Package ‘IBMPopSim’

October 15, 2024

Type Package

Title Individual Based Model Population Simulation

Version 1.1.0

Date 2024-10-10

Maintainer Daphné Giorgi <daphne.giorgi@sorbonne-universite.fr>

Description

Simulation of the random evolution of heterogeneous populations using stochastic Individual-Based Models (IBMs) <doi:10.48550/arXiv.2303.06183>.
The package enables users to simulate population evolution, in which individuals are characterized by their age and some characteristics, and the population is modified by different types of events, including births/arrivals, death/exit events, or changes of characteristics. The frequency at which an event can occur to an individual can depend on their age and characteristics, but also on the characteristics of other individuals (interactions).
Such models have a wide range of applications. For instance, IBMs can be used for simulating the evolution of a heterogeneous insurance portfolio with selection or for validating mortality forecasts.
This package overcomes the limitations of time-consuming IBMs simulations by implementing new efficient algorithms based on thinning methods, which are compiled using the 'Rcpp' package while providing a user-friendly interface.

URL https://github.com/DaphneGiorgi/IBMPopSim,
https://DaphneGiorgi.github.io/IBMPopSim/

BugReports https://github.com/DaphneGiorgi/IBMPopSim/issues

License MIT + file LICENSE

Depends R (>= 3.5.0)

Imports Rcpp (>= 0.12), checkmate, stats, readr, rlang, dplyr (>= 0.8.0), ggplot2

Suggests RcppArmadillo, knitr, rmarkdown, bookdown, ggfortify, magick, colorspace, gganimate, gridExtra

LinkingTo Rcpp

LazyData true

NeedsCompilation yes

VignetteBuilder knitr

RoxygenNote 7.3.2

Encoding UTF-8

Author Daphné Giorgi [aut, cre],
Sarah Kaakai [aut],
Vincent Lemaire [aut]

Repository CRAN

Date/Publication 2024-10-15 14:20:07 UTC

Contents

`IBMPopSim-package` *IBMPopSim: Individual Based Model Population Simulation*

Description

Simulation of the random evolution of heterogeneous populations using stochastic Individual-Based Models (IBMs) doi:10.48550/arXiv.2303.06183. The package enables users to simulate population evolution, in which individuals are characterized by their age and some characteristics, and the population is modified by different types of events, including births/arrivals, death/exit events, or changes of characteristics. The frequency at which an event can occur to an individual can depend on their age and characteristics, but also on the characteristics of other individuals (interactions). Such models have a wide range of applications. For instance, IBMs can be used for simulating the evolution of a heterogeneous insurance portfolio with selection or for validating mortality forecasts. This package overcomes the limitations of time-consuming IBMs simulations by implementing new efficient algorithms based on thinning methods, which are compiled using the 'Rcpp' package while providing a user-friendly interface.

Author(s)

Maintainer: Daphné Giorgi <`daphne.giorgi@sorbonne-universite.fr`>

Authors:

- Sarah Kaakai <`sarah.kaakai@univ-lemans.fr`>
- Vincent Lemaire <`vincent.lemaire@sorbonne-universite.fr`>

See Also

Useful links:

- https://github.com/DaphneGiorgi/IBMPopSim
- https://DaphneGiorgi.github.io/IBMPopSim/
- Report bugs at https://github.com/DaphneGiorgi/IBMPopSim/issues

---

`add_characteristic` *Generic method for add_characteristic*

Description

Generic method for add_characteristic

Usage

```
add_characteristic(x, name, value = NA)
```

Arguments

`x`	An object.
`name`	Name of the characteristic to add.
`value`	Value of the characteristic. By default `NA`.

`add_characteristic.population` *Add characteristic to a population*

Description

Add characteristic to a population

Usage

```
## S3 method for class 'population'
add_characteristic(x, name, value = NA)
```

Arguments

`x`	Object of population class representing a population.
`name`	Name of the characteristic to add.
`value`	Value of the characteristic. By default `NA`.

`age_pyramid` *Generic method for age_pyramid*

Description

Generic method for age_pyramid

Usage

```
age_pyramid(object, time = 0, ages = c(0:110, Inf), ...)
```

Arguments

`object`	Population.
`time`	The age pyramid is computed at instant `time`. Must be a numeric greater than or equal to 0.
`ages`	*(Optional)* A numeric vector of distinct positive values composing age groups. Must be in increasing order.
`...`	Additional parameters

Value

An object of class `pyramid` containing the age pyramid of a population at instant `time`.

`age_pyramid.population`
Age pyramid from a population at a given time.

Description

Reduce a population containing all individuals (with some characteristics) to an age-groups data frame (preserving characteristics). The function computes the number of individuals at `time` in each age group `[ages[i],ages[i+1][`, for `i` in `{1,...,N-1}`.

Usage

```
## S3 method for class 'population'
age_pyramid(object, time = 0, ages = c(0:110, Inf), ...)
```

Arguments

`object`	Object of `population` class representing a population.
`time`	The age pyramid is computed at instant `time`. Must be a numeric greater than or equal to 0.
`ages`	*(Optional)* A numeric vector of distinct positive values composing age groups. Must be in increasing order.
`...`	Additional parameters

Value

An object of class `pyramid` containing the age pyramid of the given population at instant `time`.

