Cervix Microsimulation Model Follow-Up Document

Carlos J. Dommar D’Lima

2024-04-05

This document details the progression of the development of microsimulation models for cervical cancer built upon the currently existing models Markov cohort models in the group.

R options for Microsimulation

There are a different ways one can build and implement microsimulation models based in existing libraries and dedicated platforms (such as ABMs, in Repast, Netlogo, etc.). However I aim to use R as much as possible or make it R-user friendly. For this regard there is several ways to proceed:

- Build everything from scratch
- Leverage existing coded libraries/packages

Existing code/packages for micorsimulation using R (that I am aware of, so far)

- Krijkamp et al (2018) (https://journals.sagepub.com/doi/abs/10.1177/0272989X18754513)
- Clements et al (2018?) `microsimulation` R package (https://cran.r-project.org/web/packages/microsimulation/index.html)
- Tikka’s et al (2021) `Sima` R open-source simulation framework (https://microsimulation.pub/articles/00240)

Krijkamp et al microsimulation code (2018)

Among the reasons Krijkamps’s code is a good start for implementing microsimulation models i R rather than start from scratch are the

a. the documentation is good;

b. the code for the simple case Health-Sick-Sicker-Death model is also simple and concise; the code seems to be maintained in a git repository (https://github.com/DARTH-git/Microsimulation-tutorial) and it is part of a larger open source set of tools of a group called Decision Analysis in R for Technologies in Health - [DARTH] (http://darthworkgroup.com/ (http://darthworkgroup.com/)) with repositories (https://github.com/DARTH-git) of a number of tools that can be useful such cohort modeling (https://github.com/DARTH-git/Cohort-modeling-tutorial), and a decision-analytic modeling coding [framework] (https://github.com/DARTH-git/darthpack (https://github.com/DARTH-git/darthpack));

c. Educational

Some possible drawbacks include slow code and difficulties in scaling up; as the model complexity increases, the code may become less clean and readable. Adopting an Object-Oriented (OO) approach would likely be a better long-term solution for production code. In this scenario, exploring `Sima` would be worthwhile.

Krijkamp implementation for a Cervix model with 12 cancer-

related states and 15 age-dependent transition matrices.

The idea is to build on the simple sick-sicker (https://github.com/DARTH-git/Microsimulation-tutorial) model introduced by Krijkamp et al. The initial step is to incorporate the model framework of the Markov cohort cervix model which has 12 mutually exclusive cancer-related states.

```
################################################################################
# This code is a modified version of the original code from:
# [https://github.com/DARTH-git/Microsimulation-tutorial] (Krijkamp et al 2018 Sick-Sicker mo
del)
# Modifications by: Carlos Dommar D'Lima - carlos.dommar@gmail.com
# This code extended the "sick-sicker" model of the original authors to a
# multi-state cervix cancer model
################################################################################

library(tidyverse)
rm(list = ls())

# Define the base directory and subdirectory components
base_dir <- "Q:/my_Q_docs"
project_dir <- "Cervix_MicroSim/CervixMicroSim_Carlos/carlos__Krijkamp_ver/data"
filename <- "probs.rds"
#filename <- "probs2.rds #this have probabilities way larger than 1 (error)
#filename <- "probs3.rds #this have probabilities way larger than 1 (error)
# Construct the full path using file.path() with line continuation
rds_file <- file.path(base_dir, project_dir, filename)

# Read the RDS file using the constructed path
my_Probs <- readRDS(rds_file)

# choose, as a test, only one of the 15 age-related transition matrices,
my_Probs <- # transition matrix (for all sim cycles)
  my_Probs %>%
  #dplyr::filter(Age.group == "25-29") %>% # choose one for test
  as_tibble() # I need a tibble to use 'rename' function down there:

# tidying up a bit the transition matrix:
my_Probs <- my_Probs %>% dplyr::rename("H" = "Well")
# Rename the 'old_name' column to 'new_name'

my_Probs <- my_Probs %>% as.data.frame() # convert back to data.frame (no needed?)

################################################################
# Function to extract and convert numbers from factor levels
extract_numbers <- function(range_factor) {
  range_string <- as.character(range_factor)
  numbers <- as.numeric(unlist(strsplit(range_string, "-")))
  return(numbers)
}
################################################################

# before apply the function, convert Age.group from factor to character
# my_Probs$Age.group <- as.character(my_Probs$Age.group)
# Apply the function to the Range column and create new columns
my_Probs$Lower  <- sapply(my_Probs$Age.group, function(x) extract_numbers(x)[1])
my_Probs$Larger <- sapply(my_Probs$Age.group, function(x) extract_numbers(x)[2])

#rownames(my_Probs) <-
#  my_Probs %>%
#  colnames() %>%
#  tail(-1)
```

```
#my_rownames <-
#  my_Probs %>%
#  colnames() %>%
#  tail(-1)
#rm(my_probs)  # no needed any longer

# for feeding the microsimulation function with aged-based multiple transition
# matrices, they need a bit previous prep:
# so I going to make a list with elements containing a transition matrix
# and pass it to the microsimulation function
#age_interv <-  # list with all age intervals
#  my_Probs %>%
#  select(Age.group) %>% unique()
#list_matrices <- list()
#for (age in age_interv$Age.group)
#{
#  #print(age)
#  #rownames(my_Probs[my_Probs$Age.group==age]) <-
#
#  my_Probs %>% filter(Age.group == age) %>% head() %>% print()
#  #list_matrices <- c(list_matrices, my_Probs %>% filter(Age.group == age))
#}
```

```
my_Probs %>%
  head()
```

```
##   Age.group          H HR.HPV.infection      CIN1       CIN2       CIN3
## 1     10-14 0.99990473       0.00000000 0.0000000 0.00000000 0.00000000
## 2     10-14 0.69844771       0.08977946 0.1246488 0.08704182 0.00000000
## 3     10-14 0.19800834       0.01427319 0.7181387 0.04273225 0.02676601
## 4     10-14 0.17224516       0.01854424 0.5496409 0.25948797 0.00000000
## 5     10-14 0.02520056       0.01675474 0.0000000 0.59982074 0.35814244
## 6     10-14 0.00000000       0.00000000 0.0000000 0.00000000 0.00000000
##      FIGO.I   FIGO.II FIGO.III FIGO.IV Survival   CC_Death  Other.Death Lower
## 1 0.0000000 0.0000000        0       0        0 0.00000000 9.527442e-05    10
## 2 0.0000000 0.0000000        0       0        0 0.00000000 8.223003e-05    10
## 3 0.0000000 0.0000000        0       0        0 0.00000000 8.149336e-05    10
## 4 0.0000000 0.0000000        0       0        0 0.00000000 8.175864e-05    10
## 5 0.0000000 0.0000000        0       0        0 0.00000000 8.152593e-05    10
## 6 0.5487358 0.4119176        0       0        0 0.03926509 8.150000e-05    10
##   Larger
## 1     14
## 2     14
## 3     14
## 4     14
## 5     14
## 6     14
```

