Package 'hesim'

March 29, 2019

Type Package

Title Health-Economic Simulation Modeling and Decision Analysis

Version 0.2.0

Description Parameterize, simulate, and analyze health-economic simulation models. Supports N-state partitioned survival models (Glasziou et al. 1990) <doi:10.1002/sim.4780091106> and continuous time state transition models (Siebert et al. 2012) <doi:10.1016/j.jval.2012.06.014> parameterized using survival or multi-state modeling (de Wreede et al. 2011, Jackson 2015) <doi:10.18637/jss.v038.i07>, <10.18637/jss.v070.i08>. 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>. Simulation code written in C++ to boost performance.

URL https://github.com/InnovationValueInitiative/hesim

BugReports https://github.com/InnovationValueInitiative/hesim/issues

License GPL-3

LazyData true

LinkingTo Rcpp, RcppArmadillo

Depends R (>= 3.2.3)

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

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

VignetteBuilder knitr

RoxygenNote 6.1.1

SystemRequirements C++11

NeedsCompilation yes

Author Devin Incerti [aut, cre], Jeroen P. Jansen [aut]

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

Repository CRAN

Date/Publication 2019-03-28 23:10:10 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.

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` suitable for use with hesim_data.

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_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* `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 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", "patients", and "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

Returns an `R6Class` object of class `StateVals`.

See Also

`StateVals`

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)
```

ctstm3_exdata *Example data for a 3-state continuous time state transition model*

Description

A collection of example datasets containing health state transition, costs, and utility data for a 3-state continuous time state transition model.

Usage

```
ctstm3_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 dataset prothr 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 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.

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

strategy_id The treatment strategy identification number.

costs Annualized drug costs.

Utility data

The data frame has the following columns:

state_id The health state identification number.

mean Mean costs.

se Standard error.

`expand.hesim_data` *Expand hesim_data*

Description

Expand the data tables from an object of class `hesim_data` into a data table in long format (one row for each combination of observations as specified with the ID variables from the tables specified with the by argument). See "Details" for an explanation of how the expansion is done.

Usage

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

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.

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`. The supplied tables are determined using the by argument. The resulting dataset is sorted by prioritizing ID variables as follows: (i) `strategy_id`, (ii) `line`, (iii) `patient_id`, (iv) the health-related ID variable (either `state_id` or `transition_id`), and (v) the time interval (i.e., `time_id`).

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.

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"))
```

`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

Return an object of class "flexsurvreg_list" from multiple objects of class flexsurvreg.

Usage

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

Arguments

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

Value

Returns 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)
```

formula_list *List of* formula *objects*

Description

Create an object of class "formula_list". The object can be created from either multiple objects of class formula or from another "formula_list" object.

Usage

```
formula_list(...)
```

Arguments

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

Value

Returns an object of class "formula_list".

Examples

```
# Create from "formula" objects
flist_wei <- formula_list(shape = formula(~ 1), scale = formula(~ x))
class(flist_wei)

# Create from "formula_list" objects
flist <- formula_list(exponential = formula_list(rate = formula(~1)),
                              weibull = flist_wei)
```

`hesim` *hesim: an R package for high performance health-economic simulation modeling*

Description

hesim: an R package for high performance health-economic simulation modeling

---

`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, lines = NULL, states = NULL,
  transitions = NULL, times = 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 patient observations. 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. Other columns are variables describing the characteristics of a patient.
`lines`	A table of treatment lines used for each treatment strategy. Must contain the columns `strategy_id`, denoting a treatment strategy, and `line`, denoting a treatment line. Other columns are variables describing the characteristics of a treatment line for a given treatment strategy. A column denoting the treatment used for a given strategy and line would often be specified. Not currently supported.

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.
times	A table of time intervals. Must contain the column time_start, which denotes the starting time of times intervals defined between time_start and time_stop. hesim_data() automatically creates the columns time_stop and time_id (a vector of integers sorted by time_start). Time intervals are closed on the left and open on the right and the final time interval is defined from time_start to infinity.