See Also

`age_pyramids.population`

Examples

```
age_pyramid(population(EW_pop_14$sample), time = 0)

age_pyramid(population(EW_popIMD_14$sample), time = 0, ages = seq(0, 120, by=2))
```

`age_pyramids` *Generic method for age_pyramids*

Description

Generic method for age_pyramids

Usage

```
age_pyramids(object, time = 0, ages = c(0:110, Inf))
```

Arguments

`object`	Population.
`time`	The age pyramid is computed at instant `time`. Must be a numeric greater than or equal to 0.
`ages`	*(Optional)* A numeric vector of distinct positive values composing age groups. Must be in increasing order.

`age_pyramids.population` *Age pyramid from a population data frame at some given times.*

Description

Vectorial version in time of the function `age_pyramid.population`. Not compatible with IBMs including swap events.

Usage

```
## S3 method for class 'population'
age_pyramids(object, time = 0, ages = c(0:110, Inf))
```

Arguments

`object`	Object of `population` class representing a population.
`time`	The age pyramid is computed at instant `time`. Must be a numeric greater than or equal to 0.
`ages`	*(Optional)* A numeric vector of distinct positive values composing age groups. Must be in increasing order.

Details

For convenience. This is a just a `lapply` call of `age_pyramid.population` on the vector `time`.

`check_intensity_code` *Check the intensity code.*

Description

Verifies that the intensity contains the string 'result'.

Usage

```
check_intensity_code(code)
```

Arguments

`code`	String containing the intensity code.

`check_interaction_code` *Check the interaction code.*

Description

Verifies that the interaction contains the string 'result'.

Usage

```
check_interaction_code(code)
```

Arguments

`code`	String containing the interaction code.

`check_kernel_code` *Check the kernel code.*

Description

Verifies the kernel code.

Usage

```
check_kernel_code(code)
```

Arguments

`code`	String containing the kernel code.

`compatibility_chars_events` *Check characteristics-events compatibility*

Description

A function to check the compatibility between characteristics and events

Usage

```
compatibility_chars_events(characteristics, events)
```

Arguments

`characteristics`	List of characteristics
`events`	List of events

`compatibility_pop_model`

Check population-model compatibility

Description

A function to check the compatibility between a population and a model

Usage

```
compatibility_pop_model(pop, model)
```

Arguments

`pop`	An object of class population
`model`	An Individual Based Model created with the mk_model function

---

`death_table`

Death table

Description

Creates a death table from a population object. For each `i=1..N-1` and `j=1..M`, the number of individuals with age at last birthday in `[ages[i],ages[i+1])` and died in `[times[j],times[j+1])` is computed.

Usage

```
death_table(pop, ages, period)
```

Arguments

`pop`	Object of class `population`.
`ages`	A vector of size `N` composed of age groups.
`period`	A vector of size `M` composed of time intervals.

Details

The function computes the number of death in each time interval `[times[j],times[j+1])`, `j=1..M`.

Value

A death table matrix.

Examples

```
dth_table <-  death_table(population(EW_pop_out), 0:101, 0:11)
```

`EWdata_hmd` *England and Wale mortality data (source: Human Mortality Database)*

Description

Obtained with

`EWdata_hmd <- hmd.mx(country = "GBRTENW", username = ... , password = ...,label = "England and Wales")`

Usage

```
EWdata_hmd
```

Format

An object of class `demogdata` of length 7.

`EW_popIMD_14` *England and Wales (EW) 2014 population and death rates by Index of Multiple Deprivation (IMD).*

Description

EW population, death rates by age, gender and IMD for year 2014 (Source: Office for National Statistics, reference number 006518).

Usage

```
EW_popIMD_14
```

Format

A list containing:

- `age_pyramid` Data frame containing EW age pyramid for year 2014, by gender, IMD and single year of age (0-115).

  Individuals in the age class 90+ are distributed in the single year of age classes as in the EW population.
- `death_rates` List containing 4 fields:
  - `male` Male death rates data frame, by IMD and single year of age (0-90+).
  - `female` Female death rates dataframe, by IMD and single year of age (0-90+).
- `sample` Population dataframe composed of 100 000 individuals, sampled from `age_pyramid`.

`EW_pop_14` *England and Wales (EW) 2014 population, death and birth rates.*

Description

EW 2014 population and death rates by age and gender (Source: Office for National Statistics, reference number 006518).

Female birth rates by age of the mother (Source: Office for National Statistics birth summary tables).

Usage

```
EW_pop_14
```

Format

A list containing:

- `age_pyramid` Data frame containing EW age pyramid for year 2014, by gender and single year of age (0-115).
- `rates` A list containing three data frames:
  - `birth` Birth rates data frame, by age of mother and 5 years age groups.
  - `death_male` Male death rates data frame, by single year of age (0-90+).
  - `death_female` Female death rates dataframe, by single year of age (0-90+).
- `sample` Population dataframe composed of 100 000 individuals, sampled from `age_pyramid`.

`EW_pop_out` *Example of "human population" after 100 years of simulation.*

Description

Example of "human population" data frame after 100 years of simulation, based on a sample of England and Wales 2014 population and demographic rates.

Usage

```
EW_pop_out
```

Format

Data frame containing a population structured by age and gender, simulated with an initial population of 100 000 individuals sampled from `EW_pop_14$age_pyramid` over 100 years, with birth and death events.

`exposure_table` *Exposure table*

Description

Returns the Central Exposure-to-Risk for given ages groups and time period. The central Exposure-to-risk is computed as the sum of the time spent by individuals in a given age group over a given period, where age is the age at last birthday.