Now I introduce the model parameters:

```
n_i   <- 100000                 # number of simulated individuals
n_i   <- 10^6                 # number of simulated individuals
n_t   <- 75                     # time horizon, 75 cycles

# cycle_period can go from one month to one year. that is
# I think a sensible way is to offer the following frequencies, different to
# this then just set it monthly:
cycle_period <- "1mth"
cycle_period <- "2mth"
cycle_period <- "3mth"
cycle_period <- "4mth"
cycle_period <- "6mth"
cycle_period <- "1yr" # i.e. 12mth

v_n <- rownames(my_Probs)

n_s   <- length(v_n)                # the number of health states
v_M_1 <- rep("H", n_i)              # everyone begins in the healthy state
#v_M_1 <- rep("Well", n_i)              # everyone begins in the healthy state
d_c   <- d_e <- 0.03                # equal discounting of costs and QALYs by 3%
v_Trt <-
  c("No Treatment", "Treatment")    # store the strategy names

################################################################################
# Cost and utility inputs
c_H     <- 2000                # cost of remaining one cycle healthy
c_S1    <- 4000                # cost of remaining one cycle sick
c_S2    <- 15000               # cost of remaining one cycle sicker
c_Trt   <- 12000               # cost of treatment (per cycle)

u_H     <- 1                   # utility when healthy
u_S1    <- 0.75                # utility when sick
u_S2    <- 0.5                 # utility when sicker
u_Trt   <- 0.95                # utility when sick(er) and being treated

# From our Markov cervix model (CC's natural history?):
cost_Vec = c(0, 39.54, 288.91, 1552.27, 1552.27,
             5759.81, 12903.63, 23032.41, 35323.14, 0, 0, 0)
```

The Functions:

The Sampling function:

Krijkamp et al (2018) (https://journals.sagepub.com/doi/abs/10.1177/0272989X18754513) developed a sampling function they call `samplev()` by modifying and random number generating for multinomial variables from the `Hmisc` R package. The `samplev()` function randomly draws the individuals state vector at t+1. `samplev()` takes as argument `probs` and `m`. `probs` is a matrix array `n_i x n_s` of number of individuals times number of health-states in the model. Each element $\text{probs} = p_{ij}$ is the probability of the individual i to transition to the j health-state at $t+1$ given its current state at t as described in the appropriate transition matrix.

