# Package ‘MicSim’

January 23, 2024

**Type** Package

**Title** Performing Continuous-Time Microsimulation

**Version** 2.0.1

**Date** 2024-01-23

**Maintainer** Sabine Zinn <szinn@diw.de>

**Description** This 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 as described in Zinn (2014) <doi:10.34196/IJM.00105>.

**LazyData** true

**Depends** R (>= 4.2.0), snowfall, rlecuyer

**License** GPL-2

**Suggests** rmarkdown, knitr, glue

**VignetteBuilder** knitr

**Encoding** UTF-8

**NeedsCompilation** no

**Author** Sabine Zinn [aut, cre],
Claudio Bosco [ctb],
Maurizio Teobaldell [ctb]

**Repository** CRAN

**Date/Publication** 2024-01-23 17:02:50 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 state, e.g., the time that has elapsed since an individual's wedding. An event implies a change in the state of an individual. Age always runs parallel to the process time of a model. Therefore birthday, i.e., the 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 time-inhomogeneuous Markov models to describe individual life-courses. That way, transition intensities can vary at each point in time, i.e. are not assumed to be constant for predefined time intervals (such as whole years).

The transition intensities (also denoted as hazard rates or transition rates) 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. A whole bunch of statistical estimation approaches exist to estimate transition rates from (e.g., register, survey, panel) data. Furthermore, also methods for approximating transition rates from probabilities are available, e.g. by assuming that they are constant in the interval covered by a probability, yielding the so called exponential model. More details on this are given in the description of the 'micsim' function of this package, which is the actual workhorse of this toolkit.

## Details

| | |
|---|---|
| Package: | MicSim |
| Type: | Package |
| Version: | 2.0.1 |
| Date: | 2024-01-23 |
| 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 slashs), 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)
```

## getAgeInDays — *Get from a given date the age in days*

### Description

Function computes for a given date the correct age in days.

### Usage

```
getAgeInDays(currDate, birthDate)
```

### Arguments

| | |
|---|---|
| `currDate` | Reference date given as string of the format "yyyymmdd". |
| `birthDate` | Birth date given as string of the format "yyyymmdd". |

### Value

Correct age at the specific date `currDate` in days

### Author(s)

Sabine Zinn

### Examples

```
getAgeInDays("20200826", "19800605")
```

## getDay — *Get the day in a month (in a year) from days elapsed since 01-01-1970*

### Description

Function computes from days elapsed since 01-01-1970 the day in a month (in a year).

### Usage

```
getDay(daysSince01011970)
```

### Arguments

`daysSince01011970`
: Days elapsed since 1970-01-01

### Value

Day in a month (in a year) computed from days elapsed since 01-01-1970

## Author(s)

Sabine Zinn

## Examples

```
getDay(2561)
```

---

`getInDateFormat` *Get date in the format 'yyyyddmm' from days elapsed since 01-01-1970*

---

## Description

Function generates from days elapsed since 01-01-1970 the date in the string format 'yyyyddmm'.

## Usage

```
getInDateFormat(daysSince01011970)
```

## Arguments

`daysSince01011970`
: Days elapsed since 1970-01-01

## Value

Date in string format 'yyyyddmm' from days elapsed since 01-01-1970

## Author(s)

Sabine Zinn

## Examples

```
getInDateFormat(2561)
```

getInDays — *Get from a date given in the numeric format yyyymmdd the number of days elapsed since 1970-01-01*

## Description

Function computes the days that have pasted since 1970-01-01 up to the currDate (in the numeric format yyyymmdd)

## Usage

```
getInDays(currDate)
```

## Arguments

| | |
|---|---|
| `currDate` | Date given as string of the numeric format yyyymmdd. |

## Value

Number of days elapsed since 1970-01-01.

## Author(s)

Sabine Zinn

## Examples

```
getInDays(20200826)
```

getInDays_my — *Get the number of days that have pasted from 1970-01-01 until 'yyyymm11'.*

## Description

Function computes the number of days that have pasted from 1970-01-01 until 'yyyymm11'.

## Usage

```
getInDays_my(year, month)
```

## Arguments

| | |
|---|---|
| `year` | Year for which days elapsed should be computed, i.e., the yyyy in 'yyyymm11' |
| `month` | Month for which days elapsed should be computed, i.e., the mm in 'yyyymm11' |

### Value

Number of days that have pasted from 1970-01-01 until 'yyyymm11'

### Author(s)

Sabine Zinn

### Examples

```
getInDays_my(2020, 12)
```

---

## getMonth — *Get the month in a year from days elapsed since 01-01-1970*

---

### Description

Function computes from days elapsed since 01-01-1970 the related month a year.

### Usage

