Package 'MethEvolSIM'

March 17, 2025

Title Simulate DNA Methylation Dynamics on Different Genomic Structures along Genealogies

Version 0.2

Author Sara Castillo Vicente [aut, cre], Dirk Metzler [aut, ths]

Maintainer Sara Castillo Vicente <castillo@bio.lmu.de>

Description DNA methylation is an epigenetic modification involved in genomic stability, gene regulation, development and disease.
DNA methylation occurs mainly through the addition of a methyl group to cytosines, for example to cytosines in a CpG dinucleotide context (CpG stands for a cytosine followed by a guanine).
Tissue-specific methylation patterns lead to genomic regions with different characteristic methylation levels.
E.g. in vertebrates CpG islands (regions with high CpG content) that are associated to promoter regions of
expressed genes tend to be unmethylated.
'MethEvolSIM' is a model-based simulation software for the generation and modification of cytosine methylation patterns along a given tree, which can be a genealogy of cells within an organism, a coalescent tree of DNA sequences sampled from a population, or a species tree.
The simulations are based on an extension of the model of
Grosser & Metzler (2020) <doi:10.1186/s12859-020-3438-5> and allows for changes of the methylation states at single cytosine positions as well as simultaneous changes of methylation frequencies in genomic structures like CpG islands.

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.2

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

Imports R6, ape

Depends R (>= 4.0)

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2025-03-17 07:00:03 UTC

Contents

`categorize_islandGlbSt` *Categorize Global States of CpG Islands*

Description

This function categorizes CpG islands into unmethylated, methylated, or partially methylated states based on specified thresholds.

Usage

```
categorize_islandGlbSt(meanMeth_islands, u_threshold, m_threshold)
```

Arguments

`meanMeth_islands`	A numeric vector containing the mean methylation levels for CpG islands at each tip.
`u_threshold`	A numeric value (0-1) defining the threshold for categorization as unmethylated.
`m_threshold`	A numeric value (0-1) defining the threshold for categorization as methylated.

Details

The function assigns each island a state:

"u" if mean methylation lower or equal to `u_threshold`

"m" if mean methylation greater or equal to `m_threshold`

"p" if mean methylation is in between

Value

A character vector of length equal to `meanMeth_islands`, containing "u", "p", or "m" for each island.

Examples

```
meanMeth_islands <- c(0.1, 0.4, 0.8)

categorize_islandGlbSt(meanMeth_islands, 0.2, 0.8)
```

`categorize_siteMethSt` *Categorize Methylation Frequencies Based on Thresholds*

Description

This function categorizes the values in `data[[tip]][[structure]]` into three categories:

- `0` for unmethylated sites, where values are smaller or equal to `u_threshold`.
- `0.5` for partially methylated sites, where values are between `u_threshold` and `m_threshold`.
- `1` for methylated sites, where values are larger or equal to `m_threshold`.

Usage

```
categorize_siteMethSt(data, u_threshold = 0.2, m_threshold = 0.8)
```

Arguments

`data`	A list structured as `data[[tip]][[structure]]`, where `tip` corresponds to tree tips, and `structure` corresponds to each genomic structure (e.g., island/non-island).
`u_threshold`	A numeric value representing the upper bound for values to be classified as unmethylated (`0`). Default 0.2.
`m_threshold`	A numeric value representing the lower bound for values to be classified as methylated (`1`). Default 0.8.

Details

If any value in `data[[tip]][[structure]]` is outside these categories, it is transformed based on the given thresholds.

Value

A transformed version of `data` where each value is categorized as `0` (unmethylated), `0.5` (partially methylated), or `1` (methylated).

Examples

```
data <- list(
list(c(0.1, 0.2, 0.02), c(0.05, 0.25, 0.15)), # tip 1
list(c(0.01, 0.7, 0.85), c(0.3, 0.1, 0.98)) # tip 2
)

transformed_data <- categorize_siteMethSt(data, u_threshold = 0.2, m_threshold = 0.8)
```

cftpStepGenerator *cftpStepGenerator*

Description

an R6 class representing the steps for sampling a sequence of methylation states from the equilibrium (SSEi and SSEc considered, IWE neglected) using the CFTP algorithm for a given combiStructureGenerator instance.

It is stored in the private attribute CFTP_info of combiStructureGenerator instances when calling the combiStructureGenerator$cftp() method and can be retrieved with the combiStructureGenerator$get_CFTP_info()

Public fields

`singleStr_number` Public attribute: Number of singleStr instances

`singleStr_siteNumber` Public attribute: Number of sites in singleStr instances

`CFTP_highest_rate` Public attribute: CFTP highest rate

`number_steps` Public attribute: counter of steps alredy generated

`CFTP_chosen_singleStr` Public attribute: list with vectors of equal size with chosen singleStr index at each CFTP step

`CFTP_chosen_site` Public attribute: list with vectors of equal size with chosen site index at each CFTP step

`CFTP_event` Public attribute: list with vectors of equal size with type of CFTP event at each CFTP step.

`CFTP_random` Public attribute: list with vectors of equal size with CFTP threshold at each CFTP step

`steps_perVector` Public attribute: size of vectors in lists CFTP_chosen_singleStr, CFTP_chosen_site, CFTP_event and CFTP_random

Methods

Public methods:

- `cftpStepGenerator$new()`
- `cftpStepGenerator$generate_events()`
- `cftpStepGenerator$clone()`

Method `new()`**:** Create a new instance of class cftpStepGenerator with the info of the corresponding combiStrucutre instance

Usage:

```
cftpStepGenerator$new(
  singleStr_number,
  singleStr_siteNumber,
  CFTP_highest_rate
)
```

Arguments:

`singleStr_number` Number of singleStr instances

`singleStr_siteNumber` Number of sites in singleStr instances

`CFTP_highest_rate` CFTP highest rate across all singleStr withing combiStr instance

Returns: A new instance of class cftpStepGenerator

Method `generate_events()`**:** 1: SSEi to unmethylated, 2: SSEi to partially methylated, 3: SSEi to methylated 4: SSEc copy left state, 5: SSEc copy right state

Public Method. Generates the events to apply for CFTP.

Usage:

```
cftpStepGenerator$generate_events(steps = 10000, testing = FALSE)
```

Arguments:

`steps` Integer value >=1

`testing` default FALSE. TRUE for testing output

Details: The function add steps to the existing ones. If called several times the given steps need to be higher than the sum of steps generated before.

Returns: NULL when testing FALSE. Testing output when testing TRUE.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
cftpStepGenerator$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`combiStructureGenerator`

combiStructureGenerator

Description

an R6 class representing several genomic structures. Each genomic structure contained is an object of class singleStructureGenerator. Note that default clone(deep=TRUE) fails to clone singleStructureGenerator objects contained, use method $copy() instead.

Public fields

`testing_output` Public attribute: Testing output for initialize

Methods

Public methods:

- combiStructureGenerator$new()
- combiStructureGenerator$get_singleStr()
- combiStructureGenerator$get_singleStr_number()
- combiStructureGenerator$get_singleStr_siteNumber()
- combiStructureGenerator$get_island_number()
- combiStructureGenerator$get_island_index()
- combiStructureGenerator$set_IWE_events()
- combiStructureGenerator$get_IWE_events()
- combiStructureGenerator$set_name()
- combiStructureGenerator$get_name()
- combiStructureGenerator$get_own_index()
- combiStructureGenerator$set_own_index()
- combiStructureGenerator$get_parent_index()
- combiStructureGenerator$set_parent_index()
- combiStructureGenerator$get_offspring_index()
- combiStructureGenerator$set_offspring_index()
- combiStructureGenerator$add_offspring_index()
- combiStructureGenerator$get_mu()
- combiStructureGenerator$get_id()
- combiStructureGenerator$set_id()
- combiStructureGenerator$get_sharedCounter()
- combiStructureGenerator$reset_sharedCounter()
- combiStructureGenerator$set_singleStr()
- combiStructureGenerator$copy()
- combiStructureGenerator$branch_evol()
- combiStructureGenerator$get_highest_rate()
- combiStructureGenerator$set_CFTP_info()
- combiStructureGenerator$get_CFTP_info()
- combiStructureGenerator$cftp_apply_events()
- combiStructureGenerator$cftp()
- combiStructureGenerator$clone()

Method `new()`**:** Create a new combiStructureGenerator object.

Note that this object can be generated within a treeMultiRegionSimulator object.

Usage:

```
combiStructureGenerator$new(infoStr, params = NULL, testing = FALSE)
```

Arguments:

`infoStr` A data frame containing columns 'n' for the number of sites, and 'globalState' for the favoured global methylation state. If initial equilibrium frequencies are given the dataframe must contain 3 additional columns: 'u_eqFreq', 'p_eqFreq' and 'm_eqFreq'

`params` Default NULL. When given: data frame containing model parameters.

`testing` Default FALSE. TRUE for writing in public field of new instance $testing_output

Returns: A new `combiStructureGenerator` object.

Method `get_singleStr()`**:** Public method: Get one singleStructureGenerator object in $singleStr

Usage:

```
combiStructureGenerator$get_singleStr(i)
```

Arguments:

`i` index of the singleStructureGenerator object in $singleStr

Returns: the singleStructureGenerator object in $singleStr with index i

Method `get_singleStr_number()`**:** Public method: Get number of singleStructureGenerator objects in $singleStr

Usage:

```
combiStructureGenerator$get_singleStr_number()
```

Returns: number of singleStructureGenerator object contained in $singleStr

Method `get_singleStr_siteNumber()`**:** Public method: Get number of sites in all singleStructureGenerator objects

Usage:

```
combiStructureGenerator$get_singleStr_siteNumber()
```

Returns: number of sites in all singleStructureGenerator objects

Method `get_island_number()`**:** Public method: Get number of singleStructureGenerator objects in $singleStr with $globalState "U" (CpG islands)

Usage:

```
combiStructureGenerator$get_island_number()
```

Returns: number of singleStructureGenerator in $singleStr objects with $globalState "U" (CpG islands)

Method `get_island_index()`**:** Public method: Get index of singleStructureGenerator objects in $singleStr with $globalState "U" (CpG islands)

Usage:

```
combiStructureGenerator$get_island_index()
```

Returns: index of singleStructureGenerator objects in $singleStr with $globalState "U" (CpG islands)

Method `set_IWE_events()`**:** Public method: Set information of the IWE events sampled in a tree branch

Usage:

```
combiStructureGenerator$set_IWE_events(a)
```

Arguments:

a value to which IWE_events should be set

Returns: NULL

Method `get_IWE_events()`**:** Public method: Get information of the IWE events sampled in a tree branch

Usage:

```
combiStructureGenerator$get_IWE_events()
```

Returns: information of the IWE events sampled in a tree branch

Method `set_name()`**:** Public method: Set the name of the leaf if evolutionary process (simulated from class treeMultiRegionSimulator) ends in a tree leaf.

Usage:

```
combiStructureGenerator$set_name(a)
```

Arguments:

a value to which name should be set

Returns: NULL

Method `get_name()`**:** Public method: Get the name of the leaf if evolutionary process (simulated from class treeMultiRegionSimulator) ended in a tree leaf.

Usage:

```
combiStructureGenerator$get_name()
```

Returns: Name of the leaf if evolutionary process (simulated from class treeMultiRegionSimulator) ended in a tree leaf. For iner tree nodes return NULL

Method `get_own_index()`**:** Public method: Set own branch index in the tree along which the evolutionary process is simulated (from class treeMultiRegionSimulator).

Usage:

```
combiStructureGenerator$get_own_index()
```

Returns: NULL

Method `set_own_index()`**:** Public method: Get own branch index in the tree along which the evolutionary process is simulated (from class treeMultiRegionSimulator).

