Package ‘dyngen’

September 20, 2021

Type Package

Title A Multi-Modal Simulator for Spearheading Single-Cell Omics Analyses

Version 1.0.3

Description A novel, multi-modal simulation engine for studying dynamic cellular processes at single-cell resolution. ‘dyngen’ is more flexible than current single-cell simulation engines. It allows better method development and benchmarking, thereby stimulating development and testing of novel computational methods. Cannoodt et al. (2021) <doi:10.1038/s41467-021-24152-2>.

License MIT + file LICENSE


BugReports https://github.com/dynverse/dyngen/issues

Depends R (>= 3.5.0)

Imports assertthat, dplyr, dynutils (>= 1.0.4), ggplot2, ggraph (>= 2.0), ggrepel, GillespieSSA2 (>= 0.2.6), grDevices, grid, igraph, lmds, Matrix, methods, patchwork, pbapply, purrr, rlang (>= 0.4.1), stats, tibble, tidygraph, tidyr, utils, viridis

Suggests annndata, covr, dynwrap (>= 1.2.0), R.rsp, rmarkdown, Seurat, SingleCellExperiment (>= 1.5.3), SummarizedExperiment, testthat

VignetteBuilder R.rsp

biocViews

Encoding UTF-8

LazyData TRUE

RoxygenNote 7.1.1

NeedsCompilation no

Author Robrecht Cannoodt [aut, cre, cph] (<https://orcid.org/0000-0003-3641-729X>), Wouter Saelens [aut] (<https://orcid.org/0000-0002-7114-6248>)

Maintainer Robrecht Cannoodt <rcannood@gmail.com>
R topics documented:

as_dyno .................................................. 3
backbone .................................................. 4
bblego ..................................................... 6
combine_models .......................................... 8
dyngen ..................................................... 9
example_model .......................................... 12
generate_cells .......................................... 12
generate_dataset ....................................... 14
generate_experiment .................................... 15
generate_feature_network ......................... 17
generate_gold_standard ............................... 18
generate_kinetics ..................................... 19
generate_tf_network ................................... 21
get_timings ............................................... 22
initialise_model ....................................... 23
kinetics_noise_none .................................. 24
list_backbones ......................................... 25
plot_backbone_modulenet ............................ 27
plot_backbone_statenet ............................... 28
plot_experiment_dimred ................................ 28
plot_feature_network .................................. 29
plot_gold_expression .................................. 30
plot_gold_mappings .................................... 30
plot_gold_simulations ................................ 31
plot_simulations ....................................... 32
plot_simulation_expression ....................... 33
plot_summary .......................................... 34
realcounts ............................................. 34
realnets ............................................... 35
rnorm_bounded ........................................ 35
runif_subrange ....................................... 36
simtime_from_backbone ............................... 36

Index 38
as_dyno

Convert simulation output to different formats.

Description

For use with other packages compatible with dyno, anndata, SingleCellExperiment, or Seurat.

Usage

```r
as_dyno(
  model,
  store_dimred = !is.null(model$simulations$dimred),
  store_cellwise_grn = !is.null(model$experiment$cellwise_grn),
  store_rna_velocity = !is.null(model$experiment$rna_velocity)
)

as_anndata(
  model,
  store_dimred = !is.null(model$simulations$dimred),
  store_cellwise_grn = !is.null(model$experiment$cellwise_grn),
  store_rna_velocity = !is.null(model$experiment$rna_velocity)
)

as_sce(
  model,
  store_dimred = !is.null(model$simulations$dimred),
  store_cellwise_grn = !is.null(model$experiment$cellwise_grn),
  store_rna_velocity = !is.null(model$experiment$rna_velocity)
)

as_seurat(
  model,
  store_dimred = !is.null(model$simulations$dimred),
  store_cellwise_grn = !is.null(model$experiment$cellwise_grn),
  store_rna_velocity = !is.null(model$experiment$rna_velocity)
)

as_list(
  model,
  store_dimred = !is.null(model$simulations$dimred),
  store_cellwise_grn = !is.null(model$experiment$cellwise_grn),
  store_rna_velocity = !is.null(model$experiment$rna_velocity)
)

wrap_dataset(
  model,
  format = c("list", "dyno", "sce", "seurat", "anndata", "none"),
)
store_dimred = !is.null(model$simulations$dimred),
store_cellwise_grn = !is.null(model$experiment$cellwise_grn),
store_rna_velocity = !is.null(model$experiment$rna_velocity)
)

Arguments

model A dyngen output model for which the experiment has been emulated with `generate_experiment()`.
store_dimred Whether or not to store the dimensionality reduction constructed on the true counts.
store_cellwise_grn Whether or not to also store cellwise GRN information.
store_rna_velocity Whether or not to store the log propensity ratios.
format Which output format to use, must be one of 'dyno' (requires `dynwrap`), 'sce' (requires `SingleCellExperiment`), 'seurat' (requires `Seurat`), 'anndata' (requires `anndata`), 'list' or 'none'.

Value

A dataset object.

Examples

data("example_model")
dataset <- wrap_dataset(example_model)

backbone backbone of the simulation model

Description

A module is a group of genes which, to some extent, shows the same expression behaviour. Several modules are connected together such that one or more genes from one module will regulate the expression of another module. By creating chains of modules, a dynamic behaviour in gene regulation can be created.

Usage

backbone(module_info, module_network, expression_patterns)
**Arguments**

**module_info**  
A tibble containing meta information on the modules themselves.
- `module_id` (character): the name of the module
- `basal` (numeric): basal expression level of genes in this module, must be between [0, 1]
- `burn` (logical): whether or not outgoing edges of this module will be active during the burn in phase
- `independence` (numeric): the independence factor between regulators of this module, must be between [0, 1]

**module_network**  
A tibble describing which modules regulate which other modules.
- `from` (character): the regulating module
- `to` (character): the target module
- `effect` (integer): `1L` if the regulating module upregulates the target module, `-1L` if it downregulates
- `strength` (numeric): the strength of the interaction
- `hill` (numeric): hill coefficient, larger than 1 for positive cooperativity, between 0 and 1 for negative cooperativity

**expression_patterns**  
A tibble describing the expected expression pattern changes when a cell is simulated by dyngen. Each row represents one transition between two cell states.
- `from` (character): name of a cell state
- `to` (character): name of a cell state
- `module_progression` (character): differences in module expression between the two states. Example: `"+4,-1|+9|-4"` means the expression of module 4 will go up at the same time as module 1 goes down; afterwards module 9 expression will go up, and afterwards module 4 expression will go down again.
- `start` (logical): Whether or not this from cell state is the start of the trajectory
- `burn` (logical): Whether these cell states are part of the burn in phase. Cells will not get sampled from these cell states.
- `time` (numeric): The duration of an transition.

**Value**  
A dyngen backbone.

**See Also**

dyngen on how to run a dyngen simulation

**Examples**

