Package: simulator (via r-universe)

August 24, 2024

Title An Engine for Running Simulations

Version 0.2.4

Date 2023-02-01

Description A framework for performing simulations such as those common in methodological statistics papers. The design principles of this package are described in greater depth in Bien, J. (2016) ``The simulator: An Engine to Streamline Simulations,'' which is available at <arXiv:1607.00021>.

Depends R (>= 3.5.0)

Imports methods, graphics, grDevices, stats, utils, magrittr

Suggests digest, ggplot2, knitr, rmarkdown, testthat, glmnet, mvtnorm

License GPL-3

LazyData true

VignetteBuilder knitr

URL `https://github.com/jacobbien/simulator`

BugReports `https://github.com/jacobbien/simulator/issues`

Encoding UTF-8

Collate 'evals-class.R' 'output-class.R' 'component-class.R' 'model-class.R' 'draws-class.R' 'reference-classes.R' 'simulation-class.R' 'add-to-simulation.R' 'aggregator-class.R' 'utils.R' 'create.R' 'draws.R' 'evaluate.R' 'metric-class.R' 'method-class.R' 'examples.R' 'extended-method-class.R' 'get-from-simulation.R' 'import_from.R' 'load.R' 'manage.R' 'method-extension-class.R' 'methods.R' 'models.R' 'parallel-draws.R' 'parallel-methods.R' 'parallel.R' 'plot_eval.R' 'plot_eval_by.R' 'plot_evals.R' 'tables.R' 'zzz.R'

RoxygenNote 7.2.3

Repository https://jacobbien.r-universe.dev

RemoteUrl https://github.com/jacobbien/simulator

RemoteRef HEAD

RemoteSha 3d325ec78d48c57cbdc16d1455c2ffcc85ae8bb1

Contents

`+,ExtendedMethod,MethodExtension-method`
Create an ExtendedMethod from an ExtendedMethod and MethodExtension

---

Description

Create an ExtendedMethod from an ExtendedMethod and MethodExtension

Usage

```
## S4 method for signature 'ExtendedMethod,MethodExtension'
e1 + e2
```

Arguments

e1	an object of class ExtendedMethod
e2	an object of class MethodExtension

+,list,MethodExtension-method

Create a list of ExtendedMethod from a list of Methods and a Method-Extension

Description

Create a list of ExtendedMethod from a list of Methods and a MethodExtension

Usage

```
## S4 method for signature 'list,MethodExtension'
e1 + e2
```

Arguments

e1	a list of objects of class Method or of class ExtendedMethod
e2	an object of class MethodExtension

+,Method,MethodExtension-method

Create an ExtendedMethod from a Method and MethodExtension

Description

Create an ExtendedMethod from a Method and MethodExtension

Usage

```
## S4 method for signature 'Method,MethodExtension'
e1 + e2
```

Arguments

e1	an object of class Method
e2	an object of class MethodExtension

add *Add a reference to a simulation*

Description

Adds a ModelRef, DrawsRef, OutputRef, or EvalsRef to a simulation object. To add a DrawsRef, the corresponding ModelRef must already be added. Likewise, to add an OutputRef, the corresponding DrawsRef must already be added. And to add an EvalsRef, the corresponding OutputRef must be added. One can also pass a list of such objects.

Usage

```
add(sim, ref, ...)

## S4 method for signature 'Simulation,ModelRef'
add(sim, ref, update_saved = TRUE)

## S4 method for signature 'Simulation,DrawsRef'
add(sim, ref, update_saved = TRUE)

## S4 method for signature 'Simulation,OutputRef'
add(sim, ref, update_saved = TRUE)

## S4 method for signature 'Simulation,EvalsRef'
add(sim, ref, update_saved = TRUE)

## S4 method for signature 'Simulation,list'
add(sim, ref, update_saved = TRUE)
```

Arguments

`sim`	simulation being added to
`ref`	the reference object being added
`...`	not used
`update_saved`	default is TRUE. Determines whether change to simulation object should be saved to file

Details

The modified simulation object is saved to file if `update_saved` is TRUE.

`add_bold` *Make a string bold in a certain format*

Description

For example, in latex it would take "2" and output "\bf 2"; in html it would output "<b>2</b>".

Usage

```
add_bold(str, output_type)
```

Arguments

`str`	string or strings (character) to make bold
`output_type`	output type (see knitr::kable's format)

`aggregate_evals` *Apply aggregator to a list of Evals objects*

Description

Returns a num_models by num_methods matrix

Usage

```
aggregate_evals(evals_list, aggregator)
```

Arguments

`evals_list`	a list of Evals objects
`aggregator`	object of class Aggregator

`Aggregator-class` *An S4 class for aggregating evaluated metrics*

Description

An object of class `Aggregator` consists of a label and a function `aggregate` that has a single argument ev that is a list of length equal to the number of draws. This list consists of the evaluated values of all metrics on a single method for a single model.

Slots

`label` a human readable label that will be a prefix to the Eval's label

`aggregate` a function with argument `ev` that is a list of length `nsim` and returns a scalar.

`as.data.frame.Evals` *Convert an Evals to a data.frame*

Description

This is equivalent to calling `as(x, "data.frame")`

Usage

```
## S3 method for class 'Evals'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`	object of class `Evals`
`row.names`	not used
`optional`	not used
`...`	not used

`as.data.frame.listofEvals` *Convert a list of Evals to a data.frame*

Description

When `load` generates a list of Evals, it assigns this to be of (S3) class listofEvals, inherited from list, so that this function will be invoked instead of as.data.frame.list, which is defined in base.

Usage

```
## S3 method for class 'listofEvals'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`	a listofEvals object
`row.names`	not used
`optional`	not used
`...`	not used

`as.data.frame.listofModels` *Convert a List of Models to a data.frame*

Description

When `load` generates a list of Models, it assigns this to be of (S3) class listofModels, inherited from list, so that this function will be invoked instead of as.data.frame.list, which is defined in base.

Usage

```
## S3 method for class 'listofModels'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`	list
`row.names`	not used
`optional`	not used
`...`	not used

`as.data.frame.Model` *Convert a Model to a data.frame*

Description

Ignores any params that are not length 1 and numeric or character. This is equivalent to calling `as(x, "data.frame")`

Usage

```
## S3 method for class 'Model'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`	object of class `Model`
`row.names`	not used
`optional`	not used
`...`	not used

`catsim` *Concatenate and print for the simulator*

Description

For internal use. This calls `cat` only when `getOption("simulator.verbose")`.

Usage

```
catsim(...)
```

Arguments

`...`	arguments to be passed to `cat`

`Component-class` *An S4 class representing a component of the simulator.*

Description

This is a virtual class.

Slots

`name` a short name identifier. Must be alphanumeric.

`label` a longer, human readable label that can have other characters such as spaces, hyphens, etc.

`create` *Create template for a new set of simulations*

Description

This function is the fastest way to get started. Creates the skeleton of a simulation.

Usage

```
create(dir = "./my_sims")
```

Arguments

`dir`	where to create the skeleton of a new set of simulations

Examples

```
## Not run:
 create("./examples")

## End(Not run)
```

---

describe — *Describe the contents of a simulator directory*

Description

Describe the contents of a simulator directory

Usage

```
describe(dir = ".")
```

Arguments

dir	name of the directory where directory named "files" exists

---

draws — *Get one or more draws from a simulation*

Description

Returns either the draws objects themselves or references to them. See `model` function for more information on the `...` and `subset` arguments, which are used to specify a subset of the models.