Usage:

```
combiStructureGenerator$set_own_index(i)
```

Arguments:

i index of focal object

Returns: Own branch index in the tree along which the evolutionary process is simulated (from class treeMultiRegionSimulator).

Method `get_parent_index()`**:** Public method: Get parent branch index in the tree along which the evolutionary process is simulated (from class treeMultiRegionSimulator).

Usage:

```
combiStructureGenerator$get_parent_index()
```

Returns: Parent branch index in the tree along which the evolutionary process is simulated (from class treeMultiRegionSimulator).

Method `set_parent_index()`**:** Public method: Set parent branch index in the tree along which the evolutionary process is simulated (from class treeMultiRegionSimulator).

Usage:

```
combiStructureGenerator$set_parent_index(i)
```

Arguments:

`i` set parent_index to this value

Returns: NULL

Method `get_offspring_index()`**:** Public method: Get offspring branch index in the tree along which the evolutionary process is simulated (from class treeMultiRegionSimulator).

Usage:

```
combiStructureGenerator$get_offspring_index()
```

Returns: Offspring branch index in the tree along which the evolutionary process is simulated (from class treeMultiRegionSimulator).

Method `set_offspring_index()`**:** Public method: Set offspring branch index in the tree along which the evolutionary process is simulated (from class treeMultiRegionSimulator).

Usage:

```
combiStructureGenerator$set_offspring_index(i)
```

Arguments:

`i` set offspring_index to this value

Returns: NULL

Method `add_offspring_index()`**:** Public method: Add offspring branch index in the tree along which the evolutionary process is simulated (from class treeMultiRegionSimulator).

Usage:

```
combiStructureGenerator$add_offspring_index(i)
```

Arguments:

`i` index to be added

Returns: NULL

Method `get_mu()`**:** Public method.

Usage:

```
combiStructureGenerator$get_mu()
```

Returns: Model parameter for the rate of the IWE evolutionary process (per island and branch length).

Method `get_id()`**:** Public method. Get the unique ID of the instance

Usage:

```
combiStructureGenerator$get_id()
```

Returns: A numeric value representing the unique ID of the instance.

Method `set_id()`**:** Public method. Set the unique ID of the instance

Usage:

```
combiStructureGenerator$set_id(id)
```

Arguments:

`id` integer value to identificate the combiStructure instance

Returns: A numeric value representing the unique ID of the instance.

Method `get_sharedCounter()`**:** Public method. Get the counter value from the shared environment between instances of combiStructureGenerator class

Usage:

```
combiStructureGenerator$get_sharedCounter()
```

Returns: Numeric counter value.

Method `reset_sharedCounter()`**:** Public method. Reset the counter value of the shared environment between instances of combiStructureGenerator class

Usage:

```
combiStructureGenerator$reset_sharedCounter()
```

Returns: NULL

Method `set_singleStr()`**:** Public method: Clone each singleStructureGenerator object in $singleStr

Usage:

```
combiStructureGenerator$set_singleStr(singStrList)
```

Arguments:

`singStrList` object to be cloned

Returns: NULL

Method `copy()`**:** Public method: Clone combiStructureGenerator object and all singleStructureGenerator objects in it

Usage:

```
combiStructureGenerator$copy()
```

Returns: cloned combiStructureGenerator object

Method `branch_evol()`**:** Simulate CpG dinucleotide methylation state evolution along a tree branch. The function samples the IWE events on the tree branch and simulates the evolution through the SSE and IWE processes.

Usage:

```
combiStructureGenerator$branch_evol(branch_length, dt, testing = FALSE)
```

Arguments:

`branch_length` Length of the branch.

`dt` Length of SSE time steps.

`testing` Default FALSE. TRUE for testing purposes.

Details: It handles both cases where IWE events are sampled or not sampled within the branch.

Returns: Default NULL. If testing = TRUE it returns information for testing purposes.

Method `get_highest_rate()`**:** Public Method. Gets the highest rate among all singleStructure-Generator objects for CFTP.

Usage:

```
combiStructureGenerator$get_highest_rate()
```

Returns: Highest rate value.

Method `set_CFTP_info()`**:** Public Method. Sets a cftpStepGenerator instance asthe CFTP info.

Usage:

```
combiStructureGenerator$set_CFTP_info(CFTP_instance)
```

Arguments:

`CFTP_instance` CFTP info.

Returns: NULL

Method `get_CFTP_info()`**:** Public Method. Gets the CFTP info.

Usage:

```
combiStructureGenerator$get_CFTP_info()
```

Returns: CFTP info.

Method `cftp_apply_events()`**:** Public Method. Applies the CFTP events.

Usage:

```
combiStructureGenerator$cftp_apply_events(testing = FALSE)
```

Arguments:

`testing` default FALSE. TRUE for testing output

Returns: NULL when testing FALSE. Testing output when testing TRUE.

Method `cftp()`**:** Public Method. Applies the CFTP algorithm to evolve a structure and checks for convergence by comparing methylation states.

This method generates CFTP steps until the methylation sequences of the current structure and a cloned structure become identical across all singleStr instances or a step limit is reached. If the step limit is exceeded, an approximation method is applied to finalize the sequence.

Usage:

```
combiStructureGenerator$cftp(
  steps = 10000,
  step_limit = 327680000,
  testing = FALSE
)
```

Arguments:

`steps` minimum number of steps to apply (default 10000).

`step_limit` maximum number of steps before applying an approximation method (default 327680000 corresponding to size of CFTP info of approx 6.1 GB). If this limit is reached, the algorithm stops and an approximation is applied.

`testing` logical. If TRUE, returns additional testing output including the current structure, the cloned structure, the counter value, total steps, and the chosen site for the CFTP events. Default is FALSE.

Returns: NULL when testing is FALSE. If testing is TRUE, returns a list with:

- `self`: the current object after applying the CFTP algorithm.
- `combi_m`: a deep cloned object with applied CFTP events.
- `counter`: the number of iterations performed.
- `total_steps`: the number of steps applied by the CFTP algorithm.
- `CFTP_chosen_site`: the site selected during the CFTP event application.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
combiStructureGenerator$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`compare_CherryFreqs` *Compare Methylation Frequencies Between Two Tips*

Description

Performs a chi-squared test to compare the distribution of methylation states (unmethylated `0`, partially-methylated `0.5`, and methylated `1`) between two cherry tips. A cherry is a pair of leaf nodes (also called tips or terminal nodes) in a phylogenetic tree that share a direct common ancestor.

Usage

```
compare_CherryFreqs(tip1, tip2, testing = FALSE)
```

Arguments

`tip1`	A numeric vector representing methylation states (`0`, `0.5`, `1`) at tip 1.
`tip2`	A numeric vector representing methylation states (`0`, `0.5`, `1`) at tip 2.
`testing`	Logical; if `TRUE`, returns additional intermediate data including the contingency table and test result.

Details

The function uses `simulate.p.value = TRUE` in `chisq.test` to compute the p-value via Monte Carlo simulation to improve reliability regardless of whether the expected frequencies meet the assumptions of the chi-squared test (i.e., expected counts of at least 5 in each category).

Value

If `testing = TRUE`, returns a list with the contingency table and chi-squared test results. Otherwise, returns the p-value of the test.

Examples

```
tip1 <- c(0, 0, 1, 0.5, 1, 0.5)
tip2 <- c(0, 1, 1, 0, 0.5, 0.5)
compare_CherryFreqs(tip1, tip2)
```

---

computeFitch_islandGlbSt

Compute Fitch Parsimony for Global Methylation States at CpG Islands

---

Description

This function categorizes CpG islands into methylation states and applies Fitch parsimony to estimate the minimum number of state changes in a phylogenetic tree.

Usage

```
computeFitch_islandGlbSt(
  index_islands,
  data,
  tree,
  u_threshold,
  m_threshold,
  testing = FALSE
)
```

Arguments

`index_islands`	A numeric vector specifying the indices of genomic structures corresponding to islands.
`data`	A list containing methylation states at tree tips, structured as `data[[tip]][[structure]]`, where each tip has the same number of structures, and each structure has the same number of sites across tips.
`tree`	A rooted binary tree in Newick format (character string) or as an `ape` phylo object. Must have at least two tips.
`u_threshold`	A numeric threshold value (0-1) defining the unmethylated category.
`m_threshold`	A numeric threshold value (0-1) defining the methylated category.
`testing`	Logical; if `TRUE`, returns additional intermediate data.

Details

The function first validates the input data and categorizes CpG islands using `categorize_islandGlbSt`. It then structures the data into a matrix matching tree tip labels and applies `compute_fitch` to infer the minimum number of changes.

Value

If `testing = TRUE`, returns a list containing the categorized data matrix; otherwise, returns a numeric vector of minimum state changes.

Examples

```
tree <- "((a:1,b:1):2,(c:2,d:2):1.5);"

data <- list(
  list(rep(1,10), rep(0,5), rep(1,8)),
  list(rep(1,10), rep(0.5,5), rep(0,8))
)

index_islands <- c(1,3)

computeFitch_islandGlbSt(index_islands, data, tree, 0.2, 0.6)
```

`compute_fitch` *Compute Fitch Parsimony for Methylation Categories*

Description

This function applies Fitch parsimony to determine the minimum number of changes required for methylation categories at tree tips.

Usage

```
compute_fitch(tree, meth, input_control = TRUE)
```

Arguments

`tree`	A rooted binary tree in Newick format (character string) or as an `ape` phylo object. Must have at least two tips.
`meth`	A matrix of methylation categories at the tree tips, with rows corresponding to tips (names matching tree tip labels) and columns corresponding to sites or structures.
`input_control`	Logical; if `TRUE`, validates input consistency.

Value

A list containing:

`optStateSet` A list of sets of optimal states for the root at each site/structure.

`minChange_number` A numeric vector indicating the minimum number of changes for each site.

Examples

```
tree <- "((a:1,b:1):2,(c:2,d:2):1.5);"

meth <- matrix(c("u", "m", "p", "u", "p", "m", "m", "u"),
               nrow=4, byrow=TRUE, dimnames=list(c("a", "b", "c", "d")))

compute_fitch(tree, meth)
```

---

compute_meanCor_i *Compute the Mean Correlation of Methylation State in Islands*

Description

This function calculates the mean correlation of methylation states within island structures, allowing to exclude the shores.

Usage

```
compute_meanCor_i(
  index_islands,
  minN_CpG,
  shore_length,
  data,
  sample_n,
  categorized_data = FALSE
)
```

Arguments

`index_islands`	A vector containing the structural indices for islands.
`minN_CpG`	The minimum number of central CpGs required for computation.
`shore_length`	The number of CpGs at each side of an island to exclude (shores).
`data`	A list containing methylation states at tree tips for each genomic structure (island / non-island) For a single tip: `data[[structure]]`. For multiple tips: `data[[tip]][[structure]]`. Each element contains the methylation states at the sites in a given tip and structure represented as 0, 0.5 or 1 (for unmethylated, partially-methylated and methylated). If methylation states are not represented as 0, 0.5, 1 they are categorized as 0 when value equal or under 0.2 0.5 when value between 0.2 and 0.8 and 1 when value over 0.8. For customized categorization thresholds use `categorize_siteMethSt`

`sample_n` The number of tips (samples) to process.

`categorized_data`
Logical defaulted to FALSE. TRUE to skip redundant categorization when methylation states are represented as 0, 0.5, and 1.

Details

The function processes only islands with a minimum length equal to `2 * shore_length + minN_CpG`. If none has minimum length, returns NA

Value

A numeric value representing the mean correlation of methylation states in the central CpGs of islands.

