Package 'MicSim'

January 31, 2022

Type Package

Title Performing Continuous-Time Microsimulation

Version 1.1.0

Date 2022-01-28

Author Sabine Zinn

Maintainer Sabine Zinn <szinn@diw.de>

Description This entry-level toolkit allows performing continuous-time microsimulation for a wide range of life science (demography, social sciences, epidemiology) applications. Individual life-courses are specified by a continuous-time multi-state model.

Depends R (>= 3.0.1), chron, snowfall, rlecuyer

License GPL-2

NeedsCompilation no

Repository CRAN

Date/Publication 2022-01-31 11:50:02 UTC

R topics documented:

`MicSim-package` *MicSim: Continuous-time microsimulation for population projection*

Description

In life sciences, the central device of microsimulations is the life-course of an individual, which is defined by the sequence of states that the individual visits over time, and the waiting times between these state transitions. Modelling and simulating the life courses of a representative share of population members allows mapping population dynamics on a very detailed scale.

A standard approach to describe individual behavior is a continuous-time multi-state model. A multi-state model is a stochastic process that at any point in time occupies one out of a set of discrete states. These states summarize the demographically relevant categories an individual can belong to. Generally, the state space is determined by the problem to be studied, but commonly it will at least comprise the elementary demographic characteristics of sex and marital status. One element always present in the state space is "dead", a risk to which each individual is always exposed to.

In demographic microsimulations life-courses usually evolve along two time scales: individual age and calendar time. A possible third time scale is the time that an individual has already spent in his/her current demographic state, e.g., the time that has elapsed since the individual's wedding. A demographic event implies a change in the state of an individual. It should be emphasized that age runs parallel to the process time in the model, and therefore birthdays, i.e., completion of another year of life, is not an event in itself.

A common way to characterize an individual life-course is via a trajectory of a stochastic process from the family of Markovian processes, where the process time maps the time span over which we "observe" an individual life-course. The MicSim package uses non-homogeneous continuous-time Markov chains to describe individual life-courses.

The transition rates (also denoted as hazard rates or intensities) of Markovian processes are their key quantities. Once they are known one can compute the distribution functions of sojourn times and thus simulate synthetic life-courses. That is, to run a microsimulation model, for all transitions and time scales considered transition rates have to be provided.

Details

Package:	MicSim
Type:	Package
Version:	1.1.0
Date:	2022-01-28
License:	GPL-2

Author(s)

Sabine Zinn

Maintainer: szinn@diw.de

References

S. Zinn (2014). The MicSim Package of R: An Entry-Level Toolkit for Continuous-Time Microsimulation. In International Journal of Microsimulation 7(3), 3-32.

Willekens, F., & Putter, H. (2014). Software for multistate analysis. Demographic Research, 31, 381-420.

`buildTransitionMatrix` *Determining transition pattern and transition functions*

Description

The function `buildTransitionMatrix` supports the constructing of the 'transition matrix', which determines the transition pattern of the microsimulation model. The actual microsimulation is performed by micSim (sequentially) or by micSimParallel (parallel computing).

Usage

```
buildTransitionMatrix(allTransitions, absTransitions, stateSpace)
```

Arguments

`allTransitions`	A matrix comprising all possible transitions between values of state variables in the first column and in the second column the names of the functions defining the corresponding transition rates.
`absTransitions`	A matrix comprising the names of the absorbing states which individuals are always exposed to (such as "dead"" and emigrated labeled as "rest") in the first column and in the second column the names of the functions defining the corresponding transition rates.
`stateSpace`	A matrix comprising all nonabsorbing states considered during simulation.

Details

The function `buildTransitionMatrix` is an auxiliary function for building the transition matrix required to run the microsimulation using micSim or micSimParallel.

In `stateSpace` all state variables considered during simulation including their values have to be defined. Values are always described using labels. For example, label "M" for being married. Each column of `stateSpace` refers to one state variable considered and each row refers to one state of the state space. Apart from "m" and "f" reserved for male and female (state variable: gender) and "no" and "low" reserved for no education and elementary school attended (state variable: educational attainment), labels can be set arbitrarily.

Each element of the first column of `allTransitions` has to be of the form "A->B" with indicating "A" the starting value of a transition and "B" the arrival value. ("->" is the placeholder defined to mark a transition.) For example, "0" (childless) describes the starting value of the transition marking a first birth event and "1" (first child) its arrival value. All value labels used have to be identical to the value labels of the state variables specifying the simulation model.

All absorbing states listed in the first column of absTransitions have to be given as strings such as "dead" for being dead or "rest" for emigrated. Since dying is a competing risk all individuals are always exposed to, "dead" is a mandatory part of absTransitions.

All transitions can be defined to depend on several state variables. For example, a divorce rate depends on gender and on the fertility status. Therefore, the starting value and the arrival value of a transition have to be specified as a combination of the considered attributes, separated by a forward slash and in accordance with the ordering of the state variables in the state space. For example, "f/A->f/B" describes a female specific transition from "A" to "B" and "f/M/1 -> f/D/1" might describe a mother's (indicated by "1") transition from "M" (e.g., married) to "D" (e.g., divorced). For absorbing states, a prefix indicates the attributes on which a transition is assumed to depend (also separated by forward slashes), e.g., "f/dead" and "m/dead" describe gender specific mortality transitions and "f/M/dead" and "m/M/dead" indicate gender specific mortality rates for married persons.