Usage

```
exposure_table(pop, ages, period)
```

Arguments

`pop`	Object of class `population`.
`ages`	A vector of size `N` composed of age groups.
`period`	A vector of size `M` composed of time intervals.

Details

The function computes the central exposure-to-risk in each time interval `[t[j],t[j+1])`, `j=1..M`, and age groups.

Value

An exposure matrix

Examples

```
ex_table <- exposure_table(population(EW_pop_out),0:101,0:11)
```

`get_characteristics` *Generic method for get_characteristics*

Description

Generic method for get_characteristics

Usage

```
get_characteristics(object, ...)
```

Arguments

`object`	An object.
`...`	Additional parameters.

`get_characteristics.population`

Returns names and C types of the characteristics.

Description

Returns names and C types of the characteristics (other than birth and death) of the individuals in a population, from a population data frame.

Usage

```
## S3 method for class 'population'
get_characteristics(object, ...)
```

Arguments

`object`	Object of population class representing a population.
`...`	additional arguments.

Value

Named vector composed of characteristics names and C types. If the population has no characteristics, which means that it has only the `birth` and `death` columns, this returns `NULL`.

Examples

```
get_characteristics(population(EW_pop_14$sample))
```

`gompertz` *Gompertz–Makeham intensity function.*

Description

The intensity function (or hazard function) for the Gompertz-Makeham law of mortality distribution is defined as

$$h(x) = \alpha e^{\beta x} + \lambda$$

with $\alpha, \beta, \lambda \in R_+$.

Usage

```
gompertz(alpha, beta, lambda = 0)
```

Arguments

`alpha`	Non-negative real parameter.
`beta`	Non-negative real parameter.
`lambda`	Non-negative real parameter.

Details

A C++ version of this function is available. See `vignette('IBMPopSim_cpp')` for more details.

Value

Function which associates `x` to $\alpha exp(\beta x) + \lambda$.

See Also

https://en.wikipedia.org/wiki/Gompertz%E2%80%93Makeham_law_of_mortality

---

`linfun` *Linear interpolation function.*

Description

Return a function performing the linear interpolation.

Usage

```
linfun(x, y, yleft = y[1], yright = y[length(y)])
```

Arguments

`x, y`	Numeric vectors giving the coordinates of the points to be interpolated.
`yleft`	The value to be returned when input `x` values are less than `min(x)`.
`yright`	The value to be returned when input `x` values are greater than `max(x)`.

Details

A C++ version of this function is available. See `vignette('IBMPopSim_cpp')` for more details.

Value

Objet of class `linfun` and `function` which is an `approxfun` function with `method = 'linear'`.

`max.stepfun` *Returns the maximum of a function of class* `stepfun`.

Description

Returns the maximum of a function of class `stepfun`.

Usage

```
## S3 method for class 'stepfun'
max(..., na.rm = FALSE)
```

Arguments

`...`	argument of class `stepfun`
`na.rm`	a logical indicating whether missing values should be removed

Value

The maximum of the step function.

`merge_pop_withid` *A function returning a merged dataframe from a list of population dataframes with id.*

Description

A function returning a merged dataframe from a list of population dataframes with id.

Usage

```
merge_pop_withid(pop_df_list, chars_tracked = NULL)
```

Arguments

`pop_df_list`	A list of population dataframe where the first three columns of each dataframe are id, birth and death.
`chars_tracked`	A vector of characteristics to be tracked over time.

Value

A dataframe composed of all individuals with their characteristics at each simulation time.

`mk_event_individual` *Creating an event with intensity of class individual*

Description

Creates an event with intensity of class individual (without interactions). When the event occurs, something happens to an individual `I` in the population. The created event must be used with `mk_model`.

Usage

```
mk_event_individual(type, name, intensity_code, kernel_code = "")
```

Arguments

`type`	Must be one of `'birth'`, `'death'`, `'entry'`, `'exit'`, `'swap'` or `'custom'`. See details.
`name`	*(Optional)* If not specified, the name given to the event is its type.
`intensity_code`	String containing some C++ code describing the intensity function. See details.
`kernel_code`	String containing some C++ code describing the event action. Optional for `'birth'`, `'death'` and `'exit'` events. See details.

Details

The `type` argument is one of the following

- `'birth'` By default, a new individual `newI` is created, with the same characteristics of the parent `I` and birth date equal to the current time. Optional code can be precised in `kernel_code`.
- `'death'` By default, the individual `I` dies. Optional code can be precised in `kernel_code`.
- `'entry'` A new individual `newI` is added to the population, and its characteristics have to be defined by the user in the entry `kernel_code`.
- `'exit'` An individual `I` exits from the population. Optional code can be precised in `kernel_code`.
- `'swap'` The user can change the characteristics of the selected individual `I`. This requires `kernel_code`.
- `'custom'` None of the above types, the user defines `kernel_code` that can act on the selected individual `I` and on the population pop.

The `intensity_code` argument is a string containing some C++ code describing the event intensity for individual `I` at time `t`. The intensity value **must be stored** in the variable `result`. Some of available variables in the C++ code are: `t` (the current time), `I` (the current individual selected for the event), the name of the model parameters (some variables, or functions, see `mk_model`). See `vignette('IBMPopSim_Cpp')` for more details.

