

# Package ‘WARDEN’

July 21, 2025

**Title** Workflows for Health Technology Assessments in R using Discrete  
EveNts

**Version** 1.2.5

**Description** Toolkit to support and perform discrete event simulations without  
resource constraints in the context of health technology assessments (HTA).  
The package focuses on cost-effectiveness modelling and aims to be submission-ready  
to relevant HTA bodies in alignment with 'NICE TSD 15'  
<<https://sheffield.ac.uk/nice-dsu/tsds/patient-level-simulation>>.  
More details an examples can be found in the package website <<https://jsanchezalv.github.io/WARDEN/>>.

**License** GPL (>= 3)

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.3.2

**BugReports** <https://github.com/jsanchezalv/WARDEN/issues>

**Suggests** dplyr, ggplot2, knitr, rmarkdown, kableExtra, testthat (>=  
3.0.0), survminer, survival

**Imports** purrr, data.table, foreach, future, doFuture, stats, utils,  
flexsurv, MASS, zoo, progressr, magrittr, rlang, tidyr

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**Depends** R (>= 2.10)

**URL** <https://jsanchezalv.github.io/WARDEN/>

**NeedsCompilation** no

**Author** Javier Sanchez Alvarez [aut, cre],  
Gabriel Lemyre [ctb],  
Valerie Aponte Ribero [ctb]

**Maintainer** Javier Sanchez Alvarez <javiersanchezeco@gmail.com>

**Repository** CRAN

**Date/Publication** 2025-07-10 07:10:06 UTC

## Contents

add_item . . . . .	3
add_item2 . . . . .	4
add_reactevt . . . . .	5
add_tte . . . . .	6
adj_val . . . . .	7
ast_as_list . . . . .	8
ceac_des . . . . .	10
cond_dirichlet . . . . .	11
cond_mvn . . . . .	11
create_indicators . . . . .	12
disc_cycle . . . . .	13
disc_cycle_v . . . . .	14
disc_instant . . . . .	15
disc_instant_v . . . . .	16
disc_ongoing . . . . .	16
disc_ongoing_v . . . . .	17
draw_tte . . . . .	18
evpi_des . . . . .	19
extract_elements_from_list . . . . .	20
extract_from_reactions . . . . .	21
extract_psa_result . . . . .	22
luck_adj . . . . .	23
modify_event . . . . .	24
modify_item . . . . .	25
modify_item_seq . . . . .	26
new_event . . . . .	27
pcond_gompertz . . . . .	28
pick_psa . . . . .	29
pick_val_v . . . . .	30
qbeta_mse . . . . .	32
qcond_exp . . . . .	33
qcond_gamma . . . . .	33
qcond_gompertz . . . . .	34
qcond_llogis . . . . .	35
qcond_lnorm . . . . .	35
qcond_norm . . . . .	36
qcond_weibull . . . . .	37
qcond_weibullPH . . . . .	37
qgamma_mse . . . . .	38
qtimecov . . . . .	39
random_stream . . . . .	42
rbeta_mse . . . . .	43
rcond_gompertz . . . . .	44
rcond_gompertz_lu . . . . .	44
rdirichlet . . . . .	45
rdirichlet_prob . . . . .	46

<code>add_item</code>	3
<code>replicate_profiles</code>	46
<code>rgamma_mse</code>	47
<code>rpoisgamma</code>	48
<code>run_sim</code>	49
<code>run_sim_parallel</code>	54
<code>sens_iterator</code>	58
<code>summary_results_det</code>	59
<code>summary_results_sens</code>	60
<code>summary_results_sim</code>	61
<code>tte.df</code>	62
<b>Index</b>	<b>63</b>

---

<code>add_item</code>	<i>Define parameters that may be used in model calculations (list)</i>
-----------------------	--

---

## Description

Define parameters that may be used in model calculations (list)

## Usage

```
add_item(.data = NULL, ...)
```

## Arguments

<code>.data</code>	Existing data
<code>...</code>	Items to define for the simulation

## Details

The functions to add/modify events/inputs use lists. Whenever several inputs/events are added or modified, it's recommended to group them within one function, as it reduces the computation cost. So rather than use two `add_item` with a list of one element, it's better to group them into a single `add_item` with a list of two elements.

Whenever a function is directly implemented which must be evaluated later and that has no object name attached (e.g., `pick_val_v`), it should be implemented after a first `add_item()` (empty or with content) to avoid confusing the `.data` argument, or wrapping the function within `substitute()`

## Value

A list of items

**Examples**

```

library(magrittr)

add_item(f1.idfs = 0)
add_item(util_idfs = if(psa_bool){rnorm(1,0.8,0.2)} else{0.8}, util.mbc = 0.6, cost_idfs = 2500)
common_inputs <- add_item() %>%
add_item(pick_val_v(
  base      = l_statics[["base"]],
  psa       = pick_psa(
    l_statics[["function"]],
    l_statics[["n"]],
    l_statics[["a"]],
    l_statics[["b"]]
  ),
  sens      = l_statics[[sens_name_used]],
  psa_ind   = psa_bool,
  sens_ind  = sensitivity_bool,
  indicator = indicators_statics,
  names_out = l_statics[["parameter_name"]]
)
)

```

---

add_item2	<i>Define parameters that may be used in model calculations (uses expressions)</i>
-----------	--

---

**Description**

Define parameters that may be used in model calculations (uses expressions)

**Usage**

```
add_item2(.data = NULL, input)
```

**Arguments**

.data	Existing data
input	Items to define for the simulation as an expression (i.e., using )

**Details**

The functions to add/modify events/inputs use lists. If chaining together add\_item2, it will just append the expressions together in the order established.

If using pick\_val\_v, note it should be used with the deploy\_env = TRUE argument so that add\_item2 process it correctly.

**Value**

A substituted expression to be evaluated by engine

**Examples**

```
library(magrittr)

add_item2(input = {fl.idfs <- 0})
add_item2(input = {
  util_idfs <- if(psa_bool){rnorm(1,0.8,0.2)} else{0.8}
  util.mbc <- 0.6
  cost_idfs <- 2500})
common_inputs <- add_item2(input = {
pick_val_v(
  base      = l_statics[["base"]],
  psa      = pick_psa(
    l_statics[["function"]],
    l_statics[["n"]],
    l_statics[["a"]],
    l_statics[["b"]]
  ),
  sens      = l_statics[[sens_name_used]],
  psa_ind   = psa_bool,
  sens_ind  = sensitivity_bool,
  indicator = indicators_statics,
  names_out = l_statics[["parameter_name"]],
  deploy_env = TRUE #Note this option must be active if using it at add_item2
)
}
)
```

---

add_reactvt	<i>Define the modifications to other events, costs, utilities, or other items affected by the occurrence of the event</i>
-------------	---

---

**Description**

Define the modifications to other events, costs, utilities, or other items affected by the occurrence of the event

**Usage**

```
add_reactvt(.data = NULL, name_evt, input)
```

**Arguments**

.data	Existing data for event reactions
name_evt	Name of the event for which reactions are defined.
input	Expressions that define what happens at the event, using functions as defined in the Details section

**Details**

There are a series of objects that can be used in this context to help define the event reactions.

The following functions may be used to define event reactions within this `add_reactevt()` function: `modify_item()` | Adds & Modifies items/flags/variables for future events (does not consider sequential) `modify_item_seq()` | Adds & Modifies items/flags/variables for future events in a sequential manner `new_event()` | Adds events to the vector of events for that patient `modify_event()` | Modifies existing events by changing their time

Apart from the items defined with `add_item()`, we can also use standard variables that are always defined within the simulation: `curtime` | Current event time (numeric) `prevtime` | Time of the previous event (numeric) `cur_evtlist` | Named vector of events that is yet to happen for that patient (named numeric vector) `evt` | Current event being processed (character) `i` | Patient being iterated (character) `simulation` | Simulation being iterated (numeric)

The model will run until `curtime` is set to `Inf`, so the event that terminates the model should modify `curtime` and set it to `Inf`.

The user can use `extract_from_reactions` function on the output to obtain a `data.frame` with all the relationships defined in the reactions in the model.

**Value**

A named list with the event name, and inside it the substituted expression saved for later evaluation

**Examples**

```
add_reactevt(name_evt = "start",input = {})
add_reactevt(name_evt = "ids",input = {modify_item(list("fl.idfs"= 0))})
```

---

add\_tte

*Define events and the initial event time*

---

**Description**

Define events and the initial event time

**Usage**

```
add_tte(.data = NULL, arm, evts, other_inp = NULL, input)
```

**Arguments**

.data	Existing data for initial event times
arm	The intervention for which the events and initial event times are defined
evts	A vector of the names of the events
other_inp	A vector of other input variables that should be saved during the simulation
input	The definition of initial event times for the events listed in the evts argument

**Details**

Events need to be separately defined for each intervention.

For each event that is defined in this list, the user needs to add a reaction to the event using the `add_reactevt()` function which will determine what calculations will happen at an event.

**Value**

A list of initial events and event times

**Examples**

```
add_tte(arm="int",evts = c("start","ttot","idfs","os"),
input={
start <- 0
idfs <- draw_tte(1,'lnorm',coef1=2, coef2=0.5)
ttot <- min(draw_tte(1,'lnorm',coef1=1, coef2=4),idfs)
os <- draw_tte(1,'lnorm',coef1=0.8, coef2=0.2)
})
```

---

adj\_val

*Adjusted Value Calculation*


---

**Description**

This function calculates an adjusted value over a time interval with optional discounting. This is useful for instances when adding cycles may not be desirable, so one can perform "cycle-like" calculations without needing cycles, offering performance speeds. See the vignette on avoiding cycles for an example in a model.

**Usage**

```
adj_val(curtime, nexttime, by, expression, discount = NULL)
```

**Arguments**

curtime	Numeric. The current time point.
nexttime	Numeric. The next time point. Must be greater than or equal to curtime.
by	Numeric. The step size for evaluation within the interval.
expression	An expression evaluated at each step. Use time as the variable within the expression.
discount	Numeric or NULL. The discount rate to apply, or NULL for no discounting.

**Details**

The user can use the `.time` variable to select the corresponding time of the sequence being evaluated. For example, in `curtime = 0`, `nexttime = 4`, `by = 1`, time would correspond to 0, 1, 2, 3. If using `nexttime = 4.2`, 0, 1, 2, 3, 4

**Value**

Numeric. The calculated adjusted value.