Examples

```
# Example usage:
index_islands <- c(1, 2)
data <- list(
  list(c(0, 1, 0.5, 1, 0.5, 0), c(0.5, 0.5, 1, 1, 0, 0)), # tip 1
  list(c(1, 0, 1, 1, 0.5, 0), c(1, 1, 0.5, 0.5, 0, 1))   # tip 2
)
minN_CpG <- 2
shore_length <- 1
sample_n <- 2
compute_meanCor_i(index_islands, minN_CpG, shore_length, data, sample_n,
                  categorized_data = TRUE)
```

---

compute_meanCor_ni *Compute the Mean Correlation of Methylation State in Non-islands*

Description

This function calculates the mean correlation of methylation states within non-island structures, allowing to exclude the shores.

Usage

```
compute_meanCor_ni(
  index_nonislands,
  minN_CpG,
  shore_length,
  data,
  sample_n,
  categorized_data = FALSE
)
```

Arguments

index_nonislands	A vector containing the structural indices for non-islands.
minN_CpG	The minimum number of central CpGs required for computation.
shore_length	The number of CpGs at each side of an non-island to exclude (shores).
data	A list containing methylation states at tree tips for each genomic structure (island / non-island) For a single tip: `data[[structure]]`. For multiple tips: `data[[tip]][[structure]]`. Each element contains the methylation states at the sites in a given tip and structure represented as 0, 0.5 or 1 (for unmethylated, partially-methylated and methylated). If methylation states are not represented as 0, 0.5, 1 they are categorized as 0 when value equal or under 0.2 0.5 when value between 0.2 and 0.8 and 1 when value over 0.8. For customized categorization thresholds use `categorize_siteMethSt`
sample_n	The number of tips (samples) to process.
categorized_data	Logical defaulted to FALSE. TRUE to skip redundant categorization when methylation states are represented as 0, 0.5, and 1.

Details

The function processes only non-islands with a minimum length equal to `2 * shore_length + minN_CpG`. If none has minimum length, returns NA

Value

A numeric value representing the mean correlation of methylation states in the central CpGs of non-islands.

Examples

```
# Example usage:
index_nonislands <- c(1, 2)
data <- list(
  list(c(0, 1, 0.5, 1, 0.5, 0), c(0.5, 0.5, 1, 1, 0, 0)), # tip 1
  list(c(1, 0, 1, 1, 0.5, 0), c(1, 1, 0.5, 0.5, 0, 1))   # tip 2
)
minN_CpG <- 2
shore_length <- 1
sample_n <- 2
compute_meanCor_ni(index_nonislands, minN_CpG, shore_length, data, sample_n,
                   categorized_data = TRUE)
```

`countSites_cherryMethDiff` *Count Methylation Differences Between Cherry Pairs*

Description

This function calculates the number of methylation differences between pairs of cherry tips in a phylogenetic tree. A cherry is a pair of leaf nodes that share a direct common ancestor. The function quantifies full and half methylation differences for each genomic structure (e.g., island/non-island) across all sites.

Usage

```
countSites_cherryMethDiff(
  cherryDist,
  data,
  categorized_data = FALSE,
  input_control = TRUE
)
```

Arguments

`cherryDist` A data frame containing pairwise distances between the tips of a phylogenetic tree that form cherries. This should be as the output of `get_cherryDist`, and must include the following columns:

- **first_tip_name** A character string representing the name of the first tip in the cherry.
- **second_tip_name** A character string representing the name of the second tip in the cherry.
- **first_tip_index** An integer representing the index of the first tip in the cherry.
- **second_tip_index** An integer representing the index of the second tip in the cherry.
- **dist** A numeric value representing the sum of the branch lengths between the two tips (i.e., the distance between the cherries).

`data` A list containing methylation states at tree tips for each genomic structure (e.g., island/non-island). The data should be structured as `data[[tip]][[structure]]`, where each structure has the same number of sites across tips. The input data must be prefiltered to ensure CpG sites are represented consistently across different tips. Each element contains the methylation states at the sites in a given tip and structure represented as 0, 0.5 or 1 (for unmethylated, partially-methylated and methylated). If methylation states are not represented as 0, 0.5, 1 they are categorized as 0 when value equal or under 0.2 0.5 when value between 0.2 and 0.8 and 1 when value over 0.8. For customized categorization thresholds use `categorize_siteMethSt`

`categorized_data`
: Logical defaulted to FALSE. TRUE to skip redundant categorization when methylation states are represented as 0, 0.5, and 1.

`input_control`
: A logical value indicating whether to validate the input data. If `TRUE` (default), the function checks that the data has the required structure. It ensures that the number of tips is sufficient and that the data structure is consistent across tips and structures. If `FALSE`, the function assumes the tree is already valid and skips the validation step.

Details

The function first verifies that `cherryDist` contains the required columns and has at least one row. It also ensures that `data` contains a sufficient number of tips and that all structures have the same number of sites. The function then iterates over each cherry and genomic structure to compute the number of full and half methylation differences between the two tips of each cherry.

Value

A data frame with one row per cherry, containing the following columns:

tip_names A character string representing the names of the two tips in the cherry, concatenated with a hyphen.

tip_indices A character string representing the indices of the two tips in the cherry, concatenated with a hyphen.

dist A numeric value representing the sum of the branch distances between the cherry tips.

One column for each structure named with the structure number followed by _f An integer count of the sites with a full methylation difference (where one tip is methylated and the other is unmethylated) for the given structure.

One column for each structure named with the structure number followed by _h An integer count of the sites with a half methylation difference (where one tip is partially methylated and the other is either fully methylated or unmethylated) for the given structure.

Examples

```
# Example data setup

data <- list(
  list(c(0, 1, 0.5, 0), c(1, 1, 0, 0.5)),
  list(c(1, 0, 0.5, 1), c(0, 1, 0.5, 0.5))
)

tree <- "(tip1:0.25, tip2:0.25);"

cherryDist <- get_cherryDist(tree)

countSites_cherryMethDiff(cherryDist, data, categorized_data = TRUE)
```

count_upm *Count Methylation States*

Description

This internal function counts the number of sites with unmethylated, partially-methylated, and methylated states in a given vector.

Usage

```
count_upm(data)
```

Arguments

data	A numeric vector with methylation values: 0 (unmethylated), 0.5 (partially-methylated), and 1 (methylated).

Value

An integer vector of length 3 containing counts of unmethylated, partially-methylated, and methylated sites, respectively.

freqSites_cherryMethDiff *Compute Methylation Frequency Differences Between Cherry Pairs*

Description

This function calculates the frequency of methylation differences between pairs of cherry tips in a phylogenetic tree. A cherry is a pair of leaf nodes that share a direct common ancestor. The function quantifies full and half methylation differences for each genomic structure (e.g., island/non-island) across all sites and normalizes these counts by the number of sites per structure to obtain frequencies.

Usage

```
freqSites_cherryMethDiff(
  tree,
  data,
  categorized_data = FALSE,
  input_control = TRUE
)
```

Arguments

`tree`	A phylogenetic tree object. The function assumes it follows an appropriate format for downstream processing.
`data`	A list containing methylation states at tree tips for each genomic structure (e.g., island/non-island). The data should be structured as `data[[tip]][[structure]]`, where each structure has the same number of sites across tips. The input data must be prefiltered to ensure CpG sites are represented consistently across different tips. Each element contains the methylation states at the sites in a given tip and structure represented as 0, 0.5 or 1 (for unmethylated, partially-methylated and methylated). If methylation states are not represented as 0, 0.5, 1 they are categorized as 0 when value equal or under 0.2 0.5 when value between 0.2 and 0.8 and 1 when value over 0.8. For customized categorization thresholds use `categorize_siteMethSt`
`categorized_data`	Logical defaulted to FALSE. TRUE to skip redundant categorization when methylation states are represented as 0, 0.5, and 1.
`input_control`	A logical value indicating whether to validate the input data. If `TRUE` (default), the function checks that the data has the required structure. It ensures that the number of tips is sufficient and that the data structure is consistent across tips and structures. If `FALSE`, the function assumes the tree is already valid and skips the validation step.

Details

The function first validates the tree structure and extracts pairwise distances between cherry tips. It then counts methylation differences using `countSites_cherryMethDiff` and normalizes these counts by the number of sites per structure to compute frequencies. The resulting data frame provides a per-cherry frequency of methylation differences (half or full difference) across different genomic structures.

Value

A data frame with one row per cherry, containing the following columns:

tip_names A character string representing the names of the two tips in the cherry, concatenated with a hyphen.

tip_indices A character string representing the indices of the two tips in the cherry, concatenated with a hyphen.

dist A numeric value representing the sum of the branch distances between the cherry tips.

One column for each structure named with the structure number followed by _f A numeric value representing the frequency of sites with a full methylation difference (where one tip is methylated and the other is unmethylated) for the given structure.

One column for each structure named with the structure number followed by _h A numeric value representing the frequency of sites with a half methylation difference (where one tip is partially methylated and the other is either fully methylated or unmethylated) for the given structure.

Examples

```
# Example data setup

data <- list(
list(rep(1,10), rep(0,5), rep(1,8)),
list(rep(1,10), rep(0.5,5), rep(0,8)),
list(rep(1,10), rep(0.5,5), rep(0,8)),
list(c(rep(0,5), rep(0.5, 5)), c(0, 0, 1, 1, 1), c(0.5, 1, rep(0, 6))))

tree <- "((a:1.5,b:1.5):2,(c:2,d:2):1.5);"

freqSites_cherryMethDiff(tree, data, categorized_data = TRUE)
```

---

get_cherryDist *Get Cherry Pair Distances from a Phylogenetic Tree*

Description

This function computes the pairwise distances between the tips of a phylogenetic tree that are part of cherries. A cherry is a pair of leaf nodes (also called tips or terminal nodes) in a phylogenetic tree that share a direct common ancestor. In other words, if two leaves are connected to the same internal node and no other leaves are connected to that internal node, they form a cherry. The distance is calculated as the sum of the branch lengths between the two cherry tips.

Usage

```
get_cherryDist(tree, input_control = TRUE)
```

Arguments

`tree`	A tree in Newick format (as a character string) or an object of class `phylo` from the `ape` package. If the input is a character string, it must follow the Newick or New Hampshire format (e.g. `"((tip_1:1,tip_2:1):5,tip_3:6);"`). If an object of class `phylo` is provided, it should represent a valid phylogenetic tree.
`input_control`	A logical value indicating whether to validate the input tree. If `TRUE` (default), the function checks that the tree is in a valid format and has at least two tips. If `FALSE`, the function assumes the tree is already valid and skips the validation step.

Details

The function first checks if the input is either a character string in the Newick format or an object of class `phylo`, unless `input_control` is set to `FALSE`. It then computes the pairwise distances between the tips in the tree and identifies the sister pairs (cherries). The distance between each cherry is the sum of the branch lengths leading to the sister tips.

The tips of each cherry are identified by their names and indices. The tip indices correspond to (a) the index from left to right on the Newick string, (b) the order of the tip label in the `phylo_object$tip.label`,

and (c) the index in the methylation data list (`data[[tip]][[structure]]`) as obtained with the function `simulate_evolData()` when the given tree has several tips.

If the tree is provided in Newick format, it will be parsed using the `ape::read.tree` function.

Value

A data frame with five columns:

`first_tip_name` A character string representing the name of the first tip in the cherry.

`second_tip_name`
A character string representing the name of the second tip in the cherry.

`first_tip_index`
An integer representing the index of the first tip in the cherry.

`second_tip_index`
An integer representing the index of the second tip in the cherry.