Value

The transitionMatrix that is mandatory to perform a microsimulation run by micSim (sequentially) or by micSimParallel (parallel computing) is returned. The matrix has as many rows as the simulation model comprises nonabsorbing states and as many columns as the simulation model comprises absorbing and nonabsorbing states. The rows indicate starting states of transitions and the columns signify arrival states. At positions indicating impossible transitions, the matrix contains zeros. Otherwise the name of the function defining the respective transition rates is given.

Author(s)

Sabine Zinn

Examples

```
##############################################################################################
# 1. Example: Transition rates are specified to depend on only one state variable
##############################################################################################

# Defintion of state space, i.e., nonabsorbing and absorbing states
sex <- c("m","f")
fert <- c("0","1","2","3+")
marital <- c("NM","M","D","W")
edu <- c("no","low","med","high")
stateSpace <- expand.grid(sex=sex,fert=fert,marital=marital,edu=edu)

# Possible transitions indicating fertility behavior are "0->1", "1->2", "2->3+",
# and "3+->3+". Here, "->" is the defined placeholder defining a transition.
# `fert1Rates' marks the name of the function defining the transition rates to
# parity one and `fert2Rates' marks the name of the function defining the transition
# rates to higher parities.
# Note: The functions `fert1Rates' and `fert1Rates' are transition rate functions
# defined by the user. Their naming depends on the user's choice.
fertTrMatrix <- cbind(c("0->1","1->2","2->3+","3+->3+"),
                c("fert1Rates", "fert2Rates", "fert2Rates","fert2Rates"))

# Possible transitions indicating changes in the marital status are "NM->M", "M->D",
```

```
# "M->W", "D->M", and "W->M".
# `marriage1Rates' marks the name of the function defining the transition rates for first
# marriage and `marriage2Rates' marks the name of the function defining the transition rates
# for further marriages. `divorceRates' marks the name of the function defining divorce
# rates and `widowhoodRates' marks the name of the function describing transition rates to
# widowhood.
# Note: The functions `marriage1Rates',`marriage2Rates', `divorceRates', and
# `widowhoodRates' are transition rate functions defined by the user.
# Their naming depends on the user's choice.
maritalTrMatrix <- cbind(c("NM->M","M->D","M->W","D->M","W->M"),
                  c("marriage1Rates","divorceRates","widowhoodRates","marriage2Rates",
                    "marriage2Rates"))

# Possible transitions indicating changes in the educational attainment are "no->low",
# "low->med", and "med->high".
# `noToLowEduRates' marks the name of the function defining transition rates for accessing
# primary education, `noToLowEduRates' marks the name of the function defining transition
# rates for graduating with a lower secondary education, and `medToHighEduRates' marks the
# name of the function defining transition rates for graduating with a higher secondary
# education.
# Note: The functions `noToLowEduRates',`noToLowEduRates', and `medToHighEduRates' are
# transition rate functions defined by the user. Their naming depends on the user's
# choice.
eduTrMatrix <- cbind(c("no->low","low->med","med->high"),
               c("noToLowEduRates","noToLowEduRates","medToHighEduRates"))

# Combine all possible transitions and the related transition function into one matrix.
allTransitions <- rbind(fertTrMatrix, maritalTrMatrix, eduTrMatrix)

# Possible absorbing states are `dead' and `rest'. (The latter indicates leaving the
# population because of emigration). The accordant transition rate functions are named
# `mortRates' and `emigrRates'. (Again, naming is up to the user.)
absTransitions <- rbind(c("dead","mortRates"),c("rest","emigrRates"))

# Construct `transition matrix'.
transitionMatrix <- buildTransitionMatrix(allTransitions,absTransitions,stateSpace)

##############################################################################################
# 2. Example: Transition rates are gender specific
##############################################################################################
# Defintion of nonabsorbing and absorbing states
sex <- c("m","f")
stateX <- c("H","P")
stateSpace <- expand.grid(sex=sex,stateX=stateX)
absStates <- c("dead")

# Transitions indicating changes in `stateX'.
# We assume distinct transition rates for females and males.
# Note: The functions `ratesHP_f',`ratesHP_m', `ratesPH_f', and
# `ratesPH_m' are transition rate functions defined by the user.
trMatrix_f <- cbind(c("f/H->f/P","f/P->f/H"),c("ratesHP_f", "ratesPH_f"))
trMatrix_m <- cbind(c("m/H->m/P","m/P->m/H"),c("ratesHP_m", "ratesPH_m"))
allTransitions <- rbind(trMatrix_f,trMatrix_m)
```

```
# We assume gender specific mortality rates.
# Note: The naming and specification of the respective mortality rate functions
# `mortRates_f' and `mortRates_m' depend on the user.
absTransitions <- rbind(c("f/dead","mortRates_f"), c("m/dead","mortRates_m"))

transitionMatrix <- buildTransitionMatrix(allTransitions=allTransitions,
                      absTransitions=absTransitions, stateSpace=stateSpace)
```

`convertToLongFormat` *Reshaping microsimulation output into long format*

Description

The function reshapes the output given by micSim or by micSimParallel into long format. In long format, the data comprises for each episode which an individual experiences one row.

