Package 'heemod'

February 24, 2019

Title Markov Models for Health Economic Evaluations

Version 0.9.4

Description An implementation of the modelling and reporting features described in reference textbook and guidelines (Briggs, Andrew, et al. Decision Modelling for Health Economic Evaluation. Oxford Univ. Press, 2011; Siebert, U. et al. State-Transition Modeling. Medical Decision Making 32, 690-700 (2012).): deterministic and probabilistic sensitivity analysis, heterogeneity analysis, time dependency on state-time and model-time (semi-Markov and non-homogeneous Markov models), etc.

Depends R (>= 3.3.0)

Imports dplyr (>= 0.7.2), ggplot2 (>= 2.2.0), lazyeval (>= 0.2.0), memoise (>= 1.1.0), mvnfast (>= 0.2.2), plyr (>= 1.8.0), pryr (>= 0.1.2), tibble (>= 1.3.0)

License GPL (>= 3)

LazyData true

VignetteBuilder knitr

RoxygenNote 6.1.1

Suggests BCEA, diagram, flexsurv, knitr, logitnorm, lpSolve, mgcv, optimx, parallel, readxl, rgho, rmarkdown, stringr, survival, testthat, triangle, XLConnect

URL https://github.com/pierucci/heemod, https://pierucci.org

BugReports https://github.com/pierucci/heemod/issues

NeedsCompilation no

Author Kevin Zarca [aut, cre],
Antoine Filipovic-Pierucci [aut],
Matthew Wiener [ctb],
Zdenek Kabat [ctb],
Vojtech Filipec [ctb],
Jordan Amdahl [ctb],
Yonatan Carranza Alarcon [ctb],
Vince Daniels [ctb]

Maintainer Kevin Zarca <kevin.zarca@gmail.com>

Repository CRAN

Date/Publication 2019-02-24 17:00:07 UTC

R topics documented:

add_hazards . . . 3
apply_af . . . 4
apply_hr . . . 4
apply_or . . . 5
apply_shift . . . 6
calibrate_model . . . 6
cluster . . . 8
combine_probs . . . 9
compute_surv . . . 10
construct_part_surv_tib . . . 10
define_calibration_fn . . . 11
define_correlation . . . 12
define_dsa . . . 13
define_inflow . . . 13
define_init . . . 14
define_parameters . . . 14
define_part_surv . . . 16
define_psa . . . 17
define_spline_survival . . . 18
define_starting_values . . . 19
define_state . . . 20
define_strategy . . . 21
define_survival . . . 22
define_surv_table . . . 23
define_transition . . . 23
dispatch_strategy . . . 26
distributions . . . 26
export_savi . . . 28
get_counts.updated_model . . . 29
get_values.updated_model . . . 30
heemod . . . 30
join . . . 31
load_surv_models . . . 31
look_up . . . 32
mix . . . 33
modify . . . 34
part_survs_from_surv_inputs . . . 35
plot.dsa . . . 35
plot.psa . . . 36
plot.run_model . . . 37
plot.surv_obj . . . 38

probability . . . 39
reindent_transition . . . 40
rescale_discount_rate . . . 41
run_bcea . . . 42
run_dsa . . . 42
run_model . . . 44
run_model_tabular . . . 46
run_psa . . . 47
set_covariates . . . 49
summary.run_model . . . 50
summary.surv_shift . . . 50
update_model . . . 51
who_mortality . . . 53

Index **55**

`add_hazards` *Add Hazards*

Description

Get a survival distribution reflecting the independent hazards from two or more survival distributions.

Usage

```
add_hazards(...)

add_hazards_(dots)
```

Arguments

`...`	Survival distributions to be used in the projection.
`dots`	Used to work around non-standard evaluation.

Value

A `surv_add_haz` object.

Examples

```
dist1 <- define_survival(distribution = "exp", rate = .125)
dist2 <- define_survival(distribution = "weibull", shape = 1.2, scale = 50)
combined_dist <- add_hazards(dist1, dist2)
```

apply_af *Apply an Acceleration Factor*

Description

Proportionally increase or reduce the time to event of a survival distribution.

Usage

```
apply_af(dist, af, log_af = FALSE)
```

Arguments

dist	A survival distribution.
af	An acceleration factor to be applied.
log_af	If TRUE, the accleration factor is exponentiated before being applied.

Value

A surv_aft object.

Examples

```
dist1 <- define_survival(distribution = "exp", rate = .25)
aft_dist <- apply_af(dist1, 1.5)
```

apply_hr *Apply a Hazard Ratio*

Description

Proportional reduce or increase the hazard rate of a distribution.

Usage

```
apply_hr(dist, hr, log_hr = FALSE)
```

Arguments

dist	A survival distribution.
hr	A hazard ratio to be applied.
log_hr	If TRUE, the hazard ratio is exponentiated before being applied.

Value

A surv_ph object.

Examples

```
dist1 <- define_survival(distribution = "exp", rate = .25)
ph_dist <- apply_hr(dist1, 0.5)
```

---

apply_or *Apply an Odds Ratio*

Description

Proportionally increase or reduce the odds of an event of a survival distribution.

Usage

```
apply_or(dist, or, log_or = FALSE)
```

Arguments

dist	A survival distribution.
or	An odds ratio to be applied.
log_or	If TRUE, the odds ratio is exponentiated before being applied.

Value

A surv_po object.

Examples

```
dist1 <- define_survival(distribution = "exp", rate = .25)
po_dist <- apply_or(dist1, 1.2)
```

apply_shift *Apply a time shift to a survival distribution*

Description

Apply a time shift to a survival distribution

Usage

```
apply_shift(dist, shift)
```

Arguments

dist	A survival distribution.
shift	A time shift to be applied.

Details

A positive shift moves the fit backwards in time. That is, a shift of 4 will cause time 5 to be evaluated as time 1, and so on. If `shift == 0`, `dist` is returned unchanged.

Value

A `surv_shift` object.

Examples

```
dist1 <- define_survival(distribution = "gamma", rate = 0.25, shape = 3)
shift_dist <- apply_shift(dist1, 4)
compute_surv(dist1, 1:10)
compute_surv(shift_dist, 1:10)
```

calibrate_model *Calibrate Model Parameters*

Description

Search for the appropriate value of unknown parameters to obtain specific model results.

Usage

```
calibrate_model(x, parameter_names, fn_values, target_values,
  initial_values = NULL, method = c("Nelder-Mead", "BFGS", "L-BFGS-B"),
  ...)
```

Arguments

x	Result from run_model() or update().
parameter_names	Names of the parameters to calibrate.
fn_values	Function applied to the model that returns the values of interest as a numeric vector.
target_values	Values to match, same length as the output from fn_values.
initial_values	Optional starting values. See details.
method	Optimisation method (Nelder-Mead, BFGS, or L-BFGS-B).
...	Optional arguments passed to optimx::optimx().

Details

Parameters not being optimized are unchanged from the values in the model run. If initial_values is NULL, the initial parameter values will also be taken from the model run.

initial_values can be a vector or a table. In the second case each row corresponds to a set of initial parameter values: the calibration will be run once per set.

Passing in multiple initial values allows (among other things) the user to check whether the calibration gets the same results from different starting points.

Multi-dimensional problems are optimized with optimx::optimx(), 1-dimensional problems with stats::optimise() (except when a method is given). convcode is always NA with stats::optimise().

Running calibrate_model() does not change the model parameters; the user must create a new model and run it if desired.