`dist` A numeric value representing the sum of the branch lengths between the two tips (i.e., the distance between the cherries).

Examples

```
# Example of a tree in Newick format

newick_tree <- "((a:1,b:2):5,c:6);"

get_cherryDist(newick_tree)

# Example of using a phylo object from ape

library(ape)
tree_phylo <- read.tree(text = "((a:1,b:1):5,c:6);")

get_cherryDist(tree_phylo)
```

---

get_islandMeanFreqM *Calculate the Mean Frequency of Methylated Sites in Islands*

Description

This function computes the mean frequency of methylated sites (with methylation state 1) for a set of structures identified as islands.

Usage

```
get_islandMeanFreqM(index_islands, data, sample_n, categorized_data = FALSE)
```

Arguments

index_islands	A vector containing the structural indices for islands.
data	A list containing methylation states at tree tips for each genomic structure (island / non-island) For a single tip: `data[[structure]]`. For multiple tips: `data[[tip]][[structure]]`. Each element contains the methylation states at the sites in a given tip and structure represented as 0, 0.5 or 1 (for unmethylated, partially-methylated and methylated). If methylation states are not represented as 0, 0.5, 1 they are categorized as 0 when value equal or under 0.2 0.5 when value between 0.2 and 0.8 and 1 when value over 0.8. For customized categorization thresholds use `categorize_siteMethSt`
sample_n	The number of samples (tips) to process.
categorized_data	Logical defaulted to FALSE. TRUE to skip redundant categorization when methylation states are represented as 0, 0.5, and 1.

Value

A numeric value representing the mean frequency of methylated sites in the islands.

Examples

```
# Example usage:
index_islands <- c(1, 3)
data <- list(
  list(c(0.5, 1, 0.5), c(0, 0.5, 1), c(1, 0, 0.5)), # tip 1
  list(c(0.5, 0.5, 0), c(1, 0.5, 0.5), c(0.5, 0.5, 1)) # tip 2
)
sample_n <- 2
get_islandMeanFreqM(index_islands, data, sample_n, categorized_data = TRUE)
```

---

get_islandMeanFreqP *Calculate the Mean Frequency of Partially Methylated Sites in Islands*

Description

This function computes the mean frequency of partially methylated sites (with methylation state 0.5) for the set of genomic structures identified as islands.

Usage

```
get_islandMeanFreqP(index_islands, data, sample_n, categorized_data = FALSE)
```

Arguments

index_islands	A vector containing the structural indices for islands.
data	A list containing methylation states at tree tips for each genomic structure (island / non-island) For a single tip: `data[[structure]]`. For multiple tips: `data[[tip]][[structure]]`. Each element contains the methylation states at the sites in a given tip and structure represented as 0, 0.5 or 1 (for unmethylated, partially-methylated and methylated). If methylation states are not represented as 0, 0.5, 1 they are categorized as 0 when value equal or under 0.2 0.5 when value between 0.2 and 0.8 and 1 when value over 0.8. For customized categorization thresholds use `categorize_siteMethSt`
sample_n	The number of samples (tips) to process.
categorized_data	Logical defaulted to FALSE. TRUE to skip redundant categorization when methylation states are represented as 0, 0.5, and 1.

Value

A numeric value representing the mean frequency of partially methylated sites in the islands.

Examples

```
# Example usage:
index_islands <- c(1, 3)
data <- list(
  list(c(0.5, 1, 0.5), c(0, 0.5, 1), c(1, 0, 0.5)), # tip 1
  list(c(0.5, 0.5, 0), c(1, 0.5, 0.5), c(0.5, 0.5, 1)) # tip 2
)
sample_n <- 2
get_islandMeanFreqP(index_islands, data, sample_n, categorized_data = TRUE)
```

---

get_islandSDFreqM *Calculate the Mean Standard Deviation of Methylated Sites in Islands*

Description

This function computes the mean standard deviation of methylated sites (with methylation state 1) for a set of genomic structures identified as islands.

Usage

```
get_islandSDFreqM(index_islands, data, sample_n, categorized_data = FALSE)
```

Arguments

index_islands	A vector containing the structural indices for islands.
data	A list containing methylation states at tree tips for each genomic structure (island / non-island) For a single tip: `data[[structure]]`. For multiple tips: `data[[tip]][[structure]]`. Each element contains the methylation states at the sites in a given tip and structure represented as 0, 0.5 or 1 (for unmethylated, partially-methylated and methylated). If methylation states are not represented as 0, 0.5, 1 they are categorized as 0 when value equal or under 0.2 0.5 when value between 0.2 and 0.8 and 1 when value over 0.8. For customized categorization thresholds use `categorize_siteMethSt`
sample_n	The number of tips (samples) to process.
categorized_data	Logical defaulted to FALSE. TRUE to skip redundant categorization when methylation states are represented as 0, 0.5, and 1.

Value

A numeric value representing the mean standard deviation of methylated sites in the islands.

Examples

```
# Example usage:
index_islands <- c(1, 3)
data <- list(
  list(c(0.5, 1, 0.5), c(0, 0.5, 1), c(1, 0, 0.5)), # tip 1
  list(c(0.5, 0.5, 0), c(1, 0.5, 0.5), c(0.5, 0.5, 1)) # tip 2
)
sample_n <- 2
get_islandSDFreqM(index_islands, data, sample_n, categorized_data = TRUE)
```

---

get_islandSDFreqP *Calculate the Mean Standard Deviation of Partially Methylated Sites in Islands*

Description

This function computes the mean standard deviation of partially methylated sites (with methylation state 0.5) for a set of genomic structures identified as islands.

Usage

```
get_islandSDFreqP(index_islands, data, sample_n, categorized_data = FALSE)
```

Arguments

index_islands	A vector containing the structural indices for islands.
data	A list containing methylation states at tree tips for each genomic structure (island / non-island) For a single tip: `data[[structure]]`. For multiple tips: `data[[tip]][[structure]]`. Each element contains the methylation states at the sites in a given tip and structure represented as 0, 0.5 or 1 (for unmethylated, partially-methylated and methylated). If methylation states are not represented as 0, 0.5, 1 they are categorized as 0 when value equal or under 0.2 0.5 when value between 0.2 and 0.8 and 1 when value over 0.8. For customized categorization thresholds use `categorize_siteMethSt`
sample_n	The number of samples (tips) to process.
categorized_data	Logical defaulted to FALSE. TRUE to skip redundant categorization when methylation states are represented as 0, 0.5, and 1.

Value

A numeric value representing the mean standard deviation of partially methylated sites in the islands.

Examples

```
# Example usage:
index_islands <- c(1, 3)
data <- list(
  list(c(0.5, 1, 0.5), c(0, 0.5, 1), c(1, 0, 0.5)), # tip 1
  list(c(0.5, 0.5, 0), c(1, 0.5, 0.5), c(0.5, 0.5, 1)) # tip 2
)
sample_n <- 2
get_islandSDFreqP(index_islands, data, sample_n, categorized_data = TRUE)
```

`get_meanMeth_islands` *Compute the Mean Methylation of CpG Islands*

Description

This function calculates the mean methylation level for CpG islands across all tree tips.

Usage

```
get_meanMeth_islands(index_islands, data)
```

Arguments

index_islands	A numeric vector specifying the indices of genomic structures corresponding to islands.
data	A list containing methylation states at tree tips for each genomic structure (e.g., island/non-island). The data should be structured as `data[[tip]][[structure]]`, where each tip has the same number of structures, and each structure has the same number of sites across tips.

Value

A list where each element corresponds to a tree tip and contains a numeric vector representing the mean methylation levels for the indexed CpG islands.

Examples

```
# Example data setup

data <- list(
  # Tip 1
  list(rep(1,10), rep(0,5), rep(1,8)),
  # Tip 2
  list(rep(1,10), rep(0.5,5), rep(0,8))
)

index_islands <- c(1,3)

get_meanMeth_islands(index_islands, data)
```

---

get_nonislandMeanFreqM

Calculate the Mean Frequency of Methylated Sites in Non-Islands

---

Description

This function computes the mean frequency of methylated sites (with methylation state 1) for a set of structures identified as non-islands.

Usage

```
get_nonislandMeanFreqM(
  index_nonislands,
  data,
  sample_n,
  categorized_data = FALSE
)
```

Arguments

`index_nonislands`
: A vector containing the structural indices for non-islands.

`data`
: A list containing methylation states at tree tips for each genomic structure (island / non-island) For a single tip: `data[[structure]]`. For multiple tips: `data[[tip]][[structure]]`. Each element contains the methylation states at the sites in a given tip and structure represented as 0, 0.5 or 1 (for unmethylated, partially-methylated and methylated). If methylation states are not represented as 0, 0.5, 1 they are categorized as 0 when value equal or under 0.2 0.5 when value between 0.2 and 0.8 and 1 when value over 0.8. For customized categorization thresholds use `categorize_siteMethSt`

`sample_n`
: The number of samples (tips) to process.

`categorized_data`
: Logical defaulted to FALSE. TRUE to skip redundant categorization when methylation states are represented as 0, 0.5, and 1.

Value

A numeric value representing the mean frequency of methylated sites in the non-islands.

Examples

```
# Example usage:
index_nonislands <- c(1, 3)
data <- list(
  list(c(1, 0, 1), c(0.5, 1, 1), c(1, 0, 0.5)), # tip 1
  list(c(1, 0.5, 1), c(0.5, 1, 1), c(1, 0.5, 0.5)) # tip 2
)
sample_n <- 2
get_nonislandMeanFreqM(index_nonislands, data, sample_n, categorized_data = TRUE)
```

---

get_nonislandMeanFreqP

Calculate the Mean Frequency of Partially Methylated Sites in Non-Islands

---

Description

This function computes the mean frequency of partially methylated sites (with methylation state 0.5) for a set of genomic structures identified as non-islands.

Usage

```
get_nonislandMeanFreqP(
  index_nonislands,
  data,
  sample_n,
  categorized_data = FALSE
)
```

Arguments

index_nonislands	A vector containing the structural indices for non-islands.
data	A list containing methylation states at tree tips for each genomic structure (island / non-island) For a single tip: `data[[structure]]`. For multiple tips: `data[[tip]][[structure]]`. Each element contains the methylation states at the sites in a given tip and structure represented as 0, 0.5 or 1 (for unmethylated, partially-methylated and methylated). If methylation states are not represented as 0, 0.5, 1 they are categorized as 0 when value equal or under 0.2 0.5 when value between 0.2 and 0.8 and 1 when value over 0.8. For customized categorization thresholds use `categorize_siteMethSt`
sample_n	The number of samples (tips) to process.
categorized_data	Logical defaulted to FALSE. TRUE to skip redundant categorization when methylation states are represented as 0, 0.5, and 1.

Value

A numeric value representing the mean frequency of partially methylated sites in the non-islands.

Examples

```
# Example usage:
index_nonislands <- c(1, 3)
data <- list(
  list(c(0.5, 1, 0.5), c(0, 0.5, 1), c(1, 0, 0.5)), # tip 1
  list(c(0.5, 0.5, 0), c(1, 0.5, 0.5), c(0.5, 0.5, 1)) # tip 2
)
sample_n <- 2
get_nonislandMeanFreqP(index_nonislands, data, sample_n, categorized_data = TRUE)
```

---

get_nonislandSDFreqM *Calculate the Mean Standard Deviation of Methylated Sites in Non-Islands*

Description

This function computes the mean standard deviation of methylated sites (with methylation state 1) for a set of genomic structures identified as non-islands.