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))
times <- data.frame(time_id = c(1, 2, 3),
                    time_start = c(0, 4, 9),
                    time_stop = c(4, 9, Inf))
hesim_dat <- hesim_data(strategies = strategies,
                        patients = patients,
                        states = states,
                        times = times)
```

---

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 containing four data.tables:

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 containing four data.tables:

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

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). 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`.

Usage

```
IndivCtstm
```

Format

`R6Class` object.

Fields

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

`utility_model` The model used to predict utility by health state. 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_` A `data.table` of health state probabilities as a function of time simulated using `sim_stateprobs`. See the description of the output of `sim_stateprobs` in `IndivCtstmTrans`.

`qalys_` A `data.table` of quality-adjusted life-years (QALYs) simulated using `sim_qalys`. Columns are:

- **sample** A random sample from the probabilistic sensitivity analysis (PSA).
- **strategy_id** The treatment strategy ID.
- **patient_id** A patient ID.

dr The discount rate.

qalys Simulated QALYs.

lys Simulated life-years.

Note that the `lys` column is only computed if `lys = TRUE`.

`costs_` A `data.table` of costs by category simulated using `sim_costs`. Columns are:

sample A random sample from the PSA.

strategy_id The treatment strategy ID.

patient_id A patient ID.

dr The discount rate.

category The cost category.

costs Simulated costs.

Methods

`new(trans_model = NULL, utility_model = NULL, cost_models = NULL)` Constructor for the class.

`sim_disease(max_t = 100, max_age = 100, progress = NULL)` Simulate disease progression.

- `max_t`: A scalar or vector denoting the number of time periods 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_`.

`sim_stateprobs(t)` Simulate the probability of being in each health state as a function of time using the simulation output stored in `disprog_`.

- `t`: A numeric vector of times.

Returns an instance of self with simulated output stored in `stateprobs_`.

`sim_qalys(dr = .03, type = c("predict", "random"), by_patient = FALSE, lys = TRUE)` Compute simulated (mean discounted) QALYs using the simulation output stored in `disprog_` by random sample from the PSA, treatment strategy, health state, and (optionally) patient.

- `dr`: Discount rate to apply to QALYs. May be a vector in which case QALYs are calculated for each element in `dr`.
- `type`: `predict` for mean values or `random` for random samples as in `$sim()` in `StateVals`.
- `by_patient`: If `TRUE`, QALYs are computed at the patient level. If `FALSE`, QALYs are averaged across patients by health state.
- `lys`: If `TRUE`, then life-years are computed in addition to QALYs.

Returns an instance of self with simulated output stored in `qalys_`.

`sim_costs(dr = .03, type = c("predict", "random"), by_patient = FALSE, max_t = Inf)` Compute simulated (mean discounted) costs using the simulation output stored in `disprog_` by random sample from the PSA, treatment strategy, health state, and (optionally) patient.

- dr: Discount rate to apply to costs. May be a vector in which case costs are calculated for each element in dr.
- type: predict for mean values or random for random samples as in $sim() in StateVals.
- by_patient: If TRUE, costs are computed at the patient level. If FALSE, costs 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 4 cost categories).

Returns an instance of self with simulated output stored in costs_. ,

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

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(ctstm3_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$state_id,
                                       mean = ctstm3_exdata$utility$mean,
```

```
                                    se = ctstm3_exdata$utility$se),
                            dist = "beta",
                            hesim_data = hesim_dat)
## Costs
drugcost_tbl <- stateval_tbl(data.frame(strategy_id = strategies$strategy_id,
                                    est = ctstm3_exdata$costs$drugs$costs),
                            dist = "fixed",
                            hesim_data = hesim_dat)
medcost_tbl <- stateval_tbl(data.frame(state_id = states$state_id,
                                    mean = ctstm3_exdata$costs$medical$mean,
                                    se = ctstm3_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

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

Usage

```
IndivCtstmTrans
```

Format

R6Class object.

Fields

`input_mats` Input matrices used to simulate health state transitions by treatment strategy and patient. Must be an object of class `input_mats`. If `params` is a list of models, then `input_mats` must contain a unique row for each treatment strategy and patient; if `params` is a joint model, then `input_mats` must contain a unique row for each treatment strategy, patient, and transition.

`params` An object of class `params_surv` or `params_surv_list`.

`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 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.

`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"`.

`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).

Methods

`new(input_mats, params, trans_mat, start_state = 1, start_age = 38, death_state = NULL, clock = c("` Constructor for the class.

`hazard(t)` Predict the hazard functions for each health state transition.

- `t`: A numeric vector of times.

`cumhazard(t)` Predict the cumulative hazard functions for each health state transition.

- t: A numeric vector of times.

,

`sim_stateprobs(t, ...)` Simulate health state probabilities at distinct times.

- t: A numeric vector of times.
- ...: Additional arguments to pass to `sim_disease`.

Returns a `data.table` with the following columns:

sample A random sample from the PSA.

strategy_id The treatment strategy 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.

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

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, (ii) ID variables indexing the rows of each matrix in X, and (iii) the dimensions of the X matrices. More details are provided under "Details" below. Note that an "input_mats" object should be created using create_input_mats.

Usage

