Package 'sismonr'

February 11, 2020

Type Package

Title Simulation of in Silico Multi-Omic Networks

Version 2.1.0

Maintainer Olivia Angelin-Bonnet <olivia.angelinbonnet@gmail.com>

Description A tool for the simulation of gene expression profiles for in silico regulatory networks. The package generates gene regulatory networks, which include protein-coding and non-coding genes linked via different types of regulation: regulation of transcription, translation, RNA or protein decay, and post-translational modifications. The effect of genetic mutations on the system behaviour is accounted for via the simulation of genetically different in silico individuals. The ploidy of the system is not restricted to the usual haploid or diploid situations, but is defined by the user. A choice of stochastic simulation algorithms allow us to simulate the expression profiles (RNA and if applicable protein abundance) of the genes in the in silico system for the different in silico individuals. A tutorial explaining how to use the package is available at <https://oliviaab.github.io/sismonr/>. Manuscript in preparation; see also Angelin-Bonnet O., Biggs P.J. and Vignes M. (2018) <doi:10.1109/BIBM.2018.8621131>. Note that sismonr relies on Julia code called internally by the functions. No knowledge of Julia is required in order to use sismonr, but Julia must be installed on the computer (instructions can be found in the tutorial, the GitHub page or the vignette of the package).

License GPL (>= 2)

Encoding UTF-8

LazyData true

URL https://oliviaab.github.io/sismonr/

BugReports https://github.com/oliviaAB/sismonr/issues

Imports XRJulia (>= 0.9.0), parallel, truncnorm, tictoc, stats, utils, dplyr, magrittr, stringr, XR, jsonlite, methods, tidyr, ggpubr, ggplot2, rlang, grDevices, igraph, graphics, scales

SystemRequirements Julia, v 1.0 or later

RoxygenNote 7.0.2

Suggests knitr, rmarkdown, tcltk, testthat (>= 2.1.0)

VignetteBuilder knitr

NeedsCompilation no

Author Olivia Angelin-Bonnet [aut, cre]
(<https://orcid.org/0000-0002-7708-2919>),
Patrick Biggs [aut] (<https://orcid.org/0000-0002-0285-4101>),
Matthieu Vignes [aut] (<https://orcid.org/0000-0001-8929-2975>),
John M. Chambers [ctb]

Repository CRAN

Date/Publication 2020-02-11 06:50:02 UTC

R topics documented:

addComplex — *Adds a regulatory complex in the in silico system.*

Description

Adds a regulatory complex in the in silico system with specified parameters (if provided), or with parameters sampled according to the system parameters.

Usage

```
addComplex(
  insilicosystem,
  compo,
  formationrate = NULL,
  dissociationrate = NULL
)
```

Arguments

`insilicosystem`	The in silico system (see `createInSilicoSystem`).
`compo`	An character vector, each element corresponding to the ID of the genes or regulatory complexes composing the complex. All genes/complexes composing the complex must have the same biological function (i.e. same `TargetReaction` parameter).
`formationrate`	The formation rate of the complex. If none provided, randomly chosen according to the parameter `complexesformationrate_samplingfct` provided in `sysargs` (see `insilicosystemargs`).
`dissociationrate`	The dissociation rate of the complex. If none provided, randomly chosen according to the parameter `complexesdissociationrate_samplingfct` provided in `sysargs` (see `insilicosystemargs`).

Value

Returns the modified in silico system.

Examples

```
mysystem = createInSilicoSystem(G = 10, PC.p = 1, PC.TC.p = 1)
mysystem$complexes ## list of complexes existing in the system
mysystem2 = addComplex(mysystem, c(1, 2, 3))
mysystem2$complexes
```

`addEdge` *Adds an edge in the in silico system's regulatory network.*

Description

Adds an edge in the in silico system's regulatory network between specified genes.

Usage

```
addEdge(insilicosystem, regID, tarID, regsign = NULL, kinetics = list())
```

Arguments

`insilicosystem`	The in silico system (see `createInSilicoSystem`).
`regID`	Character. The ID of the regulator gene or complex.
`tarID`	Character. The ID of the target gene.
`regsign`	The sign of the regulation: either "1" (positive regulation) or "-1" (negative regulation). If none provided, will be randomly chosen according to the parameter `TC.pos.p`, `TL.pos.p` or `PTM.pos.p` (depending on the type of regulation - regulation of RNA or protein decay can only be negative) provided in sysargs (see `insilicosystemargs`).
`kinetics`	Optional: named list of kinetics parameters of the reaction. If none provided, will be randomly chosen according to the parameters `[name of the param]_samplingfct` provided in sysargs (see `insilicosystemargs`). The parameters to provide depend on the type of regulation (i.e. parameter `TargetReaction` of the regulator): • TargetReaction = "TC": the parameters to specify are "TCbindingrate", "TCunbindingrate" and "TCfoldchange"; • TargetReaction = "TL": the parameters to specify are "TLbindingrate", "TLunbindingrate" and "TLfoldchange"; • TargetReaction = "RD": the parameter to specify is "RDregrate"; • TargetReaction = "PD": the parameter to specify is "PDregrate"; • TargetReaction = "PTM": the parameter to specify is "PTMregrate".

Details

It must be noted that the type of regulation depends on the biological function of the regulator gene/complex (parameter `TargetReaction`).

Value

The modified in silico system.

Examples

```
## creates a system with no regulation
mysystem = createInSilicoSystem(G = 10, PC.p = 1, PC.TC.p = 1, empty = TRUE)
mysystem$edg
mysystem2 = addEdge(mysystem, 1, 2, regsign = "1",
 kinetics = c("TCbindingrate"= 0.01, "TCunbindingrate" = 0.1, "TCfoldchange" = 10))
## check all existing interactions in the system (no kinetic parameters)
mysystem2$edg
## check the interactions targeting transcription, with kinetic parameters
mysystem2$mosystem$TCRN_edg

## creates a system with no regulation
mysystem = createInSilicoSystem(G = 5, PC.p = 1, PC.PD.p = 1, empty = TRUE)
mysystem$edg
mysystem2 = addEdge(mysystem, 1, 2)
## check all existing interactions in the system (no kinetic parameters)
mysystem2$edg
## check the interactions targeting protein decay, with kinetic parameters
mysystem2$mosystem$PDRN_edg
```

---

addGene *Adds a gene in the in silico system.*

Description

Adds a gene in the in silico system with specified parameters if provided, or with parameters sampled according to the system parameters.

Usage

```
addGene(
  insilicosystem,
  coding = NULL,
  TargetReaction = NULL,
  TCrate = NULL,
  TLrate = NULL,
  RDrate = NULL,
  PDrate = NULL
)
```

Arguments

`insilicosystem`	The in silico system (see `createInSilicoSystem`).
`coding`	String. The coding status of the gene (either "PC" for protein-coding or "NC" for noncoding). If none provided, randomly chosen according to the parameter `PC.p` provided in sysargs (see `insilicosystemargs`).

`TargetReaction`	String. The biological function of the gene, i.e. the gene expression step targeted by the active product of the gene. If none provided, randomly chosen according to the parameters `PC.TC.p`, etc or `NC.TC.p`, etc (depending on the coding status of the gene) provided in sysargs (see `insilicosystemargs`).
`TCrate`	Numeric. The transcription rate of the gene. If none provided, randomly chosen according to the parameter `basal_transcription_rate_samplingfct` provided in sysargs (see `insilicosystemargs`).
`TLrate`	Numeric. The translation rate of the gene. If none provided, randomly chosen according to the parameter `basal_translation_rate_samplingfct` provided in sysargs (see `insilicosystemargs`).
`RDrate`	Numeric. The RNA decay rate of the gene. If none provided, randomly chosen according to the parameter `basal_RNAlifetime_samplingfct` provided in sysargs (see `insilicosystemargs`).
`PDrate`	Numeric. The protein decay rate of the gene. If none provided, randomly chosen according to the parameter `basal_protlifetime_samplingfct` provided in sysargs (see `insilicosystemargs`).

Value

The modified in silico system.

Examples

```
mysystem = createInSilicoSystem(G = 5)
mysystem$genes
mysystem2 = addGene(mysystem, "PC", "TC", TCrate = 0.0001, TLrate = 0.001)
mysystem2$genes

mysystem3 = addGene(mysystem2)
mysystem3$genes
```

`callJuliaStochasticSimulation`
Calls the Julia simulation function.

Description

Calls the Julia function for simulating a stochastic system. Should not be used by itself (this function is called by the wrapper functions simulateInSilicoSystem and simulateParallelInSilicoSystem).

Usage

```
callJuliaStochasticSimulation(
  stochmodel,
  QTLeffects,
  InitAbundance,
  genes,
  simtime,
  modelname = "MySimulation",
  ntrials,
  nepochs,
  simalgorithm,
  ev = getJuliaEvaluator()
)
```

Arguments

`stochmodel`	A Julia proxy object to retrieve the stochastic system in the Julia evaluator.
`QTLeffects`	The list of QTL effects coefficients of the in silico individual to be simulated (see `createIndividual`).
`InitAbundance`	The list of initial abundances of the molecules for the in silico individual to be simulated (see `createIndividual`).
`genes`	The data-frame of genes in the system.
`simtime`	Numeric. The amount of time to simulate the model (in seconds).
`modelname`	String. The name of the model. Default value "MySimulation".
`ntrials`	Integer. The number of times the simulation must be replicated.
`nepochs`	Integer. The number of times to record the state of the system during the simulation.
`simalgorithm`	String. The name of the simulation algorithm to use in the Julia function `simulate` from the module `BioSimulator`. Possible values are: "Direct", "EnhancedDirect", "SortingDirect", "FirstReaction", "NextReaction", "TauLeapingDG2001", "TauLeapingDGLP2003", "StepAnticipation", "HybridSAL". See `https://alanderos91.github.io/BioSimulator.jl/dev/man/algorithms/` for details about the algorithms.
`ev`	A Julia evaluator. If none provided select the current evaluator or create one if no evaluator exists.

Value

The result of the simulation (a data-frame).

createEmptyMultiOmicNetwork

Creates an empty in silico system.

Description

Creates an empty in silico system (i.e. no regulatory interactions) given a data-frame of genes (cf `createMultiOmicNetwork`).

Usage

```
createEmptyMultiOmicNetwork(genes)
```

Arguments

`genes`	A data-frame of the genes existing in the system (see `createGenes`).