```
getMonth(daysSince01011970)
```

### Arguments

`daysSince01011970`
: Days elapsed since 1970-01-01

### Value

Month in a year computed from days elapsed since 01-01-1970

### Author(s)

Sabine Zinn

### Examples

```
getMonth(2561)
```

## getYear — *Get the calendar year from days elapsed since 01-01-1970*

### Description

Function computes from days elapsed since 01-01-1970 the related calendar year.

### Usage

```
getYear(daysSince01011970)
```

### Arguments

`daysSince01011970`
: Days elapsed since 1970-01-01

### Value

Calendar year from days elapsed since 01-01-1970

### Author(s)

Sabine Zinn

### Examples

```
getYear(2561)
```

## immigrPopMigrExp — *One possible population of migrants for the MicSim package.*

### Description

Population of migrants for the MicSim package with 3758 migrants and migration dates between 01-01-2014 and 30-12-2018.

### Usage

```
data("immigrPopMigrExp")
```

### Format

A data frame with information of birthdates, date of immigration and state at immigration of 3758 migrants.

`ID` Personal identifier

`immigrDate` Immigration date

`birthDate` Birth date of migrants

`immigrInitState` State at immigration

## Details

This is a example data set for the MicSim package. The population of migrants is already in the format that is required by the package. For more details on this see micSim. The state space for the states and the related state domains are defined in the vignette of this package. The related application is also part of the vignette.

## Source

European Commission

## Examples

```
data(immigrPopMigrExp)
```

---

`initPopMigrExp` *One possible initial population for the MicSim package.*

---

## Description

Initial population for the MicSim package with 72965 persons with birthdates from 08-03-1914 to 30-12-2013.

## Usage

```
data("initPopMigrExp")
```

## Format

A data frame with information of birthdates, date of immigration and state at immigration of 3758 migrants.

`ID` Personal identifier

`birthDate` Birth date of entity

`initState` Initial state

## Details

This is a example data set for the MicSim package. The initial population is already in the format that is required by the package. For more details on this see micSim. The state space for the states and the related state domains are defined in the vignette of this package. The related application is also part of the vignette.

## Source

European Commission

## Examples

```
data(initPopMigrExp)
```

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,
        fixInitStates = c(), varInitStates=c(), initStatesProb=c(),
        maxAge=99, simHorizon, fertTr=c(), monthSchoolEnrol=c())
```

## 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 (with rates to be returned as vectors, i.e. for input age 0 to 10 eleven rate values have to be returned). |
| `absStates` | A vector indicating the absorbing states of the model. |
| `fixInitStates` | (Vector of) Indices of substates determining the attributes/substates that a newborn will be taken over from the mother. If empty or not defined, no attributes will be inherited. |
| `varInitStates` | (A vector comprising the) Substates / attributes that are assigned to a newborn randomly according to the probabilities `initStatesProb`, i.e. that are not inherited from the mother. |
| `initStatesProb` | A vector comprising the probabilities corresponding to `varInitStates`. If `fixInitStates` are given (i.e. attributes from the mother are inherited), these probabilities have to sum to one conditioned on the inherited attributes, i.e. for each (set of) inherited attribute(s) separately. Otherwise, the sum of `initStatesProb` has to be one. |
| `maxAge` | A scalar indicating the exact maximal age (i.e., sharp 100.00 years) 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 given as strings in the format 'yyyymmdd'. 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. |
| `monthSchoolEnrol` | The month (as numeric value from 1 to 12) indicating the general enrollment month for elementary school, e.g., 9 for September. If transition to elementary school is not defined (see below under 'details') and no such month is given school enrollment to elementary school is not modelled / simulated. |

## 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 the first day of the `monthSchoolEnrol`th month 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. A related illustration is given below in the second example.

If school enrollment is not of interest to the modeller, `monthSchoolEnrol` can let be unspecified. Then during simulation that feature is ignored.

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 given as strings in the format ‘yyyymmdd’, e.g. ‘20220815’ for Aug 15th 2022. Be aware that at simulation starting date all individuals in the initial population have already to be born and younger than `maxAge`. Otherwise, `micSim` throws an error message pointing to this issue.

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 provided as strings in the format ‘yyyymmdd’, e.g. ‘20220815’ for Aug 15th 2022. Immigration dates have to be specified to occur after simulation starting date and before simulation stopping date. Immigrants must be born when they migrate. Otherwise, `micSim` throws error messages pointing to this issues.

