Package 'mcwr'

October 13, 2022

Type Package

Title Markov Chains with Rewards

Version 1.0.0

Date 2021-03-08

Author Daniel Schneider [aut, cre],
Mikko Myrskyla [ctb],
Alyson van Raalte [ctb]

Maintainer Daniel Schneider <schneider@demogr.mpg.de>

Description In the context of multistate models, which are popular in sociology, demography, and epidemiology, Markov chain with rewards calculations can help to refine transition timings and so obtain more accurate estimates. The package code accommodates up to nine transient states and irregular age (time) intervals. Traditional demographic life tables result as a special case. Formulas and methods involved are explained in detail in the accompanying article: Schneider / Myrskyla / van Raalte (2021): Flexible Transition Timing in Discrete-Time Multistate Life Tables Using Markov Chains with Rewards, MPIDR Working Paper WP-2021-002.

License MIT + file LICENSE

LazyLoad no

LazyData true

Depends R (>= 3.6.0)

Encoding UTF-8

RoxygenNote 7.1.1

NeedsCompilation no

Repository CRAN

Date/Publication 2021-03-10 19:40:05 UTC

R topics documented:

`mcwr` *mcwr: Markov chain with rewards calculations*

Description

mcwr implements Markov chain with rewards calculations. It accompanies the article:

Schneider / Myrskyla / van Raalte (2021):
Flexible Transition Timing in Discrete-Time Multistate Life Tables
Using Markov Chains with Rewards. MPIDR Working Paper WP-2021-002.
Available here.

In the `mcwr` help entries, this article will be referred to as "the paper" and its appendix as "the appendix".

Overview

The main function, which does the actual calculations, is `mcwr_expectancies`. The other functions either check the data for consistency or carry out data management tasks.

Running mcwr demands a very specific data frame (called mcwr data frame, laid out in section 'Data setup' below). Several data management functions aide in creating such a specific data frame.

The main data management function is `mcwr_genvars`, which generates p- and r- variables that are missing from the data frame. These variables specify transition probabilities and rewards. Other data management functions check the data frame for consistency (`mcwr_check`), transform variable names (`mcwr_switch`), or check or create an exit row (`mcwr_exit`).

Abbreviations and definitions used in this help entry

MCWR Markov chain with rewards

p-variables Variables whose name is of the format 'p##'. They contain transition probability data. The numbers encode from and target state. The default for this number pair is the ji-format (see below). For example, variable 'p12' has transition probabilities from-state 1 to state 2.

r-variables Variables whose name is of the format 'r#_##'. They contain rewards data. The first number encodes the state that receives the reward. The second and third numbers encode from and target state. The default for this number pair is the ji-format (see below).

ji/ij-format Variable (names) are said to be in ji-format if the first number of the variable name specifies the from-state and the second number specifies the target state. The index j always denotes the from-state. Consequently, variable (names) in ij-format state the target state first and then the from-state.

from/target/initial states From-state and target state are the two states that are connected by a state transition. The initial state is the state at baseline age.

mcwr data frame A data frame that fulfills all requirements for mcwr to run on it without error. The structure of the data frame is set forth in section Data setup.

Data setup

mcwr requires the transition data to be in a very specific format, called 'mcwr data frame' in this help entry. To give you a quick idea of what is required, execute:

```
data <- mcwr_exampledata(1)
head(data)
tail(data)
```

In general, the following rules and conventions apply:

- Transition probabilities are specified in p-variables. The default convention for specifying from and target states is the ji-format: The first number encodes the from-state, the second number the target state.
- Rewards are specified in r-variables. The numbers occuring in the variable names specify the rewarded state, the from-state, and the target state, respectively.
- A maximum number of 9 states (including the absorbing state) is allowed. States must be encoded using numbers 1-9. 0 is not allowed.
- Only a single absorbing state is allowed. It is encoded by the highest number occuring for all states. In the example, 5 is the aborbing state.
- States can be non-contiguous. In the example, states 2 and 3 are missing from the model. The states of the model are 1, 4, 5.
- Transition probabilities must sum to 1. For example, columns p11, p14, p15 sum to 1.
- As long as the sums-to-unity condition is satisfied, not all p-variables must be present. In the example, variable p41 is missing, and all-zero by implication.
- Age (or, more general, time) is specified in a variable called 'age'.
- Irregular age intervals are allowed. The Examples section illustrates this with a life table on a 'demographic' 5-year age grid (childhood age intervals are shorter).
- The first row of the data frame specifies the baseline age. All p- and r-variables must have missing values in the first row.
- The last row of the data frame corresponds to the exit age. At this age, all subjects are required to die (enter the absorbing state). In the example, variables p15 and p45 are set to 1 at age 111. All other transitions at the exit age must be set to missing.
- In general, whenever values of a mcwr data frame are certain to not enter matrix calculations, they must be set to missing in the data frame. Conversely, data points that are certain to enter matrix calculations must never be missing.
- The first transition takes place in row 2 (age 51 in the example). It is important to keep in mind that age specifies a point in time, not an interval. It is the point in time when the subject turns 51.