```
input_mats(X, strategy_id, n_strategies, patient_id, n_patients,
  line = NULL, n_lines = NULL, state_id = NULL, n_states = NULL,
  transition_id = NULL, n_transitions = NULL, time_id = NULL,
  time_intervals = NULL, n_times = NULL, time_fun = NULL)
```

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).
strategy_id	A numeric vector of integers denoting the treatment strategy represented by each row in X.
n_strategies	A scalar denoting the number of unique treatment strategies.
patient_id	A numeric vector of integers denoting the patient represented by each row in X.
n_patients	A scalar denoting the number of unique patients.
line	A numeric vector of integers denoting the treatment line represented by each row in X. Not supported by currently available models.
n_lines	A data.table denoting the number of treatment lines associated with each treatment strategy. Should contain a column, "strategy_id", and a column, "N". Not supported by currently available models.
state_id	A numeric vector of integers denoting the health state represented by each row in X.
n_states	A scalar denoting the number of unique health states.
transition_id	A numeric vector denoting the health state transition represented by each row in X. This must only be specified when estimating the health state transitions with a joint likelihood function. If independent models are fit for each transition, then separate X matrices must be specified for each transition. Note that this is not currently supported but will be supported once hesim provides support for state transition modeling.
n_transitions	A scalar denoting the number of unique transitions. Not supported by currently available models.
time_id	A numeric vector of integers denoting a unique time interval.

`time_intervals` A `data.table` denotes 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. Time intervals are closed on the left and open on the right, and in the final interval, `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`.

`time_fun` A pointer to a C++ functor that can be used to update `X` as a function of time in a simulation model. Not currently supported.

Details

Each row of each matrix `X` is an input vector, x_{hijk}, where h denotes a health-related index, i indexes a patient, j indexes a treatment line, 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. Likewise, models can be fit without multiple treatment lines as would, again, be the case in a partitioned survival analysis where sequential treatment would be incorporated by adding additional health states rather than by using the index j.

The rows of the matrices in `X` must be sorted in a manner consistent with the ID variables. The sorting order should be the same as specified in `expand.hesim_data`; that is, the rows of `X` must be sorted by: (i) `strategy_id`, (ii) `line`, (iii) `patient_id`, and (iv) the health-related ID variable (either `state_id` or `transition_id`).

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))
```

`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 from the posterior distribution of the regression coefficients.
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_mean *Parameters of a mean model*

Description

Create a list containing the parameters of a mean model.

Usage

```
params_mean(mu, sigma = NULL)
```

Arguments

mu	Matrix of samples from the posterior distribution of the mean. Columns denote random samples and rows denote means for different observations.
sigma	A vector of samples of the standard deviation.

Details

The mean model is given by,

$$y_j = \mu_j + \epsilon,$$

where μ_j is the mean value for observation j. Predicted means are consequently given by $\hat{\mu}_j$, which is an estimate of μ_j from the available data. 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_mean", which is a list containing mu, sigma, and n_samples. n_samples is equal to the number of columns in mu.

Examples

```
params <- params_mean(mu = matrix(seq(1, 4), nrow = 2),
                      sigma = c(0, 0))
print(params)
```

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 from the posterior distribution of the regression coefficients 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 `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 `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 `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 `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 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 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 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 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 "sample" for randomly sampling from discrete survival probabilities.

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

Value

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

Usage

```
Psm
```

Format

R6Class object.

Fields

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

`utility_model` The model used to predict utility by health state. 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 generated using `sim_curves`.

`stateprobs_` Health state probabilities as a function of time generated using `sim_stateprobs`.

`costs_` Total (discounted) costs by type generated using `sim_costs`.

`qalys_` Total (discounted) quality-adjusted life-years (QALYs) generated using `sim_qalys`.

Methods

`new(survival_models, utility_model = NULL, cost_models = NULL)` Constructor for the class. Note that the number of health states, `n_states`, is set equal to the number of survival models plus one.

`sim_survival(t)` Simulate survival curves as a function of time. Equivalent to the member function `survival` in PsmCurves.

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

`sim_stateprobs()` Simulate the probability of being in each of N health states using the survival curves generated from `sim_curves`.

`sim_qalys(dr = .03)` Simulate (discounted) QALYs over the times selected in `t` associated with each health state based on the state probabilities calculated using `sim_stateprobs`. See "Details".