Value

An in silico system, that is a list of:

- `genes`: the modified data-frame of genes;
- `edg`: A data-frame of edges in the regulatory network of the system (1 row = 1 edge, here empty dataframe). It contains the following parameters:
  - `from`: gene ID of the edge origin (character).
  - `to`: gene ID of edge destination (character).
  - `TargetReaction`: Type of regulation (ID of the controlled reaction: "TC", "TL", "RD", "PD" or "PTM").
  - `RegSign`: Sign of the reaction ("1" for positive regulation, "-1" for negative regulation).
  - `RegBy`: Type of the regulator: "PC" for protein-coding regulation, "NC" for noncoding regulator, "C" for regulatory complex.
- `mosystem`: A list of the different regulatory networks (each corresponding to a different type of regulation) in the system and associated information. Elements of the list are `TCRN_edg`, `TLRN_edg`, `RDRN_edg`, `PDRN_edg` and `PTMRN_edg`: data-frames of edges for the different regulatory networks, which in addition to the usual fields in the edg data frame, contain columns for kinetic parameters of the regulation. All empty.
- `complexes`: a list of regulatory complexes composition. The names of the elements are the IDs of the complexes, and the values are vectors of gene IDs constituting each regulatory complex. Empty list.
- `complexeskinetics`: a list of regulatory complexes kinetic parameters. Empty list.
- `complexesTargetReaction`: a list defining which expression step is targeted by each regulatory complex.

Examples

```
mysysargs = insilicosystemargs(G = 5)
mygenes = createGenes(mysysargs)
mynetwork = createEmptyMultiOmicNetwork(mygenes)
```

`createGenes` *Creates genes for the in silico system.*

Description

Generates the genes in the system and their attributes, according to the user parameters.

Usage

```
createGenes(sysargs)
```

Arguments

`sysargs`	An object of class `insilicosystemargs` (i.e. a list with parameters for in silico system generation).

Value

A data frame of in silico genes. Attributes:

- `id`: Integer, ID of the genes;
- `coding`: coding status of the genes (either "PC" for protein-coding or "NC" for noncoding). Sampled according to the parameter `PC.p` in `sysargs`;
- `TargetReaction`: the biological function of the genes ("TC": transcription regulator, "TL": translation regulator, "RD": RNA decay regulator, "PD": protein decay regulator, "PTM": post-translational modification regulator, "MR": metabolic enzyme). Sampled according to the parameters `PC.TC.p`, etc for protein-coding genes or `NC.TC.p`, etc for noncoding genes, in `sysargs`;
- `PTMform`: Does the gene have a PTM form? "0" or "1" (here all "0", PTM form will be assigned later);
- `Active form`: what is the active form of the gene? "R" for noncoding genes, "P" for protein-coding genes, "Pm" for protein-coding genes with a PTM form;
- `TCrate`: transcription rate of the genes. Sampled according to the parameter `basal_transcription_rate_samplingfct` in `sysargs`;
- `TLrate`: translation rate of the genes. Sampled according to the parameter `basal_translation_rate_samplingfct` in `sysargs` (0 for noncoding genes);
- `RDrate`: RNA decay rate of the genes. Sampled according to the parameter `basal_RNAlifetime_rate_samplingfct` in `sysargs`;
- `PDrate`: Protein decay rate of the genes. Sampled according to the parameter `basal_protlifetime_rate_samplingfct` in `sysargs` (0 for noncoding genes).

Examples

```
createGenes(insilicosystemargs(G = 5))
```

`createIndividual` *Creates an in silico individual.*

Description

Creates a in silico individual to be simulated (object of class `insilicoindividual`).

Usage

```
createIndividual(
  insilicosystem,
  variantsList,
  variantsFreq,
  indargs,
  InitVar = NULL,
  initialNoise = TRUE
)
```

Arguments

`insilicosystem`	An `insilicosystem` object. The in silico system based on which individuals are created. See `createInSilicoSystem`.
`variantsList`	A named list giving the variants segregating in the population for each gene (e.g. created by `createVariants`). Each element corresponds to one gene in the system (name of the element = gene ID). Each element is a matrix, in which each column represents a variant of the gene segregating in the population. The rows represent the QTL effect coefficients of each variant (i.e. the impact of each mutation the variant carries).
`variantsFreq`	A named list giving for each gene the allelic frequency of each segregating variant. Each element corresponds to one gene in the system (name of the element = gene ID). Each element is a vector, of length equal to the number of variants of the gene segregating in the population, giving the allele frequency of each of the variants.
`indargs`	An object of class `insilicoindividualargs` (i.e. a list with parameters for in silico individuals generation).
`InitVar`	A list of the multiplicative coefficients to be applied to the initial abundance of the different molecules: elements "R" and "P" of the list giving the coefficients for the RNA and protein form of the genes, respectively (coefficient for gene `i` at the `i`-th position in the vectors). If NULL, all coefficients set to 1.
`initialNoise`	Logical. Is stochastic noise applied to the initial abundance of the different molecules? Default value is `TRUE` (see Details).

Details

initialNoise: by default, the initial abundance of a molecule is equal to its steady state abundance in the absence of any regulation (e.g. for the RNA abundance of a gene, it is transcription rate / decay rate). If `initialNoise = TRUE`, instead the initial abundance of the molecule will be sampled from a truncated Normal distribution of mean `SSabund` and SD `sqrt(SSabund)`, where `SSabund` is its steady state abundance in the absence of any regulation, as specified above. The Normal distribution is truncated to only return positive values.

Value

An object of class `insilicoindividual`, that is a list composed of:

- `QTLeffects`: a list of the variants carried by the individual. 1st level of the list: the different "GCN" (Gene Copy Number), that is the different alleles of the genes (as defined by the ploidy of the individual: a diploid will have GCN1 and GCN2); 2nd level: the different QTL effect coefficients. The elements in this 2nd-level list are vectors of QTL effect coefficients for the different genes (coefficient for gene `i` at the `i`-th position in the vector).
- `haplotype`: data-frame (rows = genes, columns = Gene copy number) giving the ID of the gene variant carried by the individual for each gene copy number (allele).
- `InitAbundance`: A list of the initial abundance of the different molecules. 1st level of the list: the different "GCN" (Gene Copy Number), that is the different alleles of the genes (as defined by the ploidy of the individual: a diploid will have GCN1 and GCN2); 2nd level of the list: initial abundance of the protein ("P") and RNA ("R") form of the genes (coefficient for gene `i` at the `i`-th position in the vectors).

Examples

```
mysystem = createInSilicoSystem(G = 3, ploidy = 4)
indargs = insilicoindividualargs()
## We will create only 1 variant of gene 1, 3 variants of gene 2 and
## 2 variants of gene 3
nbvariants = c(1, 3, 2)

qtlnames = c("qtlTCrate", "qtlRDrate",
             "qtlTCregbind", "qtlRDregrate",
             "qtlactivity", "qtlTLrate",
             "qtlPDrate", "qtlTLregbind",
             "qtlPDregrate", "qtlPTMregrate")

genvariants = lapply(nbvariants, function(x){
  matrix(1, nrow = length(qtlnames), ncol = x,
         dimnames = list(qtlnames, 1:x))
})
names(genvariants) = 1:length(nbvariants)

## the 2nd variant of gene 2 has a mutation reducing its transcription rate by 3
genvariants$`2`["qtlTCrate", 2] = 0.33
## and the 3rd variant has an increased translation rate
genvariants$`2`["qtlTLrate", 2] = 1.5
```

```
## The 2nd variant of gene 3 has a mutation decreasing the activity of
## its active product
genvariants$`3`["qtlactivity", 2] = 0.7

## Allelic frequency of each variant
genvariants.freq = list('1' = c(1),
                        '2' = c(0.6, 0.3, 0.1),
                        '3' = c(0.9, 0.1))

## The third gene is not expressed at the beginning of the simulation
## (its initial abundance is 0)
InitVar = list("R" = c(1, 1, 0), "P" = c(1, 1, 0))

myind = createIndividual(mysystem, genvariants, genvariants.freq, indargs, InitVar = InitVar)
```

`createInSilicoPopulation`

Creates a population of in silico individuals.

Description

Creates a population of in silico individuals to be simulated.

Usage

```
createInSilicoPopulation(
  nInd,
  insilicosystem,
  genvariants = NULL,
  genvariants.freq = NULL,
  InitVar = NULL,
  initialNoise = TRUE,
  ...
)
```

Arguments

`nInd`	Integer. The number of in silico individuals to create.
`insilicosystem`	An `insilicosystem` object. The in silico system based on which individuals are created. See `createInSilicoSystem`.
`genvariants`	A named list giving the variants segregating in the population for each gene. Each element corresponds to one gene in the system (name of the element = gene ID). Each element is a matrix, in which each column represents a variant of the gene segregating in the population. The rows represent the QTL effect coefficients of each variant (i.e. the impact of each mutation the variant carries). If none provided, will be automatically generated by the function `createVariants`.

`genvariants.freq`	A named list giving for each gene the allelic frequency of each segregating variant. Each element corresponds to one gene in the system (name of the element = gene ID). Each element is a vector, of length equal to the number of variants of the gene segregating in the population, giving the allele frequency of each of the variants. If none provided, it is assumed that all variants of a given gene have the same allelic frequency.
`InitVar`	A list of the multiplicative coefficients to be applied to the initial abundance of the different molecules: elements "R" and "P" of the list giving the coefficients for the RNA and protein form of the genes, respectively (coefficient for gene `i` at the `i`-th position in the vectors). If NULL, all coefficients set to 1.
`initialNoise`	Logical. Is stochastic noise applied to the initial abundance of the different molecules? Default value is `TRUE` (see Details).
`...`	Other arguments to be passed to the function `insilicoindividualargs` (i.e. parameters for the generation of the in silico individuals).

Details

initialNoise: by default, the initial abundance of a molecule is equal to its steady state abundance in the absence of any regulation (e.g. for the RNA abundance of a gene, it is transcription rate / decay rate). If `initialNoise` = `TRUE`, instead the initial abundance of the molecule will be sampled from a truncated Normal distribution of mean `SSabund` and SD `sqrt(SSabund)`, where `SSabund` is its steady state abundance in the absence of any regulation, as specified above. The Normal distribution is truncated to only return positive values.

Value

An object of class `insilicopopulation`, that is a list composed of:

- `GenesVariants` A list of variants segregating in the population for each genes (see `createVariants`).
- `individualsList` A list of in silico individuals (i.e. objects of class `insilicoindividual`, see `createIndividual`).
- `indargs` An object of class `insilicoindividualargs`; the parameters used to create the in silico individuals.