```r
library(tibble)
backbone <- backbone(
  module_info = tribble(
    ~module_id, ~basal, ~burn, ~independence,
  )
)```
Design your own custom backbone easily

**Description**

You can use the `bblego` functions in order to create custom backbones using various components. Please note that the `bblego` functions currently only allow you to create tree-like backbones.

**Usage**

```r
bblego(..., .list = NULL)

bblego_linear(
  from,
  to,
  type = sample(c("simple", "doublerep1", "doublerep2"), 1),
  num_modules = sample(4:6, 1),
  burn = FALSE
)

bblego_branching(
  from,
  to,
  type = "simple",
  num_steps = 3,
  num_modules = 2 + length(to) * (3 + num_steps),
  burn = FALSE
)

bblego_start(
  to,
  type = sample(c("simple", "doublerep1", "doublerep2"), 1),
  `bblego

module_network = tribble(  
  -from, -to, -effect, -strength, -hill,
  "M1", "M2", 1L, 1, 2,
  "M2", "M3", 1L, 1, 2
),

expression_patterns = tribble(  
  -from, -to, -module_progression, -start, -burn, -time,
  "s0", "s1", "+M1", TRUE, TRUE, 30,
  "s1", "s2", "+M2,+M3", FALSE, FALSE, 80
  }

)
num_modules = sample(4:6, 1)
)

bblego_end(
  from,
  type = sample(c("simple", "doublerep1", "doublerep2"), 1),
  num_modules = sample(4:6, 1)
)

Arguments

..., .list bblego components, either as separate args or as a list.
from The begin state of this component.
to The end state of this component.
type Some components have alternative module regulatory networks.
ibblego_start(), bblego_linear(), bblego_end():
  • "simple": a sequence of modules in which every module upregulates the
    next module.
  • "doublerep1": a sequence of modules in which every module downregu-
    lates the next module, and each module has positive basal expression.
  • "doublerep2": a sequence of modules in which every module upregulates
    the next module, but downregulates the one after that.
  • "flipflop": a sequence of modules in which every module upregulates
    the next module. In addition, the last module upregulates itself and strongly
    downregulates the first module.
ibblego_branching():
  • "simple": a set of \( n \) modules (with \( n = \text{length(to)} \)) which all downregu-
    late one another and upregulate themselves. This causes a branching to
    occur in the trajectory.
num_modules The number of modules this component is allowed to use. Various components
might require a minimum number of components in order to work properly.
burn Whether or not these components are part of the warm-up simulation.
num_steps The number of branching steps to reduce the odds of double positive cells oc-
curring.

Details

A backbone always needs to start with a single bblego_start() state and needs to end with one
or more bblego_end() states. The order of the mentioned states needs to be such that a state is
never specified in the first argument (except for bblego_start()) before having been specified as
the second argument.

Value

A dyngen backbone.
**Examples**

```r
combine_models <- bblego缺点(8, type = "simple", num_modules = 2),
  bblego_linear("A", "B", type = "simple", num_modules = 3),
  bblego_branching("B", c("C", "D"), type = "simple", num_steps = 3),
  bblego_end("C", type = "flipflop", num_modules = 4),
  bblego_end("D", type = "doublerep1", num_modules = 7)
)
```

**Description**

Assume the given models have the exact same feature ids and ran up until the `generate_cells()` step. In addition, the user is expected to run `generate_experiment()` on the combined models.

**Usage**

```r
combine_models(models, duplicate_gold_standard = TRUE)
```

**Arguments**

- `models` A named list of models. The names of the list will be used to prefix the different cellular states in the combined model.
- `duplicate_gold_standard` Whether or not the gold standards of the models are different and should be duplicated and prefixed.

**Details**

See the vignette on simulating batch effects on how to use this function.

**Examples**

```r
data("example_model")
model_ab <- combine_models(list("left" = example_model, "right" = example_model))

