

# Package ‘heemod’

May 8, 2026

**Title** Markov Models for Health Economic Evaluations

**Version** 1.1.0

**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 (>= 4.1.0)

**Imports** dplyr (>= 1.1.0), ggplot2 (>= 3.4.0), mvnfast (>= 0.2.5), tibble (>= 3.2.0), rlang (>= 1.1.0), purrr (>= 1.0.0), glue (>= 1.6.0), lifecycle (>= 1.0.0), vctrs (>= 0.6.0)

**VignetteBuilder** knitr

**RoxygenNote** 7.3.2

**Suggests** BCEA, diagram, flexsurv, knitr, logitnorm, lpSolve, mgcv, optimx, parallel, readxl, rmarkdown, stringr, survival, testthat (>= 3.0.0), triangle, magrittr, cli

**BugReports** <https://github.com/aphp/heemod/issues>

**Encoding** UTF-8

**Config/testthat/edition** 3

**License** GPL (>= 3)

**URL** <https://aphp.github.io/heemod/>

**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** 2025-06-16 13:10:06 UTC

## Contents

add_hazards . . . . .	3
apply_af . . . . .	4
apply_hr . . . . .	4
apply_or . . . . .	5
apply_shift . . . . .	5
calibrate_model . . . . .	6
cluster . . . . .	8
combine_probs . . . . .	9
compute_surv . . . . .	9
construct_part_surv_tib . . . . .	10
define_calibration_fn . . . . .	11
define_correlation . . . . .	12
define_dsa . . . . .	12
define_inflow . . . . .	13
define_init . . . . .	14
define_parameters . . . . .	14
define_part_surv . . . . .	16
define_psa . . . . .	17
define_starting_values . . . . .	18
define_state . . . . .	19
define_strategy . . . . .	20
define_surv_dist . . . . .	21
define_surv_fit . . . . .	22
define_surv_spline . . . . .	23
define_surv_table . . . . .	23
define_transition . . . . .	24
dispatch_strategy . . . . .	26
distributions . . . . .	27
export_savi . . . . .	29
get_counts.updated_model . . . . .	29
get_values.updated_model . . . . .	30
join . . . . .	31
load_surv_models . . . . .	32
look_up . . . . .	32
mix . . . . .	34
modify . . . . .	34
part_survs_from_surv_inputs . . . . .	35
plot.dsa . . . . .	36
plot.psa . . . . .	37
plot.run_model . . . . .	38
plot.surv_object . . . . .	39

probability . . . . .	40
reindent_transition . . . . .	42
rescale_discount_rate . . . . .	42
run_bcea . . . . .	43
run_dsa . . . . .	43
run_model . . . . .	45
run_model_tabular . . . . .	48
run_psa . . . . .	49
set_covariates . . . . .	51
summary.run_model . . . . .	52
summary.surv_shift . . . . .	52
update_model . . . . .	53
who_mortality . . . . .	55

<b>Index</b>	<b>57</b>
--------------	-----------

---

add_hazards	<i>Add Hazards</i>
-------------	--------------------

---

## 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_surv_dist(distribution = "exp", rate = .125)
dist2 <- define_surv_dist(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 acceleration factor is exponentiated before being applied.

**Value**

A surv\_aft object.

**Examples**

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

---

apply_or	<i>Apply an Odds Ratio</i>
----------	----------------------------

---

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

---

apply_shift	<i>Apply a time shift to a survival distribution</i>
-------------	--

---

**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_surv_dist(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**

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

## 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" & model_time == 10
  )$count
}
f(res_mod)
```

```

#' \dontrun{
#' 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 automatically 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 workflow 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 cluster 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	<i>Combine Probabilities</i>
---------------	------------------------------

---

**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	<i>Evaluate Survival Distributions</i>
--------------	--

---

**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 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.

**Value**

Returns either the survival probabilities 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

<code>type</code>	Type of model values (count or value).
<code>strategy_names</code>	Names of strategies.
<code>element_names</code>	Names of states (for counts) or of state values (for values).
<code>cycles</code>	Cycles of interest.
<code>groups</code>	Optional grouping of values (values in a same group have the same groups).
<code>aggreg_fn</code>	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 coefficients of the form var1, var2, cor(var1, var2), var3, var4, cor(var3, var4), ....

.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(var2), max(var2), ....

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	<i>Define Inflow for a BIA</i>
---------------	--------------------------------

---

**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	<i>Define Initial Counts</i>
-------------	------------------------------

---

**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	<i>Define Markov Model Parameters</i>
-------------------	---------------------------------------

---

**Description**

Define parameters called to compute the transition matrix or state values for a Markov model. Parameters can be time dependent by using the `model_time` 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 <code>uneval_parameters</code> .