Examples

```
## Creating a first population with 3 diploid individuals,
## with 2 variants of each gene segregating in the population
mysystem = createInSilicoSystem(G = 6, ploidy = 2)
mypop1 = createInSilicoPopulation(nInd = 3, mysystem, ngenevariants = 2)

## Creating a population with 10 tetraploid individuals
mysystem = createInSilicoSystem(G = 6, ploidy = 4)
mypop2 = createInSilicoPopulation(nInd = 10, mysystem)

## Creating a population with a given list of gene variants
mysystem = createInSilicoSystem(G = 3, PC.p = 1, ploidy = 2)
```

```
## We will create only 1 variant of gene 1, 3 variants of gene 2 and
## 2 variants of gene 3
nbvariants = c(1, 3, 2)

qtlnames = c("qtlTCrate", "qtlRDrate",
             "qtlTCregbind", "qtlRDregrate",
             "qtlactivity", "qtlTLrate",
             "qtlPDrate", "qtlTLregbind",
             "qtlPDregrate", "qtlPTMregrate")

genvariants = lapply(nbvariants, function(x){
  matrix(1, nrow = length(qtlnames), ncol = x,
         dimnames = list(qtlnames, 1:x))
})
names(genvariants) = mysystem$genes$id

## the 2nd variant of gene 2 has a mutation reducing its transcription rate by 3
genvariants$`2`["qtlTCrate", 2] = 0.33
## and the 3rd variant has an increased translation rate
genvariants$`2`["qtlTLrate", 2] = 1.5

## The 2nd variant of gene 3 has a mutation decreasing the activity of
## its active product
genvariants$`3`["qtlactivity", 2] = 0.7

## Allelic frequency of each variant
genvariants.freq = list('1' = c(1),
                        '2' = c(0.6, 0.3, 0.1),
                        '3' = c(0.9, 0.1))

## The third gene is not expressed at the beginning of the simulation
## (its initial abundance is 0)
InitVar = list("R" = c(1, 1, 0), "P" = c(1, 1, 0))

mypop = createInSilicoPopulation(10, mysystem,
                                 genvariants = genvariants,
                                 genvariants.freq = genvariants.freq,
                                 InitVar = InitVar)
```

`createInSilicoSystem` *Creates an in silico system.*

Description

Creates an in silico system, i.e. the genes and the regulatory network defining the system.

Usage

```
createInSilicoSystem(empty = F, ev = getJuliaEvaluator(), ...)
```

Arguments

`empty`	Logical. Does the regulatory network is empty (= no regulation)? Default value is `FALSE`.
`ev`	A Julia evaluator (for the XRJulia package). If none provided select the current evaluator or create one if no evaluator exists.
`...`	Other arguments to be passed to the function `insilicosystemargs` (i.e. parameters for the generation of the in silico system).

Value

An object of class `insilicosystem`, that is a list composed of:

- `genes`: a data-frame of genes (see `createGenes`);
- `edg`: a data-frame of edges in the regulatory network (see `createMultiOmicNetwork`);
- `mosystem`: a list defining the regulatory network (see `createMultiOmicNetwork`);
- `sysargs`: An object of class `insilicosystemargs`; the parameters used to create the system.

Examples

```
## Creates an in silico system composed of 20 genes
mysystem1 = createInSilicoSystem(G = 20)
mysystem1$edg ## see all regulations in the system
mysystem1$mosystem$TCRN_edg ## see only regulations targeting transcription

## Creates an in silico systerm composed of 10 genes, all protein-coding
mysystem2 = createInSilicoSystem(G = 10, PC.p = 1)
mysystem2$genes

## Creates an in silico systerm composed of 5 genes,
## all noncoding and all regulators of transcription
mysystem3 = createInSilicoSystem(G = 5, PC.p = 0, NC.TC.p = 1)
mysystem3$edg
mysystem3$mosystem$TCRN_edg
```

`createMultiOmicNetwork`

Creates an in silico system.

Description

Creates an in silico system from a data-frame of genes in the system.

Usage

```
createMultiOmicNetwork(genes, sysargs, ev = getJuliaEvaluator())
```

Arguments

`genes`	A data-frame of the genes existing in the system (see `createGenes`).
`sysargs`	An object of class `insilicosystemargs` (i.e. a list with parameters for in silico system generation).
`ev`	A Julia evaluator. If none provided select the current evaluator or create one if no evaluator exists.

Value

An in silico system, that is a list of:

- `genes`: the modified data-frame of genes;
- `edg`: A data-frame of edges in the regulatory network of the system (1 row = 1 edge). It contains the following parameters:
  - `from`: gene ID of the edge origin (character).
  - `to`: gene ID of edge destination (character).
  - `TargetReaction`: Type of regulation (ID of the controlled reaction: "TC", "TL", "RD", "PD" or "PTM").
  - `RegSign`: Sign of the reaction ("1" for positive regulation, "-1" for negative regulation).
  - `RegBy`: Type of the regulator: "PC" for protein-coding regulation, "NC" for noncoding regulator, "C" for regulatory complex.
- `mosystem`: A list of the different regulatory networks (each corresponding to a different type of regulation) in the system and associated information. Elements are `TCRN_edg`, `TLRN_edg`, `RDRN_edg`, `PDRN_edg` and `PTMRN_edg`: data-frames of edges for the different regulatory networks, which in addition to the usual fields in the edg data frame, contain columns for kinetic parameters of the regulation.
- `complexes`: a list of regulatory complexes composition. The names of the elements are the IDs of the complexes, and the values are vectors of gene IDs constituting each regulatory complex.
- `complexeskinetics`: a list of regulatory complexes kinetic parameters.
- `complexesTargetReaction`: a list defining which expression step is targeted by each regulatory complex.

Examples

```
mysysargs = insilicosystemargs(G = 5)
mygenes = createGenes(mysysargs)
mynetwork = createMultiOmicNetwork(mygenes, mysysargs)
```

createRegulatoryNetwork

Creates an in silico regulatory network.

Description

Creates an in silico regulatory network given a list of regulators and targets.

Usage

```
createRegulatoryNetwork(
  regsList,
  tarsList,
  reaction,
  sysargs,
  ev = getJuliaEvaluator()
)
```

Arguments

`regsList`	A named list of length 2. Element "PC" (resp."NC") is a vector of gene IDs of the protein-coding (resp. noncoding) regulators for the network.
`tarsList`	A named list of length 2. Element "PC" (resp."NC") is a vector of gene IDs of the potential targets of the protein-coding (resp. noncoding) regulators.
`reaction`	String. The ID of the reaction targeted by the interactions ("TC", "TL", "RD", "PD" or "PTM").
`sysargs`	An object of class `insilicosystemargs` (i.e. a list with parameters for in silico system generation).
`ev`	A Julia evaluator (for the XRJulia package). If none provided select the current evaluator or create one if no evaluator exists.

Value

A list of two elements:

- `edg`: a data-frame of edges of the network with the following variables:
  - `from`: gene ID of the regulator, as a character;
  - `to`: gene ID of the target, as an integer;
  - `TargetReaction`: the ID of the reaction (as given by `reaction`);
  - `RegSign`: The sign of the reaction ("1" or "-1");
  - `RegBy`: Is the regulator a protein-coding gene ("PC"), a noncoding gene ("NC") or a complex ("C")?
- `complexes`: a list of complexes composition (each element is named with the complex ID, the components are given as gene IDs).
- `complexesTargetReaction`: a list defining which expression step the different regulatory complexes target (each element is named with the complex ID, the targeted reaction are given with a reaction ID, e.g. "TC" for transcription).

Examples

```
## We want to create a small transcription regulatory network
## In this example, genes 1 and 2 are protein-coding regulators (say transcription factors),
## gene 3 is a noncoding regulator (say an miRNA), and genes 4-6 are the genes to be regulated
## (all protein-coding, e.g. all encoding enzymes)
createRegulatoryNetwork(regsList = list("PC" = c(1:2), "NC" = c(3)),
     tarsList = list("PC" = c(4:6), "NC" = integer(0)), reaction = "TC",
     sysargs = insilicosystemargs(G = 6))
```

---

createStochSystem *Creates a stochastic system from an in silico system.*

Description

Creates a list of molecules, reactions and associated propensities to represent the in silico system.

Usage

```
createStochSystem(
  insilicosystem,
  writefile = F,
  filepath = NULL,
  filename = "simulation",
  verbose = T,
  ev = getJuliaEvaluator()
)
```

Arguments

`insilicosystem`	The in silico system (object of class `insilicosystem`, see `createInSilicoSystem`).
`writefile`	Does the julia function write the species and reactions lists in a text file?
`filepath`	If writefile = `TRUE`, path to the folder in which the files will be created (default: current working directory).
`filename`	If writefile = `TRUE`, prefix of the files created to store the lists of species and reactions (default: none).
`verbose`	If TRUE (default), print messages to signal the start and finish of the function.
`ev`	A Julia evaluator (for the XRJulia). If none provided select the current evaluator or create one if no evaluator exists.

Value

A Julia proxy object to retrieve the stochastic system in the Julia evaluator.

Examples

```
mysystem = createInSilicoSystem(G = 5)
stochsys = createStochSystem(mysystem)
```

`createVariants` *Create variants for genes in the system.*

Description

Create variants that segregate in the in silico population for each gene in the system.

Usage

```
createVariants(genes, indargs)
```

Arguments

`genes`	A data frame of genes in the system (created by the function `createGenes`).
`indargs`	An object of class `insilicoindividualargs` (i.e. a list with parameters for in silico individuals generation).

Value

A list of size G (number of genes in the system) where each element is a matrix corresponding to the QTL effect coefficients (rows) of the different variants (columns) segregating in the in silico population for the corresponding gene. A variant is defined by a set of QTL effect coefficients ("qtlTCrate", "qtlRDrate", "qtlTCregbind", "qtlRDregrate", "qtlactivity", "qtlTLrate", "qtlPDrate", "qtlTLregbind", "qtlPDregrate") that correspond to the impact of genetic mutations carried by the variant on the different kinetic parameters of the gene, as follow:

- `qtlTCrate`: affects the basal transcription rate of the gene;
- `qtlRDrate`: Affects the basal RNA decay rate of the gene;
- `qtlTCregbind`: Affects the binding rate of the regulators of transcription on the gene's promoter (affects all transcription regulators targeting this gene);
- `qtlRDregrate`: Affects the rate at which regulators of RNA decay encountering the RNAs of the gene trigger their degradation (affects all RNA decay regulators targeting this gene);
- `qtlactivity`: Affects the activity of the active product of the gene. If the gene is encoding for a regulator of transcription or translation, this affects the binding rate of its active products (i.e. RNAs or proteins) to their binding sites on their targets (affects the binding to all targets of the gene). If the gene encodes a regulator of RNA or protein decay or of protein post-translational modification, this affects the rate at which its active products (i.e. RNAs or proteins) trigger the degradation/transformation of their targets (effect for all targets of the gene);
- `qtlTLrate`: Affects the basal translation rate of the gene;

- `qtlPDrate`: Affects the basal protein decay rate of the gene;
- `qtlTLregbind`: Affects the binding rate of the regulators of translation on the gene's RNA binding sites (affects all translation regulators targeting this gene);
- `qtlPDregrate`: Affects the rate at which regulators of protein decay encountering the proteins of the gene trigger their degradation (affects all protein decay regulators targeting this gene);
- `qtlPTMregrate`: Affects the rate at which regulators of protein post-translational modification encountering the proteins of the gene trigger their modification (affects all protein post-translational modification regulators targeting this gene).

Examples

```
indargs = insilicoindividualargs()
genes = createGenes(insilicosystemargs(G = 5))
variants = createVariants(genes, indargs)
```

`df2list` *Tranforms a data-frame into a list.*

Description

Transforms a data-frame into a list. The elements of the list correspond to the columns of the data-frame.

Usage

```
df2list(mydf)
```

Arguments

`mydf`	A data-frame.

Value

A named list with elements corresponding to the columns of the input data-frame.

`findJuliaNoError` *Find a Julia executable*

Description

The function is almost identical to the `findJulia` function from the `XRJulia` package, but prevent errors arising from looking for a Julia executable with the "which"/"where" function if Julia is not present.