Usage

```
draws(sim, ..., subset = NULL, index, reference = FALSE)
```

Arguments

sim	a simulation object
...	logical conditions to specify a subset of models. Conditions can only involve params of model that have length 1 and are of class numeric or character.
subset	a vector of integers indexing the models or a vector of model names. To select models based on parameter values, use `...`. However, using `...` is slower than using subset.
index	a vector of positive integers specifying which draws objects are desired. If missing, then all draws' outputs are returned.
reference	whether to return the ModelRef or the Model object itself

Examples

```
## Not run:
 # suppose previously we had run the following:
 sim <- new_simulation(name = "normal-example",
                       label = "Normal Mean Estimation",
                       dir = tempdir()) %>%
   generate_model(make_my_example_model, n = 20) %>%
   simulate_from_model(nsim = 50, index = 1:3)
 # then we could get the simulated draws as follows:
 d <- draws(sim)
 d@draws$r1.1 # first random draw

## End(Not run)
```

`Draws-class` *An S4 class representing the random draws from a Model object.*

Description

An object of class Draws represents the randomly drawn simulated data that is generated when `simulate_from_model` is called on an object of class Model. In particular, it contains a named list of `nsim` simulated draws from a model object. The Model object's `simulate` function populates this list.

Details

This class inherits from the `Component` class.

Slots

`name` a short name identifier. Must be alphanumeric. Should use the name of the Model object that generated it.

`label` a longer, human readable label that indicates what has been randomly drawn.

`draws` a list with `nsim` elements as created by calling the `simulate` function of a Model object. This is a named list with each element labeled as `ri.j` where `i` is the `index` and `j` ranges from 1 to `nsim`. The names are assigned by `simulate_from_model`.

`index` an integer-valued numeric that indicates which block of random draws this is

`DrawsRef-class` *An S4 class representing a reference to an object of class Draws.*

Description

This identifies the necessary information to locate a saved object of class `Draws`.

Slots

`dir` directory where the directory `getOption("simulator.files")` is that contains the referenced `Model` object

`model_name` name of the referenced `Model` object

`index` the index of the referenced `Draws` object. Can alternately be a vector of such indices.

`simulator.files` simulator functions will use `getOption("simulator.files")` if simulator.files not provided.

`evals` *Get one or more evals from a simulation*

Description

Returns either the Evals object itself or a reference to it.

Usage

```
evals(sim, ..., subset = NULL, index, methods, reference = FALSE)
```

Arguments

`sim`	a simulation object
`...`	logical conditions to specify a subset of models. Conditions can only involve params of model that have length 1 and are of class numeric or character.
`subset`	a vector of integers indexing the models or a vector of model names. To select models based on parameter values, use `...`. However, using `...` is slower than using subset.
`index`	a vector of positive integers specifying which draws' objects are desired. If missing, then all draws' evals are returned.
`methods`	character vector of method names of interest. If missing, then all methods' evals are returned
`reference`	whether to return the ModelRef or the Model object itself

See Also

`as.data.frame`

Examples

```
## Not run:
 # suppose previously we had run the following:
 sim <- new_simulation(name = "normal-example",
                       label = "Normal Mean Estimation",
                       dir = tempdir()) %>%
   generate_model(make_my_example_model, n = 20) %>%
   simulate_from_model(nsim = 50, index = 1:3) %>%
   run_method(my_example_method) %>%
   evaluate(my_example_loss)
 # then we could get the metric evaluated on the method's output:
 e <- evals(sim)
 # we can export it as a data.frame
 as.data.frame(e)
 # or we can get at a particular draw-method-metric triplet
 e@evals$`my-method`$r1.1$myloss

## End(Not run)
```

`Evals-class` *An S4 class representing the evaluation of a metric run by simulator.*

Description

An object of class `Evals` consists of information to identify the model, draws, method, and metric objects this output was derived from. It also has a list called `evals`, which is where the output of the metric is stored. Currently, the labels of all these objects are also included so that plot functions can use human-readable labels without requiring re-loading these.

Slots

`model_name` the name of the `Model` object this output is derived from.

`model_label` the label of the `Model` object this output is derived from.

`index` the index of the `Draws` object this output is derived from.

`method_name` the name of the `Method` object this output is derived from.

`method_label` the label of the `Method` object this output is derived from.

`metric_name` the name of the `Metric` object this output is derived from.

`metric_label` the label of the `Metric` object this output is derived from.

`evals` a named list with each element labeled by a method_name each evals[[m]] is itself a named list with each element labeled as `ri.j` where `i` is the `index` and `j` ranges from `1` to `nsim`. Element `out$ri.j` is output of metric `metric_name` on random draw `ri.j`.

See Also

`evaluate` `as.data.frame.Evals`

`EvalsRef-class` *An S4 class representing a reference to an object of class Evals*

Description

This identifies the necessary information to locate a saved object of class `Evals`. Note that `metric_names` is not needed to identify an Evals object since Evals objects combine all metrics together into a single file and object.

Slots

`dir` directory where the directory `getOption("simulator.files")` is that contains the referenced `Model` object

`model_name` name of the referenced `Model` object

`index` the index of the referenced `Draws` object.

`method_name` the name of the `Method` object this output is derived from.

`out_loc` a length-1 character vector that gives location (relative to model's path) that method outputs are stored.This can be useful for staying organized when multiple simulations are based on the same Model and Draws objects.

`simulator.files` simulator functions will use `getOption("simulator.files")` if simulator.files not provided.

`evaluate` *Evaluate outputs of methods according to provided metrics.*

Description

Given a `Metric` object or list of `Metric` objects, this function evaluates an `Output` object according to these metrics. The computed values of the metrics are saved to file. The "user" time to run the method (as measured by `system.time`) is added to `metrics` by default unless one of the passed metrics has name "time".

Usage

```
evaluate(object, metrics)
```

Arguments

`object`	object of class `OutputRef` as produced by `run_method` (or list of such objects). If `object` is a `Simulation`, then function is applied to the referenced outputs in that simulation and returns the same `Simulation` object but with references added to the new evals created.
`metrics`	a list of `Metric` objects or a single `Metric` object.

Details

This function creates objects of class Evals and saves each to file (at dir/model_name/<out_loc>/r<index>_<method_name>_
Since evaluating metrics is usually (in statistical methodological papers) fast, parallel functionality has not been developed for the evaluation component.

See Also

generate_model simulate_from_model run_method

Examples

```
## Not run:
 # suppose previously we had run the following:
 sim <- new_simulation(name = "normal-example",
                       label = "Normal Mean Estimation",
                       dir = tempdir()) %>%
   generate_model(make_my_example_model, n = 20) %>%
   simulate_from_model(nsim = 50, index = 1:3) %>%
   run_method(my_example_method)
 # then we could add
 sim <- evaluate(sim, my_example_loss)

## End(Not run)
```

---

evaluate_internal *Evaluate outputs of methods according to provided metrics.*

Description

Given a Metric object or list of Metric objects, this function evaluates an Output object according to these metrics. The computed values of the metrics are saved to file.

Usage

```
evaluate_internal(
  metrics,
  dir = ".",
  model_name,
  index,
  method_names,
  out_loc = "out"
)
```

Arguments

metrics	a list of Metric objects or a single Metric object
dir	the directory where Model object was saved (by generate_model)

`model_name`	the `Model` object's `name` attribute
`index`	the index of a computed `Draws` object. Can alternately be a vector of such indices.
`method_names`	the `Method` objects' `name` attributes as a character vector.
`out_loc`	(optional) a length-1 character vector that gives location (relative to model's path) that method outputs are stored.