## 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 `model_time` (whose length depends on the number of simulation cycles). Alternatively, `dplyr` functions such as `dplyr::n()` 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 quosures).

## Examples

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

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

# other uses of model_time are possible

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

# more elaborate: risk function

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

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

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

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

```

)

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

# 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 <code>flexsurv::flexsurvreg()</code> or <code>define_surv_dist()</code> .
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_surv_dist("exp", rate = 1)
dist_os <- define_surv_dist("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 quosures.

**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 matrix 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\_starting\_values

*Define Starting State Values*

---

**Description**

This function is meant to be used inside [define\\_strategy\(\)](#) and [define\\_state\(\)](#).

**Usage**

```
define_starting_values(...)

define_starting_values_(.dots)
```

**Arguments**

<code>...</code>	Name-value pairs of expressions defining starting values. The names must correspond to an existing state value.
<code>.dots</code>	Used to work around non-standard evaluation.

**Details**

The behaviour is different following the function using `define_starting_values()` as an argument.

- When used inside `define_strategy()`, the state values are modified for the first cycle in each state
- When used inside `define_state()`, the state values are modified for counts entering the state

**Value**

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

---

define_state	<i>Define a Markov Model State</i>
--------------	------------------------------------

---

**Description**

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

**Usage**

```
define_state(..., starting_values = define_starting_values())
```

```
define_state_(x)
```

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

**Arguments**

...	Name-value pairs of expressions defining state values.
starting_values	Optional starting values defined with <code>define_starting_values()</code> .
x	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 quosures).

**Examples**

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

---

define_strategy	<i>Define a Markov Model</i>
-----------------	------------------------------

---

**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**

...	Objects generated by <a href="#">define_state()</a> . Each object should be named with the state names of the transition matrix.
transition	An object generated by <a href="#">define_transition()</a> .
starting_values	Optional starting values defined with <a href="#">define_starting_values()</a> .
states	List of states, only used by <code>define_strategy_</code> 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\_surv\_dist      *Define a Survival Distribution*

---

**Description**

Define a parametric survival distribution.

**Usage**

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

---

define_surv_fit	<i>Define a Fitted Survival Model</i>
-----------------	---------------------------------------

---

**Description**

Define a fitted survival models with a Kaplan-Meier estimator or parametric distributions

**Usage**

```
define_surv_fit(x)
```

**Arguments**

x a survfit or flexsurvreg object

**Value**

A surv\_object object.

**Examples**

```
library(survival)

define_surv_fit(
  survfit(Surv(time, status) ~ 1, data = colon)
)

define_surv_fit(
  flexsurv::flexsurvreg(Surv(time, status) ~ 1, data = colon, dist = "exp")
)
```

---

define\_surv\_spline     *Define a Restricted Cubic Spline Survival Distribution*

---

**Description**

Define a restricted cubic spline parametric survival distribution.

**Usage**

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

---

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 defining matrix cells. Can refer to parameters defined with <code>define_parameters()</code> . For plot, additional arguments passed to <code>diagram::plotmat()</code> .
state_names	character vector, optional. State names.
.dots	Used to work around non-standard evaluation.
.OBJECT	An object of class <code>uneval_matrix</code> .
x	An <code>uneval_matrix</code> to plot.
resize	Argument passed to <code>diagram::plotmat()</code> .
shadow.size	Argument passed to <code>diagram::plotmat()</code> .
latex	Argument passed to <code>diagram::plotmat()</code> .

**Details**

Matrix 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 quosures 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 implicitly 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 + model_time  
  )  
)
```

---

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 bandwidth 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**

<code>x</code>	PSA result.
<code>folder</code>	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: <https://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.

---

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

---

load_surv_models	<i>Load a set of survival fits</i>
------------------	------------------------------------

---

**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	<i>Look Up Values in a Data Frame</i>
---------	---------------------------------------

---

**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 from the reference data frame.

## Details

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

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

## Value

A vector of values, same length 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 also be given as a character 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	<i>Mix Two or More Survival Distributions</i>
-----	---

---

### Description

Mix a set of survival distributions using the specified weights.

### Usage

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

mix_(dots, weights = 1)
```

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

---

modify	<i>Modify Object</i>
--------	----------------------

---

### 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.

**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_byBars = TRUE,
  resolve_labels = FALSE,
  shorten_labels = FALSE,
  remove_ns = FALSE,
  bw = FALSE,
  ...
)
```

**Arguments**

x	A result of <a href="#">run_dsa()</a> .
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_byBars	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 <code>run_model()</code> .
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 <code>diff = TRUE</code> , threshold 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 <code>run_model()</code> .
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\_object      *Plot general survival models*

---

**Description**

Plot general survival models

**Usage**

```
## S3 method for class 'surv_object'
plot(
  x,
  times = seq.int(0, 30),
  type = c("surv", "prob"),
  psa,
  Nrep = 100,
  join_opts = list(join_col = "red", join_pch = 20, join_size = 3),
  ...
)
```