We then need to both sample random numbers and make a transition selection based on the random number to progress with the evolution of states of individuals. That is the procedure is i a as follows:

`From my AI prompt (<https://g.co/gemini/share/a4be68ba9202>):`

- Random Number Sampling: When an individual needs to make a state transition, a random number is generated between 0 and 1 (uniform distribution). This random number is then compared to the cumulative probabilities of all possible transitions from the current state.
- Transition Selection: The transition with a cumulative probability range that encompasses the generated random number is selected. This essentially means that transitions with higher probabilities have a larger range within the 0-1 interval, making them more likely to be chosen by the random number.

Cumulative Probabilities and Binning:

So what `samplv()` does is

1. Calculate Cumulative Probabilities: For each state, all the individual transition probabilities are summed up sequentially. This creates a series of cumulative probabilities. For example, if you have three transitions (A, B, and C) with probabilities 0.3, 0.4, and 0.3 respectively, their cumulative probabilities would be:

- Transition A: 0.3
- Transition B: 0.3 + 0.4 = 0.7
- Transition C: 0.7 + 0.3 = 1.0 (This must always sum to 1)

2. Binning the Range (0-1): The range between 0 and 1 is then conceptually divided into bins based on these cumulative probabilities. In our example:

- Transition A: 0 - 0.3 (occupies the first 30% of the range)
- Transition B: 0.3 - 0.7 (occupies the next 40% of the range)
- Transition C: 0.7 - 1.0 (occupies the last 30% of the range)

3. Selecting the Transition:

3.1 Sample a Random Number: As you mentioned, a random number between 0 and 1 is generated.

3.2 Identify the Winning Bin: This random number is then compared to the binned ranges. The transition whose cumulative probability range encompasses the random number is chosen as the next state.

```r
# efficient implementation of the rMultinom() function of the Hmisc package ####
samplev <- function (probs, m) {
  d <- dim(probs) # i.e. number of individuals times number of states: n_i x n_s
  n <- d[1]       # number of individuals n_s
  k <- d[2]       # number of states
  lev <- dimnames(probs)[[2]] # vector with  names of health states
  if (!length(lev)) # checks if `lev` vector (states names) is empty
                    # or has length 0
    lev <- 1:k # if empty (evaluates to `TRUE`), it assigns numeric state labels
             # (1:k) to `lev`
  ran <-
    matrix(lev[1], ncol = m, nrow = n) # create array n_s x m (m=1)
                                       # consisting in of health-state stored in
                                       # `lev[1]`, "H" in our case.



  ################################################################################
  ########## Creating the matrix of cumulative distributions U ##################
  U <- t(probs)    # transpose probs from (`n_i*n_s`) to (`n_s*n_i`)
  for(i in 2:k) {
    # This loop fills U with the cumulative probabilities of each individual
    # across all its possible transitions (`v_s`or `lev` within thus function).
    # That is each column of `U` represents the cumulative distribution for each
    # individual across its corresponding transitions.
    # The last element of each column must sum 1 (or close enough:)
    U[i, ] <- U[i, ] + U[i - 1, ]
  }
  if (any((U[k, ] - 1) > 1e-04))
    stop("error in multinom: probabilities do not sum to 1")
  ################################################################################
  ################################################################################

  ### Random sampling, binning, and moving states:
  for (j in 1:m) {
    un <- rep(runif(n), rep(k, n)) # repeat `runif(n)` `rep(k,n)`times
                                   # this create a numeric of `n_i x n_s` that
                                   # sample  an uniformed distributed number
                                   # between 0 and 1. The generated random number
                                   # repeats itself `n_s` times and then another
                                   # rand unif number is drawn. This process is
                                   # carried out `n_i` times. NOTE: every time
                                   # runif() is run it produces a new random sample
                                   # i.e. it does not seem dependent on the seed

    # here's where we choose the individuals' next states:
    ran[, j] <- lev[1 + colSums(un > U)]
   #print(ran[,j])
  }
  ran
}
```