Usage

```
convertToLongFormat(pop,migr=FALSE)
```

Arguments

`pop`	The data frame `pop` contains the whole synthetic population considered during simulation including all events generated. For each individidual `pop` contains as many rows as the individual performed transitions during simulation.
`migr`	A logical variable indicating whether the simulation model considers immigration. The default setting is "no immigration considered": `migr=FALSE`.

Details

`convertToLongFormat` uses information from the definition of the microsimulation model. In particular, it uses `stateSpace`, `absTransitions`, `allTransitions`, `simHorizon`, and optionally `immigrPop`. (For a description of these objects see micSim.) `stateSpace`, `absTransitions`, `allTransitions`, `simHorizon`, and `immigrPop` are globally defined, i.e., they are already part of the workspace. Thus, they do not have to be given to `convertToLongFormat` as extra input parameters.

Value

A data frame comprising the microsimulation output in long format.

- `ID` is the unique numerical person identifier of an individual.

- `birthDate` is the birth date of an individual.

- The variables `Tstart` and `Tstop` mark the start und the ending dates of episodes.

- statusEntry specifies whether the entry into an episode has been observed. Value "1" marks an observed entry and "0" marks a left truncated episode.

- statusExit specifies whether a transition between two states or right censoring completed an episode. Value "1" indicates a transition and "0" a censoring event.

- OD names the transition which completed an episode. Here, right censoring is marked by "cens".

- ns gives the number of episodes an individual has passed.

- Episode enumerates the episodes an individual has passed.

- The last columns of the data frame contain for each individual and episode the values of the state variables during that episode such as 'sex', 'education', etc.

- Birth and transition times are given as calendar dates in form of chron objects.

Author(s)

Sabine Zinn

Examples

```
# Run microsimulation before, e.g., the complex example described on the
# help page of the function "micSim".
## Not run:
pop <- micSim(initPop, immigrPop, transitionMatrix, absStates, initStates, initStatesProb,
        maxAge, simHorizon, fertTr)
popLong <- convertToLongFormat(pop,migr=TRUE)

## End(Not run)
```

---

convertToWideFormat *Reshaping microsimulation output into wide format*

Description

The function reshapes the output given by micSim or by micSimParallel into wide format. In wide format, the data comprises for each episode which an individual experiences additional column entries.

Usage

```
convertToWideFormat(pop)
```

Arguments

pop	The data frame pop contains the whole synthetic population considered during simulation including all events generated. For each individidual pop contains as many rows as the individual performed transitions during simulation.

Value

A data frame comprising the microsimulation output in wide format.

- `ID` is the unique numerical person identifier of an individual.

- `birthDate` is the birth date of an individual.

- `initState` is the state in which an individual initially entered the virtual population of the simulation.

- `ns` gives the number of (completed) episodes an individual has passed.

- The variables `From.i` and `To.i` mark the start und the arrival state of the transition corresponding to episode `i`. The variables `transitionTime.i` and `transitionAge.i` give the corresponding transition time and age. The enumerator `i` ranges from 1 to the maximal number of transitions which an individual experienced during simulation. Only completed episodes are counted.

Author(s)

Sabine Zinn

Examples

```
# Run microsimulation before, e.g., the complex example described on the
# help page of the function "micSim".
## Not run:
pop <- micSim(initPop, immigrPop, transitionMatrix, absStates, initStates,
    initStatesProb, maxAge, simHorizon, fertTr)
popWide <- convertToWideFormat(pop)

## End(Not run)
```

`micSim` *Run microsimulation (sequentially)*

Description

Performs a continuous-time microsimulation run (sequentially, i.e., using only one CPU core).

Usage

```
micSim(initPop, immigrPop = NULL, transitionMatrix, absStates = NULL,
  initStates = c(), initStatesProb = c(), maxAge = 99, simHorizon,
  fertTr = c(), dateSchoolEnrol="09/01", reportMothers=FALSE)
```

Arguments

initPop	Data frame comprising the starting population of the simulation.
immigrPop	Data frame comprising information about the immigrants entering the population across simulation time.
transitionMatrix	A matrix indicating the transition pattern and the names of the functions determining the respective transition rates.
absStates	A vector indicating the absorbing states of the model.
initStates	A vector comprising all initial states that newborns might enter.
initStatesProb	A vector comprising the probabilities corresponding to initStates. In sum, these probabilities have to be one.
maxAge	A scalar indicating the maximal age which an individual can reach during simulation. maxAge has to be greater than zero
simHorizon	A vector comprising the starting and ending date of the simulation. Both dates have to be chron objects. The starting date has to precede the ending date.
fertTr	A vector indicating all transitions triggering a child birth event during simulation, that is, the creation of a new individual.
dateSchoolEnrol	A string of the form "month/day" indicating the general enrollment date for elementary school, e.g., "09/01" for September 1st. The default setting is "09/01".
reportMothers	A logical indicating whether the ID of the mother should be stored in the function's output for all newborns generated during simulation.