Usage

```
findJuliaNoError(test = FALSE)
```

Arguments

`test`	If TRUE, returns TRUE/FALSE depending on whether or not a Julia executable is found. If FALSE, returns the path to the Julia executable if it exists.

Value

TRUE/FALSE or the path to the Julia executable

Examples

```
finJuliaNoError(test = T)
```

`getGRN` *Returns an igraph object (network) of the GRN of the in silico system.*

Description

Returns an igraph object (network) corresponding to the gene regulatory network of the insilico system, including all types of regulation of only those defined by the user.

Usage

```
getGRN(insilicosystem, edgeType = NULL, showAllVertices = F)
```

Arguments

`insilicosystem`	The in silico system (see `createInSilicoSystem`).
`edgeType`	The type of interactions to include in the network. If NULL (default value), all the interactions are included. Otherwise, can be either: • "TC": return only regulation of transcription • "TL": return only regulation of translation • "RD": return only regulation of RNA decay • "PD": return only regulation of protein decay • "PTM": return only regulation of protein post-translational modification • "RegComplexes": return only binding interactions, i.e. linking the regulatory complexes to their components.
`showAllVertices`	Display vertices that don't have any edge? Default is FALSE.

Examples

```
mysystem = createInSilicoSystem(G = 10)
grn = getGRN(mysystem)
grnTC = getGRN(mysystem, edgeType = "TC", showAllVertices = F)
grnTCall = getGRN(mysystem, edgeType = "TC", showAllVertices = T)
```

`getJuliaEvaluator` *Returns the current Julia evaluator.*

Description

Returns the current Julia evaluator; if none, starts a new one.

Usage

```
getJuliaEvaluator()
```

Details

`getJuliaEvaluator` is similar to the `XRJulia` function `RJulia`, but if no evaluator exists, creates a new one and loads `sismonr` Julia functions on it.

Value

A Julia evaluator from `XRJulia` package.

Examples

```
getJuliaEvaluator()
```

getRNAseqMatrix — *Transforms a simulation time-point into RNA-seq-like data.*

Description

Transforms a time-point of a simulation into RNA-seq-like data, i.e. simulates a read count for each RNA molecule per individual (each individual is considered as a sample).

Usage

```
getRNAseqMatrix(
  simdf,
  insilicosystem,
  samplingTime = max(simdf$time),
  laneEffect = F,
  nLanes = 2,
  propRnasSampled = 0.9,
  samplesLibSize = NULL,
  genesLength = NULL,
  mrnasOnly = T,
  mergeComplexes = F,
  meanLogLibSize_lane = 7,
  sdLogLibSize_lane = 0.5,
  sdLogLibSize_samples = 0.2
)
```

Arguments

`simdf`	The data-frame with the result of the simulation (see `simulateInSilicoSystem`).
`insilicosystem`	The simulated in silico system (see `createInSilicoSystem`).
`samplingTime`	Numeric. Time-point of the simulation to be transformed. By default, the maximum time of the simulation is used.
`laneEffect`	Boolean. Are the samples processed on different lanes/batches (see `sampleLibrarySize`)? Ignored if `samplesLibSize` is provided. Default value is FALSE.
`nLanes`	Numeric. How many lanes are there in the experiment (see `sampleLibrarySize`)? Automatically set to 1 if `laneEffect = F`. Ignored if `samplesLibSize` is provided. Default value is 2.
`propRnasSampled`	Numeric. The proportion of molecules of RNAs that are sampled in each individual. Must be between 0 and 1. Default value is 0.9.
`samplesLibSize`	Vector of expected library size for each individual/sample. If named, the names of the vector must correspond to the names of the individuals as specified in the result of the simulation. If none provided, will be sampled from a log-normal distribution (see `sampleLibrarySize`). Default value is NULL.

`genesLength`	Vector of gene length for each gene in the system. Its length must be equal to the number of protein-coding genes in the system (if `mrnasOnly` is TRUE) or of genes in the system (if `mrnasOnly` is FALSE). If named, the names must correspond to the ID of the (protein-coding) genes in the system. If none provided, all genes are assumed to be of length 1.
`mrnasOnly`	Boolean. Are the noncoding RNAs to be discarded before the transformation? If TRUE, read counts will be returned only for protein-coding RNAs. Default value is TRUE.
`mergeComplexes`	Boolean. Are the RNAs in complex accounted for in the read counts (i.e. are they detected by the RNA-seq experiment)? Default value is FALSE. See also `mergeComplexesAbundance`.
`meanLogLibSize_lane`	Numeric. The mean of the log10 mean library size normal distribution (see `sampleLibrarySize`). Ignored if `samplesLibSize` is provided. Default value of 7.
`sdLogLibSize_lane`	Numeric. The sd of the log10 mean library size normal distribution (see `sampleLibrarySize`). Ignored if `samplesLibSize` is provided. Default value of 0.5.
`sdLogLibSize_samples`	Numeric. The sd of the log10 samples library size normal distribution (see `sampleLibrarySize`). Ignored if `samplesLibSize` is provided. Default value of 0.2.

Details

The abundance of the RNA form of each gene at time `samplingTime` is extracted from the result of the simulation. If `mrnasOnly = TRUE`, non-coding RNAs are discarded. If the simulation contains several trials per individual (see `simulateInSilicoSystem`), the abundance of each RNA is summed over the different trials for each individual. The abundance of the RNAs in each individual are then transformed into "noisy proportions": for each individual, `tot_RNAs` times `propRnasSampled` RNAs molecules are sampled from the total RNA molecules in the individual (where `tot_RNAs` is the total number of RNA molecules of a given individual). If `propRnasSampled` is 1, this step simply corresponds to dividing the abundance of each RNA by the total number of RNAs for the individual. Otherwise, if `propRnasSampled` is less than 1, it introduces some stochasticity in the "measurement" of RNAs as some low-expressed RNAs might not be represented. The noisy proportion of for each RNA is multiplied by the gene length (by default set to 1 for all genes), to reproduce the bias of RNA-seq experiments in which longer genes get more reads. The proportions are re-scaled such that their sum over all RNAs for each individual equals 1. The expected count of each RNA in each individual is then computed as the product of the corresponding noisy proportion and the library size of the individual. The latter can be provided by the user (parameter `samplesLibSize`); otherwise it is sampled using the function `sampleLibrarySize`. Finally, the actual read count of each RNA for each individual is sampled from a Poisson distribution with parameter lambda equal to the corresponding expected count.

Value

A list:

- `rnaSeqMatrix` A tibble giving for each RNA (column "Molecule") the observed read count in each individual (other columns, one per individual).
- `samplesLibSize` The expected library size of each individual. May not be equal to the total read counts for the individual, as the actual counts are sampled from a Poisson distribution.
- `genesLength` The length of each gene.

Examples

```
mysystem = createInSilicoSystem(G = 5, regcomplexes = "none",
                                ploidy = 2, PC.p = 1)
mypop = createInSilicoPopulation(10, mysystem)
sim = simulateInSilicoSystem(mysystem, mypop, simtime = 1000,
                             ntrials = 10, nepochs = 5)
rnaSeq = getRNAseqMatrix(sim$Simulation, mysystem, laneEffect = F)

## With a batch/lane effect on the library size of the samples
rnaSeq = getRNAseqMatrix(sim$Simulation, mysystem, laneEffect = T)

## Providing the library size of each sample/individual
libsize = rnorm(length(unique(sim$Simulation$Ind)), 1e7, 1e5)
names(libsize) = unique(sim$Simulation$Ind)
rnaSeq = getRNAseqMatrix(sim$Simulation, mysystem, samplesLibSize = libsize)

## Accounting for different gene lengths
genes_length = sample(1:200, nrow(mysystem$genes))
names(genes_length) = as.character(1:nrow(mysystem$genes))
rnaSeq = getRNAseqMatrix(sim$Simulation, mysystem,
                         samplesLibSize = libsize, genesLength = genes_length)
```

`insilicoindividualargs` *Constructor function for the* `insilicoindividualargs` *class.*

Description

Constructor function for the `insilicoindividualargs` class, with default values for the parameters if not provided by the user.

Usage

```
insilicoindividualargs(
  ngenevariants = 5,
 qtleffect_samplingfct = function(x) {    truncnorm::rtruncnorm(x, a = 0, b = Inf,
    mean = 1, sd = 0.1) },
 initvar_samplingfct = function(x) {    truncnorm::rtruncnorm(x, a = 0, b = Inf, mean
    = 1, sd = 0.1) }
)
```

Arguments

`ngenevariants`	Integer. Number of alleles existing for each gene and segregating in the in silico population. Default value is 5.
`qtleffect_samplingfct`	Function from which is sampled the value of a QTL effect coefficient (input x is the required sample size). Default value is a truncated normal distribution with mean 1 and sd 0.1 (only gives positive values).
`initvar_samplingfct`	Function from which is sampled the variation of the initial abundance of a species (input x is the required sample size). Default value is a truncated normal distribution with mean 1 and sd 0.1 (only gives positive values).

Value

An object of the class `insilicoindividualargs`, that is a named list of the different parameters.

Examples

```
indargs = insilicoindividualargs(ngenevariants = 3)
```

`insilicosystemargs` *Constructor function for the* `insilicosystemsargs` *class.*

Description

Constructor function for the `insilicosystemsargs` class, with default values for the parameters if not provided by the user.