Usage

```
get_nonislandSDFreqM(
  index_nonislands,
  data,
  sample_n,
  categorized_data = FALSE
)
```

Arguments

index_nonislands	A vector containing the structural indices for non-islands.
data	A list containing methylation states at tree tips for each genomic structure (island / non-island) For a single tip: `data[[structure]]`. For multiple tips: `data[[tip]][[structure]]`. Each element contains the methylation states at the sites in a given tip and structure represented as 0, 0.5 or 1 (for unmethylated, partially-methylated and methylated). If methylation states are not represented as 0, 0.5, 1 they are categorized as 0 when value equal or under 0.2 0.5 when value between 0.2 and 0.8 and 1 when value over 0.8. For customized categorization thresholds use `categorize_siteMethSt`
sample_n	The number of tips (samples) to process.
categorized_data	Logical defaulted to FALSE. TRUE to skip redundant categorization when methylation states are represented as 0, 0.5, and 1.

Value

A numeric value representing the mean standard deviation of methylated sites in the non-islands.

Examples

```
# Example usage:
index_nonislands <- c(1, 3)
data <- list(
  list(c(1, 1, 1), c(0, 1, 0.5), c(1, 0, 1)), # tip 1
  list(c(1, 0.5, 0), c(1, 1, 0.5), c(1, 1, 1)) # tip 2
)
sample_n <- 2
get_nonislandSDFreqM(index_nonislands, data, sample_n, categorized_data = TRUE)
```

`get_nonislandSDFreqP` *Calculate the Mean Standard Deviation of Partially Methylated Sites in Non-Islands*

Description

This function computes the mean standard deviation of partially methylated sites (with methylation state 0.5) for a set of genomic structures identified as non-islands.

Usage

```
get_nonislandSDFreqP(
  index_nonislands,
  data,
  sample_n,
  categorized_data = FALSE
)
```

Arguments

`index_nonislands`	A vector containing the structural indices for non-islands.
`data`	A list containing methylation states at tree tips for each genomic structure (island / non-island) For a single tip: `data[[structure]]`. For multiple tips: `data[[tip]][[structure]]`. Each element contains the methylation states at the sites in a given tip and structure represented as 0, 0.5 or 1 (for unmethylated, partially-methylated and methylated). If methylation states are not represented as 0, 0.5, 1 they are categorized as 0 when value equal or under 0.2 0.5 when value between 0.2 and 0.8 and 1 when value over 0.8. For customized categorization thresholds use `categorize_siteMethSt`
`sample_n`	The number of samples (tips) to process.
`categorized_data`	Logical defaulted to FALSE. TRUE to skip redundant categorization when methylation states are represented as 0, 0.5, and 1.

Value

A numeric value representing the mean standard deviation of partially methylated sites in the non-islands.

Examples

```
# Example usage:
index_nonislands <- c(1, 3)
data <- list(
  list(c(0.5, 1, 0.5), c(0, 0.5, 1), c(1, 0, 0.5)), # tip 1
  list(c(0.5, 0.5, 0), c(1, 0.5, 0.5), c(0.5, 0.5, 1)) # tip 2
```

```
)
sample_n <- 2
get_nonislandSDFreqP(index_nonislands, data, sample_n, categorized_data = TRUE)
```

`get_parameterValues` *Get Default Parameter Values*

Description

This function retrieves parameter values for the DNA methylation simulation.

Usage

```
get_parameterValues(rootData = NULL)
```

Arguments

`rootData`	NULL to return default parameter values. For data parameter values, provide rootData as the output of simulate_initialData()$data.

Details

The function called without arguments returns default parameter values. When rootData (as $data output of simulate_initialData()) is given, it returns data parameter values.

Value

A data frame containing default parameter values.

Examples

```
# Get default parameter values
default_values <- get_parameterValues()

# Get parameter values of simulate_initialData() output
custom_params <- get_parameterValues()
infoStr <- data.frame(n = c(5, 10), globalState = c("M", "U"))
rootData <- simulate_initialData(infoStr = infoStr, params = custom_params)$data
rootData_paramValues <- get_parameterValues(rootData = rootData)
```

`get_siteFChange_cherry` *Compute Site Frequency of Methylation Changes per Cherry*

Description

This function calculates the total frequency of methylation differences (both full and half changes) for each genomic structure for each cherry in a phylogenetic tree. A cherry is a pair of leaf nodes (also called tips or terminal nodes) in a phylogenetic tree that share a direct common ancestor. In other words, if two leaves are connected to the same internal node and no other leaves are connected to that internal node, they form a cherry.

Usage

```
get_siteFChange_cherry(tree, data, categorized_data = FALSE)
```

Arguments

`tree`	A phylogenetic tree in Newick format or a phylo object from the ape package. The function ensures the tree has a valid structure and at least two tips.
`data`	A list containing methylation states at tree tips for each genomic structure (e.g., island/non-island). The data should be structured as `data[[tip]][[structure]]`, where each structure has the same number of sites across tips. The input data must be prefiltered to ensure CpG sites are represented consistently across different tips. Each element contains the methylation states at the sites in a given tip and structure represented as 0, 0.5 or 1 (for unmethylated, partially-methylated and methylated). If methylation states are not represented as 0, 0.5, 1 they are categorized as 0 when value equal or under 0.2 0.5 when value between 0.2 and 0.8 and 1 when value over 0.8. For customized categorization thresholds use `categorize_siteMethSt`
`categorized_data`	Logical defaulted to FALSE. TRUE to skip redundant categorization when methylation states are represented as 0, 0.5, and 1.

Details

The function first verifies that `tree` and `data` have valid structures and the minimum number of tips. It then extracts per-cherry methylation differences using `freqSites_cherryMethDiff`, handling potential errors. Finally, it aggregates the full and half methylation differences for each genomic structure at each cherry.

Value

A data frame with one row per cherry, containing the following columns:

tip_names A character string representing the names of the two tips in the cherry, concatenated with a hyphen.

tip_indices A character string representing the indices of the two tips in the cherry, concatenated with a hyphen.

dist A numeric value representing the sum of the branch distances between the cherry tips.

One column for each structure named with the structure number A numeric value representing the total frequency of methylation changes (both full and half) for the given structure.

Examples

```
# Example data setup

data <- list(
list(rep(1,10), rep(0,5), rep(1,8)),
list(rep(1,10), rep(0.5,5), rep(0,8)),
list(rep(1,10), rep(0.5,5), rep(0,8)),
list(c(rep(0,5), rep(0.5, 5)), c(0, 0, 1, 1, 1), c(0.5, 1, rep(0, 6))))

tree <- "((a:1.5,b:1.5):2,(c:2,d:2):1.5);"

get_siteFChange_cherry(tree, data, categorized_data = TRUE)
```

`MeanSiteFChange_cherry` *Compute the Mean Site Frequency of Methylation Changes per Cherry*

Description

This function calculates the weighted mean frequency of methylation changes at island and non-island genomic structures for each cherry in a phylogenetic tree. A cherry is a pair of leaf nodes (also called tips or terminal nodes) in a phylogenetic tree that share a direct common ancestor.

Usage

```
MeanSiteFChange_cherry(
  data,
  categorized_data = FALSE,
  tree,
  index_islands,
  index_nonislands
)
```

Arguments

`data`	A list containing methylation states at tree tips for each genomic structure (e.g., island/non-island). The data should be structured as `data[[tip]][[structure]]`, where each structure has the same number of sites across tips. The input data must be prefiltered to ensure CpG sites are represented consistently across different tips. Each element contains the methylation states at the sites in a given tip

and structure represented as 0, 0.5 or 1 (for unmethylated, partially-methylated and methylated). If methylation states are not represented as 0, 0.5, 1 they are categorized as 0 when value equal or under 0.2 0.5 when value between 0.2 and 0.8 and 1 when value over 0.8. For customized categorization thresholds use `categorize_siteMethSt`

`categorized_data`
: Logical defaulted to FALSE. TRUE to skip redundant categorization when methylation states are represented as 0, 0.5, and 1.

`tree`
: A phylogenetic tree in Newick format or a `phylo` object from the `ape` package. The function ensures the tree has a valid structure and at least two tips.

`index_islands`
: A numeric vector specifying the indices of genomic structures corresponding to islands.

`index_nonislands`
: A numeric vector specifying the indices of genomic structures corresponding to non-islands.

Details

The function first validates the tree and the input data structure. It then computes the per-cherry frequency of sites with different methylation states using `get_siteFChange_cherry`. The indices provided for islands and non-islands are checked for validity using `validate_structureIndices`. Finally, the function calculates the weighted mean site frequency of methylation changes for each cherry, separately for islands and non-islands.

Value

A data frame with one row per cherry, containing the following columns:

tip_names A character string representing the names of the two tips in the cherry, concatenated with a hyphen.

tip_indices A character string representing the indices of the two tips in the cherry, concatenated with a hyphen.

dist A numeric value representing the sum of the branch distances between the cherry tips.

nonisland_meanFChange A numeric value representing the weighted mean frequency of methylation changes in non-island structures.

island_meanFChange A numeric value representing the weighted mean frequency of methylation changes in island structures.

Examples

```
# Example data setup
data <- list(
list(rep(1,10), rep(0,5), rep(1,8)), # Tip a
list(rep(1,10), rep(0.5,5), rep(0,8)), # Tip b
list(rep(1,10), rep(0.5,5), rep(0,8)), # Tip c
list(c(rep(0,5), rep(0.5, 5)), c(0, 0, 1, 1, 1), c(0.5, 1, rep(0, 6)))) # Tip d

tree <- "((a:1.5,b:1.5):2,(c:2,d:2):1.5);"
```

```
index_islands <- c(1,3)
index_nonislands <- c(2)

MeanSiteFChange_cherry(data = data,
                       categorized_data = TRUE,
                       tree = tree,
                       index_islands = index_islands,
                       index_nonislands = index_nonislands)
```

`mean_CherryFreqsChange_i` *Mean Number of Significant Methylation Frequency Changes per Island in Cherries*

Description

Computes the mean number of significant changes per island in phylogenetic tree cherries, based on a specified p-value threshold.

Usage

```
mean_CherryFreqsChange_i(
  data,
  categorized_data = FALSE,
  index_islands,
  tree,
  pValue_threshold
)
```

Arguments

`data`	A list containing methylation states at tree tips for each genomic structure (e.g., island/non-island). The data should be structured as `data[[tip]][[structure]]`, where each structure has the same number of sites across tips. The input data must be prefiltered to ensure CpG sites are represented consistently across different tips. Each element contains the methylation states at the sites in a given tip and structure represented as 0, 0.5 or 1 (for unmethylated, partially-methylated and methylated). If methylation states are not represented as 0, 0.5, 1 they are categorized as 0 when value equal or under 0.2 0.5 when value between 0.2 and 0.8 and 1 when value over 0.8. For customized categorization thresholds use `categorize_siteMethSt`
`categorized_data`	Logical defaulted to FALSE. TRUE to skip redundant categorization when methylation states are represented as 0, 0.5, and 1.
`index_islands`	A numeric vector specifying the indices of islands to analyze.

`tree` A rooted binary tree in Newick format (character string) or as an ape phylo object.

`pValue_threshold` A numeric value between 0 and 1 that serves as the threshold for statistical significance in the chi-squared test.

Details

The function uses `simulate.p.value = TRUE` in `chisq.test` to compute the p-value via Monte Carlo simulation to improve reliability regardless of whether the expected frequencies meet the assumptions of the chi-squared test (i.e., expected counts of at least 5 in each category).

Value

A data frame containing the same information as `pValue_CherryFreqsChange_i`, but with additional columns indicating whether p-values are below the threshold (significant changes) and the mean frequency of significant changes per island.