Details

All nonabsorbing states considered during simulation have to be defined as composite states. In more detail, they consist of labels indicating values of state variables. Within states, labels are separated by a forward slash "/". Possible state variables are, for example, gender, number of children ever born, and educational attainment. Corresponding values are, for example, "m" and "f" (gender), "0","1","2", and "3+" (number of children ever born), "no", "low", "med", and "high" (educational attainment). Possible examples of states are "m/0/low" for a childless male with elementary education or "f/1/high" for a female with one child and a higher secondary school degree. All state variables considered plus accordant value labels have to be provided by the user. The only exception is gender which is predefined by labels "m" and "f" indicating male and female individuals. The label values "no" and "low" are reserved for enrolment events to elementary school (see below).

Nonabsorbing states have to be given as strings such as "dead" for being dead or "rest" for emigrated.

micSim is able to conduct enrollment events to elementary school such that they take place on dateSchoolEnrol of a particular year. For this purpose, a state variable defining educational attainment has to be created first. Then, labels of possible values have to be defined such that "no" describes no education and "low" describes elementary education. Finally, the transition function determining the transition rate for the respective enrollment event has to be defined to return "Inf" for the age x at which children should be enrolled (e.g., at age seven) and zero otherwise. That way,

an event "school enrollment on dateSchoolEnrol of the year in which a child turns x years old" is enforced.

If educational attainment is not considered, dateSchoolEnrol can let be unspecified: dateSchoolEnrol=c().

The starting population initPop has to be given in the form of a data frame. Each row of the data frame corresponds to one individual. initPop has to comprise the following information: unique numerical person identifier (ID), birth date, and initial state (i.e., the state occupied by the individual when entering the synthetic population). Birth dates have to be chron objects.

Information about immigrants has to be given in the form of a data frame (immigrPop). Each row of the data frame corresponds to one immigrant. immigrPop contains the following data: unique numerical person identifier (ID), immigration date, birth date, and initial state (i.e., the state occupied by the immigrant when entering the simulated population). Immigration dates and birth dates have to be chron objects.

For each transition that should be considered during simulation accordant transition rates have to be provided. micSim requires these rates in form of functions which are handed over via the transition matrix transitionMatrix (described in the subsequent paragraph). The MicSim package allows rates to depend on three time scales: age, calendar time, and the time that has elapsed since the last change of a particular state variable (e.g., the time elapsed since wedding). In accordance therewith, micSim requires transition rates functions to feature three input parameters, namely age, calTime, and duration. Via age the age of an individual is handed over, via caltime the calendar time, and via duration the time that has elapsed since the last change of the affected state variable. All three input parameters might vary, or only one or two of them. Also none of the input parameters can be specified to vary, i.e., a transition rate can be defined to be constant. If rates are assumed to be independent of a specific time scale, the corresponding input argument can simply be ignored within the body of the rates function (i.e., is not used to determine a specific rate value). For illustration, see the examples in the example section. Note that allowing transition rates to vary along the time elapsed since a last transition facilitates modelling gestation gaps after a delivery: For a period of nine or ten months transition rates for higher order parities are simply set to zero (cf., the complex example in the example section).

The transition matrix transitionMatrix has as many rows as the simulation model comprises nonabsorbing states and as many columns as the simulation model comprises absorbing and nonabsorbing states. The rows of transitionMatrix mark starting states of transitions and the columns mark arrival states. At positions of transitionMatrix indicating impossible transitions, the matrix contains zeros. Otherwise the name of the function determining the respective transition rates has to be given. The function buildTransitionMatrix supports the construction of transitionMatrix.

If, during simulation, an individual reaches maxAge, he/she stays in his/her current state until simulation ending date is reached, that is, the respective individual is no longer at risk of experiencing any events and his/her ongoing episode will be censored at simlation ending date.

It is recommended to set simHorizon using the function setSimHorizon.

Each element of fertTr has to be of the form "A->B", that is, "A" indicates the starting attribute of the transition and "B" the arrival attribute. ("->" is the placeholder defined to mark a transition.) For example, "0" (childless) gives the starting point of the transition marking a first birth event and "1" (first child) its arrival point. All fertility attributes given in fertTr have to be part of the state variable specifiying fertility in the state space. That is, if there is none, fertTr is empty: fertTr=c().

Value

The data frame pop contains the whole synthetic population considered during simulation including all events generated. In more detail, pop contains as many rows as there are transitions performed by the individuals. (Also, "entering the population" is considered as an event. In general, individuals can enter the simulation via three channels: by being part of the starting population, by immigration, and by being born during simulation). If `reportMothers` is set to `TRUE` in the function's input arguments and fertility events are part of the model's specification, pop contains an additional column indicating the ID of the mother for individuals born during simulation.