- At this point, rewards can be distributed for the previous age (interval). Standard Markov chain calculations would distribute occupancy times end-of-period (the transition takes place at exact age 51). By contrast, in the example we distribute time rewards and assume mid-period transitions, i.e. we assume that state transitions take place at ages 50.5, 51.5, etc. The reward for state 1 of the 1->4 transition at age 51 is specified to be 0.5. This covers the period [50 50.5). The reward for state 4 of the 1->4 transition at age 51 is also specified to be 0.5. This covers the period [50.5 51). From the same logic, staying in the same state carries a reward of 1. It is made up of rewards of 0.5 for each one of the periods [50 50.5) and [50.5 51), respectively.
- Rewards can only flow to from or target states. For example, a variable 'r3_12' in the data frame will be flagged as an error.
- Rewards can only flow to transient states. In the example, where the absorbing state is 5, a variable 'r5_45' will generate an error.
- Rewards can take on any numeric value, including negative ones.
- You may have additional variables (variables other than p- and r-variables) in the data frame. They will be ignored.

As a minor remark on the example data frame, it is derived from the SHARE application in the paper. In that application, state 1 is left with certainty when turning 71, with no return possible. The values shown in the data frame for p14 and p15 for ages 71 and higher are therefore immaterial, given that this state is never occupied at these ages.

Limitations

At the risk of redundancy, here are the limitations that **mcwr** places on the model setup:

- The maximum number of states is 9 (including the absorbing state).
- Only a single absorbing state is allowed.
- Rewards can only flow to from or target states.

Object design

We implemented **mcwr** using plain data frames. The input argument to all package functions is a data frame, as opposed to a specialized S3 object. We did so on purpose, since an S3 object here provides only moderate gains and yet complicates things in some respects. This design decision may change in the future, so the current `mcwr_*()` function names may change (e.g., `mcwr_check()` will turn into `check()`). However, the `mcwr_*()` functions are guaranteed to continue to work in the future in order to ensure backward compatibility.

Life tables and open age intervals

The last age group of life tables is frequently an open interval. What does this mean for mcwr data frames? In the present context, the important thing to recognize is that mcwr expectancies must have access to the proper ax value (Chiangs a). The best way to achieve this is to set exit age (certain transition into death) to some number greater than the previous age. It does not really matter which age you choose as long as you assign the ax value to the correct rewards variable. The Examples section below illustrates this.