Usage

```
insilicosystemargs(
  G = 10,
  ploidy = 2,
  PC.p = 0.7,
  PC.TC.p = NULL,
  PC.TL.p = NULL,
  PC.RD.p = NULL,
  PC.PD.p = NULL,
  PC.PTM.p = NULL,
  PC.MR.p = NULL,
  NC.TC.p = NULL,
  NC.TL.p = NULL,
  NC.RD.p = NULL,
  NC.PD.p = NULL,
  NC.PTM.p = NULL,
  TC.pos.p = 0.5,
  TL.pos.p = 0.5,
  PTM.pos.p = 0.5,
  basal_transcription_rate_samplingfct = NULL,
  basal_translation_rate_samplingfct = NULL,
  basal_RNAlifetime_samplingfct = NULL,
  basal_protlifetime_samplingfct = NULL,
  TC.PC.outdeg.distr = "powerlaw",
  TC.NC.outdeg.distr = "powerlaw",
  TC.PC.outdeg.exp = 3,
  TC.NC.outdeg.exp = 5,
  TC.PC.indeg.distr = "powerlaw",
  TC.NC.indeg.distr = "powerlaw",
  TC.PC.autoregproba = 0.2,
  TC.NC.autoregproba = 0,
  TC.PC.twonodesloop = FALSE,
  TC.NC.twonodesloop = FALSE,
  TCbindingrate_samplingfct = NULL,
  TCunbindingrate_samplingfct = NULL,
  TCfoldchange_samplingfct = NULL,
  TL.PC.outdeg.distr = "powerlaw",
  TL.NC.outdeg.distr = "powerlaw",
  TL.PC.outdeg.exp = 4,
  TL.NC.outdeg.exp = 6,
  TL.PC.indeg.distr = "powerlaw",
  TL.NC.indeg.distr = "powerlaw",
  TL.PC.autoregproba = 0.2,
  TL.NC.autoregproba = 0,
  TL.PC.twonodesloop = FALSE,
  TL.NC.twonodesloop = FALSE,
  TLbindingrate_samplingfct = NULL,
  TLunbindingrate_samplingfct = NULL,
  TLfoldchange_samplingfct = NULL,
  RD.PC.outdeg.distr = "powerlaw",
  RD.NC.outdeg.distr = "powerlaw",
  RD.PC.outdeg.exp = 4,
  RD.NC.outdeg.exp = 6,
  RD.PC.indeg.distr = "powerlaw",
  RD.NC.indeg.distr = "powerlaw",
  RD.PC.autoregproba = 0.2,
  RD.NC.autoregproba = 0,
  RD.PC.twonodesloop = FALSE,
  RD.NC.twonodesloop = FALSE,
  RDregrate_samplingfct = NULL,
  PD.PC.outdeg.distr = "powerlaw",
  PD.NC.outdeg.distr = "powerlaw",
  PD.PC.outdeg.exp = 4,
  PD.NC.outdeg.exp = 6,
  PD.PC.indeg.distr = "powerlaw",
  PD.NC.indeg.distr = "powerlaw",
  PD.PC.autoregproba = 0.2,
  PD.NC.autoregproba = 0,
  PD.PC.twonodesloop = FALSE,
  PD.NC.twonodesloop = FALSE,
  PDregrate_samplingfct = NULL,
  PTM.PC.outdeg.distr = "powerlaw",
  PTM.NC.outdeg.distr = "powerlaw",
  PTM.PC.outdeg.exp = 4,
  PTM.NC.outdeg.exp = 6,
  PTM.PC.indeg.distr = "powerlaw",
  PTM.NC.indeg.distr = "powerlaw",
  PTM.PC.autoregproba = 0.2,
  PTM.NC.autoregproba = 0,
  PTM.PC.twonodesloop = FALSE,
  PTM.NC.twonodesloop = FALSE,
  PTMregrate_samplingfct = NULL,
  regcomplexes = "prot",
  regcomplexes.p = 0.3,
  regcomplexes.size = 2,
  complexesformationrate_samplingfct = NULL,
  complexesdissociationrate_samplingfct = NULL
)
```

Arguments

`G`	Integer. Number of genes in the system. Default value is 10.
`ploidy`	Numeric. The ploidy of the system, i.e. how many copies of each gene are present in the system. Default value is 2.
`PC.p`	Numeric. Probability of each gene to be a protein-coding gene. Default value is 0.7.
`PC.TC.p`	Numeric. Probability of a protein-coding gene to be a regulator of transcription. Default value is 0.4 (see details).
`PC.TL.p`	Numeric. Probability of a protein-coding gene to be a regulator of translation. Default value is 0.3 (see details).
`PC.RD.p`	Numeric. Probability of a protein-coding gene to be a regulator of RNA decay. Default value is 0.1 (see details).
`PC.PD.p`	Numeric. Probability of a protein-coding gene to be a regulator of protein decay. Default value is 0.1 (see details).
`PC.PTM.p`	Numeric. Probability of a protein-coding gene to be a regulator of protein post-translational modification. Default value is 0.05 (see details).
`PC.MR.p`	Numeric. Probability of a protein-coding gene to be a metabolic enzyme. Default value is 0.05 (see details).
`NC.TC.p`	Numeric. Probability of a noncoding gene to be a regulator of transcription. Default value is 0.3 (see details).

`NC.TL.p`	Numeric. Probability of a noncoding gene to be a regulator of translation. Default value is 0.3 (see details).
`NC.RD.p`	Numeric. Probability of a noncoding gene to be a regulator of RNA decay. Default value is 0.3 (see details).
`NC.PD.p`	Numeric. Probability of a noncoding gene to be a regulator of protein decay. Default value is 0.05 (see details).
`NC.PTM.p`	Numeric. Probability of a noncoding gene to be a regulator of protein post-translational modification. Default value is 0.05 (see details).
`TC.pos.p`	Numeric. Probability of a regulation targeting gene transcription to be positive. Default value is 0.5.
`TL.pos.p`	Numeric. Probability of a regulation targeting gene translation to be positive. Default value is 0.5.
`PTM.pos.p`	Numeric. Probability of a regulation targeting protein post-translational modification to be positive (i.e the targeted protein is transformed into its modified form, as opposed to the modified protein being transformed back into its original form). Default value is 0.5.
`basal_transcription_rate_samplingfct`	Function from which the transcription rates of genes are sampled (input x is the required sample size). Default value is a function returning `(10^v)/3600`, with `v` a vector of size x sampled from a normal distribution with mean of 3 and sd of 0.5.
`basal_translation_rate_samplingfct`	Function from which the translation rates of genes are sampled (input x is the required sample size). Default value is a function returning `(10^v)/3600`, with `v` a vector of size x sampled from a normal distribution with mean of 2.146 and sd of 0.7.
`basal_RNAlifetime_samplingfct`	Function from which the transcript lifetimes are sampled (input x is the required sample size). Default value is a function returning `(10^v)*3600`, with `v` a vector of size x sampled from a normal distribution with mean of 0.95 and sd of 0.2.
`basal_protlifetime_samplingfct`	Function from which the protein lifetime are sampled (input x is the required sample size). Default value is a function returning `(10^v)*3600`, with `v` a vector of size x sampled from a normal distribution with mean of 1.3 and sd of 0.4.
`TC.PC.outdeg.distr`	Form of the distribution of the number of targets (out-degree) of protein regulators in the transcription regulation graph; can be either "powerlaw" or "exponential". Default value is "powerlaw".
`TC.NC.outdeg.distr`	Form of the distribution of the number of targets (out-degree) of noncoding regulators in the transcription regulation graph; can be either "powerlaw" or "exponential". Default value is "powerlaw".
`TC.PC.outdeg.exp`	Numeric. Exponent of the distribution for the out-degree of the protein regulators in the transcription regulation graph. Default value is 3.

`TC.NC.outdeg.exp`
: Numeric. Exponent of the distribution for the out-degree of the noncoding regulators in the transcription regulation graph. Default value is 5.

`TC.PC.indeg.distr`
: Type of preferential attachment for the targets of protein regulators in the transcription regulation graph; can be either "powerlaw" or "exponential". Default value is "powerlaw".

`TC.NC.indeg.distr`
: Type of preferential attachment for the targets of noncoding regulators in the transcription regulation graph; can be either "powerlaw" or "exponential". Default value is "powerlaw".

`TC.PC.autoregproba`
: Numeric. Probability of protein regulators to perform autoregulation in the transcription regulation graph. Default value is 0.2.

`TC.NC.autoregproba`
: Numeric. Probability of noncoding regulators to perform autoregulation in the transcription regulation graph. Default value is 0.

`TC.PC.twonodesloop`
: Logical. Are 2-nodes loops authorised in the transcription regulation graph with protein regulators? Default value is FALSE.

`TC.NC.twonodesloop`
: Logical. Are 2-nodes loops authorised in the transcription regulation graph with noncoding regulators? Default value is FALSE.

`TCbindingrate_samplingfct`
: Function from which the binding rates of transcription regulators on their targets are sampled (input `means` is a vector of length equal to the required sample size, giving for each edge (regulatory interaction) for which a binding rate is being sampled the value of the sampled unbinding rate divided by the steady-state abundance of the regulator in absence of any regulation in the system). Default value is a function returning `10^v`, where `v` is a vector with the same length as `means` whose elements are sampled from a truncated normal distribution with mean equal to the log10 of the corresponding element in `means`, and sd = 0.1, the minimum authorised value being the log10 of the corresponding element in `means`.

`TCunbindingrate_samplingfct`
: Function from which the unbinding rates of transcription regulators from their target are sampled (input x is the required sample size). Default value is a function returning `10^v`, with `v` a vector of size x sampled from a normal distribution with mean of -3 and sd of 0.2.

`TCfoldchange_samplingfct`
: Function from which the transcription fold change induced by a bound regulator is sampled (input x is the required sample size). Default value is a truncated normal distribution with a mean of 3, sd of 10 and minimum authorised value of 1.5.

`TL.PC.outdeg.distr`
: Form of the distribution of the number of targets (out-degree) of protein regulators in the translation regulation graph; can be either "powerlaw" or "exponential". Default value is "powerlaw".

`TL.NC.outdeg.distr`
: Form of the the distribution of the number of targets (out-degree) of noncoding regulators in the translation regulation graph; can be either "powerlaw" or "exponential". Default value is "powerlaw".

`TL.PC.outdeg.exp`
: Numeric. Exponent of the distribution for the out-degree of the protein regulators in the translation regulation graph. Default value is 4.

`TL.NC.outdeg.exp`
: Numeric. Exponent of the distribution for the out-degree of the noncoding regulators in the translation regulation graph. Default value is 6.

`TL.PC.indeg.distr`
: Type of preferential attachment for the targets of protein regulators in the translation regulation graph; can be either "powerlaw" or "exponential". Default value is "powerlaw".

`TL.NC.indeg.distr`
: Type of preferential attachment for the targets of noncoding regulators in the translation regulation graph; can be either "powerlaw" or "exponential". Default value is "powerlaw".

`TL.PC.autoregproba`
: Numeric. Probability of protein regulators to perform autoregulation in the translation regulation graph. Default value is 0.2.

`TL.NC.autoregproba`
: Numeric. Probability of noncoding regulators to perform autoregulation in the translation regulation graph. Default value is 0.

`TL.PC.twonodesloop`
: Logical. Are 2-nodes loops authorised in the translation regulation graph with protein regulators? Default value is FALSE.

`TL.NC.twonodesloop`
: Logical. Are 2-nodes loops authorised in the translation regulation graph with noncoding regulators? Default value is FALSE.

`TLbindingrate_samplingfct`
: Function from which the binding rate of translation regulators on target are sampled (input `means` is a vector of length equal to the required sample size, giving for each edge (regulatory interaction) for which a binding rate is being sampled the value of the sampled unbinding rate divided by the steady-state abundance of the regulator in absence of any regulation in the system). Default value is a function returning `10^v`, where `v` is a vector with the same length as `means` whose elements are sampled from a truncated normal distribution with mean equal to the log10 of the corresponding element in `means`, and sd = 0.1, the minimum authorised value being the log10 of the corresponding element in `means`.

`TLunbindingrate_samplingfct`
: Function from which the unbinding rate of translation regulators from target are sampled (input `x` is the required sample size). Default value is a function returning `10^v`, with `v` a vector of size `x` sampled from a normal distribution with mean of -3 and sd of 0.2.

`TLfoldchange_samplingfct`
: Function from which the translation fold change induced by a bound regulator are sampled (input `x` is the required sample size). Default value is a truncated

normal distribution with a mean of 3, sd of 10 and minimum authorised value of 1.5.