The function convertToLongFormat reshapes the microsimulation output into long format, while the function convertToWideFormat gives the microsimulation in wide format.

Note

Concerning run times `micSim` is not very performant. That is because it is purely implemented in R, i.e., it does not incorporate routines implemented in a high level programming language like Java, C++ or Python. Furthermore, the current code contains a lot of string splits and format conversions. The reason for this are twofold (i) minimize dependencies to other R packages and (ii) a allow a handy and flexible model specification and structure. The long long run times are certainly an issue to tackle with the next version(s) of the package. For the meantime, if a computer cluster is accessible, I recommend parallel computing using micSimParallel. This speeds up execution times considerably.

Author(s)

Sabine Zinn

Examples

```
##############################################################################################
# 1. Simple example only dealing with mortality events
##############################################################################################

# Clean workspace
rm(list=ls())

# Defining simulation horizon
simHorizon <- setSimHorizon(startDate="01/01/2000", endDate="31/12/2100")

# Seed for random number generator
set.seed(234)

# Definition of maximal age
maxAge <- 120

# Defintion of nonabsorbing and absorbing states
sex <- c("m","f")
stateSpace <- sex
attr(stateSpace,"name") <- "sex"
absStates <- "dead"
```

```
# Definition of an initial population
dts <- c("31/12/1930","03/04/1999","15/10/1956","11/11/1991","01/01/1965")
birthDates <- chron(dates=dts,format=c(dates="d/m/Y"))
initStates <- c("f","m","f","m","m")
initPop <- data.frame(ID=1:5,birthDate=birthDates,initState=initStates)

# Definition of mortality rates (Gompertz model).
mortRates <- function(age, calTime, duration){
  a <- 0.00003
  b <- ifelse(calTime<=2020, 0.1, 0.097)
  rate <- a*exp(b*age)
  return(rate)
}

# Transition pattern and assignment of functions specifying transition rates
absTransitions <- c("dead","mortRates")
transitionMatrix <- buildTransitionMatrix(allTransitions=NULL,
  absTransitions=absTransitions, stateSpace=stateSpace)

# Execute microsimulation (sequentially, i.e., using only one CPU)
pop <- micSim(initPop=initPop, transitionMatrix=transitionMatrix, absStates=absStates,
  maxAge=maxAge, simHorizon=simHorizon)


##############################################################################################
# 2. More complex example dealing with mortality, changes in the fertily and the marital
# status, in the educational attainment, as well as dealing with migration
##############################################################################################
# Clean workspace
rm(list=ls())

# Defining simulation horizon
simHorizon <- setSimHorizon(startDate="01/01/2014", endDate="31/12/2024")

# Seed for random number generator
set.seed(234)

# Definition of maximal age
maxAge <- 100

# Defintion of nonabsorbing and absorbing states
sex <- c("m","f")
fert <- c("0","1+")
marital <- c("NM","M","D","W")
edu <- c("no","low","med","high")
stateSpace <- expand.grid(sex=sex,fert=fert,marital=marital,edu=edu)
absStates <- c("dead","rest")

# General date of enrollment to elementary school
dateSchoolEnrol <- "09/01"

# Assign to all newborns born during simulation the ID of the mother
reportMothers <- TRUE
```

```
# Definition of an initial population (for illustration purposes, create a random population)
N = 100
initBirthDatesRange <- chron(dates=c("31/12/1950","01/01/2014"), format=c(dates="d/m/Y"),
  out.format=c(dates="d/m/year"))
birthDates <- dates(initBirthDatesRange[1] + runif(N, min=0, max=diff(initBirthDatesRange)))
getRandInitState <- function(birthDate){
  age <- trunc(as.numeric(simHorizon[1] - birthDate)/365.25)
  s1 <- sample(sex,1)
  s2 <- ifelse(age<=18, fert[1], sample(fert,1))
  s3 <- ifelse(age<=18, marital[1], ifelse(age<=22, sample(marital[1:3],1),
    sample(marital,1)))
 s4 <- ifelse(age<=7, edu[1], ifelse(age<=18, edu[2], ifelse(age<=23, sample(edu[2:3],1),
    sample(edu[-1],1))))
  initState <- paste(c(s1,s2,s3,s4),collapse="/")
  return(initState)
}
initPop <- data.frame(ID=1:N, birthDate=birthDates,
  initState=sapply(birthDates, getRandInitState))

# Definition of immigrants entering the population (for illustration purposes, create immigrants
# randomly)
M = 20
immigrDatesRange <- as.numeric(simHorizon)
