Package ‘microbiomeDASim’

December 17, 2024

Type Package

Title Microbiome Differential Abundance Simulation

Version 1.21.0

Author Justin Williams, Hector Corrada Bravo, Jennifer Tom, Joseph Nathaniel Paulson

Maintainer Justin Williams <williazo@ucla.edu>

Description A toolkit for simulating differential microbiome data designed for longitudinal analyses. Several functional forms may be specified for the mean trend. Observations are drawn from a multivariate normal model. The objective of this package is to be able to simulate data in order to accurately compare different longitudinal methods for differential abundance.

License MIT + file LICENSE

Imports graphics, ggplot2, MASS, tmvtnorm, Matrix, mvtnorm, pbapply, stats, phyloseq, metagenomeSeq, Biobase

Depends R (>= 3.6.0)

Encoding UTF-8

LazyData false

Roxygen list(markdown = TRUE)

RoxygenNote 7.0.2

Suggests testthat (>= 2.1.0), knitr, devtools

VignetteBuilder knitr

biocViews Microbiome, Visualization, Software

BugReports https://github.com/williazo/microbiomeDASim/issues

URL https://github.com/williazo/microbiomeDASim

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

git_branch devel

git_last_commit a5f0a43

git_last_commit_date 2024-10-29

Repository Bioconductor 3.21

Date/Publication 2024-12-16

Contents

`final_output_gen` *Generating the final combined bug output*

Description

Generating the final combined bug output

Usage

```
final_output_gen(
  no_diff_feat,
  diff_abun_features,
  diff_Y,
  null_Y,
  diff_bugs,
  nodiff_bugs,
  final_output = NULL
)
```

Arguments

`no_diff_feat`	number of non differentially abundant features
`diff_abun_features`	number of differentially abundant features
`diff_Y`	simulated outcome for differentially abundant features

`null_Y`	simulated outcome for non differentially abundant features
`diff_bugs`	sample information for differentially abundant features
`nodiff_bugs`	sample information for non differentially abundant features
`final_output`	final object that will store the simulated data

Value

final output list with the OTU table and corresponding bug feature data.frame

`form_beta_check` *Beta Specification Check*

Description

Function for checking that the appopriate beta parameters are specified for each of the mean trend specifications

Usage

```
form_beta_check(form, beta, IP, timepoints)
```

Arguments

`form`	character value specifying the type of time trend. Options include 'linear', 'quadratic', 'cubic', 'M', 'W', 'L_up', and 'L_down'.
`beta`	vector specifying the appropriate parameters for functional trend. See details of `mean_trend` for explanation for each form
`IP`	vector specifying the inflection points. See details of `mean_trend` for explanation for each form
`timepoints`	numeric vector specifying the points to fit the functional trend. @keywords internal

Value

Nothing returned unless an error is returned.

`gen_microbiome_norm_feature_check` *Checking that features are specified appopriately*

Description

Checking that features are specified appopriately

Usage

```
gen_microbiome_norm_feature_check(features, diff_abun_features)
```

Arguments

`features`	Numeric value specifying the total number of features to simulate in the microbiome. Must be greater than zero
`diff_abun_features`	Number of features to simulate with differentially abundant pattern. Must be between zero and number of features specified

Value

Potential warning message if no differentially abundant features or all differentially abundant features are specified

`gen_norm_microbiome` *Generate Longitduinal Differential Abundance from Multivariate Normal*

Description

Generate Longitduinal Differential Abundance from Multivariate Normal

Usage

```
gen_norm_microbiome(
  features = 10,
  diff_abun_features = 5,
  n_control,
  n_treat,
  control_mean,
  sigma,
  num_timepoints,
  t_interval,
  rho,
  corr_str = c("ar1", "compound", "ind"),
  func_form = c("linear", "quadratic", "cubic", "M", "W", "L_up", "L_down"),
  beta,
  IP = NULL,
  missing_pct,
  missing_per_subject,
  miss_val = NA,
  dis_plot = FALSE,
  plot_trend = FALSE,
  zero_trunc = TRUE,
  asynch_time = FALSE
)
```

