Package 'hesim'

June 3, 2020

Type Package

Title Health-Economic Simulation Modeling and Decision Analysis

Version 0.3.1

Description A modular and computationally efficient R package for parameterizing, simulating, and analyzing health-economic simulation models. The package supports cohort discrete time state transition models (Briggs et al. 1998) <doi:10.2165/00019053-199813040-00003>, N-state partitioned survival models (Glasziou et al. 1990) <doi:10.1002/sim.4780091106>, and individual-level continuous time state transition models (Siebert et al. 2012) <doi:10.1016/j.jval.2012.06.014>, encompassing both Markov (time-homogeneous and time-inhomogeneous) and semi-Markov processes. Decision uncertainty from a cost-effectiveness analysis is quantified with standard graphical and tabular summaries of a probabilistic sensitivity analysis (Claxton et al. 2005, Barton et al. 2008) <doi:10.1002/hec.985>, <doi:10.1111/j.1524-4733.2008.00358.x>. Use of C++ and data.table make individual-patient simulation, probabilistic sensitivity analysis, and incorporation of patient heterogeneity fast.

URL https://github.com/hesim-dev/hesim

BugReports https://github.com/hesim-dev/hesim/issues

License GPL-3

LazyData true

LinkingTo Rcpp, RcppArmadillo

Depends R (>= 3.5.0)

Imports data.table, flexsurv, MASS, Rcpp (>= 0.12.16), R6, stats, survival

Suggests covr, ggplot2, kableExtra, knitr, mstate, msm, nnet, numDeriv, pracma, rmarkdown, scales, testthat, truncnorm

VignetteBuilder knitr

RoxygenNote 7.1.0

SystemRequirements C++11

NeedsCompilation yes

Author Devin Incerti [aut, cre],
Jeroen P. Jansen [aut],
R Core Team [ctb] (hesim uses some slightly modified C functions from base R)

Maintainer Devin Incerti <devin.incerti@gmail.com>

Repository CRAN

Date/Publication 2020-06-02 23:20:18 UTC

R topics documented:

`bootstrap` *Bootstrap a statistical model*

Description

`bootstrap` is a generic function for generating bootstrap replicates of the parameters of a fitted statistical model.

Usage

```
bootstrap(object, B, ...)

## S3 method for class 'partsurvfit'
bootstrap(object, B, max_errors = 0, ...)
```

Arguments

`object`	A statistical model.
`B`	Number of bootstrap replications.
`...`	Further arguments passed to or from other methods. Currently unused.
`max_errors`	Maximum number of errors that are allowed when fitting statistical models during the bootstrap procedure. This argument may be useful if, for instance, the model fails to converge during some bootstrap replications. Default is 0.

Value

Sampled values of the parameters.

---

ce *A cost-effectiveness object*

Description

An object that summarizes simulated measures of clinical effectiveness and costs from a simulation model for use in a cost-effectiveness analysis.

Format

A list containing two elements:

- costs Total (discounted) costs by category.
- qalys (Discounted) quality-adjusted life-years.

Costs

The 'costs' `data.table` contains the following columns:

category The cost category.

dr The discount rate.

sample A randomly sampled parameter set from the probabilistic sensitivity analysis (PSA)

strategy_id The treatment strategy ID.

grp An optional column denoting a subgroup. If not included, it is assumed that a single subgroup is being analyzed.

costs Costs.

Quality-adjusted life-years

The 'qalys' `data.table` contains the following columns:

dr The discount rate.

sample A randomly sampled parameter set from the probabilistic sensitivity analysis (PSA)

strategy_id The treatment strategy ID.

grp An optional column denoting a subgroup. If not included, it is assumed that a single subgroup is being analyzed.

qalys Quality-adjusted life-years

`check_edata` *Check data argument for* `create_input_mats`

Description

Check that data argument for `create_input_mats` exists and that it is of the correct type.

Usage

```
check_edata(data)
```

Arguments

`data`	An object of class "expanded_hesim_data" returned by the function expand.hesim_data.

Value

If all tests passed, returns nothing; otherwise, throws an exception.

`CohortDtstm` *Cohort discrete time state transition model*

Description

Simulate outcomes from a cohort discrete time state transition model.

Format

An R6::R6Class object.

Public fields

`trans_model` The model for health state transitions. Must be an object of class CohortDtstmTrans.

`utility_model` The model for health state utility. Must be an object of class StateVals.

`cost_models` The models used to predict costs by health state. Must be a list of objects of class StateVals, where each element of the list represents a different cost category.

`stateprobs_` An object of class stateprobs simulated using $sim_stateprobs().

`qalys_` An object of class qalys simulated using $sim_qalys().

`costs_` An object of class costs simulated using $sim_costs().

Methods

Public methods:

- `CohortDtstm$new()`
- `CohortDtstm$sim_stateprobs()`
- `CohortDtstm$sim_qalys()`
- `CohortDtstm$sim_costs()`
- `CohortDtstm$summarize()`
- `CohortDtstm$clone()`

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

Usage:

```
CohortDtstm$new(trans_model = NULL, utility_model = NULL, cost_models = NULL)
```

Arguments:

`trans_model` The `trans_model` field.

`utility_model` The `utility_model` field.

`cost_models` The `cost_models` field.

Returns: A new `CohortDtstm` object.

Method `sim_stateprobs()`**:** Simulate health state probabilities using `CohortDtstmTrans$sim_stateprobs()`.

Usage:

```
CohortDtstm$sim_stateprobs(n_cycles)
```

Arguments:

`n_cycles` The number of model cycles to simulate the model for.

Returns: An instance of `self` with simulated output of class stateprobs stored in `stateprobs_`.

Method `sim_qalys()`**:** Simulate quality-adjusted life-years (QALYs) as a function of `stateprobs_` and `utility_model`. See `vignette("expected-values")` for details.

Usage:

```
CohortDtstm$sim_qalys(
  dr = 0.03,
  integrate_method = c("trapz", "riemann_left", "riemann_right"),
  lys = FALSE
)
```

Arguments:

`dr` Discount rate.

`integrate_method` Method used to integrate state values when computing (QALYs).

`lys` If `TRUE`, then life-years are simulated in addition to QALYs.

Returns: An instance of `self` with simulated output of class qalys stored in `qalys_`.

Method `sim_costs()`**:** Simulate costs as a function of `stateprobs_` and `cost_models`. See `vignette("expected-values")` for details.

Usage:

```
CohortDtstm$sim_costs(
  dr = 0.03,
  integrate_method = c("trapz", "riemann_left", "riemann_right")
)
```

Arguments:

`dr` Discount rate.

`integrate_method` Method used to integrate state values when computing costs.

Returns: An instance of `self` with simulated output of class costs stored in `costs_`.

Method `summarize()`**:** Summarize costs and QALYs so that cost-effectiveness analysis can be performed. See `summarize_ce()`.

Usage:

```
CohortDtstm$summarize(by_grp = FALSE)
```

Arguments:

`by_grp` If `TRUE`, then costs and QALYs are computed by subgroup. If `FALSE`, then costs and QALYs are aggregated across all patients (and subgroups).

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

`create_CohortDtstm()`, CohortDtstmTrans, `create_CohortDtstmTrans()`

Examples

```
library("data.table")
library("nnet")
transitions_data <- data.table(multinom3_exdata$transitions)

# Treatment strategies, target population, and model structure
n_patients <- 4
patients <- transitions_data[year == 1, .(patient_id, age, female)][
 sort(sample.int(nrow(transitions_data[year == 1]), n_patients))][
```

```
   , grp_id := 1:n_patients]
hesim_dat <- hesim_data(
  patients = patients,
  strategies = data.table(strategy_id = 1:2,
                          strategy_name = c("Reference", "Intervention")),
  states = data.table(state_id = c(1, 2),
                      state_name = c("Healthy", "Sick")) # Non-death health states
)
tmat <- rbind(c(0, 1, 2),
              c(NA, 0, 1),
              c(NA, NA, NA))

# Parameter estimation
## Multinomial logistic regression
data_healthy <- transitions_data[state_from == "Healthy"]
fit_healthy <- multinom(state_to ~ strategy_name + female + age,
                        data = data_healthy, trace = FALSE)
data_sick <- droplevels(transitions_data[state_from == "Sick"])
fit_sick <- multinom(state_to ~ strategy_name + female + age,
                     data = data_sick, trace = FALSE)
transfits <- multinom_list(healthy = fit_healthy, sick = fit_sick)

## Utility
utility_tbl <- stateval_tbl(multinom3_exdata$utility,
                            dist = "beta",
                            hesim_data = hesim_dat)

## Costs
drugcost_tbl <- stateval_tbl(multinom3_exdata$costs$drugs,
                             dist = "fixed",
                             hesim_data = hesim_dat)
medcost_tbl <- stateval_tbl(multinom3_exdata$costs$medical,
                            dist = "gamma",
                            hesim_data = hesim_dat)

# Economic model
n_samples <- 3

## Construct model
### Transitions
transmod_data <- expand(hesim_dat)
transmod <- create_CohortDtstmTrans(transfits,
                                    input_data = transmod_data,
                                    trans_mat = tmat,
                                    n = n_samples,
                                    point_estimate = FALSE)

### Utility
utilitymod <- create_StateVals(utility_tbl, n = n_samples)

### Costs
drugcostmod <- create_StateVals(drugcost_tbl, n = n_samples)
medcostmod <- create_StateVals(medcost_tbl, n = n_samples)
```

```
costmods <- list(Drug = drugcostmod,
                 Medical = medcostmod)

### Combine
econmod <- CohortDtstm$new(trans_model = transmod,
                           utility_model = utilitymod,
                           cost_models = costmods)

## Simulate outcomes
econmod$sim_stateprobs(n_cycles = 20)
econmod$sim_qalys(dr = .03)
econmod$sim_costs(dr = .03)
econmod$summarize()
econmod$summarize(by_grp = TRUE)
```

`CohortDtstmTrans` *Transitions for a cohort discrete time state transition model*

Description

Simulate health state transitions in a cohort discrete time state transition model.

Format

An R6::R6Class object.

Public fields

`params` Parameters for simulating health state transitions. Supports objects of class tparams_transprobs or params_mlogit.

`input_data` An object of class input_mats.

`cycle_length` The length of a model cycle in terms of years. The default is `1` meaning that model cycles are 1 year long.

Active bindings

`start_stateprobs` A non-negative vector with length equal to the number of health states containing the probability that the cohort is in each health state at the start of the simulation. For example, if there were three states and the cohort began the simulation in state 1, then `start_stateprobs = c(1,0,0)`. Automatically normalized to sum to 1. If `NULL`, then a vector with the first element equal to 1 and all remaining elements equal to 0.

`trans_mat` A transition matrix describing the states and transitions in a discrete-time multi-state model. Only required if the model is parameterized using multinomial logistic regression. The (i,j) element represents a transition from state `i` to state `j`. Each possible transition from row `i` should be based on a separate multinomial logistic regression and ordered from `0` to `K -1` where `K` is the number of possible transitions. Transitions that are not possible should be `NA`. and the reference category for each row should be `0`.

Methods

Public methods:

- CohortDtstmTrans$new()
- CohortDtstmTrans$sim_stateprobs()
- CohortDtstmTrans$clone()

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

Usage:

```
CohortDtstmTrans$new(
  params,
  input_data = NULL,
  trans_mat = NULL,
  start_stateprobs = NULL,
  cycle_length = 1
)
```

Arguments:

`params` The `params` field.

`input_data` The `input_data` field.

`trans_mat` The `trans_mat` field.

`start_stateprobs` The `start_stateprobs` field.

`cycle_length` The `cycle_length` field.

Returns: A new `CohortDtstmTrans` object.

Method `sim_stateprobs()`**:** Simulate probability of being in each health state during each model cycle.

Usage:

```
CohortDtstmTrans$sim_stateprobs(n_cycles)
```

Arguments:

`n_cycles` The number of model cycles to simulate the model for.

Returns: An object of class stateprobs.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

`create_CohortDtstmTrans()`, CohortDtstm

costs *Costs object*

Description

An object of class `costs` returned from methods $sim_costs() in model classes that store simulated costs.

Components

A `costs` object inherits from `data.table` and contains the following columns:

sample A random sample from the PSA.

strategy_id The treatment strategy ID.

patient_id The patient ID.

state_id The health state ID.

dr The rate used to discount costs.

category The cost category (e.g., drug costs, medical costs, etc).

costs The simulated cost values.

create_CohortDtstm *Create* `CohortDtstm` *object*

Description

A generic function for creating an object of class CohortDtstm.

Usage

```
create_CohortDtstm(object, ...)

## S3 method for class 'model_def'
create_CohortDtstm(
  object,
  input_data,
  cost_args = NULL,
  utility_args = NULL,
  ...
)
```

Arguments

object	An object of the appropriate class.
...	Further arguments passed to CohortDtstmTrans$new() in CohortDtstmTrans.
input_data	An object of class expanded_hesim_data.
cost_args	A list of further arguments passed to StateVals$new() in StateVals when initiating cost models.
utility_args	A list of further arguments passed to StateVals$new() in StateVals when initiating the utility model.

---

create_CohortDtstmTrans

Create CohortDtstmTrans *object*

Description

A generic function for creating an object of class CohortDtstmTrans.

Usage

```
create_CohortDtstmTrans(object, ...)

## S3 method for class 'multinom_list'
create_CohortDtstmTrans(
  object,
  input_data,
  trans_mat,
  n = 1000,
  point_estimate = FALSE,
  ...
)
```

Arguments

object	An object of the appropriate class.
...	Further arguments passed to CohortDtstmTrans$new() in CohortDtstmTrans.
input_data	An object of class expanded_hesim_data returned by expand.hesim_data()
trans_mat	A transition matrix describing the states and transitions in a discrete-time multistate model. See CohortDtstmTrans.
n	Number of random observations of the parameters to draw.
point_estimate	If TRUE, then the point estimates are returned and and no samples are drawn.

create_IndivCtstmTrans *Create* IndivCtstmTrans *object*

Description

A generic function for creating an object of class IndivCtstmTrans.

Usage

```
create_IndivCtstmTrans(object, ...)

## S3 method for class 'flexsurvreg_list'
create_IndivCtstmTrans(
  object,
  input_data,
  trans_mat,
  clock = c("reset", "forward"),
  n = 1000,
  point_estimate = FALSE,
  ...
)

## S3 method for class 'flexsurvreg'
create_IndivCtstmTrans(
  object,
  input_data,
  trans_mat,
  clock = c("reset", "forward"),
  n = 1000,
  point_estimate = FALSE,
  ...
)

## S3 method for class 'params_surv'
create_IndivCtstmTrans(
  object,
  input_data,
  trans_mat,
  clock = c("reset", "forward", "mix"),
  reset_states = NULL,
  ...
)
```

Arguments

object	A fitted survival model or the parameters of a survival model.