See also vignette("k-calibration").

Value

A data frame in which each row has the calibrated values of parameters given in parameter_names, for the corresponding row of initial_values, along with the convergence code for each run.

Examples

```
param <- define_parameters(p = 0.8)

mat <- define_transition(
  p, C,
  0, 1
)
mod <- define_strategy(
  transition = mat,
  A = define_state(cost=10, effect = 0.5),
  B = define_state(cost = 5, effect = 0.8)
)

res_mod <- run_model(
```

```
  mod = mod,
  parameters = param,
  init = c(1000L, 0L),
  cycles = 10,
  cost = cost,
  effect = effect,
  method = "end"
)

f <- function(x) {
  dplyr::filter(
    get_counts(x),
    state_names == "A" & markov_cycle == 10
  )$count
}
f(res_mod)

calibrate_model(
  res_mod,
  parameter_names = "p",
  fn_values = f,
  target_values = 130,
  initial_values = data.frame(p = c(0.5, 0.9)),
  lower = 0, upper = 1
)
```

`cluster` *Run* heemod *on a Cluster*

Description

These functions create or delete a cluster for heemod. When the cluster is created it is automagically used by heemod functions.

Usage

```
use_cluster(num_cores, cluster = NULL, close = TRUE)

status_cluster(verbose = TRUE)

close_cluster()
```

Arguments

num_cores	Number of core.
cluster	A custom cluster. See details.
close	Close existing cluster before defining a new one?
verbose	Print cluster info.

Details

The usual wokflow is to create the cluster with `use_cluster`, then run functions such as `run_psa()` that make use of the cluster. To stop using the cluster run `close_cluster()`.

The cluster status is given by `status_cluster`.

A custom cluster can be passed to `use_cluster` with the `cluster` argument. This custom custer needs to work with `parallel::parLapply()`.

Value

`use_cluster` and `close_cluster` return `TRUE` invisibly in case of success. `status_cluster` returns `TRUE` if a cluster is defined, `FALSE` otherwise.

`combine_probs` *Combine Probabilities*

Description

Given several independent probabilities of an event, return the final probability of the event.

Usage

```
combine_probs(...)
```

Arguments

`...`	Probability vectors.

Details

This function is only correct if the probabilities are independent!

Value

A probability vector.

Examples

```
(p1 <- runif(5))
(p2 <- runif(5))
combine_probs(p1, p2)
```

compute_surv *Evaluate Survival Distributions*

Description

Generate either survival probabilities or conditional probabilities of event for each model cycle.

Usage

```
compute_surv(x, time, cycle_length = 1, type = c("prob", "survival"),
  ...)
```

Arguments

x	A survival distribution object
time	The model_time or state_time for which to predict.
cycle_length	The value of a Markov cycle in absolute time units.
type	Either prob, for transition probabilities, or surv, for survival.
...	arguments passed to methods.

Details

The results of compute_surv() are memoised for options("heemod.memotime") (default: 1 hour) to increase resampling performance.

Value

Returns either the survival probalities or conditional probabilities of event for each cycle.

construct_part_surv_tib *construct a survival object from tabular specification*

Description

construct a survival object from tabular specification

Usage

```
construct_part_surv_tib(surv_def, ref, state_names, env = new.env())
```

Arguments

surv_def	a data frame with the specification. See details.
ref	data frame with information about the fits.
state_names	names of the model states
env	an environment

Details

This function is meant to be used only from within tabular_input.R. It won't work well otherwise, in that the environment is unlikely to have what you need.

columns of surv_def: .strategy, .type, .subset, dist, until where dist can be either the name of a distribution along with parameters, or a reference to a fit for example: fit('exp') or exp(rate = 0.5)

Value

a list with one element for each strategy. Each element is in turn a part_surv object, a list with two elements, pfs and os. And those elements are survival objects of various kinds, with the commonality that they can be used in compute_surv().

---

define_calibration_fn *Define Calibration Function*

Description

Define a function to be passed to the fn_values argument of calibrate_model().

Usage

```
define_calibration_fn(type, strategy_names, element_names, cycles,
  groups = NULL, aggreg_fn = sum)
```

Arguments

type	Type of model values (count or value).
strategy_names	Names of strategies.
element_names	Names of states (for counts) or of state values (for values).
cycles	Cycles of interest.
groups	Optional grouping of values (values in a same group have the same groups).
aggreg_fn	A function to aggregate values in a same group.

Value

A numeric vector.

Examples

```
example("run_model")

f <- define_calibration_fn(
  type = c("count", "count", "value"),
  strategy_names = c("I", "I", "II"),
  element_names = c("A", "B", "ly"),
  cycles = c(3, 5, 9),
  groups = c(1, 1, 2),
  aggreg_fn = mean
)
```

`define_correlation` *Define a Correlation Structure for Probabilistic Uncertainty Analysis*

Description

Not all correlation need to be specified for all variable combinations, unspecified correlations are assumed to be 0.

Usage

```
define_correlation(...)

define_correlation_(.dots)
```

Arguments

`...`	A list of parameter names and correlation coeficients of the form `var1, var2, cor(var1, var2), var3`
`.dots`	Used to work around non-standard evaluation.

Value

An object of class `correlation_matrix`.

Examples

```
cm <- define_correlation(
    var1, var2, .4,
    var1, var3, -.2,
    var2, var3, .1
  )
```

define_dsa *Define a Sensitivity Analysis*

Description

Define parameter variations for a Markov model sensitivity analysis.

Usage

```
define_dsa(...)

define_dsa_(par_names, low_dots, high_dots)
```

Arguments

`...`	A list of parameter names and min/max values of the form `var1, min(var1), max(var1), var2, min(`
`par_names`	String vector of parameter names.
`low_dots, high_dots`	Used to work around non-standard evaluation.

Value

A `sensitivity` object.

Examples

```
define_dsa(
  a, 10, 45,
  b, .5, 1.5
)
```

define_inflow *Define Inflow for a BIA*

Description

Define Inflow for a BIA

Usage

```
define_inflow(...)

define_inflow_(.dots)
```

Arguments

...	Name-value pairs of expressions defining inflow counts.
.dots	Used to work around non-standard evaluation.

Value

An object similar to the return value of `define_parameters()`.

---

define_init *Define Initial Counts*

Description

Define Initial Counts

Usage

```
define_init(...)

define_init_(.dots)
```

Arguments

...	Name-value pairs of expressions defining initial counts.
.dots	Used to work around non-standard evaluation.

Value

An object similar to the return value of `define_parameters()`.

---

define_parameters *Define Markov Model Parameters*

Description

Define parameters called to compute the transition matrix or state values for a Markov model. Parameters can be time dependent by using the `markov_cycle` parameter.

Usage

```
define_parameters(...)

define_parameters_(.dots)

## S3 method for class 'uneval_parameters'
modify(.OBJECT, ...)
```

Arguments

...	Name-value pairs of expressions defining parameters.
.dots	Used to work around non-standard evaluation.
.OBJECT	An object of class uneval_parameters.

Details

Parameters are defined sequentially, parameters defined earlier can be called in later expressions.

Vector length should not be explicitly set, but should instead be stated relatively to markov_cycle (whose length depends on the number of simulation cycles). Alternatively, dplyr functions such as dplyr::n() or dplyr::row_number() can be used.

This function relies heavily on the dplyr package. Parameter definitions should thus mimic the use of functions such as dplyr::mutate().