immigrDates <- dates(chron(immigrDatesRange[1] + runif(M, min=0,max=diff(immigrDatesRange)),
  format=c(dates="d/m/Y", times="h:m:s"), out.format=c(dates="d/m/year",times="h:m:s")))
immigrAges <- runif(M, min=15*365.25, max=70*365.25)
immigrBirthDates <- dates(chron(as.numeric(immigrDates) - immigrAges,
 format=c(dates="d/m/Y", times="h:m:s"), out.format=c(dates="d/m/year", times="h:m:s")))
IDmig <- max(as.numeric(initPop[,"ID"]))+(1:M)
immigrPop <- data.frame(ID = IDmig, immigrDate = immigrDates, birthDate=immigrBirthDates,
  immigrInitState=sapply(immigrBirthDates, getRandInitState))

# Definition of initial states for newborns
initStates <- rbind(c("m","0","NM","no"),c("f","0","NM","no"))
# Definition of related occurrence probabilities
initStatesProb <- c(0.515,0.485)

# Definition of (possible) transition rates
# (1) Fertility rates (Hadwiger mixture model)
fert1Rates <- function(age, calTime, duration){  # parity 1
  b <- ifelse(calTime<=2020, 3.9, 3.3)
  c <- ifelse(calTime<=2020, 28, 29)
  rate <-  (b/c)*(c/age)^(3/2)*exp(-b^2*(c/age+age/c-2))
  rate[age<=15 | age>=45] <- 0
  return(rate)
}
fert2Rates <- function(age, calTime, duration){  # partiy 2+
  b <- ifelse(calTime<=2020, 3.2, 2.8)
  c <- ifelse(calTime<=2020, 32, 33)
  rate <-  (b/c)*(c/age)^(3/2)*exp(-b^2*(c/age+age/c-2))
  rate[age<=15 | age>=45 | duration<0.75] <- 0
  return(rate)
```

```
}
# (2) Rates for first marriage (normal density)
marriage1Rates <- function(age, calTime, duration){
  m <- ifelse(calTime<=2020, 25, 30)
  s <- ifelse(calTime<=2020, 3, 3)
  rate <- dnorm(age, mean=m, sd=s)
  rate[age<=16] <- 0
  return(rate)
}
# (3) Remariage rates (log-logistic model)
marriage2Rates <- function(age, calTime, duration){
  b <- ifelse(calTime<=2020, 0.07, 0.10)
  p <- ifelse(calTime<=2020, 2.7,2.7)
  lambda <- ifelse(calTime<=1950, 0.04, 0.03)
  rate <- b*p*(lambda*age)^(p-1)/(1+(lambda*age)^p)
  rate[age<=18] <- 0
  return(rate)
}
# (4) Divorce rates (normal density)
divorceRates <- function(age, calTime, duration){
  m <- 40
  s <- ifelse(calTime<=2020, 7, 6)
  rate <- dnorm(age,mean=m,sd=s)
  rate[age<=18] <- 0
  return(rate)
}
# (5) Widowhood rates (gamma cdf)
widowhoodRates <- function(age, calTime, duration){
  rate <- ifelse(age<=30, 0, pgamma(age-30, shape=6, rate=0.06))
  return(rate)
}
# (6) Rates to change educational attainment
# Set rate to `Inf' to make transition for age 7 deterministic.
noToLowEduRates <- function(age, calTime, duration){
  rate <- ifelse(age==7,Inf,0)
  return(rate)
}
lowToMedEduRates <- function(age, calTime, duration){
  rate <- dnorm(age,mean=16,sd=1)
  rate[age<=15 | age>=25] <- 0
  return(rate)
}
medToHighEduRates <- function(age, calTime, duration){
  rate <- dnorm(age,mean=20,sd=3)
  rate[age<=18 | age>=35] <- 0
  return(rate)
}
# (7) Mortality rates (Gompertz model)
mortRates <- function(age, calTime, duration){
  a <- .00003
  b <- ifelse(calTime<=2020, 0.1, 0.097)
  rate <- a*exp(b*age)
  return(rate)
```

```
}
# (8) Emigration rates
emigrRates <- function(age, calTime, duration){
  rate <- ifelse(age<=18,0,0.0025)
  return(rate)
}

# Transition pattern and assignment of functions specifying transition rates
fertTrMatrix <- cbind(c("0->1+","1+->1+"),
  c("fert1Rates", "fert2Rates"))
maritalTrMatrix <- cbind(c("NM->M","M->D","M->W","D->M","W->M"),
  c("marriage1Rates","divorceRates","widowhoodRates",
 "marriage2Rates","marriage2Rates"))
eduTrMatrix <- cbind(c("no->low","low->med","med->high"),
  c("noToLowEduRates","lowToMedEduRates","medToHighEduRates"))
allTransitions <- rbind(fertTrMatrix, maritalTrMatrix, eduTrMatrix)
absTransitions <- rbind(c("dead","mortRates"),c("rest","emigrRates"))
transitionMatrix <- buildTransitionMatrix(allTransitions=allTransitions,
  absTransitions=absTransitions, stateSpace=stateSpace)

# Define transitions triggering a birth event
fertTr <- fertTrMatrix[,1]

# Execute microsimulation (sequentially, i.e., using only one CPU core)
pop <- micSim(initPop=initPop, immigrPop=immigrPop,
  transitionMatrix=transitionMatrix, absStates=absStates,
  initStates=initStates, initStatesProb=initStatesProb,
  maxAge=maxAge, simHorizon=simHorizon, fertTr=fertTr,
  dateSchoolEnrol=dateSchoolEnrol, reportMothers=reportMothers)
```

`micSimParallel` *Run microsimulation (parallel computing)*

Description

The function `micSimParallel` is a parallelized version of the function micSim. That is, it runs a continuous-time microsimulation simulation distributed, i.e., using more than one CPU core.