**Examples**

```
# Define a function or vector to evaluate
bs_age <- 1
vec <- 1:8/10

# Calculate adjusted value without discounting
adj_val(0, 4, by = 1, expression = vec[floor(.time + bs_age)])
adj_val(0, 4, by = 1, expression = .time * 1.1)

# Calculate adjusted value with discounting
adj_val(0, 4, by = 1, expression = vec[floor(.time + bs_age)], discount = 0.03)
```

---

ast_as_list	<i>Transform a substituted expression to its Abstract Syntax Tree (AST) as a list</i>
-------------	---

---

**Description**

Transform a substituted expression to its Abstract Syntax Tree (AST) as a list

**Usage**

```
ast_as_list(ee)
```

**Arguments**

ee	Substituted expression
----	------------------------



**Value**

Nested list with the Abstract Syntax Tree (AST)

**Examples**

```
expr <- substitute({
  a <- sum(5+7)
  modify_item(list(afsa=ifelse(TRUE,"asda",NULL)))
  modify_item_seq(list(
    o_other_q_gold1 = if(gold == 1) { utility } else { 0 },
    o_other_q_gold2 = if(gold == 2) { utility } else { 0 },
    o_other_q_gold3 = if(gold == 3) { utility } else { 0 },
    o_other_q_gold4 = if(gold == 4) { utility } else { 0 },
    o_other_q_on_dup = if(on_dup) { utility } else { 0 }
  ))
  if(a==1){
    modify_item(list(a=list(6+b)))
    modify_event(list(e_exn = curtime + 14 / days_in_year + qexp(rnd_exn, r_exn)))
  } else{
    modify_event(list(e_exn = curtime + 14 / days_in_year + qexp(rnd_exn, r_exn)))
    if(a>6){
      modify_item(list(a=8))
    }
  }
}

if (sel_resp_incl == 1 & on_dup == 1) {
  modify_event(list(e_response = curtime, z = 6))
}

})

out <- ast_as_list(expr)
```

---

ceac_des	<i>Calculate the cost-effectiveness acceptability curve (CEAC) for a DES model with a PSA result</i>
----------	--

---

### Description

Calculate the cost-effectiveness acceptability curve (CEAC) for a DES model with a PSA result

### Usage

```
ceac_des(wtp, results, interventions = NULL, sensitivity_used = 1)
```

### Arguments

wtp	Vector of length $\geq 1$ with the willingness to pay
results	The list object returned by run_sim()
interventions	A character vector with the names of the interventions to be used for the analysis
sensitivity_used	Integer signaling which sensitivity analysis to use

### Value

A data frame with the CEAC results

### Examples

```
res <- list(list(list(sensitivity_name = "", arm_list = c("int", "noint"
), total_lys = c(int = 9.04687362556945, noint = 9.04687362556945
), total_qalys = c(int = 6.20743830697466, noint = 6.18115138126336
), total_costs = c(int = 49921.6357486899, noint = 41225.2544659378
), total_lys_undisc = c(int = 10.8986618377039, noint = 10.8986618377039
), total_qalys_undisc = c(int = 7.50117621700097, noint = 7.47414569286751
), total_costs_undisc = c(int = 59831.3573929783, noint = 49293.1025437205
), c_default = c(int = 49921.6357486899, noint = 41225.2544659378
), c_default_undisc = c(int = 59831.3573929783, noint = 49293.1025437205
), q_default = c(int = 6.20743830697466, noint = 6.18115138126336
), q_default_undisc = c(int = 7.50117621700097, noint = 7.47414569286751
), merged_df = list(simulation = 1L, sensitivity = 1L))))
```

```
ceac_des(seq(from=10000, to=500000, by=10000), res)
```

---

cond_dirichlet	<i>Calculate conditional dirichlet values</i>
----------------	---

---

**Description**

Calculate conditional dirichlet values

**Usage**

```
cond_dirichlet(alpha, i, xi, full_output = FALSE)
```

**Arguments**

alpha	mean vector
i	index of the known parameter (1-based index)
xi	known value of the i-th parameter (should be >0)
full_output	boolean indicating whether to return the full list of parameters

**Details**

Function to compute conditional dirichlet values

**Value**

List of length 2, one with new mu and other with covariance parameters

**Examples**

```
alpha <- c(2, 3, 4)
i <- 2 # Index of the known parameter
xi <- 0.5 # Known value of the second parameter

# Compute the conditional alpha parameters with full output
cond_dirichlet(alpha, i, xi, full_output = TRUE)
```

---

cond_mvn	<i>Calculate conditional multivariate normal values</i>
----------	---

---

**Description**

Calculate conditional multivariate normal values

**Usage**

```
cond_mvn(mu, Sigma, i, xi, full_output = FALSE)
```

**Arguments**

mu	mean vector
Sigma	covariance matrix
i	index of the known parameter (1-based index)
xi	known value of the i-th parameter
full_output	boolean indicating whether to return the full list of parameters

**Details**

Function to compute conditional multivariate normal values

**Value**

List of length 2, one with new mu and other with covariance parameters

**Examples**

```
mu <- c(1, 2, 3)
Sigma <- matrix(c(0.2, 0.05, 0.1,
                 0.05, 0.3, 0.05,
                 0.1, 0.05, 0.4), nrow = 3)

i <- 1:2 # Index of the known parameter
xi <- c(1.2, 2.3) # Known value of the first parameter

cond_mvn(mu, Sigma, i, xi, full_output = TRUE)
```

---

create\_indicators      *Creates a vector of indicators (0 and 1) for sensitivity/DSA analysis*

---

**Description**

Creates a vector of indicators (0 and 1) for sensitivity/DSA analysis

**Usage**

```
create_indicators(sens, n_sensitivity, elem, n_elem_before = 0)
```

**Arguments**

sens	current analysis iterator
n_sensitivity	total number of analyses to be run
elem	vector of 0s and 1s of elements to iterate through (1 = parameter is to be included in scenario/DSA)
n_elem_before	Sum of 1s (# of parameters to be included in scenario/DSA) that go before elem

**Details**

n\_elem\_before is to be used when several indicators want to be used (e.g., for patient level and common level inputs) while facilitating readability of the code

**Value**

Numeric vector composed of 0 and 1, where value 1 will be used by pick\_val\_v to pick the corresponding index in its sens argument

**Examples**

```
create_indicators(10,20,c(1,1,1,1))
create_indicators(7,20,c(1,0,0,1,1,1,0,0,1,1),2)
```

---

disc_cycle	<i>Cycle discounting</i>
------------	--------------------------

---

**Description**

Cycle discounting

**Usage**

```
disc_cycle(
  lcldr = 0.035,
  lclprvtime = 0,
  cyclelength,
  lclcurtime,
  lclval,
  starttime = 0
)
```

**Arguments**

lcldr	The discount rate
lclprvtime	The time of the previous event in the simulation
cyclelength	The cycle length
lclcurtime	The time of the current event in the simulation
lclval	The value to be discounted
starttime	The start time for accrual of cycle costs (if not 0)

**Details**

Note this function counts both extremes of the interval, so the example below would consider 25 cycles, while disc\_cycle\_v leave the right interval open

**Value**

Double based on cycle discounting

**Examples**

```
disc_cycle(lclldr=0.035, lclprvtime=0, cyclelength=1/12, lclcurtime=2, lclval=500, starttime=0)
```

---

disc\_cycle\_v

*Cycle discounting for vectors*

---

**Description**

Cycle discounting for vectors

**Usage**

```
disc_cycle_v(  
  lclldr = 0.035,  
  lclprvtime = 0,  
  cyclelength,  
  lclcurtime,  
  lclval,  
  starttime = 0,  
  max_cycles = NULL  
)
```

**Arguments**

lclldr	The discount rate
lclprvtime	The time of the previous event in the simulation
cyclelength	The cycle length
lclcurtime	The time of the current event in the simulation
lclval	The value to be discounted
starttime	The start time for accrual of cycle costs (if not 0)
max_cycles	The maximum number of cycles

**Details**

This function performs cycle discounting, i.e., considers that the cost/qaly is accrued per cycles, and performs it automatically without needing to create new events. It can accommodate changes in cycle length/value/starttime (e.g., in the case of induction and maintenance doses) within the same item.

**Value**

Double vector based on cycle discounting

**Examples**

```
disc_cycle_v(lclldr=0.03, lclprvtime=0, cyclelength=1/12, lclcurtime=2, lclval=500, starttime=0)
disc_cycle_v(
  lclldr=0.000001,
  lclprvtime=0,
  cyclelength=1/12,
  lclcurtime=2,
  lclval=500,
  starttime=0,
  max_cycles = 4)

#Here we have a change in cycle length, max number of cycles and starttime at time 2
#(e.g., induction to maintenance)
#In the model, one would do this by redefining cycle_l, max_cycles and starttime
#of the corresponding item at a given event time.
disc_cycle_v(lclldr=0,
  lclprvtime=c(0,1,2,2.5),
  cyclelength=c(1/12, 1/12,1/2,1/2),
  lclcurtime=c(1,2,2.5,4), lclval=c(500,500,500,500),
  starttime=c(0,0,2,2), max_cycles = c(24,24,2,2)
)
```

---

disc\_instant

*Calculate instantaneous discounted costs or qalys*

---

**Description**

Calculate instantaneous discounted costs or qalys

**Usage**

```
disc_instant(lclldr = 0.035, lclcurtime, lclval)
```

**Arguments**

lclldr	The discount rate
lclcurtime	The time of the current event in the simulation
lclval	The value to be discounted

**Value**

Double based on discrete time discounting

**Examples**

```
disc_instant(lclldr=0.035, lclcurtime=3, lclval=2500)
```

---

disc_instant_v	<i>Calculate instantaneous discounted costs or qalys for vectors</i>
----------------	--

---

**Description**

Calculate instantaneous discounted costs or qalys for vectors

**Usage**

```
disc_instant_v(lclldr = 0.035, lclcurtime, lclval)
```

**Arguments**

lclldr	The discount rate
lclcurtime	The time of the current event in the simulation
lclval	The value to be discounted

**Value**

Double based on discrete time discounting

**Examples**

```
disc_instant_v(lclldr=0.035, lclcurtime=3, lclval=2500)
```

---

disc_ongoing	<i>Calculate discounted costs and qalys between events</i>
--------------	--

---

**Description**

Calculate discounted costs and qalys between events

**Usage**

```
disc_ongoing(lclldr = 0.035, lclprvtime, lclcurtime, lclval)
```



**Arguments**

lclldr	The discount rate
lclprvtime	The time of the previous event in the simulation
lclcurtime	The time of the current event in the simulation
lclval	The value to be discounted

**Value**

Double based on continuous time discounting

**Examples**

```
disc_ongoing(lclldr=0.035,lclprvtime=0.5, lclcurtime=3, lclval=2500)
```

---

disc_ongoing_v	<i>Calculate discounted costs and qalys between events for vectors</i>
----------------	--

---

**Description**

Calculate discounted costs and qalys between events for vectors

**Usage**

```
disc_ongoing_v(lclldr = 0.035, lclprvtime, lclcurtime, lclval)
```

**Arguments**

lclldr	The discount rate
lclprvtime	The time of the previous event in the simulation
lclcurtime	The time of the current event in the simulation
lclval	The value to be discounted

**Value**

Double based on continuous time discounting

**Examples**

```
disc_ongoing_v(lclldr=0.035,lclprvtime=0.5, lclcurtime=3, lclval=2500)
```

---

draw\_tte

*Draw a time to event from a list of parametric survival functions*


---

**Description**

Draw a time to event from a list of parametric survival functions

**Usage**

```
draw_tte(
  n_chosen,
  dist,
  coef1 = NULL,
  coef2 = NULL,
  coef3 = NULL,
  ...,
  beta_tx = 1,
  seed = NULL
)
```

**Arguments**

n_chosen	The number of observations to be drawn
dist	The distribution; takes values 'lnorm', 'norm', 'mvnorm', 'weibullPH', 'weibull', 'llogis', 'gompertz', 'gengar'
coef1	First coefficient of the distribution, defined as in the coef() output on a flex-survreg object (rate in "rpoisgamma")
coef2	Second coefficient of the distribution, defined as in the coef() output on a flex-survreg object (theta in "rpoisgamma")
coef3	Third coefficient of the distribution, defined as in the coef() output on a flex-survreg object (not used in "rpoisgamma")
...	Additional arguments to be used by the specific distribution (e.g., return_ind_rate if dist = "poisgamma")
beta_tx	Parameter in natural scale applied in addition to the scale/rate coefficient -e.g., a HR if used in an exponential- (not used in "rpoisgamma" nor "beta")
seed	An integer which will be used to set the seed for this draw.