For each transition that should be considered during simulation accordant transition rates have to be provided. Since MicSim’s model is a continuous-time multi-state model these rates are transition intensities (as also used for defining time-inhomogeneuous Markov models) and not probabilities. Palloni (2000) illustrates very well the difference between both concepts. Zinn (2011) describes methods for estimating rates for MicSim’s model. A crude way of transforming transition probabilities to rates is assuming that the rates $lambda_{ij}$ (for leaving state $i$ to enter state $j$) are constant in the time interval (of length $t$) captured by a corresponding probability $p_{ij}$:

$p_{ij} = 1 - exp(-lambda_{ij} * t)$ which yields $lambda_{ij} = -1/t * ln(1 - p_{ij})$.

Be aware that this is only an approximation since this formula belongs to a time-homogeneuous Markov model and not to the more flexible time-inhomogeneuous Markov model (as used by MicSim). Thus, here for the time interval covered by $p_{ij}$ a time-homogeneuous Markov model is assumed. Many users may have annual transition probabilities at hand, i.e., $t = 1$.

`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., transition rates can be defined to be constant. Since `micSim` computes integrals of rates along simulation procedure, the rates functions must deliver vector of rates for vectors of inputs, i.e. for an input vector of ages (e.g. ages [0,1,2,3]) the rates functions have to given as many rate values as is the length of the age vector (in the example, four rate values). More details on this are given in the examples below or in the vignette to this package. 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. Beware that rates for age have to be delivered at maximal only until `maxAge`. If *more* rates are given, this does not cause an error but they are not used. 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 (e.g., see 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.

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 fertility events are part of the model's specification (i.e., `fertTr` is not empty), pop contains an additional column indicating the ID of the mother for individuals born during simulation. For all other individuals, the ID of the mother is unknown (i.e., set to 'NA').

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

### Note

For large-scale models and simulation, I recommend parallel computing using micSimParallel. This speeds up execution times considerably. However, before running an extensive simulation on multiple cores, the package user should definitely check whether the input for the simulation fits. This can best be achieved by first running a short and less extensive simulation with only one core (e.g., running only a one percent sample of the initial population).

### Author(s)

Sabine Zinn

### References

Palloni, A. (2001). Increment-Decrement Life Tables. In: Preston, S., Heuveline, P., & Guillot, M. (eds). Demography: measuring and modeling population processes. Malden, MA: Blackwell Publishers.

Zinn, S. (2011). Preparation of required input data. In: Zinn, S. A Continuous-Time Microsimulation and First Steps Towards a Multi-Level Approach in Demography, Disseration, Chapter 3, https://rosdok.uni-rostock.de/file/rosdok_derivate_0000004766/Dissertation_Zinn_2011.pdf

### Examples

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

# Clean workspace
rm(list=ls())

# Defining simulation horizon
startDate <- 20000101 # yyyymmdd
endDate   <- 21001231 # yyyymmdd
simHorizon <- c(startDate=startDate, endDate=endDate)

# 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
birthDates <- c("19301231","19990403","19561015","19911111","19650101")
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, but only illustrative example dealing with mortality, changes in
# fertily, and with the inheritance of attributes of the mother
#######################################################################################

# Clean workspace
rm(list=ls())

# Defining simulation horizon
startDate <- 20140101 # yyyymmdd
endDate   <- 20241231 # yyyymmdd
simHorizon <- c(startDate=startDate, endDate=endDate)

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

# Definition of maximal age
maxAge <- 100

# Defintion of nonabsorbing and absorbing states
sex <- c("m","f")
nat <- c("DE","AT","IT") # nationality
fert <- c("0","1")
stateSpace <- expand.grid(sex=sex,nat=nat,fert=fert)
absStates <- "dead"
```

```
# Definition of an initial population (for illustration purposes, create a random population)
N = 100
birthDates <- runif(N, min=getInDays(19500101), max=getInDays(20131231))
getRandInitState <- function(birthDate){
  age <- trunc((getInDays(simHorizon[1]) - birthDate)/365.25)
  s1 <- sample(sex,1)
  s2 <- sample(nat,1)
  s3 <- ifelse(age<=18, fert[1], sample(fert,1))
  initState <- paste(c(s1,s2,s3),collapse="/")
  return(initState)
}
initPop <- data.frame(ID=1:N, birthDate=birthDates, initState=sapply(birthDates, getRandInitState))
initPop$birthDate <- getInDateFormat(initPop$birthDate)

# Definition of initial states for newborns
# To have possibility to define distinct sex ratios for distinct nationalities,
# inherit related substate from the mother
fixInitStates <- 2 # give indices for attribute/substate that will be taken over
                   # from the mother, here: nat
varInitStates <- rbind(c("m","DE","0"), c("f","DE","0"),
                       c("m","AT","0"), c("f","AT","0"),
                       c("m","IT","0"), c("f","IT","0"))
initStatesProb <- c(0.515,0.485,
                    0.515,0.485,
                    0.515,0.485)
# Mind: depending on the inherited attribute nat="DE", nat="AT", or nat="IT"
# initials probabilites must sum to one

