Package ‘microsimulation’

April 4, 2022

Type Package

Title Discrete Event Simulation in R and C++, with Tools for Cost-Effectiveness Analysis

Version 1.3.7

Date 2022-04-04

Description Discrete event simulation using both R and C++ (Karls-

son et al 2016; <doi:10.1109/eScience.2016.7870915>). The C++ code is adapted from the SSIM li-

brary <https://www.inf.usi.ch/carzaniga/ssim/>, allowing for both event-

oriented and process-oriented simulation. The code includes a SummaryReport class for report-

ing events and costs by age and other covariates. The C++ code is available as a static li-

brary for linking to other packages. A priority queue implementation is given in C++ to-

gether with an S3 closure and a reference class implementation. Finally, some tools are pro-

vided for cost-effectiveness analysis.

License GPL (>= 3)

Depends Rcpp (>= 0.10.2), methods

Imports parallel, grDevices, ascii

Suggests testthat

LinkingTo Rcpp, BH

LazyData true

URL https://github.com/mclements/microsimulation

BugReports https://github.com/mclements/microsimulation/issues

Encoding UTF-8

RoxygenNote 7.1.1

NeedsCompilation yes

Author Mark Clements [aut, cre, cph],

Alexandra Jauhiainen [aut],

Andreas Karlsson [aut],

Antonio Carzaniga [cph],

University of Colorado [cph],

Pierre L'Ecuyer [cph]
Maintainer  Mark Clements <mark.clements@ki.se>
Repository CRAN
Date/Publication  2022-04-04 15:40:11 UTC

R topics documented:

- microsimulation-package .................................................. 2
- .microsimulationLdFlags .................................................. 3
- callCalibrationPerson ..................................................... 5
- discountedInterval ......................................................... 6
- fhcrcData ................................................................. 7
- pqueue ................................................................. 7
- pqueue__new ........................................................... 13
- RNGStream .............................................................. 14
- summary.SummaryReport .................................................. 15

Index 18

microsimulation-package  microsimulation

Description

Discrete event simulations in both R and C++ with Tools for Cost-Effectiveness Analysis.

Introduction

Discrete event simulations in both R and C++ with Tools for Cost-Effectiveness Analysis.

Author(s)

Mark Clements <mark.clements@ki.se>

References

https://github.com/mclements/microsimulation

See Also

Rcpp
Description

Is this function needed? We could define the current stream in open code.
Again, is this needed?

Usage

.microsimulationLdFlags()
inlineCxxPlugin(...)
LdFlags()
microsimulation.init(PACKAGE = "microsimulation")
microsimulation.exit(PACKAGE = "microsimulation")
unsigned(seed)
signed(seed)
rnormPos(n, mean = 0, sd = 1, lbound = 0)
set.user.Random.seed(seed, PACKAGE = "microsimulation")
advance.substream(seed, n, PACKAGE = "microsimulation")
next.user.Random.substream(PACKAGE = "microsimulation")
user.Random.seed(PACKAGE = "microsimulation")
enum(obj, labels, start = 0)
enum(obj) <- value
RNGstate()
frontier(x, y, concave = TRUE, convex = NULL)
lines_frontier(x, y, pch = 19, type = "b", ...)
discountedPoint(y, time, dr)
ICER(object1, object2, ...) 
.onLoad(lib, pkg) 
.onUnload(libpath) 

Arguments
... other arguments 
PACKAGE package for the seed 
seed random number seed 
n number of sub-streams to advance 
mean numeric for the mean of the (untruncated) normal distribution (default=0) 
sd numeric for the sd of the (untruncated) normal distribution (default=1) 
lbound numeric for the lower bound (default=0) 
obj integer or logical for factor levels 
labels labels for the factor levels 
start first value of the levels 
value labels for the factor levels 
x vector of x coordinates 
y the undiscounted value 
concave logical for whether to calculate a concave frontier (default=TRUE) 
convex logical for whether to calculate a convex frontier (default=NULL) 
pch type of pch for the plotted symbols (default=19) 
type join type (default="b") 
time the time of the event 
dr discount rate, expressed as a percentage 
object1 first object 
object2 second object 
lib library string 
pkg package string 
libpath library path string 