`RD.PC.outdeg.distr`
: Form of the distribution of the number of targets (out-degree) of protein regulators in the RNA decay regulation graph; can be either "powerlaw" or "exponential". Default value is "powerlaw".

`RD.NC.outdeg.distr`
: Form of the the distribution of the number of targets (out-degree) of noncoding regulators in the RNA decay regulation graph; can be either "powerlaw" or "exponential". Default value is "powerlaw".

`RD.PC.outdeg.exp`
: Numeric. Exponent of the distribution for the out-degree of the protein regulators in the RNA decay regulation graph. Default value is 4.

`RD.NC.outdeg.exp`
: Numeric. Exponent of the distribution for the out-degree of the noncoding regulators in the RNA decay regulation graph. Default value is 6.

`RD.PC.indeg.distr`
: Type of preferential attachment for the targets of protein regulators in the RNA decay graph; can be either "powerlaw" or "exponential". Default value is "powerlaw".

`RD.NC.indeg.distr`
: Type of preferential attachment for the targets of noncoding regulators in the RNA decay graph; can be either "powerlaw" or "exponential". Default value is "powerlaw".

`RD.PC.autoregproba`
: Numeric. Probability of protein regulators to perform autoregulation in the RNA decay regulation graph. Default value is 0.2.

`RD.NC.autoregproba`
: Numeric. Probability of noncoding regulators to perform autoregulation in the RNA decay regulation graph. Default value is 0.

`RD.PC.twonodesloop`
: Logical. Are 2-nodes loops authorised in the RNA decay regulation graph with protein regulators? Default value is FALSE.

`RD.NC.twonodesloop`
: Logical. Are 2-nodes loops authorised in the RNA decay regulation graph with noncoding regulators? Default value is FALSE.

`RDregrate_samplingfct`
: Function from which the RNA decay rates of targets of RNA decay regulators are sampled (input `x` is the required sample size). Default value is a function returning `10^v`, with `v` a vector of size `x` sampled from a normal distribution with mean of -5 and sd of 1.5.

`PD.PC.outdeg.distr`
: Form of the distribution of the number of targets (out-degree) of protein regulators in the protein decay regulation graph; can be either "powerlaw" or "exponential". Default value is "powerlaw".

`PD.NC.outdeg.distr`
: Form of the the distribution of the number of targets (out-degree) of noncoding regulators in the protein decay regulation graph; can be either "powerlaw" or "exponential". Default value is "powerlaw".

`PD.PC.outdeg.exp`
: Numeric. Exponent of the distribution for the out-degree of the protein regulators in the protein decay regulation graph. Default value is 4.

`PD.NC.outdeg.exp`
: Numeric. Exponent of the distribution for the out-degree of the noncoding regulators in the protein decay regulation graph. Default value is 6.

`PD.PC.indeg.distr`
: Type of preferential attachment for the targets of protein regulators in the protein decay regulation graph; can be either "powerlaw" or "exponential". Default value is "powerlaw".

`PD.NC.indeg.distr`
: Type of preferential attachment for the targets of noncoding regulators in the protein decay graph; can be either "powerlaw" or "exponential". Default value is "powerlaw".

`PD.PC.autoregproba`
: Numeric. Probability of protein regulators to perform autoregulation in the protein decay regulation graph. Default value is 0.2.

`PD.NC.autoregproba`
: Numeric. Probability of noncoding regulators to perform autoregulation in the protein decay regulation graph. Default value is 0.

`PD.PC.twonodesloop`
: Logical. Are 2-nodes loops authorised in the protein decay graph with protein regulators in the protein decay regulation graph? Default value is FALSE.

`PD.NC.twonodesloop`
: Logical. Are 2-nodes loops authorised in the protein decay graph with noncoding regulators in the protein decay regulation graph? Default value is FALSE.

`PDregrate_samplingfct`
: Function from which the protein decay rates of targets of protein decay regulators are sampled (input x is the required sample size). Default value is a function returning `10^v`, with `v` a vector of size x sampled from a normal distribution with mean of -5 and sd of 1.5.

`PTM.PC.outdeg.distr`
: Form of the distribution of the number of targets (out-degree) of protein regulators in the post-translational modification regulation graph; can be either "powerlaw" or "exponential". Default value is "powerlaw".

`PTM.NC.outdeg.distr`
: Form of the the distribution of the number of targets (out-degree) of noncoding regulators in the post-translational modification regulation graph; can be either "powerlaw" or "exponential". Default value is "powerlaw".

`PTM.PC.outdeg.exp`
: Numeric. Exponent of the distribution for the out-degree of the protein regulators in the protein post-translational modification graph. Default value is 4.

`PTM.NC.outdeg.exp`
: Numeric. Exponent of the distribution for the out-degree of the noncoding regulators in the protein post-translational modification graph. Default value is 6.

`PTM.PC.indeg.distr`
: Type of preferential attachment for the targets of protein regulators in the protein post-translational modification graph; can be either "powerlaw" or "exponential". Default value is "powerlaw".

`PTM.NC.indeg.distr`
: Type of preferential attachment for the targets of noncoding regulators in the protein post-translational modification graph; can be either "powerlaw" or "exponential". Default value is "powerlaw".

`PTM.PC.autoregproba`
: Numeric. Probability of protein regulators to perform autoregulation. Default value is 0.2.

`PTM.NC.autoregproba`
: Numeric. Probability of noncoding regulators to perform autoregulation. Default value is 0.

`PTM.PC.twonodesloop`
: Logical. Are 2-nodes loops authorised in the protein post-translational modification graph with protein regulators? Default value is FALSE.

`PTM.NC.twonodesloop`
: Logical. Are 2-nodes loops authorised in the protein post-translational modification graph with noncoding regulators? Default value is FALSE.

`PTMregrate_samplingfct`
: Function from which the protein transformation rates of targets of post-translational modification regulators are sampled (input x is the required sample size). Default value is a function returning `10^v`, with `v` a vector of size x sampled from a normal distribution with mean of -5 and sd of 1.5.

`regcomplexes`
: Can the regulators controlling a common target form regulatory complexes in the different regulatory graphs? Can be 'none', 'prot' (only protein can form regulatory complexes) or 'both' (both regulatory RNAs and proteins can form regulatory complexes). Default value is "prot".

`regcomplexes.p`
: Numeric. Probability that regulators controlling a common target form regulatory complexes; ignore if `regcomplexes` = 'none'. Default value is 0.3.

`regcomplexes.size`
: Integer. Number of components of a regulatory complex; ignore if `regcomplexes` = 'none'. Default value is 2.

`complexesformationrate_samplingfct`
: Function from which the formation rate of regulatory complexes are sampled (input x is the required sample size). Default value is a function returning `10^v`, with `v` a vector of size x sampled from a normal distribution with mean of -3 and sd of 0.7.

`complexesdissociationrate_samplingfct`
: Function from which the dissociation rate of regulatory complexes are sampled (input x is the required sample size). Default value is a function returning `10^v`, with `v` a vector of size x sampled from a normal distribution with mean of 3 and sd of 0.9.

Details

For the protein-coding (and non-coding) biological function ratios (i.e. PC.TC.p, PC.TL.p, etc): if none of the ratios are provided, then they are set to their default values. Otherwise, if at least one value among the 6 (5 for noncoding genes) is set by the user:

- if the sum of the provided values is 1 or more: the non-specified values are set to 0, and the specified values are normalised such that their sum is 1.
- if the sum of the provided values is less than 1: the non-specified values are set such that the sum of all ratios equals 1.

Example: if the user sets PC.TC.p to 1 and PC.TL.p to 0.6, but does not provide any values for the other ratios, then PC.TC.p is set to 1/(1+0.6)=0.625, PC.TL.p to 0.6/(1+0.6)=0.375, and PC.RD.p, PC.PD.p, PC.PTM.p and PC.MR.p are all set to 0. Accordingly, if the user only sets NC.TC.p to 0.6, then NC.TL.p, NC.RD.p, NC.PD.p and NC.PTM.p are all set to 0.1.

Value

An object of the class `insilicosystemargs`, that is a named list of the different parameters.

Examples

```
sysargs = insilicosystemargs(G = 15, PC.p = 0.2,
 basal_transcription_rate_samplingfct = function(x){runif(x, 0.1, 0.2)})
```

`mergeAlleleAbundance` *Merge the different allelic versions of the molecules.*

Description

Merge (i.e. sum) the abundance of the different allelic versions of each molecule in the results of a simulation.

Usage

```
mergeAlleleAbundance(df)
```

Arguments

`df`	A dataframe with the abundance of the different molecules over time (from `simulateInSilicoSystem` or `simulateParallelInSilicoSystem`).

Value

A dataframe in which the abundance of the different allelic versions of the same molecule have been merged to give the abundance of the molecule (without distinction of the allele of origin).

Examples

```
mysystem = createInSilicoSystem(G = 5, empty = TRUE, ploidy = 2)
mypop = createInSilicoPopulation(1, mysystem)
sim = simulateInSilicoSystem(mysystem, mypop, 100)
head(sim$Simulation)
mergedAllelic = mergeAlleleAbundance(sim$Simulation)
head(mergedAllelic)
```

mergeComplexesAbundance

Merge the free and in-complex versions of molecules.

Description

Merge (i.e. sum) the abundance of the free and in-complex versions of each molecule in the results of a simulation.

Usage

```
mergeComplexesAbundance(df)
```

Arguments

`df`	A dataframe with the abundance of the different molecules over time (from `simulateInSilicoSystem` or `simulateParallelInSilicoSystem`).

Value

A dataframe in which the abundance of free and in complex versions of a molecule have been merged to give the abundance of the molecule (without distinction of whether or not it is bound in a molecular complex).

Examples

```
mysystem = createInSilicoSystem(G = 5, PC.p = 1, PC.TC.p = 1, ploidy = 1)
mysystem = addComplex(mysystem, c(1, 2))
mypop = createInSilicoPopulation(1, mysystem)
sim = simulateInSilicoSystem(mysystem, mypop, 100)
head(sim$Simulation)
mergedComplex = mergeComplexesAbundance(sim$Simulation)
head(mergedComplex)
```

mergePTMAbundance *Merge the original and PTM versions of the proteins.*

Description

Merge (i.e. sum) the abundance of the original and modified (PTM) versions of each protein in the results of a simulation.

Usage

```
mergePTMAbundance(df)
```

Arguments

`df`	A dataframe with the abundance of the different molecules over time (from `simulateInSilicoSystem` or `simulateParallelInSilicoSystem`).

Value

A dataframe in which the abundance of original and modified versions of a protein have been merged to give the abundance of the protein (without distinction of its post-translational modification state).

Examples

```
mysystem = createInSilicoSystem(G = 5, PC.p = 1, PC.PTM.p = 0.9, regcomplexes = "none", ploidy = 1)
mypop = createInSilicoPopulation(1, mysystem)
sim = simulateInSilicoSystem(mysystem, mypop, 100)
head(sim$Simulation)
mergedPTM = mergePTMAbundance(sim$Simulation)
head(mergedPTM)
```

newJuliaEvaluator *Creates a new ready-to-use Julia evaluator.*

Description

`newJuliaEvaluator` opens a new Julia evaluator and loads the required functions on it.