Arguments

`features`	numeric value specifying the number of features/microbes to simulate. Default is 10.
`diff_abun_features`	numeric value specifying the number of differentially abundant features. Default is 5.
`n_control`	integer value specifying the number of control individuals
`n_treat`	integer value specifying the number of treated individuals
`control_mean`	numeric value specifying the mean value for control subjects. all control subjects are assummed to have the same population mean value.
`sigma`	numeric value specifying the global population standard deviation for both control and treated individuals.
`num_timepoints`	integer value specifying the number of timepoints per subject.
`t_interval`	numeric vector of length two specifying the interval of time from which to draw observatoins [t_1, t_q]. Assumed to be equally spaced over the interval unless `asynch_time` is set to TRUE.
`rho`	value for the correlation parameter. must be between [0, 1]. see `mvrnorm_corr_gen` for details.
`corr_str`	correlation structure selected. see `mvrnorm_corr_gen` for details.
`func_form`	character value specifying the functional form for the longitduinal mean trend. see `mean_trend` for details.
`beta`	vector value specifying the parameters for the differential abundance function. see `mean_trend` for details.
`IP`	vector specifying any inflection points. depends on the type of functional form specified. see `mean_trend` for details. by default this is set to NULL.
`missing_pct`	numeric value that must be between [0, \1] that specifies what percentage of the individuals will have missing values.
`missing_per_subject`	integer value specifying how many observations per subject should be dropped. note that we assume that all individuals must have baseline value, meaning that the maximum number of `missing_per_subject` is equal to `num_timepoints` - 1.

`miss_val`	value used to induce missingness from the simulated data. by default missing values are assummed to be NA but other common choices include 0.
`dis_plot`	logical argument on whether to plot the simulated data or not. by default plotting is turned off.
`plot_trend`	specifies whether to plot the true mean trend. see `mean_trend` for details.
`zero_trunc`	logical indicator designating whether simulated outcomes should be zero truncated. default is set to TRUE
`asynch_time`	logical indicator designed to randomly sample timepoints over a specified interval if set to TRUE. default is FALSE.

Value

This function returns a list with the following objects

`Y` The full simulated feature sample matrix where each row represent a feature and each column a sample. Note that the differential and non-differential bugs are marked by row.names

Examples

```
gen_norm_microbiome(features = 5, diff_abun_features = 2,
               n_control = 10, n_treat = 10, control_mean = 8, sigma = 1,
               num_timepoints = 5, t_interval=c(0, 4), rho = 0.8,
               corr_str = "compound", func_form = "linear", beta =  c(0, 1),
               missing_pct = 0.3, missing_per_subject = 2)
```

`gen_norm_microbiome_obs` *Generate Longitduinal Differential Abundance from Multivariate Normal with Observed Data*

Description

Generate Longitduinal Differential Abundance from Multivariate Normal with Observed Data

Usage

```
gen_norm_microbiome_obs(
  features = 10,
  diff_abun_features = 5,
  id,
  time,
  group,
  ref,
  control_mean,
  sigma,
  rho,
  corr_str = c("ar1", "compound", "ind"),
  func_form = c("linear", "quadratic", "cubic", "M", "W", "L_up", "L_down"),
  beta,
  IP = NULL,
  dis_plot = FALSE,
  plot_trend = FALSE,
  zero_trunc = TRUE
)
```

Arguments

`features`	numeric value specifying the number of features/microbes to simulate. Default is 10.
`diff_abun_features`	numeric value specifying the number of differentially abundant features. Default is 5.
`id`	vector of length `N` that identifies repeated measurements for each unit
`time`	vector of length `N` that determines when values will be sampled for each unit
`group`	factor vector with two levels indicating the group assignment for each respective id
`ref`	character value identifying which group value to treat as control and which value to treat as treatment
`control_mean`	numeric value specifying the mean value for control subjects. all control subjects are assummed to have the same population mean value.
`sigma`	numeric value specifying the global population standard deviation for both control and treated individuals.
`rho`	value for the correlation parameter. must be between [0, 1]. see `mvrnorm_corr_gen` for details.
`corr_str`	correlation structure selected. see `mvrnorm_corr_gen` for details.
`func_form`	character value specifying the functional form for the longitduinal mean trend. see `mean_trend` for details.
`beta`	vector value specifying the parameters for the differential abundance function. see `mean_trend` for details.
`IP`	vector specifying any inflection points. depends on the type of functional form specified. see `mean_trend` for details. by default this is set to NULL.
`dis_plot`	logical argument on whether to plot the simulated data or not. by default plotting is turned off.
`plot_trend`	specifies whether to plot the true mean trend. see `mean_trend` for details.
`zero_trunc`	logical indicator designating whether simulated outcomes should be zero truncated. default is set to TRUE