Details

This function creates objects of class `Evals` and saves each to file (at dir/model_name/<out_loc>/r<index>_<method_name>_ Since evaluating metrics is usually (in statistical methodological papers) fast, parallel functionality has not been developed for the evaluation component.

`evaluate_single` *Run one or more metrics on outputs.*

Description

This is an internal function. Users should call the wrapper function `evaluate`. Here "single" refers to a single output (and thus a single method, though not necessarily a single index). The metrics provided are run and saved together in a file.

Usage

```
evaluate_single(metrics, model, output, draws = NULL)
```

Arguments

`metrics`	a list of `Metric` objects
`model`	a `Model` object
`output`	a `Output` object
`draws`	(optional) a `Draws` object or NULL

`ExtendedMethod-class` *An S4 class representing the extension of a method*

Description

An object of class `ExtendedMethod` is like a `Method` except it uses the output of another method in addition to the `Model` and `Draws`. We can also form chains of `ExtendedMethod`'s, in which one `ExtendedMethod` is taken to be the "base_method" of a subsequent `ExtendedMethod`. This means that the latter `ExtendedMethod` would use the output of the former `ExtendedMethod`.

Details

While one can create an `ExtendedMethod` from scratch, typically it will be cleaner to write a `MethodExtension` object and then use the addition operator: `my_extended_method = my_base_method + my_method_extension`. For example, if `my_base_method` is the lasso, `my_method_extension` might be cross-validation, and the resulting `my_extended_method` would be the lasso with tuning parameter chosen by cross-validation. The advantage is that if we have several methods, we only have to write the cross-validation `MethodExtension` object once.

For an example in which one has a chain of `ExtendedMethod`'s, consider the lasso example in which we have a `MethodExtension` called, say, `refit`, which takes the nonzeros from the lasso's output and performs least squares on these selected variables. Let `cv` be another `MethodExtension`. Then, `refitted_lasso = lasso + refit` is an `ExtendedMethod` and `refitted_lasso + cv` is as well.

This class inherits from the `Component` class.

Slots

`name` a short name identifier. Must be alphanumeric.

`label` a longer, human readable label that can have other characters such as spaces, hyphens, etc.

`base_method` a list of length 1 containing the object of class `Method` or `ExtendedMethod` that is being extended

`extended_method` a function with arguments "model", "draw", "out", and "base_method".

`generate_model` *Generate a model.*

Description

This function executes the `make_model` function provided by the user and writes to file the resulting `Model` object(s). For example, when simulating regression with a fixed design, `X` would be generated in this function and `n`, `p`, `beta`, and `sigma` would also be specified.

Usage

```
generate_model(object = ".", make_model, ..., seed = 123, vary_along = NULL)
```

Arguments

`object`	the name of the directory where directory named "files" exists (or should be created) to save `Model` object in. Default is current working directory. Or can be an object of class `Simulation`, in which case the `object@dir` is used and a simulation object is returned instead of an object of class `ModelRef`.
`make_model`	a function that outputs an object of class `Model`. Or a list of such functions.
`...`	optional parameters that may be passed to make_model
`seed`	an integer seed for the random number generator.
`vary_along`	character vector with all elements contained in names(...) See description for more details.

Details

When make_model has arguments, these can be passed using .... These will be passed directly to make_model except for any arguments named in vary_along. These arguments should be lists and a separate model will be created for each combination of elements in these lists. For example, if vary_along = c("n", "p"), then we can pass n=as.list(c(50, 100, 150)) and p=as.list(c(10, 100)) and 6 models will be created, one for each pair of n and p. For each pair (n,p), a distinct extension is added to the end of the model name. This extension is generated using a hash function so that different values of the vary_along parameters will lead to different model name extensions. This ensures that if one later decides to add more values of the vary_along parameters, this will not lead to pre-existing files being overwritten (unless the same values of the vary_along combination are used again.

If object is a directory name, the function returns a reference or list of references to the model(s) generated. If object is a Simulation, then function returns the same Simulation object but with references added to the new models created. These changes to the Simulation object are saved to file.

make_model is called generating an object of class Model, called model, which is saved to dir/name/model.Rdata (where name is the name attribute of model). This file also contains the random number generator state and other information such as the function make_model itself and the date when model was created.

See Also

new_model simulate_from_model run_method

Examples

```
# initialize a new simulation
sim <- new_simulation(name = "normal-example",
                      label = "Normal Mean Estimation",
                      dir = tempdir())
# generate a model (and add it to the simulation)
sim <- generate_model(sim, make_my_example_model, n = 20)
# generate a sequence of models (and add them to the simulation)
sim <- generate_model(sim, make_my_example_model,
                      n = list(10, 20, 30),
                      vary_along = "n")
```

---

get_contents *Get the contents of a simulator directory*

Description

This function gives detailed information about what is being stored in the "files" directory. In particular, it gives the complete paths for all the draws, outputs, and evals files. This can be useful in situations in which the draws or outputs files are no longer needed and take up a lot of memory. In such a case a user could delete these files with a command such as system(paste(c("rm", contents$out_files), collapse = " ")). That said, one must be cautious in deleting these files since the simulator generally assumes that earlier stages' files will be available and so deleting these may cause errors. However, if one is essentially finished with a simulation and evaluated metrics have been computed and if the methods' raw outputs are taking up a lot of disk space, then one might consider deleting the out_files (and/or the draws_files).

Usage

```
get_contents(dir = ".", out_loc = "out")
```

Arguments

`dir`	name of the directory where directory named "files" exists
`out_loc`	a length-1 character vector that gives location (relative to model's path) that method outputs are stored.This can be useful for staying organized when multiple simulations are based on the same Model and Draws objects. Usually this is just "out"

`get_files_not_in_simulations` *Find files in simulator directory not referred to by any simulations*

Description

Once one has completed all simulation studies, this function can be called to identify any files that may have been created along the way that are no longer being used in any simulations. It would then be safe to delete these files.

Usage

```
get_files_not_in_simulations(dir, out_loc = "out")
```

Arguments

`dir`	name of the directory where directory named "files" exists
`out_loc`	a length-1 character vector that gives location (relative to model's path) that method outputs are stored.This can be useful for staying organized when multiple simulations are based on the same Model and Draws objects. Usually this is just "out"

`get_model_indices` *Returns indices of a specified subset of sim@model_refs*

Description

See `model` for information about the various formats of subset.

Usage

```
get_model_indices(sim, subset)
```

Arguments

`sim`	a simulation object
`subset`	a vector indicating which models should be returned.

`get_relative_path` *Get relative path*

Description

Given a base path and a specific path, returns a string str such that file.path(base_path, str) is the same location as path.