**Arguments**

x	a survival object of class <code>surv_aft</code> , <code>surv_add_haz</code> , <code>surv_ph</code> , <code>surv_po</code> , <code>surv_model</code> , <code>surv_pooled</code> , or <code>surv_projection</code> .
times	Times at which to evaluate and plot the survival object.
type	either <code>surv</code> (the default) or <code>prob</code> , depending on whether you want to plot survival from the start or conditional probabilities.
psa	a <code>define_psa</code> object
Nrep	The number of replications to estimate the variability of x

`join_opts` A list of 3 graphical parameters for points at which different survival functions are joined: `join_col`, `join_pch` and `join_size`.

... additional arguments to pass to `compute_surv` 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.

### Examples

```
## Evaluation of the variability of the survival distribution

surv1 <- define_surv_dist("exp", rate = 0.1)
psa <- define_psa(surv1 ~ resample_surv(n = 100))
plot(surv1, psa=psa)

## plot surv_projection object
surv2 <- define_surv_dist("exp", rate = 0.5)
plot(join(surv1, surv2, at = 2), psa = psa, Nrep = 50)

## surv_fit object
library(survival)
km <- define_surv_fit(survfit(formula = Surv(time, status) ~ 1, data = aml))
fs <- flexsurv::flexsurvreg(formula = Surv(time, status) ~ 1,
                           data = aml,
                           dist = "weibull") |>
  define_surv_fit()

psa2 <- define_psa(km ~ resample_surv(),
                  fs ~ resample_surv(),
                  surv1 ~ resample_surv(100))
plot(km, psa = psa2)

plot(join(km, surv1, at = 6), psa = psa2)
plot(join(fs, surv1, at = 6), psa = psa2)
```

### Description

These convenience functions make it easier to compute transition probabilities from incidence 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 arguments 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)

#' \dontrun{
#' 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
)

#' \dontrun{
#' 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 = c("life-table", "beginning", "end"),
  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 <code>uneval_model</code> object.
<code>parameters</code>	Optional. An object generated by <code>define_parameters()</code> .
<code>init</code>	numeric vector or result of <code>define_init()</code> , same length as number of states. Number of individuals in each state at the beginning.
<code>cycles</code>	positive integer. Number of Markov Cycles to compute.
<code>method</code>	Counting method. See details.
<code>cost</code>	Names or expression to compute cost on the cost-effectiveness plane.
<code>effect</code>	Names or expression to compute effect on the cost-effectiveness plane.
<code>state_time_limit</code>	Optional expansion limit for <code>state_time</code> , see details.
<code>central_strategy</code>	character. The name of the strategy at the center of the cost-effectiveness plane, for readability.
<code>inflow</code>	numeric vector or result of <code>define_inflow()</code> , similar to <code>init</code> . Number of new individuals in each state per cycle.
<code>uneval_strategy_list</code>	List of models, only used by <code>run_model_()</code> 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.

Counting method represents where the transition should occur, based on <https://journals.sagepub.com/doi/10.1177/0272989X> "beginning" overestimates costs and "end" underestimates costs.

**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 demographic analysis?
save	Should the outputs be saved?
overwrite	Should the outputs be overwritten?

### Details

The reference file should have two columns. `data` 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, keep = FALSE)
```

**Arguments**

model	The result of <code>run_model()</code> .
psa	Resampling distribution for parameters defined by <code>define_psa()</code> .
N	> 0. Number of simulation to run.
keep	logical; if TRUE, all models will be returned

**Value**

A list with the following elements

- `psa`: a `data.frame` with one row per model.
- `run_model`: a `data.frame` with mean cost and utility for each strategy
- `model`: the initial model object
- `N`: the number of simulations ran
- `resamp_par`: the resampled parameters
- `full`: if `keep` is TRUE, a list of each model objects created at each iteration

**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 + model_time,
    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	<i>Set Covariates of a Survival Distribution</i>
----------------	--

---

### 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 heterogeneous 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	<i>Run Model on New Data</i>
--------------	------------------------------

---

### 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 <code>run_model()</code> .
newdata	A <code>data.frame</code> of new parameter sets, one column per parameter and one row per parameter set. An optional <code>.weights</code> column can be included for a weighted analysis.
...	Additional arguments passed to <code>geom_histogram</code> . Especially useful to specify <code>binwidth</code> .
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 automatically 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 + model_time
  ),
  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)
)

#'\dontrun{
#'\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 region

## Usage

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

## 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.
year	Use data from that year. Defaults to "latest".

## Details

Only locally cached data are available. For memory space reasons local data is only available for WHO high-income countries (pooled), and only for the latest year.

**Value**

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

**Examples**