Value

This function returns a list with the following objects

`Y` The full simulated feature sample matrix where each row represent a feature and each column a sample. Note that the differential and non-differential bugs are marked by row.names

Examples

```
set.seed(011520)
id_list <- lapply(seq_len(60), function(i){
obs <- sample(5:10, size=1)
id_rep <- rep(i, obs)
})

time_interval <- c(0, 10)
time_list <- lapply(id_list, function(x){
time_len <- length(x)
times <- runif(time_len, min=time_interval[1], max=time_interval[2])
times <- times[order(times)]
})

group_list <- lapply(id_list, function(x){
group_len <- length(x)
tx_ind <- sample(seq_len(2), 1)
tx_group <- ifelse(tx_ind==1, "Control", "Treatment")
groups <- rep(tx_group, group_len)
})
id <- unlist(id_list)
group <- factor(unlist(group_list), levels = c("Control", "Treatment"))
time <- unlist(time_list)

# control times
ct <- unlist(lapply(unique(id[group=="Control"]), function(x){
length(id[id==x])
}))

tt <- unlist(lapply(unique(id[group=="Treatment"]), function(x){
length(id[id==x])
}))
mean(ct)
mean(tt)

gen_norm_microbiome_obs(features=4, diff_abun_features=2,
id=id, time=time, group=group, ref="Control", control_mean=2,
               sigma=1, rho=0.7, corr_str="compound", func_form="L_up",
               beta=1, IP=5, zero_trunc=TRUE)
```

---

ggplot_spaghetti — *Spaghetti Plots using* ggplot2

Description

This function allows the user to create spaghetti plots for individuals with time varying covariates. You can also break this down into subgroups to analyze different trentds.

Usage

```
ggplot_spaghetti(
  y,
  id,
  time,
  alpha = 0.2,
  method = "loess",
  jit = 0,
  group = NULL
)
```

Arguments

`y`	This is the y-axis parameter to specify. Generally it is a continuous variable.
`id`	This is the id parameter that identifies the unique individuals or units.
`time`	This is the time vector and must be numeric.
`alpha`	Scalar value between [0,1] that specifies the transparencey of the lineplots.
`method`	Character value that specifies which type of method to use for fitting. Optional methods come from `geom_smooth` function.
`jit`	Scalar value that specifies how much you want to jitter each individual observation. Useful if many of the values share the same y values at a time point.
`group`	Specifies a grouping variable to be used, and will plot it by color on one single plot.

Details

Note that the data must be in long format.

Value

Plots a time series data by each individual/unit with group trends overlayed.

Examples

```
library(ggplot2)
num_subjects_per_group <- 15
sim_obj <- mvrnorm_sim(n_control=num_subjects_per_group,
                       n_treat=num_subjects_per_group,
                       control_mean=5, sigma=1, num_timepoints=5,
                       t_interval = c(0, 4),
                       rho=0.95, corr_str='ar1', func_form='linear',
                       beta=c(0, 0.25),
                       missing_pct=0.6, missing_per_subject=2)

with(sim_obj$df, suppressWarnings(ggplot_spaghetti(y=Y_obs, id=ID, time=time,
                                                   jit=0.1, group=group)))+
  labs(title="Simulated Microbiome Data from Multivariate Normal",
       y="Normalized Reads", x="Time") +
  scale_linetype_manual(values=c("solid","dashed"), name="Group") +
  scale_color_manual(values=c("#F8766D", "#00BFC4"), name="Group")
```

IP_form_check *Inflection point check for* `mean_trend`

Description

Inflection point check for `mean_trend`

Usage

```
IP_form_check(form, beta, IP, timepoints)
```

Arguments

`form`	character value specifying the type of time trend. Options include 'linear', 'quadratic', 'cubic', 'M', 'W', 'L_up', and 'L_down'.
`beta`	vector specifying the appropriate parameters for functional trend. See details of `mean_trend` for explanation for each form
`IP`	vector specifying the inflection points. See details of `mean_trend` for explanation for each form
`timepoints`	numeric vector specifying the points to fit the functional trend.