**Details**

Other arguments relevant to each function can be called directly

**Value**

A vector of time to event estimates from the given parameters

**Examples**

```
draw_tte(n_chosen=1,dist='exp',coef1=1,beta_tx=1)
draw_tte(n_chosen=10,"poisgamma",coef1=1,coef2=1,obs_time=1,return_ind_rate=FALSE)
```

---

evpi_des	<i>Calculate the Expected Value of Perfect Information (EVPI) for a DES model with a PSA result</i>
----------	---

---

**Description**

Calculate the Expected Value of Perfect Information (EVPI) for a DES model with a PSA result

**Usage**

```
evpi_des(wtp, results, interventions = NULL, sensitivity_used = 1)
```

**Arguments**

wtp                    Vector of length  $\geq 1$  with the willingness to pay

results                The list object returned by run\_sim()

interventions        A character vector with the names of the interventions to be used for the analysis

sensitivity\_used       Integer signaling which sensitivity analysis to use

**Value**

A data frame with the EVPI results

**Examples**

```
res <- list(list(list(sensitivity_name = "", arm_list = c("int", "noint"
), total_lys = c(int = 9.04687362556945, noint = 9.04687362556945
), total_qalys = c(int = 6.20743830697466, noint = 6.18115138126336
), total_costs = c(int = 49921.6357486899, noint = 41225.2544659378
), total_lys_undisc = c(int = 10.8986618377039, noint = 10.8986618377039
), total_qalys_undisc = c(int = 7.50117621700097, noint = 7.47414569286751
), total_costs_undisc = c(int = 59831.3573929783, noint = 49293.1025437205
), c_default = c(int = 49921.6357486899, noint = 41225.2544659378
), c_default_undisc = c(int = 59831.3573929783, noint = 49293.1025437205
), q_default = c(int = 6.20743830697466, noint = 6.18115138126336
), q_default_undisc = c(int = 7.50117621700097, noint = 7.47414569286751
), merged_df = list(simulation = 1L, sensitivity = 1L))))

evpi_des(seq(from=10000,to=500000,by=10000),res)
```

---

```
extract_elements_from_list
```

*Extracts items and events by looking into assignments, modify\_item, modify\_item\_seq, modify\_event and new\_event*

---

### Description

Extracts items and events by looking into assignments, modify\_item, modify\_item\_seq, modify\_event and new\_event

### Usage

```
extract_elements_from_list(node, conditional_flag = FALSE)
```

### Arguments

node	Relevant node within the nested AST list
conditional_flag	Boolean whether the statement is contained within a conditional statement

### Value

A data.frame with the relevant item/event, the event where it's assigned, and whether it's contained within a conditional statement

### Examples

```
expr <- substitute({
  a <- sum(5+7)
  modify_item(list(afsa=ifelse(TRUE,"asda",NULL)))
  modify_item_seq(list(
    o_other_q_gold1 = if(gold == 1) { utility } else { 0 },
    o_other_q_gold2 = if(gold == 2) { utility } else { 0 },
    o_other_q_gold3 = if(gold == 3) { utility } else { 0 },
    o_other_q_gold4 = if(gold == 4) { utility } else { 0 },
    o_other_q_on_dup = if(on_dup) { utility } else { 0 }
  ))
  if(a==1){
    modify_item(list(a=list(6+b)))
```

```
    modify_event(list(e_exn = curtime + 14 / days_in_year + qexp(rnd_exn, r_exn)))
  } else{
    modify_event(list(e_exn = curtime + 14 / days_in_year + qexp(rnd_exn, r_exn)))
    if(a>6){
      modify_item(list(a=8))
    }
  }

if (sel_resp_incl == 1 & on_dup == 1) {
  modify_event(list(e_response = curtime, z = 6))
}

})

out <- ast_as_list(expr)

results <- extract_elements_from_list(out)
```

---

extract\_from\_reactions

*Extract all items and events and their interactions from the event reactions list*

---

### **Description**

Extract all items and events and their interactions from the event reactions list

### **Usage**

```
extract_from_reactions(reactions)
```

### **Arguments**

reactions      list generated through add\_reactevt

### **Value**

A data.frame with the relevant item/event, the event where it's assigned, and whether it's contained within a conditional statement

**Examples**

```

evt_react_list2 <-
  add_reactevt(name_evt = "sick",
    input = {modify_item(list(a=1+5/3))
      assign("W", 5 + 3 / 6 )
      x[5] <- 18
      for(i in 1:5){
        assign(paste0("x_",i),5+3)
      }
      if(j == TRUE){
        y[["w"]] <- 612-31+3
      }#'
      q_default <- 0
      c_default <- 0
      curtime <- Inf
      d <- c <- k <- 67
    })

extract_from_reactions(evt_react_list2)

```

---

extract_psa_result	<i>Extract PSA results from a treatment</i>
--------------------	---

---

**Description**

Extract PSA results from a treatment

**Usage**

```
extract_psa_result(x, element)
```

**Arguments**

x	The output_sim data frame from the list object returned by run_sim()
element	Variable for which PSA results are being extracted (single string)

**Value**

A dataframe with PSA results from the specified intervention

**Examples**

```

res <- list(list(list(sensitivity_name = "", arm_list = c("int", "noint"
), total_lys = c(int = 9.04687362556945, noint = 9.04687362556945
), total_qalys = c(int = 6.20743830697466, noint = 6.18115138126336
), total_costs = c(int = 49921.6357486899, noint = 41225.2544659378
), total_lys_undisc = c(int = 10.8986618377039, noint = 10.8986618377039

```

```

), total_qalys_undisc = c(int = 7.50117621700097, noint = 7.47414569286751
), total_costs_undisc = c(int = 59831.3573929783, noint = 49293.1025437205
), c_default = c(int = 49921.6357486899, noint = 41225.2544659378
), c_default_undisc = c(int = 59831.3573929783, noint = 49293.1025437205
), q_default = c(int = 6.20743830697466, noint = 6.18115138126336
), q_default_undisc = c(int = 7.50117621700097, noint = 7.47414569286751
), merged_df = list(simulation = 1L, sensitivity = 1L)))

```

```
extract_psa_result(res[[1]], "total_costs")
```

---

luck_adj	<i>Perform luck adjustment</i>
----------	--------------------------------

---

### Description

Perform luck adjustment

### Usage

```
luck_adj(prevsurv, cursurv, luck, condq = TRUE)
```

### Arguments

prevsurv	Value of the previous survival
cursurv	Value of the current survival
luck	Luck used to be adjusted (number between 0 and 1)
condq	Conditional quantile approach or standard approach

### Details

This function performs the luck adjustment automatically for the user, returning the adjusted luck number. Luck is interpreted in the same fashion as is standard in R (higher luck, higher time to event).

Note that if TTE is predicted using a conditional quantile function (e.g., conditional gompertz, conditional quantile weibull...) prevsurv and cursurv are the unconditional survival using the "previous" parametrization but at the previous time for presurv and at the current time for cursurv. For other distributions, presurv is the survival up to current time using the previous parametrization, and cursurv is the survival up to current time using the current parametrization.

Note that the advantage of the conditional quantile function is that it does not need the new parametrization to update the luck, which makes this approach computationally more efficient. This function can also work with vectors, which could allow to update multiple lucks in a single approach, and it can preserve names

### Value

Adjusted luck number between 0 and 1

**Examples**

```

luck_adj(prevsurv = 0.8,
  cursurv = 0.7,
  luck = 0.5,
  condq = TRUE)

luck_adj(prevsurv = c(1,0.8,0.7),
  cursurv = c(0.7,0.6,0.5),
  luck = setNames(c(0.5,0.6,0.7),c("A","B","C")),
  condq = TRUE)

luck_adj(prevsurv = 0.8,
  cursurv = 0.7,
  luck = 0.5,
  condq = FALSE) #different results

#Unconditional approach, timepoint of change is 25,
# parameter goes from 0.02 at time 10 to 0.025 to 0.015 at time 25,
# starting luck is 0.37
new_luck <- luck_adj(prevsurv = 1 - pweibull(q=10,3,1/0.02),
  cursurv = 1 - pweibull(q=10,3,1/0.025),
  luck = 0.37,
  condq = FALSE) #time 10 change

new_luck <- luck_adj(prevsurv = 1 - pweibull(q=25,3,1/0.025),
  cursurv = 1 - pweibull(q=25,3,1/0.015),
  luck = new_luck,
  condq = FALSE) #time 25 change

qweibull(new_luck, 3, 1/0.015) #final TTE

#Conditional quantile approach
new_luck <- luck_adj(prevsurv = 1-pweibull(q=0,3,1/0.02),
  cursurv = 1- pweibull(q=10,3,1/0.02),
  luck = 0.37,
  condq = TRUE) #time 10 change, previous time is 0 so prevsurv will be 1

new_luck <- luck_adj(prevsurv = 1-pweibull(q=10,3,1/0.025),
  cursurv = 1- pweibull(q=25,3,1/0.025),
  luck = new_luck,
  condq = TRUE) #time 25 change

qcond_weibull(rnd = new_luck,
  shape = 3,
  scale = 1/0.015,
  lower_bound = 25) + 25 #final TTE

```



**Description**

Modify the time of existing events

**Usage**

```
modify_event(evt, create_if_null = TRUE)
```

**Arguments**

`evt` A list of events and their times

`create_if_null` A boolean. If TRUE, it will create non-existing events with the chosen time to event. If FALSE, it will ignore those.

**Details**

The functions to add/modify events/inputs use lists. Whenever several inputs/events are added or modified, it's recommended to group them within one function, as it reduces the computation cost. So rather than use two `modify_event` with a list of one element, it's better to group them into a single `modify_event` with a list of two elements.

This function does not evaluate sequentially.

This function is intended to be used only within the `add_reactevt` function in its input parameter and should not be run elsewhere or it will return an error.

**Value**

No return value, modifies/adds event to `cur_evtlist` and integrates it with the main list for storage

**Examples**

```
add_reactevt(name_evt = "ids", input = {modify_event(list("os"=5))})
```

---

modify\_item

*Modify the value of existing items*

---

**Description**

Modify the value of existing items

**Usage**

```
modify_item(list_item)
```

**Arguments**

`list_item` A list of items and their values or expressions

**Details**

The functions to add/modify events/inputs use lists. Whenever several inputs/events are added or modified, it's recommended to group them within one function, as it reduces the computation cost. So rather than use two `modify_item` with a list of one element, it's better to group them into a single `modify_item` with a list of two elements.

Note that `modify_item` nor `modify_item_seq` can work on subelements (e.g., `modify_item(list(obj$item = 5))` will not work as intended, for that is better to assign directly using the expression approach, so `obj$item <- 5`).

Costs and utilities can be modified by using the construction `type_name_category`, where `type` is either "qaly" or "cost", `name` is the name (e.g., "default") and `category` is the category used (e.g., "instant"), so one could pass `cost_default_instant` and modify the cost. This will overwrite the value defined in the corresponding cost/utility section.

This function is intended to be used only within the `add_reactevt` function in its input parameter and should not be run elsewhere or it will return an error.

**Value**

No return value, modifies/adds item to the environment and integrates it with the main list for storage

**Examples**

```
add_reactevt(name_evt = "ids", input = {modify_item(list("cost.it"=5))})
```

---

modify_item_seq	<i>Modify the value of existing items</i>
-----------------	---

---

**Description**

Modify the value of existing items

**Usage**

```
modify_item_seq(...)
```

**Arguments**

... A list of items and their values or expressions. Will be evaluated sequentially (so one could have `list(a= 1, b = a +2 )`)

## Details

The functions to add/modify events/inputs use lists. Whenever several inputs/events are added or modified, it's recommended to group them within one function, as it reduces the computation cost. So rather than use two `modify_item` with a list of one element, it's better to group them into a single `modify_item` with a list of two elements.