...	Further arguments passed to IndivCtstmTrans$new() in IndivCtstmTrans.
input_data	An object of class "expanded_hesim_data" returned by expand.hesim_data.
trans_mat	The transition matrix describing the states and transitions in a multi-state model in the format from the mstate package. See IndivCtstmTrans.
clock	"reset" for a clock-reset model, "forward" for a clock-forward model, and "mix" for a mixture of clock-reset and clock-forward models. See the field clock in IndivCtstmTrans.
n	Number of random observations of the parameters to draw.
point_estimate	If TRUE, then the point estimates are returned and and no samples are drawn.
reset_states	A vector denoting the states in which time resets. See the field reset_states in IndivCtstmTrans.

Value

Returns an R6Class object of class IndivCtstmTrans.

See Also

IndivCtstmTrans

---

create_lines_dt *Create a data table of treatment lines*

Description

Convert a list of treatment lines for multiple treatment strategies to a data.table.

Usage

```
create_lines_dt(strategy_list, strategy_ids = NULL)
```

Arguments

strategy_list	A list where each element is a treatment strategy consisting of a vector of treatments.
strategy_ids	A numeric vector denoting the numeric id of each strategy in strategy_list.

Value

Returns a data.table in tidy format with three columns:

strategy_id Treatment strategy ids.

line Line of therapy.

treatment_id Treatment ID for treatment used at a given line of therapy within a treatment strategy.

Examples

```
strategies <- list(c(1, 2, 3),
                   c(1, 2))
create_lines_dt(strategies)
```

`create_params` *Create a parameter object from a fitted model*

Description

`create_params` is a generic function for creating an object containing parameters from a fitted statistical model. If `point_estimate = FALSE`, then random samples from suitable probability distributions are returned.

Usage

```
create_params(object, ...)

## S3 method for class 'lm'
create_params(object, n = 1000, point_estimate = FALSE, ...)

## S3 method for class 'flexsurvreg'
create_params(object, n = 1000, point_estimate = FALSE, ...)

## S3 method for class 'multinom'
create_params(object, n = 1000, point_estimate = FALSE, ...)

## S3 method for class 'multinom_list'
create_params(object, n = 1000, point_estimate = FALSE, ...)

## S3 method for class 'flexsurvreg_list'
create_params(object, n = 1000, point_estimate = FALSE, ...)

## S3 method for class 'partsurvfit'
create_params(
  object,
  n = 1000,
  point_estimate = FALSE,
  bootstrap = TRUE,
  max_errors = 0,
  ...
)
```

Arguments

`object`	A statistical model to randomly sample parameters from.
`...`	Further arguments passed to or from other methods. Currently unused.

n	Number of random observations to draw.
point_estimate	If TRUE, then the point estimates are returned and and no samples are drawn.
bootstrap	If bootstrap is FALSE or not specified, then n parameter sets are drawn by sampling from a multivariate normal distribution. If bootstrap is TRUE, then parameters are bootstrapped using bootstrap.
max_errors	Equivalent to the max_errors argument in bootstrap.

Value

An object prefixed by params_. Mapping between create_params and the classes of the returned objects are:

- create_params.lm -> params_lm
- create_params.multinom -> params_mlogit
- create_params.multinom_list -> params_mlogit_list
- create_params.flexsurvreg -> params_surv
- create_params.flexsurvreg_list -> params_surv_list
- create_params.partsurvfit -> params_surv_list

Examples

```
# create_params.lm
fit <- stats::lm(costs ~ female, data = psm4_exdata$costs$medical)
n <- 5
params_lm <- create_params(fit, n = n)
head(params_lm$coefs)
head(params_lm$sigma)

# create_params.flexsurvreg
library("flexsurv")
fit <- flexsurv::flexsurvreg(formula = Surv(futime, fustat) ~ 1,
                    data = ovarian, dist = "weibull")
n <- 5
params_surv_wei <- create_params(fit, n = n)
print(params_surv_wei$dist)
head(params_surv_wei$coefs)
```

---

create_PsmCurves *Create* PsmCurves *object*

Description

create_PsmCurves is a function for creating an object of class PsmCurves.

Usage

```
create_PsmCurves(object, ...)

## S3 method for class 'flexsurvreg_list'
create_PsmCurves(
  object,
  input_data,
  n = 1000,
  point_estimate = FALSE,
  bootstrap = FALSE,
  est_data = NULL,
  ...
)

## S3 method for class 'params_surv_list'
create_PsmCurves(object, input_data, ...)
```

Arguments

object	Fitted survival models.
...	Further arguments passed to or from other methods. Currently unused.
input_data	An object of class "expanded_hesim_data" returned by expand.hesim_data. Must be expanded by the data tables "strategies" and "patients".
n	Number of random observations of the parameters to draw.
point_estimate	If TRUE, then the point estimates are returned and and no samples are drawn.
bootstrap	If TRUE, then n bootstrap replications are drawn by refitting the survival models in object on resamples of the sample data; if FALSE, then the parameters for each survival model are independently draw from multivariate normal distributions.
est_data	A data.table or data.frame of estimation data used to fit survival models during bootstrap replications.

Value

Returns an R6Class object of class PsmCurves.

See Also

PsmCurves

`create_StateVals` *Create a* `StateVals` *object*

Description

`create_StateVals()` is a generic function for creating an object of class StateVals from a fitted statistical model or a stateval_tbl object.

Usage

```
create_StateVals(object, ...)

## S3 method for class 'lm'
create_StateVals(
  object,
  input_data = NULL,
  n = 1000,
  point_estimate = FALSE,
  ...
)

## S3 method for class 'stateval_tbl'
create_StateVals(object, n = 1000, ...)
```

Arguments

`object`	A model object of the appropriate class.
`...`	Further arguments (`time_reset` and `method`) passed to StateVals$new().
`input_data`	An object of class expanded_hesim_data. Must be expanded by treatment strategies, patients, and health states.
`n`	Number of random observations of the parameters to draw when parameters are fit using a statistical model.
`point_estimate`	If `TRUE`, then the point estimates are returned and and no samples are drawn.

Value

A StateVals object.

See Also

StateVals, stateval_tbl

create_trans_dt *Create a data table of health state transitions*

Description

Create a data table of health state transitions from a transition matrix describing the states and transitions in a multi-state model suitable for use with hesim_data.

Usage

```
create_trans_dt(trans_mat)
```

Arguments

trans_mat	A transition matrix in the format from the mstate package. See IndivCtstmTrans.

Value

Returns a `data.table` in tidy format with three columns

transition_id Health state transition ID.

from The starting health state.

to The health state that will be transitions to.

Examples

```
tmat <- rbind(c(NA, 1, 2),
              c(NA, NA, 3),
              c(NA, NA, NA))
create_trans_dt(tmat)
```

CtstmTrans *An R6 base class for continuous time state transition models*

Description

A non-exported base class for continuous time state transition models containing methods that can be used to summarize fitted multi-state models.

Format

An R6::R6Class object.

Methods

Public methods:

- CtstmTrans$hazard()
- CtstmTrans$cumhazard()
- CtstmTrans$clone()

Method `hazard()`**:** Predict the hazard functions for each health state transition.

Usage:

```
CtstmTrans$hazard(t)
```

Arguments:

`t` A numeric vector of times.

Returns: A `data.table` with columns `trans` (the transition number), `sample`, `strategy_id`, `grp_id`, `t`, and `hazard`.

Method `cumhazard()`**:** Predict the cumulative hazard functions for each health state transition.

Usage:

```
CtstmTrans$cumhazard(t)
```

Arguments:

`t` A numeric vector of times.

Returns: A `data.table` with columns `trans`, `sample`, `strategy_id`, `grp_id`, `t`, and `hazard`.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

`create_IndivCtstmTrans()`, IndivCtstmTrans

`define_model` *Define and evaluate model expression*

Description

A model expression is defined by specifying random number generation functions for a probabilistic sensitivity analysis and transformations of the sampled parameters as a function of `input_data`. The unevaluated expressions are evaluated with `eval_model()` and used to generate the model inputs needed to create an economic model.

Usage

```
define_model(tparams_def, rng_def, params = NULL, n_states = NULL)

eval_model(x, input_data)
```

Arguments

tparams_def	A tparams_def object or a list of tparams_def objects. A list might be considered if time intervals specified with the times argument in define_tparams() vary across parameters. Parameters for a transition probability matrix (tpmatrix), utilities (utility), and/or cost categories (costs) are returned as a named list (see define_tparams() for more details).
rng_def	A rng_def object used to randomly draw samples of the parameters from suitable probability distributions.
params	A list containing the values of parameters for random number generation.
n_states	The number of health states (inclusive of all health states including the the death state) in the model. If tpmatrix is an element returned by tparams_def, then it will be equal to the number of states in the transition probability matrix; otherwise it must be specified as an argument.
x	An object of class model_def created with define_model().
input_data	An object of class expanded_hesim_data expanded by patients and treatment strategies.

Details

eval_model() evaluates the expressions in an object of class model_def returned by define_model() and is, in turn, used within functions that instantiate economic models (e.g., create_CohortDtstm()). The direct output of eval_model() can also be useful for understanding and debugging model definitions, but it is not used directly for simulation.

A model_def object would typically be defined in four steps by specifying:

1. *Data* (input_data) of class expanded_hesim_data consisting of the treatment strategies and patient population
2. *Parameters* (params) objects storing the values of all parameters used in the model
3. *Random number generation* (define_rng()) expressions that randomly sample values of the parameters from suitable probability distributions for probabilistic sensitivity analysis
4. *Transformed parameter* (define_tparams()) expressions that transform the parameter estimates into values used for simulation

Step 2 can be omitted if underlying parameter values are defined in steps 3. The output of step 4 is used to instantiate the economic model (or a portion of an economic model) as a function of a transition probability matrix, utilities, and/or costs.

Value

`define_model()` returns an object of class `model_def`, which is a list containing the arguments to the function. `eval_model()` returns a list containing ID variables identifying parameter samples, treatment strategies, patient cohorts, and time intervals; the values of parameters of the transition probability matrix, utilities, and/or cost categories; the number of health states; and the number of random number generation samples for the PSA.

See Also

`define_tparams()`, `define_rng()`

Examples

```
# Data
library("data.table")
strategies <- data.table(strategy_id = 1:2,
                         strategy_name = c("Monotherapy", "Combination therapy"))
patients <- data.table(patient_id = 1)
hesim_dat <- hesim_data(strategies = strategies,
                        patients = patients)
data <- expand(hesim_dat)

# Define the model
rng_def <- define_rng({
  alpha <- matrix(c(1251, 350, 116, 17,
                    0, 731, 512, 15,
                    0, 0, 1312, 437,
                    0, 0, 0, 469),
                  nrow = 4, byrow = TRUE)
  rownames(alpha) <- colnames(alpha) <- c("A", "B", "C", "D")
  lrr_mean <- log(.509)
  lrr_se <- (log(.710) - log(.365))/(2 * qnorm(.975))
  list(
    p_mono = dirichlet_rng(alpha),
    rr_comb = lognormal_rng(lrr_mean, lrr_se),
    u = 1,
    c_zido = 2278,
    c_lam = 2086.50,
    c_med = gamma_rng(mean = c(A = 2756, B = 3052, C = 9007),
                      sd = c(A = 2756, B = 3052, C = 9007))
  )
}, n = 2)

tparams_def <- define_tparams({
  rr = ifelse(strategy_name == "Monotherapy", 1, rr_comb)
  list(
    tpmatrix = tpmatrix(
      C, p_mono$A_B * rr, p_mono$A_C * rr, p_mono$A_D * rr,
      0, C, p_mono$B_C * rr, p_mono$B_D * rr,
      0, 0, C, p_mono$C_D * rr,
      0, 0, 0, 1),
```

```
    utility = u,
    costs = list(
      drug = ifelse(strategy_name == "Monotherapy",
                    c_zido, c_zido + c_lam),
      medical = c_med
    )
  )
})

model_def <- define_model(
  tparams_def = tparams_def,
  rng_def = rng_def)

# Evaluate the model expression to generate model inputs
# This can be useful for understanding the output of a model expression
eval_model(model_def, data)

# Create an economic model with a factory function
econmod <- create_CohortDtstm(model_def, data)
```

`define_rng` *Define and evaluate random number generation expressions*

Description

Random number generation expressions are used to randomly sample model parameters from suitable distributions for probabilistic sensitivity analysis. These functions are typically used when evaluating an object of class `model_def` defined using `define_model()`.

Usage

```
define_rng(expr, n = 1, ...)

eval_rng(x, params = NULL)
```

Arguments

`expr`	An expression used to randomly draw variates for each parameter of interest in the model. Braces should be used so that the result of the last expression within the braces is evaluated. The expression must return a list where each element is either a `vector`, `matrix`, `data.frame`, or `data.table`. The length of the `vector` and number of rows in the `matrix`/`data.frame`/`data.table`, must either be 1 or `n`.
`n`	Number of samples of the parameters to draw.
`...`	Additional arguments to pass to the environment used to evaluate `expr`.
`x`	An object of class `rng_def` created with `define_rng()`.
`params`	A list containing the values of parameters for random number generation. Each element of the list should either be a `vector`, `matrix`, `data.frame`, or `data.table`

Details

hesim contains a number of random number generation functions that return parameter samples in convenient formats and do not require the number of samples, n, as arguments (see rng_distributions). The random number generation expressions are evaluated using eval_rng() and used within expr in define_rng(). If multivariate object is returned by eval_rng(), then the rows are random samples and columns are distinct parameters (e.g., costs for each health state, elements of a transition probability matrix).

Value

define_rng() returns an object of class rng_def, which is a list containing the unevaluated random number generation expressions passed to expr, n, and any additional arguments passed to ... . eval_rng() evaluates the rng_def object and should return a list.