Variable names are searched first in the parameter definition (only parameters defined earlier are visible) then in the environment where define_parameters was called.

For the modify function, existing parameters are modified, but no new parameter can be added. Parameter order matters since only parameters defined earlier can be referenced in later expressions.

Value

An object of class uneval_parameters (actually a named list of lazy expressions).

Examples

```
# parameter 'age' depends on time:
# simulating a cohort starting at 60 yo

define_parameters(
  age_start = 60,
  age = age_start + markov_cycle
)

# other uses of markov_cycle are possible

define_parameters(
  top_time = ifelse(markov_cycle < 10, 1, 0)
)

# more elaborate: risk function

define_parameters(
  rate = 1 - exp(- markov_time * .5)
)

# dont explicitly state lengths
# define_parameters(
#   var = seq(1, 15, 2)
# )
```

```
# instead rely on markov_cycle or dplyr
# functions such as n() or row_number()

define_parameters(
  var = seq(from = 1, length.out = n(), by = 3),
  var2 = seq(1, length(markov_cycle), 2)
)

param <- define_parameters(
  age_start = 60,
  age = age_start + markov_cycle
)

# modify existing parameters

modify(
  param,
  age_start = 40
)

# cannot add new parameters

# modify(
#   param,
#   const = 4.4,
#   age_2 = age ^ 2
# )
```

`define_part_surv` *Define Partitioned Survival*

Description

Define a partitioned survival model with progression-free survival and overall survival.

Usage

```
define_part_surv(pfs, os, state_names, terminal_state = FALSE,
  cycle_length = 1)

define_part_surv_(pfs, os, state_names, cycle_length = 1)
```

Arguments

`pfs, os`	Either results from `flexsurv::flexsurvreg()` or `define_survival()`.

state_names	named character vector, length 3 or 4. State names for progression-free state, progression, (optionally terminal) and death respectively. Elements should be named "progression_free", "progression", (optionally "terminal"), and "death". See examples.
terminal_state	Should a terminal state be included? Only used when state names are not provided.
cycle_length	The value of a Markov cycle in absolute time units.

Value

A part_surv object.

Examples

```
dist_pfs <- define_survival("exp", rate = 1)
dist_os <- define_survival("exp", rate = .5)

define_part_surv(
  pfs = dist_pfs,
  os = dist_os,
  state_names = c(
    progression_free = "A",
    progression = "B",
    terminal = "C",
    death = "D"
  )
)
# identical to:
define_part_surv(
  pfs = dist_pfs,
  os = dist_os,
  terminal_state = TRUE
)
```

---

define_psa *Define Parameters Distribution for Probabilistic Analysis*

Description

Define the properties of parameter distributions and their correlation structure for probabilistic uncertainty analysis of Markov models.

Usage

```
define_psa(..., correlation)

define_psa_(.dots = list(), correlation)
```

Arguments

`...`	Formulas defining parameter distributions.
`correlation`	A correlation matrix for parameters or the output of `define_correlation()`.
`.dots`	Pair/values of expressions coercible to lazy objects.

Details

The distributions must be defined within heemod (see distributions), or defined with `define_distribution()`.

If no correlation matrix is specified parameters are assumed to be independant.

The correlation patrix need only be specified for correlated parameters.

Value

An object of class `resamp_definition`. Contains `list_qdist`, a list of quantile functions and `correlation` a correlation matrix.

Examples

```
mc <- define_correlation(
  age_init, cost_init, .4
)

define_psa(
    age_init ~ normal(60, 10),
    cost_init ~ normal(1000, 100),
    correlation = mc
)

# example with multinomial parameters

define_psa(
  rate1 + rate2 + rate3 ~ multinomial(10, 50, 40),
  a + b ~ multinomial(15, 30)
)
```

`define_spline_survival` *Define a Restricted Cubic Spline Survival Distribution*

Description

Define a restricted cubic spline parametric survival distribution.

Usage

```
define_spline_survival(scale = c("hazard", "odds", "normal"), ...)
```

Arguments

scale	"hazard", "odds", or "normal", as described in flexsurvspline. With the default of no knots in addition to the boundaries, these models reduce to the Weibull, log-logistic and log-normal respectively. The scale must be common to all times.
...	Additional distribution parameters (see respective distribution help pages).

Value

A surv_dist object.

Examples

```
define_spline_survival(
  scale = "hazard",
  gamma = c(-18.3122, 2.7511, 0.2292),
  knots=c(4.276666, 6.470800, 7.806289)
)
define_spline_survival(
  scale = "odds",
  gamma = c(-18.5809, 2.7973, 0.2035),
  knots=c(4.276666, 6.470800, 7.806289)
)
```

---

define_starting_values

Define Starting State Values

Description

Define Starting State Values

Usage

```
define_starting_values(...)

define_starting_values_(.dots)
```

Arguments

...	Name-value pairs of expressions defining starting values.
.dots	Used to work around non-standard evaluation.

Value

An object similar to the return value of define_parameters().

`define_state` *Define a Markov Model State*

Description

Define the values characterising a Markov Model state for 1 cycle.

Usage

```
define_state(...)

define_state_(.dots)

## S3 method for class 'state'
modify(.OBJECT, ...)
```

Arguments

`...`	Name-value pairs of expressions defining state values.
`.dots`	Used to work around non-standard evaluation.
`.OBJECT`	An object of class `state`.

Details

As with `define_parameters()`, state values are defined sequentially. Later state definition can thus only refer to values defined earlier.

For the `modify` function, existing values are modified, no new values can be added. Values order matters since only values defined earlier can be referenced in later expressions.

Value

An object of class `state` (actually a named list of `lazy` expressions).

Examples

```
st <- define_state(
  cost = 6453,
  utility = .876
)
st
```

`define_strategy` *Define a Markov Model*

Description

Combine information on parameters, transition matrix and states defined through `define_parameters()`, `define_transition()` and `define_state()` respectively.

Usage

```
define_strategy(..., transition = define_transition(),
  starting_values = define_starting_values())

define_strategy_(transition, states, starting_values)
```

Arguments

`...`	Object generated by `define_state()`.
`transition`	An object generated by `define_transition()`.
`starting_values`	Optional starting values defined with `define_starting_values()`.
`states`	List of states, only used by `define_strategy_` to avoid using `....`

Details

This function checks whether the objects are compatible in the same model (same state names...).

State values and transition probabilities referencing `state_time` are automatically expanded to implicit tunnel states.

Value

An object of class `uneval_model` (a list containing the unevaluated parameters, matrix and states).

Examples

```
mat <- define_transition(
  state_names = c("s1", "s2"),
  1 / c, 1 - 1/ c,
  0, 1
)

s1 <- define_state(
  cost = 234,
  utility = 1
  )
s2 <- define_state(
  cost = 421,
```

```
  utility = .5
  )

define_strategy(
  transition = mat,
  s1 = s1,
  s2 = s2
)
```

`define_survival` *Define a Survival Distribution*

Description

Define a parametric survival distribution.

Usage

```
define_survival(distribution = c("exp", "weibull", "weibullPH", "lnorm",
  "llogis", "gamma", "gompertz", "gengamma", "gengamma.orig", "genf",
  "genf.orig"), ...)
```

Arguments

`distribution`	A parametric survival distribution.
`...`	Additional distribution parameters (see respective distribution help pages).

Value

A `surv_dist` object.