The Probability function

```
knitr::opts_chunk$set(tidy = TRUE, out.width = 60)
######################### Probability function ##################################
# The Probs function that updates the transition probabilities
# of every cycle is shown below.
Probs <- function(M_it, my_Probs) {
  # M_it:    health state occupied by individual i at cycle t (character variable)
  # my mod:
  # my_Probs: Transition matrix from our Markov cohort model

  m_P_it <- matrix(NA, n_s, n_i)     # create vector of state transition probabilities
  rownames(m_P_it) <- v_n            # assign names to the vector

  ## update the v_p with the appropriate probabilities

  # remind that v_n are the vector names of health states.
  # This goes eventually within a loop or a lapply func over all health states
  m_P_it[,M_it == v_n[1]]   <-
    lapply(X = v_n, function(x) trans_prb(P = my_Probs,state1 = v_n[1], state2 = x)) %>%
    unlist()
  m_P_it[,M_it == v_n[2]]   <-
    lapply(X = v_n, function(x) trans_prb(P = my_Probs,state1 = v_n[2], state2 = x)) %>%
    unlist()
  m_P_it[,M_it == v_n[3]]   <-
    lapply(X = v_n, function(x) trans_prb(P = my_Probs,state1 = v_n[3], state2 = x)) %>%
    unlist()
  m_P_it[,M_it == v_n[4]]   <-
    lapply(X = v_n, function(x) trans_prb(P = my_Probs,state1 = v_n[4], state2 = x)) %>%
    unlist()
  m_P_it[,M_it == v_n[5]]   <-
    lapply(X = v_n, function(x) trans_prb(P = my_Probs,state1 = v_n[5], state2 = x)) %>%
    unlist()
  m_P_it[,M_it == v_n[6]]   <-
    lapply(X = v_n, function(x) trans_prb(P = my_Probs,state1 = v_n[6], state2 = x)) %>%
    unlist()
  m_P_it[,M_it == v_n[7]]   <-
    lapply(X = v_n, function(x) trans_prb(P = my_Probs,state1 = v_n[7], state2 = x)) %>%
    unlist()
  m_P_it[,M_it == v_n[8]]   <-
    lapply(X = v_n, function(x) trans_prb(P = my_Probs,state1 = v_n[8], state2 = x)) %>%
    unlist()
  m_P_it[,M_it == v_n[9]]   <-
    lapply(X = v_n, function(x) trans_prb(P = my_Probs,state1 = v_n[9], state2 = x)) %>%
    unlist()
  m_P_it[,M_it == v_n[10]]  <-
    lapply(X = v_n, function(x) trans_prb(P = my_Probs,state1 = v_n[10], state2 = x)) %>%
    unlist()
  m_P_it[,M_it == v_n[11]]  <-
    lapply(X = v_n, function(x) trans_prb(P = my_Probs,state1 = v_n[11], state2 = x)) %>%
    unlist()
  m_P_it[,M_it == v_n[12]]  <-
    lapply(X = v_n, function(x) trans_prb(P = my_Probs,state1 = v_n[12], state2 = x)) %>%
    unlist()
  ifelse(colSums(m_P_it) >=  # return the transition probabilities or produce an error
          .991, return(t(m_P_it)), print("Probabilities do not sum to 1"))
}
################################################################################
```

The Costs function:

For testing purpose I implement a very simple cost function, that is I apply the same cost for all disease stages:

```
### Costs function The Costs function estimates the costs at every cycle.
Costs <- function(M_it, cost_Vec, Trt = FALSE) {
    # my mode with costs taken from our Markov cohort model:
    c_it <- 0  # by default the cost for everyone is zero
    c_it[M_it == "H"] <- cost_Vec[1]  # update the cost
    c_it[M_it == "Survival"] <- cost_Vec[2]  # update the cost
    c_it[M_it == "HR.HPV.infection"] <- cost_Vec[3]  # update the cost
    c_it[M_it == "CIN1"] <- cost_Vec[4] + c_Trt * Trt  # update the cost
    c_it[M_it == "CIN2"] <- cost_Vec[5] + c_Trt * Trt  # update the cost
    c_it[M_it == "CIN3"] <- cost_Vec[6] + c_Trt * Trt  # update the cost
    c_it[M_it == "FIGO.I"] <- cost_Vec[7] + c_Trt * Trt  # update the cost
    c_it[M_it == "FIGO.II"] <- cost_Vec[8] + c_Trt * Trt  # update the cost
    c_it[M_it == "FIGO.III"] <- cost_Vec[9] + c_Trt * Trt  # update the cost
    c_it[M_it == "FIGO.IV"] <- cost_Vec[10] + c_Trt * Trt  # update the cost
    c_it[M_it == "CC_Death"] <- cost_Vec[11]  # update the cost
    c_it[M_it == "Other.Death"] <- cost_Vec[12]  # update the cost

    return(c_it)  # return the costs
}
```