Value
No return value, called for side effects 
No return value, called for side effects 
No return value, called for side effects 
unsigned seed 
signed seed
callCalibrationPerson

numeric vector
invisibly returns the new seed
the advanced seed
invisibly returns TRUE – called for side effect
random seed
the new factor
update the factor
a list with oldseed (the old value of .Random.seed), and reset(), which resets .Random.seed
a list with components x and y for the frontier
No return value, called for side effect
umeric vector

callCalibrationPerson  call CalibrationPerson example

Description

Example that uses the RngStream random number generator
Example that uses the Mersenne-Twister random number generator
Example that uses the Mersenne-Twister random number generator
Example that uses the Mersenne-Twister random number generator

Usage

callCalibrationPerson(
  seed = 12345,
  n = 500,
  runpar = c(4, 0.5, 0.05, 10, 3, 0.5),
  mc.cores = 1
)
callPersonSimulation(n = 20, seed = rep(12345, 6))
callSimplePerson(n = 10)
callSimplePerson2(n = 10)
callIllnessDeath(n = 10L, cure = 0.1, zsd = 0)
**Arguments**

- **seed**: random number seed
- **n**: number of simulations (default=10)
- **runpar**: parameters
- **mc.cores**: number of cores
- **cure**: probability of cure
- **zsd**: frailty standard deviation

**Value**

data-frame

data-frame

data-frame

data-frame

data-frame

data-frame

discountedInterval **Integrate a discounted value**

**Description**

Integrate a discounted value

**Usage**

discountedInterval(\(y, \text{start}, \text{finish}, \text{dr}\))

**Arguments**

- **y**: the undiscounted value
- **start**: the start time
- **finish**: the finish time
- **dr**: discount rate, expressed as a percentage

**Value**

numeric discounted value
fhcrcData

*Old data used in the prostatas model*

**Description**

Old data used in the prostatas model

**Usage**

fhcrcData

**Format**

An object of class list of length 10.

pqueue

*S3 priority queue implementation using C++*

**Description**

This provides a priority queue that is sorted by the priority and entry order. The priority is assumed to be numeric. The events can be of any type. As an extension, events can be cancelled if they satisfy a certain predicate. Note that the inactive events are not removed, rather they are marked as cancelled and will not be available to be popped.

Based on C++ code. See also the S3 implementation pqueue.

This event queue is simple and useful for pedagogic purposes.

Inherit from this class to represent a discrete event simulation. The API is similar to that for Omnet++, where an init method sets up the initial events using the scheduleAt(time,event) method, the messages are handled using the handleMessage(event) method, the simulation is run using the run method, and the final method is called at the end of the simulation.

**Usage**

pqueue(lower = TRUE)

**Arguments**

- **lower**
  
  boolean to determine whether to give priority to lower values (default=TRUE) or higher values

**Details**

The algorithm for pushing values into the queue is computationally very simple: simply rank the times using order() and re-order times and events. This approach is probably of acceptable performance for smaller queue. A more computationally efficient approach for pushing into larger queues would be to use a binary search (e.g. using findInterval()).

For faster alternatives, see pqueue and PQueueRef.
Value

a list with

push function with arguments priority (numeric) and event (SEXP). Pushes an event with a given priority

pop function to return a list with a priority (numeric) and an event (SEXP). This pops the first active event.

cancel function that takes a predicate (or R function) for a given event and returns a logical that indicates whether to cancel that event or not. This may cancel some events that will no longer be popped.

eempty function that returns whether the priority queue is empty (or has no active events).

clear function to clear the priority queue.

ptr XPtr value

Fields

ptr External pointer to the C++ class
times vector of times
events list of events
times vector of times
events list of events