Usage

```
micSimParallel(initPop, immigrPop = NULL, transitionMatrix, absStates = NULL,
  initStates = c(), initStatesProb = c(), maxAge = 99, simHorizon, fertTr = c(),
  dateSchoolEnrol="09/01", reportMothers=FALSE, cores=1, seeds=1254)
```

Arguments

`initPop`

`immigrPop`

transitionMatrix

absStates

initStates

initStatesProb

maxAge

simHorizon

fertTr

dateSchoolEnrol

reportMothers	See micSim.
cores	Number of CPUs to be used.
seeds	Seeds for pseudo number generators used for parallel computing.

Details

The argument cores must not exceed the number of cores of the computer (cluster) used.

In seeds as many seeds should be given as cores are used. If less are given, the latter are repeated to complete the set of seeds.

Value

The data frame pop contains the whole synthetic population considered during simulation including all events generated. For more details, see micSim.

Author(s)

Sabine Zinn

Examples

```
##############################################################################################
# Complex example dealing with mortality, changes in the fertily and the marital
# status, in the educational attainment, as well as dealing with migration
# (A simpler example is given on the help page of the "micSim" function of this package.)
##############################################################################################

# Clean workspace
rm(list=ls())

# Defining simulation horizon
simHorizon <- setSimHorizon(startDate="01/01/2014", endDate="31/12/2024")

# Seed for random number generator
set.seed(234)

# Definition of maximal age
maxAge <- 100
```

```
# Defintion of nonabsorbing and absorbing states
sex <- c("m","f")
fert <- c("0","1+")
marital <- c("NM","M","D","W")
edu <- c("no","low","med","high")
stateSpace <- expand.grid(sex=sex,fert=fert,marital=marital,edu=edu)
absStates <- c("dead","rest")

# General date of enrollment to elementary school
dateSchoolEnrol <- "09/01"

# Definition of an initial population (for illustration purposes, create a random population)
N = 10000
initBirthDatesRange <- chron(dates=c("31/12/1950","01/01/2014"),format=c(dates="d/m/Y"),
  out.format=c(dates="d/m/year"))
set.seed(124) # Set a seed for the random number generator (to make results repeatable)
birthDates <- dates(initBirthDatesRange[1] + runif(N, min=0, max=diff(initBirthDatesRange)))
getRandInitState <- function(birthDate){
  age <- trunc(as.numeric(simHorizon[1] - birthDate)/365.25)
  s1 <- sample(sex,1)
  s2 <- ifelse(age<=18, fert[1], sample(fert,1))
  s3 <- ifelse(age<=18, marital[1], ifelse(age<=22, sample(marital[1:3],1),
    sample(marital,1)))
 s4 <- ifelse(age<=7, edu[1], ifelse(age<=18, edu[2], ifelse(age<=23, sample(edu[2:3],1),
    sample(edu[-1],1))))
  initState <- paste(c(s1,s2,s3,s4),collapse="/")
  return(initState)
}
initPop <- data.frame(ID=1:N, birthDate=birthDates,
  initState=sapply(birthDates, getRandInitState))

# Definition of immigrants entering the population (for illustration purposes, create immigrants
# randomly)
M = 2000
immigrDatesRange <- as.numeric(simHorizon)
immigrDates <- dates(chron(immigrDatesRange[1] + runif(M, min=0,max=diff(immigrDatesRange)),
  format=c(dates="d/m/Y", times="h:m:s"), out.format=c(dates="d/m/year",times="h:m:s")))
immigrAges <- runif(M, min=15*365.25, max=70*365.25)
immigrBirthDates <- dates(chron(as.numeric(immigrDates) - immigrAges,
 format=c(dates="d/m/Y", times="h:m:s"), out.format=c(dates="d/m/year", times="h:m:s")))
IDmig <- max(as.numeric(initPop["ID"]))+(1:M)
immigrPop <- data.frame(ID = IDmig, immigrDate = immigrDates, birthDate=immigrBirthDates,
  immigrInitState=sapply(immigrBirthDates, getRandInitState))

# Definition of initial states for newborns
initStates <- rbind(c("m","0","NM","no"),c("f","0","NM","no"))
# Definition of related occurrence probabilities
initStatesProb <- c(0.515,0.485)

# Definition of (possible) transition rates
# (1) Fertility rates (Hadwiger mixture model)
fert1Rates <- function(age, calTime, duration){  # parity 1
```

```
  b <- ifelse(calTime<=2020, 3.9, 3.3)