Examples

```
tree <- "((d:1,e:1):2,a:2);"
data <- list(
  #Tip 1
  list(c(rep(1,9), rep(0,1)),
       c(rep(0,9), 1),
       c(rep(0,9), rep(0.5,1))),
  #Tip 2
  list(c(rep(0,9), rep(0.5,1)),
       c(rep(0.5,9), 1),
       c(rep(1,9), rep(0,1))),
  #Tip 3
  list(c(rep(1,9), rep(0.5,1)),
       c(rep(0.5,9), 1),
       c(rep(0,9), rep(0.5,1))))

index_islands <- c(1,3)
mean_CherryFreqsChange_i(data, categorized_data = TRUE,
                         index_islands, tree, pValue_threshold = 0.05)
```

`mean_TreeFreqsChange_i` *Mean Number of Significant Frequency Changes per Island Across all Tree Tips*

Description

This function analyzes the frequency changes of methylation states (unmethylated, partially methylated, methylated) across tree tips for a given set of islands. It performs a chi-squared test for each island to check for significant changes in frequencies across tips and returns the proportion of islands showing significant changes.

Usage

```
mean_TreeFreqsChange_i(
  tree,
  data,
  categorized_data = FALSE,
  index_islands,
  pValue_threshold,
  testing = FALSE
)
```

Arguments

`tree`	A phylogenetic tree object, typically of class `phylo`, containing tip labels.
`data`	A list containing methylation states at tree tips for each genomic structure (e.g., island/non-island). The data should be structured as `data[[tip]][[structure]]`, where each structure has the same number of sites across tips. The input data must be prefiltered to ensure CpG sites are represented consistently across different tips. Each element contains the methylation states at the sites in a given tip and structure represented as 0, 0.5 or 1 (for unmethylated, partially-methylated and methylated). If methylation states are not represented as 0, 0.5, 1 they are categorized as 0 when value equal or under 0.2 0.5 when value between 0.2 and 0.8 and 1 when value over 0.8. For customized categorization thresholds use `categorize_siteMethSt`
`categorized_data`	Logical defaulted to FALSE. TRUE to skip redundant categorization when methylation states are represented as 0, 0.5, and 1.
`index_islands`	A vector of indices of genomic structures corresponding to islands in data.
`pValue_threshold`	A numeric value between 0 and 1 that serves as the threshold for statistical significance in the chi-squared test.
`testing`	Logical defaulted to FALSE. TRUE for testing output.

Details

The function uses `simulate.p.value = TRUE` in `chisq.test` to compute the p-value via Monte Carlo simulation to improve reliability regardless of whether the expected frequencies meet the assumptions of the chi-squared test (i.e., expected counts of at least 5 in each category).

Throws errors if:

- The `tree` is not valid.

- `data` is not structured correctly across tips.
- `index_islands` is empty.
- `pValue_threshold` is not between 0 and 1.

Value

A numeric value representing the mean proportion of islands with significant frequency changes across tips.

Examples

```
# Example of usage:

tree <- "((d:1,e:1):2,a:2);"

data <- list(
  #Tip 1
  list(c(rep(1,9), rep(0,1)),
       c(rep(0,9), 1),
       c(rep(0,9), rep(0.5,1))),
  #Tip 2
  list(c(rep(1,9), rep(0.5,1)),
       c(rep(0.5,9), 1),
       c(rep(1,9), rep(0,1))),
  #Tip 3
  list(c(rep(1,9), rep(0.5,1)),
       c(rep(0.5,9), 1),
       c(rep(0,9), rep(0.5,1))))

index_islands <- c(1,3)


mean_TreeFreqsChange_i(tree,
                       data, categorized_data = TRUE,
                       index_islands,
                       pValue_threshold = 0.05)
```

---

pValue_CherryFreqsChange_i

Compute p-Values for Methylation Frequency Changes in Cherries

---

Description

Calculates p-values for changes in methylation frequency between pairs of cherry tips in a phylogenetic tree. A cherry is a pair of leaf nodes (also called tips or terminal nodes) in a phylogenetic tree that share a direct common ancestor.

Usage

```
pValue_CherryFreqsChange_i(
  data,
  categorized_data = FALSE,
  index_islands,
  tree,
  input_control = TRUE
)
```

Arguments

data	A list containing methylation states at tree tips for each genomic structure (e.g., island/non-island). The data should be structured as `data[[tip]][[structure]]`, where each structure has the same number of sites across tips. The input data must be prefiltered to ensure CpG sites are represented consistently across different tips. Each element contains the methylation states at the sites in a given tip and structure represented as 0, 0.5 or 1 (for unmethylated, partially-methylated and methylated). If methylation states are not represented as 0, 0.5, 1 they are categorized as 0 when value equal or under 0.2 0.5 when value between 0.2 and 0.8 and 1 when value over 0.8. For customized categorization thresholds use `categorize_siteMethSt`
categorized_data	Logical defaulted to FALSE. TRUE to skip redundant categorization when methylation states are represented as 0, 0.5, and 1.
index_islands	A numeric vector specifying the indices of islands to analyze.
tree	A rooted binary tree in Newick format (character string) or as an ape phylo object with minimum 2 tips.
input_control	Logical; if `TRUE`, validates input.

Details

The function uses `simulate.p.value = TRUE` in `chisq.test` to compute the p-value via Monte Carlo simulation to improve reliability regardless of whether the expected frequencies meet the assumptions of the chi-squared test (i.e., expected counts of at least 5 in each category).

Value

A data frame containing tip pair information (first tip name, second tip name, first tip index, second tip index, distance) and one column per island with the p-values from the chi-squared tests.

Examples

```
# Example with hypothetical tree and data structure

tree <- "((d:1,e:1):2,a:2);"
data <- list(
  #Tip 1
  list(c(rep(1,9), rep(0,1)),
```

```
      c(rep(0,9), 1),
      c(rep(0,9), rep(0.5,1))),
  #Tip 2
  list(c(rep(0,9), rep(0.5,1)),
      c(rep(0.5,9), 1),
      c(rep(1,9), rep(0,1))),
  #Tip 3
  list(c(rep(1,9), rep(0.5,1)),
      c(rep(0.5,9), 1),
      c(rep(0,9), rep(0.5,1))))

index_islands <- c(1,3)

pValue_CherryFreqsChange_i(data, categorized_data = TRUE, index_islands, tree)
```

`simulate_evolData` *Simulate Data Evolution along a Tree*

Description

This function simulates methylation data evolution along a tree. Either by simulating data at the root of the provided evolutionary tree (if infoStr is given) or by using pre-existing data at the root (if rootData is given) and letting it evolve along the tree.

Usage

```
simulate_evolData(
  infoStr = NULL,
  rootData = NULL,
  tree = NULL,
  params = NULL,
  dt = 0.01,
  CFTP = FALSE,
  CFTP_step_limit = 327680000,
  n_rep = 1,
  only_tip = TRUE
)
```

Arguments

`infoStr`	A data frame containing columns 'n' for the number of sites, and 'globalState' for the favoured global methylation state. If customized initial equilibrium frequencies are given, it also contains columns 'u_eqFreq', 'p_eqFreq', and 'm_eqFreq' with the equilibrium frequency values for unmethylated, partially methylated, and methylated.
`rootData`	The output of the simulate_initialData()$data function. It represents the initial data at the root of the evolutionary tree.

tree	A string in Newick format representing the evolutionary tree.
params	Optional data frame with specific parameter values. Structure as in get_parameterValues() output. If not provided, default values will be used.
dt	Length of time step for Gillespie's Tau-Leap Approximation (default is 0.01).
CFTP	Default FALSE. TRUE for calling cftp algorithm to set root state according to model equilibrium (Note that current implementation neglects IWE process).
CFTP_step_limit	when CFTP = TRUE, maximum number of steps before applying an approximation method (default 327680000 corresponding to size of CFTP info of approx 6.1 GB).
n_rep	Number of replicates to simulate (default is 1).
only_tip	Logical indicating whether to extract data only for tips (default is TRUE, FALSE to extract the information for all the tree branches).

Value

A list containing the parameters used (`$params`), the length of the time step used for the Gillespie's tau-leap approximation (`$dt`, default 0.01), the tree used (`$tree`). simulated data and the simulated data (`$data`). In `$data`, each list element corresponds to a simulation replicate.

- If only_tip is TRUE: In `$data`, each list element corresponds to a simulation replicate. Each replicate includes one list per tree tip, each containing:
  - The name of each tip in the simulated tree (e.g. replicate 2, tip 1: `$data[[2]][[1]]$name`).
  - A list with the sequence of methylation states for each tip-specific structure (e.g. replicate 1, tip 2, 3rd structure: `$data[[1]][[2]]$seq[[3]]`. The methylation states are encoded as 0 for unmethylated, 0.5 for partially methylated, and 1 for methylated.
- If only_tip is FALSE, `$data` contains 2 lists:
  - `$data$branchInTree`: a list in which each element contains the information of the relationship with other branches:
    - Index of the parent branch (e.g. branch 2): `$data$branchInTree[[2]]$parent_index`)
    - Index(es) of the offspring branch(es) (e.g. branch 1 (root)): `$data$branchInTree[[1]]$offspring_index`)
  - `$data$sim_data`: A list containing simulated data. Each list element corresponds to a simulation replicate. Each replicate includes one list per tree branch, each containing:
    - The name of each branch in the simulated tree. It's NULL for the tree root and inner nodes, and the name of the tips for the tree tips. (e.g. replicate 2, branch 1: `$data$sim_data[[2]][[1]]$name`)
    - Information of IWE events on that branch. It's NULL for the tree root and FALSE for the branches in which no IWE event was sampled, and a list containing `$islands` with the index(ces) of the island structure(s) that went through the IWE event and `$times` for the branch time point(s) in which the IWE was sampled. (e.g. replicate 1, branch 3: `$data$sim_data[[1]][[3]]$IWE`)
    - A list with the sequence of methylation states for each structure (the index of the list corresponds to the index of the structures). The methylation states are encoded as 0 for unmethylated, 0.5 for partially methylated, and 1 for methylated. (e.g. replicate 3, branch 2, structure 1: `$data$sim_data[[3]][[2]]$seq[[1]]`)

* A list with the methylation equilibrium frequencies for each structure (the index of the list corresponds to the index of the structures). Each structure has a vector with 3 values, the first one corresponding to the frequency of unmethylated, the second one to the frequency of partially methylated, and the third one to the frequency of methylated CpGs. (e.g. replicate 3, branch 2, structure 1: `$data$sim_data[[3]][[2]]$eqFreqs[[1]]`)

Examples

```
# Example data
infoStr <- data.frame(n = c(10, 100, 10), globalState = c("M", "U", "M"))

# Simulate data evolution along a tree with default parameters
simulate_evolData(infoStr = infoStr, tree = "(A:0.1,B:0.1);")

# Simulate data evolution along a tree with custom parameters
custom_params <- get_parameterValues()
custom_params$iota <- 0.5
simulate_evolData(infoStr = infoStr, tree = "(A:0.1,B:0.1);", params = custom_params)
```

`simulate_initialData` *Simulate Initial Data*

Description

This function simulates initial data based on the provided information and parameters.

Usage

```
simulate_initialData(
  infoStr,
  params = NULL,
  CFTP = FALSE,
  CFTP_step_limit = 327680000
)
```

Arguments

`infoStr`	A data frame containing columns 'n' for the number of sites, and 'globalState' for the favoured global methylation state. If customized equilibrium frequencies are given, it also contains columns 'u_eqFreq', 'p_eqFreq' and 'm_eqFreq' with the equilibrium frequency values for unmethylated, partially methylated and methylated.
`params`	Optional data frame with specific parameter values.
`CFTP`	Default FALSE. TRUE for calling cftp algorithm to set root state according to model equilibrium (Note that current implementation neglects IWE process). Structure as in get_parameterValues() output. If not provided, default values will be used.