Value

Updated inflection point vector

mean_trend *Function for Generating Various Longitudinal Mean Trends*

Description

In order to investigate different functional forms of longitudinal differential abundance we allow the mean time trend to take a variety of forms. These functional forms include linear, quadratic, cubic, M, W, L_up, or L_down. For each form the direction/concavity/fold change can be specified using the beta parameter.

Usage

```
mean_trend(
  timepoints,
  form = c("linear", "quadratic", "cubic", "M", "W", "L_up", "L_down"),
  beta,
  IP = NULL,
  plot_trend = FALSE
)
```

Arguments

`timepoints`	numeric vector specifying the points to fit the functional trend.
`form`	character value specifying the type of time trend. Options include 'linear', 'quadratic', 'cubic', 'M', 'W', 'L_up', and 'L_down'.
`beta`	vector specifying the appropriate parameters for the equation. In the case of 'linear', beta should be a two-dimensional vector specifying the intercept and slope. See details for the further explanation of the beta value for each form.
`IP`	vector specifying the inflection points where changes occur for functional forms M, W, and L trends.
`plot_trend`	logical value indicating whether a plot should be produced for the time trend. By default this is set to TRUE.

Details

Linear Form Notes:

$$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2$$

- Sign of β_1 determines whether the trend is increasing (+) or decreasing (-)

Quadratic Form Notes:

$$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2$$

- Critical point for quadratic function occurs at the point $\frac{-\beta_1}{2\beta_2}$
- β_2 determines whether the quadratic is concave up (+) or concave down (-)

Cubic Form Notes:

$$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$$

- Point of Inflection for cubic function occurs $\frac{-\beta_2}{(3\beta_3)}$
- Critical points for cubic function occur at $\frac{-\beta_2 \pm \sqrt{\beta_2^2 - 3\beta_1\beta_3}}{3\beta_3}$
- Can generate piecewise linear trends, i.e. 'V' form, by placing either one of the IP points outside of the timepoints specified

M/W Form Notes:

- Must specify beta as (β_0, β_1) and IP as (IP_1, IP_2, IP_3)
- This form should be specified with an initial intercept, β_0, and slope, β_1, that will connect to the first point of change (IP) specified.
- Subsequent slopes are constructed such that the mean value at the second IP value and final timepoint are 0
- The mean value at the third IP is set to be equal to the calculcated mean value at the first IP based on the specified intercept and slope.
- β_0=intercept, i.e. timepoint when y=0
- β_1=slope between β_0 and IP_1

L_up Form Notes:

The structure of this form assumes that there is no trend from t_1 to IP_1. Then at the point of change specified, IP_1, there occurs a linearly increasing trend with slope equal to β_{slope} up to the last specified timepoint t_q.

- Must specify beta as (β_{slope}), and must be positive
- Specify a single point of change (IP) variable where positive trend will start
- IP must be between [t_1, t_q]

L_down Form Notes:

Similarily, the L_down form assumes that there are two region within the range of timepoints. The first region is a decreasing trend and the second region has no trend. The decreasing trend must start with a Y intercept greater than zero, and the slope must be specified as negative. There is one point of change (IP), but this is calculated automatically based on the values of the Y intercept and slope provided, IP=$-\beta_{yintercept}/\beta_{slope}$.

- Must specify beta as ($\beta_{yintercept}$, β_{slope}) where $\beta_{yintercept}$>0 and β_{slope}<0
- IP variable should be specified as NULL, if value is provided it will be ignored.