# Definition of (possible) transition rates
# Fertility rates (Hadwiger mixture model)
fertRates <- function(age, calTime, duration){
  b <- ifelse(calTime<=2020, 3.5, 3.0)
  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)
}
# 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)
}

fertTrMatrix <- cbind(c("f/DE/0->f/DE/1", "f/AT/0->f/AT/1", "f/IT/0->f/IT/1"),
                      c(rep("fertRates",3)))
allTransitions <- fertTrMatrix

absTransitions <- cbind(c("f/DE/dead", "f/AT/dead", "f/IT/dead",
                          "m/DE/dead", "m/AT/dead", "m/IT/dead"),
                       c(rep("mortRates",6)))
```

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

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

# Execute microsimulation
pop <- micSim(initPop=initPop,
               transitionMatrix=transitionMatrix, absStates=absStates,
               varInitStates=varInitStates, initStatesProb=initStatesProb,
               fixInitStates=fixInitStates,
               maxAge=maxAge, simHorizon=simHorizon,fertTr=fertTr)

#####################################################################################
# 3. 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
startDate <- 20140101 # yyyymmdd
endDate   <- 20241231 # yyyymmdd
simHorizon <- c(startDate=startDate, endDate=endDate)

# 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 month of enrollment to elementary school
monthSchoolEnrol <- 9

# Definition of an initial population (for illustration purposes, create a random population)
N = 100
birthDates <- runif(N, min=getInDays(19500101), max=getInDays(20131231))
getRandInitState <- function(birthDate){
  age <- trunc((getInDays(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))
initPop$birthDate <- getInDateFormat(initPop$birthDate)
range(initPop$birthDate)

# Definition of immigrants entering the population (for illustration purposes, create immigrants
# randomly)
M = 20
immigrDates <- runif(M, min=getInDays(20140101), max=getInDays(20241231))
immigrAges <- runif(M, min=15*365.25, max=70*365.25)
immigrBirthDates <- immigrDates - immigrAges
IDmig <- max(as.numeric(initPop[,"ID"]))+(1:M)
immigrPop <- data.frame(ID = IDmig, immigrDate = immigrDates, birthDate=immigrBirthDates,
                        immigrInitState=sapply(immigrBirthDates, getRandInitState))
immigrPop$birthDate <- getInDateFormat(immigrPop$birthDate)
immigrPop$immigrDate <- getInDateFormat(immigrPop$immigrDate)

# Definition of initial states for newborns
varInitStates <- 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
pop <- micSim(initPop=initPop, immigrPop=immigrPop,
              transitionMatrix=transitionMatrix,
              absStates=absStates,
              varInitStates=varInitStates,
              initStatesProb=initStatesProb,
              maxAge=maxAge,
              simHorizon=simHorizon,
              fertTr=fertTr,
              monthSchoolEnrol=monthSchoolEnrol)
```

---

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, initPopList = c(), immigrPopList = c(),
  transitionMatrix, absStates = NULL, varInitStates = c(), initStatesProb = c(),
  fixInitStates = c(), maxAge = 99, simHorizon, fertTr = c(),
  monthSchoolEnrol=c(), cores=1, seeds=1254)
```

## Arguments

initPop Either an initial population has to be given as a whole or splitted to be run at the distinct cores, see arguments initPopList. If it is given as a whole it is automatically splitted by MicSim such that the population parts run on the distinct cores are approx. equally sized.

immigrPop Optionally, a population of migrants entering the virtual population along simulation time can be given, see arguments immigrPopList. This migrants population can either be given as a whole or splitted to be run at the distinct cores. If it is given as a whole it is automatically splitted by MicSim such that the population parts run on the distinct cores are approx. equally sized.

`transitionMatrix`
: See micSim.

`absStates` See micSim.

`varInitStates` See micSim.

`initStatesProb` See micSim.

`initPopList` Optional: A list containing the initial population split for the distinct cores.

`immigrPopList` Optional: A list containing the immigration population split for the distinct cores.

`fixInitStates` See micSim.

`maxAge` See micSim.

`simHorizon` See micSim.

`fertTr` See micSim.

`monthSchoolEnrol`
: 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

```
# Clean workspace
rm(list=ls())

# Defining simulation horizon
startDate <- 20140101 # yyyymmdd
endDate   <- 20241231 # yyyymmdd
simHorizon <- c(startDate=startDate, endDate=endDate)