# show a dimensionality reduction
plot_simulations(model_ab)
plot_gold_mappings(model_ab, do_facet = FALSE)
```
dyngen

**dyngen: A multi-modal simulator for spearheading single-cell omics analyses**

---

**Description**

A toolkit for generating synthetic single cell data.

**Step 1, initialise dyngen model**

- **initialise_model():** Define and store settings for all following steps. See each of the sections below for more information.
  - Use a predefined backbone:
    - list_backbones()
    - backbone_bifurcating()
    - backbone_bifurcating_converging()
    - backbone_bifurcating_cycle()
    - backbone_bifurcating_loop()
    - backbone_branching()
    - backbone_binary_tree()
    - backbone_consecutive_bifurcating()
    - backbone_trifurcating()
    - backbone_converging()
    - backbone_cycle()
    - backbone_cycle_simple()
    - backbone_linear()
    - backbone_linear_simple()
    - backbone_disconnected()
  - Create a custom backbone:
    - backbone()
    - bblego()
    - bblego_linear()
    - bblego_branching()
    - bblego_start()
    - bblego_end()
  - Visualise the backbone:
    - plot_backbone_modulenet()
    - plot_backbone_statenet()

**Step 2, generate TF network**

- **generate_tf_network():** Generate a transcription factor network from the backbone
- **tf_network_default():** Parameters for configuring this step
Step 3, add more genes to the gene network

- `generate_feature_network()`: Generate a target network
- `feature_network_default()`: Parameters for configuring this step
- `plot_feature_network()`: Visualise the gene network

Step 4, generate gene kinetics

- `generate_kinetics()`: Generate the gene kinetics
- `kinetics_default(), kinetics_random_distributions()`: Parameters for configuring this step

Step 5, simulate the gold standard

- `generate_gold_standard()`: Simulate the gold standard backbone, used for mapping to cell states afterwards
- `gold_standard_default()`: Parameters for configuring this step
- `plot_gold_mappings()`: Visualise the mapping of the simulations to the gold standard
- `plot_gold_simulations()`: Visualise the gold standard simulations using the dimred
- `plot_gold_expression()`: Visualise the expression of the gold standard over simulation time

Step 6, simulate the cells

- `generate_cells()`: Simulate the cells based on its GRN
- `simulation_default()`: Parameters for configuring this step
- `simulation_type_wild_type(), simulation_type_knockdown()`: Used for configuring the type of simulation
- `kinetics_noise_none(), kinetics_noise_simple()`: Different kinetics randomisers to apply to each simulation
- `plot_simulations()`: Visualise the simulations using the dimred
- `plot_simulation_expression()`: Visualise the expression of the simulations over simulation time

Step 7, simulate cell and transcripting sampling

- `generate_experiment()`: Sample cells and transcripts from experiment
- `list_experiment_samplers(), experiment_snapshot(), experiment_synchronised()`: Parameters for configuring this step
- `simtime_from_backbone()`: Determine the simulation time from the backbone
- `plot_experiment_dimred()`: Plot a dimensionality reduction of the final dataset
Step 8, convert to dataset

- `as_dyno()`, `wrap_dataset()`: Convert a dyngen model to a dyno dataset
- `as_anndata()`: Convert a dyngen model to an anndata dataset
- `as_sce()`: Convert a dyngen model to a SingleCellExperiment dataset
- `as_seurat()`: Convert a dyngen model to a Seurat dataset

One-shot function

- `generate_dataset()`: Run through steps 2 to 8 with a single function
- `plot_summary()`: Plot a summary of all dyngen simulation steps

Data objects

- `example_model`: A (very) small toy dyngen model, used for documentation and testing purposes
- `realcounts`: A set of real single-cell expression datasets, to be used as reference datasets
- `realnets`: A set of real gene regulatory networks, to be sampled in step 3

Varia functions

- `dyngen`: This help page
- `get_timings()`: Extract execution timings for each of the dyngen steps
- `combine_models()`: Combine multiple dyngen models
- `rnorm_bounded()`: A bounded version of `rnorm()`
- `runif_subrange()`: A subrange version of `runif()`

Examples

```r
model <- initialise_model(
  backbone = backbone_bifurcating()
)

model <- model %>%
  generate_tf_network() %>%
  generate_feature_network() %>%
  generate_kinetics() %>%
  generate_gold_standard() %>%
  generate_cells() %>%
  generate_experiment()

dataset <- wrap_dataset(model, format = "dyno")
# format can also be set to "sce", "seurat", "anndata" or "list"

# library(dynplot)
# plot_dimred(dataset)
```
example_model  

A (very!) small toy dyngen model

---

**Description**

Used for showcasing examples of functions.

**Usage**

```r
element_model
```

**Format**

An object of class list (inherits from `dyngen::init`) of length 19.

---

**generate_cells**

Simulate the cells

---

**Description**

generate_cells() runs simulations in order to determine the gold standard of the simulations. 
simulation_default() is used to configure parameters pertaining this process.

**Usage**

generate_cells(model)

```r
simulation_default(  
  burn_time = NULL,  
  total_time = NULL,  
  ssa_algorithm = ssa_etl(tau = 30/3600),  
  census_interval = 4,  
  experiment_params = bind_rows(simulation_type_wild_type(num_simulations = 32),  
                              simulation_type_knockdown(num_simulations = 0)),  
  store_reaction_firings = FALSE,  
  store_reaction_propensities = FALSE,  
  compute_cellwise_grn = FALSE,  
  compute_dimred = TRUE,  
  compute_rna_velocity = FALSE,  
  kinetics_noise_function = kinetics_noise_simple(mean = 1, sd = 0.005)  
)

simulation_type_wild_type(  
  num_simulations,  
  seed = sample.int(10 * num_simulations, num_simulations)
```
```r
generate_cells

  )

  simulation_type_knockdown(
    num_simulations,
    timepoint = runif(num_simulations),
    genes = "*",
    num_genes = sample(1:5, num_simulations, replace = TRUE, prob = 0.25^((1:5)),
    multiplier = runif(num_simulations, 0, 1),
    seed = sample.int(10 * num_simulations, num_simulations)
  )

Arguments

  model
    A dyngen intermediary model for which the gold standard been generated with
generate_gold_standard().

  burn_time
    The burn in time of the system, used to determine an initial state vector. If NULL,
    the burn time will be inferred from the backbone.

  total_time
    The total simulation time of the system. If NULL, the simulation time will be
    inferred from the backbone.