Methods

cancel(predicate) Method to cancel events that satisfy some predicate
clear() Method to clear the event queue
eempty() Method to check whether there are no events in the queue
initialize(lower = TRUE) Method to initialize the object. lower argument indicates whether lowest priority or highest priority
pop() Method to remove the head of the event queue and return its value
push(priority, event) Method to push an event with a given priority
cancel(predicate, ...) Method to remove events that satisfy some predicate
clear() Method to clear the event queue
eempty() Method to check whether there are no events in the queue
pop() Method to remove the head of the event queue and return its value
push(time, event) Method to insert the event at the given time
final() Method for finalising the simulation
handleMessage(event) Virtual method to handle the messages as they arrive
init() Virtual method to initialise the event queue and attributes
reset(startTime = 0) Method to reset the event queue
run(startTime = 0) Method to run the simulation
scheduleAt(time, event) Method that adds attributes for the event time and the sendingTime, and then insert the event into the event queue
Examples

```r
pq = pqueue()
pq$push(3, "Clear drains")
pq$push(4, "Feed cat")
pq$push(5, "Make tea")
pq$push(1, "Solve RC tasks")
pq$push(2, "Tax return")
while(!pq$empty())
  print(pq$pop())

pq = new("PQueueRef")
pq$push(3, "Clear drains")
pq$push(4, "Feed cat")
pq$push(5, "Make tea")
pq$push(1, "Solve RC tasks")
pq$push(2, "Tax return")
while(!pq$empty())
  print(pq$pop())

pq = new("EventQueue")
pq$push(3, "Clear drains")
pq$push(4, "Feed cat")
pq$push(5, "Make tea")
pq$push(1, "Solve RC tasks")
pq$push(2, "Tax return")
while(!pq$empty())
  print(pq$pop())

DES = setRefClass("DES",
  contains = "BaseDiscreteEventSimulation",
  methods=list(
    init=function() {
      scheduleAt(3, "Clear drains")
      scheduleAt(4, "Feed cat")
      scheduleAt(5, "Make tea")
      scheduleAt(1, "Solve RC tasks")
      scheduleAt(2, "Tax return")
    },
    handleMessage=function(event) print(event))
)

des = new("DES")
des$run()
```

```r
## Not run:
testRsimulation1 <- function() {
  ## A simple example
  Simulation <-
    setRefClass("Simulation",
      contains = "BaseDiscreteEventSimulation")
  Simulation$methods(
    init = function() {
      scheduleAt(rweibull(1,8,85), "Death due to other causes")
      scheduleAt(rweibull(1,3,90), "Cancer diagnosis")
    }
  )
}
```
### An extension with individual life histories

testRsimulation2 <- function(n=100) {
  Simulation <-
    setRefClass("Simulation",
      contains = "BaseDiscreteEventSimulation",
      fields = list(state = "character", report = "data.frame"))
  Simulation$methods(
    init = function() {
      report <<- data.frame()
      state <<- "Healthy"
      scheduleAt(rweibull(1,8,85), "Death due to other causes")
      scheduleAt(rweibull(1,3,90), "Cancer diagnosis")
    },
    handleMessage = function(event) {
      report <<- rbind(report, data.frame(state = state,
                                           begin = attr(event,"sendingTime"),
                                           end = currentTime,
                                           event = event,
                                           stringsAsFactors = FALSE))
      if (event %in% c("Death due to other causes", "Cancer death")) {
        clear()
      }
      else if (event == "Cancer diagnosis") {
        state <<- "Cancer"
        if (runif(1) < 0.5)
          scheduleAt(now() + rweibull(1,2,10), "Cancer death")
        print(event)
      }},
    final = function() report)
  sim <- Simulation$new()
  do.call("rbind", lapply(1:n, function(id) data.frame(id=id,sim$run())))
}

