)
plot_estimates(data = data, delta_t = TRUE, MDVs = FALSE, distance_conditions = FALSE)
plot_estimates(data = data, delta_t = FALSE, MDVs = TRUE, distance_conditions = FALSE)
plot_estimates(data = data, delta_t = FALSE, MDVs = FALSE, distance_conditions = TRUE)
```

---

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",
  patchwork = FALSE,
  plot_cluster = NULL
)
```

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

patchwork	should result be patchworked to results of plot_cluster()?
plot_cluster	if patchwork = TRUE this needs to be the result of plot_cluster()

Value

a plot of the over-representation analysis and lsit of plots suitable to patchwork with cluster visualization if patchwork=TRUE

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)
data("IDs")
head(IDs)
# middly hierachy
longitudinalMetabolomics <- ORA_hypergeometric(
  data = longitudinalMetabolomics,
  annotations = metabolite_modules,
  background = modules_compounds,
  tested_column = "middle_hierarchy",
  IDs = IDs
)
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 dataframe 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(
  model = "scaled_log",
  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,
  fit = metadata(data)[["dynamic_fit"]]
)
plot_PPC(
  data = data, assay = "scaled_log"
)
```

Index

∗ **datasets**
  IDs, 14
  longitudinalMetabolomics, 15
  longitudinalMetabolomics_df, 18
  metabolite_modules, 21
  modules_compounds, 22

cluster_dynamics, 3
cluster_dynamics(), *4, 5*, *23, 24*
compare_dynamics, 4
compare_dynamics(), *6*, *13*
compare_metabolites, 5
compare_metabolites(), *5*, *14*
cutreeDynamic, *3*

diagnostics_dynamics, 6
diagnostics_dynamics(), *8*, *11*, *25*, *29*
dist, *3*
dynamicTreeCut, *3*

estimates_dynamics, 7
estimates_dynamics(), *4*, *7*, *11*, *27*

fit_dynamics_model, 9
fit_dynamics_model(), *4*, *7, 8*

get_ORA_annotations, 11
get_ORA_annotations(), *23*

hclust, *3*
heatmap_dynamics, 12
heatmap_dynamics(), *5*
heatmap_metabolites, 13
heatmap_metabolites(), *6*

IDs, 14

longitudinalMetabolomics, *11*, 15
longitudinalMetabolomics_df, 18

metabolite_modules, 21, *22*
modules_compounds, *21*, 22, *23*

ORA_hypergeometric, 22
ORA_hypergeometric(), *12*, *14*, *27, 28*

plot_cluster, 24
plot_cluster(), *4*, *28*
plot_diagnostics, 25
plot_diagnostics(), *7*, *29*
plot_estimates, 26
plot_estimates(), *8*
plot_ORA, 27
plot_ORA(), *23*
plot_PPC, 28
plot_PPC(), *7*, *25*

SummarizedExperiment, *3, 4*, *6*, *8*, *10*, *12–15*, *18*, *23*, *25–27*, *29*