  ssa_algorithm
    Which SSA algorithm to use for simulating the cells with GillespieSSA2::ssa()

  census_interval
    A granularity parameter for the outputted simulation.

  experiment_params
    A tibble generated by rbinding multiple calls of simulation_type_wild_type()
    and simulation_type_knockdown().

  store_reaction_firings
    Whether or not to store the number of reaction firings.

  store_reaction_propensities
    Whether or not to store the propensity values of the reactions.

  compute_cellwise_grn
    Whether or not to compute the cellwise GRN activation values.

  compute_dimred
    Whether to perform a dimensionality reduction after simulation.

  compute_rna_velocity
    Whether or not to compute the propensity ratios after simulation.

  kinetics_noise_function
    A function that will generate noise to the kinetics of each simulation. It takes the
    feature_info and feature_network as input parameters, modifies them, and
    returns them as a list. See kinetics_noise_none() and kinetics_noise_simple().

  num_simulations
    The number of simulations to run.

  seed
    A set of seeds for each of the simulations.

  timepoint
    The relative time point of the knockdown

  genes
    Which genes to sample from. "*" for all genes.

  num_genes
    The number of genes to knockdown.

  multiplier
    The strength of the knockdown. Use 0 for a full knockout, 0<x<1 for a knock-
down, and >1 for an overexpression.
```
generate_dataset

Generate a dataset

description

This function contains the complete pipeline for generating a dataset with dyngen. In order to have more control over how the dataset is generated, run each of the steps in this function separately.

Usage

generate_dataset(
  model,
  format = c("list", "dyno", "sce", "seurat", "anndata", "none"),
  output_dir = NULL,
  make_plots = FALSE,
  store_dimred = model$simulation_params$compute_dimred,
  store_cellwise_grn = model$simulation_params$compute_cellwise_grn,
  store_rna_velocity = model$simulation_params$compute_rna_velocity
)
Arguments

- **model**: A dyngen initial model created with `initialise_model()`.
- **format**: Which output format to use, must be one of 'dyno' (requires dynwrap), 'sce' (requires SingleCellExperiment), 'seurat' (requires Seurat), 'anndata' (requires anndata), 'list' or 'none'.
- **output_dir**: If not NULL, then the generated model and dynwrap dataset will be written to files in this directory.
- **make_plots**: Whether or not to generate an overview of the dataset.
- **store_dimred**: Whether or not to store the dimensionality reduction constructed on the true counts.
- **store_cellwise_grn**: Whether or not to also store cellwise GRN information.
- **store_rna_velocity**: Whether or not to store the log propensity ratios.

Value

A list containing a dyngen model (li$model) and a dynwrap dataset (li$dataset).

Examples

```r
model <- initialise_model(
  backbone = backbone_bifurcating()
)

out <- generate_dataset(model, format = "list")

model <- out$model
dataset <- out$dataset

# can also generate other dataset formats:
# out <- generate_dataset(model, format = "dyno")
# out <- generate_dataset(model, format = "sce")
# out <- generate_dataset(model, format = "seurat")
# out <- generate_dataset(model, format = "anndata")
```

Description

`generate_experiment()` samples cells along the different simulations. Two approaches are implemented: sampling from an unsynchronised population of single cells (snapshot) or sampling at multiple time points in a synchronised population (time series).
Usage

```r
generate_experiment(model)

list_experiment_samplers()

eperiment_snapshot(
  realcount = NULL,
  map_reference_cpm = TRUE,
  map_reference_l = TRUE,
  weight_bw = 0.1
)

experiment_synchronised(
  realcount = NULL,
  map_reference_cpm = TRUE,
  map_reference_l = TRUE,
  num_timepoints = 8,
  pct_between = 0.75
)
```

Arguments

- **model**
  A dyngen intermediary model for which the simulations have been run with `generate_cells()`.

- **realcount**
  The name of a dataset in `realcounts`. If `NULL`, a random dataset will be sampled from `realcounts`.

- **map_reference_cpm**
  Whether or not to try to match the CPM distribution to that of a reference dataset.

- **map_reference_l**
  Whether or not to try to match the distribution of the library sizes to that of the reference dataset.

- **weight_bw**
  [snapshot] A bandwidth parameter for determining the distribution of cells along each edge in order to perform weighted sampling.

- **num_timepoints**
  [synchronised] The number of time points used in the experiment.

- **pct_between**
  [synchronised] The percentage of 'unused' simulation time.

Details

- `experiment_snapshot()` samples the cells using the length of each edge in the milestone network as weights. See Supplementary Figure 7A from the dyngen paper for an illustration of how these weights are computed.

- `experiment_synchronised()` samples the cells along the simulation timeline by binning it into `num_timepoints` groups separated by `num_timepoints-1` gaps. See Supplementary Figure 7B from the dyngen paper for an illustration of how the timepoint groups are computed.

Value

A dyngen model.
Examples

\begin{verbatim}
names(list_experiment_samplers())

c <- initialise_model(
   backbone = backbone_bifurcating(),
   experiment = experiment_synchronised()
)

## Not run:
data("example_model")
c <- example_model %>% generate_experiment()

plot_experiment_dimred(c)

## End(Not run)
\end{verbatim}

---

generate_feature_network

*Generate a target network*

Description

generate_feature_network() generates a network of target genes that are regulated by the previously generated TFs, and also a separate network of housekeeping genes (HKs). feature_network_default() is used to configure parameters pertaining this process.

Usage

generate_feature_network(model)

feature_network_default(
   realnet = NULL,
   damping = 0.01,
   target_resampling = Inf,
   max_in_degree = 5
)

Arguments

- **model**: A dyngen intermediary model for which the transcription network has been generated with generate_tf_network()
- **realnet**: The name of a gene regulatory network (GRN) in realnets. If NULL, a random network will be sampled from realnets. Alternatively, a custom GRN can be used by passing a weighted sparse matrix.
- **damping**: A damping factor used for the page rank algorithm used to subsample the realnet.
generate_gold_standard

Simulate the gold standard

Description

generate_gold_standard() runs simulations in order to determine the gold standard of the simulations. gold_standard_default() is used to configure parameters pertaining this process.

Usage

generate_gold_standard(model)

gold_standard_default(
  tau = 30/3600,
  census_interval = 10/60,
  simulate_targets = FALSE
)
generate_kinetics

Arguments

model A dyngen intermediary model for which the kinetics of the feature network has been generated with generate_kinetics().

tau The time step of the ODE algorithm used to generate the gold standard.

census_interval A granularity parameter of the gold standard time steps. Should be larger than or equal to tau.

simulate_targets Also simulate the targets during the gold standard simulation

Value

A dyngen model.

See Also

dyngen on how to run a complete dyngen simulation

Examples