Note that `modify_item` nor `modify_item_seq` can work on subelements (e.g., `modify_item_seq(list(obj$item = 5))` will not work as intended, for that is better to assign directly using the expression approach, so `obj$item <- 5`).

Costs and utilities can be modified by using the construction `type_name_category`, where `type` is either "qaly" or "cost", `name` is the name (e.g., "default") and `category` is the category used (e.g., "instant"), so one could pass `cost_default_instant` and modify the cost. This will overwrite the value defined in the corresponding cost/utility section.

The function is different from `modify_item` in that this function evaluates sequentially the arguments within the list passed. This implies a slower performance relative to `modify_item`, but it can be more cleaner and convenient in certain instances.

This function is intended to be used only within the `add_reactevt` function in its input parameter and should not be run elsewhere or it will return an error.

## Value

No return value, modifies/adds items sequentially and deploys to the environment and with the main list for storage

## Examples

```
add_reactevt(name_evt = "ids", input = {
  modify_item_seq(list(cost.ids = 500, cost.tx = cost.ids + 4000))
})
```

---

new\_event

*Generate new events to be added to existing vector of events*

---

## Description

Generate new events to be added to existing vector of events

## Usage

```
new_event(evt)
```

## Arguments

evt                      Event name and event time

**Details**

The functions to add/modify events/inputs use lists. Whenever several inputs/events are added or modified, it's recommended to group them within one function, as it reduces the computation cost. So rather than use two `new_event` with a list of one element, it's better to group them into a single `new_event` with a list of two elements.

This function is intended to be used only within the `add_reactevt` function in its input parameter and should not be run elsewhere or it will return an error.

**Value**

No return value, adds event to `cur_evtlist` and integrates it with the main list for storage

**Examples**

```
add_reactevt(name_evt = "ids", input = {new_event(list("ae"=5))})
```

---

pcond_gompertz	<i>Survival Probaility function for conditional Gompertz distribution (lower bound only)</i>
----------------	--

---

**Description**

Survival Probaility function for conditional Gompertz distribution (lower bound only)

**Usage**

```
pcond_gompertz(time = 1, shape, rate, lower_bound = 0)
```

**Arguments**

time	Vector of times
shape	The shape parameter of the Gompertz distribution, defined as in the <code>coef()</code> output on a <code>flexsurvreg</code> object
rate	The rate parameter of the Gompertz distribution, defined as in the <code>coef()</code> output on a <code>flexsurvreg</code> object
lower_bound	The lower bound of the conditional distribution

**Value**

Estimate(s) from the conditional Gompertz distribution based on given parameters

**Examples**

```
pcond_gompertz(time=1, shape=0.05, rate=0.01, lower_bound = 50)
```

---

pick_psa	<i>Helper function to create a list with random draws or whenever a series of functions needs to be called. Can be implemented within pick_val_v.</i>
----------	---

---

### Description

Helper function to create a list with random draws or whenever a series of functions needs to be called. Can be implemented within pick\_val\_v.

### Usage

```
pick_psa(f, ...)
```

### Arguments

f	A string or vector of strings with the function to be called, e.g., "rnorm"
...	parameters to be passed to the function (e.g., if "rnorm", arguments n, mean, sd)

### Details

This function can be used to pick values for the PSA within pick\_val\_v.

The function will ignore NA items within the respective parameter (see example below). If an element in f is NA (e.g., a non PSA input) then it will return NA as its value This feature is convenient when mixing distributions with different number of arguments, e.g., rnorm and rgengamma.

While it's slightly lower than individually calling each function, it makes the code easier to read and more transparent

### Value

List with length equal to f of parameters called

### Examples

```
params <- list(
  param=list("a","b"),
  dist=list("rlnorm","rnorm"),
  n=list(4,1),
  a=list(c(1,2,3,4),1),
  b=list(c(0.5,0.5,0.5,0.5),0.5),
  dsa_min=list(c(1,2,3,4),2),
  dsa_max=list(c(1,2,3,4),3)
)
pick_psa(params[["dist"]],params[["n"]],params[["a"]],params[["b"]])

#It works with functions that require different number of parameters
params <- list(
  param=list("a","b","c"),
```

```

dist=list("rlnorm","rnorm","rgamma"),
n=list(4,1,1),
a=list(c(1,2,3,4),1,0),
b=list(c(0.5,0.5,0.5,0.5),0.5,1),
c=list(NA,NA,0.2),
dsa_min=list(c(1,2,3,4),2,1),
dsa_max=list(c(1,2,3,4),3,3)
)

pick_psa(params[["dist"]],params[["n"]],params[["a"]],params[["b"]],params[["c"]])

#Can be combined with multiple type of functions and distributions if parameters are well located

params <- list(
param=list("a","b","c","d"),
dist=list("rlnorm","rnorm","rgamma","draw_tte"),
n=list(4,1,1,1),
a=list(c(1,2,3,4),1,0,"norm"),
b=list(c(0.5,0.5,0.5,0.5),0.5,1,1),
c=list(NA,NA,0.2,0.5),
c=list(NA,NA,NA,NA), #NA arguments will be ignored
dsa_min=list(c(1,2,3,4),2,1,0),
dsa_max=list(c(1,2,3,4),3,3,2)
)

```

---

pick\_val\_v

*Select which values should be applied in the corresponding loop for several values (vector or list).*

---

### Description

Select which values should be applied in the corresponding loop for several values (vector or list).

### Usage

```

pick_val_v(
  base,
  psa,
  sens,
  psa_ind = psa_bool,
  sens_ind = sens_bool,
  indicator,
  indicator_psa = NULL,
  names_out = NULL,
  indicator_sens_binary = TRUE,
  sens_iterator = NULL,
  distributions = NULL,
  covariances = NULL,
  deploy_env = FALSE
)

```

**Arguments**

base	Value if no PSA/DSA/Scenario
psa	Value if PSA
sens	Value if DSA/Scenario
psa_ind	Boolean whether PSA is active
sens_ind	Boolean whether Scenario/DSA is active
indicator	Indicator which checks whether the specific parameter/parameters is/are active in the DSA or Scenario loop
indicator_psa	Indicator which checks whether the specific parameter/parameters is/are active in the PSA loop. If NULL, it's assumed to be a vector of 1s of length equal to length(indicator)
names_out	Names to give the output list
indicator_sens_binary	Boolean, TRUE if parameters will be varied fully, FALSE if some elements of the parameters may be changed but not all
sens_iterator	Current iterator number of the DSA/scenario being run, e.g., 5 if it corresponds to the 5th DSA parameter being changed
distributions	List with length equal to length of base where the distributions are stored
covariances	List with length equal to length of base where the variance/covariances are stored (only relevant if multivariate normal are being used)
deploy_env	Boolean, if TRUE will deploy all objects in the environment where the function is called for. Must be active if using add_item2 (and FALSE if using add_item)

**Details**

This function can be used with vectors or lists, but will always return a list. Lists should be used when correlated variables are introduced to make sure the selector knows how to choose among those. This function allows to choose between using an approach where only the full parameters are varied, and an approach where subelements of the parameters can be changed.

**Value**

List used for the inputs

**Examples**

```
pick_val_v(base = list(0,0),
           psa =list(rnorm(1,0,0.1),rnorm(1,0,0.1)),
           sens = list(2,3),
           psa_ind = FALSE,
           sens_ind = TRUE,
           indicator=list(1,2),
           indicator_sens_binary = FALSE,
           sens_iterator = 2,
           distributions = list("rnorm","rnorm")
)
```

```

pick_val_v(base = list(2,3,c(1,2)),
  psa =sapply(1:3,
    function(x) eval(call(
      c("rnorm","rnorm","mvrnorm")[[x]],
      1,
      c(2,3,list(c(1,2)))[[x]],
      c(0.1,0.1,list(matrix(c(1,0.1,0.1,1),2,2)))[[x]]
    ))),
  sens = list(4,5,c(1.3,2.3)),
  psa_ind = FALSE,
  sens_ind = TRUE,
  indicator=list(1,2,c(3,4)),
  names_out=c("util","util2","correlated_vector") ,
  indicator_sens_binary = FALSE,
  sens_iterator = 4,
  distributions = list("rnorm","rnorm","mvrnorm"),
  covariances = list(0.1,0.1,matrix(c(1,0.1,0.1,1),2,2))
)

```

---

qbeta\_mse

*Draw from a beta distribution based on mean and se (quantile)*


---

### Description

Draw from a beta distribution based on mean and se (quantile)

### Usage

```
qbeta_mse(q, mean_v, se)
```

### Arguments

q	Quantiles to be used
mean_v	A vector of the mean values
se	A vector of the standard errors of the means

### Value

A single estimate from the beta distribution based on given parameters

### Examples

```
qbeta_mse(q=0.5,mean_v=0.8,se=0.2)
```



---

qcond\_exp                      *Conditional quantile function for exponential distribution*

---

**Description**

Conditional quantile function for exponential distribution

**Usage**

```
qcond_exp(rnd = 0.5, rate)
```

**Arguments**

rnd	Vector of quantiles
rate	The rate parameter

Note taht the conditional quantile for an exponential is independent of time due to constant hazard

**Value**

Estimate(s) from the conditional exponential distribution based on given parameters

**Examples**

```
qcond_exp(rnd = 0.5, rate = 3)
```

---

qcond\_gamma                      *Conditional quantile function for gamma distribution*

---

**Description**

Conditional quantile function for gamma distribution

**Usage**

```
qcond_gamma(rnd = 0.5, shape, rate, lower_bound = 0, s_obs)
```

**Arguments**

rnd	Vector of quantiles
shape	The shape parameter
rate	The rate parameter
lower_bound	The lower bound to be used (current time)
s_obs	is the survival observed up to lower_bound time, normally defined from time 0 as $1 - \text{pgamma}(q = \text{lower\_bound}, \text{rate}, \text{shape})$ but may be different if parametrization has changed previously

**Value**

Estimate(s) from the conditional gamma distribution based on given parameters

**Examples**

```
qcond_gamma(rnd = 0.5, shape = 1.06178, rate = 0.01108, lower_bound = 1, s_obs=0.8)
```

---

qcond_gompertz	<i>Quantile function for conditional Gompertz distribution (lower bound only)</i>
----------------	---

---

**Description**

Quantile function for conditional Gompertz distribution (lower bound only)

**Usage**

```
qcond_gompertz(rnd = 0.5, shape, rate, lower_bound = 0)
```

**Arguments**

rnd	Vector of quantiles
shape	The shape parameter of the Gompertz distribution, defined as in the coef() output on a flexsurvreg object
rate	The rate parameter of the Gompertz distribution, defined as in the coef() output on a flexsurvreg object
lower_bound	The lower bound of the conditional distribution

**Value**

Estimate(s) from the conditional Gompertz distribution based on given parameters

**Examples**

```
qcond_gompertz(rnd=0.5, shape=0.05, rate=0.01, lower_bound = 50)
```

---

qcond\_llogis                      *Conditional quantile function for loglogistic distribution*

---

**Description**

Conditional quantile function for loglogistic distribution

**Usage**

```
qcond_llogis(rnd = 0.5, shape, scale, lower_bound = 0)
```

**Arguments**

rnd	Vector of quantiles
shape	The shape parameter
scale	The scale parameter
lower_bound	The lower bound to be used (current time)

**Value**

Estimate(s) from the conditional loglogistic distribution based on given parameters