Value

This function returns a list of the following

`form` - character value repeating the form selected

`trend` - data.frame with the variables `mu` representing the estimated mean value at `timepoints` used for fitting the trend

`beta` - returning the numeric vector used to fit the functional form

Examples

```
#Quadratic Form
mean_trend(timepoints=seq(0, 6, length.out=20),
               form='quadratic', beta=1/4 * c(-1, 3, -0.5), plot_trend=TRUE)
#M Form
mean_trend(timepoints=seq(0, 10,length.out=100), form='M',
               beta=c(0, 5), IP=10 * c(1/4, 2/4, 3/4), plot_trend=TRUE)
#in this case the IP points are selected so that peaks are evenly
#distributed but this does not have to be true in general

#L_up Form
mean_trend(timepoints=seq(0, 10, length.out=100), form='L_up',
           beta=1, IP=5, plot_trend=TRUE)

#L_down Form
mean_trend(timepoints=seq(0, 10,length.out=100), form='L_down',
           beta=c(4, -0.5), IP=NULL, plot_trend=TRUE)
```

`mean_trend_beta_vec` *Create beta vector for* `mean_trend` *for all functional forms*

Description

Create beta vector for `mean_trend` for all functional forms

Usage

```
mean_trend_beta_vec(form, beta, IP, timepoints)
```

Arguments

`form`	character value specifying the type of time trend. Options include 'linear', 'quadratic', 'cubic', 'M', 'W', 'L_up', and 'L_down'.
`beta`	vector specifying the appropriate parameters for functional trend. See details of `mean_trend` for explanation for each form
`IP`	vector specifying the inflection points. See details of `mean_trend` for explanation for each form
`timepoints`	numeric vector specifying the points to fit the functional trend.

Value

Vector with beta values used to create mean_tend

`mean_trend_design_mat` *Create Design Matrix for* `mean_trend` *function*

Description

By taking in the user specified parameters, we can return a design matrix to use when creating the differential longitudinal abundance.

Usage

```
mean_trend_design_mat(form, beta, IP, timepoints)
```

Arguments

`form`	character value specifying the type of time trend. Options include 'linear', 'quadratic', 'cubic', 'M', 'W', 'L_up', and 'L_down'.
`beta`	vector specifying the appropriate parameters for functional trend. See details of `mean_trend` for explanation for each form
`IP`	vector specifying the inflection points. See details of `mean_trend` for explanation for each form
`timepoints`	numeric vector specifying the points to fit the functional trend.

Value

Numeric matrix with values that will be used to generate functional trends

---

mvrnorm_corr_gen *Generate Multivariate Random Normal Longitudinal Data*

Description

For this methodology we assume that we draw a set of n independent each with q_i observations.

Usage

```
mvrnorm_corr_gen(
  n,
  obs,
  t,
  mu,
  sigma,
  rho,
  corr_str = c("ar1", "compound", "ind"),
  zero_trunc = TRUE
)
```

Arguments

n	integer scalar representing the total number of individuals
obs	vector of length n specifying the number of observations per indivdiual.
t	vector corresponding to the timepoints for each individual.
mu	vector specifying the mean value for individuals.
sigma	scalar specifying the standard deviation for all observations.
rho	numeric scalar value between [0, 1] specifying the amount of correlation between. assumes that the correlation is consistent for all subjects.
corr_str	character value specifying the correlation structure. Currently available methods are \'ar1\', \'compound\', and \'ind\' which correspond to first-order autoregressive, compound or equicorrelation, and independence respecitvely.
zero_trunc	logical value to specifying whether the generating distribution should come from a multivariate zero truncated normal or an untruncated multivariate normal. by default we assume that zero truncation occurs since this is assummed in our microbiome setting.

Value

This function returns a list with the following objects:

`df` - data.frame object with complete outcome `Y`, subject ID, time, group, and outcome with missing data

`Y` - vector of complete outcome

`Mu` - vector of complete mean specifications used during simulation

`Sigma` - block diagonal symmetric matrix of complete data used during simulation

`N` - total number of observations

Examples

```
size <- 15
reps <- 4
N <- size*reps
mvrnorm_corr_gen(n=size, obs=rep(reps, size), t=rep(seq_len(4), size),
mu=rep(1, N), sigma=2, rho=0.9, corr_str="ar1")
```

`mvrnorm_sim` *Simulate Microbiome Longitudinal Data from Multivariate Random Normal*

Description

This function is used in the `gen_norm_microbiome` call when the user specified the method as mvrnorm.