Examples

```
define_survival(distribution = "exp", rate = .5)
define_survival(distribution = "gompertz", rate = .5, shape = 1)
```

define_surv_table *Define a survival distribution based on explicit survival probabilities*

Description

Define a survival distribution based on explicit survival probabilities

Usage

```
define_surv_table(x)

## S3 method for class 'data.frame'
define_surv_table(x)

## S3 method for class 'character'
define_surv_table(x)
```

Arguments

x	a data frame with columns `time` and `survival`

Value

a `surv_table` object, which can be used with `compute_surv()`.

Examples

```
x <- data.frame(time = c(0, 1, 5, 10), survival = c(1, 0.9, 0.7, 0.5))
define_surv_table(x)
```

define_transition *Define Transition Matrix for Markov Model*

Description

Define a matrix of transition probabilities. Probability can depend on parameters defined with `define_parameters()`, and can thus be time-dependent.

Usage

```
define_transition(..., state_names)

define_transition_(.dots, state_names)

## S3 method for class 'uneval_matrix'
modify(.OBJECT, ...)

## S3 method for class 'uneval_matrix'
plot(x, relsize = 0.75, shadow.size = 0,
  latex = TRUE, ...)
```

Arguments

`...`	Name-value pairs of expressions definig matrix cells. Can refer to parameters defined with `define_parameters()`. For `plot`, additional arguments passed to `diagram::plotmat()`.
`state_names`	character vector, optional. State names.
`.dots`	Used to work around non-standard evaluation.
`.OBJECT`	An object of class `uneval_matrix`.
`x`	An `uneval_matrix` to plot.
`relsize`	Argument passed to `diagram::plotmat()`.
`shadow.size`	Argument passed to `diagram::plotmat()`.
`latex`	Argument passed to `diagram::plotmat()`.

Details

Matric cells are listed by row.

Parameters names are searched first in a parameter object defined with `define_parameters()` and linked with the matrix through `define_strategy()`; then in the environment where the matrix was defined.

The complementary probability of all other row probabilities can be conveniently referred to as `C`.

The matrix code can be re-indented for readability with `reindent_transition()`.

Only matrix size is checked during this step (the matrix must be square). Other conditions (such as row sums being equal to 1) are tested later, during model evaluation.

For the `modify` function, existing matrix cells are replaced with the new expression. Cells are referenced by name. Cell naming follows the `cell_x_y` convention, with `x` being the row number and `y` the column number.

Value

An object of class `uneval_matrix` (actually a named list of `lazy` expressions).

Examples

```
# simple 3x3 transition matrix

mat_1 <- define_transition(
  .2, 0, .8,
  0, .1, .9,
  0, 0, 1
)
mat_1

plot(mat_1)

# referencing parameters
# rr must be present in a parameter object
# that must later be linked with define_strategy

mat_2 <- define_transition(
  .5 - rr, rr,
  .4, .6
)
mat_2

reindent_transition(mat_2)

# can also use C

define_transition(
  C, rr,
  .4, .6
)

# updating cells from mat_1

modify(
  mat_1,
  cell_2_1 = .2,
  cell_2_3 = .7
)

# only matrix size is check, it is thus possible
# to define an incorrect matrix

# this matrix will generate an error later,
# during model evaluation

define_transition(
  .5, 3,
  -1, 2
)
```

dispatch_strategy *Dispatch Values According to Strategy*

Description

Returns different values depending on the strategy.

Usage

```
dispatch_strategy(.strategy, ...)
```

Arguments

.strategy	Optional strategy name. If not specified it is implicitely added.
...	Values of the parameter named depending on the strategy.

Value

A vector of values.

Examples

```
define_parameters(
  val = 456,
  x = dispatch_strategy(
    strat_1 = 1234,
    strat_2 = 9876,
    strat_3 = val * 2 + markov_cycle
  )
)
```

distributions *Probability Density Functions for Probabilistic Uncertainty Analysis*

Description

Define a distribution for PSA parameters.

Usage

```
normal(mean, sd)

lognormal(mean, sd, meanlog, sdlog)

gamma(mean, sd)

binomial(prob, size)

multinomial(...)

logitnormal(mu, sigma)

beta(shape1, shape2)

triangle(lower, upper, peak = (lower + upper)/2)

poisson(mean)

define_distribution(x)

beta(shape1, shape2)

triangle(lower, upper, peak = (lower + upper)/2)

use_distribution(distribution, smooth = TRUE)
```

Arguments

mean	Distribution mean.
sd	Distribution standard deviation.
meanlog	Mean on the log scale.
sdlog	SD on the log scale.
prob	Proportion.
size	Size of sample used to estimate proportion.
...	Dirichlet distribution parameters.
mu	Mean on the logit scale.
sigma	SD on the logit scale.
shape1	for beta distribution
shape2	for beta distribution
lower	lower bound of triangular distribution.
upper	upper bound of triangular distribution.
peak	peak of triangular distribution.
x	A distribution function, see details.

distribution	A numeric vector of observations defining a distribution, usually the output from an MCMC fit.
smooth	Use gaussian kernel smoothing?

Details

These functions are not exported, but only used in `define_psa()`. To specify a user-made function use `define_distribution()`.

`use_distribution()` uses gaussian kernel smoothing with a bandwith parameter calculated by `stats::density()`. Values for unobserved quantiles are calculated by linear interpolation.

`define_distribution()` takes as argument a function with a single argument, `x`, corresponding to a vector of quantiles. It returns the distribution values for the given quantiles. See examples.

Examples

```
define_distribution(
  function(x) stats::qexp(p = x, rate = 0.5)
)

# a mixture of 2 gaussians
x <- c(rnorm(100), rnorm(100, 6))
plot(density(x))

use_distribution(x)
```

---

export_savi *Export PSA Results for SAVI*

Description

Export the result of a PSA in a format compatible with Sheffield Accelerated Value of Information software.

Usage

```
export_savi(x, folder = ".")
```

Arguments

x	PSA result.
folder	A folder where to save the `csv` files.

Details

This function saves 3 files at the path given by `folder`: `param.csv`, the parameter values, `cost.csv` and `effect.csv` the cost and effect results.

The official SAVI website can be found at this URL: http://http://savi.shef.ac.uk/SAVI/

Value

Nothing. Creates 3 files.

`get_counts.updated_model` *Get State Membership Counts*

Description

Given a result from `run_model()`, return state membership counts for a specific strategy.

Usage

```
## S3 method for class 'updated_model'
get_counts(x, ...)

## S3 method for class 'combined_model'
get_counts(x, ...)

get_counts(x, ...)

## S3 method for class 'run_model'
get_counts(x, ...)

## S3 method for class 'eval_strategy'
get_counts(x, ...)

## S3 method for class 'list'
get_counts(x, ...)
```

Arguments

`x`	Result from `run_model()`.
`...`	further arguments passed to or from other methods.

Value

A data frame of counts per state.

get_values.updated_model *Get Strategy Values*

Description

Given a result from `run_model()`, return cost and effect values for a specific strategy.

Usage

```
## S3 method for class 'updated_model'
get_values(x, ...)

## S3 method for class 'combined_model'
get_values(x, ...)

get_values(x, ...)

## S3 method for class 'run_model'
get_values(x, ...)

## S3 method for class 'eval_strategy'
get_values(x, ...)

## S3 method for class 'list'
get_values(x, ...)
```

Arguments

x	Result from `run_model()`.
...	further arguments passed to or from other methods.