**Examples**

```
qcond_llogis(rnd = 0.5, shape = 1, scale = 1, lower_bound = 1)
```

---

qcond\_lnorm                      *Conditional quantile function for lognormal distribution*

---

**Description**

Conditional quantile function for lognormal distribution

**Usage**

```
qcond_lnorm(rnd = 0.5, meanlog, sdlog, lower_bound = 0, s_obs)
```

**Arguments**

rnd	Vector of quantiles
meanlog	The meanlog parameter
sdlog	The sdlog parameter
lower_bound	The lower bound to be used (current time)
s_obs	is the survival observed up to lower_bound time, normally defined from time 0 as $1 - \text{plnorm}(q = \text{lower\_bound}, \text{meanlog}, \text{sdlog})$ but may be different if parametrization has changed previously

**Value**

Estimate(s) from the conditional lognormal distribution based on given parameters

**Examples**

```
qcond_lnorm(rnd = 0.5, meanlog = 1, sdlog = 1, lower_bound = 1, s_obs=0.8)
```

---

qcond\_norm

*Conditional quantile function for normal distribution*

---

**Description**

Conditional quantile function for normal distribution

**Usage**

```
qcond_norm(rnd = 0.5, mean, sd, lower_bound = 0, s_obs)
```

**Arguments**

rnd	Vector of quantiles
mean	The mean parameter
sd	The sd parameter
lower_bound	The lower bound to be used (current time)
s_obs	is the survival observed up to lower_bound time, normally defined from time 0 as $1 - \text{pnorm}(q = \text{lower\_bound}, \text{mean}, \text{sd})$ but may be different if parametrization has changed previously

**Value**

Estimate(s) from the conditional normal distribution based on given parameters

**Examples**

```
qcond_norm(rnd = 0.5, mean = 1, sd = 1, lower_bound = 1, s_obs=0.8)
```

---

qcond_weibull	<i>Conditional quantile function for weibull distribution</i>
---------------	---

---

**Description**

Conditional quantile function for weibull distribution

**Usage**

```
qcond_weibull(rnd = 0.5, shape, scale, lower_bound = 0)
```

**Arguments**

rnd	Vector of quantiles
shape	The shape parameter as in R stats package weibull
scale	The scale parameter as in R stats package weibull
lower_bound	The lower bound to be used (current time)

**Value**

Estimate(s) from the conditional weibull distribution based on given parameters

**Examples**

```
qcond_weibull(rnd = 0.5, shape = 3, scale = 66.66, lower_bound = 50)
```

---

qcond_weibullPH	<i>Conditional quantile function for WeibullPH (flexsurv)</i>
-----------------	---

---

**Description**

Conditional quantile function for WeibullPH (flexsurv)

**Usage**

```
qcond_weibullPH(rnd = 0.5, shape, scale, lower_bound = 0)
```

**Arguments**

rnd	Vector of quantiles (between 0 and 1)
shape	Shape parameter of WeibullPH
scale	Scale (rate) parameter of WeibullPH (i.e., as in hazard = scale * t^(shape - 1))
lower_bound	Lower bound (current time)

**Value**

Estimate(s) from the conditional weibullPH distribution based on given parameters

**Examples**

```
qcond_weibullPH(rnd = 0.5, shape = 2, scale = 0.01, lower_bound = 5)
```

---

qgamma\_mse

*Use quantiles from a gamma distribution based on mean and se*

---

**Description**

Use quantiles from a gamma distribution based on mean and se

**Usage**

```
qgamma_mse(q = 1, mean_v, se, seed = NULL)
```

**Arguments**

q	Quantile to draw
mean_v	A vector of the mean values
se	A vector of the standard errors of the means
seed	An integer which will be used to set the seed for this draw.

**Value**

A single estimate from the gamma distribution based on given parameters

**Examples**

```
qgamma_mse(q=0.5, mean_v=0.8, se=0.2)
```

---

qtimecov	<i>Draw Time-to-Event with Time-Dependent Covariates and Luck Adjustment</i>
----------	--

---

### Description

Simulate a time-to-event (TTE) from a parametric distribution with parameters varying over time. User provides parameter functions and distribution name. The function uses internal survival and conditional quantile functions, plus luck adjustment to simulate the event time. See the vignette on avoiding cycles for an example in a model.

### Usage

```
qtimecov(
  luck,
  a_fun,
  b_fun = function(.time) NA,
  dist,
  dt = 0.1,
  max_time = 100,
  return_luck = FALSE,
  start_time = 0
)
```

### Arguments

luck	Numeric between 0 and 1. Initial random quantile (luck).
a_fun	Function of time .time returning the first distribution parameter (e.g., rate, shape, meanlog).
b_fun	Function of time .time returning the second distribution parameter (e.g., scale, sdlog). Defaults to a function returning NA.
dist	Character string specifying the distribution. Supported: "exp", "gamma", "lnorm", "norm", "weibull", "llogis", "gompertz".
dt	Numeric. Time step increment to update parameters and survival. Default 0.1.
max_time	Numeric. Max allowed event time to prevent infinite loops. Default 100.
return_luck	Boolean. If TRUE, returns a list with tte and luck (useful if max_time caps TTE)
start_time	Numeric. Time to use as a starting point of reference (e.g., curtime).

### Details

The objective of this function is to avoid the user to have cycle events with the only scope of updating some variables that depend on time and re-evaluate a TTE. The idea is that this function should only be called at start and when an event impacts a variable (e.g., stroke event impacting death TTE), in which case it would need to be called again at that point. In that case, the user would need to call e.g., `a <- qtimecov` with `max_time = curtime`, `return_luck = TRUE` arguments, and then

call it again with no `max_time`, `return_luck = FALSE`, and `luck = a$luck`, `start_time=a$tte` (so there is no need to add `curtime` to the resulting time).

It's recommended to play with `dt` argument to balance running time and precision of the estimates. For example, if we know we only update the equation annually (not continuously), then we could just set `dt = 1`, which would make computations faster.

## Value

Numeric. Simulated time-to-event.

## Examples

```
param_fun_factory <- function(p0, p1, p2, p3) {
  function(.time) p0 + p1*.time + p2*.time^2 + p3*(floor(.time) + 1)
}
```

```
set.seed(42)
```

```
# 1. Exponential Example
```

```
rate_exp <- param_fun_factory(0.1, 0, 0, 0)
qtimecov(
  luck = runif(1),
  a_fun = rate_exp,
  dist = "exp"
)
```

```
# 2. Gamma Example
```

```
shape_gamma <- param_fun_factory(2, 0, 0, 0)
rate_gamma <- param_fun_factory(0.2, 0, 0, 0)
qtimecov(
  luck = runif(1),
  a_fun = shape_gamma,
  b_fun = rate_gamma,
  dist = "gamma"
)
```

```
# 3. Lognormal Example
```

```
meanlog_lnorm <- param_fun_factory(log(10) - 0.5*0.5^2, 0, 0, 0)
sdlog_lnorm <- param_fun_factory(0.5, 0, 0, 0)
qtimecov(
  luck = runif(1),
  a_fun = meanlog_lnorm,
  b_fun = sdlog_lnorm,
  dist = "lnorm"
)
```

```
# 4. Normal Example
```

```
mean_norm <- param_fun_factory(10, 0, 0, 0)
sd_norm <- param_fun_factory(2, 0, 0, 0)
```



```

qtimecov(
  luck = runif(1),
  a_fun = mean_norm,
  b_fun = sd_norm,
  dist = "norm"
)

# 5. Weibull Example
shape_weibull <- param_fun_factory(2, 0, 0, 0)
scale_weibull <- param_fun_factory(10, 0, 0, 0)
qtimecov(
  luck = runif(1),
  a_fun = shape_weibull,
  b_fun = scale_weibull,
  dist = "weibull"
)

# 6. Loglogistic Example
shape_llogis <- param_fun_factory(2.5, 0, 0, 0)
scale_llogis <- param_fun_factory(7.6, 0, 0, 0)
qtimecov(
  luck = runif(1),
  a_fun = shape_llogis,
  b_fun = scale_llogis,
  dist = "llogis"
)

# 7. Gompertz Example
shape_gomp <- param_fun_factory(0.01, 0, 0, 0)
rate_gomp <- param_fun_factory(0.091, 0, 0, 0)
qtimecov(
  luck = runif(1),
  a_fun = shape_gomp,
  b_fun = rate_gomp,
  dist = "gompertz"
)

#Time varying example, with change at time 8
rate_exp <- function(.time) 0.1 + 0.01*.time * 0.00001*.time^2
rate_exp2 <- function(.time) 0.2 + 0.02*.time
time_change <- 8
init_luck <- 0.95

a <- qtimecov(luck = init_luck,a_fun = rate_exp,dist = "exp", dt = 0.005,
             max_time = time_change, return_luck = TRUE)
qtimecov(luck = a$luck,a_fun = rate_exp2,dist = "exp", dt = 0.005, start_time=a$time)

#An example of how it would work in the model, this would also work with time varying covariates!
rate_exp <- function(.time) 0.1

```

```

rate_exp2 <- function(.time) 0.2
rate_exp3 <- function(.time) 0.3
time_change <- 10 #evt 1
time_change2 <- 15 #evt2
init_luck <- 0.95
#at start, we would just draw TTE
qtimecov(luck = init_luck,a_fun = rate_exp,dist = "exp", dt = 0.005)

#at event in which rate changes (at time 10) we need to do this:
a <- qtimecov(luck = init_luck,a_fun = rate_exp,dist = "exp", dt = 0.005,
              max_time = time_change, return_luck = TRUE)
new_luck <- a$luck
qtimecov(luck = new_luck,a_fun = rate_exp2,dist = "exp", dt = 0.005, start_time=a$tte)

#at second event in which rate changes again (at time 15) we need to do this:
a <- qtimecov(luck = new_luck,a_fun = rate_exp2,dist = "exp", dt = 0.005,
              max_time = time_change2, return_luck = TRUE, start_time=a$tte)
new_luck <- a$luck
#final TTE is
qtimecov(luck = new_luck,a_fun = rate_exp3,dist = "exp", dt = 0.005, start_time=a$tte)

```

---

random_stream	<i>Creates an environment (similar to R6 class) of random uniform numbers to be drawn from</i>
---------------	--

---

## Description

Creates an environment (similar to R6 class) of random uniform numbers to be drawn from

## Usage

```
random_stream(stream_size = 100)
```

## Arguments

stream\_size      Length of the vector of random uniform values to initialize

## Details

This function creates an environment object that behaves similar to an R6 class but offers more speed vs. an R6 class.

The object is always initialized (see example below) to a specific vector of random uniform values. The user can then call the object with `obj$draw_number(n)`, where `n` is an integer, and will return the first `n` elements of the created vector of uniform values. It will automatically remove those indexes from the vector, so the next time the user calls `obj$draw_n()` it will already consider the next index.

The user can also access the latest elements drawn by accessing `obj$random_n` (useful for when performing a luck adjustment), the current stream still to be drawn using `obj$stream` and the original size (when created) using `obj$stream_size`.

If performing luck adjustment, the user can always modify the random value by using `obj$random_n <- luck_adj(...)` (only valid if used with the expression approach, not with `modify_item`)

### Value

Self (environment) behaving similar to R6 class

### Examples

```
stream_1 <- random_stream(1000)
number_1 <- stream_1$draw_n() #extract 1st index from the vector created
identical(number_1,stream_1$random_n) #same value
number_2 <- stream_1$draw_n() #gets 1st index (considers previous)
identical(number_2,stream_1$random_n) #same value
```

---

rbeta\_mse

*Draw from a beta distribution based on mean and se*

---

### Description

Draw from a beta distribution based on mean and se

### Usage

```
rbeta_mse(n = 1, mean_v, se, seed = NULL)
```

### Arguments

<code>n</code>	Number of draws (must be $\geq 1$ )
<code>mean_v</code>	A vector of the mean values
<code>se</code>	A vector of the standard errors of the means
<code>seed</code>	An integer which will be used to set the seed for this draw.