Usage

```
get_relative_path(base_path, path)
```

Arguments

`base_path`	the base path
`path`	a specific path

`get_simulation_with_all_files` *Returns a simulation object containing references to all files in directory*

Description

Returns a simulation object containing references to all files in directory

Usage

```
get_simulation_with_all_files(dir, out_loc = "out")
```

Arguments

`dir`	name of the directory where directory named "files" exists
`out_loc`	a length-1 character vector that gives location (relative to model's path) that method outputs are stored.This can be useful for staying organized when multiple simulations are based on the same Model and Draws objects. Usually this is just "out"

`load,DrawsRef-method` *Load a DrawsRef*

Description

Load a DrawsRef

Usage

```
## S4 method for signature 'DrawsRef'
load(file)
```

Arguments

`file`	object to load

`load,EvalsRef-method` *Load an EvalsRef*

Description

Load an EvalsRef

Usage

```
## S4 method for signature 'EvalsRef'
load(file)
```

Arguments

`file`	object to load

`load,list-method` *Load a list of reference objects*

Description

Load a list of reference objects

Usage

```
## S4 method for signature 'list'
load(file)
```

Arguments

`file`	list of objects to load

`load,ModelRef-method` *Load a ModelRef*

Description

Load a ModelRef

Usage

```
## S4 method for signature 'ModelRef'
load(file)
```

Arguments

`file`	object to load

`load,OutputRef-method` *Load an OutputRef*

Description

Load an OutputRef

Usage

```
## S4 method for signature 'OutputRef'
load(file)
```

Arguments

`file`	object to load

`load_draws` *Load one or more draws objects from file.*

Description

After `simulate_from_model` has been called, this function can be used to load one or more of the saved `Draws` object(s) (along with RNG information). If multiple indices are provided, these will be combined into a new single `Draws` object. If simulation object is available, it is easier to use the function `draws` to load it.

Usage

```
load_draws(dir, model_name, index, more_info = FALSE, simulator.files = NULL)
```

Arguments

`dir`	the directory passed to `generate_model`)
`model_name`	the Model object's name attribute
`index`	a vector of positive integers.
`more_info`	if TRUE, then returns additional information such as state of RNG after calling `generate_model`
`simulator.files`	if NULL, then `getOption("simulator.files")` will be used.

See Also

`simulate_from_model` `draws`

`load_evals` *Load one or more Evals objects from file.*

Description

After `evaluate` has been called, this function can be used to load one or more of the saved `Evals` object(s). If multiple indices are provided, these will be combined by index into a new single `Evals` object. If multiple methods are provided, a list of `Evals` objects will be returned.

Usage

```
load_evals(
  dir,
  model_name,
  index,
  method_names,
  metric_names = NULL,
  out_loc = "out",
  simulator.files = NULL
)

load_evals_from_ref(ref, metric_names = NULL)
```

Arguments

`dir`	the directory passed to `generate_model`)
`model_name`	the `Model` object's name
`index`	a vector of positive integers.
`method_names`	the name of one or more `Method` objects.
`metric_names`	(optional) a character vector of which elements of evals should be loaded. If NULL, then all elements are loaded.
`out_loc`	only needed if it was used in call to
`simulator.files`	if NULL, then `getOption("simulator.files")` will be used. `run_method`.
`ref`	an object of class `EvalsRef`

See Also

`load_model` `load_draws` `as.data.frame.Evals`

`load_model` *Load a model from file.*

Description

After `generate_model` has been called, this function can be used to load the saved `Model` object (along with the RNG state and other information if desired).

Usage

```
load_model(dir, model_name, more_info = FALSE, simulator.files = NULL)
```

Arguments

`dir`	the directory passed to `generate_model`)
`model_name`	the Model object's name attribute
`more_info`	if TRUE, then returns additional information such as state of RNG after calling `generate_model`
`simulator.files`	if NULL, then `getOption("simulator.files")` will be used.

Details

Depending on `more_info`, either returns `Model` object or a list containing `Model` object and other information. If simulation object is available, it is easier to use the function `model` to load the model.

See Also

`generate_model` `model`

`load_simulation` *Load a simulation object*

Description

Loads an object of class `Simulation`. Note that `dir` gives the directory where the Simulation object is stored. Thus, if the working directory is different from the working directory when the Simulation object was created, then `dir` will be different from the one passed to `new_simulation`.

Usage

```
load_simulation(name, dir = ".")
```

Arguments

`name`	a short name identifier. Must be alphanumeric.
`dir`	directory that contains "files" directory for this simulation

See Also

`new_simulation` `save_simulation`

Examples

```
sim <- new_simulation(name = "normal-example",
                      label = "Normal Mean Estimation",
                      dir = tempdir())
rm(sim)
sim <- load_simulation("normal-example", dir = tempdir())
```

`make_my_example_model` *Make My Example Model*

Description

This function is used in the examples. It returns a `Model` object. In particular, it represents `n` i.i.d. draws from a normal with mean 2 and variance 1.

Usage

```
make_my_example_model(n)
```

Arguments

`n`	number of i.i.d. draws

See Also

`my_example_method` `my_example_loss`

`memory_as_string` *Write memory in human readable way*

Description

Write memory in human readable way

Usage

```
memory_as_string(memory_in_bytes)
```

Arguments

`memory_in_bytes`	the amount of memory in Bytes.

`Method-class` *An S4 class representing a method to be run by simulator.*

Description

An object of class `Method` consists of a name, label, and a function `method` that takes arguments model and draw. A draw refers to a single element of the list in an object of class `Draws`.

Details

This class inherits from the `Component` class.

Slots

`name` a short name identifier. Must be alphanumeric.

`label` a longer, human readable label that can have other characters such as spaces, hyphens, etc.

`settings` (optional) a list of "settings" for the method (e.g., tuning parameters or related information that might distinguish two otherwise identical methods).

`method` a function that has arguments "model", "draw" and (optionally) names matching elements within `names(settings)`

`MethodExtension-class` *An S4 class used to create an extended version of a method*

Description

An object of class `MethodExtension` when added to a `Method` creates a `ExtendedMethod`.

Details

This class inherits from the `Component` class.

Slots

`name` a short name identifier. Must be alphanumeric.

`label` a longer, human readable label that can have other characters such as spaces, hyphens, etc.

`method_extension` a function with arguments "model", "draw", "out", and "base_method". This will become the function `extended_method` in the `ExtendedMethod` object that is created.

`Metric-class` *An S4 class representing an evaluation metric to be used by simulator.*

Description

An object of class `Metric` consists of a name, label, and a function `metric` that takes arguments `model` (of class `Model`) and `out` (of class `Output`), which is the output of a method.

Details

This class inherits from the `Component` class.

Slots

`name` a short name identifier. Must be alphanumeric.

`label` a longer, human readable label that can have other characters such as spaces, hyphens, etc.

`metric` a function with arguments "model" and "out" (and optionally "draw") `model` *Get one or more models from a simulation*

Description

Returns either the models themselves or references to them.

Usage

```
model(sim, ..., subset = NULL, reference = FALSE)
```

Arguments

`sim`	a simulation object
`...`	logical conditions to specify a subset of models. Conditions can only involve params of model that have length 1 and are of class numeric or character.
`subset`	a vector of integers indexing the models or a vector of model names. To select models based on parameter values, use `...`. However, using `...` is slower than using subset.
`reference`	whether to return the ModelRef or the Model object itself

Details

There are two main ways to specify a subset of the models. (1) The easiest way is by writing a conditional expression involving the parameters and passing it through `...`. For example, `n > 100 & p <= 20`. Only parameters that are length one and either numeric or character can be used in these expressions. (2) The faster way to retrieve a subset of models is to use the `subset` argument. This can be either a set of numerical values (specifying which models to load based on the order in which the models are stored in the simulation object. This order can be ascertained by printing the simulation object.) or as a set of a character vector of the model names desired.

While approach (1) is very convenient, it requires loading all models from file. This may be slow in situations in which there are a lot of models and/or the models are large and thus slow to load.

`Model-class` *An S4 class representing the model component of the simulator.*

Description

An object of class Model specifies the statistical model. In particular, all parameters are specified in addition to a function called `simulate` that allows one to draw random samples from this model.

