

# Package ‘MethEvolSIM’

January 20, 2025

**Title** Simulate DNA Methylation Dynamics on Different Genomic Structures along Genealogies

**Version** 0.1.7

**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](https://doi.org/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

**Depends** R (>= 4.0)

**VignetteBuilder** knitr

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2024-09-22 08:20:02 UTC

## Contents

combiStructureGenerator . . . . .	2
get_parameterValues . . . . .	7
simulate_evolData . . . . .	7
simulate_initialData . . . . .	9
singleStructureGenerator . . . . .	10
treeMultiRegionSimulator . . . . .	17

**Index** **20**

---

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.

## Methods

### Public methods:

- `combiStructureGenerator$new()`
- `combiStructureGenerator$get_singleStr()`
- `combiStructureGenerator$get_singleStr_number()`
- `combiStructureGenerator$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$get_mu()`
- `combiStructureGenerator$set_singleStr()`
- `combiStructureGenerator$copy()`
- `combiStructureGenerator$branch_evol()`
- `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 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_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 inner 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** `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_evolution()`: 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_evolution(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** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

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

*Arguments:*

deep Whether to make a deep clone.

---

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)
```

---

simulate\_evolveData     *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,
  params = NULL,
  dt = 0.01,
  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).
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)
```

**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. Structure as in <code>get_parameterValues()</code> output. If not provided, default values will be used.

**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

**Methods****Public methods:**

- `singleStructureGenerator$init_neighbSt()`
- `singleStructureGenerator$initialize_ratetree()`
- `singleStructureGenerator$new()`
- `singleStructureGenerator$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\$clone()

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

This function initiates each CpG position \$neighbSt as encoded in \$mapNeighbSt\_matrix 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 testing output

*Returns:* A new *singleStructureGenerator* object.

**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(
  leftNeighbSt,
  rightNeighbSt
)
```

*Arguments:*

leftNeighbSt \$seq state of left neighbor (left neighbor is in previous singleStructureGenerator object)

rightNeighbSt \$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(
  leftNeighbSt,
  rightNeighbSt
)
```

*Arguments:*

leftNeighbSt \$seq state of right neighbor (left neighbor is in next singleStructureGenerator object)

rightNeighbSt \$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** `SSEevol()`: Public method. Simulate how CpG dinucleotide methylation state changes due to the SSE process along a time step of length dt

*Usage:*

```
singleStructureGenerator$SSEevol(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 number 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** `IWEevol()`: 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$IWEevol(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  $R_i$  values

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

*Usage:*

```
singleStructureGenerator$get_iota()
```

*Returns:* Model parameter for gamma distribution expected value to initialize the 3  $R_i$  values

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

*Usage:*

```
singleStructureGenerator$get_Ri_values()
```

*Returns:* The 3  $R_i$  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 map-NeighbSt\_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 assign to that position

*Returns:* NULL

**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**

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 leaves

*Usage:*

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

*Arguments:*

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

dt Length of SSE time steps.

parent\_index Default 1. When called recursively 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,
  params = NULL,
  dt = 0.01,
  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 rootData is given, its parameter values are used.

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

`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.

# Index

`combiStructureGenerator`, [2](#)  
`get_parameterValues`, [7](#)  
`simulate_evolData`, [7](#)  
`simulate_initialData`, [9](#)  
`singleStructureGenerator`, [10](#)  
`treeMultiRegionSimulator`, [17](#)