### Value

A single estimate from the beta distribution based on given parameters

### Examples

```
rbeta_mse(n=1,mean_v=0.8,se=0.2)
```

---

rcond_gompertz	<i>Draw from a conditional Gompertz distribution (lower bound only)</i>
----------------	---

---

**Description**

Draw from a conditional Gompertz distribution (lower bound only)

**Usage**

```
rcond_gompertz(n = 1, shape, rate, lower_bound = 0, seed = NULL)
```

**Arguments**

n	The number of observations to be drawn
shape	The shape parameter of the Gompertz distribution, defined as in the coef() output on a flexsurvreg object
rate	The rate parameter of the Gompertz distribution, defined as in the coef() output on a flexsurvreg object
lower_bound	The lower bound of the conditional distribution
seed	An integer which will be used to set the seed for this draw.

**Value**

Estimate(s) from the conditional Gompertz distribution based on given parameters

**Examples**

```
rcond_gompertz(1, shape=0.05, rate=0.01, lower_bound = 50)
```

---

rcond_gompertz_lu	<i>Draw from a Conditional Gompertz distribution (lower and upper bound)</i>
-------------------	--

---

**Description**

Draw from a Conditional Gompertz distribution (lower and upper bound)

**Usage**

```
rcond_gompertz_lu(
  n,
  shape,
  rate,
  lower_bound = 0,
  upper_bound = Inf,
  seed = NULL
)
```

**Arguments**

n	The number of observations to be drawn
shape	The shape parameter of the Gompertz distribution, defined as in the coef() output on a flexsurvreg object
rate	The rate parameter of the Gompertz distribution, defined as in the coef() output on a flexsurvreg object
lower_bound	The lower bound of the conditional distribution
upper_bound	The upper bound of the conditional distribution
seed	An integer which will be used to set the seed for this draw.

**Value**

Estimate(s) from the Conditional Gompertz distribution based on given parameters

**Examples**

```
rcond_gompertz_lu(1,shape=0.05,rate=0.01,lower_bound = 50)
```

---

rdirichlet	<i>Draw from a dirichlet distribution based on number of counts in transition. Adapted from brms::rdirichlet</i>
------------	--

---

**Description**

Draw from a dirichlet distribution based on number of counts in transition. Adapted from brms::rdirichlet

**Usage**

```
rdirichlet(n = 1, alpha, seed = NULL)
```

**Arguments**

n	Number of draws (must be $\geq 1$ ). If $n > 1$ , it will return a list of matrices.
alpha	A matrix of alphas (transition counts)
seed	An integer which will be used to set the seed for this draw.

**Value**

A transition matrix. If  $n > 1$ , it will return a list of matrices.

**Examples**

```
rdirichlet(n=1,alpha= matrix(c(1251, 0, 350, 731),2,2))
rdirichlet(n=2,alpha= matrix(c(1251, 0, 350, 731),2,2))
```

---

rdirichlet_prob	<i>Draw from a dirichlet distribution based on mean transition probabilities and standard errors</i>
-----------------	--

---

**Description**

Draw from a dirichlet distribution based on mean transition probabilities and standard errors

**Usage**

```
rdirichlet_prob(n = 1, alpha, se, seed = NULL)
```

**Arguments**

n	Number of draws (must be $\geq 1$ ). If $n > 1$ , it will return a list of matrices.
alpha	A matrix of transition probabilities
se	A matrix of standard errors
seed	An integer which will be used to set the seed for this draw.

**Value**

A transition matrix. If  $n > 1$ , it will return a list of matrices.

**Examples**

```
rdirichlet_prob(n=1,alpha= matrix(c(0.7,0.3,0,0.1,0.7,0.2,0.1,0.2,0.7),3,3),
se=matrix(c(0.7,0.3,0,0.1,0.7,0.2,0.1,0.2,0.7)/10,3,3))
```

```
rdirichlet_prob(n=2,alpha= matrix(c(0.7,0.3,0,0.1,0.7,0.2,0.1,0.2,0.7),3,3),
se=matrix(c(0.7,0.3,0,0.1,0.7,0.2,0.1,0.2,0.7)/10,3,3))
```

---

replicate_profiles	<i>Replicate profiles data.frame</i>
--------------------	--------------------------------------

---

**Description**

Replicate profiles data.frame

**Usage**

```
replicate_profiles(
  profiles,
  replications,
  probabilities = NULL,
  replacement = TRUE,
  seed_used = NULL
)
```

**Arguments**

profiles	data.frame of profiles
replications	integer, final number of observations
probabilities	vector of probabilities with the same length as the number of rows of profiles. Does not need to add up to 1 (are reweighted)
replacement	Boolean whether replacement is used
seed_used	Integer with the seed to be used for consistent results

**Value**

Resampled data.frame of profiles

**Examples**

```
replicate_profiles(profiles=data.frame(id=1:100,age=rnorm(100,60,5)),
  replications=200,probabilities=rep(1,100))
```

---

 rgamma\_mse

*Draw from a gamma distribution based on mean and se*

---

**Description**

Draw from a gamma distribution based on mean and se

**Usage**

```
rgamma_mse(n = 1, mean_v, se, seed = NULL)
```

**Arguments**

n	Number of draws (must be >= 1)
mean_v	A vector of the mean values
se	A vector of the standard errors of the means
seed	An integer which will be used to set the seed for this draw.

**Value**

A single estimate from the gamma distribution based on given parameters

**Examples**

```
rgamma_mse(n=1,mean_v=0.8,se=0.2)
```

---

rpoisgamma	<i>Draw time to event (tte) from a Poisson or Poisson-Gamma (PG) Mixture/Negative Binomial (NB) Process</i>
------------	---

---

### Description

Draw time to event (tte) from a Poisson or Poisson-Gamma (PG) Mixture/Negative Binomial (NB) Process

### Usage

```
rpoisgamma(
  n,
  rate,
  theta = NULL,
  obs_time = 1,
  t_reps,
  seed = NULL,
  return_ind_rate = FALSE,
  return_df = FALSE
)
```

### Arguments

n	The number of observations to be drawn
rate	rate of the event (in terms of events per observation-time)
theta	Optional. When omitted, the function simulates times for a Poisson process. Represents the shape of the gamma mixture distribution. Estimated and reported as theta in negative binomial regression analyses in r.
obs_time	period over which events are observable
t_reps	Optional. Number of TBES to be generated to capture events within the observation window. When omitted, the function sets t_reps to the 99.99th quantile of the Poisson (if no theta is provided) or negative binomial (if theta is provided). Thus, the risk of missing possible events in the observation window is 0.01%.
seed	An integer which will be used to set the seed for this draw.
return_ind_rate	A boolean that indicates whether an additional vector with the rate parameters used per observation is used. It will alter the structure of the results to two lists, one storing tte with name tte, and the other with name ind_rate
return_df	A boolean that indicates whether a data.table object should be returned



## Details

Function to simulate event times from a Poisson or Poisson-Gamma (PG) Mixture/Negative Binomial (NB) Process. Event times are determined by sampling times between events (TBEs) from an exponential distribution, and cumulating these to derive the event times. Events occurring within the set observation time window are retained and returned. For times for a Poisson process, the provided rate is assumed constant. For a PG or NB, the individual rates are sampled from a Gamma distribution with shape = theta and scale = rate/theta.

## Value

Estimate(s) from the time to event based on poisson/Poisson-Gamma (PG) Mixture/Negative Binomial (NB) distribution based on given parameters

## Examples

```
rpoisgamma(1,rate=1,obs_time=1,theta=1)
```

---

run_sim	<i>Run the simulation</i>
---------	---------------------------

---

## Description

Run the simulation

## Usage

```
run_sim(  
  arm_list = c("int", "noint"),  
  sensitivity_inputs = NULL,  
  common_all_inputs = NULL,  
  common_pt_inputs = NULL,  
  unique_pt_inputs = NULL,  
  init_event_list = NULL,  
  evt_react_list = evt_react_list,  
  util_ongoing_list = NULL,  
  util_instant_list = NULL,  
  util_cycle_list = NULL,  
  cost_ongoing_list = NULL,  
  cost_instant_list = NULL,  
  cost_cycle_list = NULL,  
  other_ongoing_list = NULL,  
  other_instant_list = NULL,  
  npats = 500,  
  n_sim = 1,  
  psa_bool = NULL,  
  sensitivity_bool = FALSE,  
  sensitivity_names = NULL,
```

```

n_sensitivity = 1,
input_out = character(),
ipd = 1,
timed_freq = NULL,
debug = FALSE,
accum_backwards = FALSE,
continue_on_error = FALSE,
seed = NULL
)

```

### Arguments

**arm\_list** A vector of the names of the interventions evaluated in the simulation

**sensitivity\_inputs** A list of sensitivity inputs that do not change within a sensitivity in a similar fashion to **common\_all\_inputs**, etc

**common\_all\_inputs** A list of inputs common across patients that do not change within a simulation

**common\_pt\_inputs** A list of inputs that change across patients but are not affected by the intervention

**unique\_pt\_inputs** A list of inputs that change across each intervention

**init\_event\_list** A list of initial events and event times. If no initial events are given, a "Start" event at time 0 is created automatically

**evt\_react\_list** A list of event reactions

**util\_ongoing\_list** Vector of QALY named variables that are accrued at an ongoing basis (discounted using **drq**)

**util\_instant\_list** Vector of QALY named variables that are accrued instantaneously at an event (discounted using **drq**)

**util\_cycle\_list** Vector of QALY named variables that are accrued in cycles (discounted using **drq**)

**cost\_ongoing\_list** Vector of cost named variables that are accrued at an ongoing basis (discounted using **drc**)

**cost\_instant\_list** Vector of cost named variables that are accrued instantaneously at an event (discounted using **drc**)

**cost\_cycle\_list** Vector of cost named variables that are accrued in cycles (discounted using **drc**)

**other\_ongoing\_list** Vector of other named variables that are accrued at an ongoing basis (discounted using **drq**)

other_instant_list	Vector of other named variables that are accrued instantaneously at an event (discounted using drq)
npats	The number of patients to be simulated (it will simulate npats * length(arm_list))
n_sim	The number of simulations to run per sensitivity
psa_bool	A boolean to determine if PSA should be conducted. If n_sim > 1 and psa_bool = FALSE, the differences between simulations will be due to sampling
sensitivity_bool	A boolean to determine if Scenarios/DSA should be conducted.
sensitivity_names	A vector of scenario/DSA names that can be used to select the right sensitivity (e.g., c("Scenario_1", "Scenario_2")). The parameter "sens_name_used" is created from it which corresponds to the one being used for each iteration.
n_sensitivity	Number of sensitivity analysis (DSA or Scenarios) to run. It will be interacted with sensitivity_names argument if not null (n_sensitivity * length(sensitivity_names)). For DSA, it should be as many parameters as there are. For scenario, it should be 1.
input_out	A vector of variables to be returned in the output data frame
ipd	Integer taking value 1 for full IPD data returned, and 2 IPD data but aggregating events (returning last value for numeric/character/factor variables. For other objects (e.g., matrices), the IPD will still be returned as the aggregation rule is not clear). Other values mean no IPD data returned (removes non-numerical or length>1 items)
timed_freq	If NULL, it does not produce any timed outputs. Otherwise should be a number (e.g., every 1 year)
debug	If TRUE, will generate a log file
accum_backwards	If TRUE, the ongoing accumulators will count backwards (i.e., the current value is applied until the previous update). If FALSE, the current value is applied between the current event and the next time it is updated. If TRUE, user must use modify_item and modify_item_seq or results will be incorrect.
continue_on_error	If TRUE, on error it will attempt to continue by skipping the current simulation
seed	Starting seed to be used for the whole analysis. If null, it's set to 1 by default.