Details

To get parameters stored in a `Model` object, a shortcut for `my_model@params$my_parameter` is `my_model$my_parameter`.

This class inherits from the `Component` class.

Slots

`name` a short name identifier. Must be alphanumeric (though -, _, and / are allowed as long as they are not at the start or end of name.

`label` a longer, human readable label that can have other characters such as spaces, hyphens, etc.

`params` a list that contains the Model object's parameters

`simulate` a function that has arguments `nsim` and names matching elements within `names(params)`. It returns a list of length nsim, where each element of the list represents a random draw from the `Model` object.

`ModelRef-class` *An S4 class representing a reference to an object of class Model.*

Description

This identifies the necessary information to locate a saved object of class `Model`.

Slots

`dir` directory where the directory "files" is that contains the referenced `Model` object

`name` a short name identifier.

`label` a longer, human readable label that can have other characters

`simulator.files` simulator functions will use `getOption("simulator.files")` if simulator.files not provided.

`models_as_data.frame` *Convert a list of Model objects into a data.frame*

Description

Ignores any params that are not length 1 and numeric or character

Usage

```
models_as_data.frame(m)
```

Arguments

`m`	model object

`model_names` *Get model names in a Simulation*

Description

Get model names in a Simulation

Usage

```
model_names(sim)
```

Arguments

`sim`	object of class `Simulation`

`my_example_loss` *My Example Loss*

Description

This `Metric` object is used in the examples. It is squared error loss.

Usage

```
my_example_loss
```

Format

An object of class `Metric` of length 1.

See Also

`make_my_example_model` `my_example_loss`

`my_example_method` *My Example Method*

Description

This `Method` object is used in the examples. It is the sample mean of the data.

Usage

```
my_example_method
```

Format

An object of class `Method` of length 1.

See Also

`make_my_example_model` `my_example_loss`

`new_aggregator` *Create an Aggregator object*

Description

Creates a new `Aggregator` object.

Usage

```
new_aggregator(label, aggregate)
```

Arguments

`label`	a human readable label
`aggregate`	a function with argument `ev` that is a list of length equal to the number of draws with each element itself being a named list. Each element of this list corresponds to a metric that has been computed. In particular, given an `Evals` object `o`, `aggregate` takes as input `o@evals[[method_name]]` (which is a list of the kind just described). The function aggregate should return a scalar.

`new_extended_method` *Create an ExtendedMethod object*

Description

Creates a new `ExtendedMethod` object.

Usage

```
new_extended_method(name, label, base_method, extended_method)
```

Arguments

`name`	a short name identifier. Must be alphanumeric.
`label`	a longer, human readable label that can have other characters such as spaces, hyphens, etc.
`base_method`	the object of class `Method` or of class `Method` that is being extended
`extended_method`	a function with arguments "model", "draw", "out", and "base_method".

`new_method` *Create a Method object*

Description

Creates a new `Method` object.

Usage

```
new_method(name, label, method, settings = list())
```

Arguments

`name`	a short name identifier. Must be alphanumeric.
`label`	a longer, human readable label that can have other characters such as spaces, hyphens, etc.
`method`	a function that has arguments "model", "draw" and (optionally) names matching elements within `names(settings)`
`settings`	(optional) a list of "settings" for the method (e.g., tuning parameters or related information that might distinguish two otherwise identical methods).

`new_method_extension` *Create an object that can be used to make an extended version of a method*

Description

Creates an object of class `MethodExtension`, which when added to a `Method` creates an `ExtendedMethod`.

Usage

```
new_method_extension(name, label, method_extension)
```

Arguments

`name`	a short name identifier. Must be alphanumeric.
`label`	a longer, human readable label that can have other characters such as spaces, hyphens, etc.
`method_extension`	a function with arguments "model", "draw", "out", and "base_method". This will become the function `extended_method` in the `ExtendedMethod` object that is created.

Details

This class inherits from the `Component` class.

`new_metric` *Create a Metric object*

Description

Creates a new `Metric` object.

Usage

```
new_metric(name, label, metric)
```

Arguments

`name`	a short name identifier. Must be alphanumeric.
`label`	a longer, human readable label that can have other characters such as spaces, hyphens, etc.
`metric`	a function with arguments "model" and "out" (and optionally "draw")

`new_model` *Create a Model object*

Description

Creates a new `Model` object.

Usage

```
new_model(name, label, params = list(), simulate)
```

Arguments

`name`	a short name identifier. Must be alphanumeric (though -, _, and / are allowed as long as they are not at the start or end of name.
`label`	a longer, human readable label that can have other characters such as spaces, hyphens, etc.
`params`	a list that contains the Model object's parameters
`simulate`	a function that has arguments `nsim` and names matching elements within `names(params)`. It returns a list of length nsim, where each element of the list represents a random draw from the `Model` object.

Examples

```
make_my_example_model <- function(n) {
  new_model(name = "normal-data",
            label = sprintf("Normal (n = %s)", n),
            params = list(n = n, mu = 2),
            simulate = function(n, mu, nsim) {
              # this function must return a list of length nsim
              x <- matrix(rnorm(n * nsim), n, nsim)
              x <- mu + x # true mean is mu
              return(split(x, col(x))) # make each col its own list element
            })
}
```

`new_simulation` *Make a new simulation object*

Description

Creates an object of class `Simulation`. In addition to having a name and label, this object consists of a set of references to objects of class `ModelRef`, `DrawsRef`, `OutputRef`, and `EvalsRef`.

Usage

```
new_simulation(name, label, dir = ".", refs = list(), save_to_file = TRUE)
```

Arguments

`name`	a short name identifier. Must be alphanumeric.
`label`	a longer, human readable label that can have other characters such as spaces, hyphens, etc.
`dir`	a directory that reference's directories are relative to
`refs`	a list containing objects of class `ModelRef`, `DrawsRef`, `OutputRef`, and `EvalsRef`
`save_to_file`	whether this new simulation should be saved to file. Default is TRUE. If TRUE, then this simulation can be loaded in a new R session using `dir` and `name`.

Details

A Simulation object is the basic unit of a simulation study. Roughly, one can think of it as all the files relevant to a single figure. This might be a single plot or a series of related plots/panels. It could also correspond to a single table. Note that a Simulation object is light-weight even for large simulations because it only stores references to the objects not the objects themselves. The functions `model`, `draws`, `output`, `evals` can be used to load individual objects of a simulation.

The Simulation object created is saved to a file so that it can be loaded in a new R session. The simulation is saved in dir/files/name.Rdata. Note: while "files" is the default, the name of this directory is from getOption("simulator.files"), which is the value of getOption("simulator.files") when the model was created.

See Also

`load_simulation` `save_simulation`

Examples

```
sim <- new_simulation(name = "normal-example",
                      label = "Normal Mean Estimation",
                      dir = tempdir())
```

`output` *Get one or more outputs from a simulation*

Description

Returns either the output object itself or a reference to it.

Usage

```
output(sim, ..., subset = NULL, index, methods, reference = FALSE)
```

Arguments

sim	a simulation object
...	logical conditions to specify a subset of models. Conditions can only involve params of model that have length 1 and are of class numeric or character.
subset	a vector of integers indexing the models or a vector of model names. To select models based on parameter values, use .... However, using ... is slower than using subset.
index	a vector of positive integers specifying which draws' objects are desired. If missing, then all draws' outputs are returned.
methods	character vector of method names of interest. If missing, then all methods' outputs are returned
reference	whether to return the ModelRef or the Model object itself

Examples

```
## Not run:
 # suppose previously we had run the following:
 sim <- new_simulation(name = "normal-example",
                       label = "Normal Mean Estimation",
                       dir = tempdir()) %>%
   generate_model(make_my_example_model, n = 20) %>%
   simulate_from_model(nsim = 50, index = 1:3) %>%
   run_method(my_example_method)
 # then we could get the method's output as follows:
 o <- output(sim)
 o@out$r1.1 # first random draw's output

## End(Not run)
```

Output-class *An S4 class representing the output of a method run by simulator.*

Description

An object of class Output consists of information to identify the model, draws, and method objects this output was derived from. It also has a list called out, which is where the output of the method is stored.