See Also

rng_distributions, define_model(), define_tparams()

Examples

```
params <- list(
  alpha = matrix(c(75, 25, 33, 67), byrow = TRUE, ncol = 2),
  inptcost_mean = c(A = 900, B = 1500, C = 2000),
  outptcost_mean = matrix(c(300, 600, 800,
                            400, 700, 700),
                          ncol = 3, byrow = TRUE)
)
rng_def <- define_rng({
  aecost_mean <- c(500, 800, 1000) # Local object not
                                   # not returned by eval_rng()
  list( # Sampled values of parameters returned by eval_rng()
    p = dirichlet_rng(alpha), # Default column names
    inptcost = gamma_rng(mean = inptcost_mean, # Column names based on
                         sd = inptcost_mean),  # named vector
    outptcost = outptcost_mean, # No column names because
                                # outptcost_mean has none.
    aecost = gamma_rng(mean = aecost_mean, # Explicit naming of columns
                       sd = aecost_mean,
                       names = aecost_colnames)
  )
}, n = 2, aecost_colnames = c("A", "B", "C")) # Add aecost_colnames to environment
eval_rng(x = rng_def, params)
```

---

define_tparams *Define and evaluate transformed parameter expressions*

Description

Transformed parameter expressions are used to transform the parameter values sampled with `eval_rng()` as a function of input data (treatment strategies and patients) and time intervals. These functions are used when evaluating an object of class `model_def` defined using `define_model()`. The transformed parameters are ultimately converted into tparams objects and used to simulate outcomes with an economic model.

Usage

```
define_tparams(expr, times = NULL, ...)

eval_tparams(x, input_data, rng_params)
```

Arguments

`expr`	Expressions used to transform parameters. As with `define_rng()`, braces should be used so that the result of the last expression within the braces is evaluated. The expression must return a named list with the following possible elements: • *tpmatrix*: The transition probability matrix used to simulate transition probabilities in the economic model. Typically the output of `tpmatrix()`. • *utility*: The utility values to attach to states and used to simulate quality-adjusted life-years in the economic model. Either a vector (in which case utility is the same in each health state) or a `data.table`/`data.frame`/`matrix` with a column for each (non-death) health state. • *costs*: A named list of costs for each category used to simulate costs in the economic model. Each element of the list must be in the same format as `utility`.
`times`	Distinct times denoting the stopping time of time intervals.
`...`	Additional arguments to pass to the environment used to evaluate `expr`.
`x`	An object of class `tparams_def`.
`input_data`	An object of class expanded_hesim_data (as in `eval_model()`) expanded by the distinct times in `times`.
`rng_params`	Random samples of the parameters returned by `eval_rng()`.

Details

`define_tparams()` is evaluated when creating economic models as a function of `model_def` objects defined with `define_model()`. Operations are "vectorized" in the sense that they are performed for each unique combination of `input_data` and `params`. `expr` is evaluated in an environment including each variable from `input_data`, all elements of `rng_params`, and a variable `time` containing the values from `times`. The `time` variable can be used to create models where parameters vary as a function of time. `eval_tparams()` is not exported and is only meant for use within `eval_model()`.

Value

`define_tparams()` returns an object of class `tparams_def`, which is a list containing the unevaluated "transformation" expressions passed to `expr`, `times`, and any additional arguments passed to `....` `eval_tparams()` evaluates the `tparams_def` object and should return a list of transformed parameter objects.

See Also

`define_model()`, `define_rng()`

`expand.hesim_data` *Expand hesim_data*

Description

Create a data table in long format from all combinations of specified tables from an object of class hesim_data and optionally time intervals. See "Details" for an explanation of how the expansion is done.

Usage

```
## S3 method for class 'hesim_data'
expand(object, by = c("strategies", "patients"), times = NULL)
```

Arguments

`object`	An object of class `hesim_data`.
`by`	A character vector of the names of the data tables in `hesim_data` to expand by.
`times`	Either a numeric vector of distinct times denoting the start of time intervals or q time_intervals object.

Details

This function is similar to `expand.grid()`, but works for data frames or data tables. Specifically, it creates a `data.table` from all combinations of the supplied tables in `object` and optionally the start of times intervals in `times`. The supplied tables are determined using the `by` argument. The resulting dataset is sorted by prioritizing ID variables as follows: (i) `strategy_id`, (ii) `patient_id`, (iii) the health-related ID variable (either `state_id` or `transition_id`, and (iv) the time intervals from `times`.

Value

An object of class `expanded_hesim_data`, which is a `data.table` with an "id_vars" attribute containing the names of the ID variables in the data table and, if `times` is not `NULL`, a `time_intervals` object derived from `times`.

Examples

```
strategies <- data.frame(strategy_id = c(1, 2))
patients <- data.frame(patient_id = seq(1, 3), age = c(65, 50, 75),
                         gender = c("Female", "Female", "Male"))
states <- data.frame(state_id =  seq(1, 3),
                     state_var = c(2, 1, 9))
hesim_dat <- hesim_data(strategies = strategies,
                        patients = patients,
                        states = states)
expand(hesim_dat, by = c("strategies", "patients"))
expand(hesim_dat, by = c("strategies", "patients"),
       times = c(0, 2, 10))
```

`fast_rgengamma` *Random generation for generalized gamma distribution*

Description

Draw random samples from a generalized gamma distribution using the parameterization from `flexsurv`. Written in C++ for speed. Equivalent to `flexsurv::rgengamma`.

Usage

```
fast_rgengamma(n, mu = 0, sigma = 1, Q)
```

Arguments

`n`	Number of random observations to draw.
`mu`	Vector of location parameters. and columns correspond to rates during specified time intervals.
`sigma`	Vector of scale parameters as described in `flexsurv`.
`Q`	Vector of shape parameters.

Value

A vector of random samples from the generalized gamma distribution. The length of the sample is determined by n. The numerical arguments other than n are recycled so that the number of samples is equal to n.

Examples

```
n <- 1000
m <- 2 ; s <- 1.7; q <- 1
ptm <- proc.time()
r1 <- fast_rgengamma(n, mu = m, sigma = s, Q = q)
proc.time() - ptm
ptm <- proc.time()
library("flexsurv")
```

```
r2 <- flexsurv::rgengamma(n, mu = m, sigma = s, Q = q)
proc.time() - ptm
summary(r1)
summary(r2)
```

`flexsurvreg_list` *List of* `flexsurvreg` *objects*

Description

Combine flexsurvreg into a list.

Usage

```
flexsurvreg_list(...)
```

Arguments

...	Objects of class flexsurvreg, which can be named.

Value

An object of class `flexsurvreg_list`.

Examples

```
library("flexsurv")
fit1 <- flexsurv::flexsurvreg(formula = Surv(futime, fustat) ~ 1, data = ovarian, dist = "weibull")
fit2 <- flexsurv::flexsurvreg(formula = Surv(futime, fustat) ~ 1, data = ovarian, dist = "exp")
fsreg_list <- flexsurvreg_list(wei = fit1, exp = fit2)
class(fsreg_list)
```

`hesim` `hesim`*: Health-Economic Simulation Modeling and Decision Analysis*

Description

To learn more about `hesim` visit the website.

hesim_data *Data for health-economic simulation modeling*

Description

A list of tables required for health-economic simulation modeling. Each table must either be a `data.frame` or `data.table`. All ID variables within each table must be numeric vectors of integers.

Usage

```
hesim_data(strategies, patients, states = NULL, transitions = NULL)
```

Arguments

`strategies`	A table of treatment strategies. Must contain the column `strategy_id` denoting a unique strategy. Other columns are variables describing the characteristics of a treatment strategy.
`patients`	A table of patients. Must contain the column `patient_id` denoting a unique patient. The number of rows should be equal to the number of patients in the model. The table may also include columns for `grp_id` for subgroups and `patient_wt` specifying the weight to apply to each patient (within a subgroup). If `grp_id` is `NULL`, then it is assumed that there is only 1 subgroup. If `patient_wt` is `NULL`. then each patient is given the same weight. Weights within subgroups are normalized to sum to one. Other columns are variables describing the characteristics of a patient.
`states`	A table of health states. Must contain the column `state_id`, which denotes a unique health state. The number of rows should be equal to the number of health states in the model. Other columns can describe the characteristics of a health state.
`transitions`	A table of health state transitions. Must contain the column `transition_id`, which denotes a unique transition; `from`, which denotes the starting health state; and `to` which denotes the state that will be transitioned to.

Value

Returns an object of class `hesim_data`, which is a list of data tables for health economic simulation modeling.

See Also

`expand.hesim_data()`

Examples

```
strategies <- data.frame(strategy_id = c(1, 2))
patients <- data.frame(patient_id = seq(1, 3), age = c(65, 50, 75),
                       gender = c("Female", "Female", "Male"))
states <- data.frame(state_id =  seq(1, 3),
                     state_var = c(2, 1, 9))
hesim_dat <- hesim_data(strategies = strategies,
                        patients = patients,
                        states = states)
```

`hesim_survdists` *List of survival distributions*

Description

List of additional distributions for parametric survival analysis that are not contained in flexsurv. Can be used to fit models with flexsurvreg. Same format as flexsurv.dists in flexsurv.

Usage

```
hesim_survdists
```

Format

A list with the following elements:

name Name of the probability distribution.

pars Vector of strings naming the parameters of the distribution. These must be the same names as the arguments of the density and probability functions.

location Name of the location parameter.

transforms List of R functions which transform the range of values taken by each parameter onto the real line. For example, `c(log,log)` for a distribution with two positive parameters.

inv.transforms List of R functions defining the corresponding inverse transformations. Note these must be lists, even for single parameter distributions they should be supplied as, e.g. `c(exp)` or `list(exp)`.

inits A function of the observed survival times `t` (including right-censoring times, and using the halfway point for interval-censored times) which returns a vector of reasonable initial values for maximum likelihood estimation of each parameter. For example, `function(t){ c(1,mean(t)) }` will always initialize the first of two parameters at 1, and the second (a scale parameter, for instance) at the mean of `t`.

icea *Individualized cost-effectiveness analysis*

Description

Conduct individualized cost-effectiveness analysis (ICEA) given output of an economic model; that is, summarize a probabilistic sensitivity analysis (PSA) by subgroup.

- `icea()` computes the probability that each treatment is most cost-effective, output for a cost-effectiveness acceptability frontier, the expected value of perfect information, and the net monetary benefit for each treatment.
- `icea_pw()` conducts pairwise ICEA by comparing strategies to a comparator. Computed quantities include the incremental cost-effectiveness ratio, the incremental net monetary benefit, output for a cost-effectiveness plane, and output for a cost-effectiveness acceptability curve.

Usage

```
icea(x, ...)

icea_pw(x, ...)

## Default S3 method:
icea(x, k = seq(0, 2e+05, 500), sample, strategy, grp = NULL, e, c, ...)

## Default S3 method:
icea_pw(
  x,
  k = seq(0, 2e+05, 500),
  comparator,
  sample,
  strategy,
  grp = NULL,
  e,
  c,
  ...
)

## S3 method for class 'ce'
icea(x, k = seq(0, 2e+05, 500), dr_qalys, dr_costs, ...)

## S3 method for class 'ce'
icea_pw(x, k = seq(0, 2e+05, 500), comparator, dr_qalys, dr_costs, ...)
```

Arguments

x	An object of simulation output characterizing the probability distribution of clinical effectiveness and costs. If the default method is used, then x must be

	a data.frame or data.table containing columns of mean costs and clinical effectiveness where each row denotes a randomly sampled parameter set and treatment strategy.
...	Further arguments passed to or from other methods. Currently unused.
k	Vector of willingness to pay values.
sample	Character name of column from x denoting a randomly sampled parameter set.
strategy	Character name of column from x denoting treatment strategy.
grp	Character name of column from x denoting subgroup. If NULL, then it is assumed that there is only one group.
e	Character name of column from x denoting clinical effectiveness.
c	Character name of column from x denoting costs.
comparator	Name of the comparator strategy in x.
dr_qalys	Discount rate for quality-adjusted life-years (QALYs).
dr_costs	Discount rate for costs.

Value

icea() returns a list of four data.table elements.

summary A data.table of the mean, 2.5% quantile, and 97.5% quantile by strategy and group for clinical effectiveness and costs.

mce The probability that each strategy is the most effective treatment for each group for the range of specified willingness to pay values. In addition, the column best denotes the optimal strategy (i.e., the strategy with the highest expected net monetary benefit), which can be used to plot the cost-effectiveness acceptability frontier (CEAF).

evpi The expected value of perfect information (EVPI) by group for the range of specified willingness to pay values. The EVPI is computed by subtracting the expected net monetary benefit given current information (i.e., the strategy with the highest expected net monetary benefit) from the expected net monetary benefit given perfect information.

nmb The mean, 2.5% quantile, and 97.5% quantile of net monetary benefits for the range of specified willingness to pay values.

icea_pw also returns a list of four data.table elements:

summary A data.table of the mean, 2.5% quantile, and 97.5% quantile by strategy and group for clinical effectiveness and costs.

delta Incremental effectiveness and incremental cost for each simulated parameter set by strategy and group. Can be used to plot a cost-effectiveness plane.

ceac Values needed to plot a cost-effectiveness acceptability curve by group. The CEAC plots the probability that each strategy is more cost-effective than the comparator for the specified willingness to pay values.

inmb The mean, 2.5% quantile, and 97.5% quantile of incremental net monetary benefits for the range of specified willingness to pay values.

Examples

```
# simulation output
n_samples <- 100
sim <- data.frame(sample = rep(seq(n_samples), 4),
                  c = c(rlnorm(n_samples, 5, .1), rlnorm(n_samples, 5, .1),
                        rlnorm(n_samples, 11, .1), rlnorm(n_samples, 11, .1)),
                  e = c(rnorm(n_samples, 8, .2), rnorm(n_samples, 8.5, .1),
                        rnorm(n_samples, 11, .6), rnorm(n_samples, 11.5, .6)),
                  strategy = rep(paste0("Strategy ", seq(1, 2)),
                                 each = n_samples * 2),
                  grp = rep(rep(c("Group 1", "Group 2"),
                            each = n_samples), 2)
)

# icea
icea <- icea(sim, k = seq(0, 200000, 500), sample = "sample", strategy = "strategy",
             grp = "grp", e = "e", c = "c")
names(icea)
# The probability that each strategy is the most cost-effective
# in each group with a willingness to pay of 20,000
library("data.table")
icea$mce[k == 20000]

# icea_pw
icea_pw <-  icea_pw(sim,  k = seq(0, 200000, 500), comparator = "Strategy 1",
                    sample = "sample", strategy = "strategy", grp = "grp",
                     e = "e", c = "c")
names(icea_pw)
# cost-effectiveness acceptability curve
head(icea_pw$ceac[k >= 20000])
icer_tbl(icea_pw)
```

---

icer_tbl *ICER table*

Description

Generate a table of incremental cost-effectiveness ratios given output from `icea_pw()`.