Usage

```
newJuliaEvaluator(port = NULL)
```

Arguments

`port`	An integer specifying the port to be used. Default `NULL`.

Details

If no port number is specified, the `RJulia` function chooses a port to open the evaluator.

Value

A Julia Evaluator from the XRJulia package.

Examples

```
ev = newJuliaEvaluator()
```

`plotGRN` *Plots the GRN of the in silico system.*

Description

Plots the gene regulatory network of the insilico system, including all types of regulation or only those defined by the user.

Usage

```
plotGRN(
  insilicosystem,
  edgeType = NULL,
  showAllVertices = F,
  plotType = "2D",
  ...
)
```

Arguments

`insilicosystem`	The in silico system (see `createInSilicoSystem`).
`edgeType`	The type of interactions to plot. If NULL (default value), all the interactions are plotted. Otherwise, can be either: • "TC": plot only regulation of transcription • "TL": plot only regulation of translation • "RD": plot only regulation of RNA decay • "PD": plot only regulation of protein decay • "PTM": plot only regulation of protein post-translational modification

- "RegComplexes": plot only binding interactions, i.e. linking the regulatory complexes to their components.

`showAllVertices`	Display vertices that don't have any edge? Default is FALSE
`plotType`	The type of plot function to use for the network: can be either "2D" (default, use the function `plot.igraph`) or "interactive2D" (use the function `tkplot`).
`...`	any other arguments to be passed to the plot function, see `igraph.plotting`.

Examples

```
mysystem = createInSilicoSystem(G = 10)
plotGRN(mysystem)
plotGRN(mysystem, edgeType = "TC")
plotGRN(mysystem, edgeType = "TC", showAllVertices = T)
```

`plotHeatMap` *Plots the result of a simulation as a heatmap.*

Description

Automatically plots the result of a simulation for the selected in silico individuals as a heatmap.

Usage

```
plotHeatMap(
  simdf,
  molecules = NULL,
  inds = unique(simdf$Ind),
  trials = unique(simdf$trial),
  timeMin = min(simdf$time),
  timeMax = max(simdf$time),
  mergeAllele = T,
  mergePTM = T,
  mergeComplexes = F,
  yLogScale = T,
  nIndPerRow = 3,
  VirPalOption = "plasma",
  ...
)
```

Arguments

`simdf`	The dataframe with the result of the simulation (see `simulateInSilicoSystem`).
`molecules`	A vector of gene IDs (numeric or character) and/or complex IDs (e.g. CTC1) to be plotted.

`inds`	A vector of in silico individual names for which to plot the expression profiles.
`trials`	A vector of trials ID (= number) to use for the plot (see details).
`timeMin`	Numeric. The minimum simulation time to plot. Default value set to the minimum time in the simulation.
`timeMax`	Numeric. The maximum simulation time to plot. Default value set to the maximum time in the simulation.
`mergeAllele`	Are the gene products originating from different alleles merged? Default TRUE. Also see `mergeAlleleAbundance`
`mergePTM`	Are the modified and non-modified versions of the proteins merged? Default TRUE. Also see `mergePTMAbundance`
`mergeComplexes`	Are the free and in complex gene products merged? Default FALSE. Also see `mergeComplexesAbundance`
`yLogScale`	Plot the y-axis in log10-scale? If so, the abundance of each species at each time-point is increased by 1 to avoid zero values. Default TRUE.
`nIndPerRow`	Positive integer, the number of individuals to plot per row. Default 3.
`VirPalOption`	String, palette name option to be passed to `scale_fill_viridis_c`; can be one of "magma", "inferno", "plasma", "viridis" or "cividis". Default value is "plasma".
`...`	Any additional parameter to be passed to `theme` for the plot of each individual.

Details

If more than one trial is to be plotted, the mean abundance of each molecule over the different trials is plotted with a solid line, and the min and max abundances represented as coloured areas around the mean.

Value

A plot from `ggarrange`.

Examples

```
mysystem = createInSilicoSystem(G = 5, ploidy = 2)
mypop = createInSilicoPopulation(10, mysystem)
sim = simulateInSilicoSystem(mysystem, mypop, 100, ntrials = 5)
plotHeatMap(sim$Simulation,
 c(1, 2, 3),
 c("Ind1", "Ind2", "Ind3", "Ind4"),
 axis.title = element_text(color = "red"))
```

`plotMutations` *Plots the QTL effect coefficients of a population.*

Description

Plots the QTL effect coefficients for all the genes of a in silico system of the in silico individuals.

Usage

```
plotMutations(
  insilicopopulation,
  insilicosystem,
  scaleLims = NULL,
  qtlEffectCoeffs = insilicopopulation$indargs$qtlnames,
  inds = names(insilicopopulation$individualsList),
  alleles = insilicosystem$sysargs$gcnList,
  genes = 1:length(insilicopopulation$GenesVariants),
  nGenesPerRow = 10,
  ...
)
```

Arguments

`insilicopopulation`	The in silico population to be simulated (see `createInSilicoPopulation`).
`insilicosystem`	The in silico system (object of class `insilicosystem`, see `createInSilicoSystem`).
`scaleLims`	A vector of length 2 giving the lower and upper limits of the continuous scale (of the QTL effect coefficient values) to be plotted. QTL effect coefficients with values outside these limits are plotted as NA (gray). If NULL (default), the limits are automatically set to the min and max values in the dataset.
`qtlEffectCoeffs`	A character vector of QTL effect coefficient names to plot. By default, all QTL effect coefficients are represented.
`inds`	A character vector giving the names of the individuals to plot. By default, all individuals in the population are represented.
`alleles`	A character vector giving the names of the alleles to plot. By default, all the alleles are represented.
`genes`	A character or numeric vector of gene IDs to plot. By default, all the genes in the system are represented.
`nGenesPerRow`	Integer. Number of genes to plot per row.
`...`	Any additional parameter to be passed to `theme` for the plot.

Value

A plot representing the value (colour) of each QTL effect coefficient (x-axis) of each allele (y-axis) of the different individuals (rows) for each gene (column) in the system. For noncoding genes, some QTL effect coefficients are not relevant (the ones related to protein or translation) and are represented in gray as NA.

Examples

```
mysystem = createInSilicoSystem(G = 10, ploidy = 2)
mypop = createInSilicoPopulation(10, mysystem)
plotMutations(mypop, mysystem)
## Only plot the 1st allele of each genes for the genes 1 to 5 and the individuals 1 to 3
plotMutations(mypop, mysystem, alleles = c("GCN1"), genes = 1:5,
 inds = c("Ind1", "Ind2", "Ind3"))
```

`plotSimulation` *Plots the result of a simulation.*

Description

Automatically plots the result of a simulation (i.e. the abundance of RNAs, proteins and complexes over time) for the selected in silico individuals.

Usage

```
plotSimulation(
  simdf,
  molecules = NULL,
  inds = unique(simdf$Ind),
  trials = unique(simdf$trial),
  timeMin = min(simdf$time),
  timeMax = max(simdf$time),
  mergeAllele = T,
  mergePTM = T,
  mergeComplexes = F,
  yLogScale = T,
  nIndPerRow = 3,
  nCompPerRow = 10,
  ...
)
```

Arguments

`simdf`	The dataframe with the result of the simulation (see `simulateInSilicoSystem`).
`molecules`	A vector of gene IDs (numeric or character) and/or complex IDs (e.g. CTC1) to be plotted.

`inds`	A vector of in silico individual names for which to plot the expression profiles.
`trials`	A vector of trials ID (= number) to use for the plot (see details).
`timeMin`	Numeric. The minimum simulation time to plot. Default value set to the minimum time in the simulation.
`timeMax`	Numeric. The maximum simulation time to plot. Default value set to the maximum time in the simulation.
`mergeAllele`	Are the gene products originating from different alleles merged? Default TRUE. Also see `mergeAlleleAbundance`
`mergePTM`	Are the modified and non-modified versions of the proteins merged? Default TRUE. Also see `mergePTMAbundance`
`mergeComplexes`	Are the free and in complex gene products merged? Default FALSE. Also see `mergeComplexesAbundance`
`yLogScale`	Plot the y-axis in log10-scale? If so, the abundance of each species at each time-point is increased by 1 to avoid zero values. Default TRUE.
`nIndPerRow`	Positive integer, the number of individuals to plot per row. Default 3.
`nCompPerRow`	Positive integer, the number of components to plot per row in the legend. Default 10.
`...`	Any additional parameter to be passed to `theme` for the plot of each individual.

Details

If more than one trial is to be plotted, the mean abundance of each molecule over the different trials is plotted with a solid line, and the min and max abundances represented as coloured areas around the mean.

Value

A plot from `ggarrange`.

Examples

```
mysystem = createInSilicoSystem(G = 5, regcomplexes = "none", ploidy = 2)
mypop = createInSilicoPopulation(15, mysystem)
sim = simulateInSilicoSystem(mysystem, mypop, 100, ntrials = 5)
plotSimulation(sim$Simulation,
 c(1, 2, 3),
 c("Ind1", "Ind2", "Ind3", "Ind4"),
 axis.title = element_text(color = "red"))
```

removeComplex *Removes a regulatory complex from the in silico system.*

Description

Removes a regulatory complex from the in silico system. Any edge involving this complex is removed from the system.

Usage

```
removeComplex(insilicosystem, name)
```

Arguments

`insilicosystem`	The in silico system (see `createInSilicoSystem`).
`name`	Character. The name of the regulatory complex to remove.

Value

The modified in silico system.

Examples

```
mysystem = createInSilicoSystem(G = 10, PC.p = 1, PC.TC.p = 1, regcomplexes.p = 0.8)
mysystem$complexes
mysystem$edg
mysystem2 = removeComplex(mysystem, "CTC1")
mysystem2$complexes
mysystem2$edg
```

removeEdge *Removes an edge from the in silico system.*

Description

Removes an edge in the in silico system between specified genes.

Usage

```
removeEdge(insilicosystem, regID, tarID)
```

Arguments

`insilicosystem`	The in silico system (see `createInSilicoSystem`).
`regID`	Integer or character. The ID of the regulator gene or the name of the regulatory complex.
`tarID`	Integer or character. The ID of the target gene.

Value

The modified in silico system.

Examples

```
mysystem = createInSilicoSystem(G = 10)
mysystem$edg
## we'll remove the first edge
regToRemove = mysystem$edg$from[1]
tarToRemove = mysystem$edg$to[1]
mysystem2 = removeEdge(mysystem, regToRemove, tarToRemove)
mysystem2$edg
```

`removeJuliaEvaluator` *Closes a Julia evaluator.*

Description

`removeJuliaEvaluator` closes a Julia evaluator from XRJulia package.

Usage

```
removeJuliaEvaluator(ev)
```

Arguments

`ev`	A Julia evaluator from the XRJulia package.