- dr: Discount rate to apply to QALYs. May be a vector in which case QALYs are calculated for each element in dr.

`sim_costs(dr = .03)` Simulate (discounted) costs for each cost type over the times selected in t associated with each health state based on the state probabilities calculated using sim_stateprobs. See "Details".

- dr: Discount rate to apply to costs. May be a vector in which case costs for each cost type are calculated for each element in dr.

,

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

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

Details

Discounted costs and QALYs are calculated by integrating the "weighted" probability of being in each state. Weights are a function of the discount factor and the state value predicted using either the cost or QALY model. Mathematically, discounted costs and QALYs in health state s are calculated as,

$$\int_0^T w_h e^{-rt} P_h(t) dt$$

,

where for health state h and time t, w_h is the predicted cost or QALY weight, r is the discount rate, and $P_h(t)$ is the probability of being in a given health state. The integral is calculated numerically using the composite trapezoid rule from the points in t_.

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.

Usage

```
PsmCurves
```

Format

R6Class object.

Fields

`input_mats` Input matrices used to predict state values by strategy and patient. Must be an object of class input_mats where each row of a matrix in "X" is a unique strategy and patient. All matrices in "X" must be sorted by strategy and patient.

`params` An object of class params_surv_list.

Methods

`new(input_mats, params)` Constructor for the class.

`hazard(t)` Predict the hazard function as a function of time.

- t: A numeric vector of times.

`cumhazard(t)` Predict the cumulative hazard function as a function of time.

- t: A numeric vector of times.

`survival(t)` Predict the survival function as a function of time.

- t: A numeric vector of times.

`rmst(t, dr = 0)` Predict (discounted) restricted mean survival time.

- t: A numeric vector of times.
- dr: Discount rate.

`quantile(p)` Predict quantiles of the survival distributions.

- p: A numeric vector of probabilities for calculating quantiles.

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

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)))
```

`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. `rdirichlet_mat` is vectorized and written in C++ for speed.

Usage

```
rdirichlet_mat(n, alpha)
```

Arguments

`n`	Number of samples to draw.
`alpha`	A matrix where each row is a separate vector of shape parameters.

Details

This function is particularly useful for representing the distribution of transition probabilities in a transition matrix.

Value

An array of matrices where each row of each matrix is a sample from the Dirichlet distribution.

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])
```

---

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)
```

StateVals *Model for state values*

Description

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

Usage

```
StateVals
```

Format

R6Class object.

Fields

`input_mats` Input matrices used to simulate state values by strategy, patient, and health state. Must be an object of class input_mats where each row in the matrix "X" is a unique strategy, patient, and health state. "X" must be sorted by strategy, patient, and health state.

`params` An object of class params_lm, which contains the parameters for simulating state values.

Methods

`new(input_mats, params)` Constructor for the class.

`sim(t, type = c("predict", "random"))` Simulate state values with either predicted means or random samples by treatment strategy, patient, and health state.

- `t`: A numeric vector of times. The first element must be 0.
- `type`: `"predict"` for mean values or `"random"` for random samples.

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

Examples

```
# 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))
dt_states <- data.frame(state_id =  seq(1, 3),
                           state_name = paste0("state", seq(1, 3)))
hesim_dat <- hesim_data(strategies = dt_strategies,
                              patients = dt_patients,
                              states = dt_states)

# Create StateVals object
fit_costs_medical <- stats::lm(costs ~ female + state_name, data = psm4_exdata$costs$medical)
```

```
dat <- expand(hesim_dat, by = c("strategies", "patients", "states"))
costs_medical <- create_StateVals(fit_costs_medical, input_data = dat, n = 5)

# Predict
head(costs_medical$sim(t = 5, type = "predict"))
```

`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)
```

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_id` column in the `patients` table; if `hesim_data = NULL` but `grp_id` is included as a column in `tbl`, then each group is assumed to be a unique patient.

Details

`tbl` is a `data.table` containing columns for treatment strategies (`strategy_id`), patient subgroups (`grp_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`, `grp_id`, or `state_id`, but does not need to contain all of them. Each row denotes a unique treatment strategy, patient subgroup, health state, and/or time interval pair.

`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, grp_id, 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, and state_id. 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_id.

See Also

create_StateVals

Examples

```
strategies <- data.frame(strategy_id = c(1, 2))
patients <- data.frame(patient_id = seq(1, 3),
                       grp_id = 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_id = 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)
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)
```

---

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")
```

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.
...	Additional arguments to pass to random sampling functions.

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