```r
model <- initialise_model(
    backbone = backbone_bifurcating(),
    gold_standard = gold_standard_default(tau = .01, census_interval = 1)
)

data("example_model")
model <- example_model %>% generate_gold_standard()

plot_gold_simulations(model)
plot_gold_mappings(model)
plot_gold_expression(model)
```

generate_kinetics  Determine the kinetics of the feature network

Description

generate_kinetics() samples the kinetics of genes in the feature network for which the kinetics have not yet been defined. kinetics_default() is used to configure parameters pertaining this process. kinetics_random_distributions() will do the same, but the distributions are also randomised.
Usage

generate_kinetics(model)

kinetics_default()

kinetics_random_distributions()

Arguments

model A dyngen intermediary model for which the feature network has been generated with generate_feature_network().

Details

To write different kinetics settings, you need to write three functions with interface function(feature_info, feature_network, cache_dir, verbose). Described below are the default kinetics samplers.

sampler_tfs() mutates the feature_info data frame by adding the following columns:

- transcription_rate: the rate at which pre-mRNAs are transcribed, in pre-mRNA / hour. Default distribution: U(1, 2).
- translation_rate: the rate at which mRNAs are translated into proteins, in protein / mRNA / hour. Default distribution: U(100, 150).
- mrna_halflife: the half-life of (pre-)mRNA molecules, in hours. Default distribution: U(2.5, 5).
- protein_halflife: the half-life of proteins, in hours. Default distribution: U(5, 10).
- splicing_rate: the rate at which pre-mRNAs are spliced into mRNAs, in reactions / hour. Default value: \(\log(2)/(10/60)\), which corresponds to a half-life of 10 minutes.
- independence: the degree to which all regulators need to be bound for transcription to occur (0), or whether transcription can occur if only one of the regulators is bound (1).

sampler_nontfs() samples the transcription_rate, translation_rate, mrna_halflife and protein_halflife from a supplementary file of Schwannhäuser et al., 2011, doi.org/10.1038/nature10098. splicing_rate is by default the same as in sampler_tfs(). independence is sampled from U(0, 1).

sampler_interactions() mutates the feature_network data frame by adding the following columns:

- effect: the effect of the interaction; upregulating = +1, downregulating = -1. By default, sampled from -1, 1 with probabilities .25, .75.
- strength: the strength of the interaction. Default distribution: \(10^U(0, 2)\).
- hill: the hill coefficient. Default distribution: N(2, 2) with a minimum of 1 and a maximum of 10.

Value

A dyngen model.
generate_tf_network

See Also
dyngen on how to run a complete dyngen simulation

Examples

```r
model <-
  initialise_model(
    backbone = backbone_bifurcating(),
    kinetics_params = kinetics_default()
  )

data("example_model")
model <- example_model %>%
  generate_kinetics()
```

---

**generate_tf_network**

Generate a transcription factor network from the backbone

**Description**

generate_tf_network() generates the transcription factors (TFs) that drive the dynamic process a cell undergoes. tf_network_default() is used to configure parameters pertaining this process.

**Usage**

generate_tf_network(model)

tf_network_default(
  min_tfs_per_module = 1L,
  sample_num_regulators = function() 2,
  weighted_sampling = FALSE
)

**Arguments**

- `model` A dyngen initial model created with initialise_model().
- `min_tfs_per_module` The number of TFs to generate per module in the backbone.
- `sample_num_regulators` A function to generate the number of TFs per module each TF will be regulated by.
- `weighted_sampling` When determining what TFs another TF is regulated by, whether to perform weighted sampling (by rank) or not.
get_timings

Value

A dyngen model.

See Also

dyngen on how to run a complete dyngen simulation

Examples

```r
model <-
  initialise_model(
    backbone = backbone_bifurcating()
  )
model <- model %>%
  generate_tf_network()

plot_feature_network(model)
```

data("example_model")
timings <- get_timings(example_model)

get_timings  

**Description**

Return the timings of each of the dyngen steps

**Usage**

```r
get_timings(model)
```

**Arguments**

model  
A dyngen object

**Examples**

```r
data("example_model")
timings <- get_timings(example_model)
```
Initial settings for simulating a dyngen dataset

Usage