# 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 month of enrollment to elementary school
monthSchoolEnrol <- 9

# Definition of an initial population (for illustration purposes, create a random population)
N = 10000
birthDates <- runif(N, min=getInDays(19500101), max=getInDays(20131231))
getRandInitState <- function(birthDate){
  age <- trunc((getInDays(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))
initPop$birthDate <- getInDateFormat(initPop$birthDate)
range(initPop$birthDate)

# Definition of immigrants entering the population (for illustration purposes, create immigrants
# randomly)
M = 2000
immigrDates <- runif(M, min=getInDays(20140101), max=getInDays(20241231))
immigrAges <- runif(M, min=15*365.25, max=70*365.25)
immigrBirthDates <- immigrDates - immigrAges
IDmig <- max(as.numeric(initPop[,"ID"]))+(1:M)
immigrPop <- data.frame(ID = IDmig, immigrDate = immigrDates, birthDate=immigrBirthDates,
                        immigrInitState=sapply(immigrBirthDates, getRandInitState))
immigrPop$birthDate <- getInDateFormat(immigrPop$birthDate)
immigrPop$immigrDate <- getInDateFormat(immigrPop$immigrDate)

# Definition of initial states for newborns
varInitStates <- 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]

# Run microsimulation on cluster with three cores (settings depend on cluster used)
## Not run:
cores <- 3
seeds <- c(1233,1245,265)
initPopList <- list(initPop[1:5000,], initPop[5001:8000,],initPop[8001:nrow(initPop),])
immigrPopList <- list(immigrPop[1:1000,], immigrPop[1001:1500,],immigrPop[1501:nrow(immigrPop),])

pop <- micSimParallel(initPopList=initPopList, immigrPopList=immigrPopList,
  transitionMatrix=transitionMatrix, absStates=absStates, varInitStates=varInitStates,
  initStatesProb=initStatesProb, maxAge=maxAge, simHorizon=simHorizon,
  fertTr=fertTr, monthSchoolEnrol=monthSchoolEnrol,
  cores=cores, seeds=seeds)

## End(Not run)
```

---

`migrExpRates` *Transition rates for migration example of MicSim package.*

---

## Description

Transition rates for fertility (up to parity 4), migration between Spain / Netherlands / Sweden, mortality rates and emigration rates (leaving the Spain, the Netherlands, Sweden to some other country than these three). Rates have been estimated by the Unit "Migration, Demography and Governance Unit" of the European Commission.

## Usage

```
data("migrExpRates")
```

## Format

A data frame with transition rates for 30 states and ages from 0 to 99.

`mort_f_ES` mortality rates for females in Spain

`mort_f_NL` mortality rates for females in the Netherlands

`mort_f_SE` mortality rates for females in Sweden

`mort_m_ES` mortality rates for males in Spain

`mort_m_NL` mortality rates for males in the Netherlands

`mort_m_SE` mortality rates for males in Sweden

`fert_ES_0_1` fertility rates for partity 1 for Spain

`fert_ES_1_2` fertility rates for partity 2 for Spain

`fert_ES_2_3` fertility rates for partity 3 for Spain

`fert_ES_3_4` fertility rates for partity 4 for Spain

`fert_NL_0_1` fertility rates for partity 1 for the Netherlands

`fert_NL_1_2` fertility rates for partity 2 for the Netherlands

`fert_NL_2_3` fertility rates for partity 3 for the Netherlands

`fert_NL_3_4` fertility rates for partity 4 for the Netherlands

`fert_SE_0_1` fertility rates for partity 1 for Sweden

`fert_SE_1_2` fertility rates for partity 2 for Sweden

`fert_SE_2_3` fertility rates for partity 3 for Sweden

`fert_SE_3_4` fertility rates for partity 4 for Sweden

`rate_ES_NL` migration rates for Spain to the Netherlands

`rate_ES_SE` migration rates for Spain to the Sweden

`rate_NL_ES` migration rates for the Netherlands to Spain

`rate_NL_SE` migration rates for the Netherlands to Sweden

`rate_SE_ES` migration rates for Sweden to Spain

`rate_SE_NL` migration rates for Sweden to the Netherlands

`emig_f_ES` emigration rates for females in Spain

`emig_f_NL` emigration rates for females in the Netherlands

`emig_f_SE` emigration rates for females in Sweden

`emig_m_ES` emigration rates for females in Spain

`emig_m_NL` emigration rates for females in the Netherlands

`emig_m_SE` emigration rates for females in Sweden

## Source

European Commission

## Examples

```
data(migrExpRates)
```

# Index