The `kernel_code` argument is a string containing some C++ code which describing the action of the event. Some of available variables in the C++ code are: `t` (the current time), pop (the current population), `I` (the current individual selected for the event), `newI` (the new individual if `'birth'` or `'entry'` event), the name of the model parameters (some variables, or functions, see `mk_model`). See `vignette('IBMPopSim')` for more details.

Value

An S3 object of class `event` of type individual.

See Also

`mk_model`, `mk_event_poisson`, `mk_event_inhomogeneous_poisson`, and `mk_event_interaction`.

Examples

```
params <- list("p_male"= 0.51,
              "birth_rate" = stepfun(c(15,40), c(0,0.05,0)),
              "death_rate" = gompertz(0.008, 0.02))

death_event <- mk_event_individual(type = "death",
                name = "my_death_event",
                intensity_code = "result = death_rate(age(I,t));")

birth_event <- mk_event_individual(type = "birth",
                intensity_code = "if (I.male) result = 0;
                                  else result = birth_rate(age(I,t));",
                kernel_code = "newI.male = CUnif(0, 1) < p_male;")
```

`mk_event_inhomogeneous_poisson`

Creating inhomogeneous Poisson class event

Description

The function `mk_event_inhomogeneous_poisson` is used to create an event with intensity type inhomogeneous Poisson (time dependent intensity which does not depend on population). When the event occurs, something happens in the population. The created event must be used with `mk_model`.

Usage

```
mk_event_inhomogeneous_poisson(type, name, intensity_code, kernel_code = "")
```

Arguments

`type`	Must be one of `'birth'`, `'death'`, `'entry'`, `'exit'`, `'swap'` or `'custom'`. See details.
`name`	*(Optional)* If not specified, the name given to the event is its type.
`intensity_code`	String containing some C++ code describing the intensity function. See details.
`kernel_code`	String containing some C++ code describing the event action. Optional for `'birth'`, `'death'` and `'exit'` events. See details.

Details

The `type` argument is one of the following

- `'birth'` By default, a new individual `newI` is created, with the same characteristics of the parent `I` and birth date equal to the current time. Optional code can be precised in `kernel_code`.
- `'death'` By default, the individual `I` dies. Optional code can be precised in `kernel_code`.
- `'entry'` A new individual `newI` is added to the population, and its characteristics have to be defined by the user in the entry `kernel_code`.
- `'exit'` An individual `I` exits from the population. Optional code can be precised in `kernel_code`.
- `'swap'` The user can change the characteristics of the selected individual `I`. This requires `kernel_code`.
- `'custom'` None of the above types, the user defines `kernel_code` that can act on the selected individual `I` and on the population `pop`.

The `intensity_code` argument is a string containing some C++ code describing the event intensity for individual `I` at time `t`. The intensity value **must be stored** in the variable `result`. Some of available variables in the C++ code are: `t` (the current time), `I` (the current individual selected for the event), the name of the model parameters (some variables, or functions, see `mk_model`). See `vignette('IBMPopSim_Cpp')` for more details.

The `kernel_code` argument is a string containing some C++ code which describing the action of the event. Some of available variables in the C++ code are: `t` (the current time), `pop` (the current population), `I` (the current individual selected for the event), `newI` (the new individual if `'birth'` or `'entry'` event), the name of the model parameters (some variables, or functions, see `mk_model`). See `vignette('IBMPopSim')` for more details.

Value

An S3 object of class `event` of type inhomogeneous Poisson.

See Also

`mk_model`, `mk_event_poisson`, `mk_event_individual`, `mk_event_interaction`.

`mk_event_interaction` *Creating an event with intensity of type interaction*

Description

Creates an event whose intensity depends on an individual and interactions with the population. When the event occurs, something happens to an individual `I` in the population. The intensity of the event can depend on time, the characteristics of I and other individuals in the population, and can be written as

$$d(I, t, pop) = \sum_{J \in pop} U(I, J, t),$$

where U is called the interaction function. The created event must be used with `mk_model`.

Usage

```
mk_event_interaction(
  type,
  name,
  interaction_code,
  kernel_code = "",
  interaction_type = "random"
)
```

Arguments

`type`	Must be one of `'birth'`, `'death'`, `'entry'`, `'exit'`, `'swap'` or `'custom'`. See details.
`name`	*(Optional)* If not specified, the name given to the event is its type.
`interaction_code`	String containing some C++ code describing the interaction function. See details.
`kernel_code`	String containing some C++ code describing the event action. Optional for `'birth'`, `'death'` and `'exit'` events. See details.
`interaction_type`	*(Optional)* Either `'random'` or `'full'`. By default `'random'` which is faster than `'full'`.

Details

The `type` argument is one of the following

- `'birth'` By default, a new individual `newI` is created, with the same characteristics of the parent `I` and birth date equal to the current time. Optional code can be precised in `kernel_code`.
- `'death'` By default, the individual `I` dies. Optional code can be precised in `kernel_code`.
- `'entry'` A new individual `newI` is added to the population, and its characteristics have to be defined by the user in the entry `kernel_code`.
- `'exit'` An individual `I` exits from the population. Optional code can be precised in `kernel_code`.
- `'swap'` The user can change the characteristics of the selected individual `I`. This requires `kernel_code`.
- `'custom'` None of the above types, the user defines `kernel_code` that can act on the selected individual `I` and on the population pop.