```r
initialise_model(
  backbone,
  num_cells = 1000,
  num_tfs = nrow(backbone$module_info),
  num_targets = 100,
  num_hks = 50,
  distance_metric = c("pearson", "spearman", "cosine", "euclidean", "manhattan"),
  tf_network_params = tf_network_default(),
  feature_network_params = feature_network_default(),
  kinetics_params = kinetics_default(),
  gold_standard_params = gold_standard_default(),
  simulation_params = simulation_default(),
  experiment_params = experiment_snapshot(),
  verbose = TRUE,
  download_cache_dir = getOption("dyngen_download_cache_dir"),
  num_cores = getOption("Ncpus") %||% 1L,
  id = NULL
)
```

Arguments

- **backbone**: The gene module configuration that determines the type of dynamic process being simulated. See `list_backbones()` for a full list of different backbones available in this package.
- **num_cells**: The number of cells to sample.
- **num_tfs**: The number of transcription factors (TFs) to generate. TFs are the main drivers of the changes that occur in a cell. TFs are regulated only by other TFs.
- **num_targets**: The number of target genes to generate. Target genes are regulated by TFs and sometimes by other target genes.
- **num_hks**: The number of housekeeping genes (HKs) to generate. HKs are typically highly expressed, and are not regulated by the TFs or targets.
- **distance_metric**: The distance metric to be used to calculate the distance between cells. See `dynutils::calculate_distance()` for a list of possible distance metrics.
- **tf_network_params**: Settings for generating the TF network with `generate_tf_network()`, see `tf_network_default()`.
kinetics_noise_none

feature_network_params

Settings for generating the feature network with `generate_feature_network()`, see `feature_network_default()`.

kinetics_params

Settings for determining the kinetics of the feature network with `generate_kinetics()`, see `kinetics_default()`.

gold_standard_params

Settings pertaining simulating the gold standard with `generate_gold_standard()`, see `gold_standard_default()`.

simulation_params

Settings pertaining the simulation itself with `generate_cells()`, see `simulation_default()`.

experiment_params

Settings related to how the experiment is simulated with `generate_experiment()`, see `experiment_snapshot()` or `experiment_synchronised()`.

verbose

Whether or not to print messages during the simulation.

download_cache_dir

If not NULL, temporary downloaded files will be cached in this directory.

num_cores

Parallelisation parameter for various steps in the pipeline.

id

An identifier for the model.

Value

A dyngen model.

See Also

dyngen on how to run a complete dyngen simulation

Examples

```r
model <- initialise_model(
  backbone = backbone_bifurcating(),
  num_cells = 555,
  verbose = FALSE,
  download_cache_dir = "~/.cache/dyngen"
)
```

---

**kinetics_noise_none**  
Add small noise to the kinetics of each simulation

Description

Add small noise to the kinetics of each simulation
Usage

kinetics_noise_none()

kinetics_noise_simple(mean = 1, sd = 0.005)

Arguments

mean
The mean level of noise (should be 1)

sd
The sd of the noise (should be a relatively small value)

Value

A list of noise generators for the kinetics.

---

list_backbones

List of all predefined backbone models

Description

A module is a group of genes which, to some extent, shows the same expression behaviour. Several modules are connected together such that one or more genes from one module will regulate the expression of another module. By creating chains of modules, a dynamic behaviour in gene regulation can be created.

Usage

list_backbones()

backbone_bifurcating()

backbone_bifurcating_converging()

backbone_bifurcating_cycle()

backbone_bifurcating_loop()

backbone_branching(
    num_modifications = rbinom(1, size = 6, 0.25) + 1,
    min_degree = 3,
    max_degree = sample(min_degree:5, 1)
)

backbone_binary_tree(num_modifications = rbinom(1, size = 6, 0.25) + 1)

backbone_consecutive_bifurcating()
backbone_trifurcating()
backbone_converging()
backbone_cycle()
backbone_cycle_simple()
backbone_linear()
backbone_linear_simple()

backbone_disconnected(
  left_backbone = list_backbones() %>% keep(., names(.) != "disconnected") %>%
    sample(1) %>% first(),
  right_backbone = list_backbones() %>% keep(., names(.) != "disconnected") %>%
    sample(1) %>% first(),
  num_common_modules = 10
)

Arguments

num_modifications
  The number of branch points in the generated backbone.
min_degree
  The minimum degree of each node in the backbone.
max_degree
  The maximum degree of each node in the backbone.
left_backbone
  A backbone (other than a disconnected backbone), see \texttt{list_backbones()}.
right_backbone
  A backbone (other than a disconnected backbone), see \texttt{list_backbones()}.
num_common_modules
  The number of modules which are regulated by either backbone.

Value

A list of all the available backbone generators.

See Also

\texttt{dyngen} on how to run a dyngen simulation

Examples

names(list_backbones())

bb <- backbone_bifurcating()
bb <- backbone_bifurcating_converging()
bb <- backbone_bifurcating_cycle()
bb <- backbone_bifurcating_loop()
bb <- backbone_binary_tree()
bb <- backbone_branching()
```r
bb <- backbone_consecutive_bifurcating()
bb <- backbone_converging()
bb <- backbone_cycle()
bb <- backbone_cycle_simple()
bb <- backbone_disconnected()
bb <- backbone_linear()
bb <- backbone_linear_simple()
bb <- backbone_trifurcating()

model <- initialise_model(
  backbone = bb
)
```

---

**plot_backbone_modulenet**

*Visualise the backbone of a model*

**Description**

Visualise the backbone of a model

**Usage**

```r
plot_backbone_modulenet(model)
```

**Arguments**

- `model`: A dyngen initial model created with `initialise_model()`.

**Value**

A ggplot2 object.

**Examples**

```r
data("example_model")
plot_backbone_modulenet(example_model)
```
plot_backbone_statenet

Visualise the backbone state network of a model

Description
Visualise the backbone state network of a model

Usage
plot_backbone_statenet(model, detailed = FALSE)

Arguments
- model: A dyngen initial model created with initialise_model()
- detailed: Whether or not to also plot the substates of transitions.

Value
A ggplot2 object.

Examples