Usage

```
mvrnorm_sim(
  n_control,
  n_treat,
  control_mean,
  sigma,
  num_timepoints,
  t_interval,
  rho,
  corr_str = c("ar1", "compound", "ind"),
  func_form = c("linear", "quadratic", "cubic", "M", "W", "L_up", "L_down"),
  beta,
  IP = NULL,
  missing_pct,
  missing_per_subject,
  miss_val = NA,
  dis_plot = FALSE,
  plot_trend = FALSE,
  zero_trunc = TRUE,
  asynch_time = FALSE
)
```

Arguments

`n_control`	integer value specifying the number of control individuals
`n_treat`	integer value specifying the number of treated individuals
`control_mean`	numeric value specifying the mean value for control subjects. all control subjects are assummed to have the same population mean value.
`sigma`	numeric value specifying the global population standard deviation for both control and treated individuals.
`num_timepoints`	either an integer value specifying the number of timepoints per subject or a vector of timepoints for each subject. If supplying a vector the lenght of the vector must equal the total number of subjects.
`t_interval`	numeric vector of length two specifying the interval of time from which to draw observatoins [t_1, t_q]. Assumed to be equally spaced over the interval unless `asynch_time` is set to TRUE.
`rho`	value for the correlation parameter. must be between [0, 1]. see `mvrnorm_corr_gen` for details.
`corr_str`	correlation structure selected. see `mvrnorm_corr_gen` for details.
`func_form`	character value specifying the functional form for the longitduinal mean trend. see `mean_trend` for details.
`beta`	vector value specifying the parameters for the differential abundance function. see `mean_trend` for details.
`IP`	vector specifying any inflection points. depends on the type of functional form specified. see `mean_trend` for details. by default this is set to NULL.
`missing_pct`	numeric value that must be between [0, \1] that specifies what percentage of the individuals will have missing values.
`missing_per_subject`	integer value specifying how many observations per subject should be dropped. note that we assume that all individuals must have baseline value, meaning that the maximum number of `missing_per_subject` is equal to `num_timepoints` - 1.
`miss_val`	value used to induce missingness from the simulated data. by default missing values are assummed to be NA but other common choices include 0.
`dis_plot`	logical argument on whether to plot the simulated data or not. by default plotting is turned off.
`plot_trend`	specifies whether to plot the true mean trend. see `mean_trend` for details.
`zero_trunc`	logical indicator designating whether simulated outcomes should be zero truncated. default is set to TRUE
`asynch_time`	logical indicator designed to randomly sample timepoints over a specified interval if set to TRUE. default is FALSE.

Value

This function returns a list with the following objects:

`df` - data.frame object with complete outcome `Y`, subject ID, time, group, and outcome with missing data

`Y` - vector of complete outcome

`Mu` - vector of complete mean specifications used during simulation

`Sigma` - block diagonal symmetric matrix of complete data used during simulation

`N` - total number of observations

`miss_data` - data.frame object that lists which ID's and timepoints were randomly selected to induce missingness

`Y_obs` - vector of outcome with induced missingness

Examples

```
num_subjects_per_group <- 20
sim_obj <- mvrnorm_sim(n_control=num_subjects_per_group,
                       n_treat=num_subjects_per_group,
                       control_mean=5, sigma=1, num_timepoints=5,
                       t_interval=c(0, 4), rho=0.95, corr_str='ar1',
                       func_form='linear', beta=c(0, 0.25),
                       missing_pct=0.6, missing_per_subject=2)
#checking the output
head(sim_obj$df)

#total number of observations is 2(num_subjects_per_group)(num_timeponts)
sim_obj$N

#there should be approximately 60% of the IDs with missing observations
length(unique(sim_obj$miss_data$miss_id))/length(unique(sim_obj$df$ID))

#checking the subject covariance structure
sim_obj$Sigma[seq_len(5), seq_len(5)]
```

`mvrnorm_sim_obs` *Simulate Microbiome Longitudinal Data from Multivariate Random Normal with Observed Data*

Description

This function is used in the `gen_norm_microbiome_obs` call.