Value

A data frame of values per state.

heemod *Markov Models for Health Economic Evaluations*

Description

An implementation of the modelling and reporting features described in reference textbooks and guidelines: deterministic and probabilistic sensitivity analysis, heterogeneity analysis, time dependency on state-time and model-time (semi-Markov and non-homogeneous Markov models), etc.

join *Project Beyond a Survival Distribution with Another*

Description

Project survival from a survival distribution using one or more survival distributions using the specified cut points.

Usage

```
join(..., at)

project(...)

project_(...)

join_(dots, at)
```

Arguments

...	Survival distributions to be used in the projection.
at	A vector of times corresponding to the cut point(s) to be used.
dots	Used to work around non-standard evaluation.

Value

A surv_projection object.

Examples

```
dist1 <- define_survival(distribution = "exp", rate = .5)
dist2 <- define_survival(distribution = "gompertz", rate = .5, shape = 1)
join_dist <- join(dist1, dist2, at=20)
```

load_surv_models *Load a set of survival fits*

Description

Load a set of survival fits

Usage

```
load_surv_models(location, survival_specs, use_envir)
```

Arguments

location	base directory
survival_specs	information about fits
use_envir	an environment

Value

A list with two elements:

- best_models, a list with the fits for each data file passed in; and
- envir, an environment containing the models so they can be referenced to get probabilities.

---

look_up *Look Up Values in a Data Frame*

Description

A convenience function to easily look for values in a data frame.

Usage

```
look_up(data, ..., bin = FALSE, value = "value")
```

Arguments

data	A reference data frame.
...	Individual characteristics, should be named like the columns of data.
bin	Either logical: should all numeric variable be binned, or character vector giving the names of variables to bin (see examples).
value	The value to extract ffrom the reference data frame.

Details

This function is mostly used to extract population informations (such as mortality rates), given some individual caracteristics.

If binning is activated, numeric individual characteristics are matched to the corresponding reference value that is directly inferior.

Value

A vector of values, same lenght as ....

Examples

```
tempdf <- expand.grid(arg1 = c("A", "B", "C"), arg2 = 1:4, arg3 = 1:5)
tempdf$value <- 1:60

look_up(
  data = tempdf,
  value = "value",
  arg1 = c("A", "B", "C", "B", "A"),
  arg2 = c(1, 1, 3.2, 3.0, 5),
  arg3 = c(-1, 1, 1, 2, 3)
)

# binning doesnt catch values lesser than the smaller
# reference value
look_up(
  data = tempdf,
  value = "value",
  arg1 = c("A", "B", "C", "B", "A"),
  arg2 = c(1, 1, 3.2, 3.0, 5),
  arg3 = c(-1, 1, 1, 2, 3),
  bin = TRUE
)
# bin can alos be given as a charater vector
# to avoid binning all numeric variables
look_up(
  data = tempdf,
  value = "value",
  arg1 = c("A", "B", "C", "B", "A"),
  arg2 = c(1, 1, 3.2, 3.0, 5),
  arg3 = c(-1, 1, 1, 2, 3),
  bin = c("arg2")
)

age_related_df <- data.frame(age = 10 * 0:9, decade = 1:10)

look_up(age_related_df, age = c(0, 10, 20), value = "decade")

# binning might help in the situation
look_up(age_related_df, age = c(5, 15, 23.5),
        value = "decade")
look_up(age_related_df, age = c(5, 15, 23.5),
        value = "decade", bin = TRUE)
```

`mix` *Mix Two or More Survival Distributions*

Description

Mix a set of survival distributions using the specified weights.

Usage

```
mix(..., weights = 1)

mix_(dots, weights = 1)

pool(...)

pool_(...)
```

Arguments

...	Survival distributions to be used in the projection.
weights	A vector of weights used in pooling.
dots	Used to work around non-standard evaluation.

Value

A surv_pooled object.

Examples

```
dist1 <- define_survival(distribution = "exp", rate = .5)
dist2 <- define_survival(distribution = "gompertz", rate = .5, shape = 1)
pooled_dist <- mix(dist1, dist2, weights = c(0.25, 0.75))
```

---

modify *Modify Object*

Description

This generic function allows the modification of various objects such as parameters, transitions matrix or states.

Usage

```
modify(.OBJECT, ...)
```

Arguments

.OBJECT	Various objects.
...	Modifications.

Details

More details are available on the respective help page of each object definition.

Value

Same class as x.

---

part_survs_from_surv_inputs *Convert saved fits to partitioned survival objects*

Description

Convert saved fits to partitioned survival objects

Usage

```
part_survs_from_surv_inputs(surv_inputs, state_names)
```

Arguments

surv_inputs	a list of matrices of flexsurvreg objects, for example the first element of the output of survival_from_data.
state_names	names of states of the model

Details

surv_inputs is a tibble with columns type (PFS or OS, not case sensitive), treatment, set_name (for data subsets), dist (for survival distribution assumptions), fit (for the fitted survival object) and set_def (how the subset of data was defined, just to keep it around)

Value

a tibble of partitioned survival objects, similar to the original tibble of survival fits, with all the columns except type and fit, and a new column part_surv.

---

plot.dsa *Plot Sensitivity Analysis*

Description

Plot the results of a sensitivity analysis as a tornado plot.

Usage

```
## S3 method for class 'dsa'
plot(x, type = c("simple", "difference"),
  result = c("cost", "effect", "icer"), strategy = NULL,
  widest_on_top = TRUE, limits_by_bars = TRUE,
  resolve_labels = FALSE, shorten_labels = FALSE, remove_ns = FALSE,
  bw = FALSE, ...)
```

Arguments

x	A result of run_dsa().
type	Type of plot (see details).
result	Plot cost, effect, or ICER.
strategy	Name or index of strategies to plot.
widest_on_top	logical. Should bars be sorted so widest are on top?
limits_by_bars	logical. Should the limits used for each parameter be printed in the plot, next to the bars?
resolve_labels	logical. Should we resolve all labels to numbers instead of expressions (if there are any)?
shorten_labels	logical. Should we shorten the presentation of the parameters on the plot to highlight where the values differ?
remove_ns	Remove variables that are not sensitive.
bw	Black & white plot for publications?
...	Additional arguments passed to plot.

Details

Plot type simple plots variations of single strategy values, while difference plots incremental values.

Value

A ggplot2 object.

---

plot.psa *Plot Results of Probabilistic Analysis*

Description

Various plots for Markov models probabilistic analysis.

Usage

```
## S3 method for class 'psa'
plot(x, type = c("ce", "ac", "cov", "evpi"),
  max_wtp = 1e+05, n = 100, log_scale = TRUE, diff = FALSE,
  threshold, bw = FALSE, ...)
```

Arguments

x	Result from run_model().
type	Type of plot, see details.
max_wtp	Maximal willingness to pay.
n	Number of CECA points to estimate (values above 100 may take significant time).
log_scale	Show willingness to pay on a log scale?
diff	Logical, perform covariance analysis on strategy differences?
threshold	When diff = TRUE, threshlod value for net monetary benefit computation.
bw	Black & white plot for publications?
...	Additional arguments, depends on type.

Details

type = "ac" plots cost-effectiveness acceptability curves, type = "ce" plots results on the cost-efficiency plane, type = "cov" to perform covariance analysis on the results, type = "evpi" for expected value of perfect information.

Value

A ggplot2 object.

---

plot.run_model *Plot Results of a Markov Model*

Description

Various plots for Markov models.