Slots

model_name the name of the Model object this output is derived from.

index the index of the Draws object this output is derived from.

method_name the name of the Method object this output is derived from.

method_label the label of the Method object this output is derived from.

out a named list with each element labeled as ri.j where i is the index and j ranges from 1 to nsim. Element out$ri.j is output of method method_name on random draw ri.j.

`OutputRef-class` *An S4 class representing a reference to an object of class Output.*

Description

This identifies the necessary information to locate a saved object of class `Output`.

Slots

`dir` directory where the directory `getOption("simulator.files")` is that contains the referenced `Model` object

`model_name` name of the referenced `Model` object

`index` the index of the referenced `Draws` object. Can alternately be a vector of such indices.

`method_name` the name of the `Method` object this output is derived from.

`out_loc` a length-1 character vector that gives location (relative to model's path) that method outputs are stored.This can be useful for staying organized when multiple simulations are based on the same Model and Draws objects.

`simulator.files` simulator functions will use `getOption("simulator.files")` if simulator.files not provided.

`plot_eval` *Plot a metric's value for each method*

Description

When the evaluted metric is scalar-valued, this functions makes a boxplot of this metric for each method. When the metric is vector-valued, this function makes a curve with this metric on the y-axis, with one curve for each method (the x-axis is the corresponding entry of that metric's vector). If evals is a `listofEvals`, then each model will be its own plot.

Usage

```
plot_eval(
  object,
  metric_name,
  use_ggplot2 = TRUE,
  main,
  facet_mains,
  ylab,
  ylim,
  include_zero = FALSE,
  angle = 0,
  ...
)
```

Arguments

`object`	an object of class `Simulation`, `Evals`, or `listofEvals`
`metric_name`	the name of a metric to plot
`use_ggplot2`	whether to use `ggplot2` (requires installation of `ggplot2`)
`main`	title of plot. Default is `model_label` when evals is a single Evals.
`facet_mains`	only to be used when evals is a `listofEvals` and should be of the same length. Default will be the model_label for each model.
`ylab`	the y-axis label (default is `metric_label`)
`ylim`	the y-axis limits to use (across all plots)
`include_zero`	whether ylim should include 0. Ignored if ylim is passed explicitly
`angle`	angle of labels (only when `use_ggplot2 = FALSE`)
`...`	additional arguments to pass to `boxplot` (only when `use_ggplot2 = FALSE`).

See Also

`plot_evals` `plot_eval_by` `tabulate_eval`

Examples

```
## Not run:
 # suppose previously we had run the following:
 sim <- new_simulation(name = "normal-example",
                       label = "Normal Mean Estimation",
                       dir = tempdir()) %>%
   generate_model(make_my_example_model, n = 20) %>%
   simulate_from_model(nsim = 50, index = 1:3) %>%
   run_method(my_example_method) %>%
   evaluate(my_example_loss)
   # then we could plot this
   plot_eval(sim, "myloss") # "myloss" is my_example_loss@name

## End(Not run)
```

`plot_evals` *Plot one metric versus another for each method*

Description

This function is used when both evaluated metrics are vector-valued, so a curve is plotted, parametrized by the two metrics. To plot a single metric that is vector-valued, pass NULL for metric_name_x. This behaves similarly to `plot(runif(5))`, in which the x-axis variable is simply `1:5`. If evals is a `listofEvals`, then each model will be its own plot.

Usage

```
plot_evals(
  object,
  metric_name_x,
  metric_name_y,
  use_ggplot2 = TRUE,
  main,
  facet_mains,
  xlab,
  ylab,
  xlim,
  ylim,
  include_zero = FALSE,
  legend_location = "topright",
  method_col = seq(num_methods),
  method_lty = rep(1, num_methods),
  method_lwd = rep(1, num_methods),
  method_pch = rep(NA, num_methods),
  ...
)
```

Arguments

`object`	an object of class `Simulation`, `Evals`, or `listofEvals`
`metric_name_x`	the name of metric to plot on x axis (or NULL)
`metric_name_y`	the name of metric to plot on y axis
`use_ggplot2`	whether to use `ggplot2` (requires installation of `ggplot2`)
`main`	title of plot. Default is `model_label` when evals is a single Evals.
`facet_mains`	only to be used when evals is a `listofEvals` and should be of the same length. Default will be the model_label for each model.
`xlab`	the x-axis label (default is `metric_label_x`)
`ylab`	the y-axis label (default is `metric_label_y`)
`xlim`	the limits of the x-axis
`ylim`	the limits of the y-axis
`include_zero`	whether ylim should include 0. Ignored if ylim is passed explicitly
`legend_location`	location of legend. Set to NULL to remove legend.
`method_col`	color to use for each method
`method_lty`	line style to use for each method
`method_lwd`	line thickness to use for each method
`method_pch`	point style to use for each method (default is that no points, only lines are drawn)
`...`	additional arguments to pass to `boxplot` (only when `use_ggplot2 = FALSE`).

plot_eval_by *Plot a metric across multiple values of a model parameter*

Description

This function is to be used on simulations in which generate_model was called using the vary_along parameter. When this is a single (scalar) numeric parameter, a single plot is created in which the x-axis is this parameter. Eventually, this function should handle one or two categorical variables (in which facets are used) and one categorical combined with one continuous variable.

Usage

```
plot_eval_by(
  sim,
  metric_name,
  varying,
  type = c("aggregated", "raw"),
  center_aggregator = NULL,
  spread_aggregator = NULL,
  use_ggplot2 = TRUE,
  main,
  xlab,
  ylab,
  xlim,
  ylim,
  include_zero = FALSE,
  legend_location = "topright",
  method_col = seq(num_methods),
  method_lty = rep(1, num_methods),
  method_lwd = rep(1, num_methods),
  method_pch = rep(1, num_methods),
  ...
)
```

Arguments

sim	an object of class Simulation
metric_name	the name of a metric to plot (ignored if custom aggregator is provided)
varying	character vector giving the name of a parameter that is varied across the models in evals. For now, this parameter must be numeric and there cannot be multiple models having the same value of this parameter.
type	if "aggregated" then shows line with error bars (line represents center_aggregator and error bars represent spread_aggregator; by default these are sample mean and estimated standard error); if type is "raw" then shows the raw data as points (with smoother overlayed)

`center_aggregator`	ignored if `type` is "raw". When NULL (which is default), the sample mean aggregator is used. User can write specialized aggregators (see definition of class `Aggregator`) as necessary, for example, when the evaluated metric is not scalar-valued.
`spread_aggregator`	ignored if `type` is "raw". When NULL (which is default), the sample mean aggregator is used. User can write specialized aggregators (see definition of class `Aggregator`) as necessary, for example, when the evaluated metric is not scalar-valued. Set `spread_aggregator` to `NA` to hide error bars.
`use_ggplot2`	whether to use `ggplot2` (requires installation of `ggplot2`)
`main`	title of plot.
`xlab`	the x-axis label (default is `varying`)
`ylab`	the y-axis label (default is `metric_label`)
`xlim`	the x-axis limits to use
`ylim`	the y-axis limits to use
`include_zero`	whether ylim should include 0. Ignored if ylim is passed explicitly
`legend_location`	location of legend. Set to NULL to remove legend.
`method_col`	color to use for each method
`method_lty`	line style to use for each method
`method_lwd`	line thickness to use for each method
`method_pch`	point style to use for each method (default is that no points, only lines are drawn)
`...`	additional arguments to pass to `plot` (only when `use_ggplot2 = FALSE`).