Examples

```
## The first example uses a subset of the data from the retirement
## example in the paper.  The example data frame
## only contains transition variables.  We verify that the data frame
## is suitable for mcwr and gather some model information.

ex_dat <- mcwr_exampledata(1)
ex_modelinfo <- mcwr_check(ex_dat)
names(ex_modelinfo)
ex_modelinfo$s_frm
ex_modelinfo$s_abs
ex_modelinfo$numages

## modelinfo$s_frm tells us that the from-states are 1 2 4,
## and modelinfo$s_abs indicates 5 to be the absorbing state.
## There are 61 age classes in the model, which we can see
## from modelinfo$numages.  Given initial proportions, we
## can immediately calculate expectancies with different timing assumptions:

ex_init <- c(0.95, 0.04, 0.01)
names(ex_init) <- c(1,2,4)
mcwr_expectancies(ex_dat, timing='mid', initprop=ex_init)[[1]]
mcwr_expectancies(ex_dat, timing='eop', initprop=ex_init)[[1]]

## Since we have a regular age grid of 1, the corresponding
## magnitudes in the 'end-of-period' specification
## are higher by 0.5. The output also tells us that total
## life expectancy does not depend on the initial
## state and that some occupation times are zero.
## This comes from the particular data restrictions and
## assumptions of the application in the paper:
## Mortality does not differ across states, and transitions
## from state 4 (retirement) to states 1 (working) or
## 2 (unemployed) never occur (by assumption). See the
## paper for more details.
##
## Since we are talking about the meaning of the state encodings:
## Let's label the numeric values with meaningful descriptions.
## This can be done via the vllmap option:

ex_MCWR <- c(1,2,3,4,5)
names(ex_MCWR) <- c('work', 'unem', 'oolf', 'retr', 'dead')
mcwr_expectancies(ex_dat, timing='eop', initprop=ex_init, vllmap=ex_MCWR)[[1]]

## Let's now assume that retirement occurs on average
## 3 months into the retirement year. The easiest way to
## specify this is to generate all rewards variables first
## as mid period and then edit them as needed:

ex_dat <- mcwr_genvars(ex_dat, timing='mid', order=TRUE)
names(ex_dat)
```

```
ex_dat[2:(nrow(ex_dat)-1),'r1_14'] <- 0.25
ex_dat[2:(nrow(ex_dat)-1),'r4_14'] <- 0.75
ex_dat[2:(nrow(ex_dat)-1),'r2_24'] <- 0.25
ex_dat[2:(nrow(ex_dat)-1),'r4_24'] <- 0.75

## We do not have to worry about transitions out
## of retirement since their probability is zero.
##
## At this point it is useful to look at the data. In particular,
## take note of how the first and last few rows are structured.

head(ex_dat)
tail(ex_dat)

## We recalculate:

mcwr_expectancies(ex_dat, initprop=ex_init, vllmap=ex_MCWR)[[1]]

## As an alternative to the above data replacement statements,
## we could have solely used function mcwr_genvars() in
## conjunction with the timing option of mcwr_expectancies().
## This has the advantage of not having to worry
## about getting the baseline and exit age rows right
## (the row-indexing part of the statements).

ex_dat[,grep('r._..', names(ex_dat))] <- NULL
ex_dat <- mcwr_genvars(ex_dat, timing=0.25, order=TRUE)
ex_getn <- function(pat) grep(pat, names(ex_dat), value=TRUE)
ex_keepvars <- c('age', ex_getn('p..'), ex_getn('r._14'), ex_getn('r._24'))
ex_dat <- ex_dat[,ex_keepvars]
mcwr_expectancies(ex_dat, initprop=ex_init, vllmap=ex_MCWR, timing='mid', add=TRUE)[[1]]

## After the creation of the 'timing(0.25)' rewards variables,
## we could have generated the remaining rewards variables explicitly:

ex_dat <- mcwr_genvars(ex_dat, timing='mid', add=TRUE, order=TRUE)
mcwr_expectancies(ex_dat, initprop=ex_init, vllmap=ex_MCWR)[[1]]

## Next, we illustrate the equivalence to standard life table
## calculations. We first look at a regularly spaced 1-year life table.

ex_dat <- mcwr_exampledata(2)
names(ex_dat)

## Checking whether we have a proper mcwr data frame reveals problems:

junk <- try(ex_modelinfo <- mcwr_check(ex_dat)) # generates error

## We look at the beginning and ending rows of the data frame:

head(ex_dat)
tail(ex_dat)
```

```
## We see that a number of mcwr data requirements, in particular
## those concerning the first and last row of the data
## frame, are not met. The data in proper mcwr format look like this:

ex_dat <- mcwr_exampledata(6)
head(ex_dat)
tail(ex_dat)

## The correct life table value for e0 is 80.16.
## Mid-period mcwr calculations yield

mcwr_expectancies(ex_dat, timing='mid')[[1]]

## This ignores the values in the ax variable, which are different
## from 0.5 for the first and last age. We take them
## into account be generating an r-variable with corresponding values.

ex_dat[,'r1_12'] <- ex_dat[,'ax']
mcwr_expectancies(ex_dat, timing='mid', add=TRUE)[[1]]

## The result is a little more accurate than the previous calculation.
##
## Let's look at the corresponding 5-year life table.

ex_dat <- mcwr_exampledata(3)
junk <- try(modelinfo <- mcwr_check(ex_dat)) # generates error
head(ex_dat)
tail(ex_dat)

## The data set must again be first transformed to a valid mcwr data set.
## It then looks like:

ex_dat <- mcwr_exampledata(7)
head(ex_dat)
tail(ex_dat)

## We again perform calculations with and without taking ax into account:

mcwr_expectancies(ex_dat, timing='mid')[[1]]

ex_dat[,'r1_12'] <- ex_dat[,'ax']
ex_dat <- mcwr_genvars(ex_dat, timing='mid', add=TRUE, order=TRUE)
ex_dat <- mcwr_expectancies(ex_dat)[[1]]

ex_dat

## We explicitly created all rewards variables using mcwr_genvars()
## instead of using them implicitly in mcwr_expectancies().
## We did this to illustrate how mcwr uses rewards to
## handle irregularly spaced age intervals.
```

`mcwr_check` *Check mcwr data frame*

Description

Check that mcwr data frame is set up correctly

Usage

```
mcwr_check(data, ij = FALSE, skipexit = FALSE, noexit = FALSE)
```

Arguments

`data`	an mcwr data.frame
`ij`	a boolean TRUE/FALSE. If TRUE, states that data set is in ij-format.
`skipexit`	a boolean TRUE/FALSE. If TRUE, does not check the last row of the data frame.
`noexit`	a boolean TRUE/FALSE. If TRUE, states that the exit row is missing from the data frame.