The QALYs function:

```
### Health outcome function The Effs function to update the utilities at every
### cycle.
Effs <- function(M_it, Trt = FALSE, cl = 1) {
    ## M_it: health state occupied by individual i at cycle t (character
    ## variable) Trt: is the individual treated? (default is FALSE) cl: cycle
    ## length (default is 1)

    # My cervix model mod:
    u_it <- 0  # by default the utility for everyone is zero
    # I assume healthy/infected and survival have the same utility:
    u_it[M_it == "H"] <- u_H  # update the utility if healthy
    u_it[M_it == "HR.HPV.infection"] <- u_H  # update the utility if infected
    u_it[M_it == "Survival"] <- u_H  # update the utility if Survived
    # _again, for testing purpose I assume all CIN states have the same utility
    u_it[M_it == "CIN1"] <- Trt * u_Trt + (1 - Trt) * u_S1  # update the utility
    # if sick conditional on treatment
    u_it[M_it == "CIN2"] <- Trt * u_Trt + (1 - Trt) * u_S1  # update the utility
    # if sick conditional on treatment
    u_it[M_it == "CIN3"] <- Trt * u_Trt + (1 - Trt) * u_S1  # update the utility
    # if sick conditional on treatment for testing I assume all FIGO states
    # have the same utility:
    u_it[M_it == "FIGO.I"] <- u_S2  # update the utility if sicker
    u_it[M_it == "FIGO.II"] <- u_S2  # update the utility if sicker
    u_it[M_it == "FIGO.III"] <- u_S2  # update the utility if sicker
    u_it[M_it == "FIGO.IV"] <- u_S2  # update the utility if sicker
    u_it[M_it == "CC_Death"] <- 0  # update the utility if dead
    u_it[M_it == "Other.Death"] <- 0  # update the utility if dead

    QALYs <- u_it * cl  # calculate the QALYs during cycle t
    return(QALYs)  # return the QALYs
}
```