Details

When `type` is "raw", the individual evals are shown (one point per model-draw-method triplet) along with a loess smooth. When `type` is "aggregated", then `center_aggregator` and `spread_aggregator` are used. `center_aggregator` is used to draw a single line per method in which the individual evals computed for each draw has been been aggregated in some way. By default, the `mean_aggregator` is used, which simply averages the evals computed across all draws. When `spread_aggregator` is non-NULL, "error bars" are drawn with (half)widths computed using `spread_aggregator`. By default, the `se_aggregator` is used, which gives an estimate of the standard error of the sample mean.

The arguments method_col, method_lty, method_lwd, method_pch only apply when use_ggplot2 is FALSE.

Examples

```
## Not run:
 # suppose previously we had run the following:
 sim <- new_simulation(name = "normal-example",
                       label = "Normal Mean Estimation",
                       dir = tempdir()) %>%
   generate_model(make_my_example_model,
                  n = list(10, 20, 30),
                  vary_along = "n") %>%
   simulate_from_model(nsim = 50, index = 1:3) %>%
   run_method(my_example_method) %>%
   evaluate(my_example_loss)
   # then we could plot this
   plot_eval_by(sim, "myloss", varying = "n", include_zero = TRUE)

## End(Not run)
```

recycle — *Recycles elements to create vector of desired length*

Description

Recycles elements to create vector of desired length

Usage

```
recycle(x, length)
```

Arguments

`x`	vector to be expanded to proper length
`length`	desired length

relabel — *Give simulation a new label*

Description

Note that `save_simulation` needs to be called for this change to be saved to file.

Usage

```
relabel(sim, label)
```

Arguments

`sim`	object of class `Simulation`
`label`	a longer, human readable label that can have other characters such as spaces, hyphens, etc.

See Also

`rename`

rename *Give simulation a new name*

Description

Note that save_simulation needs to be called for this change to be saved to file.

Usage

```
rename(sim, name)
```

Arguments

sim	object of class Simulation
name	a short name identifier. Must be an alphanumeric (but can also have - or _ within

See Also

relabel

run_extendedmethod_single *Run a single extended method on a single index of simulated data.*

Description

This is an internal function. Users should call the wrapper function. run_method. Here "single" refers to a single index-ExtendedMethod pair.

Usage

```
run_extendedmethod_single(extmethod, model, draws, base_output_list)
```

Arguments

extmethod	a ExtendedMethod object
model	a Model object
draws	a Draws object generated by model
base_output_list	the result of loading a Output object with more_info = TRUE so that it includes RNG endstate.

`run_method` *Run one or more methods on simulated data.*

Description

Given a `Method` object or list of `Method` objects, this function runs the method(s) on the draws passed through `object`. The output of each method is saved to file.

Usage

```
run_method(object, methods, out_loc = "out", parallel = NULL)
```

Arguments

`object`	an object of class `DrawsRef` (or a list of such objects) as returned by `link{simulate_from_model}`. If `object` is a `Simulation`, then function is applied to the referenced draws in that simulation and returns the same `Simulation` object but with references added to the new outputs created.
`methods`	a list of `Method` and/or `ExtendedMethod` objects or a single `Method` or object `ExtendedMethod`
`out_loc`	(optional) a length-1 character vector that gives location (relative to model's path) that method outputs are stored.This can be useful for staying organized when multiple simulations are based on the same Model and Draws objects.
`parallel`	either `NULL` or a list containing `socket_names` and (optionally) `libraries` and `save_locally` (see Details for more information)

Details

This function creates objects of class `Output` and saves each to file (at dir/model_name/<out_loc>/r<index>_<method_name> If parallel is not NULL, then it must be a list containing `socket_names`, which can either be a positive integer specifying the number of copies to run on localhost or else a character vector of machine names (e.g., "mycluster-0-0"). The list `parallel` can also contain `libraries`, a character vector of R packages that will be needed on the slaves and `save_locally`, a logical that indicates whether the files generated should be saved on the slaves (i.e., locally) or on the master.

Before running each method on index i, the RNG state is restored to what it was at the end of calling `simulate_from_model` on this index. This is only relevant for randomized methods. The choice to do this ensures that one will get identical results regardless of the order in which methods and indices are run in. When `ExtendedMethod` objects are passed, these are run after all `Method` objects have been run. This is because each `ExtendedMethod` object depends on the output of its base method. Furthermore, before an `ExtendedMethod` is called, the RNG state is restored to what it was after the base method had been called.

See Also

`generate_model` `simulate_from_model`

Examples

```
## Not run:
 # suppose previously we had run the following:
 sim <- new_simulation(name = "normal-example",
                       label = "Normal Mean Estimation",
                       dir = tempdir()) %>%
   generate_model(make_my_example_model, n = 20) %>%
   simulate_from_model(nsim = 50, index = 1:3)
 # then we could add
 sim <- run_method(sim, my_example_method)

## End(Not run)
```

run_method_single *Run a single method on a single index of simulated data.*

Description

This is an internal function. Users should call the wrapper function. run_method. Here "single" refers to a single index-method pair.

Usage

```
run_method_single(method, model, draws_list)
```

Arguments

method	a Method object
model	a Model object
draws_list	the result of loading a Draws object with more_info = TRUE so that it includes RNG endstate.

save_simulation *Save a simulation object*

Description

Saves an object of class Simulation to sim@dir/files/sim@name.Rdata. Note: while "files" is the default, the name of this directory is from getOption("simulator.files"), which is the value of getOption("simulator.files") when the model was created.

Usage

```
save_simulation(sim)
```

Arguments

`sim`	an object of class `Simulation`

Details

This function overwrites any pre-existing file in that location without apology.

See Also

`new_simulation` `load_simulation`

`simulate_from_model` *Simulate from a model.*

Description

Given a reference to a `Model` object, this function calls the model's `simulate` function on its `params`. It repeats this `nsim` times. For example, when simulating regression with a fixed design, this function would generate `nsim` response vectors `y`.

Usage

```
simulate_from_model(object, nsim, index = 1, parallel = NULL)
```

Arguments

`object`	an object of class `ModelRef` as returned by `link{generate_model}`. Or a list of such objects. If `object` is a `Simulation`, then function is applied to the referenced models in that simulation and returns the same `Simulation` object but with references added to the new draws created.
`nsim`	number of simulations to be conducted. If a scalar, then value repeated for each index. Otherwise can be a vector of length `length(index)`
`index`	a vector of positive integer indices. Allows simulations to be carried out in chunks. Each chunk gets a separate RNG stream, meaning that the results will be identical whether we run these in parallel or sequentially.
`parallel`	either `NULL` or a list containing `socket_names` and (optionally) `libraries` and `save_locally` (see Details for more information)

Details

This function creates objects of class `Draws` and saves each to file (at dir/files/model_name/r<index>.Rdata). Note: while "files" is the default, the name of this directory is from getOption("simulator.files"), which is the value of getOption("simulator.files") when the model was created.