### reversible illness-death model

testRsimulation3 <- function(n=100) {
  Simulation <-
    setRefClass("Simulation",
      contains = "BaseDiscreteEventSimulation",
      fields = list(state = "character", report = "data.frame"),
      methods = list(
        init = function() {
          report <<- data.frame()
          state <<- "Healthy"
          scheduleAt(rweibull(1,8,85), "Death due to other causes")
          scheduleAt(rweibull(1,3,90), "Cancer diagnosis")
        },
        handleMessage = function(event) {
          report <<- rbind(report, data.frame(state = state,
                                               begin = attr(event,"sendingTime"),
                                               end = currentTime,
                                               event = event,
                                               stringsAsFactors = FALSE))
          if (event %in% c("Death due to other causes", "Cancer death")) {
            clear()
          }
          else if (event == "Cancer diagnosis") {
            state <<- "Cancer"
            if (runif(1) < 0.5)
              scheduleAt(now() + rweibull(1,2,10), "Cancer death")
          }},
        final = function() report)
    Simulation$new()
    do.call("rbind", lapply(1:n, function(id) data.frame(id=id,sim$run())))

}
```r
fields = list(state = "character", everCancer = "logical", report = "data.frame")
Simulation$methods(
  init = function() {
    report <<- data.frame()
    state <<- "Healthy"
    everCancer <<- FALSE
    scheduleAt(rweibull(1,8,85), "Death due to other causes")
    scheduleAt(rweibull(1,3,90), "Cancer diagnosis")
  },
  handleMessage = function(event) {
    report <<- rbind(report, data.frame(state = state,
                                            everCancer = everCancer,
                                            begin = attr(event,"sendingTime"),
                                            end = currentTime,
                                            event = event,
                                            stringsAsFactors = FALSE))
    if (event %in% c("Death due to other causes", "Cancer death")) {
      clear()
    } else if (event == "Cancer diagnosis") {
      state <<- "Cancer"
      everCancer <<- TRUE
      if (runif(1) < 0.5)
        scheduleAt(now() + rweibull(1,2,10), "Cancer death")
      scheduleAt(now() + 10, "Recovery")
    } else if (event == "Recovery") {
      state <<- "Healthy"
      scheduleAt(now() + rexp(1,10), "Cancer diagnosis")
    }
  },
  final = function() report)
sim <- Simulation$new()
do.call("rbind", lapply(1:n, function(id) data.frame(id=id,sim$run())))
```