Usage

```
icer_tbl(
  x,
  k = 50000,
  cri = TRUE,
  prob = 0.95,
  digits_qalys = 2,
  digits_costs = 0,
  output = c("matrix", "data.table"),
```

```
  rownames = NULL,
  colnames = NULL,
  drop = TRUE
)
```

Arguments

x	An object of class icea_pw returned by icea_pw().
k	Willingness to pay.
cri	If TRUE, credible intervals are computed; otherwise they are not.
prob	A numeric scalar in the interval (0,1) giving the credible interval. Default is 0.95 for a 95 percent credible interval.
digits_qalys	Number of digits to use to report QALYs.
digits_costs	Number of digits to use to report costs.
output	Should output be a data.table or a list of matrices for each group.
rownames	Row names for matrices when output = "matrix".
colnames	Column names for matrices when output = "matrix".
drop	If TRUE, then the result is coerced to the lowest possible dimension. Relevant if output = "matrix" and there is one group, in which case a single matrix will be returned if drop = TRUE and a list of length 1 will be returned if drop = FALSE.

Value

If output = "matrix", then a list of matrices (or a matrix if drop = TRUE) reporting incremental cost-effectiveness ratios (ICERs) by group. Specifically, each matrix contains five rows for: (i) incremental quality-adjusted life-years (QALYs), (ii) incremental costs, (iii) the incremental net monetary benefit (NMB), (iv) the ICER, and (v) a conclusion stating whether each strategy is cost-effective relative to a comparator. The number of columns is equal to the number of strategies (including the comparator).

If output = "data.table", then the results are reported as a data.table, with one row for each strategy and group combination.

See Also

icea_pw()

---

id_attributes *Attributes for ID variables*

Description

Stores metadata related to the ID variables used to index input_mats and transformed parameter objects already predicted from covariates.

Usage

```
id_attributes(
  strategy_id,
  n_strategies,
  patient_id,
  n_patients,
  state_id = NULL,
  n_states = NULL,
  transition_id = NULL,
  n_transitions = NULL,
  time_id = NULL,
  time_intervals = NULL,
  n_times = NULL,
  sample = NULL,
  n_samples = NULL,
  grp_id = NULL,
  patient_wt = NULL
)
```

Arguments

strategy_id	A numeric vector of integers denoting the treatment strategy.
n_strategies	A scalar denoting the number of unique treatment strategies.
patient_id	A numeric vector of integers denoting the patient.
n_patients	A scalar denoting the number of unique patients.
state_id	A numeric vector of integers denoting the health state.
n_states	A scalar denoting the number of unique health states.
transition_id	A numeric vector denoting the health state transition. This is only used for state transition models.
n_transitions	A scalar denoting the number of unique transitions.
time_id	A numeric vector of integers denoting a unique time interval.
time_intervals	A data.table denoting unique time intervals. Must contain the columns time_id, time_start, and time_stop. time_start is the starting time of an interval and time_stop is the stopping time of an interval. Following the survival package, time intervals are closed on the right and open on the left (except in the final interval where time_stop is equal to infinity).
n_times	A scalar denoting the number of time intervals. Equal to the number of rows in time_intervals.
sample	A numeric vector of integer denoting the sample from the posterior distribution of the parameters.
n_samples	A scalar denoting the number of samples.
grp_id	An optional numeric vector of integers denoting the subgroup.
patient_wt	An optional numeric vector denoting the weight to apply to each patient within a subgroup.

Details

When using the ID variables to index input_mats, sorting order should be the same as specified in `expand.hesim_data()`; that is, observations must be sorted by: (i) `strategy_id`, (ii) `patient_id`, and (iii) the health-related ID variable (either `state_id` or `transition_id`). When using ID variables to index transformed parameter objects and `sample` is used for indexing, then observations must be sorted by: (i) `sample`, (ii) `strategy_id`, (iii) `patient_id`, and (iv) the health-related ID variable.

See Also

`hesim_data()`,`expand.hesim_data()`, input_mats

---

incr_effect *Incremental treatment effect*

Description

Computes incremental effect for all treatment strategies on outcome variables from a probabilistic sensitivity analysis relative to a comparator.

Usage

```
incr_effect(x, comparator, sample, strategy, grp = NULL, outcomes)
```

Arguments

`x`	A `data.frame` or `data.table` containing simulation output with information on outcome variables for each randomly sampled parameter set from a PSA. Each row should denote a randomly sampled parameter set and treatment strategy.
`comparator`	The comparator strategy. If the strategy column is a character variable, then must be a character; if the strategy column is an integer variable, then must be an integer.
`sample`	Character name of column denoting a randomly sampled parameter set.
`strategy`	Character name of column denoting treatment strategy.
`grp`	Character name of column denoting subgroup. If `NULL`, then it is assumed that there is only one group.
`outcomes`	Name of columns to compute incremental changes for.

Value

A `data.table` containing the differences in the values of each variable specified in outcomes between each treatment strategy and the comparator.

Examples

```
# simulation output
n_samples <- 100
sim <- data.frame(sample = rep(seq(n_samples), 4),
             c = c(rlnorm(n_samples, 5, .1), rlnorm(n_samples, 5, .1),
                    rlnorm(n_samples, 11, .1), rlnorm(n_samples, 11, .1)),
             e = c(rnorm(n_samples, 8, .2), rnorm(n_samples, 8.5, .1),
                   rnorm(n_samples, 11, .6), rnorm(n_samples, 11.5, .6)),
             strategy = rep(paste0("Strategy ", seq(1, 2)),
                            each = n_samples * 2),
             grp = rep(rep(c("Group 1", "Group 2"),
                            each = n_samples), 2)
)
# calculate incremental effect of Strategy 2 relative to Strategy 1 by group
ie <- incr_effect(sim, comparator = "Strategy 1", sample = "sample",
                       strategy = "strategy", grp = "grp", outcomes = c("c", "e"))
head(ie)
```

`IndivCtstm` *Individual-level continuous time state transition model*

Description

Simulate outcomes from an individual-level continuous time state transition model (CTSTM) from a fitted multi-state model. The class supports "clock-reset" (i.e., semi-Markov), "clock-forward" (i.e., Markov), and mixtures of clock-reset and clock-forward models as described in IndivCtstmTrans.

Format

An R6::R6Class object.

Public fields

`trans_model` The model for health state transitions. Must be an object of class IndivCtstmTrans.

`utility_model` The model for health state utility. Must be an object of class StateVals.

`cost_models` The models used to predict costs by health state. Must be a list of objects of class StateVals, where each element of the list represents a different cost category.

`disprog_` A `data.table` simulated using $sim_disease() containing the following columns:

- `sample`: A random sample from the PSA.
- `strategy_id`: The treatment strategy ID.
- `patient_id`: The patient ID.
- `from`: The health state ID transitioned from.
- `to`: The health state ID transitioned to.
- `final`: An indicator equal to 1 if a patient is in their final health state during the simulation and 0 otherwise.
- `time_start`: The time at the start of the interval.

- time_stop: The time at the end of the interval.

stateprobs_ An object of class stateprobs simulated using $sim_stateprobs().

qalys_ An object of class qalys simulated using $sim_qalys().

costs_ An object of class costs simulated using $sim_costs().

Methods

Public methods:

- IndivCtstm$new()
- IndivCtstm$sim_disease()
- IndivCtstm$sim_stateprobs()
- IndivCtstm$sim_qalys()
- IndivCtstm$sim_costs()
- IndivCtstm$summarize()
- IndivCtstm$clone()

Method new(): Create a new IndivCtstm object.

Usage:

```
IndivCtstm$new(trans_model = NULL, utility_model = NULL, cost_models = NULL)
```

Arguments:

trans_model The trans_model field.

utility_model The utility_model field.

cost_models The cost_models field.

Returns: A new IndivCtstm object.

Method sim_disease(): Simulate disease progression (i.e., individual trajectories through a multi-state model using an individual patient simulation).

Usage:

```
IndivCtstm$sim_disease(max_t = 100, max_age = 100, progress = NULL)
```

Arguments:

max_t A scalar or vector denoting the length of time to simulate the model. If a vector, must be equal to the number of simulated patients.

max_age A scalar or vector denoting the maximum age to simulate each patient until. If a vector, must be equal to the number of simulated patients.

progress An integer, specifying the PSA iteration (i.e., sample) that should be printed every progress PSA iterations. For example, if progress = 2, then every second PSA iteration is printed. Default is NULL, in which case no output is printed.

Returns: An instance of self with simulated output stored in disprog_.

Method sim_stateprobs(): Simulate health state probabilities as a function of time using the simulation output stored in disprog.

Usage:

```
IndivCtstm$sim_stateprobs(t)
```

Arguments:

`t` TA numeric vector of times.

Returns: An instance of `self` with simulated output of class stateprobs stored in `stateprobs_`.

Method `sim_qalys()`**:** Simulate quality-adjusted life-years (QALYs) as a function of `disprog_` and `utility_model`. See `vignette("expected-values")` for details.

Usage:

```
IndivCtstm$sim_qalys(
  dr = 0.03,
  type = c("predict", "random"),
  lys = TRUE,
  by_patient = FALSE
)
```

Arguments:

`dr` Discount rate.

`type` `"predict"` for mean values or `"random"` for random samples as in $sim() in StateVals.

`lys` If `TRUE`, then life-years are simulated in addition to QALYs.

`by_patient` If `TRUE`, then QALYs are computed at the patient level. If `FALSE`, then QALYs are averaged across patients by health state.

Returns: An instance of `self` with simulated output of class qalys stored in `qalys_`.

Method `sim_costs()`**:** Simulate costs as a function of `disprog_` and `cost_models`. See `vignette("expected-values")` for details.

Usage:

```
IndivCtstm$sim_costs(
  dr = 0.03,
  type = c("predict", "random"),
  by_patient = FALSE,
  max_t = Inf
)
```

Arguments:

`dr` Discount rate.

`type` `"predict"` for mean values or `"random"` for random samples as in $sim() in StateVals.

`by_patient` If `TRUE`, then QALYs are computed at the patient level. If `FALSE`, then QALYs are averaged across patients by health state.

`max_t` Maximum time duration to compute costs once a patient has entered a (new) health state. By default, equal to `Inf`, so that costs are computed over the entire duration that a patient is in a given health state. If time varies by each cost category, then time can also be passed as a numeric vector of length equal to the number of cost categories (e.g., `c(1,2,Inf,3)` for a model with four cost categories).

Returns: An instance of `self` with simulated output of class costs stored in `costs_`.

Method `summarize()`**:** Summarize costs and QALYs so that cost-effectiveness analysis can be performed. See `summarize_ce()`.

Usage:

```
IndivCtstm$summarize(by_grp = FALSE)
```

Arguments:

`by_grp` If `TRUE`, then costs and QALYs are computed by subgroup. If `FALSE`, then costs and QALYs are aggregated across all patients (and subgroups).

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

`create_IndivCtstmTrans()`, IndivCtstmTrans

Examples

```
library("flexsurv")

# Treatment strategies, target population, and model structure
strategies <- data.frame(strategy_id = c(1, 2))
patients <- data.frame(patient_id = seq(1, 3),
                       age = c(45, 50, 60),
                       female = c(0, 0, 1))
states <- data.frame(state_id = c(1, 2))
hesim_dat <- hesim_data(strategies = strategies,
                        patients = patients,
                        states = states)

# Parameter estimation
## Multi-state model
tmat <- rbind(c(NA, 1, 2),
              c(3, NA, 4),
              c(NA, NA, NA))
fits <- vector(length = max(tmat, na.rm = TRUE), mode = "list")
surv_dat <- data.frame(mstate3_exdata$transitions)
for (i in 1:length(fits)){
  fits[[i]] <- flexsurvreg(Surv(years, status) ~ factor(strategy_id),
                           data = surv_dat,
                           subset = (trans == i),
                           dist = "weibull")
}
fits <- flexsurvreg_list(fits)

## Utility
utility_tbl <- stateval_tbl(data.frame(state_id = states$state_id,
                                       mean = mstate3_exdata$utility$mean,
                                       se = mstate3_exdata$utility$se),
                            dist = "beta",
```

```
                              hesim_data = hesim_dat)
## Costs
drugcost_tbl <- stateval_tbl(data.frame(strategy_id = strategies$strategy_id,
                                        est = mstate3_exdata$costs$drugs$costs),
                             dist = "fixed",
                             hesim_data = hesim_dat)
medcost_tbl <- stateval_tbl(data.frame(state_id = states$state_id,
                                       mean = mstate3_exdata$costs$medical$mean,
                                       se = mstate3_exdata$costs$medical$se),
                            dist = "gamma",
                            hesim_data = hesim_dat)

# Economic model
n_samples = 2

## Construct model
### Transitions
transmod_data <- expand(hesim_dat)
transmod <- create_IndivCtstmTrans(fits, input_data = transmod_data,
                                   trans_mat = tmat,
                                   n = n_samples)

### Utility
utilitymod <- create_StateVals(utility_tbl, n = n_samples)

### Costs
drugcostmod <- create_StateVals(drugcost_tbl, n = n_samples)
medcostmod <- create_StateVals(medcost_tbl, n = n_samples)
costmods <- list(drugs = drugcostmod,
                 medical = medcostmod)

### Combine
ictstm <- IndivCtstm$new(trans_model = transmod,
                         utility_model = utilitymod,
                         cost_models = costmods)


## Simulate outcomes
head(ictstm$sim_disease()$disprog_)
head(ictstm$sim_stateprobs(t = c(0, 5, 10))$stateprobs_[t == 5])
ictstm$sim_qalys(dr = .03)
ictstm$sim_costs(dr = .03)
head(ictstm$summarize())
```

`IndivCtstmTrans` *Transitions for an individual-level continuous time state transition model*

Description

Simulate health state transitions in an individual-level continuous time state transition model with parameters that were estimated using a multi-state model.

Format

An R6::R6Class object.

Super class

`hesim::CtstmTrans` -> `IndivCtstmTrans`

Public fields

`params` An object of class params_surv or params_surv_list.

`input_data` Input data used to simulate health state transitions by sample from the probabilistic sensitivity analysis (PSA), treatment strategy and patient. Must be an object of class input_mats. If `params` contains parameters from a list of models (i.e., of class params_surv_list), then `input_data` must contain a unique row for each treatment strategy and patient; if `params` contains parameters from a joint model (i.e., of class params_surv), then `input_data` must contain a unique row for each treatment strategy, patient, and transition.

`trans_mat` A transition matrix describing the states and transitions in a multi-state model in the format from the mstate package. See the documentation for the argument `"trans"` in mstate::msprep.

`start_state` A scalar or vector denoting the starting health state. Default is the first health state. If a vector, must be equal to the number of simulated patients.

`start_age` A scalar or vector denoting the starting age of each patient in the simulation. Default is 38. If a vector, must be equal to the number of simulated patients.