```
define_transition(  
  C, get_who_mr(age = 50 + model_time, sex = "FMLE", region = "EUR"),  
  0, 1  
)
```

# Index

add\_hazards, 3  
add\_hazards\_(add\_hazards), 3  
apply\_af, 4  
apply\_hr, 4  
apply\_or, 5  
apply\_shift, 5

BCEA::bcea(), 43  
beta (distributions), 27  
binomial (distributions), 27

calibrate\_model, 6  
calibrate\_model(), 7, 11  
close\_cluster (cluster), 8  
close\_cluster(), 8  
cluster, 8  
combine\_probs, 9  
compute\_surv, 9  
compute\_surv(), 10, 24  
construct\_part\_surv\_tib, 10

define\_calibration\_fn, 11  
define\_correlation, 12  
define\_correlation(), 17  
define\_correlation\_  
    (define\_correlation), 12  
define\_distribution (distributions), 27  
define\_distribution(), 18, 28  
define\_dsa, 12  
define\_dsa(), 43  
define\_dsa\_ (define\_dsa), 12  
define\_inflow, 13  
define\_inflow(), 46  
define\_inflow\_ (define\_inflow), 13  
define\_init, 14  
define\_init(), 46  
define\_init\_ (define\_init), 14  
define\_parameters, 14  
define\_parameters(), 13, 14, 19, 20, 24, 25,  
    46, 53, 56

define\_parameters\_ (define\_parameters),  
    14  
define\_part\_surv, 16  
define\_part\_surv\_ (define\_part\_surv), 16  
define\_psa, 17  
define\_psa(), 28, 49  
define\_psa\_ (define\_psa), 17  
define\_starting\_values, 18  
define\_starting\_values(), 19, 20  
define\_starting\_values\_  
    (define\_starting\_values), 18  
define\_state, 19  
define\_state(), 18–20  
define\_state\_ (define\_state), 19  
define\_strategy, 20  
define\_strategy(), 18, 19, 25  
define\_strategy\_ (define\_strategy), 20  
define\_surv\_dist, 21  
define\_surv\_dist(), 16  
define\_surv\_fit, 22  
define\_surv\_spline, 23  
define\_surv\_table, 23  
define\_transition, 24  
define\_transition(), 20, 56  
define\_transition\_ (define\_transition),  
    24

diagram::plotmat(), 25  
dispatch\_strategy, 26  
distributions, 18, 27  
dplyr::mutate(), 15  
dplyr::n(), 15

export\_savi, 29

flexsurv::flexsurvreg(), 16

gamma (distributions), 27  
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), 55  
 ggplot2::ggplot(), 40  
  
 join, 31  
 join\_ (join), 31  
  
 load\_surv\_models, 32  
 logitnormal (distributions), 27  
 lognormal (distributions), 27  
 look\_up, 32  
  
 mix, 34  
 mix\_ (mix), 34  
 modify, 34  
 modify.state (define\_state), 19  
 modify.uneval\_matrix  
     (define\_transition), 24  
 modify.uneval\_parameters  
     (define\_parameters), 14  
 multinomial (distributions), 27  
  
 normal (distributions), 27  
  
 optimx::optimx(), 6, 7  
 or\_to\_prob (probability), 40  
  
 parallel::parLapply(), 8  
 part\_survs\_from\_surv\_inputs, 35  
 plot.dsa, 36  
 plot.psa, 37  
 plot.run\_model, 38  
 plot.surv\_object, 39  
 plot.uneval\_matrix (define\_transition),  
     24  
 plot.updated\_model (update\_model), 53  
 poisson (distributions), 27  
 prob\_to\_prob (probability), 40  
 probability, 40  
  
 rate\_to\_prob (probability), 40  
 reindent\_transition, 42  
 reindent\_transition(), 25  
 rescale\_discount\_rate, 42  
 rescale\_prob (probability), 40  
 rr\_to\_prob (probability), 40  
 run\_bcea, 43  
 run\_dsa, 43  
  
 run\_dsa(), 36  
 run\_model, 45  
 run\_model(), 6, 29–31, 37, 38, 49, 52, 53  
 run\_model\_ (run\_model), 45  
 run\_model\_(), 46  
 run\_model\_tabular, 48  
 run\_psa, 49  
 run\_psa(), 8, 43  
  
 set\_covariates, 51  
 set\_covariates\_ (set\_covariates), 51  
 stats::density(), 28  
 stats::optimise(), 7  
 status\_cluster (cluster), 8  
 summary.run\_model, 52  
 summary.surv\_shift, 52  
  
 triangle (distributions), 27  
  
 update(), 6  
 update.run\_model (update\_model), 53  
 update\_model, 53  
 use\_cluster (cluster), 8  
 use\_distribution (distributions), 27  
 use\_distribution(), 28  
  
 who\_mortality, 55