## cancer screening

testRsimulation4 <- function(n=1) {
  Simulation <-
    setRefClass("Simulation",
      contains = "BaseDiscreteEventSimulation",
      fields = list(state = "character", report = "data.frame"))
Simulation$methods(
  init = function() {
    report <<- data.frame()
    state <<- "Healthy"
    scheduleAt(rweibull(1,8,85), "Death due to other causes")
    scheduleAt(rweibull(1,3,90), "Cancer onset")
    scheduleAt(50,"Screening")
  },
  handleMessage = function(event) {
    report <<- rbind(report, data.frame(state = state,
```
begin = attr(event, "sendingTime"),
end = currentTime,
event = event,
stringsAsFactors = FALSE))

if (event %in% c("Death due to other causes", "Cancer death")) {
  clear()
}
else if (event == "Cancer onset") {
  state <<- event
  dx <- now() + rweibull(1,2,10)
  scheduleAt(dx, "Clinical cancer diagnosis")
  scheduleAt(dx + rweibull(1,1,10), "Cancer death")
  scheduleAt(now() + rweibull(1,1,10), "Metastatic cancer")
}
else if (event == "Metastatic cancer") {
  state <<- event
  cancel(function(event) event %in%
    c("Clinical cancer diagnosis","Cancer death")) # competing events
  scheduleAt(now() + rweibull(1,2,5), "Cancer death")
}
else if (event == "Clinical cancer diagnosis") {
  state <<- event
  cancel(function(event) event == "Metastatic cancer")
}
else if (event == "Screening") {
  switch(state,
    "Cancer onset" = {
      state <<- "Screen-detected cancer diagnosis"
      cancel(function(event) event %in%
        c("Clinical cancer diagnosis","Metastatic cancer"))
    },
    "Metastatic cancer" = {}, # ignore
    "Clinical cancer diagnosis" = {}, # ignore
    "Healthy" = {
      if (now()<=68) scheduleAt(now()+2, "Screening")
    }
  }
  else stop(event)
}
final = function() report
sim <- Simulation$new()
do.call("rbind", lapply(1:n, function(id) data.frame(id=id, sim$run())))

## ticking bomb - toy example

testRsimulation5 <- function(n=1) {
  Simulation <-
    setRefClass("Simulation",
      contains = "BaseDiscreteEventSimulation",
      fields = list(report = "data.frame"))
  Simulation$methods(
    init = function() {
      report <<- data.frame()
scheduleAt(rexp(1,1), "tick")
if (runif(1)<0.1)
    scheduleAt(rexp(1,1), "explosion")
},
handleMessage = function(event) {
    report <<- rbind(report, data.frame(begin = attr(event,"sendingTime"),
        end = currentTime,
        event = event,
        stringsAsFactors = FALSE))

    if (event == "explosion")
        clear()
    else {
        clear() # queue
        if (event == "tick") scheduleAt(currentTime+rexp(1,1), "tock")
        else scheduleAt(currentTime+rexp(1,1), "tick")
        if (runif(1)<0.1)
            scheduleAt(currentTime+rexp(1,1), "explosion")
    }
},
final = function() report

sim <- Simulation$new()
do.call("rbind", lapply(1:n, function(id) data.frame(id=id,sim$run())))

## End(Not run)
Value
data-frame
No return value, called for side effects
No return value, called for side effects
No return value, called for side effects
No return value, called for side effects
No return value, called for side effects
No return value, called for side effects

RNGStream

S3 class to work with RngStream objects

Description
S3 class to work with RngStream objects
Use RNGStream as an old class
With method for RNGStream S3 class

Usage
RNGStream(nextStream = TRUE, iseed = NULL)

## S3 method for class 'RNGStream'
with(data, expr, ...)

Arguments
nextStream whether to move to the next stream (default=TRUE)
iseed set seed after changing RNG (otherwise keep the current seed)
data object of type RNGStream
expr expression using the RNGStream
... other arguments passed to eval()

Value
list of class RNGStream with components:
resetRNGkind function to reset to the previous RNG and seed
seed function to return the current seed itemopen function to use the current seed itemclose function to make the current seed equal to .Random.seed itemresetStream function to move back to start of stream itemresetSubStream function to move back to start of sub-stream itemnextSubStream function to move back to sub-stream itemnextStream function to move to next sub-stream itemnextStream function to move to next stream the value from the expression
Examples

## set up one stream
s1 <- RNGStream()
s1$open()
rnorm(1)
s1$nextSubStream()
rnorm(1)
## reset the stream
s1$resetStream()
rnorm(2)
s1$nextSubStream()
rnorm(2)

## now do with two streams
s1$resetStream()
s2 <- RNGStream()
with(s1,rnorm(1))
with(s2,rnorm(1))
s1$nextSubStream()
with(s1,rnorm(1))
## now reset the streams and take two samples each time
s1$resetStream()
s2$resetStream()
with(s1,rnorm(2))
with(s2,rnorm(2))