## Details

This function is slightly different from `run_sim_parallel`. `run_sim_parallel` only runs multiple-core at the simulation level. `run_sim` uses only-single core. `run_sim` can be more efficient if using only one simulation (e.g., deterministic), while `run_sim_parallel` will be more efficient if the number of simulations is >1 (e.g., PSA).

Event ties are processed in the order declared within the `init_event_list` argument (evts argument within the first sublist of that object). To do so, the program automatically adds a sequence from 0 to the (number of events - 1) times 1e-10 to add to the event times when selecting the event with minimum time. This time has been selected as it's relatively small yet not so small as to be ignored by `which.min` (see `.Machine` for more details)

A list of protected objects that should not be used by the user as input names or in the global environment to avoid the risk of overwriting them is as follows: c("arm", "arm\_list", "categories\_for\_export", "cur\_evtlist", "curtime", "evt", "i", "prevtime", "sens", "simulation", "sens\_name\_used", "list\_env", "uc\_lists", "npats", "ipd").

The engine uses the L'Ecuyer-CMRG for the random number generator. Note that the random seeds are set to be unique in their category (i.e., at patient level, patient-arm level, etc.)

If no `drc` or `drq` parameters are passed within `sensitivity` or `common_all` input lists, these are assigned a default value 0.03 for discounting costs, QALYs and others.

Ongoing items will look backward to the last time updated when performing the discounting and accumulation. This means that the user does not necessarily need to keep updating the value, but only add it when the value changes looking forward (e.g., `o_q` = utility at event 1, at event 2 utility does not change, but at event 3 it does, so we want to make sure to add `o_q` = utility at event 3 before updating utility. The program will automatically look back until event 1). Note that in previous versions of the package backward was the default, and now this has switched to forward.

If using `accum_backwards = TRUE`, then it is mandatory for the user to use `modify_item` and `modify_item_seq` in event reactions, as the standard assignment approach (e.g., `a <- 5`) will not calculate the right results, particularly in the presence of conditional statements.

It is important to note that the QALYs and Costs (ongoing or instant or per cycle) used should be of length 1. If they were of length  $> 1$ , the model would expand the data, so instead of having each event as a row, the event would have  $N$  rows (equal to the length of the costs/qalys to discount passed). This means more processing of the results data would be needed in order for it to provide the correct results.

If the cycle lists are used, then it is expected the user will declare as well the name of the variable pasted with `cycle_l` and `cycle_starttime` (e.g., `c_default_cycle_l` and `c_default_cycle_starttime`) to ensure the discounting can be computed using cycles, with `cycle_l` being the cycle length, and `cycle_starttime` being the starting time in which the variable started counting. Optionally, `max_cycles` must also be added (if no maximum number of cycles, it should be set equal to NA).

`debug = TRUE` will export a log file with the timestamp up the error in the main working directory. Note that using this mode without `modify_item` or `modify_item_seq` may lead to inaccuracies if assignments are done in non-standard ways, as the AST may not catch all the relevant assignments (e.g., an assignment like `assign(paste("x_",i),5)` in a loop will not be identified, unless using `modify_item(_seq)`).

`continue_on_error` will skip the current simulation (so it won't continue for the rest of patient-arms) if TRUE. Note that this will make the progress bar not correct, as a set of patients that were expected to be run is not.

## Value

A list of data frames with the simulation results

## Examples

```
library(magrittr)
common_all_inputs <- add_item(
  util.sick = 0.8,
  util.sicker = 0.5,
  cost.sick = 3000,
  cost.sicker = 7000,
```

```

cost.int = 1000,
coef_noint = log(0.2),
HR_int = 0.8,
drc = 0.035, #different values than what's assumed by default
drq = 0.035,
random_seed_sicker_i = sample.int(100000,5,replace = FALSE)
)

common_pt_inputs <- add_item(death= max(0.0000001,rnorm(n=1, mean=12, sd=3)))

unique_pt_inputs <- add_item(fl.sick = 1,
                             q_default = util.sick,
                             c_default = cost.sick + if(arm=="int"){cost.int}else{0})

init_event_list <-
add_tte(arm=c("noint","int"), evts = c("sick","sicker","death") ,input={
  sick <- 0
  sicker <- draw_tte(1,dist="exp",
                    coef1=coef_noint, beta_tx = ifelse(arm=="int",HR_int,1),
                    seed = random_seed_sicker_i[i])
})

evt_react_list <-
add_reactevt(name_evt = "sick",
             input = {}) %>%
add_reactevt(name_evt = "sicker",
             input = {
               modify_item(list(q_default = util.sicker,
                                c_default = cost.sicker + if(arm=="int"){cost.int}else{0},
                                fl.sick = 0))
             }) %>%
add_reactevt(name_evt = "death",
             input = {
               modify_item(list(q_default = 0,
                                c_default = 0,
                                curtime = Inf))
             })

util_ongoing <- "q_default"
cost_ongoing <- "c_default"

run_sim(arm_list=c("int","noint"),
        common_all_inputs = common_all_inputs,
        common_pt_inputs = common_pt_inputs,
        unique_pt_inputs = unique_pt_inputs,
        init_event_list = init_event_list,
        evt_react_list = evt_react_list,
        util_ongoing_list = util_ongoing,
        cost_ongoing_list = cost_ongoing,
        npats = 2,
        n_sim = 1,

```

```
psa_bool = FALSE,  
ipd = 1)
```

---

run_sim_parallel	<i>Run simulations in parallel mode (at the simulation level)</i>
------------------	---

---

### Description

Run simulations in parallel mode (at the simulation level)

### Usage

```
run_sim_parallel(  
  arm_list = c("int", "noint"),  
  sensitivity_inputs = NULL,  
  common_all_inputs = NULL,  
  common_pt_inputs = NULL,  
  unique_pt_inputs = NULL,  
  init_event_list = NULL,  
  evt_react_list = evt_react_list,  
  util_ongoing_list = NULL,  
  util_instant_list = NULL,  
  util_cycle_list = NULL,  
  cost_ongoing_list = NULL,  
  cost_instant_list = NULL,  
  cost_cycle_list = NULL,  
  other_ongoing_list = NULL,  
  other_instant_list = NULL,  
  npats = 500,  
  n_sim = 1,  
  psa_bool = NULL,  
  sensitivity_bool = FALSE,  
  sensitivity_names = NULL,  
  n_sensitivity = 1,  
  ncores = 1,  
  input_out = character(),  
  ipd = 1,  
  timed_freq = NULL,  
  debug = FALSE,  
  accum_backwards = FALSE,  
  continue_on_error = FALSE,  
  seed = NULL  
)
```

**Arguments**

arm_list	A vector of the names of the interventions evaluated in the simulation
sensitivity_inputs	A list of sensitivity inputs that do not change within a sensitivity in a similar fashion to common_all_inputs, etc
common_all_inputs	A list of inputs common across patients that do not change within a simulation
common_pt_inputs	A list of inputs that change across patients but are not affected by the intervention
unique_pt_inputs	A list of inputs that change across each intervention
init_event_list	A list of initial events and event times. If no initial events are given, a "Start" event at time 0 is created automatically
evt_react_list	A list of event reactions
util_ongoing_list	Vector of QALY named variables that are accrued at an ongoing basis (discounted using drq)
util_instant_list	Vector of QALY named variables that are accrued instantaneously at an event (discounted using drq)
util_cycle_list	Vector of QALY named variables that are accrued in cycles (discounted using drq)
cost_ongoing_list	Vector of cost named variables that are accrued at an ongoing basis (discounted using drc)
cost_instant_list	Vector of cost named variables that are accrued instantaneously at an event (discounted using drc)
cost_cycle_list	Vector of cost named variables that are accrued in cycles (discounted using drc)
other_ongoing_list	Vector of other named variables that are accrued at an ongoing basis (discounted using drq)
other_instant_list	Vector of other named variables that are accrued instantaneously at an event (discounted using drq)
npats	The number of patients to be simulated (it will simulate npats * length(arm_list))
n_sim	The number of simulations to run per sensitivity
psa_bool	A boolean to determine if PSA should be conducted. If n_sim > 1 and psa_bool = FALSE, the differences between simulations will be due to sampling
sensitivity_bool	A boolean to determine if Scenarios/DSA should be conducted.

sensitivity_names	A vector of scenario/DSA names that can be used to select the right sensitivity (e.g., c("Scenario_1", "Scenario_2")). The parameter "sens_name_used" is created from it which corresponds to the one being used for each iteration.
n_sensitivity	Number of sensitivity analysis (DSA or Scenarios) to run. It will be interacted with sensitivity_names argument if not null (n_sensitivity = n_sensitivity * length(sensitivity_names)). For DSA, it should be as many parameters as there are. For scenario, it should be 1.
ncores	The number of cores to use for parallel computing
input_out	A vector of variables to be returned in the output data frame
ipd	Integer taking value 0 if no IPD data returned, 1 for full IPD data returned, and 2 IPD data but aggregating events
timed_freq	If NULL, it does not produce any timed outputs. Otherwise should be a number (e.g., every 1 year)
debug	If TRUE, will generate a log file
accum_backwards	If TRUE, the ongoing accumulators will count backwards (i.e., the current value is applied until the previous update). If FALSE, the current value is applied between the current event and the next time it is updated. If TRUE, user must use modify_item and modify_item_seq or results will be incorrect.
continue_on_error	If TRUE, on error at patient stage will attempt to continue to the next simulation (only works if n_sim and/or n_sensitivity are > 1, not at the patient level)
seed	Starting seed to be used for the whole analysis. If null, it's set to 1 by default.

## Details

This function is slightly different from run\_sim. run\_sim allows to run single-core. run\_sim\_parallel allows to use multiple-core at the simulation level, making it more efficient for a large number of simulations relative to run\_sim (e.g., for PSA).

Event ties are processed in the order declared within the init\_event\_list argument (evts argument within the first sublist of that object). To do so, the program automatically adds a sequence from 0 to the (number of events - 1) times 1e-10 to add to the event times when selecting the event with minimum time. This time has been selected as it's relatively small yet not so small as to be ignored by which.min (see .Machine for more details)

A list of protected objects that should not be used by the user as input names or in the global environment to avoid the risk of overwriting them is as follows: c("arm", "arm\_list", "categories\_for\_export", "cur\_evtlist", "curtime", "evt", "i", "prevtime", "sens", "simulation", "sens\_name\_used", "list\_env", "uc\_lists", "npats", "ipd").

The engine uses the L'Ecuyer-CMRG for the random number generator. Note that if ncores > 1, then results per simulation will only be exactly replicable if using run\_sim\_parallel (as seeds are automatically transformed to be seven integer seeds -i.e, L'Ecuyer-CMRG seeds-) Note that the random seeds are set to be unique in their category (i.e., at patient level, patient-arm level, etc.)

If no drc or drq parameters are passed within sensitivity or common\_all input lists, these are assigned a default value 0.03 for discounting costs, QALYs and others.