`death_state` The death state in `trans_mat`. Used with `max_age` in `sim_disease` as patients transition to this state upon reaching maximum age. By default, it is set to the final absorbing state (i.e., a row in `trans_mat` with all NAs).

`clock` "reset" for a clock-reset model, "forward" for a clock-forward model, and "mix" for a mixture of clock-reset and clock-forward models. A clock-reset model is a semi-Markov model in which transition rates depend on time since entering a state. A clock-forward model is a Markov model in which transition rates depend on time since entering the initial state. If `"mix"` is used, then `reset_states` must be specified.

`reset_states` A vector denoting the states in which time resets. Hazard functions are always a function of elapsed time since either the start of the model or from when time was previously reset. Only used if `clock = "mix"`.

Methods

Public methods:

- `IndivCtstmTrans$new()`
- `IndivCtstmTrans$sim_stateprobs()`
- `IndivCtstmTrans$check()`
- `IndivCtstmTrans$clone()`

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

Usage:

```
IndivCtstmTrans$new(
  params,
  input_data,
  trans_mat,
  start_state = 1,
  start_age = 38,
  death_state = NULL,
  clock = c("reset", "forward", "mix"),
  reset_states = NULL
)
```

Arguments:

`params` The `params` field.

`input_data` The `input_data` field.

`trans_mat` The `trans_mat` field.

`start_state` The `start_state` field.

`start_age` The `start_age` field.

`death_state` The `death_state` field.

`clock` The `clock` field.

`reset_states` The `reset_states` field.

Returns: A new `IndivCtstmTrans` object.

Method `sim_stateprobs()`**:** Simulate health state probabilities at distinct times by first simulating trajectories through a multi-state model with `IndivCtstm$sim_disease()`.

Usage:

```
IndivCtstmTrans$sim_stateprobs(t, ...)
```

Arguments:

`t` A numeric vector of times.

`...` Additional arguments to pass to `IndivCtstm$sim_disease()`.

Returns: An object of class stateprobs.

Method `check()`**:** Input validation for class. Checks that fields are the correct type.

Usage:

```
IndivCtstmTrans$check()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

`create_IndivCtstmTrans()`, IndivCtstm

Examples

```
library("flexsurv")

# Simulation data
strategies <- data.frame(strategy_id = c(1, 2, 3))
patients <- data.frame(patient_id = seq(1, 3),
                       age = c(45, 50, 60),
                       female = c(0, 0, 1))

# Multi-state model with transition specific models
tmat <- rbind(c(NA, 1, 2),
              c(NA, NA, 3),
              c(NA, NA, NA))
fits <- vector(length = max(tmat, na.rm = TRUE), mode = "list")
for (i in 1:length(fits)){
  fits[[i]] <- flexsurvreg(Surv(years, status) ~ 1,
                           data = bosms3[bosms3$trans == i, ],
                           dist = "exp")
}
fits <- flexsurvreg_list(fits)

# Simulation model
hesim_dat <- hesim_data(strategies = strategies,
                        patients = patients)
fits_data <- expand(hesim_dat)
transmod <- create_IndivCtstmTrans(fits, input_data = fits_data,
                                   trans_mat = tmat,
                                   n = 2,
                                   point_estimate = FALSE)
head(transmod$hazard(c(1, 2, 3)))
head(transmod$cumhazard(c(1, 2, 3)))
transmod$sim_stateprobs(t = c(0, 5, 10))[t == 5]
```

`input_mats` *Input matrices for a statistical model*

Description

Create an object of class `input_mats`, which contains inputs matrices for simulating a statistical model. Consists of (i) input matrices, `X`, and (ii) metadata used to index each matrix in `X`. More details are provided under "Details" below.

Usage

```
input_mats(X, ...)
```

Arguments

X	A list of input matrices for predicting the values of each parameter in a statistical model. May also be a list of lists of input matrices when a list of separate models is fit (e.g., with `flexsurvreg_list()`).
...	Arguments to pass to `id_attributes()`.

Details

Each row of each matrix `X` is an input vector, x_{hik}, where h denotes a health-related index, i indexes a patient, and k is a treatment strategy. A health-related index is either a health state (e.g., `state_id` or a transition between health states (e.g., `transition_id`). In some cases, the health-related index h can be suppressed and separate models can be fit for each health index. This is, for instance, the case in a partitioned survival model where separate models are fit for each survival endpoint.

The rows of the matrices in `X` must be sorted in a manner consistent with the ID variables as described in `id_attributes()`.

See Also

`create_input_mats()`

Examples

```
strategies <- data.frame(strategy_id = c(1, 2))
patients <- data.frame(patient_id = seq(1, 3),
                       age = c(45, 47, 60),
                       female = c(1, 0, 0),
                       group = factor(c("Good", "Medium", "Poor")))
hesim_dat <- hesim_data(strategies = strategies,
                        patients = patients)

dat <- expand(hesim_dat, by = c("strategies", "patients"))
input_mats <- input_mats(X = list(mu = model.matrix(~ age, dat)),
                         strategy_id = dat$strategy_id,
                         n_strategies = length(unique(dat$strategy_id)),
                         patient_id = dat$patient_id,
                        n_patients = length(unique(dat$patient_id)))
print(input_mats)
```

---

mom_beta *Method of moments for beta distribution*

Description

Compute the parameters `shape1` and `shape2` of the beta distribution using method of moments given the mean and standard deviation of the random variable of interest.

Usage

```
mom_beta(mean, sd)
```

Arguments

mean	Mean of the random variable.
sd	Standard deviation of the random variable.

Details

If μ is the mean and σ is the standard deviation of the random variable, then the method of moments estimates of the parameters shape1 = $\alpha > 0$ and shape2 = $\beta > 0$ are:

$$\alpha = \mu \left(\frac{\mu(1-\mu)}{\sigma^2} - 1 \right)$$

and

$$\beta = (1-\mu) \left(\frac{\mu(1-\mu)}{\sigma^2} - 1 \right)$$

Value

A list containing the parameters shape1 and shape2.

Examples

```
mom_beta(mean = .8, sd = .1)
# The function is vectorized.
mom_beta(mean = c(.6, .8), sd = c(.08, .1))
```

---

mom_gamma *Method of moments for gamma distribution*

Description

Compute the shape and scale (or rate) parameters of the gamma distribution using method of moments for the random variable of interest.

Usage

```
mom_gamma(mean, sd, scale = TRUE)
```

Arguments

mean	Mean of the random variable.
sd	Standard deviation of the random variable.
scale	Logical. If TRUE (default), then the scale parameter is returned; otherwise, the rate parameter is returned.

Details

If μ is the mean and σ is the standard deviation of the random variable, then the method of moments estimates of the parameters `shape` = $\alpha > 0$ and `scale` = $\theta > 0$ are:

$$\theta = \frac{\sigma^2}{\mu}$$

and

$$\alpha = \frac{\mu}{\theta}$$

The inverse of the scale parameter, $\beta = 1/\theta$, is the rate parameter.

Value

If `scale` = `TRUE`, then a list containing the parameters `shape` and `scale`; otherwise, if `scale` = `FALSE`, then a list containing the parameters `shape` and `rate`.

Examples

```
mom_gamma(mean = 10000, sd = 2000)
# The function is vectorized.
mom_gamma(mean = c(8000, 10000), sd = c(1500, 2000))
```

`mstate3_exdata` *Example data for a 3-state multi-state model*

Description

Example multi-state data for parameterizing a continuous time state transition model. Costs and utility are also included to facilitate cost-effectiveness analysis.

Usage

```
mstate3_exdata
```

Format

A list containing the following elements:

- transitions A data frame containing the times at which patient transitions between health states based on the prothr dataset from the mstate package.
- costs A list of data frames. The first data frame contains summary medical cost estimates and the second data frame contains drug cost data.
- utility A data frame of summary utility estimates.

Transitions data

The data frame has the following columns:

strategy_id Treatment strategy identification number.

patient_id Patient identification number.

age Patient age (in years).

female 1 if a patient is female; 0 if male.

from Starting state.

to Receiving state.

trans Transition number.

Tstart Starting time.

Tstop Transition time.

years Elapsed years between `Tstart` and `Tstop`.

status Status variable; 1=transition, 0=censored.

Cost data

The cost list contains two data frames. The first data frame contains data on the drug costs associated with each treatment strategy.

strategy_id The treatment strategy identification number.

costs Annualized drug costs.

The second data frame contains summary data on medical costs by health state, and contains the following columns:

state_id The health state identification number.

mean Mean costs.

se Standard error of medical costs.

Utility data

The data frame has the following columns:

state_id The health state identification number.

mean Mean utility

se Standard error of utility

`multinom3_exdata` *Example data for a 3-state multinomial model*

Description

Example discrete time health state transitions data simulated using multinomial logistic regression. Costs and utility are also included to facilitate cost-effectiveness analysis.

Usage

```
multinom3_exdata
```

Format

A list containing the following elements:

- transitions A data frame containing patient transitions between health states at discrete time intervals (i.e., on a yearly basis).
- costs A list of data frames. The first data frame contains drug cost data and the second contains summary medical cost estimates.
- utility A data frame of summary utility estimates.

Transitions data

The data frame has the following columns:

patient_id Patient identification number.

strategy_id Treatment strategy identification number.

strategy_name Treatment strategy name.

age Patient age (in years).

age_cat A factor variable with 3 age groups: (i) age less than 40, (ii) age at least 40 and less than 60, and (iii) age at least 60.

female 1 if a patient is female; 0 if male.

year The year since the start of data collection with the first year equal to 1.

state_from State making a transition from.

state_to State making a transition to.

year_cat Factor variable for year with 3 categories: (i) year 3 and below, (ii) year between 3 and 6, and (iii) year 7 and above.

Cost data

The cost list contains two data frames. The first data frame contains data on the drug costs associated with each treatment strategy.

strategy_id The treatment strategy identification number.

strategy_name The treatment strategy name.

costs Annualized drug costs.

The second data frame contains summary data on medical costs by health state, and contains the following columns:

state_id The health state identification number.

state_name The name of the health state.

mean Mean medical costs.

se Standard error of medical costs.

Utility data

The data frame has the following columns:

state_id The health state identification number.

state_name The name of the health state.

mean Mean utility

se Standard error of utility.

---

multinom_list *List of* multinom *objects*

Description

Combine multinom objects into a list.

Usage

```
multinom_list(...)
```

Arguments

...	Objects of class multinom, which can be named.

Value

An object of class multinom_list.

Examples

```
library("nnet")
library("data.table")
trans_data <- data.table(multinom3_exdata$transitions)
dat_healthy <- trans_data[state_from == "Healthy"]
fit_healthy <- multinom(state_to ~ strategy_name + female + age_cat + year_cat,
                        data = dat_healthy)
dat_sick <- trans_data[state_from == "Sick"]
dat_sick$state_to <- droplevels(dat_sick$state_to)
fit_sick <- multinom(state_to ~ strategy_name + female + age_cat + year_cat,
                     data = dat_sick)
fits <- multinom_list(healthy = fit_healthy, sick = fit_sick)
class(fits)
```

---

params *Parameter object*

Description

Objects prefixed by "params_" are lists containing the parameters of a statistical model used for simulation modeling. The parameters are used to simulate outcomes as a function of covariates contained in input matrices (input_mats).

See Also

tparams

---

params_joined_surv *Parameters of joined survival models*

Description

Create a list containing the parameters of survival models joined at specified time points. See `joined` for more details.

Usage

```
params_joined_surv(..., times)
```

Arguments

`...`	Objects of class `params_surv`, which can be named.
`times`	A numeric vector of times at which to join models.

Value

An object of class "params_joined_surv", which is a list containing two elements:

models A list of params_surv objects from each statistical model to be joined.

times Equivalent to the argument `times`.

Examples

```
library("flexsurv")
fit_exp <- flexsurv::flexsurvreg(formula = Surv(futime, fustat) ~ 1,
                                 data = ovarian, dist = "exp")
fit_wei <- flexsurv::flexsurvreg(formula = Surv(futime, fustat) ~ 1,
                                 data = ovarian, dist = "weibull")
params_surv_exp <- create_params(fit_exp, n = 2)
params_surv_wei <- create_params(fit_wei, n = 2)
params_joined_surv <- params_joined_surv(exp = params_surv_exp,
                                         wei = params_surv_wei,
                                         times = 3)
print(params_joined_surv)
```

---

params_joined_surv_list

Parameters of joined lists of survival models

Description

Create a list containing the parameters of multiple sets of survival models, each joined at specified time points. See joined for more details.

Usage

```
params_joined_surv_list(..., times)
```

Arguments

`...`	Objects of class params_surv_list, which can be named.
`times`	A list of sorted numeric vectors, with the length of each list element equal to the number of sets of models.

Value

An object of class "params_joined_surv_list", which is a list containing two elements:

models A list of params_surv_list, each containing codeparams_surv objects to be joined.

times Equivalent to the argument `times`.

Examples

```
library("flexsurv")
fit_exp <- flexsurv::flexsurvreg(formula = Surv(futime, fustat) ~ 1,
                                 data = ovarian, dist = "exp")
fit_wei <- flexsurv::flexsurvreg(formula = Surv(futime, fustat) ~ 1,
                                 data = ovarian, dist = "weibull")
fit_lnorm <- flexsurv::flexsurvreg(formula = Surv(futime, fustat) ~ 1,
                                  data = ovarian, dist = "lognormal")

params_exp <- create_params(fit_exp, n = 2)
params_wei <- create_params(fit_wei, n = 2)
params_lnorm <- create_params(fit_lnorm, n = 2)

params_list1 <- params_surv_list(params_exp, params_wei)
params_list2 <- params_surv_list(params_exp, params_lnorm)
params_joined <- params_joined_surv_list(model1 = params_list1,
                                         model2 = params_list2,
                                         times = list(3, 5))
print(params_joined)
```

---

params_lm *Parameters of a linear model*

Description

Create a list containing the parameters of a fitted linear regression model.

Usage

```
params_lm(coefs, sigma = NULL)
```

Arguments

coefs	Matrix of samples of the coefficients under sampling uncertainty.
sigma	A vector of samples of the standard error of the regression model. Must only be specified if the model is used to randomly simulate values (rather than to predict means).

Details

Fitted linear models are used to predict values, y, as a function of covariates, x,

$$y = x^T \beta + \epsilon.$$

Predicted means are given by $x^T \hat{\beta}$ where $\hat{\beta}$ is the vector of estimated regression coefficients. Random samples are obtained by sampling the error term from a normal distribution, $\epsilon \sim N(0, \hat{\sigma}^2)$.

Value

An object of class `params_lm`, which is a list containing `coefs`, `sigma`, and `n_samples`. `n_samples` is equal to the number of rows in `coefs`.

Examples