The `interaction_code` argument is a string containing some C++ code describing the event interaction function U at time `t`. The interaction value **must be stored** in the variable `result`. Some of available variables in the C++ code are: `t` (the current time), `I` (the current individual selected for the event), `J` (another individual if `interaction_type` is `'random'`), the name of the model parameters (some variables, or functions, see `mk_model`). See `vignette('IBMPopSim_Cpp')` for more details.

The `kernel_code` argument is a string containing some C++ code which describing the action of the event. Some of available variables in the C++ code are: `t` (the current time), pop (the current population), `I` (the current individual selected for the event), `newI` (the new individual if `'birth'` or `'entry'` event), the name of the model parameters (some variables, or functions, see `mk_model`). See `vignette('IBMPopSim')` for more details.

Value

An S3 object of class `event` of type interaction.

See Also

`mk_model`, `mk_event_poisson`, `mk_event_inhomogeneous_poisson`, `mk_event_individual`.

Examples

```
death_interaction_code<- " result = max(J.size -I.size,0);"
event <- mk_event_interaction(type="death",
                              interaction_code = death_interaction_code)
```

`mk_event_poisson` *Creating Poisson class event*

Description

The function `mk_event_poisson` is used to create an event with intensity of type Poisson (constant intensity which does not depend on population or time). When the event occurs, something happens in the population. The created event must be used with `mk_model`.

Usage

```
mk_event_poisson(type, name, intensity, kernel_code = "")
```

Arguments

`type`	Must be one of `'birth'`, `'death'`, `'entry'`, `'exit'`, `'swap'` or `'custom'`. See details.
`name`	*(Optional)* If not specified, the name given to the event is its type.
`intensity`	String containing some constant positive value, or name of a parameter which is a constant positive value.
`kernel_code`	String containing some C++ code describing the event action. Optional for `'birth'`, `'death'` and `'exit'` events. See details.

Details

The `type` argument is one of the following

- `'birth'` By default, a new individual `newI` is created, with the same characteristics of the parent `I` and birth date equal to the current time. Optional code can be precised in `kernel_code`.
- `'death'` By default, the individual `I` dies. Optional code can be precised in `kernel_code`.
- `'entry'` A new individual `newI` is added to the population, and its characteristics have to be defined by the user in the entry `kernel_code`.

`'exit'` An individual `I` exits from the population. Optional code can be precised in `kernel_code`.

`'swap'` The user can change the characteristics of the selected individual `I`. This requires `kernel_code`.

`'custom'` None of the above types, the user defines `kernel_code` that can act on the selected individual `I` and on the population `pop`.

The `kernel_code` argument is a string containing some C++ code which describing the action of the event. Some of available variables in the C++ code are: `t` (the current time), `pop` (the current population), `I` (the current individual selected for the event), `newI` (the new individual if `'birth'` or `'entry'` event), the name of the model parameters (some variables, or functions, see `mk_model`). See `vignette('IBMPopSim')` for more details.

Value

An S3 object of class `event` of type Poisson.

See Also

`mk_model`, `mk_event_inhomogeneous_poisson`, `mk_event_individual`, `mk_event_interaction`.

Examples

```
birth <- mk_event_poisson('birth', intensity = 10)


params <- list(beta = 10)
death <- mk_event_poisson('death', intensity = 'beta') # name of one parameter
mk_model(events = list(birth, death), parameters = params)
```

`mk_model` *Creates a model for IBMPopSim.*

Description

This function creates an Individual Based Model describing the population, events which can occur in the population, and the model parameters.

Usage

```
mk_model(
  characteristics = NULL,
  events,
  parameters = NULL,
  with_compilation = TRUE
)
```

Arguments

`characteristics`	List containing names and types of characteristics of individuals in the population. See `get_characteristics`.
`events`	List of events in the model. See `mk_event_poisson`, `mk_event_inhomogeneous_poisson`, `mk_event_individual`, and `mk_event_interaction`.
`parameters`	Model parameters. A list of parameters of the model.
`with_compilation`	*(Optional)* Logical parameter, `TRUE` by default. If `FALSE` the `sourceCpp` function is not called.

Details

It builds the C++ model code and produces the function `popsim_cpp` which will be used for simulating the model. The function used to simulate a population from a model is `popsim`.

Value

model List containing the built model :

- `individual_type`: Names and types (R and C++) of characteristics.
- `parameters_types`: Names and types (R and C++) of model parameters.
- `events`: List of events.
- `cpp_code`: Output of C++ compilation.

See Also

`popsim`, `mk_event_poisson`, `mk_event_individual`, `mk_event_interaction`.

Examples

```
params <- list("p_male"= 0.51,
              "birth_rate" = stepfun(c(15,40),c(0,0.05,0)),
              "death_rate" = gompertz(0.008,0.02))

death_event <- mk_event_individual(type = "death",
                                  intensity_code = "result = death_rate(age(I,t));")

birth_event <- mk_event_individual(type = 'birth',
                                  intensity_code = "if (I.male) result = 0;
                                    else result=birth_rate(age(I,t));",
                                  kernel_code = "newI.male = CUnif(0, 1) < p_male;")

model <- mk_model(characteristics = get_characteristics(population(EW_pop_14$sample)),
                 events = list(death_event,birth_event),
                 parameters = params)

summary(model)
```

`piecewise_x` *Piecewise real function.*

Description

Given the vectors `(breaks[1],...,breaks[n])` and the list of `IBMPopSim` compatible functions `funs = (f[0],f[1],...,f[n])` (one value more!), `piecewise_x(breaks, funs)` returns the function

$$f(x) = f_0(x)1_{x \leq breaks[1]} + \sum_{k=1}^{n-1} f_k(x)1_{[breaks_k, breaks_{k+1})}(x) + f_n(x)1_{x \geq breaks[n]}$$

Usage

```
piecewise_x(breaks, funs)
```

Arguments

`breaks`	Numeric vector giving the breaks of functions given in `funs`. Must be sorted with unique values.
`funs`	List of functions.