Ongoing items will look backward to the last time updated when performing the discounting and accumulation. This means that the user does not necessarily need to keep updating the value, but only add it when the value changes looking forward (e.g., `o_q` = utility at event 1, at event 2 utility does not change, but at event 3 it does, so we want to make sure to add `o_q` = utility at event 3 before updating utility. The program will automatically look back until event 1). Note that in previous versions of the package backward was the default, and now this has switched to forward.

If using `accum_backwards = TRUE`, then it is mandatory for the user to use `modify_item` and `modify_item_seq` in event reactions, as the standard assignment approach (e.g., `a <- 5`) will not calculate the right results, particularly in the presence of conditional statements.

If the cycle lists are used, then it is expected the user will declare as well the name of the variable pasted with `cycle_l` and `cycle_starttime` (e.g., `c_default_cycle_l` and `c_default_cycle_starttime`) to ensure the discounting can be computed using cycles, with `cycle_l` being the cycle length, and `cycle_starttime` being the starting time in which the variable started counting. Optionally, `max_cycles` must also be added (if no maximum number of cycles, it should be set equal to NA).

`debug = TRUE` will export a log file with the timestamp up the error in the main working directory. Note that using this mode without `modify_item` or `modify_item_seq` may lead to inaccuracies if assignments are done in non-standard ways, as the AST may not catch all the relevant assignments (e.g., an assignment like `assign(paste("x_",i),5)` in a loop will not be identified, unless using `modify_item()`).

If `continue_on_error` is set to `FALSE`, it will only export analysis level inputs due to the parallel engine (use single-engine for those inputs) `continue_on_error` will skip the current simulation (so it won't continue for the rest of patient-arms) if `TRUE`. Note that this will make the progress bar not correct, as a set of patients that were expected to be run is not.

## Value

A list of lists with the analysis results

## Examples

```
library(magrittr)
common_all_inputs <- add_item(
  util.sick = 0.8,
  util.sicker = 0.5,
  cost.sick = 3000,
  cost.sicker = 7000,
  cost.int = 1000,
  coef_noint = log(0.2),
  HR_int = 0.8,
  drc = 0.035, #different values than what's assumed by default
  drq = 0.035,
  random_seed_sicker_i = sample.int(100000,5,replace = FALSE)
)

common_pt_inputs <- add_item(death= max(0.0000001,rnorm(n=1, mean=12, sd=3)))

unique_pt_inputs <- add_item(fl.sick = 1,
  q_default = util.sick,
  c_default = cost.sick + if(arm=="int"){cost.int}else{0})
```

```

init_event_list <-
add_tte(arm=c("noint","int"), evts = c("sick","sicker","death") ,input={
  sick <- 0
  sicker <- draw_tte(1,dist="exp",
  coef1=coef_noint, beta_tx = ifelse(arm=="int",HR_int,1),
  seed = random_seed_sicker_i[i])

})

evt_react_list <-
add_reactevt(name_evt = "sick",
  input = {}) %>%
add_reactevt(name_evt = "sicker",
  input = {
  modify_item(list(q_default = util.sicker,
  c_default = cost.sicker + if(arm=="int"){cost.int}else{0},
  fl.sick = 0))
  }) %>%
add_reactevt(name_evt = "death",
  input = {
  modify_item(list(q_default = 0,
  c_default = 0,
  curtime = Inf))
  })

util_ongoing <- "q_default"
cost_ongoing <- "c_default"

run_sim_parallel(arm_list=c("int","noint"),
common_all_inputs = common_all_inputs,
common_pt_inputs = common_pt_inputs,
unique_pt_inputs = unique_pt_inputs,
init_event_list = init_event_list,
evt_react_list = evt_react_list,
util_ongoing_list = util_ongoing,
cost_ongoing_list = cost_ongoing,
npats = 2,
n_sim = 1,
psa_bool = FALSE,
ipd = 1,
ncores = 1)

```

**Description**

Create an iterator based on sens of the current iteration within a scenario (DSA)

**Usage**

```
sens_iterator(sens, n_sensitivity)
```

**Arguments**

sens                    current analysis iterator  
n\_sensitivity        total number of analyses to be run

**Details**

In a situation like a DSA, where two (low and high) scenarios are run, sens will go from 1 to n\_sensitivity\*2. However, this is not ideal as the parameter selector may depend on knowing the parameter order (i.e., 1, 2, 3...), which means resetting the counter back to 1 once sens reaches n\_sensitivity (or any multiple of n\_sensitivity) is needed.

**Value**

Integer iterator based on the number of sensitivity analyses being run and the total iterator

**Examples**

```
sens_iterator(5,20)
sens_iterator(25,20)
```

---

summary\_results\_det     *Deterministic results for a specific treatment*

---

**Description**

Deterministic results for a specific treatment

**Usage**

```
summary_results_det(out = results[[1]][[1]], arm = NULL, wtp = 50000)
```

**Arguments**

out                    The final\_output data frame from the list object returned by run\_sim()  
arm                    The reference treatment for calculation of incremental outcomes  
wtp                    Willingness to pay to have INMB

**Value**

A dataframe with absolute costs, LYs, QALYs, and ICER and ICUR for each intervention

**Examples**

```
res <- list(list(list(sensitivity_name = "", arm_list = c("int", "noint")
), total_lys = c(int = 9.04687362556945, noint = 9.04687362556945
), total_qalys = c(int = 6.20743830697466, noint = 6.18115138126336
), total_costs = c(int = 49921.6357486899, noint = 41225.2544659378
), total_lys_undisc = c(int = 10.8986618377039, noint = 10.8986618377039
), total_qalys_undisc = c(int = 7.50117621700097, noint = 7.47414569286751
), total_costs_undisc = c(int = 59831.3573929783, noint = 49293.1025437205
), c_default = c(int = 49921.6357486899, noint = 41225.2544659378
), c_default_undisc = c(int = 59831.3573929783, noint = 49293.1025437205
), q_default = c(int = 6.20743830697466, noint = 6.18115138126336
), q_default_undisc = c(int = 7.50117621700097, noint = 7.47414569286751
), merged_df = list(simulation = 1L, sensitivity = 1L)))
```

```
summary_results_det(res[[1]][[1]], arm="int")
```

---

summary\_results\_sens *Summary of sensitivity outputs for a treatment*

---

**Description**

Summary of sensitivity outputs for a treatment

**Usage**

```
summary_results_sens(out = results, arm = NULL, wtp = 50000)
```

**Arguments**

out	The list object returned by run_sim()
arm	The reference treatment for calculation of incremental outcomes
wtp	Willingness to pay to have INMB

**Value**

A data frame with each sensitivity output per arm

**Examples**

```
res <- list(list(list(sensitivity_name = "", arm_list = c("int", "noint")
), total_lys = c(int = 9.04687362556945, noint = 9.04687362556945
), total_qalys = c(int = 6.20743830697466, noint = 6.18115138126336
), total_costs = c(int = 49921.6357486899, noint = 41225.2544659378
), total_lys_undisc = c(int = 10.8986618377039, noint = 10.8986618377039
), total_qalys_undisc = c(int = 7.50117621700097, noint = 7.47414569286751
), total_costs_undisc = c(int = 59831.3573929783, noint = 49293.1025437205
```

```

), c_default = c(int = 49921.6357486899, noint = 41225.2544659378
), c_default_undisc = c(int = 59831.3573929783, noint = 49293.1025437205
), q_default = c(int = 6.20743830697466, noint = 6.18115138126336
), q_default_undisc = c(int = 7.50117621700097, noint = 7.47414569286751
), merged_df = list(simulation = 1L, sensitivity = 1L)))

```

```
summary_results_sens(res,arm="int")
```

---

summary\_results\_sim    *Summary of PSA outputs for a treatment*

---

## Description

Summary of PSA outputs for a treatment

## Usage

```
summary_results_sim(out = results[[1]], arm = NULL, wtp = 50000)
```

## Arguments

out	The output_sim data frame from the list object returned by run_sim()
arm	The reference treatment for calculation of incremental outcomes
wtp	Willingness to pay to have INMB

## Value

A data frame with mean and 95% CI of absolute costs, LYs, QALYs, ICER and ICUR for each intervention from the PSA samples

## Examples

```

res <- list(list(list(sensitivity_name = "", arm_list = c("int", "noint"
), total_lys = c(int = 9.04687362556945, noint = 9.04687362556945
), total_qalys = c(int = 6.20743830697466, noint = 6.18115138126336
), total_costs = c(int = 49921.6357486899, noint = 41225.2544659378
), total_lys_undisc = c(int = 10.8986618377039, noint = 10.8986618377039
), total_qalys_undisc = c(int = 7.50117621700097, noint = 7.47414569286751
), total_costs_undisc = c(int = 59831.3573929783, noint = 49293.1025437205
), c_default = c(int = 49921.6357486899, noint = 41225.2544659378
), c_default_undisc = c(int = 59831.3573929783, noint = 49293.1025437205
), q_default = c(int = 6.20743830697466, noint = 6.18115138126336
), q_default_undisc = c(int = 7.50117621700097, noint = 7.47414569286751
), merged_df = list(simulation = 1L, sensitivity = 1L)))

```

```
summary_results_sim(res[[1]],arm="int")
```

---

`tte.df`*Example TTE IPD data*

---

**Description**

An example of TTE IPD data for the `example_ipd` file

**Usage**`tte.df`**Format**`tte.df`:

A data frame with 1000 rows and 8 columns:

**USUBJID** Patient ID

**ARMCD, ARM** Arm code and variables

**PARAMCD, PARAM** Parameter

**AVAL, AVALCD** Values of interest

**CNSR** Censored observation?

**Source**

Simulated through FlexsurvPlus package using `sim_adtte(seed = 821, rho = 0, beta_1a = log(0.6), beta_1b = log(0.6), beta_pd = log(0.2))`

# Index

## \* datasets

tte.df, 62

add\_item, 3  
add\_item2, 4  
add\_reactevt, 5  
add\_tte, 6  
adj\_val, 7  
ast\_as\_list, 8

ceac\_des, 10  
cond\_dirichlet, 11  
cond\_mvn, 11  
create\_indicators, 12

disc\_cycle, 13  
disc\_cycle\_v, 14  
disc\_instant, 15  
disc\_instant\_v, 16  
disc\_ongoing, 16  
disc\_ongoing\_v, 17  
draw\_tte, 18

evpi\_des, 19  
extract\_elements\_from\_list, 20  
extract\_from\_reactions, 21  
extract\_psa\_result, 22

luck\_adj, 23

modify\_event, 24  
modify\_item, 25  
modify\_item\_seq, 26

new\_event, 27

pcond\_gompertz, 28  
pick\_psa, 29  
pick\_val\_v, 30

qbeta\_mse, 32

qcond\_exp, 33  
qcond\_gamma, 33  
qcond\_gompertz, 34  
qcond\_llogis, 35  
qcond\_lnorm, 35  
qcond\_norm, 36  
qcond\_weibull, 37  
qcond\_weibullPH, 37  
qgamma\_mse, 38  
qtimecov, 39

random\_stream, 42  
rbeta\_mse, 43  
rcond\_gompertz, 44  
rcond\_gompertz\_lu, 44  
rdirichlet, 45  
rdirichlet\_prob, 46  
replicate\_profiles, 46  
rgamma\_mse, 47  
rpoisgamma, 48  
run\_sim, 49  
run\_sim\_parallel, 54

sens\_iterator, 58  
summary\_results\_det, 59  
summary\_results\_sens, 60  
summary\_results\_sim, 61

tte.df, 62