The the main microsimulation function, `MicroSim`

This is a modified version of Krijkamp et al (2018) (https://journals.sagepub.com/doi/abs/10.1177/0272989X18754513) where I have extended the number of individual states to 12 and have included age-dependent transition matrices.

```
MicroSim <- function(v_M_1, n_i, n_t, v_n, d_c, d_e, TR_out = TRUE,
                     TS_out = TRUE, Trt = FALSE, seed = 1, Pmatrix) {
  #Arguments:
  # v_M_1:   vector of initial states for individuals
  # n_i:     number of individuals
  # n_t:     total number of cycles to run the model
  # v_n:     vector of health state names
  # d_c:     discount rate for costs
  # d_e:     discount rate for health outcome (QALYs)
  # TR_out:  should the output include a Microsimulation trace?
  #          (default is TRUE)
  # TS_out:  should the output include a matrix of transitions between states?
  #          (default is TRUE)
  # Trt:     are the n.i individuals receiving treatment? (scalar with a Boolean
  #          value, default is FALSE)
  # seed:    starting seed number for random number generator (default is 1)
  # Makes use of:
  # Probs:   function for the estimation of transition probabilities
  # Costs:   function for the estimation of cost state vamatrix: Matrix of
  # tranistion probabilities for each sim cycle.
  # Effs:    function for the estimation of state specific health outcomes (QALYs)
  # Pmatrix: Matrix of transition probabilities for each sim cycle.

  v_dwc <- 1 / (1 + d_c) ^ (0:n_t)   # calculate the cost discount weight based
  # on the discount rate d_c

  v_dwe <- 1 / (1 + d_e) ^ (0:n_t)   # calculate the QALY discount weight based
  # on the discount rate d.e

  # Create the matrix capturing the state name/costs/health outcomes
  # for all individuals at each time point:
  m_M <- m_C <- m_E <-  matrix(nrow = n_i, ncol = n_t + 1,
                               dimnames = list(paste("ind", 1:n_i, sep = " "),
                                               paste("cycle", 0:n_t, sep = " ")))

  m_M[, 1] <- v_M_1                  # indicate the initial health state

  set.seed(seed)                     # set the seed for every individual for the
                                     # random number generator

  m_C[, 1] <- Costs(M_it = m_M[, 1], # estimate costs per individual for the
                  cost_Vec =         # initial health state
                    cost_Vec,
                  Trt)
  m_E[, 1] <- Effs (m_M[, 1], Trt)   # estimate QALYs per individual for the
                                     # initial health state

  ###### run over all the cycles ##########
  for (t in 1:n_t) {
    # here I choose my transition matrix according to the cycle n_t:
    if (cycle_period == "1yr"){
      age_in_loop <- t + 9 # because our age intervals start at 10 years old
      #age_in_loop <-  28 # because our age intervals start at 10 years old
      my_age_prob_matrix <- my_Probs %>%
        dplyr::filter(Lower <= age_in_loop & Larger >= age_in_loop)
    }
    # Add colnames and update `v_n`:
    rownames(my_age_prob_matrix) <- v_n <<-
      my_age_prob_matrix %>%
      dplyr::select(-c(Age.group, Lower, Larger)) %>%
      colnames()

    # update/correct n_s (<<- let change variable from inside a function):
    n_s  <<- length(v_n)

    # Extract the transition probabilities of each individuals at cycle t
    # given the individual current state and the corresponding
    # transition probability matrix that depends on age:
    m_P <- Probs(M_it =  m_M[, t], my_Probs = my_age_prob_matrix)

    m_M[, t + 1] <- samplev(probs = m_P, m = 1)  # sample the next health state
    # and store that state in
    # matrix m_M

    m_C[, t + 1] <-                      # estimate costs per individual
      Costs(M_it = m_M[, t + 1],         # during cycle t + 1
            cost_Vec = cost_Vec, Trt)    # conditional on treatment


    m_E[, t + 1] <-              # estimate QALYs per individual
      Effs( m_M[, t + 1], Trt)   # during cycle t + 1 conditional on treatment


    cat('\r', paste(round(t/n_t * 100), "% done", sep = " ")) # display the
                                                              # progress of
                                                              # the simulation
  } # close the loop for the time points
  ######################################


  tc <- m_C %*% v_dwc       # total (discounted) cost per individual
  te <- m_E %*% v_dwe       # total (discounted) QALYs per individual

  tc_hat <- mean(tc)        # average (discounted) cost
  te_hat <- mean(te)        # average (discounted) QALYs

  if (TS_out == TRUE) {  # create a matrix of transitions across states
    TS <- paste(m_M, cbind(m_M[, -1], NA), sep = "->") # transitions from one
    # state to the other

    TS <- matrix(TS, nrow = n_i)
    rownames(TS) <- paste("Ind",   1:n_i, sep = " ")   # name the rows
    colnames(TS) <- paste("Cycle", 0:n_t, sep = " ")   # name the columns
  } else {
    TS <- NULL
  }

  ### to test TS and see if collect all tranistions:
  #unique_elements <-  sim_no_trt$TS %>% as_data_frame() %>%
  #  pivot_longer(-"Cycle 0") %>%
  #  distinct(value)
  if (TR_out == TRUE) {
    TR <- t(apply(m_M, 2,
                  function(x) table(factor(x, levels = v_n, ordered = TRUE))))
    #TR <- TR / n_i                                   # create a distribution
    # trace

    rownames(TR) <- paste("Cycle", 0:n_t, sep = " ") # name the rows
    colnames(TR) <- v_n                              # name the columns
  } else {
    TR <- NULL
  }

  # if TS_out == TRUE we can then compute the incidence as the number of new
  # cases for each type of cancer state. A new case of cancer state X in time t
  # is defined as an individual transition to this state X provided the
  # individual was not in that state X a time t-1
  # a characater with all transitions:
   transitions <-
     TS %>%
     as_tibble() %>%
     pivot_longer(everything(), names_to = "column") %>%
     distinct(value) %>%
     unique() %>%
     as.list() %>%
     unlist()

  if(TS_out == TRUE){
    Tot_Trans_per_t <- t(apply(TS, 2,
                  function(x) table(factor(x, levels = transitions, ordered = TRUE))))
    # trace
    rownames(Tot_Trans_per_t) <- paste("Cycle", 0:n_t, sep = " ") # name the rows
    #colnames(TR) <- v_n                              # name the columns
  } else {
    Tot_Trans_per_t <- NULL
  }


  results <- list(m_M = m_M, m_C = m_C, m_E = m_E, tc = tc, te = te,
                  tc_hat = tc_hat, te_hat = te_hat,
                  TS = TS, TR = TR,
                  Tot_Trans_per_t = Tot_Trans_per_t) # store the results from
                                                   # the simulation in a list

  return(results)  # return the results
}  # end of the MicroSim function
```

Test simulation

Perform cost-effectiveness analysis:

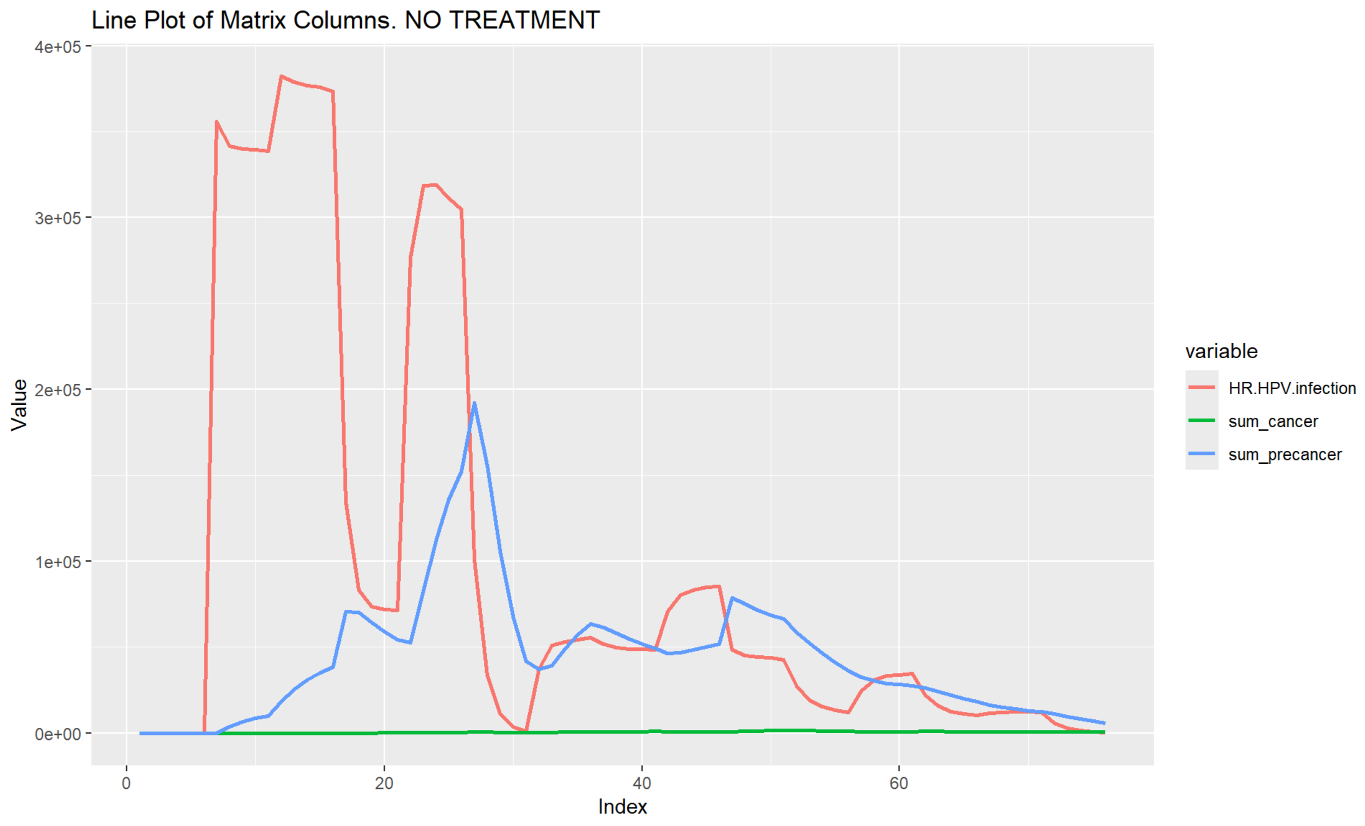
```
##################### Cost-effectiveness analysis #############################
# store the mean costs (and MCSE) of each strategy in a new variable C (vector costs)
v_C  <- c(sim_no_trt$tc_hat, sim_trt$tc_hat)
sd_C <- c(sd(sim_no_trt$tc), sd(sim_trt$tc)) / sqrt(n_i)
# store the mean QALYs (and MCSE) of each strategy in a new variable E (vector effects)
v_E  <- c(sim_no_trt$te_hat, sim_trt$te_hat)
sd_E <- c(sd(sim_no_trt$te), sd(sim_trt$te)) / sqrt(n_i)

delta_C <- v_C[2] - v_C[1]                   # calculate incremental costs
delta_E <- v_E[2] - v_E[1]                   # calculate incremental QALYs
# Monte Carlo Squared Error (MCSE) of incremental costs:
sd_delta_E <- sd(sim_trt$te - sim_no_trt$te) / sqrt(n_i)
# Monte Carlo Squared Error (MCSE) of incremental QALYs:
sd_delta_C <- sd(sim_trt$tc - sim_no_trt$tc) / sqrt(n_i)
ICER    <- delta_C / delta_E                 # calculate the ICER
results <- c(delta_C, delta_E, ICER)         # store the values in a new variable



# Create full incremental cost-effectiveness analysis table
table_micro <- data.frame(
  c(round(v_C, 0),  ""),           # costs per arm
  c(round(sd_C, 0), ""),           # MCSE for costs
  c(round(v_E, 3),  ""),           # health outcomes per arm
  c(round(sd_E, 3), ""),           # MCSE for health outcomes
  c("", round(delta_C, 0),   ""),  # incremental costs
  c("", round(sd_delta_C, 0),""),  # MCSE for incremental costs
  c("", round(delta_E, 3),   ""),  # incremental QALYs
  c("", round(sd_delta_E, 3),""),  # MCSE for health outcomes (QALYs) gained
  c("", round(ICER, 0),      "")   # ICER
)
# name the rows:
rownames(table_micro) <- c(v_Trt, "* are MCSE values")
# name the columns:
colnames(table_micro) <-
  c("Costs", "*",  "QALYs", "*", "Incremental Costs",
    "*", "QALYs Gained", "*", "ICER")
table_micro  # print the table
```