```
library("MASS")
n <- 2
params <- params_lm(coefs = MASS::mvrnorm(n, mu = c(.5,.6),
                                          Sigma = matrix(c(.05, .01, .01, .05), nrow = 2)),
                    sigma <- rgamma(n, shape = .5, rate = 4))
print(params)
```

---

params_mlogit *Parameters of a multinomial logit model*

Description

Store the parameters of a fitted multinomial logistic regression model. The model is used to predict probabilities of K classes.

Usage

```
params_mlogit(coefs)
```

Arguments

`coefs`	A 3D array of stacked matrices. The number of matrices (i.e., the number of slices in the cube) should be equal to $K - 1$. Each matrix is contains samples of the regression coefficients under sampling uncertainty corresponding to a particular class. Rows index parameter samples and columns index coefficients.

Details

Multinomial logit models are used to predict the probability of membership for subject i in each of K classes as a function of covariates:

$$Pr(y_i = c) = \frac{e^{\beta_c x_i}}{\sum_{k=1}^{K} e^{\beta_k x_i}}$$

Value

An object of class `params_mlogit`, which is a list containing `coefs` and `n_samples`, where `n_samples` is equal to the number of rows in each element of `coefs`.

Examples

```
params <- params_mlogit(coefs = array(
  c(matrix(c(intercept = 0, treatment = log(.75)), nrow = 1),
    matrix(c(intercept = 0, treatment = log(.8)), nrow = 1)),
  dim = c(1, 2, 2)
))
```

---

params_mlogit_list *Parameters of a list of multinomial logit models*

Description

Create a list containing the parameters of multiple fitted multinomial logit models.

Usage

```
params_mlogit_list(...)
```

Arguments

...	Objects of class params_mlogit, which can be named.

Value

An object of class `params_mlogit_list`, which is a list containing params_mlogit objects.

---

params_surv *Parameters of a survival model*

Description

Create a list containing the parameters of a single fitted parametric or flexibly parametric survival model.

Usage

```
params_surv(coefs, dist, aux = NULL)
```

Arguments

coefs	A list of length equal to the number of parameters in the survival distribution. Each element of the list is a matrix of samples of the regression coefficients under sampling uncertainty used to predict a given parameter.
dist	Character vector denoting the parametric distribution. See "Details".
aux	Auxiliary arguments used with splines or fractional polynomials. See "Details".

Details

Survival is modeled as a function of L parameters α_l. Letting $F(t)$ be the cumulative distribution function, survival at time t is given by

$$1 - F(t|\alpha_1(x_1), \ldots, \alpha_L(x_L)).$$

The parameters are modeled as a function of covariates, x_l, with an inverse transformation function $g^{-1}()$,

$$\alpha_l = g^{-1}(x_l^T \beta_l).$$

$g^{-1}()$ is typically $exp()$ if a parameter is strictly positive and the identity function if the parameter space is unrestricted.

The types of distributions that can be specified are:

- `exponential` or `exp` Exponential distribution. `coef` must contain the `rate` parameter on the log scale and the same parameterization as in stats::Exponential.
- `weibull` or `weibull.quiet` Weibull distribution. The first element of `coef` is the `shape` parameter (on the log scale) and the second element is the `scale` parameter (also on the log scale). The parameterization is that same as in stats::Weibull.
- `gamma` Gamma distribution. The first element of `coef` is the `shape` parameter (on the log scale) and the second element is the `rate` parameter (also on the log scale). The parameterization is that same as in stats::GammaDist.
- `lnorm` Lognormal distribution. The first element of `coef` is the `meanlog` parameter (i.e., the mean on the log scale) and the second element is the `sdlog` parameter (i.e., the standard deviation on the log scale). The parameterization is that same as in stats::Lognormal.
- `gompertz` Gompertz distribution. The first element of `coef` is the `shape` parameter and the second element is the `rate` parameter (on the log scale). The parameterization is that same as in flexsurv::Gompertz.
- `llogis` Log-logistic distribution. The first element of `coef` is the `shape` parameter (on the log scale) and the second element is the `scale` parameter (also on the log scale). The parameterization is that same as in flexsurv::Llogis.
- `gengamma` Generalized gamma distribution. The first element of `coef` is the location parameter `mu`, the second element is the scale parameter `sigma` (on the log scale), and the third element is the shape parameter `Q`. The parameterization is that same as in flexsurv::GenGamma.
- `survspline` Survival splines. Each element of `coef` is a parameter of the spline model (i.e. `gamma_0`, `gamma_1`, ...) with length equal to the number of knots (including the boundary knots). See below for details on the auxiliary arguments. The parameterization is that same as in flexsurv::Survspline.
- `fracpoly` Fractional polynomials. Each element of `coef` is a parameter of the fractional polynomial model (i.e. `gamma_0`, `gamma_1`, ...) with length equal to the number of powers minus 1. See below for details on the auxiliary arguments (i.e., `powers`).

Auxiliary arguments for spline models should be specified as a list containing the elements:

`knots` A numeric vector of knots.

`scale` The survival outcome to be modeled as a spline function. Options are `"log_cumhazard"` for the log cumulative hazard; `"log_hazard"` for the log hazard rate; `"log_cumodds"` for the log cumulative odds; and `"inv_normal"` for the inverse normal distribution function.

`timescale` If `"log"` (the default), then survival is modeled as a spline function of log time; if `"identity"`, then it is modeled as a spline function of time.

Auxiliary arguments for fractional polynomial models should be specified as a list containing the elements:

`powers` A vector of the powers of the fractional polynomial with each element chosen from the following set: -2. -1, -0.5, 0, 0.5, 1, 2, 3.

Furthermore, when splines (with `scale = "log_hazard"`) or fractional polynomials are used, numerical methods must be used to compute the cumulative hazard and for random number generation. The following additional auxiliary arguments can therefore be specified:

`cumhaz_method` Numerical method used to compute cumulative hazard (i.e., to integrate the hazard function). Always used for fractional polynomials but only used for splines if `scale = "log_hazard"`. Options are `"quad"` for adaptive quadrature and `"riemann"` for Riemann sum.

`random_method` Method used to randomly draw from an arbitrary survival function. Options are `"invcdf"` for the inverse CDF and `"discrete"` for a discrete time approximation that randomly samples events from a bernoulli distribution at discrete times.

`step` Step size for computation of cumulative hazard with numerical integration. Only required when using `"riemann"` to compute the cumulative hazard or using `"discrete"` for random number generation.

Value

An object of class `params_surv`, which is a list containing `coefs`, `dist`, and `n_samples`. n_samplesis equal to the number of rows in each element ofcoefs, which must be the same. The list may also contain aux‘ if a spline or fractional polynomial model is fit.

Examples

```
library("flexsurv")
fit <- flexsurvreg(Surv(futime, fustat) ~ 1, data = ovarian, dist = "weibull")
params <- params_surv(coefs = list(shape = fit$res.t["shape", "est", drop = FALSE],
                                   scale = fit$res.t["scale", "est", drop = FALSE]),
                      dist = fit$dlist$name)
print(params)
```

---

params_surv_list *Parameters of a list of survival models*

Description

Create a list containing the parameters of multiple fitted parametric survival models.

Usage

```
params_surv_list(...)
```

Arguments

...	Objects of class params_surv, which can be named.

Value

An object of class "params_surv_list", which is a list containing params_surv objects.

Examples

```
library("flexsurv")
fit_wei <- flexsurvreg(Surv(futime, fustat) ~ 1, data = ovarian, dist = "weibull")
params_wei <- create_params(fit_wei, n = 2)

fit_exp <- flexsurvreg(Surv(futime, fustat) ~ 1, data = ovarian, dist = "exp")
params_exp <- create_params(fit_exp, n = 2)

params_list <- params_surv_list(wei = params_wei, exp = params_exp)
print(params_list)
```

Psm *N-state partitioned survival model*

Description

Simulate outcomes from an N-state partitioned survival model.

Format

An R6::R6Class object.

Public fields

`survival_models` The survival models used to predict survival curves. Must be an object of class PsmCurves.

`utility_model` The model for health state utility. Must be an object of class StateVals.

`cost_models` The models used to predict costs by health state. Must be a list of objects of class StateVals, where each element of the list represents a different cost category.

`n_states` Number of states in the partitioned survival model.

`t_` A numeric vector of times at which survival curves were predicted. Determined by the argument `t` in $sim_curves().

`survival_` Survival curves simulated using `sim_curves()`.

`stateprobs_` An object of class stateprobs simulated using $sim_stateprobs().

`qalys_` An object of class qalys simulated using $sim_qalys().

`costs_` An object of class costs simulated using $sim_costs().

Methods

Public methods:

- Psm$new()
- Psm$sim_survival()
- Psm$sim_stateprobs()
- Psm$sim_qalys()
- Psm$sim_costs()
- Psm$summarize()
- Psm$clone()

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

Usage:

```
Psm$new(survival_models, utility_model = NULL, cost_models = NULL)
```

Arguments:

`survival_models` The `survival_models` field.

`utility_model` The `utility_model` field.

`cost_models` The `cost_models` field.

Details: `n_states` is set equal to the number of survival models plus one.

Returns: A new `Psm` object.

Method `sim_survival()`**:** Simulate survival curves as a function of time using `PsmCurves$sim_survival()`.

Usage:

```
Psm$sim_survival(t)
```

Arguments:

`t` A numeric vector of times. The first element must be `0`.

Returns: An instance of `self` with simulated output from `PsmCurves$sim_survival()` stored in `stateprobs_`.

Method `sim_stateprobs()`**:** Simulate health state probabilities from `survival_` using a partitioned survival analysis.

Usage:

```
Psm$sim_stateprobs()
```

Returns: An instance of `self` with simulated output of class stateprobs stored in `stateprobs_`.

Method `sim_qalys()`**:** Simulate quality-adjusted life-years (QALYs) as a function of `stateprobs_` and `utility_model`. See `vignette("expected-values")` for details.

Usage:

```
Psm$sim_qalys(
  dr = 0.03,
  integrate_method = c("trapz", "riemann_left", "riemann_right"),
  lys = FALSE
)
```

Arguments:

`dr` Discount rate.

`integrate_method` Method used to integrate state values when computing (QALYs).

`lys` If `TRUE`, then life-years are simulated in addition to QALYs.

Returns: An instance of `self` with simulated output of class qalys stored in `qalys_`.

Method `sim_costs()`**:** Simulate costs as a function of `stateprobs_` and `cost_models`. See `vignette("expected-values")` for details.

Usage:

```
Psm$sim_costs(
  dr = 0.03,
  integrate_method = c("trapz", "riemann_left", "riemann_right")
)
```

Arguments:

`dr` Discount rate.

`integrate_method` Method used to integrate state values when computing costs.

Returns: An instance of `self` with simulated output of class costs stored in `costs_`.

Method `summarize()`**:** Summarize costs and QALYs so that cost-effectiveness analysis can be performed. See `summarize_ce()`.

Usage:

```
Psm$summarize()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

PsmCurves, `create_PsmCurves()`

Examples

```
library("flexsurv")

# Simulation data
strategies <- data.frame(strategy_id = c(1, 2, 3))
patients <- data.frame(patient_id = seq(1, 3),
                       age = c(45, 50, 60),
                       female = c(0, 0, 1))
states <- data.frame(state_id =  seq(1, 3),
                     state_name = paste0("state", seq(1, 3)))
hesim_dat <- hesim_data(strategies = strategies,
                        patients = patients,
```

```
                        states = states)
n_samples <- 3

# Survival models
surv_est_data <- psm4_exdata$survival
fit1 <- flexsurv::flexsurvreg(Surv(endpoint1_time, endpoint1_status) ~ age,
                              data = surv_est_data, dist = "exp")
fit2 <- flexsurv::flexsurvreg(Surv(endpoint2_time, endpoint2_status) ~ age,
                              data = surv_est_data, dist = "exp")
fit3 <- flexsurv::flexsurvreg(Surv(endpoint3_time, endpoint3_status) ~ age,
                              data = surv_est_data, dist = "exp")
fits <- flexsurvreg_list(fit1, fit2, fit3)

surv_input_data <- expand(hesim_dat, by = c("strategies", "patients"))
psm_curves <- create_PsmCurves(fits, input_data = surv_input_data,
                               bootstrap = TRUE, est_data = surv_est_data,
                               n = n_samples)

# Cost model(s)
cost_input_data <- expand(hesim_dat, by = c("strategies", "patients", "states"))
fit_costs_medical <- stats::lm(costs ~ female + state_name,
                               data = psm4_exdata$costs$medical)
psm_costs_medical <- create_StateVals(fit_costs_medical,
                                      input_data = cost_input_data,
                                      n = n_samples)

# Utility model
utility_tbl <- stateval_tbl(tbl = data.frame(state_id = states$state_id,
                                             min = psm4_exdata$utility$lower,
                                             max = psm4_exdata$utility$upper),
                            dist = "unif",
                            hesim_data = hesim_dat)
psm_utility <- create_StateVals(utility_tbl, n = n_samples)

# Partitioned survival decision model
psm <- Psm$new(survival_models = psm_curves,
               utility_model = psm_utility,
               cost_models = list(medical = psm_costs_medical))
psm$sim_survival(t = seq(0, 5, .05))
psm$sim_stateprobs()
psm$sim_costs(dr = .03)
head(psm$costs_)
head(psm$sim_qalys(dr = .03)$qalys_)
```

---

psm4_exdata *Example data for a 4-state partitioned survival model*

Description

A collection of example datasets containing simulated survival, costs, and utility data for a 4-state partitioned survival model.

Usage

```
psm4_exdata
```

Format

A list containing the following elements:

- Survival A data frame containing patient information and time to 3 separate survival endpoints.
- CostsA list of data frames. The first data frame contains medical cost data and the second data frame contains drug cost data.

Survival data

The survival data frame contains a list of 3 survival curves, each containing the following columns.

female An indicator variable equal to 1 if the patient is female and 0 otherwise.

age The age of the patient in years.

strategy_id The id of the treatment strategy used.

endpoint1_time Follow up time with right censored data to survival endpoint 1.

endpoint1_status A status indicator for survival endpoint 1 equal to 0 if alive and 1 if dead.

endpoint2_time Follow up time with right censored data to survival endpoint 2.

endpoint2_status A status indicator for survival endpoint 2 equal to 0 if alive and 1 if dead.

endpoint3_time Follow up time with right censored data to survival endpoint 3.

endpoint3_status A status indicator for survival endpoint 3 equal to 0 if alive and 1 if dead.