Details

A C++ version of this function is available. See `vignette('IBMPopSim_cpp')` for more details.

Value

Piecewise function built with the given intervals and functions.

Examples

```
dr <- with(EW_pop_14$rates,
           stepfun(x=death_male[,"age"], y=c(0,death_male[,"value"])))
# before age 80 the stepfun and after age 80 the gompertz function
f <- piecewise_x(80, list(dr, gompertz(0.00006, 0.085)))
x <- seq(40:120)
plot(x, sapply(x, f))
```

`piecewise_xy` *Piecewise real function of two variables.*

Description

Given the vectors `(breaks[1],...,breaks[n])` and the list of `IBMPopSim` compatible functions `funs = (f[0],f[1],...,f[n])` (one value more!), `piecewise_xy(breaks, funs)` returns the function

$$f(x,y) = f_0(x)1_{y\leq breaks[1]} + \sum_{k=1}^{n-1} f_k(x)1_{[breaks_k, breaks_{k+1})}(y) + f_n(x)1_{y\geq breaks[n]}$$

Usage

```
piecewise_xy(breaks, funs)
```

Arguments

`breaks`	Numeric vector giving the breaks of functions given in `funs`. Must be sorted with unique values.
`funs`	List of functions.

Details

A C++ version of this function is available. See `vignette('IBMPopSim_cpp')` for more details.

Value

Piecewise bivariate function built with the given intervals and functions.

Examples

```
time_dep_function <- piecewise_xy(c(5),
                                  list(gompertz(0.1, 0.005), gompertz(0.08, 0.005)))
time_dep_function(0, 65)  # death intensity at time 0 and age 65.
```

`plot.population` *Plot the age pyramid of a population data frame (at a given time).*

Description

Plot an age pyramid from age pyramid data frame with possibly several characteristics.

Usage

```
## S3 method for class 'population'
plot(
  x,
  group_colors = NULL,
  group_legend = "Group",
  age_breaks = NULL,
  value_breaks = NULL,
  ...
)
```

Arguments

`x`	Object of class population.
`group_colors`	*(Optional)* Named character vector.
`group_legend`	*(Optional)* Legend title name. By default set to `"Group"`.
`age_breaks`	*(Optional)* An ordered vector of indexes of vector `unique(pyr$age)` used for breaks for the axis of ages.
`value_breaks`	*(Optional)* Breaks for the axis of values.
`...`	Additional arguments

Value

Plot of age pyramid.

See Also

`plot.pyramid`, `age_pyramid.population`.

Examples

```
plot(population(EW_pop_14$sample), time = 0)
```

`plot.pyramid` *Plot an age pyramid.*

Description

Plot an age pyramid from age pyramid data frame with possibly several characteristics.

Usage

```
## S3 method for class 'pyramid'
plot(
  x,
  group_colors = NULL,
  group_legend = "Group",
  age_breaks = NULL,
  value_breaks = NULL,
  ...
)
```

Arguments

`x`	Object of class `pyramid`. *(Optional)* For plotting an age pyramid composed of several subgroups, the population data frame must contain a column named `group_name`.
`group_colors`	*(Optional)* Named character vector.
`group_legend`	*(Optional)* Legend title name. By default set to `"Group"`.
`age_breaks`	*(Optional)* An ordered vector of indexes of vector `unique(pyr$age)` used for breaks for the axis of ages.
`value_breaks`	*(Optional)* Breaks for the axis of values.
`...`	Additional parameters

Value

Plot of the age pyramid.

See Also

`plot.population`

Examples

```
plot.pyramid(subset(pyramid(EW_pop_14$age_pyramid), as.numeric(age) <= 110))


library(colorspace)
pyr_IMD <- subset(pyramid(EW_popIMD_14$age_pyramid), as.numeric(age) <= 110)
```

```
pyr_IMD$group_name <- with(pyr_IMD, ifelse(male, paste('Males - IMD', IMD),
                          paste('Females - IMD', IMD)))
colors <- c(sequential_hcl(n=5, palette = "Magenta"),
            sequential_hcl(n=5, palette = "Teal"))
names(colors) <- c(paste('Females - IMD', 1:5),
                   paste('Males - IMD', 1:5))
# note that you must have setequal(names(colors), pyr_IMD$group_name) is TRUE
plot.pyramid(pyr_IMD, colors)

# age pyramids at different times
library(gganimate)
pyrs = age_pyramids(population(EW_popIMD_14$sample), time = 1:10)
plot.pyramid(pyrs) + transition_time(time) + labs(title = "Time: {frame_time}")
```

`popsample` *Generic method for popsample*

Description

Generic method for popsample

Usage

```
popsample(age_pyramid, size, age_max = 120, time = 0)
```

Arguments

`age_pyramid`	Age pyramid.
`size`	A non-negative integer giving the number of individuals in population.
`age_max`	*(Optional)* A non-negative numeric which replace (if exists) the `Inf` in `age_pyramid.population`.
`time`	*(Optional)* The age pyramid is computed at instant `time`. Must be a numeric greater than or equal to 0.