Details

Run mcwr check to check whether your data frame is a valid mcwr data frame or whether you have to modify something. It is also run internally by the other mcwr functions, so you are always safe not to be using incorrectly set up data.

A second use of the functions is to gather comprehensive information about existing and missing model variables, and more.

Value

a list containing model information. Its named elements are:

p_exi	character vector: p-variables in the data set
p_ful	character vector: full set of p-variables implied by states
p_new	character vector: p-variables that are implied by states but not in the data set
s_trn	numeric vector: list of transitions occuring in data set
s_frm	numeric vector: list of from-states
s_trg	numeric vector: list of target states
s_abs	numeric vector: absorbing state
s_omt	numeric vector: states omitted from the model
r_exi	character vector: r-variables in the data set
r_ful	character vector: full set of p-variables implied by states
r_new	character vector: r-variables that are implied by states but not in the data set
r_trn	character vector: list of transitions covered by existing r-variables
s_rcv	numeric vector: states receiving rewards
s_nrc	numeric vector: states not receiving rewards
numages	number of age classes in the model

agelist	numeric vector: age classes of the model
ageintervals	numeric vector: list of lengths of age intervals
hasexit	TRUE/FALSE: whether data has an exit row

See Also

Other mcwr: `mcwr_exit()`, `mcwr_expectancies()`, `mcwr_genvars()`, `mcwr_switch()`

`mcwr_exampledata` *Load mcwr example data*

Description

This function is used in examples sections and loads mcwr example data. It only exists for documentation purposes.

Usage

```
mcwr_exampledata(snipnum)
```

Arguments

`snipnum`	a integer [1-9]. Determines the data frame returned.

Value

A data.frame, suitable for further processing by `mcwr_*()` functions.

`mcwr_exit` *Examine last (exit) row of data set*

Description

`mcwr_exit()` allows easy editing and consistency checks of the last (exit) row of an mcwr data frame.

Usage

```
mcwr_exit(data, age, replace = FALSE, update = FALSE, rewards = 0)
```

Arguments

`data`	an mcwr data.frame
`age`	a real number *ageval*. It specifies the age of the exit row in the data frame. It may or may not exist. It may not be smaller than the largest age in the data frame. If *ageval* corresponds to the largest age in the data frame, option `replace` must be specified. The values of the corresponding row are replaced. Exit transition values for p-variables are set to 1. Exit transition values for r-variables are left as-is if they are non-missing and option `update` is not used. Otherwise they are set to *rwval*. Values of all other transitions are set to missing. If ageval is larger than the largest age in the data frame, a new row will be inserted. Exit transitions are set to 1 for p-variables and to rwval for r-variables. Values of all other transitions are set to missing.
`replace`	boolean TRUE/FALSE. See option `age`.
`update`	boolean TRUE/FALSE. See option `age`.
`rewards`	a real number *rwval*. It determines the rewards value for exit transitions.

Details

mcwr requires that all data points that do not enter matrix calculations be set to missing in order to avoid incorrectly set up data. This rule makes the last (exit) row of the mcwr data frame somewhat tedious to manage. The convenience function `mcwr_exit()` makes it easier to create or edit the last (exit) row of the data set.

Value

A data.frame, suitable for further processing by `mcwr_*()` functions.

See Also

Other mcwr: `mcwr_check()`, `mcwr_expectancies()`, `mcwr_genvars()`, `mcwr_switch()`

`mcwr_expectancies` *Calculate expectancies*

Description

Calculate state and overall expectancies. This is the function that does the actual calculations.

Usage

```
mcwr_expectancies(data, initprop = NULL, vllmap = NULL, ...)
```

Arguments

data	an mcwr data.frame
initprop	a numeric vector. It supplies information about the initial state fractions at baseline age. Its elements must be in the interval [0 1] and sum to 1. It must have element names corresponding to all from-states in the model, in ascending order. For example, If your model contains from-states 1, 2, and 7, initspec must be a 3-element numeric vector, specifying the initial proportion of each state in turn. Its names attribute must consist of the sequential from-states ('1 2 7' in the above example).
vllmap	a numeric vector. Its elements must have the from-states of the model as a subset. If elements of the vector have names, they are used in labelling output.
...	options timing, add, and replace. See mcwr_genvars. In most cases, you do not have to create a full set of r-variables using function mcwr_genvars(). r-variables that correspond to timings that can be accommodated by the timing option can be created automatically, behind the scenes. You do so by specifying the timing option in function expectancies instead in the function genvars. Any missing r-variables (and p-variables, for that matter) will then be created behind the scenes before calculations are done. They will get deleted before the function concludes. r-variables with more complicated timings have to be created explicitly before running function mcwr_expectancies().