Usage

```
## S3 method for class 'run_model'
plot(x, type = c("counts", "ce", "values"),
  panels = c("by_strategy", "by_state", "by_value"), values = NULL,
  strategy = NULL, states = NULL, free_y = FALSE, bw = FALSE, ...)
```

Arguments

x	Result from run_model().
type	Type of plot, see details.
panels	Should plots be faceted by model, by value or by state?
values	Names of values to be plotted. These can be any of the costs or effects defined in states.

strategy	Name or position of model(s) of interest.
states	Names of states to be included in the plot.
free_y	Should y limits be free between panels?
bw	Black & white plot for publications?
...	Additional arguments passed to plot. type = "counts" represents state memberships (corrected) by cycle, type = "ce" plots models on the cost-efficiency plane with the efficiency frontier, and type = "values" state values per cycle. When states is specified, the states will be turned into a factor with the ordering given in the variable, so that plotting order can be controlled.

Value

A ggplot2 object.

Examples

```
## These examples require \code{res_mod} from the hip replacement model discussed in
## `vignette("non-homogeneous", package = "heemod")`.

## Not run:
  plot(res_mod)

  plot(res_mod, model = "all")
  plot(res_mod, model = "all", panels = "by_state")

  plot(res_mod, model = "all", include_states = c("RevisionTHR", "SuccessR"))
  plot(res_mod, model = "all", panels = "by_state", include_states = c("RevisionTHR", "SuccessR"))

  plot(res_mod, model = 2, panel = "by_state", include_states = c("RevisionTHR", "SuccessR"))


## End(Not run)
```

---

plot.surv_obj *Plot general survival models*

Description

Plot general survival models

Usage

```
## S3 method for class 'surv_obj'
plot(x, times, type = c("surv", "prob"),
  join_col = "red", join_pch = 20, join_size = 3, ...)
```

Arguments

x	a survival object of class surv_aft, surv_add_haz, surv_ph, surv_po, surv_model, surv_pooled, or surv_projection.
times	Times at which to evaluate and plot the survival object.
type	either surv (the default) or prob, depending on whether you want to plot survival from the start or conditional probabilities.
join_col, join_pch, join_size	graphical parameters for points marking points at which different survival functions are joined.
...	additional arguments to pass to ggplot2 functions.

Details

The function currently only highlights join points that are at the top level; that is, for objects with class surv_projection.

To avoid plotting the join points, set join_size to a negative number.

Value

a ggplot2::ggplot() object.

---

probability *Convenience Functions to Compute Probabilities*

Description

These convienience functions make it easier to compute transition probabilities from indidence rates, OR, RR, or probabilities estimated on a different timeframe.

Usage

```
rescale_prob(p, to = 1, from = 1)

prob_to_prob(...)

rate_to_prob(r, to = 1, per = 1)

or_to_prob(or, p)

rr_to_prob(rr, p)
```

Arguments

p	Probability.
to	Compute probability for that timeframe.
from	Timeframe of the original probability.
...	For deprecated functions.
r	Rate.
per	Number of person-time corresponding to the rate.
or	Odds ratio.
rr	Relative risk.

Value

A probability.

Examples

```
# convert 5-year probability
# to 1-year probability
rescale_prob(p = .65, from = 5)

# convert 1-year probability
# to 1-month probability
rescale_prob(p = .5, to = 1/12)

# convert rate per 1000 PY
# to 5-year probability
rate_to_prob(r = 162, per = 1000, to = 5)

# convert OR to probability
or_to_prob(or = 1.9, p = .51)

# convert RR to probability
rr_to_prob(rr = 1.9, p = .51)
```

---

reindent_transition *Reindent Transition Matrix*

Description

Reindent Transition Matrix

Usage

```
reindent_transition(x, print = TRUE)
```

Arguments

x	A transition matrix.
print	Print result?

Value

The reindented matrix as a text string, invisibly.

---

rescale_discount_rate *Rescale Discount Rate*

Description

Rescale a discount rate between two time frames.

Usage

```
rescale_discount_rate(x, from, to)
```

Arguments

x	Discount rate to rescale.
from	Original time period.
to	Final time period.

Details

Continuous discounting is assumed, i.e. when converting a long-term discount rate into a short-term rate, we assume that a partial gain from one short term is multiplicatively discounted in all following short terms. At the same time, we assume the short-term rate is time-invariant.

Value

Rate rescaled under the assumption of compound discounting.

Examples

```
## 1% monthly interest rate to annual
rescale_discount_rate(0.01, 1, 12)
## 3% annual discount rate to (approximately) weekly
rescale_discount_rate(0.03, 52, 1)
```

run_bcea *Use the BCEA package*

Description

Interfaces the output of `run_psa()` into the BCEA package.

Usage

```
run_bcea(x, ...)
```

Arguments

`x`	Output from `run_psa()`.
`...`	Additional arguemnts passed to `BCEA::bcea()`.

Details

The BCEA package is needed for this function to work.

Value

A BCEA analysis.

run_dsa *Run Sensitivity Analysis*

Description

Run Sensitivity Analysis

Usage

```
run_dsa(model, dsa)
```

Arguments

`model`	An evaluated Markov model.
`dsa`	An object returned by `define_dsa()`.

Value

A `data.frame` with one row per model and parameter value.

Examples

```
param <- define_parameters(
  p1 = .5,
  p2 = .2,
  r = .05
)
mod1 <- define_strategy(
  transition = define_transition(
    C, p1,
    p2, C
  ),
  define_state(
    cost = discount(543, r),
    ly = 1
  ),
  define_state(
    cost = discount(432, r),
    ly = .5
  )
)

mod2 <- define_strategy(
  transition = define_transition(
    C, p1,
    p2, C
  ),
  define_state(
    cost = 789,
    ly = 1
  ),
  define_state(
    cost = 456,
    ly = .8
  )
)

res2 <- run_model(
  mod1, mod2,
  parameters = param,
  init = c(100, 0),
  cycles = 10,
  cost = cost,
  effect = ly
)

ds <- define_dsa(
  p1, .1, .9,
  p2, .1, .3,
  r, .05, .1
)
print(ds)
```

```
x <- run_dsa(res2, ds)

plot(x, value = "cost")

# can be specified as a function of other parameters


ds2 <- define_dsa(
  p2, p1 - .1, p1 + .1
)

run_dsa(res2, ds2)
```

`run_model` *Run Markov Model*

Description

Runs one or more strategy. When more than one strategy is provided, all strategies should have the same states and state value names.

Usage

```
run_model(..., parameters = define_parameters(), init = c(1000L,
  rep(0L, get_state_number(get_states(list(...)[[1]])) - 1)), cycles = 1,
  method = "life-table", cost = NULL, effect = NULL,
  state_time_limit = NULL, central_strategy = NULL, inflow = rep(0L,
  get_state_number(get_states(list(...)[[1]]))))

run_model_(uneval_strategy_list, parameters, init, cycles, method, cost,
  effect, state_time_limit, central_strategy, inflow)
```

Arguments

`...`	One or more `uneval_model` object.
`parameters`	Optional. An object generated by `define_parameters()`.
`init`	numeric vector or result of `define_init()`, same length as number of states. Number of individuals in each state at the beginning.
`cycles`	positive integer. Number of Markov Cycles to compute.
`method`	Counting method.
`cost`	Names or expression to compute cost on the cost-effectiveness plane.
`effect`	Names or expression to compute effect on the cost-effectiveness plane.
`state_time_limit`	Optional expansion limit for `state_time`, see details.
`central_strategy`	character. The name of the strategy at the center of the cost-effectiveness plane, for readability.