```r
data("example_model")
plot_backbone_statenet(example_model)
```

plot_experiment_dimred

Plot a dimensionality reduction of the final dataset

Description
Plot a dimensionality reduction of the final dataset

Usage
plot_experiment_dimred(model, mapping = aes_string("comp_1", "comp_2"))

Arguments
- model: A dyngen intermediary model for which the simulations have been run with generate_experiment()
- mapping: Which components to plot.
Value

A ggplot2 object.

Examples

data("example_model")
plot_experiment_dimred(example_model)

plot_feature_network

Visualise the feature network of a model

Description

Visualise the feature network of a model

Usage

plot_feature_network(
  model,
  show_tfs = TRUE,
  show_targets = TRUE,
  show_hks = FALSE
)

Arguments

model A dyngen intermediary model for which the feature network has been generated with \texttt{generate_feature_network()}. show_tfs Whether or not to show the transcription factors. show_targets Whether or not to show the targets. show_hks Whether or not to show the housekeeping genes.

Value

A ggplot2 object.

Examples

data("example_model")
plot_feature_network(example_model)
plot_gold_expression  Visualise the expression of the gold standard over simulation time

Description
Visualise the expression of the gold standard over simulation time

Usage
plot_gold_expression(
  model,
  what = c("mol_premrna", "mol_mrna", "mol_protein"),
  label_changing = TRUE
)

Arguments
model  A dyngen intermediary model for which the simulations have been run with generate_gold_standard().
what    Which molecule types to visualise.
label_changing  Whether or not to add a label next to changing molecules.

Value
A ggplot2 object.

Examples
data("example_model")
plot_gold_expression(example_model, what = "mol_mrna", label_changing = FALSE)

plot_gold_mappings  Visualise the mapping of the simulations to the gold standard

Description
Visualise the mapping of the simulations to the gold standard

Usage
plot_gold_mappings(
  model,
  selected_simulations = NULL,
  do_facet = TRUE,
  mapping = aes_string("comp_1", "comp_2")
)
Arguments

- **model**: A dyngen intermediary model for which the simulations have been run with `generate_cells()`.
- **selected_simulations**: Which simulation indices to visualise.
- **do_facet**: Whether or not to facet according to simulation index.
- **mapping**: Which components to plot.

Value

A ggplot2 object.

Examples

```r
data("example_model")
plot_gold_mappings(example_model)
```

Description

Visualise the simulations using the dimred

Usage

```r
plot_gold_simulations(
  model, 
  detailed = FALSE, 
  mapping = aes_string("comp_1", "comp_2"), 
  highlight = 0
)
```

Arguments

- **model**: A dyngen intermediary model for which the simulations have been run with `generate_cells()`.
- **detailed**: Whether or not to colour according to each separate sub-edge in the gold standard.
- **mapping**: Which components to plot.
- **highlight**: Which simulation to highlight. If `highlight == 0` then the gold simulation will be highlighted.
plot_simulations

Value

A ggplot2 object.

Examples

```r
data("example_model")
plot_simulations(example_model)
```

Description

Visualise the simulations using the dimred

Usage

```r
plot_simulations(model, mapping = aes_string("comp_1", "comp_2"))
```

Arguments

- `model`: A dyngen intermediary model for which the simulations have been run with `generate_cells()`.
- `mapping`: Which components to plot.

Value

A ggplot2 object.

Examples

```r
data("example_model")
plot_simulations(example_model)
```
plot_simulation_expression

Visualise the expression of the simulations over simulation time

Description

Visualise the expression of the simulations over simulation time

Usage

plot_simulation_expression(
  model,
  simulation_i = 1:4,
  what = c("mol_premrna", "mol_mrna", "mol_protein"),
  facet = c("simulation", "module_group", "module_id", "none"),
  label_nonzero = FALSE
)

Arguments

  model         A dyngen intermediary model for which the simulations have been run with generate_cells().
  simulation_i  Which simulation to visualise.
  what          Which molecule types to visualise.
  facet         What to facet on.
  label_nonzero Plot labels for non-zero molecules.

Value

  A ggplot2 object.

Examples

data("example_model")
plot_simulation_expression(example_model)
Plot a summary of all dyngen simulation steps.

**Description**
Plot a summary of all dyngen simulation steps.

**Usage**

```
plot_summary(model)
```

**Arguments**

- `model` A dyngen intermediary model for which the simulations have been run with `generate_experiment()`.

**Value**
A ggplot2 object.

**Examples**

```
data("example_model")
plot_summary(example_model)
```

---

*A set of real single cell expression datasets*

**Description**
Statistics are derived from these datasets in order to simulate single cell experiments.

**Usage**

```
realcounts
```

**Format**
An object of class tbl_df (inherits from tbl.data.frame) with 111 rows and 9 columns.
realnets

A set of gold standard gene regulatory networks

Description

These networks are subsampled in order to generate realistic feature and housekeeping networks.

Usage

realnets

Format

An object of class tbl_df (inherits from tbl, data.frame) with 32 rows and 2 columns.

rnorm_bounded

A bounded version of rnorm

Description

A bounded version of rnorm

Usage

rnorm_bounded(n, mean = 0, sd = 1, min = -Inf, max = Inf)

Arguments

n: number of observations. If length(n) > 1, the length is taken to be the number required.
mean: vector of means.
sd: vector of standard deviations.
min: lower limits of the distribution.
max: upper limits of the distribution.

Value

Generates values with rnorm, bounded by [min, max]

Examples

rnorm_bounded(10)
runif_subrange  
A subrange version of runif

Description
Will generate numbers from a random subrange within the given range. For example, if \([\text{min}, \text{max}]\) is set to \([0, 10]\), this function could decide to generate numbers between 2 and 6.

Usage
runif_subrange(n, min, max)

Arguments
- n: Number of observations
- min: Lower limits of the distribution.
- max: Upper limits of the distribution.

Value
Generates values with runif, bounded by a range drawn from \(\text{sort}(\text{runif}(2, \text{min}, \text{max}))\).

Examples
runif_subrange(20, 0, 10)

---

simtime_from_backbone  
Determine simulation time from backbone

Description
Determine simulation time from backbone

Usage
simtime_from_backbone(backbone, burn = FALSE)

Arguments
- backbone: A valid dyngen backbone object
- burn: Whether or not to compute the simtime for only the burn phase

Value
An estimation of the required simulation time
Examples