Usage

```
mvrnorm_sim_obs(
  id,
  time,
  group,
  ref,
  control_mean,
  sigma,
  rho,
  corr_str = c("ar1", "compound", "ind"),
  func_form = c("linear", "quadratic", "cubic", "M", "W", "L_up", "L_down"),
  beta,
  IP = NULL,
  dis_plot = FALSE,
  plot_trend = FALSE,
  zero_trunc = TRUE
)
```

Arguments

`id`	vector of length `N` that identifies repeated measurements for each unit
`time`	vector of length `N` that determines when values will be sampled for each unit
`group`	factor vector with two levels indicating the group assignment for each respective id
`ref`	character value identifying which group value to treat as control and which value to treat as treatment
`control_mean`	numeric value specifying the mean value for control subjects. all control subjects are assummed to have the same population mean value.
`sigma`	numeric value specifying the global population standard deviation for both control and treated individuals.
`rho`	value for the correlation parameter. must be between [0, 1]. see `mvrnorm_corr_gen` for details.
`corr_str`	correlation structure selected. see `mvrnorm_corr_gen` for details.
`func_form`	character value specifying the functional form for the longitduinal mean trend. see `mean_trend` for details.
`beta`	vector value specifying the parameters for the differential abundance function. see `mean_trend` for details.
`IP`	vector specifying any inflection points. depends on the type of functional form specified. see `mean_trend` for details. by default this is set to NULL.
`dis_plot`	logical argument on whether to plot the simulated data or not. by default plotting is turned off.
`plot_trend`	specifies whether to plot the true mean trend. see `mean_trend` for details.
`zero_trunc`	logical indicator designating whether simulated outcomes should be zero truncated. default is set to TRUE

Value

This function returns a list with the following objects:

`df` - data.frame object with complete outcome `Y`, subject ID, time, group, and outcome with missing data

`Y` - vector of complete outcome

`Mu` - vector of complete mean specifications used during simulation

`Sigma` - block diagonal symmetric matrix of complete data used during simulation

`N` - total number of observations

Examples

```
set.seed(011520)
id_list <- lapply(seq_len(30), function(i){
obs <- sample(seq_len(10), size=1)
id_rep <- rep(i, obs)
})

time_interval <- c(0, 10)
time_list <- lapply(id_list, function(x){
time_len <- length(x)
times <- runif(time_len, min=time_interval[1], max=time_interval[2])
times <- times[order(times)]
})

group_list <- lapply(id_list, function(x){
group_len <- length(x)
tx_ind <- sample(seq_len(2), 1)
tx_group <- ifelse(tx_ind==1, "Control", "Treatment")
groups <- rep(tx_group, group_len)
})
id <- unlist(id_list)
group <- factor(unlist(group_list), levels = c("Control", "Treatment"))
time <- unlist(time_list)

# N=173 total repeated measurements
length(id)

# 15 control and 15 treated subjects
table(group[unique(id)])

# control times
ct <- unlist(lapply(unique(id[group=="Control"]), function(x){
length(id[id==x])
}))

#treatment times
tt <- unlist(lapply(unique(id[group=="Treatment"]), function(x){
length(id[id==x])
}))
```

```
# on average the treatment group has one more observation than control
mean(ct)
mean(tt)

mvrnorm_sim_obs(id=id, time=time, group=group, ref="Control", control_mean=2,
                sigma=1, rho=0.7, corr_str="compound", func_form="L_up",
                beta=1, IP=5, plot_trend=TRUE, dis_plot=TRUE, zero_trunc=TRUE)
```

sigma_corr_function *Generating the longitudinal correlation matrix for repeated observations*

Description

Generating the longitudinal correlation matrix for repeated observations

Usage

```
sigma_corr_function(t, sigma, corr_str, rho)
```

Arguments

`t`	timepoints for repeated observations
`sigma`	the standard deviation parameter for the covariance matrix
`corr_str`	the type of correlatin structure chosen. options currently available include "ar1", "compound", and "ind"
`rho`	the correlation coefficient for non-independent structures

Value

Return the covariance matrix V as a list

simulate2MRexperiment *Convert simulated output to MRexperiment object*

Description

In order to allow investigators to more easily incorporate simulated data, this package converts the raw output into an MRexperiment object used in the `metagenomeSeq` package.