`CFTP_step_limit`
: when CFTP = TRUE, maximum number of steps before applying an approximation method (default 327680000 corresponding to size of CFTP info of approx 6.1 GB).

Details

The function performs several checks on the input data and parameters to ensure they meet the required criteria and simulates DNA methylation data.

Value

A list containing the simulated data ($data) and parameters ($params).

Examples

```
# Example data
infoStr <- data.frame(n = c(10, 100, 10), globalState = c("M", "U", "M"))

# Simulate initial data  with default parameters
simulate_initialData(infoStr = infoStr)

# Simulate data evolution along a tree with custom parameters
custom_params <- get_parameterValues()
custom_params$iota <- 0.5
simulate_initialData(infoStr = infoStr, params = custom_params)
```

`singleStructureGenerator` *singleStructureGenerator*

Description

an R6 class representing a single genomic structure

Public fields

`testing_output` Public attribute: Testing output for initialize

Methods

Public methods:

- `singleStructureGenerator$init_neighbSt()`
- `singleStructureGenerator$initialize_ratetree()`
- `singleStructureGenerator$new()`
- `singleStructureGenerator$set_myCombiStructure()`
- `singleStructureGenerator$get_seq()`

- singleStructureGenerator$get_seqFirstPos()
- singleStructureGenerator$get_seq2ndPos()
- singleStructureGenerator$get_seqLastPos()
- singleStructureGenerator$get_seq2ndButLastPos()
- singleStructureGenerator$get_combiStructure_index()
- singleStructureGenerator$update_interStr_firstNeighbSt()
- singleStructureGenerator$update_interStr_lastNeighbSt()
- singleStructureGenerator$get_eqFreqs()
- singleStructureGenerator$SSE_evol()
- singleStructureGenerator$get_transMat()
- singleStructureGenerator$IWE_evol()
- singleStructureGenerator$get_alpha_pI()
- singleStructureGenerator$get_beta_pI()
- singleStructureGenerator$get_alpha_mI()
- singleStructureGenerator$get_beta_mI()
- singleStructureGenerator$get_alpha_pNI()
- singleStructureGenerator$get_beta_pNI()
- singleStructureGenerator$get_alpha_mNI()
- singleStructureGenerator$get_beta_mNI()
- singleStructureGenerator$get_alpha_Ri()
- singleStructureGenerator$get_iota()
- singleStructureGenerator$get_Ri_values()
- singleStructureGenerator$get_Q()
- singleStructureGenerator$get_siteR()
- singleStructureGenerator$get_neighbSt()
- singleStructureGenerator$update_ratetree_otherStr()
- singleStructureGenerator$get_Qi()
- singleStructureGenerator$get_seqSt_leftneighb()
- singleStructureGenerator$get_seqSt_rightneighb()
- singleStructureGenerator$cftp_all_equal()
- singleStructureGenerator$set_seqSt_update_neighbSt()
- singleStructureGenerator$reset_seq()
- singleStructureGenerator$clone()

Method `init_neighbSt()`**:** Public method: Initialization of $neighbSt

This fuction initiates each CpG position $neighbSt as encoded in $mapNeighbSt_matrix

It uses $update_neighbSt which updates for each sequence index, the neighbSt of left and right neighbors This means that it updates position 2, then 1 and 3, then 2 and 4.. Therefore, if the combiStructure instance has several singleStr instances within and the first has length 1, the $neighbSt of that position of the first singleStr instance is initialized when the method is called from the second singleStr instance

Positions at the edge of the entire simulated sequence use their only neighbor as both neighbors.

Usage:

```
singleStructureGenerator$init_neighbSt()
```

Returns: NULL

Method `initialize_ratetree()`**:** Public method: Initialization of $ratetree
This function initializes $ratetree

Usage:

```
singleStructureGenerator$initialize_ratetree()
```

Returns: NULL

Method `new()`**:** Create a new singleStructureGenerator object.
Note that this object is typically generated withing a combiStructureGenerator object.

Usage:

```
singleStructureGenerator$new(
  globalState,
  n,
  eqFreqs = NULL,
  combiStr = NULL,
  combiStr_index = NULL,
  params = NULL,
  testing = FALSE
)
```

Arguments:

`globalState` Character. Structure's favored global state: "M" for methylated (island structures) / "U" for unmethylated (non-island structures).

`n` Numerical Value. Number of CpG positions

`eqFreqs` Default NULL. When given: numerical vector with structure's methylation state equilibrium frequencies (for unmethylated, partially methylated and methylated)

`combiStr` Default NULL. When initiated from combiStructureGenerator: object of class combiStructureGenerator containing it

`combiStr_index` Default NULL. When initiated from combiStructureGenerator: index in Object of class combiStructureGenerator

`params` Default NULL. When given: data frame containing model parameters

`testing` Default FALSE. TRUE for writing in public field of new instance $testing_output

Returns: A new `singleStructureGenerator` object.

Method `set_myCombiStructure()`**:** Public method: Set my_combiStructure. Assigns given combi instance to private field my_combiStructure

Usage:

```
singleStructureGenerator$set_myCombiStructure(combi)
```

Arguments:

`combi` instance of combiStructureGenerator

Returns: NULL

Method `get_seq()`**:** Public method: Get object's methylation state sequence
Encoded with 1 for unmethylated, 2 for partially methylated and 3 for methylated

Usage:

```
singleStructureGenerator$get_seq()
```

Returns: vector with equilibrium frequencies of unmethylated, partially methylated and methylated

Method `get_seqFirstPos()`**:** Public method: Get first sequence position methylation state

Usage:

```
singleStructureGenerator$get_seqFirstPos()
```

Returns: numerical encoding of first position's methylation state

Method `get_seq2ndPos()`**:** Public method: Get second sequence position methylation state

Usage:

```
singleStructureGenerator$get_seq2ndPos()
```

Returns: numerical encoding of second position's methylation state. NULL if position does not exist

Method `get_seqLastPos()`**:** Public method: Get first sequence position methylation state

Usage:

```
singleStructureGenerator$get_seqLastPos()
```

Returns: numerical encoding of first position's methylation state

Method `get_seq2ndButLastPos()`**:** Public method: Get second but last sequence position methylation state

Usage:

```
singleStructureGenerator$get_seq2ndButLastPos()
```

Returns: numerical encoding of second but last position's methylation state. NULL if position does not exist

Method `get_combiStructure_index()`**:** Public method: Get index in object of class combiStructureGenerator

Usage:

```
singleStructureGenerator$get_combiStructure_index()
```

Returns: index in object of class combiStructureGenerator

Method `update_interStr_firstNeighbSt()`**:** Public method: Update neighbSt of next singleStructureGenerator object within combiStructureGenerator object

This function is used when the last $seq position of a singleStructureGenerator object changes methylation state to update the neighbSt position

Usage:

```
singleStructureGenerator$update_interStr_firstNeighbSt(
  leftNeighb_seqSt,
  rightNeighb_seqSt
)
```

Arguments:

`leftNeighb_seqSt` $seq state of left neighbor (left neighbor is in previous singleStructureGenerator object)

`rightNeighb_seqSt` $seq state of right neighbor

Returns: NULL

Method `update_interStr_lastNeighbSt()`: Public method: Update neighbSt of previous singleStructureGenerator object within combiStructureGenerator object

Usage:

```
singleStructureGenerator$update_interStr_lastNeighbSt(
  leftNeighb_seqSt,
  rightNeighb_seqSt
)
```

Arguments:

`leftNeighb_seqSt` $seq state of right neighbor (left neighbor is in next singleStructureGenerator object)

`rightNeighb_seqSt` $seq state of right neighbor

Returns: NULL

Method `get_eqFreqs()`: Public method: Get object's equilibrium Frequencies

Usage:

```
singleStructureGenerator$get_eqFreqs()
```

Returns: vector with equilibrium frequencies of unmethylated, partially methylated and methylated

Method `SSE_evol()`: Public method. Simulate how CpG dinucleotide methylation state changes due to the SSE process along a time step of length dt

Usage:

```
singleStructureGenerator$SSE_evol(dt, testing = FALSE)
```

Arguments:

`dt` time step length.

`testing` logical value for testing purposes. Default FALSE.

Returns: default NULL. If testing TRUE it returns a list with the debugNov3.outnumber of events sampled and a dataframe with the position(s) affected, new state and old methylation state.

Method `get_transMat()`: Public Method. Get a transition matrix

Usage:

```
singleStructureGenerator$get_transMat(
  old_eqFreqs,
  new_eqFreqs,
  info,
  testing = FALSE
)
```

Arguments:

`old_eqFreqs` numeric vector with 3 frequency values (for old u, p and m)

`new_eqFreqs` numeric vector with 3 frequency values (for new u, p and m)

`info` character string to indicate where the method is being called

`testing` logical value for testing purposes. Default FALSE.

Details: Given a tripple of old equilibrium frequencies and new equilibrium frequencies, generates the corresponding transition matrix.

Returns: transMat. The transition matrix. If testing = TRUE it returns a list. If there was a change in the equilibrium frequencies the list contains the following 7 elements, if not it contains the first 3 elements:

`transMat` transition matrix

`case` The applied case.

Method `IWE_evol()`**:** Public Method. Simulate IWE Events

Simulates how CpG Islands' methylation state frequencies change and simultaneous sites change methylation state along a branch of length t according to the SSE-IWE model.

Usage:

```
singleStructureGenerator$IWE_evol(testing = FALSE)
```

Arguments:

`testing` logical value for testing purposes. Default FALSE.

Details: The function checks if the methylation equilibrium frequencies (`eqFreqs`) and sequence observed frequencies (`obsFreqs`) change after the IWE event. If there is a change in either frequencies, the corresponding change flag `eqFreqsChange` in the `infoIWE` list will be set to `TRUE`.

Returns: If testing = TRUE it returns a list. If there was a change in the equilibrium frequencies the list contains the following 7 elements, if not it contains the first 3 elements:

`eqFreqsChange` logical indicating if there was a change in the equilibrium frequencies.

`old_eqFreqs` Original equilibrium frequencies before the IWE event.

`new_eqFreqs` New equilibrium frequencies after the IWE event.

`old_obsFreqs` Original observed frequencies before the IWE event.

`new_obsFreqs` New observed frequencies after the IWE event.

`IWE_case` Description of the IWE event case.

`Mk` Transition matrix used for the IWE event.

Method `get_alpha_pI()`**:** Public Method.

Usage:

```
singleStructureGenerator$get_alpha_pI()
```

Returns: Model parameter alpha_pI for sampling island equilibrium frequencies

Method `get_beta_pI()`**:** Public Method.

Usage:

```
singleStructureGenerator$get_beta_pI()
```

Returns: Model parameter for sampling island equilibrium frequencies

Method `get_alpha_mI()`: Public Method.

Usage:

```
singleStructureGenerator$get_alpha_mI()
```

Returns: Model parameter for sampling island equilibrium frequencies

Method `get_beta_mI()`: Public Method.

Usage:

```
singleStructureGenerator$get_beta_mI()
```

Returns: Model parameter for sampling island equilibrium frequencies

Method `get_alpha_pNI()`: Public Method.

Usage:

```
singleStructureGenerator$get_alpha_pNI()
```

Returns: Model parameter for sampling non-island equilibrium frequencies

Method `get_beta_pNI()`: Public Method.