Cost data

The cost list contains two data frames.The first data frame contains data on the medical costs by patient and health state, and contains the following columns:

patient_id An integer denoting the id of the patient.

female An indicator variable equal to 1 if the patient is female and 0 otherwise.

state_name A categorical variable denoting the three possible health states.

costs Annualized medical costs.

The second data frame contains data on the drug costs associated with each treatment strategy.

strategy_id The id of each treatment strategy.

costs Annualized drug costs.

PsmCurves *Partitioned survival curves*

Description

Summarize n-1 survival curves for an N state partitioned survival model.

Format

An R6::R6Class object.

Public fields

`params` An object of class params_surv_list.

`input_data` An object of class input_mats. Each row in X must be a unique treatment strategy and patient.

Methods

Public methods:

- `PsmCurves$new()`
- `PsmCurves$hazard()`
- `PsmCurves$cumhazard()`
- `PsmCurves$survival()`
- `PsmCurves$rmst()`
- `PsmCurves$quantile()`
- `PsmCurves$check()`
- `PsmCurves$clone()`

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

Usage:

```
PsmCurves$new(params, input_data)
```

Arguments:

`params` The params field.

`input_data` The input_data field.

Returns: A new PsmCurves object.

Method `hazard()`**:** Predict the hazard function for each survival curve as a function of time.

Usage:

```
PsmCurves$hazard(t)
```

Arguments:

`t` A numeric vector of times.

Returns: A `data.table` with columns `sample`, `strategy_id`, `patient_id`, `grp_id`, `curve` (the curve number), `t`, and `hazard`.

Method `cumhazard()`**:** Predict the cumulative hazard function for each survival curve as a function of time.

Usage:

```
PsmCurves$cumhazard(t)
```

Arguments:

`t` A numeric vector of times.

Returns: A `data.table` with columns `sample`, `strategy_id`, `patient_id`, `grp_id`, `curve`, `t`, and `cumhazard`.

Method `survival()`**:** Predict the cumulative hazard function for each survival curve as a function of time.

Usage:

```
PsmCurves$survival(t)
```

Arguments:

`t` A numeric vector of times.

Returns: A `data.table` with columns `sample`, `strategy_id`, `patient_id`, `grp_id`, `curve`, `t`, and `survival`.

Method `rmst()`**:** Predict the restricted mean survival time up until time points `t` for each survival curve.

Usage:

```
PsmCurves$rmst(t, dr = 0)
```

Arguments:

`t` A numeric vector of times.

`dr` Discount rate.

Returns: A `data.table` with columns `sample`, `strategy_id`, `patient_id`, `grp_id`, `curve`, `t`, and `rmst`.

Method `quantile()`**:** Predict quantiles of the survival distribution for each survival curve.

Usage:

```
PsmCurves$quantile(p)
```

Arguments:

`p` A numeric vector of probabilities for computing quantiles.

Returns: A `data.table` with columns `sample`, `strategy_id`, `patient_id`, `grp_id`, `curve`, `p` and `quantile`.

Method `check()`**:** Input validation for class. Checks that fields are the correct type.

Usage:

```
PsmCurves$check()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Psm, `create_PsmCurves()`

Examples

```
library("flexsurv")

# Simulation data
dt_strategies <- data.frame(strategy_id = c(1, 2, 3))
dt_patients <- data.frame(patient_id = seq(1, 3),
                          age = c(45, 50, 60),
                          female = c(0, 0, 1))
hesim_dat <- hesim_data(strategies = dt_strategies,
                        patients = dt_patients)

# Fit survival models
surv_est_data <- psm4_exdata$survival
fit1 <- flexsurv::flexsurvreg(Surv(endpoint1_time, endpoint1_status) ~ age,
                              data = surv_est_data, dist = "exp")
fit2 <- flexsurv::flexsurvreg(Surv(endpoint2_time, endpoint2_status) ~ age,
                              data = surv_est_data, dist = "exp")
fit3 <- flexsurv::flexsurvreg(Surv(endpoint3_time, endpoint3_status) ~ age,
                              data = surv_est_data, dist = "exp")
fits <- flexsurvreg_list(fit1, fit2, fit3)

# Form PsmCurves
surv_input_data <- expand(hesim_dat, by = c("strategies", "patients"))
psm_curves <- create_PsmCurves(fits, input_data = surv_input_data, n = 3,
                               bootstrap = TRUE, est_data = surv_est_data)

# Summarize survival curves
head(psm_curves$quantile(p = c(.25, .5, .75)))
head(psm_curves$survival(t = seq(0, 3, by = .1)))
head(psm_curves$rmst(t = c(2, 5)))
```

`qalys` *Quality-adjusted life-years object*

Description

An object of class `qalys` returned from methods $sim_qalys() in model classes that store simulated quality-adjusted life-years (QALYs).

Components

A `qalys` object inherits from `data.table` and contains the following columns:

sample A random sample from the PSA.

strategy_id The treatment strategy ID.

patient_id The patient ID.

state_id The health state ID.

dr The rate used to discount QALYs.

category A single category always equal to "qalys".

qalys The simulated values of QALYs.

If the argument `lys = TRUE`, then the `data.table` also contains a column `lys` containing simulated life-years.

---

rcat *Random generation for categorical distribution*

Description

Draw random samples from a categorical distribution given a matrix of probabilities. `rcat` is vectorized and written in C++ for speed.

Usage

```
rcat(n, prob)
```

Arguments

`n`	Number of random observations to draw.
`prob`	A matrix of probabilities where rows correspond to observations and columns correspond to categories.

Value

A vector of random samples from the categorical distribution. The length of the sample is determined by n. The numerical arguments other than n are recycled so that the number of samples is equal to n.

Examples

```
p <- c(.2, .5, .3)
n <- 10000
pmat <- matrix(rep(p, n), nrow = n, ncol = length(p), byrow = TRUE)

# rcat
set.seed(100)
ptm <- proc.time()
samp1 <- rcat(n, pmat)
proc.time() - ptm
prop.table(table(samp1))

# rmultinom from base R
set.seed(100)
ptm <- proc.time()
samp2 <- t(apply(pmat, 1, rmultinom, n = 1, size = 1))
samp2 <- apply(samp2, 1, function(x) which(x == 1))
proc.time() - ptm
prop.table(table(samp2))
```

`rdirichlet_mat` *Random generation for multiple Dirichlet distributions*

Description

Draw random samples from multiple Dirichlet distributions for use in transition probability matrices.

Usage

```
rdirichlet_mat(
  n,
  alpha,
  output = c("array", "matrix", "data.frame", "data.table")
)
```

Arguments

`n`	Number of samples to draw.
`alpha`	A matrix where each row is a separate vector of shape parameters.
`output`	The class of the object returned by the function. Either an `array`, `matrix`, `data.frame`, or `data.table`.

Details

This function is meant for representing the distribution of transition probabilities in a transition matrix. The (i,j) element of `alpha` is a transition from state `i` to state `j`. It is vectorized and written in C++ for speed.

Value

If `output = "array"`, then an array of matrices is returned where each row of each matrix is a sample from the Dirichlet distribution. If `output` results in a two dimensional object (i.e., a `matrix`, `data.frame`, or `data.table`, then each row contains all elements of the sampled matrix from the Dirichlet distribution ordered rowwise; that is, each matrix is flattened. In these cases, the number of rows must be less than or equal to the number of columns.

Examples

```
alpha <- matrix(c(100, 200, 500, 50, 70, 75), ncol = 3, nrow = 2, byrow = TRUE)
samp <- rdirichlet_mat(100, alpha)
print(samp[, , 1:2])
```

---

rng_distributions *Random number generation distributions*

Description

A collection of functions for randomly generating deviates from probability distributions with `define_rng()`.

Usage

```
beta_rng(shape1 = 1, shape2 = 1, mean = NULL, sd = NULL, names = NULL)

dirichlet_rng(alpha, names = NULL)

fixed(est, names = NULL)

custom(x, names = NULL)

gamma_rng(mean, sd, names = NULL)

lognormal_rng(meanlog, sdlog, names = NULL)

multi_normal_rng(mu, Sigma, names = NULL, ...)

normal_rng(mean, sd, names = NULL)

uniform_rng(min, max, names = NULL)
```

Arguments

`shape1, shape2`	Non-negative parameters of the Beta distribution.
`mean, sd`	Mean and standard deviation of the random variable.
`names`	Names for columns if an object with multiple columns is returned by the function.

alpha	A matrix where each row is a separate vector of shape parameters.
est	A vector of estimates of the variable of interest.
x	A numeric vector, matrix, data.frame, or data.table containing random samples of the variable of interest from a suitable probability distribution. This would typically be a posterior distribution from a Bayesian analysis.
meanlog, sdlog	Mean and standard deviation of the distribution on the log scale.
mu, Sigma	mu is a vector giving the means of the variables and Sigma is a positive-definite symmetric matrix specifying the covariance matrix of the variables.
...	Additional arguments to pass to underlying random number generation functions. See "details".
min, max	Lower and upper limits of the distribution. Must be finite.

Details

These functions are not exported and are meant for use with define_rng(). They consequently assume that the number of samples to draw, n, is defined in the parent environment. Convenience random number generation functions include:

beta_rng() If mean and sd are both not NULL, then parameters of the beta distribution are derived using the methods of moments with mom_beta(). Beta variates are generated with stats::rbeta().

custom() Use previously sampled values from a custom probability distribution. There are three possibilities: (i) if n is equal to the number previously sampled values (say n_samples), then x is returned as is; (ii) if n < n_samples, then samples from x are sampled without replacement; and (iii) if n > n_samples, then samples from x are sampled with replacement and a warning is provided.

dirichlet_rng() Dirichlet variates for each row in the matrix are generated with rdirichlet_mat(). The sampled values are stored in a data.table where there is a column for each element of alpha (with elements ordered rowwise).

fixed() This function should be used when values of the variable of interest are fixed (i.e., they are known with certainty). If length(est) > 1, an n by length(est) data.table is returned meaning that each element of est is repeated n times; otherwise (if length(est) == 1), a vector is returned where est is repeated n times.

gamma_rng() The parameters of the gamma distribution are derived using the methods of moments with mom_gamma() and gamma variates are generated with stats::rgamma().

lognormal_rng() Lognormal variates are generated with stats::rlnorm().

multi_normal_rng() Multivariate normal variates are generated with MASS::mvrnorm().

normal_rng() Normal variates are generated with stats::rnorm().

uniform_rng() Uniform variates are generated with stats::runif().

Value

Functions either return a vector of length n or an n by k data.table. Multivariate distributions always return a data.table. If a univariate distribution is used, then a data.table is returned if each parameter is specified as a vector with length greater than 1; otherwise, if parameters are

scalars, then a vector is returned. In the `data.table` case, `k` is equal to the length of the parameter vectors entered as arguments. For example, if the probability distribution contained `mean` as an argument and `mean` were of length 3, then an `n` by 3 matrix would be returned. The length of all parameter vectors must be the same. For instance, if the vector `mean` were of length 3 then all additional parameters (e.g., `sd`) must also be of length 3.

If a `data.table` is returned by a distribution, then its column names are set according to the following hierarchy:

1. With the `names` argument if it is not `NULL`
2. With the names of the parameter vectors if they are named vectors. If there are multiple parameter vector arguments, then the names of the first parameter vector with non `NULL` names is used. For instance, if `mean` and `sd` are both arguments to a random number generation function and `mean` is a named vector, then the names from the vector `mean` are used.
3. As `v1`, ..., `vk` if the `names` argument is `NULL` and there are no named parameter vectors.

See Also

`define_rng()`

`rpwexp` *Random generation for piecewise exponential distribution*

Description

Draw random samples from an exponential distribution with piecewise rates. `rpwexp` is vectorized and written in C++ for speed.

Usage

```
rpwexp(n, rate = 1, time = 0)
```

Arguments

`n`	Number of random observations to draw.
`rate`	A matrix of rates where rows correspond to observations and columns correspond to rates during specified time intervals.
`time`	A vector equal to the number of columns in `rate` giving the times at which the rate changes

Value

A vector of random samples from the piecewise exponential distribution. The length of the sample is determined by `n`. The numerical arguments other than `n` are recycled so that the number of samples is equal to `n`.

Examples

```
rate <- c(.6, 1.2, 1.3)
n <- 100000
ratemat <- matrix(rep(rate, n/2), nrow = n,
                  ncol = 3, byrow = TRUE)
t <- c(0, 10, 15)
ptm <- proc.time()
samp <- rpwexp(n, ratemat, t)
proc.time() - ptm
summary(samp)
```

---

stateprobs *State probability object*

Description

An object of class `stateprobs` returned from methods $sim_stateprobs() in model classes.

Components

A `stateprobs` object inherits from `data.table` and contains the following columns:

sample A random sample from the PSA.

strategy_id The treatment strategy ID.

patient_id The patient ID.

state_id The health state ID.

t The time at which a state probability is computed.

prob The probability of being in a given health state.

When simulating individual-level models, the `patient_id` column is not included as state probabilities are computed by averaging across patients.

StateVals *Model for state values*

Description

Simulate values (i.e., utility or costs) associated with health states in a state transition or partitioned survival model.

Public fields

`params` Parameters for simulating state values. Currently supports objects of class tparams_mean or params_lm.

`input_data` An object of class input_mats. Only used for params_lm objects.

`method` The method used to simulate costs and quality-adjusted life-years (QALYs) as a function of state values. If `wlos`, then costs and QALYs are simulated by weighting state values by the length of stay in a health state. If `starting`, then state values represent a one-time value that occurs when a patient enters a health state. When `starting` is used in a cohort model, the state values only accrue at time 0; in contrast, in an individual-level model, state values accrue each time a patient enters a new state and are discounted based on time of entrance into that state.

`time_reset` If `FALSE` then time intervals are based on time since the start of the simulation. If `TRUE`, then time intervals reset each time a patient enters a new health state. This is relevant if, for example, costs vary over time within health states. Only used if `method = wlos`.

Methods

Public methods:

- `StateVals$new()`
- `StateVals$sim()`
- `StateVals$check()`
- `StateVals$clone()`

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

Usage:

```
StateVals$new(
  params,
  input_data = NULL,
  method = c("wlos", "starting"),
  time_reset = FALSE
)
```

Arguments:

`params` The `params` field.

`input_data` The `input_data` field.

`method` The `method` field.

`time_reset` The `time_reset` field.

Returns: A new `StateVals` object.

Method `sim()`**:** Simulate state values with either predicted means or random samples by treatment strategy, patient, health state, and time `t`.

Usage:

```
StateVals$sim(t, type = c("predict", "random"))
```

Arguments:

`t` A numeric vector of times.

type "predict" for mean values or "random" for random samples.

Returns: A data.table of simulated state values with columns for sample, strategy_id, patient_id, state_id, time, and value.

Method check(): Input validation for class. Checks that fields are the correct type.

Usage:

```
StateVals$check()
```

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

---

stateval_tbl *Table to store state value parameters*

Description

Create a table for storing parameter estimates used to simulate costs or utility in an economic model by treatment strategy, patient, health state, and (optionally) time interval.

Usage

```
stateval_tbl(
  tbl,
  dist = c("norm", "beta", "gamma", "lnorm", "unif", "fixed", "custom"),
  hesim_data = NULL,
  grp_var = NULL
)
```

Arguments

tbl	A data.frame or data.table for storing parameter values. See "Details" for specifics.
dist	Probability distribution used to sample parameters for a probabilistic sensitivity analysis (PSA).
hesim_data	A hesim_data object. Required to specify treatment strategies , patients, and/or health states not included as columns in tbl, or, to match patients in tbl to groups. Not required if tbl includes one row for each treatment strategy, patient, and health state combination. Patients are matched to groups by specifying both a patient_id and a grp_var column in the patients table.
grp_var	The name of the variable used to group patients.

Details

`tbl` is a `data.table` containing columns for treatment strategies (`strategy_id`), patients (`patient_id`), health states (`state_id`), and/or the start of time intervals (`time_start`). The table must contain at least one column named `strategy_id`, `patient_id`, or `state_id`, but does not need to contain all of them. Each row denotes a unique treatment strategy, patient, health state, and/or time interval pair. `tbl` may also contain a column with the name specified in `grp_var` (rather than `patient_id`) so that state values are assigned to groups of patients.

`tbl` must also contain columns summarizing the state values for each row, which depend on the probability distribution selected with `dist`. Available distributions include the normal (`norm`), beta (`beta`), gamma (`gamma`), lognormal (`lnorm`), and uniform (`unif`) distributions. In addition, the option `fixed` can be used if estimates are known with certainty and `custom` can be used if parameter values for a PSA have been previously sampled from an arbitrary probability distribution. The columns in `tbl` that must be included, by distribution, are:

norm `mean` and `sd`

beta `mean` and `se` or `shape1` and `shape2`

gamma `mean` and `se`, `shape` and `rate`, or `shape` and `scale`

lnorm `meanlog` or `sdlog`

unif `min` and `max`

fixed `est`

custom `sample` and `value`

Note that if `dist = "custom"`, then `tbl` must include a column named `sample` (an integer vector denoting a unique random draw) and `value` (denoting the value of the randomly sampled parameter). In this case, there is a unique row in `tbl` for each random draw (`sample`) and each combination of strategies, patients, health states, and/or time intervals. Again, `tbl` must contain at least one column named `strategy_id`, `patient_id` (or `grp_var`), or `state_id`, but does not need to contain them all.

Value

An object of class "stateval_tbl", which is a `data.table` of parameter values with attributes for `dist` and optionally `strategy_id`, `patients`, `state_id`, and `grp_var`. `tbl` is in the same format as described in "Details". `patients` is a `data.table` with one column containing `patient_id` and optionally a second column containing `grp_var`.

See Also

create_StateVals, StateVals

Examples

```
strategies <- data.frame(strategy_id = c(1, 2))
patients <- data.frame(patient_id = seq(1, 3),
                       grp = c(1, 1, 2),
                       age = c(45, 50, 60),
                       female = c(0, 0, 1))
```

```
states <- data.frame(state_id = c(1, 2))
hesim_dat <- hesim_data(strategies = strategies,
                        patients = patients,
                        states = states)

# Utility varies by health state and patient group
utility_tbl <- stateval_tbl(data.frame(state_id = rep(states$state_id, 2),
                                       grp = rep(rep(c(1, 2)), each = nrow(states)),
                                       mean = c(.8, .7, .75, .55),
                                       se = c(.18, .12, .10, .06)),
                            dist = "beta",
                            hesim_data = hesim_dat,
                            grp_var = "grp")
print(utility_tbl)
utilmod <- create_StateVals(utility_tbl, n = 2)

# Costs vary by treatment strategy
cost_tbl <- stateval_tbl(data.frame(strategy_id = strategies$strategy_id,
                                    mean = c(5000, 3000),
                                    se = c(200, 100)),
                         dist = "gamma",
                         hesim_data = hesim_dat)
print(cost_tbl)
costmod <- create_StateVals(cost_tbl, n = 2)
```

`summarize_ce` *Summarize costs and effectiveness*

Description

Summarize costs and quality-adjusted life-years (QALYs) given output simulated from an economic model. The summary output is used to perform cost-effectiveness analysis with `icea()` and `icea_pw()`.

Usage

```
summarize_ce(costs, qalys, by_grp = FALSE)
```

Arguments

`costs`	Simulated costs by category (objects of class costs).
`qalys`	Simulated QALYs (objects of class qalys).
`by_grp`	If `TRUE`, then costs and QALYs are computed by subgroup. If `FALSE`, then costs and QALYs are aggregated across all patients (and subgroups).

Details

If mean costs and/or QALYs have already been computed (i.e., an average within a population), then there must be one observation for each discount rate (`dr`), PSA sample (`sample`), treatment strategy (`strategy_id`), and health state (`state_id`). Alternatively, there can be a column denoting a patient (`patient_id`), in which case outcomes will first be averaged across patients. A `grp_id` column can also be used so that outcomes are computed for each subgroup (if `by_grp = TRUE`); otherwise it is assumed that there is only one subgroup.

Value

An object of class ce.

`surv_quantile` *Survival quantiles*

Description

Compute quantiles from survival curves.

Usage

```
surv_quantile(x, probs = 0.5, t, surv_cols, by)
```

Arguments

`x`	A `data.table` or `data.frame`.
`probs`	A numeric vector of probabilities with values in `[0,1]`.
`t`	A character scalar of the name of the time column.
`surv_cols`	A character vector of the names of columns containing survival curves.
`by`	A character vector of the names of columns to group by.

Value

A `data.table` of quantiles of each survival curve in `surv_cols` by each group in `by`.

Examples

```
library("data.table")
t <- seq(0, 10, by = .01)
surv1 <- seq(1, .3, length.out = length(t))
surv2 <- seq(1, .2, length.out = length(t))
strategies <- c("Strategy 1", "Strategy 2")
surv <- data.table(strategy = rep(strategies, each = length(t)),
                   t = rep(t, 2),
                   surv = c(surv1, surv2))
surv_quantile(surv, probs = c(.4, .5), t = "t",
              surv_cols = "surv", by = "strategy")
```

time_intervals *Time intervals*

Description

Create a table of time intervals given a vector or data frame of unique times. This would typically be passed to id_attributes.

Usage

```
time_intervals(times)
```

Arguments

times	Either a vector of times for each interval or a data.frame with at least one column named time_start.

Value

An object of class time_intervals that inherits from data.table in the same format as time_intervals as described in id_attributes.

See Also

id_attributes

Examples

```
time_intervals(c(0, 3, 5))
time_intervals(data.frame(time_start = c(0, 3, 5),
                          time_cat = c("Time <= 3", "3 < Time <= 5",
                                       "Time > 5")))
```

tparams *Transformed parameter object*

Description

Objects prefixed by "tparams_" are lists containing transformed parameters used to simulate outcomes. The parameters have presumably already been transformed as a function of input data and consequently do not need to be used alongside input matrices. In other words, transformed parameters are parameters that have already been predicted as a function of covariates.

See Also

params

`tparams_mean` *Predicted means*

Description

Create a list containing means predicted from a statistical model.

Usage

```
tparams_mean(value, ...)
```

Arguments

`value`	Matrix of samples from the distribution of the mean. Columns denote random samples and rows denote means for different observations.
`...`	Arguments to pass to id_attributes. Each row in `value` must be a prediction for a `strategy_id`, `patient_id`, `state_id`, and optionally `time_id` combination.

Value

An object of class `tparams_mean`, which is a list containing `value`, `n_samples`, and the ID attributes passed to id_attributes.

See Also

tparams

Examples

```
tparams_mean(value = matrix(1:8, nrow = 4),
             strategy_id = rep(1:2, each = 2),
             n_strategies = 2,
             patient_id = rep(1, 4),
             n_patients = 1,
             state_id = rep(1:2, times = 2),
             n_states = 2)
```

`tparams_transprobs` *Transition probabilities*

Description

Create a list containing predicted transition probabilities at discrete times. Since the transition probabilities have presumably already been predicted based on covariate values, no input data is required for simulation. The class can be instantiated from either an array, a data table, or a data frame.

Usage

```
tparams_transprobs(object, ...)

## S3 method for class 'array'
tparams_transprobs(object, times = NULL, grp_id = NULL, patient_wt = NULL)

## S3 method for class 'data.table'
tparams_transprobs(object)

## S3 method for class 'data.frame'
tparams_transprobs(object)
```

Arguments

`object`	An object of the appropriate class.
`...`	Further arguments passed to or from other methods. Currently unused.
`times`	An optional numeric vector of distinct times to pass to time_intervals representing time intervals indexed by the 4th dimension of the array. May either be the start or the end of intervals. This argument is not required if there is only one time interval.
`grp_id`	An optional numeric vector of integers denoting the subgroups. Must be the same length as the 3rd dimension of the array.
`patient_wt`	An optional numer vector denoting the weight to apply to each patient within a subgroup. Must be the same length as the 3rd dimension of the array.

Details

The format of `object` depends on its class:

array Must be a 4D array of matrices (i.e., a 6D array). The dimensions of the array should be indexed as follows: 1st (`sample`), 2nd (`strategy_id`), 3rd (`patient_id`), 4th (`time_id`), 5th (rows of transition matrix), and 6th (columns of transition matrix). In other words, an index of [s, k, i, t] represents the transition matrix for the `s`th sample, `k`th treatment strategy, `i`th patient, and `t`th time interval.

data.table Must contain the following:

- ID columns for the parameter sample (`sample`), treatment strategy (`strategy_id`), and patient (`patient_id`). If the number of time intervals is greater than 1 it must also contain the column `time_start` denoting the starting time of a time interval. A column `patient_wt` may also be used to denote the weight to apply to each patient.
- Columns for each element of the transition probability matrix. They should be prefixed with "probs_" and ordered rowwise. For example, the following columns would be used for a 2x2 transition probability matrix: `probs_1` (1st row, 1st column), `probs_2` (1st row, 2nd column), `probs_3` (2nd row, 1st column), and `probs_4` (2nd row, 2nd column).

data.frame Same as `data.table`.

A `tparams_transprobs` object is also instantiated when creating a cohort discrete time state transition model using `define_model()`.

Value

An object of class `tparams_transprobs`, which is a list containing `value` and relevant ID attributes. The element `value` is an array of predicted transition probability matrices from the probability distribution of the underlying statistical model. Each matrix in `value` is a prediction for a `sample`, `strategy_id`, `patient_id`, and optionally `time_id` combination.

See Also

`define_model()`, `create_CohortDtstm()`

---

tpmatrix *Transition probability matrix*

Description

`tpmatrix()` both defines and evaluates a transition probability matrix in which elements are expressions. This function is used within `define_tparams()` to create a transition probability matrix used for simulation modeling.

Usage

```
tpmatrix(...)
```

Arguments

`...`	Named values of expressions defining elements of the matrix. The parameter values of the matrix elements should refer to parameters defined using `define_rng()` or `define_tparams()`.

Details

The matrix is filled rowwise, meaning that each row should sum to 1. It is evaluated in the environment used by `eval_tparams()` so that any objects available within `define_tparams()` can be used by `tpmatrix()`. The complementary probability equal to 1 minus the sum of the probabilities of all other rows can be conveniently referred to as `C`.

Value

Returns a `data.table` where each column is an element of the transition probability matrix with elements ordered rowwise.

See Also

`define_model()`, `define_tparams()`

Examples

```
p <- c(.7, .6)
tpmatrix(
  C, p,
  0, 1
)
```

`tpmatrix_names` *Names for elements of a transition probability matrix*

Description

Create names for all elements of a transition probability matrix given names for the health states. This is useful for flattening a transition probability matrix (rowwise) into a vector and naming the resulting vector. The name of an element of the flattened vector representing a transition from the ith state to the jth state is of the form `paste0(prefix,state_i,sep,state_j)`.

Usage

```
tpmatrix_names(states, prefix = "p_", sep = "_")
```

Arguments

`states`	A character vector of the names of health states in the transition matrix.
`prefix`	A prefix that precedes the described transitions between states.
`sep`	A character string to separate the terms representing state `i` and state `j`.

Value

A character vector containing a name for each element of the transition probability matrix encompassing all possible transitions.

Examples

```
tpmatrix_names(LETTERS[1:4])
tpmatrix_names(LETTERS[1:4], prefix = "")
tpmatrix_names(LETTERS[1:4], prefix = "", sep = ".")
```

weibullNMA *Parameterization of the Weibull distribution for network meta-analysis*

Description

Density, distribution function, hazards, quantile function and random generation for the Weibull distribution when parameterized for network meta-analysis.

Usage

```
dweibullNMA(x, a0, a1 = FALSE, log = FALSE)

pweibullNMA(q, a0, a1, lower.tail = TRUE, log.p = FALSE)

qweibullNMA(p, a0, a1, lower.tail = TRUE, log.p = FALSE)

rweibullNMA(n, a0, a1)

hweibullNMA(n, a0, a1, log = FALSE)

HweibullNMA(n, a0, a1, log = FALSE)

rmst_weibullNMA(t, a0, a1, start = 0)

mean_weibullNMA(a0, a1)
```

Arguments

x, q	Vector of quantiles
a0	Intercept of reparameterization of the Weibull distribution.
a1	Slope of the reparameterization of the Weibull distribution.
log, log.p	logical; if TRUE, probabilities p are given as log(p).
lower.tail	logical; if TRUE (default), probabilities are $P(X \leq x)$, otherwise, $P(X > x)$.
p	Vector of probabilities
n	Number of observations. If length(n) > 1, the length is taken to be the number required.
t	Vector of times for which restricted mean survival time is evaluated.
start	Optional left-truncation time or times. The returned restricted mean survival will be conditional on survival up to this time.

Value

dweibullNMA gives the density, pweibullNMA gives the distribution function, qweibullNMA gives the quantile function, rweibullNMA generates random deviates, HweibullNMA returns the cumulative hazard and hweibullNMA the hazard.

See Also

dweibull

Index