```r
backbone <- backbone_linear()

simtime_from_backbone(backbone)

model <- initialise_model(
  backbone = backbone,
  simulation_params = simulation_default(
    burn_time = simtime_from_backbone(backbone, burn = TRUE),
    total_time = simtime_from_backbone(backbone, burn = FALSE)
  )
)
```
Index

* datasets
  example_model, 12
  realcounts, 34
  realnets, 35
as_anndata (as_dyno), 3
as_anndata(), 11
as_dyno, 3
as_dyno(), 11
as_list (as_dyno), 3
as_sce (as_dyno), 3
as_sce(), 11
as_seurat (as_dyno), 3
as_seurat(), 11
backbone, 4
backbone(), 9
backbone_bifurcating (list_backbones), 25
backbone_bifurcating(), 9
backbone_bifurcating_converging (list_backbones), 25
backbone_bifurcating_converging(), 9
backbone_bifurcating_cycle (list_backbones), 25
backbone_bifurcating_cycle(), 9
backbone_bifurcating_loop (list_backbones), 25
backbone_bifurcating_loop(), 9
backbone_binary_tree (list_backbones), 25
backbone_binary_tree(), 9
backbone_branching (list_backbones), 25
backbone_branching(), 9
backbone_consecutive_bifurcating (list_backbones), 25
backbone_consecutive_bifurcating(), 9
backbone_converging (list_backbones), 25
backbone_converging(), 9
backbone_cycle (list_backbones), 25
backbone_cycle(), 9
backbone_cycle_simple (list_backbones), 25
backbone_cycle_simple(), 9
backbone_disconnected (list_backbones), 25
backbone_disconnected(), 9
backbone_linear (list_backbones), 25
backbone_linear(), 9
backbone_linear_simple (list_backbones), 25
backbone_linear_simple(), 9
backbone_trifurcating (list_backbones), 25
backbone_trifurcating(), 9
bblego, 6
bblego(), 9
bblego_branching (bblego), 6
bblego_branching(), 9
bblego_end (bblego), 6
bblego_end(), 9
bblego_linear (bblego), 6
bblego_linear(), 9
bblego_start (bblego), 6
bblego_start(), 9
combine_models, 8
combine_models(), 11
dyngen, 5, 9, 11, 14, 18, 19, 21, 22, 24, 26
dynutils::calculate_distance(), 23
example_model, 11, 12
experiment_snapshot (generate_experiment), 15
experiment_snapshot(), 10, 16, 24
experiment_synchronised (generate_experiment), 15
experiment_synchronised(), 10, 16, 24

38
feature_network_default (generate_feature_network), 17
feature_network_default(), 10, 17, 24

generate_cells, 12
generate_cells(), 10, 12, 16, 24, 31-33
generate_dataset, 14
generate_dataset(), 11
generate_experiment, 15
generate_experiment(), 4, 10, 15, 24, 28, 34
generate_feature_network, 17
generate_feature_network(), 10, 17, 20, 24, 29
generate_gold_standard, 18
generate_gold_standard(), 10, 13, 18, 24, 30
generate_kinetics, 19
generate_kinetics(), 10, 19, 24
generate_tf_network, 21
generate_tf_network(), 9, 17, 21, 23
get_timings, 22
get_timings(), 11
GillespieSSA2::ssa, 13
gold_standard_default (generate_gold_standard), 18
gold_standard_default(), 10, 18, 24

initialise_model, 23
initialise_model(), 9, 15, 21, 27, 28

kinetics_default (generate_kinetics), 19
kinetics_default(), 10, 19, 24
kinetics_noise_none, 24
kinetics_noise_none(), 10, 13
kinetics_noise_simple (kinetics_noise_none), 24
kinetics_noise_simple(), 10, 13
kinetics_random_distributions (generate_kinetics), 19
kinetics_random_distributions(), 10, 19

list_backbones, 25
list_backbones(), 9, 23, 26
list_experiment_samplers (generate_experiment), 15
list_experiment_samplers(), 10

plot_backbone_modulenet, 27
plot_backbone_modulenet(), 9
plot_backbone_statenet, 28
plot_backbone_statenet(), 9
plot_experiment_dimred, 28
plot_experiment_dimred(), 10
plot_feature_network, 29
plot_feature_network(), 10
plot_gold_expression, 30
plot_gold_expression(), 10
plot_gold_mappings, 30
plot_gold_mappings(), 10
plot_gold_simulations, 31
plot_gold_simulations(), 10
plot_simulation_expression, 33
plot_simulation_expression(), 10
plot_simulations, 32
plot_simulations(), 10
plot_summary, 34
plot_summary(), 11

realcounts, 11, 16, 34
realnets, 11, 17, 35
rnorm(), 11
rnorm_bounded, 35
rnorm_bounded(), 11
runif(), 11
runif_subrange, 36
runif_subrange(), 11

simtime_from_backbone, 36
simtime_from_backbone(), 10
simulation_default (generate_cells), 12
simulation_default(), 10, 12, 24
simulation_type_knockdown (generate_cells), 12
simulation_type_knockdown(), 10, 13
simulation_type_wild_type (generate_cells), 12
simulation_type_wild_type(), 10, 13

tf_network_default (generate_tf_network), 21
tf_network_default(), 9, 21, 23

wrap_dataset (as_dyno), 3
wrap_dataset(), 11