If parallel is not NULL, then it must be a list containing `socket_names`, which can either be a positive integer specifying the number of copies to run on localhost or else a character vector of machine names (e.g., "mycluster-0-0"). The list `parallel` can also contain `libraries`, a character vector of R packages that will be needed on the slaves and `save_locally`, a logical that indicates whether the files generated should be saved on the slaves (i.e., locally) or on the master.

See Also

load_draws generate_model run_method

Examples

```
## Not run:
 sim <- new_simulation(name = "normal-example",
                       label = "Normal Mean Estimation",
                       dir = tempdir()) %>%
   generate_model(make_my_example_model, n = 20) %>%
   simulate_from_model(nsim = 50, index = 1:3,
     parallel = list(socket_names = 3))

## End(Not run)
```

---

simulate_from_model_single *Simulate from a model.*

Description

This is an internal function. Users should call the wrapper function simulate_from_model.

Usage

```
simulate_from_model_single(model, nsim, index, seed)
```

Arguments

model	a Model object
nsim	number of simulations to be conducted.
index	a positive integer index.
seed	this is the 7 digit seed used by L'Ecuyer RNG

---

simulate_parallel *Simulate from a model in parallel.*

Description

This is an internal function. Draws are done in chunks labeled by indices and of size determined by nsim. Users should call the wrapper function simulate_from_model.

Usage

```
simulate_parallel(
  model_ref,
  nsim,
  index,
  seeds,
  socket_names,
  libraries,
  save_locally = TRUE
)
```

Arguments

`model_ref`	object of class `ModelRef`
`nsim`	number of simulations to be conducted on each chunk. Vector of same length as `index`
`index`	a vector of positive integer indices. Allows simulations to be carried out in chunks. Each chunk gets a separate RNG stream, meaning that the results will be identical whether we run these in parallel or sequentially.
`seeds`	a list of `length(index)` L'Ecuyer-CMRG seed vectors. Each should be from a separate stream. In particular, starting from the seed used to generate the model object, seeds[i] should be the result of calling `nextRNGStream` index[i] times.
`socket_names`	(quoting from `makePSOCKcluster` "either a character vector of host names on which to run the worker copies of R, or a positive integer (in which case that number of copies is run on localhost)."
`libraries`	character vector of R packages that will be needed on the slaves.
`save_locally`	if TRUE, then files will be saved on slaves. If FALSE, they will be saved on master.

`Simulation-class` *An S4 class representing a simulation.*

Description

A simulation is a set of references to simulator objects that have been saved to file. The DrawsRef, OutputRef, and EvalsRef objects are organized by model into separate lists.

Details

When a reference ref is added to a simulation sim, ref@dir is changed so that the referenced file is located at file.path(sim@dir, ref@dir).

Slots

`name` a short name identifier. Must be an alphanumeric (but can also have - or _ within

`label` a longer, human readable label that can have other characters such as spaces, hyphens, etc.

`dir` name of the directory where directory named "files" exists.

`model_refs` a list of `ModelRef` objects

`draws_refs` a list of lists of `DrawsRef` objects

`output_refs` a list of lists of `OutputRef` objects

`evals_refs` a list of lists of `EvalsRef` objects

`subset_evals` *Reduce an Evals object to a subset of methods and/or metrics*

Description

If `method_names` is `NULL`, then subsetting is not done over methods. Likewise for `metric_names`.

Usage

```
subset_evals(evals, method_names = NULL, metric_names = NULL)
```

Arguments

`evals`	an object of class `Evals` or `listofEvals`.
`method_names`	a character vector of method names
`metric_names`	a character vector of metric names

`subset_models` *Subset Models*

Description

Given a list of `Model` objects, returns model names which meet conditions. Uses `subset`

Usage

```
subset_models(m, ...)
```

Arguments

`m`	list of `Model` objects
`...`	logical expression involving parameters of Models. For now, can only be parameters that are of length 1 and either of class numeric or character

`subset_simulation` *Create a simulation that is a subset of a preexisting simulation object*

Description

Given a simulation, creates a new simulation that is a subset of the preexisting simulation. Does not save this new one to file. To do so, first change the name (and, potentially, label) of the simulation and then use `save_simulation`. If you call `save_simulation` before changing the name, you will overwrite the preexisting simulation. Use `rename` and `relabel`.

Usage

```
subset_simulation(sim, ..., subset = NULL, index, methods)
```

Arguments

`sim`	a simulation object
`...`	logical conditions to specify a subset of models. Conditions can only involve params of model that have length 1 and are of class numeric or character.
`subset`	a vector of integers indexing the models or a vector of model names. To select models based on parameter values, use .... However, using ... is slower than using subset.
`index`	a vector of positive integers specifying which draws' objects are desired. If missing, then all draws' evals are returned.
`methods`	character vector of method names of interest. If missing, then all methods' evals are returned

`tabulate_eval` *Make a table of a metric for each pair of models and methods*

Description

Each row of the table corresponds to a different model and each column to a different method. The metric must be a scalar. The way in which standard error is shown (or not shown) is controlled by `se_format`.

Usage

```
tabulate_eval(
  object,
  metric_name,
  method_names = NULL,
  caption = NULL,
  center_aggregator = NULL,
  spread_aggregator = NULL,
  se_format = c("Paren", "PlusMinus", "None"),
  output_type = "latex",
  format_args = list(nsmall = 0, digits = NULL, scientific = FALSE),
  na_string = "--",
  bold = c("None", "Smallest", "Largest")
)
```

Arguments

`object`	an object of class Simulation, Evals, or `listofEvals`. Each evals object should just differ by model_name.
`metric_name`	the name of a metric to tabulate. Must be scalar valued.
`method_names`	character vector indicating methods to include in table. If NULL, then will include all methods found in object's evals.
`caption`	caption of plot. If NULL, then default caption used; if FALSE then no caption (and returns tabular without table).
`center_aggregator`	When NULL (which is default), the sample mean aggregator is used. User can write specialized aggregators (see definition of class Aggregator) as necessary, for example, when the evaluated metric is not scalar-valued.
`spread_aggregator`	When NULL (which is default), the standard error of the sample mean is used. User can write specialized aggregators (see definition of class Aggregator) as necessary, for example, when the evaluated metric is not scalar-valued. Set `spread_aggregator` to `NA` to hide error bars.
`se_format`	format of the standard error
`output_type`	see kable's argument format for options. Default is "latex" but other options include "html" and "markdown"
`format_args`	arguments to pass to the function format
`na_string`	what to write in table in place of NA
`bold`	puts in bold the value that is smallest/largest for each model

Details

Uses `knitr`'s function `kable` to put table in various formats, including latex, html, markdown, etc.

Examples

```
## Not run:
 # suppose previously we had run the following:
 sim <- new_simulation(name = "normal-example",
                       label = "Normal Mean Estimation",
                       dir = tempdir()) %>%
   generate_model(make_my_example_model,
                  n = list(10, 20, 30),
                  vary_along = "n") %>%
   simulate_from_model(nsim = 50, index = 1:3) %>%
   run_method(my_example_method) %>%
   evaluate(my_example_loss)
   # then we could plot this
   tabulate_eval(sim, "myloss")

## End(Not run)
```

$,Model-method — *Get element of* `Model`*'s* `params` *list*

Description

Get element of `Model`'s `params` list

Usage

```
## S4 method for signature 'Model'
x$name
```

Arguments

x	object of class `Model`
name	name of an element appearing in `x@params`

Index