Value

Object of `population` class representing a data frame of size `size` containing a population of individuals.

`popsample.pyramid` *Sample a population from an age pyramid (at a given time).*

Description

Sample a population from an age pyramid (at a given time).

Usage

```
## S3 method for class 'pyramid'
popsample(age_pyramid, size, age_max = 120, time = 0)
```

Arguments

`age_pyramid`	Object of `pyramid` class.
`size`	A non-negative integer giving the number of individuals in population.
`age_max`	*(Optional)* A non-negative numeric which replace (if exists) the `Inf` in `age_pyramid.population`.
`time`	*(Optional)* The age pyramid is computed at instant `time`. Must be a numeric greater than or equal to 0.

Value

Object of `population` class representing a data frame of size `size` containing a population of individuals.

Examples

```
pop_sample_1e4 <- popsample(pyramid(EW_pop_14$age_pyramid), size = 1e4)
```

`popsim` *Simulation of a model.*

Description

This function simulates the random evolution of an IBM.

Usage

```
popsim(
  model,
  initial_population,
  events_bounds,
  parameters = NULL,
  age_max = Inf,
  time,
  multithreading = FALSE,
  num_threads = NULL,
  clean_step = NULL,
  clean_ratio = 0.1,
  seed = NULL,
  verbose = FALSE
)
```

Arguments

`model`	Model resulting from a call to the function `mk_model`.
`initial_population`	Object of population class representing the initial population.
`events_bounds`	Named vector of events bounds, with names corresponding to events names.
`parameters`	List of model parameters.
`age_max`	Maximum age of individuals in the population (`Inf` by default).
`time`	Final time (Numeric). Can be of length 1 or a vector of simulation discretized times.
`multithreading`	Logical for multithread activation, `FALSE` by default. Should be only activated for IBM simulation with no interactions.
`num_threads`	*(Optional)* Number of threads used for multithreading. Set by default to the number of concurrent threads supported by the available hardware implementation.
`clean_step`	*(Optional)* Optional parameter for improving simulation time. Time step for removing dead (or exited) individuals from the population. By default, equal to age_max.
`clean_ratio`	*(Optional)* Optional parameter for improving simulation time. 0.1 by default.
`seed`	*(Optional)* Random generator seed, random by default.
`verbose`	*(Optional)* Activate verbose output, FALSE by default.

Value

List composed of

arguments Simulation inputs (initial population, parameters value, multithreading...)

logs Simulation logs (algorithm duration, accepted/rejected events...).

population If `time` is of length 1, population is an object of type population containing of all individuals who lived in the population in the time interval `[0,time]`. If `time` is a vector `(time[1], ..., time[n])`, `population` is a list of `n` objects of type population, each representing the state of the population at time `time[i]`, for `i = 1, ..., n`.

See Also

`mk_model`.

Examples

```
init_size <- 100000
pop_df <- data.frame(birth = rep(0, init_size), death = NA)
pop <- population(pop_df)

birth = mk_event_poisson(type = 'birth', intensity = 'lambda')
death = mk_event_poisson(type = 'death', intensity = 'mu')
params = list('lambda' = 100, 'mu' = 100)
birth_death <- mk_model(events = list(birth, death),
                        parameters = params)

sim_out <- popsim(model = birth_death,
                  initial_population = pop,
                  events_bounds = c('birth' = params$lambda, 'death' = params$mu),
                  parameters = params,
                  time = 10)
```

`population` *Class population*

Description

Data frame containing a population, with at least a birth and a death column, and eventually some other characteristics

Usage

```
population(x, entry = FALSE, out = FALSE, id = FALSE)
```

Arguments

`x`	Data frame or list of data frames, containing at least a birth and a death column
`entry`	Boolean flag. By default set to FALSE. If set to TRUE the population must contain a column of numerical values named "entry", If the column doesn't exist a column named "entry" is added to the data frame with all values set to NA.
`out`	Boolean flag. By default set to FALSE. If set to TRUE the population must contain a column of boolean values named "out", If the column doesn't exist a column named "out" is added to the data frame with all the values set to FALSE.

`id`	Boolean flag. By default set to FALSE. If set to TRUE the population must contain a column of integer distinct values named "id". If the column doesn't exist a column named "id" is added to the data frame with values seq(1, nrow(x)).

Value

Given data frame augmented of the "population" class. If a list of data frames is given, the column names should contain the string "id" and the list corresponds to the evolution of a population at different times. The constructor then returns the last population observed in the list (corresponding to the final state of the population).

`population_alive` *Generic method for population_alive*

Description

Generic method for population_alive

Usage

```
population_alive(object, t, a1 = 0, a2 = Inf, ...)
```

Arguments

`object`	A population.
`t`	A numeric indicating the time at which alive individuals are observed.
`a1`	0 by default. Lower bound for age.
`a2`	Inf by default. Upper bound for age.
`...`	Additional params.

Value

All individuals alive at time `t` and of age in `[a1,a2)`.

`population_alive.population` *Returns a population of individuals alive.*

Description

Returns a population of individuals alive.