`inflow` numeric vector or result of `define_inflow()`, similar to `init`. Number of new individuals in each state per cycle.

`uneval_strategy_list` List of models, only used by `run_model_()` to avoid using `...`.

Details

In order to compute comparisons strategies must be similar (same states and state value names). Thus strategies can only differ through transition matrix cell values and values attached to states (but not state value names).

The initial number of individuals in each state and the number of cycle will be the same for all strategies

`state_time_limit` can be specified in 3 different ways:

1. As a single value: the limit is applied to all states in all strategies. 2. As a named vector (where names are state names): the limits are applied to the given state names, for all strategies. 3. As a named list of named vectors: the limits are applied to the given state names for the given strategies.

Value

A list of evaluated models with computed values.

Examples

```
# running a single model

mod1 <-
  define_strategy(
    transition = define_transition(
      .5, .5,
      .1, .9
    ),
    define_state(
      cost = 543,
      ly = 1
    ),
    define_state(
      cost = 432,
      ly = 1
    )
  )


res <- run_model(
  mod1,
  init = c(100, 0),
  cycles = 2,
  cost = cost,
  effect = ly
)
```

```
# running several models
mod2 <-
  define_strategy(
    transition = define_transition(
      .5, .5,
      .1, .9
    ),
    define_state(
      cost = 789,
      ly = 1
    ),
    define_state(
      cost = 456,
      ly = 1
    )

  )


res2 <- run_model(
  mod1, mod2,
  init = c(100, 0),
  cycles = 10,
  cost = cost,
  effect = ly
)
```

`run_model_tabular` *Run Analyses From Files*

Description

This function runs a model from tabular input.

Usage

```
run_model_tabular(location, reference = "REFERENCE.csv",
  run_dsa = TRUE, run_psa = TRUE, run_demo = TRUE, save = FALSE,
  overwrite = FALSE)
```

Arguments

`location`	Directory where the files are located.
`reference`	Name of the reference file.
`run_dsa`	Run DSA?
`run_psa`	Run PSA?.
`run_demo`	Run demgraphic analysis?

save	Should the outputs be saved?
overwrite	Should the outputs be overwritten?

Details

The reference file should have two columns, data and file. An optional absolute_path column can be added, having value TRUE where an absolute file path is provided. data values must include state, tm, and parameters, and can also include options, demographics and data. The corresponding values in the file column give the names of the files (located in base_dir) that contain the corresponding information - or, in the case of data, the directory containing the tables to be loaded.

Value

A list of evaluated models (always), and, if appropriate input is provided, dsa (deterministic sensitivity analysis), psa (probabilistic sensitivity analysis) and demographics (results across different demographic groups).

---

run_psa *Run Probabilistic Uncertainty Analysis*

Description

Run Probabilistic Uncertainty Analysis

Usage

```
run_psa(model, psa, N, resample)
```

Arguments

model	The result of run_model().
psa	Resampling distribution for parameters defined by define_psa().
N	1. Number of simulation to run.
resample	Deprecated. Resampling distribution for parameters defined by define_psa().

Value

A list with one data.frame per model.

Examples

```
# example for run_psa

mod1 <- define_strategy(
  transition = define_transition(
    .5, .5,
    .1, .9
  ),
  define_state(
    cost = cost_init + age * 5,
    ly = 1
  ),
  define_state(
    cost = cost_init + age,
    ly = 0
  )
)

mod2 <- define_strategy(
  transition = define_transition(
    p_trans, C,
    .1, .9
  ),
  define_state(
    cost = 789 * age / 10,
    ly = 1
  ),
  define_state(
    cost = 456 * age / 10,
    ly = 0
  )

)

res2 <- run_model(
  mod1, mod2,
  parameters = define_parameters(
    age_init = 60,
    cost_init = 1000,
    age = age_init + markov_cycle,
    p_trans = .7
  ),
  init = 1:0,
  cycles = 10,
  cost = cost,
  effect = ly
)

rsp <- define_psa(
  age_init ~ normal(60, 10),
  cost_init ~ normal(1000, 100),
  p_trans ~ binomial(.7, 100),
```

```
  correlation = matrix(c(
    1,  .4, 0,
    .4, 1,  0,
    0,  0,  1
  ), byrow = TRUE, ncol = 3)
)


# with run_model result
# (only 10 resample for speed)
ndt1 <- run_psa(res2, psa = rsp, N = 10)
```

`set_covariates` *Set Covariates of a Survival Distribution*

Description

Set the covariate levels of a survival model to be represented in survival projections.

Usage

```
set_covariates(dist, ..., data = NULL)

set_covariates_(dist, covariates, data = NULL)
```

Arguments

`dist`	a survfit or flexsurvreg object
`...`	Covariate values representing the group for which survival probabilities will be generated when evaluated.
`data`	A an optional data frame representing multiple sets of covariate values for which survival probabilities will be generated. Can be used to generate aggregate survival for a heterogenous set of subjects.
`covariates`	Used to work around non-standard evaluation.

Value

A `surv_model` object.

Examples

```
fs1 <- flexsurv::flexsurvreg(
  survival::Surv(rectime, censrec)~group,
  data=flexsurv::bc,
  dist = "llogis"
)
good_model <- set_covariates(fs1, group = "Good")
```

```
cohort <- data.frame(group=c("Good", "Good", "Medium", "Poor"))
mixed_model <- set_covariates(fs1, data = cohort)
```

`summary.run_model` *Summarise Markov Model Results*

Description

Summarise Markov Model Results

Usage

```
## S3 method for class 'run_model'
summary(object, threshold = NULL, ...)
```

Arguments

`object`	Output from `run_model()`.
`threshold`	ICER threshold (possibly several) for net monetary benefit computation.
`...`	additional arguments affecting the summary produced.

Value

A `summary_run_model` object.

`summary.surv_shift` *Summarize surv_shift objects*

Description

Summarize surv_shift objects

Usage

```
## S3 method for class 'surv_shift'
summary(object, summary_type = c("plot",
  "standard"), ...)
```

Arguments

`object`	a `surv_shift` object
`summary_type`	"standard" or "plot" - "standard" for the usual summary of a `survfit` object, "plot" for a fuller version
`...`	other arguments

Value

A summary.

`update_model` *Run Model on New Data*

Description

Given a table of new parameter values with a new parameter set per line, runs iteratively Markov models over these sets.

Usage

```
## S3 method for class 'run_model'
update(object, newdata, ...)

## S3 method for class 'updated_model'
plot(x, type = c("simple", "difference",
  "counts", "ce", "values"), result = c("cost", "effect", "icer"),
  strategy = NULL, ...)
```

Arguments

`object`	The result of `run_model()`.
`newdata`	A `data.frame` of new parameter sets, one column per parameter and one row per parameter set. An optional `.weights` column can be included for a weighted analysis.
`...`	Additional arguments passed to `geom_histogram`. Especially usefull to specify `binwidth`.
`x`	Updated model to plot.
`type`	Plot simple values or differences?
`result`	The the result to plot (see details).
`strategy`	A model index, character or numeric.

Details

`newdata` must be a `data.frame` with the following properties: the column names must be parameter names used in `define_parameters()`; and an optional column `.weights` can give the respective weight of each row in the target population.