Usage:

```
singleStructureGenerator$get_beta_pNI()
```

Returns: Model parameter for sampling non-island equilibrium frequencies

Method `get_alpha_mNI()`: Public Method.

Usage:

```
singleStructureGenerator$get_alpha_mNI()
```

Returns: Model parameter for sampling non-island equilibrium frequencies

Method `get_beta_mNI()`: Public Method.

Usage:

```
singleStructureGenerator$get_beta_mNI()
```

Returns: Model parameter for sampling non-island equilibrium frequencies

Method `get_alpha_Ri()`: Public Method.

Usage:

```
singleStructureGenerator$get_alpha_Ri()
```

Returns: Model parameter for gamma distribution shape to initialize the 3 $Ri_values

Method `get_iota()`: Public Method.

Usage:

```
singleStructureGenerator$get_iota()
```

Returns: Model parameter for gamma distribution expected value to initialize the 3 $Ri_values

Method `get_Ri_values()`: Public Method.

Usage:

```
singleStructureGenerator$get_Ri_values()
```

Returns: The 3 $Ri_values

Method `get_Q()`**:** Public Method.

Usage:

```
singleStructureGenerator$get_Q(
  siteR = NULL,
  neighbSt = NULL,
  oldSt = NULL,
  newSt = NULL
)
```

Arguments:

`siteR` default NULL. Numerical value encoding for the sites rate of independent SSE (1, 2 or 3)

`neighbSt` default NULL. Numerical value encoding for the sites neighbouring state (as in mapNeighbSt_matrix)

`oldSt` default NULL. Numerical value encoding for the sites old methylation state (1, 2 or 3)

`newSt` default NULL. Numerical value encoding for the sites new methylation state (1, 2 or 3)

Returns: With NULL arguments, the list of rate matrices. With non NULL arguments, the corresponding rate of change.

Method `get_siteR()`**:** Public Method.

Usage:

```
singleStructureGenerator$get_siteR(index = NULL)
```

Arguments:

`index` default NULL. Numerical value for the index of the CpG position within the singleStr instance

Returns: with NULL arguments, siteR vector. non NULL arguments, the corresponding siteR

Method `get_neighbSt()`**:** Public Method.

Usage:

```
singleStructureGenerator$get_neighbSt(index = NULL)
```

Arguments:

`index` default NULL. Numerical value for the index of the CpG position within the singleStr instance

Returns: with NULL arguments, neighbSt vector. non NULL arguments, the corresponding neighbSt

Method `update_ratetree_otherStr()`**:** Public Method. Update ratetree from another singleStructure instance

Usage:

```
singleStructureGenerator$update_ratetree_otherStr(position, rate)
```

Arguments:

`position` Numerical value for the index of the CpG position within the singleStr instance

`rate` Rate of change to asign to that position

Returns: NULL

Method `get_Qi()`**:** Public Method. Get list of matrices for SSE process

Usage:

```
singleStructureGenerator$get_Qi(siteR = NULL, oldSt = NULL, newSt = NULL)
```

Arguments:

`siteR` default NULL. Numerical value encoding for the sites rate of independent SSE (1, 2 or 3)

`oldSt` default NULL. Numerical value encoding for the sites old methylation state (1, 2 or 3)

`newSt` default NULL. Numerical value encoding for the sites new methylation state (1, 2 or 3)

Returns: With NULL arguments, the list of SSEi rate matrices. With non NULL arguments, the corresponding rate of change.

Method `get_seqSt_leftneighb()`**:** Public Method. Decode methylation state of left neighbor form owns neighbSt

Usage:

```
singleStructureGenerator$get_seqSt_leftneighb(index)
```

Arguments:

`index` Integer index value for the CpG position within the singleStr instance

Returns: decoded methylation state ($seq) of left neighbor (1, 2 or 3 for unmethylated, partially methylated or methylated)

Method `get_seqSt_rightneighb()`**:** Public Method. Decode methylation state of left neighbor form owns neighbSt

Usage:

```
singleStructureGenerator$get_seqSt_rightneighb(index)
```

Arguments:

`index` Integer index value for the CpG position within the singleStr instance

Returns: decoded methylation state ($seq) of right neighbor (1, 2 or 3 for unmethylated, partially methylated or methylated)

Method `cftp_all_equal()`**:** Public Method. Make a singleStructure with the same segment lengths and parameters as the focal one but where all states are m or u

Usage:

```
singleStructureGenerator$cftp_all_equal(state, testing = FALSE)
```

Arguments:

`state` Character value "U" or "M"

`testing` default FALSE. TRUE for testing output

Returns: right neighbSt

Method `set_seqSt_update_neighbSt()`**:** Public Method. Set the methylation state of a sequence position and update the neighbor's neighbSt. It does NOT update RATETREE

Usage:

```
singleStructureGenerator$set_seqSt_update_neighbSt(
  index,
  newSt,
  testing = FALSE
)
```

Arguments:

`index` Numerical value for the index of the CpG position within the singleStr instance

`newSt` Numerical value encoding for the sites new methylation state (1, 2 or 3)

`testing` default FALSE. TRUE for testing output

Returns: NULL when testing FALSE. Testing output when testing TRUE.

Method `reset_seq()`**:** Public Method. Resets the sequence states by resampling according to the instance's equilibrium frequencies.

Usage:

```
singleStructureGenerator$reset_seq()
```

Returns: NULL. The sequence is updated in place.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
singleStructureGenerator$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`treeMultiRegionSimulator` *treeMultiRegionSimulator*

Description

an R6 class representing the methylation state of GpGs in different genomic structures in the nodes of a tree.

The whole CpG sequence is an object of class combiStructureGenerator. Each genomic structure in it is contained in an object of class singleStructureGenerator.

Public fields

`testing_output` Public attribute: Testing output for initialize

`Branch` Public attribute: List containing objects of class combiStructureGenerator

`branchLength` Public attribute: Vector with the corresponding branch lengths of each $Branch element

Methods

Public methods:

- treeMultiRegionSimulator$treeEvol()
- treeMultiRegionSimulator$new()
- treeMultiRegionSimulator$clone()

Method `treeEvol()`**:** Simulate CpG dinucleotide methylation state evolution along a tree. The function splits a given tree and simulates evolution along its branches. It recursively simulates evolution in all of the subtrees in the given tree until the tree leafs

Usage:

```
treeMultiRegionSimulator$treeEvol(
  Tree,
  dt = 0.01,
  parent_index = 1,
  testing = FALSE
)
```

Arguments:

`Tree` String. Tree in Newick format. When called recursivelly it is given the corresponding subtree.

`dt` Length of SSE time steps.

`parent_index` Default 1. When called recursivelly it is given the corresponding parent branch index.

`testing` Default FALSE. TRUE for testing purposes.

Returns: NULL

Method `new()`**:** Create a new treeMultiRegionSimulator object. $Branch is a list for the tree branches, its first element represents the tree root.

Note that one of either infoStr or rootData needs to be given. Not both, not neither.

Usage:

```
treeMultiRegionSimulator$new(
  infoStr = NULL,
  rootData = NULL,
  tree = NULL,
  params = NULL,
  dt = 0.01,
  CFTP = FALSE,
  CFTP_step_limit = 327680000,
  testing = FALSE
)
```

Arguments:

`infoStr` A data frame containing columns 'n' for the number of sites, and 'globalState' for the favoured global methylation state. If initial equilibrium frequencies are given the dataframe must contain 3 additional columns: 'u_eqFreq', 'p_eqFreq' and 'm_eqFreq'

`rootData` combiStructureGenerator object. When given, the simulation uses its parameter values.

`tree` tree

`params` Default NULL. When given: data frame containing model parameters. Note that if rootData is not null, its parameter values are used.

`dt` length of the dt time steps for the SSE evolutionary process

`CFTP` Default FALSE. TRUE for calling cftp algorithm to set root state according to model equilibrium (Note that current implementation neglects IWE process).

`CFTP_step_limit` when CFTP = TRUE, maximum number of steps before applying an approximation method (default 327680000 corresponding to size of CFTP info of approx 6.1 GB).

`testing` Default FALSE. TRUE for testing output.

Returns: A new `treeMultiRegionSimulator` object.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
treeMultiRegionSimulator$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`validate_dataAcrossTips`
Validate Data Structure Across Tips

Description

This function ensures that `data` follows the required nested structure `data[[tip]][[structure]]`, where:

- `data` is a list of at least two `tip` elements.
- Each `tip` is a list of `structure` elements.
- Each `structure` contains a numeric vector of equal length across all tips.

Usage

```
validate_dataAcrossTips(data)
```

Arguments

`data`	A list structured as `data[[tip]][[structure]]`.

Details

Throws errors if:

- `data` is not a list.
- It has fewer than two tips.
- Any tip is not a list.
- The number of structures is inconsistent across tips.
- Any structure has zero-length data at any tip.
- Structures have different site lengths across tips.

`validate_data_cherryDist` *Validate Structure of Input Data for Cherry Distance Computation*

Description

This function checks whether the provided input data has the required structure. It ensures that the number of tips is sufficient and that the data structure is consistent across tips and structures.

Usage

```
validate_data_cherryDist(cherryDist, data)
```

Arguments

`cherryDist`	A data frame containing cherry pair distances, including tip indices (output from `get_cherryDist`)
`data`	A nested list representing structured data for each tip, following the format `data[[tip]][[structure]]`.

Details

The function performs several validation steps:

- Ensures that the number of tips in `data` is at least as large as the highest tip index in `cherryDist`.
- Checks that all tips contain at least one structure and that the number of structures is consistent across tips.
- Verifies that within each structure, all tips have the same number of sites and no zero-length structures.

If any of these conditions fail, the function throws an error with a descriptive message.

validate_structureIndices *Validate Structure Indices for Island and Non-Island Data*

Description

This function checks whether the provided indices for islands and non-islands are within the valid range of structures in the dataset. It also warns if any indices are present in both `index_islands` and `index_nonislands`.

Usage

```
validate_structureIndices(data, index_islands, index_nonislands)
```

Arguments

`data`	A nested list `data[[tip]][[structure]]`. Assumes that the number of structures is consistent across tips and that within each structure, all tips have the same number of sites. The number of structures is inferred from `length(data[[1]])`.
`index_islands`	An integer vector specifying indices that correspond to island structures.
`index_nonislands`	An integer vector specifying indices that correspond to non-island structures.

Details

The funct@exportion performs the following checks:

- Ensures that all indices in `index_islands` and `index_nonislands` are within the range of available structures.
- Throws an error if any index is out of bounds.
- Issues a warning if the same index appears in both `index_islands` and `index_nonislands`.

Value

No return value. The function stops execution if invalid indices are detected.

validate_tree *Validate and Parse a Phylogenetic Tree*

Description

This function checks whether the input is a valid phylogenetic tree, either as a character string in Newick format or as an object of class phylo from the ape package. If the input is a Newick string, it is parsed into a phylo object. The function also ensures that the tree contains at least two tips.

Usage

```
validate_tree(tree)
```

Arguments

tree	A phylogenetic tree in Newick format (as a character string) or an object of class phylo from the ape package. • If the input is a character string, it must follow the Newick format (e.g., "((tip_1:1,tip_2:1):5,tip_3:6);"). • If an object of class phylo is provided, it should be a valid phylogenetic tree.

Details

- The function first verifies that the input is either a valid phylo object or a character string.
- If the input is a Newick string, it attempts to parse it into a phylo object using ape::read.tree().
- If parsing fails, an informative error message is returned.
- The function also checks that the tree contains at least two tips, as a valid phylogenetic tree should have at least one split.

Value

A phylo object representing the validated and parsed tree.

Index