s1$nextSubStream()
with(s1,rnorm(2))


Description

At present, this passes the object to summary and then prints

Usage

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

## S3 method for class 'summary.SummaryReport'
print(x, ...)

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

## S3 method for class 'SummaryReport'
rbind(...)

## S3 method for class 'SummaryReport'
ascii(
  object,
  include.rownames = FALSE,
  include.colnames = TRUE,
  header = TRUE,
  digits = c(0, 3, 2, 2, 4, 4),
  ...
)

## S3 method for class 'SummaryReport'
ICER(object1, object2, ...)

## S3 method for class 'ICER.SummaryReport'
ascii(
  object,
  include.rownames = TRUE,
  include.colnames = TRUE,
  header = TRUE,
  digits = c(1, 1, 3, 3, 1, 1, 3, 3, 1),
  rownames = c("Reference", "Treatment"),
  colnames = c("Costs", "(se)", "QALYs", "(se)", "Costs", "(se)", "QALYs", "(se)", "ICER"),
  tgroup = c("Total", "Incremental"),
  n.tgroup = c(4, 5),
  ...
)

### Arguments

object  
an ICER.SummaryReport object

...  
other arguments to pass to ascii

x  
SummaryReport object

include.rownames  
logical for whether to include rownames (default=FALSE)

include.colnames  
logical for whether to include colnames (default=TRUE)

header  
logical for whether to include the header (default=TRUE)

digits  
vector of the number of digits to use for each column

object1  
SummaryReport object (reference)

object2  
SummaryReport object

rownames  
rownames for output

colnames  
colnames for output

tgroup  
tgroup arg passed to ascii

n.tgroup  
arg passed to ascii
Value

a list of class SummaryReport with components:

- **n** Number of simulations
- **indivip** boolean with whether individual values were retained
- **utilityDiscountRate** discount rate for utilities/QALYs
- **costDiscountRate** discount rate for costs
- **QALE** Quality-adjusted life expectancy (discounted)
- **LE** Life expectancy (not discounted)
- **ECosts** Life-time expected costs (discounted)
- **se.QALE** standard error for QALE
- **se.Ecosts** standard error Ecosts

a SummaryReport object

ascii object

a list of type ICER.SummaryReport with components:

- **n** number of simulations
- **utilityDiscountRate** Discount rate for the utilities/QALE
- **costDiscountRate** Discount rate for the costs
- **s1** summary for object1
- **s2** summary for object2
- **dQALE** QALE for object2 minus QALE for object1
- **dCosts** Costs for object2 minus costs for object1
- **ICER** change of costs divided by change in QALEs
- **se.dQALE** standard error for dQALE
- **se.dCosts** standard error for dCosts

ascii object
datasets

- fhcrcData, 7
- microsimulationLdFlags, 3
- onLoad (microsimulationLdFlags), 3
- onUnload (microsimulationLdFlags), 3
- advance.substream (microsimulationLdFlags), 3
- BaseDiscreteEventSimulation (pqueue), 7
- BaseDiscreteEventSimulation-class (pqueue), 7
- callCalibrationPerson, 5
- callCalibrationSimulation (pqueue__new), 13
- callIllnessDeath (callCalibrationPerson), 5
- callPersonSimulation (callCalibrationPerson), 5
- callSimplePerson (callCalibrationPerson), 5
- callSimplePerson2 (callCalibrationPerson), 5
- discountedInterval, 6
- discountedPoint (microsimulationLdFlags), 3
- enum (microsimulationLdFlags), 3
- enum< (microsimulationLdFlags), 3
- EventQueue (pqueue), 7
- EventQueue-class (pqueue), 7
- fhcrcData, 7
- frontier (microsimulationLdFlags), 3
- ICER (microsimulationLdFlags), 3
- inlineCxxPlugin (microsimulationLdFlags), 3
- LdFlags (microsimulationLdFlags), 3
- lines.frontier (microsimulationLdFlags), 3
- microsimulation (microsimulation-package), 2
- microsimulation-package, 2
- microsimulation.exit (microsimulationLdFlags), 3
- microsimulation.init (microsimulationLdFlags), 3
- next.user.Random.substream (microsimulationLdFlags), 3
- pqueue, 7
- pqueue__cancel (pqueue__new), 13
- pqueue__clear (pqueue__new), 13
- pqueue__empty (pqueue__new), 13
- pqueue__new, 13
- pqueue__pop (pqueue__new), 13
- pqueue__push (pqueue__new), 13
- PQueueRef (pqueue), 7
- PQueueRef-class (pqueue), 7
- r_create_current_stream (pqueue__new), 13
- r_get_user_random_seed (pqueue__new), 13
- r_next_rng_substream (pqueue__new), 13
- r_remove_current_stream (pqueue__new), 13
INDEX

r_rng_advance_substream (pqueue__new), 13
r_set_user_random_seed (pqueue__new), 13
Rcpp, 2
RNGstate (.microsimulationLdFlags), 3
RNGStream, 14
RNGStream-class (RNGStream), 14
rnormPos (.microsimulationLdFlags), 3

set.user.Random.seed (.microsimulationLdFlags), 3
signed (.microsimulationLdFlags), 3
summary.SummaryReport, 15

unsigned (.microsimulationLdFlags), 3
user.Random.seed (.microsimulationLdFlags), 3

with.RNGStream (RNGStream), 14