  c <- ifelse(calTime<=2020, 28, 29)
  rate <-  (b/c)*(c/age)^(3/2)*exp(-b^2*(c/age+age/c-2))
  rate[age<=15 | age>=45] <- 0
  return(rate)
}
fert2Rates <- function(age, calTime, duration){  # partiy 2+
  b <- ifelse(calTime<=2020, 3.2, 2.8)
  c <- ifelse(calTime<=2020, 32, 33)
  rate <-  (b/c)*(c/age)^(3/2)*exp(-b^2*(c/age+age/c-2))
  rate[age<=15 | age>=45 | duration<0.75] <- 0
  return(rate)
}
# (2) Rates for first marriage (normal density)
marriage1Rates <- function(age, calTime, duration){
  m <- ifelse(calTime<=2020, 25, 30)
  s <- ifelse(calTime<=2020, 3, 3)
  rate <- dnorm(age, mean=m, sd=s)
  rate[age<=16] <- 0
  return(rate)
}
# (3) Remariage rates (log-logistic model)
marriage2Rates <- function(age, calTime, duration){
  b <- ifelse(calTime<=2020, 0.07, 0.10)
  p <- ifelse(calTime<=2020, 2.7, 2.7)
  lambda <- ifelse(calTime<=1950, 0.04, 0.03)
  rate <- b*p*(lambda*age)^(p-1)/(1+(lambda*age)^p)
  rate[age<=18] <- 0
  return(rate)
}
# (4) Divorce rates (normal density)
divorceRates <- function(age, calTime, duration){
  m <- 40
  s <- ifelse(calTime<=2020, 7, 6)
  rate <- dnorm(age,mean=m,sd=s)
  rate[age<=18] <- 0
  return(rate)
}
# (5) Widowhood rates (gamma cdf)
widowhoodRates <- function(age, calTime, duration){
  rate <- ifelse(age<=30, 0, pgamma(age-30, shape=6, rate=0.06))
  return(rate)
}
# (6) Rates to change educational attainment
# Set rate to `Inf' to make transition for age 7 deterministic.
noToLowEduRates <- function(age, calTime, duration){
  rate <- ifelse(age==7,Inf,0)
  return(rate)
}
lowToMedEduRates <- function(age, calTime, duration){
  rate <- dnorm(age,mean=16,sd=1)
  rate[age<=15 | age>=25] <- 0
  return(rate)
```

```
}
medToHighEduRates <- function(age, calTime, duration){
  rate <- dnorm(age,mean=20,sd=3)
  rate[age<=18 | age>=35] <- 0
  return(rate)
}
# (7) Mortality rates (Gompertz model)
mortRates <- function(age, calTime, duration){
  a <- 0.00003
  b <- ifelse(calTime<=2020, 0.1, 0.097)
  rate <- a*exp(b*age)
  return(rate)
}
# (8) Emigration rates
emigrRates <- function(age, calTime, duration){
  rate <- ifelse(age<=18,0,0.0025)
  return(rate)
}

# Transition pattern and assignment of functions specifying transition rates
fertTrMatrix <- cbind(c("0->1+","1+->1+"),
  c("fert1Rates", "fert2Rates"))
maritalTrMatrix <- cbind(c("NM->M","M->D","M->W","D->M","W->M"),
  c("marriage1Rates","divorceRates","widowhoodRates",
  "marriage2Rates","marriage2Rates"))
eduTrMatrix <- cbind(c("no->low","low->med","med->high"),
  c("noToLowEduRates","lowToMedEduRates","medToHighEduRates"))
allTransitions <- rbind(fertTrMatrix, maritalTrMatrix, eduTrMatrix)
absTransitions <- rbind(c("dead","mortRates"),c("rest","emigrRates"))
transitionMatrix <- buildTransitionMatrix(allTransitions=allTransitions,
  absTransitions=absTransitions, stateSpace=stateSpace)

# Define transitions triggering a birth event
fertTr <- fertTrMatrix[,1]


# Run microsimulation on cluster with five cores (settings depend on cluster used)
## Not run:
cores <- 5
seeds <- c(1233,1245,1234,5,2)
pop <- micSimParallel(initPop=initPop, immigrPop=immigrPop,
  transitionMatrix=transitionMatrix, absStates=absStates, initStates=initStates,
  initStatesProb=initStatesProb, maxAge=maxAge, simHorizon=simHorizon,
  fertTr=fertTr, dateSchoolEnrol=dateSchoolEnrol,
  cores=cores, seeds=seeds)

## End(Not run)
```

---

setSimHorizon *Set simulation horizon*

Description

The function sets the simulation horizon of the microsimulation. The actual microsimulation is performed by micSim (sequentially) or by micSimParallel (parallel computing).

Usage

```
setSimHorizon(startDate, endDate)
```

Arguments

`startDate`	Starting date of simulation given as string of the format "dd/mm/yyyy".
`endDate`	End date of simulation given as string of the format "dd/mm/yyyy".

Details

The starting date has to precede the ending date.

Value

A vector of two chron objects indicating the simulation horizon of the simulation.

Author(s)

Sabine Zinn

Examples

```
setSimHorizon(startDate="01/01/2000", endDate="31/12/2010")
```

Index