Weights are automatillcally scaled. If no weights are provided equal weights are used for each strata.

For the plotting function, the `type` argument can take the following values: `"cost"`, `"effect"` or `"icer"` to plot the heterogeneity of the respective values. Furthermore `"ce"` and `"count"` can produce from the combined model plots similar to those of `run_model()`.

Value

A `data.frame` with one row per model/value.

Warning

Histograms do not account for weights. On the other hand summary results do.

Examples

```
mod1 <-
  define_strategy(
    transition = define_transition(
      .5, .5,
      .1, .9
    ),
    define_state(
      cost = 543 + age * 5,
      ly = 1
    ),
    define_state(
      cost = 432 + age,
      ly = 1 * age / 100
    )
  )

mod2 <-
  define_strategy(
    transition = define_transition(
      .5, .5,
      .1, .9
    ),
    define_state(
      cost = 789 * age / 10,
      ly = 1
    ),
    define_state(
      cost = 456 * age / 10,
      ly = 1 * age / 200
    )
  )

res <- run_model(
  mod1, mod2,
  parameters = define_parameters(
    age_init = 60,
    age = age_init + markov_cycle
  ),
  init = 1:0,
  cycles = 10,
  cost = cost,
  effect = ly
)
```

```
# generating table with new parameter sets
new_tab <- data.frame(
  age_init = 40:45
)

# with run_model result
ndt <- update(res, newdata = new_tab)

summary(ndt)

# using weights

new_tab2 <- data.frame(
  age_init = 40:45,
  .weights = runif(6)
)
ndt2 <- update(res, newdata = new_tab2)

summary(ndt2)
```

`who_mortality` *Use WHO Mortality Rate*

Description

Returns age and sex-specific mortality probabilities for a given country.

Usage

```
get_who_mr_memo(age, sex = NULL, region = NULL, country = NULL,
  year = "latest", local = FALSE)

get_who_mr(age, sex = NULL, region = NULL, country = NULL,
  year = "latest", local = FALSE)
```

Arguments

`age`	age as a continuous variable.
`sex`	sex as `"FMLE"`-`"MLE"`, `0`-1 (male = 0, female = 1) or 1-2 (male = 1, female = 2).
`region`	Region code. Assumed `NULL` if provided along with `country`.
`country`	Country code (see details).
`year`	Use data from that year. Defaults to `"latest"`.
`local`	Fetch mortality data from package cached data?

Details

Locally cached data is used in case of connection problems, of if `local = TRUE`. For memory space reasons local data is only available for WHO high-income countries, and only for the latest year.

The results of `get_who_mr` are memoised for `options("heemod.memotime")` (default: 1 hour) to increase resampling performance.

Value

This function should be used within `define_transition()` or `define_parameters()`.

Examples

```
define_transition(
  C, get_who_mr(age = 50 + markov_cycle, sex = "FMLE", country = "FRA"),
  0, 1
)
```

Index

add_hazards, 3
add_hazards_ *(add_hazards)*, 3
apply_af, 4
apply_hr, 4
apply_or, 5
apply_shift, 6

BCEA::bcea(), *42*
beta *(distributions)*, 26
binomial *(distributions)*, 26

calibrate_model, 6
calibrate_model(), *7*, *11*
close_cluster *(cluster)*, 8
close_cluster(), *9*
cluster, 8
combine_probs, 9
compute_surv, 10
compute_surv(), *11*, *23*
construct_part_surv_tib, 10

define_calibration_fn, 11
define_correlation, 12
define_correlation(), *18*
define_correlation_ *(define_correlation)*, 12
define_distribution *(distributions)*, 26
define_distribution(), *18*, *28*
define_dsa, 13
define_dsa(), *42*
define_dsa_ *(define_dsa)*, 13
define_inflow, 13
define_inflow(), *45*
define_inflow_ *(define_inflow)*, 13
define_init, 14
define_init(), *44*
define_init_ *(define_init)*, 14
define_parameters, 14
define_parameters(), *14*, *19–21*, *23*, *24*, *44*, *51*, *54*
define_parameters_ *(define_parameters)*, 14
define_part_surv, 16
define_part_surv_ *(define_part_surv)*, 16
define_psa, 17
define_psa(), *28*, *47*
define_psa_ *(define_psa)*, 17
define_spline_survival, 18
define_starting_values, 19
define_starting_values(), *21*
define_starting_values_ *(define_starting_values)*, 19
define_state, 20
define_state(), *21*
define_state_ *(define_state)*, 20
define_strategy, 21
define_strategy(), *24*
define_strategy_ *(define_strategy)*, 21
define_surv_table, 23
define_survival, 22
define_survival(), *16*
define_transition, 23
define_transition(), *21*, *54*
define_transition_ *(define_transition)*, 23
diagram::plotmat(), *24*
dispatch_strategy, 26
distributions, *18*, 26
dplyr::mutate(), *15*
dplyr::n(), *15*
dplyr::row_number(), *15*

export_savi, 28

flexsurv::flexsurvreg(), *16*

gamma *(distributions)*, 26
get_counts *(get_counts.updated_model)*, 29
get_counts.updated_model, 29

get_values *(get_values.updated_model)*, 30
get_values.updated_model, 30
get_who_mr *(who_mortality)*, 53
get_who_mr_memo *(who_mortality)*, 53
ggplot2::ggplot(), *39*

heemod, 30
heemod-package *(heemod)*, 30

join, 31
join_ *(join)*, 31

load_surv_models, 31
logitnormal *(distributions)*, 26
lognormal *(distributions)*, 26
look_up, 32

mix, 33
mix_ *(mix)*, 33
modify, 34
modify.state *(define_state)*, 20
modify.uneval_matrix *(define_transition)*, 23
modify.uneval_parameters *(define_parameters)*, 14
multinomial *(distributions)*, 26

normal *(distributions)*, 26

optimx::optimx(), *7*
or_to_prob *(probability)*, 39

parallel::parLapply(), *9*
part_survs_from_surv_inputs, 35
plot.dsa, 35
plot.psa, 36
plot.run_model, 37
plot.surv_obj, 38
plot.uneval_matrix *(define_transition)*, 23
plot.updated_model *(update_model)*, 51
poisson *(distributions)*, 26
pool *(mix)*, 33
pool_ *(mix)*, 33
prob_to_prob *(probability)*, 39
probability, 39
project *(join)*, 31
project_ *(join)*, 31

rate_to_prob *(probability)*, 39
reindent_transition, 40
reindent_transition(), *24*
rescale_discount_rate, 41
rescale_prob *(probability)*, 39
rr_to_prob *(probability)*, 39
run_bcea, 42
run_dsa, 42
run_dsa(), *36*
run_model, 44
run_model(), *7*, *29*, *30*, *37*, *47*, *50*, *51*
run_model_ *(run_model)*, 44
run_model_(), *45*
run_model_tabular, 46
run_psa, 47
run_psa(), *9*, *42*

set_covariates, 49
set_covariates_ *(set_covariates)*, 49
stats::density(), *28*
stats::optimise(), *7*
status_cluster *(cluster)*, 8
summary.run_model, 50
summary.surv_shift, 50

triangle *(distributions)*, 26

update(), *7*
update.run_model *(update_model)*, 51
update_model, 51
use_cluster *(cluster)*, 8
use_distribution *(distributions)*, 26
use_distribution(), *28*

who_mortality, 53