Usage

```
simulate2MRexperiment(obj, missing = FALSE)
```

Arguments

obj	output from either gen_norm_microbiome or mvrnorm_sim
missing	logical indicator for objects from mvrnorm_sim. If missing = TRUE then create MRexperiment object with Y_obs else use Y.

Value

An MRexperiment object

Examples

```
bug_gen <- gen_norm_microbiome(features=6, diff_abun_features=3,
                               n_control=30, n_treat=20, control_mean=2,
                               sigma=2, num_timepoints=4, t_interval=c(0, 3),
                               rho=0.9, corr_str="compound", func_form="M",
                               beta=c(4, 3), IP=c(2, 3.3, 6),
                               missing_pct=0.2, missing_per_subject=2,
                               miss_val=0, asynch_time=TRUE)
bug_gen_MR <- simulate2MRexperiment(bug_gen)
class(bug_gen_MR)
```

---

simulate2phyloseq   *Convert simulated output to phyloseq object*

Description

This function will convert simulated data into a phyloseq object.

Usage

```
simulate2phyloseq(obj, missing = FALSE)
```

Arguments

obj	output from either gen_norm_microbiome or mvrnorm_sim
missing	logical indicator for objects from mvrnorm_sim. If missing = TRUE then create MRexperiment object with Y_obs else use Y.

Value

A phyloseq object

Examples

```
bug_gen <- gen_norm_microbiome(features=6, diff_abun_features=3,
                               n_control=30, n_treat=20, control_mean=2,
                               sigma=2, num_timepoints=4, t_interval=c(0, 3),
                               rho=0.9, corr_str="compound", func_form="M",
                               beta=c(4, 3), IP=c(2, 3.3, 6),
                               missing_pct=0.2, missing_per_subject=2,
                               miss_val=0, asynch_time=TRUE)
bug_gen_phyloseq <- simulate2MRexperiment(bug_gen)
class(bug_gen_phyloseq)
```

`timepoint_process` *Function for processing and checking the inputed timepoints*

Description

To allow for increased flexibility the user may specify the number of timepoints as either a single value or separately for each individual. There is also an added option about whether to draw the timepoints evenly spaced across the interval of interest or whether to randomly draw them.

Usage

```
timepoint_process(
  num_timepoints,
  t_interval,
  n,
  asynch_time,
  missing_per_subject
)
```

Arguments

`num_timepoints`	either an integer value specifying the number of timepoints per subject or a vector of timepoints for each subject. If supplying a vector the lenght of the vector must equal the total number of subjects.
`t_interval`	numeric vector of length two specifying the interval of time from which to draw observatoins [t_1, t_q]. Assumed to be equally spaced over the interval unless `asynch_time` is set to TRUE.
`n`	numeric value representing the total number of obserations
`asynch_time`	logical indicator designed to randomly sample timepoints over a specified interval if set to TRUE.

Details

It is assummed that there is a known time interval of interest over which samples will be collected longitudinally on subjects. This interval is specified as [t_1, t_q]. All subjects are assumed to have baseline observations, i.e., t_1.

Over this study interval each subject can have a potentially different number of measurements taken. In the most simple case we assume that all subjects will have the same number of measurements and can specify `num_timepoints` as a single scalar value. Otherwise, we must specify how many timepoints will be collected for each individual. In this latter case `num_timepoints` must have the same length as the number of subjects.

Finally, we can select whether we want the timepoints to be drawn at equal spaces over our study interal, or whether we want to randomly sample asynchronous timepoints. In the asynchronous case we randomly draw from a uniform distribution over the study interval with the restriction that the first observation must occur at t_1.

Value

Returns a list of the number of timepoints and the times for each unit

---

`trunc_bugs` *Function for inducing truncation of outcome*

Description

Function for inducing truncation of outcome

Usage

```
trunc_bugs(Y, N, Mu, Sigma, zero_trunc)
```

Arguments

`Y`	The original N x 1 vector of simulated multivariate outcomes
`N`	Total number of observations equal to sum of repeated measurements for all individuals
`Mu`	N x 1 vector representing the mean values
`Sigma`	N x N numeric matrix representing the covariance matrix for the feature
`zero_trunc`	Logical indicator whether to perform zero-truncation

Value

Potentially truncated outcome vector Y

Index