Value

a list of matrices that have been involved in the calculations. Matrix names are e, P, F, R#. Matrix e contains the overall results.

See Also

Other mcwr: mcwr_check(), mcwr_exit(), mcwr_genvars(), mcwr_switch()

---

mcwr_genvars *Generate mcwr variables*

Description

Generate (additional) transition probability or rewards variables

Usage

```
mcwr_genvars(
  data,
  timing = "",
  add = FALSE,
```

```
  replace = FALSE,
  nop = FALSE,
  nor = FALSE,
  order = FALSE
)
```

Arguments

data	an mcwr data.frame
timing	a character or numeric scalar. It specifies how rewards are distributed to from and target states. It is required if option nor is not used. timespec can be one of 'bop', 'mid', and 'eop', which stands for 'beginning-of-period', 'mid-period', and 'end-of period', respectively. Alternatively, it can also be a number in the interval [0 1] that specifies the fraction of the interval that goes to the from-state. Values of 0, 0.5, and 1 correspond to 'beginning-of-period', 'mid-period', and 'end-of-period', respectively.
add	a boolean TRUE/FALSE. If TRUE, existing r-variables are left unchanged.
replace	add a boolean TRUE/FALSE. If TRUE, existing r-variables are replaced.
nop	a boolean TRUE/FALSE. If TRUE, not generate any p-variables. By default, all missing p-variables are generated. Since existing p-variables must satisfy the sums-to-unity condition, only p-variables that are (by implication) all-zero can be missing. mcwr_expectancies() will run whether such redundant variables exist or not.
nor	a boolean TRUE/FALSE. If TRUE, does not generate any r-variables.
order	a boolean TRUE/FALSE. If TRUE, orders variables alphabetically according to the column list 'age p* r*'.

Details

The main purpose of this function is to generate rewards variables (r-variables). It examines existing p- and r-variables, determines the implied full set of states, and generates any missing variables that are missing from the data frame. It interacts flexibly with existing r-variables: You can leave them unchanged or have them replaced.

An effective way to create r-variables may be to generate a full set of r-variables using mcwr_genvars() and then edit them where necessary. This is illustrated under in the examples section of mcwr.

Value

A data.frame, suitable for further processing by mcwr_*() functions.

See Also

Other mcwr: mcwr_check(), mcwr_exit(), mcwr_expectancies(), mcwr_switch()

`mcwr_switch` *Switch from-state and target state*

Description

Change from-state and target state index convention of variable names

Usage

```
mcwr_switch(data, ji = as.logical(NA), verbose = FALSE)
```

Arguments

`data`	an mcwr data.frame
`ji`	boolean TRUE/FALSE. If TRUE, the data frame will always be in ji-format after function conclusion. That is, if the data are in ij-format they will be converted to ji-format, and will be left untouched otherwise.
`verbose`	boolan TRUE/FALSE. If TRUE, the function will display verbose error messages. This is useful if the function tells you that it can neither find a consistent ji data frame nor a consistent ij data frame. The error messages under option `verbose` may give you a clue about the source of the error.

Details

The appendix notation follows the ij-notation, where the first index refers to the target state and the second index to the from-state. This has the advantage of conforming with the conventions of matrix algebra. p- and r-variables of mcwr data frames, however, generally follow the ji-convention. The advantage of this is that sorting the variables alphabetically results in a sensible and intuitive ordering. Therefore, your data frame variables are required to follow the ji-convention. The convenience function `mcwr_switch()` allows you to switch between the two conventions. If you have a consistent data frame in ij-format, running this function will rename variables according to the ji-convention. Your data must be in ji-format before you can run any other of the mcwr functions. As a brief example, we load data in ji-format, then switch to ij-format and back:

Value

A data.frame, suitable for further processing by `mcwr_*()` functions.

See Also

Other mcwr: `mcwr_check()`, `mcwr_exit()`, `mcwr_expectancies()`, `mcwr_genvars()`

Examples

```
data <- mcwr_exampledata(1)
head(data)
data <- mcwr_switch(data)
head(data)
data <- mcwr_switch(data)
head(data)
```

Index