Examples

```
removeJuliaEvaluator(getJuliaEvaluator())
```

`sampleLibrarySize` *Samples the expected library size of individuals/samples*

Description

Samples the expected library size of each individual/sample, accounting for potential lane size effects (i.e. the impact of samples being processed on different lanes).

Usage

```
sampleLibrarySize(
  samples_list,
  meanLogLibSize_lane = 7,
  sdLogLibSize_lane = 0.5,
  sdLogLibSize_samples = 0.2,
  laneEffect = F,
  nLanes = 2
)
```

Arguments

`samples_list`	List of sample/individual names.
`meanLogLibSize_lane`	Numeric. The mean of the log10 mean library size normal distribution (see Details). Default value of 7.
`sdLogLibSize_lane`	Numeric. The sd of the log10 mean library size normal distribution (see Details). Default value of 0.5.
`sdLogLibSize_samples`	Numeric. The sd of the log10 samples library size normal distribution (see Details). Default value of 0.2.
`laneEffect`	Boolean. Are the samples processed on different lanes/batches? Default value is FALSE.
`nLanes`	Numeric. How many lanes are there in the experiment? Automatically set to 1 if `laneEffect = F`. Default value is 2.

Details

The expected library size of each individual is sampled from a log-normal distribution. The mean of this distribution depends on the lane on which the individual/sample is processed. By default, when `laneEffect = FALSE`, all samples are assumed to be processed in a single batch. Thus their library size is sampled from a log-normal distribution with identical mean (equal to `meanLogLibSize_lane`) and sd `sdLogLibSize_samples`. If `laneEffect = TRUE`, the samples are assumed to be processed in `nLanes` batches, that each have a different mean log-library size. In this case, the mean of the log-normal distribution for each lane is sampled from a normal distribution with mean `meanLogLibSize_lane` and sd `sdLogLibSize_lane`. In turn, the expected library size of each individual/sample is sampled form a log-normal distribution with the corresponding lane-dependent mean, and sd `sdLogLibSize_samples`.

Value

A list:

- `lane`: the lane on which each sample is processed.
- `expected_library_size`: the expected library size of each sample.
- `lane_mean_library_size`: the mean library size of each lane.

Examples

```
samples_list = sapply(1:10, function(x){paste0("Ind", x)})
libsize = sampleLibrarySize(samples_list)
libsize = sampleLibrarySize(samples_list, laneEffect = TRUE, nLanes = 3)
```

`simulateInSilicoSystem` *Simulates an in silico system.*

Description

Simulates (stochastically) the behaviour of an in silico system over time, i.e. the expression of the different genes.

Usage

```
simulateInSilicoSystem(
  insilicosystem,
  insilicopopulation,
  simtime,
  nepochs = -1,
  ntrials = 1,
  simalgorithm = "Direct",
  writefile = F,
  filepath = NULL,
  filename = "simulation",
  ev = getJuliaEvaluator()
)
```

Arguments

`insilicosystem`	The in silico system to be simulated (see `createInSilicoSystem`).
`insilicopopulation`	The in silico population to be simulated (see `createInSilicoPopulation`).
`simtime`	The final time of the simulation (in seconds).
`nepochs`	The number of times to record the state of the system during the simulation.
`ntrials`	The number of times the simulation must be replicated (for each individual).

`simalgorithm`	The name of the simulation algorithm to use in the Julia function `simulate` from the module `BioSimulator`. Possible values are: "Direct", "EnhancedDirect", "SortingDirect", "FirstReaction", "NextReaction", "TauLeapingDG2001", "TauLeapingDGLP2003", "StepAnticipation", "HybridSAL". See `https://alanderos91.github.io/BioSimulator.jl/dev/man/algorithms/` for details about the algorithms.
`writefile`	Does the julia function write the species and reactions lists in a text file?
`filepath`	If writefile = `TRUE`, path to the folder in which the files will be created (default: current working directory).
`filename`	If writefile = `TRUE`, prefix of the files created to store the lists of species and reactions.
`ev`	A Julia evaluator. If none provided select the current evaluator or create one if no evaluator exists.

Value

A list composed of:

- `Simulation`: A data-frame with the simulated expression profiles of the genes for the different individuals in the in silico population. For gene i, "Ri" corresponds to the RNA form of the gene, "Pi" to the protein form of the gene. The suffix "GCNj" indicates that the molecule comes from the j-th allele of the gene.
- `runningtime`: A vector of running time of all runs of the simulation for each in silico individuals.
- `stochmodel`: A Julia proxy object to retrieve the stochastic system in the Julia evaluator.

Examples

```
mysystem = createInSilicoSystem(G = 5, regcomplexes = "none", ploidy = 2)
mypop = createInSilicoPopulation(1, mysystem)
sim = simulateInSilicoSystem(mysystem, mypop, simtime = 1000, ntrials = 10)
head(sim$Simulation)
## Visualising the result
plotSimulation(sim$Simulation)
```

---

simulateParallelInSilicoSystem

Simulates an in silico system in parallel.

Description

Simulates (stochastically) the behaviour of an in silico system over time using parallelisation, i.e. the expression of the different genes.

Usage

```
simulateParallelInSilicoSystem(
  insilicosystem,
  insilicopopulation,
  simtime,
  nepochs = -1,
  ntrials = 1,
  simalgorithm = "Direct",
  writefile = F,
  filepath = NULL,
  filename = "simulation",
  no_cores = parallel::detectCores() - 1,
  ev = getJuliaEvaluator()
)
```

Arguments

`insilicosystem`	The in silico system to be simulated (see `createInSilicoSystem`).
`insilicopopulation`	The in silico population to be simulated (see `createInSilicoPopulation`).
`simtime`	The final time of the simulation (in seconds).
`nepochs`	The number of times to record the state of the system during the simulation.
`ntrials`	The number of times the simulation must be replicated (for each individual).
`simalgorithm`	The name of the simulation algorithm to use in the Julia function `simulate` from the module `BioSimulator`. Possible values are: "Direct", "EnhancedDirect", "SortingDirect", "FirstReaction", "NextReaction", "TauLeapingDG2001", "TauLeapingDGLP2003", "StepAnticipation", "HybridSAL". See `https://alanderos91.github.io/BioSimulator.jl/dev/man/algorithms/` for details about the algorithms.
`writefile`	Does the julia function write the species and reactions lists in a text file?
`filepath`	If writefile = `TRUE`, path to the folder in which the files will be created (default: current working directory).
`filename`	If writefile = `TRUE`, prefix of the files created to store the lists of species and reactions.
`no_cores`	The number of cores to use for the simulation. By default use the function `detectCores` from the `parallel` package to detect the number of available cores, and use this number - 1 for the simulation.
`ev`	A Julia evaluator. If none provided select the current evaluator or create one if no evaluator exists.

Value

A list composed of:

- `Simulation`: A data-frame with the simulated expression profiles of the genes for the different individuals in the in silico population. For gene i, "Ri" corresponds to the RNA form of the gene, "Pi" to the protein form of the gene. The suffix "GCNj" indicates that the molecule comes from the j-th allele of the gene.

- `runningtime`: The running time (elapsed seconds) of the parallel simulation (only 1 value).
- `stochmodel`: A Julia proxy object to retrieve the stochastic system in the Julia evaluator.

Examples

```
mysystem = createInSilicoSystem(G = 5, regcomplexes = "none", ploidy = 2)
mypop = createInSilicoPopulation(15, mysystem)
sim = simulateParallelInSilicoSystem(mysystem, mypop, 1000)
head(sim$Simulation)
## Visualising the result
plotSimulation(sim$Simulation)
```

`sortComponents` *Sort component names.*

Description

Sorts the names of the components of an in silico system (for plotting or summary).

Usage

```
sortComponents(componames)
```

Arguments

`componames`	A character vector, giving the names of the components.

Details

Sort components into:

- Genes (first) vs regulatory complexes

Then genes are sorted according to:

- Gene ID (numeric)
- Gene type (RNAs, then proteins, then modified proteins)
- Allele
- If the components don't have a type (e.g. see the legend resulting from `plotSimulation`), non-modified vs modified

Then sort complexes according to:

- Target reaction in the following order: regulators of transcription, translation, RNA decay, protein decay, post-translational modification
- Allele of their components

Value

A dataframe: first column: sorted names of the components, second column: is the component a regulatory complex?, third column: is the component a modified protein?

`steadyStateAbundance` *Computes the steady state abundance of a molecule.*

Description

Computes the steady state abundance of the active product of a gene/regulatory complex (in absence of any regulation).

Usage

```
steadyStateAbundance(id, genes, complexes, ploidy)
```

Arguments

`id`	the ID of the molecule.
`genes`	the data frame of genes in the in silico system.
`complexes`	the list of regulatory complexes and their composition.
`ploidy`	the ploidy of the system.

Details

If `id` represents a gene ID, returns the steady state abundance of its RNA (transcription rate/RNA decay rate) if it is a noncoding gene or the steady state abundance of its protein (RNA steady state * translation rate/protein decay rate) if it is a protein-coding gene. If `id` represents a regulatory complex, returns the minimum of the steady state abundance of its components (recursively if the regulatory complex is composed of other regulatory complexes).

Value

The steady state abundance of the active product of the gene/regulatory complex.

`summariseSimulation` *Returns a summary data-frame of a simulation.*

Description

Returns a summary data-frame of a simulation giving the maximum average abundance of each component over the different trials and the average abundance of the components at the final time of the simulation, for the selected in silico individuals.

Usage

```
summariseSimulation(
  simdf,
  inds = unique(simdf$Ind),
  trials = unique(simdf$trial),
  timeMin = min(simdf$time),
  timeMax = max(simdf$time),
  mergeAllele = T,
  mergePTM = T,
  mergeComplexes = F,
  verbose = T
)
```

Arguments

`simdf`	The data frame with the result of the simulation (see `simulateInSilicoSystem`).
`inds`	A vector of in silico individual names for which to compute the summary values.
`trials`	A vector of trials ID (= number) to use for the summary.
`timeMin`	Numeric. The minimum simulation time to take into account. Default value set to the minimum time in the simulation.
`timeMax`	Numeric. The maximum simulation time to take into account. Default value set to the maximum time in the simulation.
`mergeAllele`	Are the gene products originating from different alleles merged? Default TRUE. Also see `mergeAlleleAbundance`
`mergePTM`	Are the modified and non-modified versions of the proteins merged? Default TRUE. Also see `mergePTMAbundance`
`mergeComplexes`	Are the free and in complex gene products merged? Default FALSE. Also see `mergeComplexesAbundance`
`verbose`	If TRUE (default), print the individuals, trials, min and max time considered for the computation of the summary.

Value

A data-frame giving for each component (rows) and each individual (columns) the max and final average abundance over the different trials.

Examples

```
mysystem = createInSilicoSystem(G = 5, regcomplexes = "none", ploidy = 2)
mypop = createInSilicoPopulation(15, mysystem)
sim = simulateInSilicoSystem(mysystem, mypop, 100, ntrials = 5)
summariseSimulation(sim$Simulation, c("Ind1", "Ind2", "Ind3", "Ind4"))
```

Index