Usage

```
## S3 method for class 'population'
population_alive(object, t, a1 = 0, a2 = Inf, ...)
```

Arguments

`object`	A population data frame containing at least a column `birth` and `death`.
`t`	A numeric indicating the time.
`a1`	0 by default. All individuals of age over `a1` at `t` are selected.
`a2`	Inf by default. All individuals of age below `a2` at `t` are selected.
`...`	Additional params.

Value

The function returns a population data frame containing all individuals alive at time `t` and of age in `[a1,a2)`.

`print.event` *Print Event*

Description

`print` method for class "event" giving a short description of an event.

Usage

```
## S3 method for class 'event'
print(x, ...)
```

Arguments

`x`	Argument of class `event`.
`...`	Additional arguments affecting the summary produced.

`print.model` *Printing of a model*

Description

`print` method for class `model`.

Usage

```
## S3 method for class 'model'
print(x, ...)
```

Arguments

`x`	argument of class `model`
`...`	additional arguments affecting the summary produced.

`print.population` *Printing population*

Description

Print a population

Usage

```
## S3 method for class 'population'
print(x, ...)
```

Arguments

`x`	Object of population class representing a population.
`...`	Additional arguments

Value

Print the population

`pyramid` *Class pyramid*

Description

Data frame containing an age pyramid, with at least an `age` and a `value` column, and eventually some other characteristics. If a `male` column is present, it must be a logical vector, if a `group` column is present, it must be a vector of type character.

Usage

```
pyramid(x)
```

Arguments

`x`	Data frame, containing at least an `age` and a `value` column

Value

Given data frame augmented of the "age_pyramid" class.

`stepfun` *Step Function.*

Description

Given the vectors `(x[1],...,x[n])` and `(y[0],y[1],...,y[n])` (one value more!), `stepfun(x, y)` returns an interpolating step function, say `f_n`. This is the cadlag version (`right = FALSE`) of the `stepfun` function from package `stats`. The step function value `f_n(t)` equals to the constant `y[k-1]` for `t` in `[x[k-1], x[k])` so that

$$f_n(t) = \sum_{k=1}^{n+1} y_{k-1} 1_{[x_{k-1},x_k)}(t),$$

with$x_0 = -\infty$ and $x_{n+1} = +\infty$.

Usage

```
stepfun(x, y)
```

Arguments

`x`	Numeric vector giving the knots or jump locations of the step function. Must be sorted with unique values.
`y`	Numeric vector one longer than x, giving the heights of the function values between the cx values.

Details

This function is defined for documentation purposes only. See `stepfun` and `approxfun`.

A C++ version of this function is available. See `vignette('IBMPopSim_cpp')` for more details.

Value

Objet of class `stepfun` with option `right = FALSE` (cadlag function).

See Also

`plot.stepfun` and `max.stepfun`.

`summary.event` *Summarizing an event*

Description

`summary` method for class `event` giving a detailed description of an event.

Usage

```
## S3 method for class 'event'
summary(object, ...)
```

Arguments

`object`	Argument of class `event`.
`...`	Additional arguments affecting the summary produced.

`summary.logs` *Summary logs*

Description

Summary of the logs of a simulation

Usage

```
## S3 method for class 'logs'
summary(object, ...)
```

Arguments

`object`	Logs of the output of a call to popsim function
`...`	Additional arguments affecting the summary produced

Value

Print column names and number of individuals

`summary.model` *Summary of a model*

Description

`summary` method for class `model`.

Usage

```
## S3 method for class 'model'
summary(object, ...)
```

Arguments

`object`	argument of class `model`
`...`	additional arguments affecting the summary produced.

`summary.population` *Summary population*

Description

Summary of a population with column names and number of individuals

Usage

```
## S3 method for class 'population'
summary(object, ...)
```

Arguments

`object`	Object of population class representing a population.
`...`	Additional arguments affecting the summary produced

Value

Print column names and number of individuals

`summary.simulation_output` *Summary simulation output*

Description

Summary of a simulation output

Usage

```
## S3 method for class 'simulation_output'
summary(object, ...)
```

Arguments

`object`	Output of a call to popsim function
`...`	Additional arguments affecting the summary produced

Value

Summary of population(s) and the logs

`toy_params` *Toy parameters for IBMPopSim-human_popIMD vignette example.*

Description

Toy parameters for IBMPopSim-human_popIMD vignette example.

Usage

```
toy_params
```

Format

A list containing:

- `birth` A list of 3 numeric vectors (`alpha`, `beta`, `TFR_weights`) for creating birth intensity with the Weibull probability density function.
- `swap` A List of one numeric vector and two data frames (`ages`, `intensities` and `distribution`) for creating the swap intensity and kernel functions.

`weibull` *Weibull function.*

Description

The Weibull (density) function is defined as

$$h(x) = \left(\frac{k}{\lambda}\right)\left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k}$$

with $k, \lambda \in (0, +\infty)$.

Usage

```
weibull(k, lambda = 1)
```

Arguments

`k`	Shape parameter, a positive real number.
`lambda`	Scale parameter, a positive real number, defaults to 1.

Details

A C++ version of this function is available. See `vignette('IBMPopSim_cpp')` for more details.

Value

The Weibull density function `dweibull` with shape parameter `k` and scale parameter `lambda`, see `dweibull`.

See Also

`https://en.wikipedia.org/wiki/Weibull_distribution`

Index