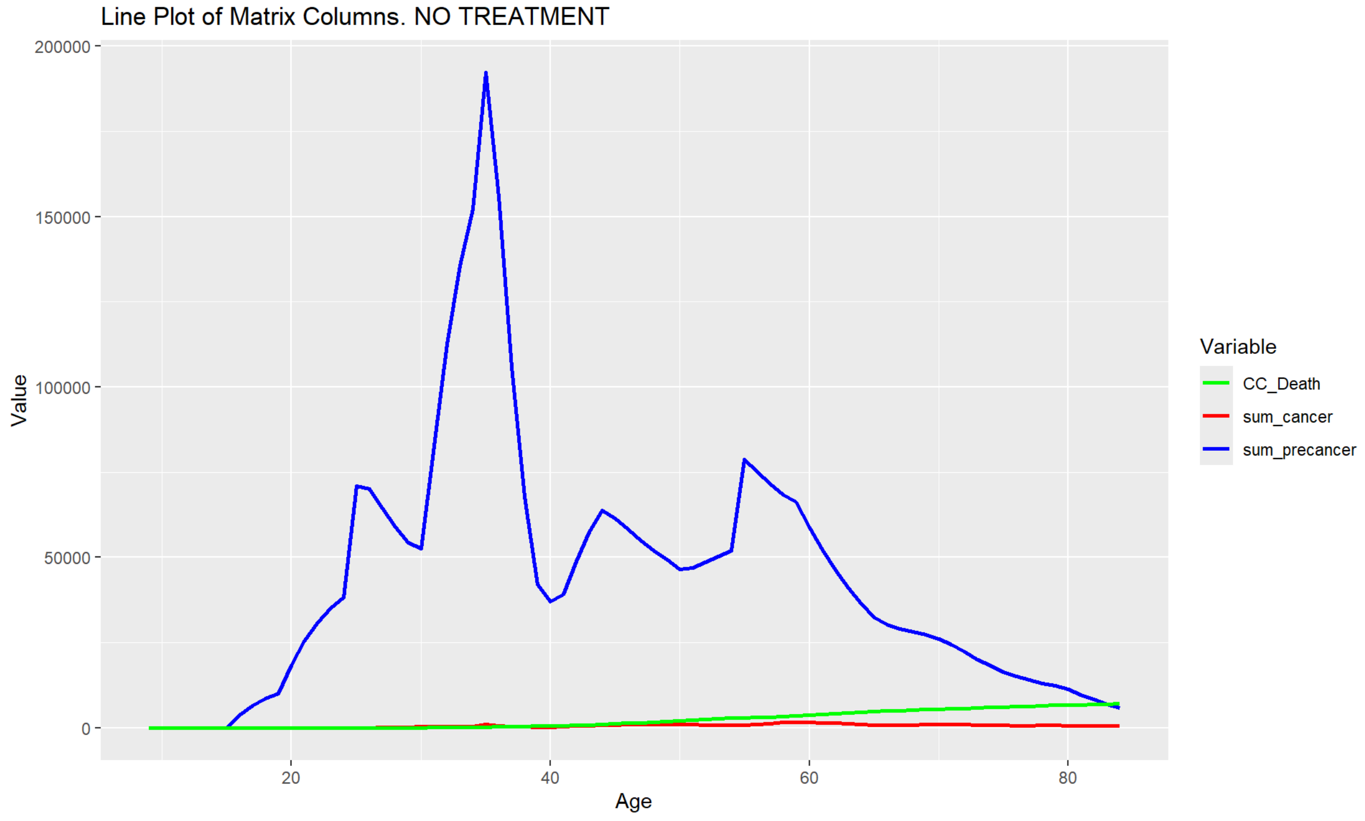
```
##                   Costs  *  QALYs     * Incremental Costs  * QALYs Gained *
## No Treatment       3560  4 29.644 0.002
## Treatment         19008 25 29.899 0.002             15448 21        0.255 0
## * are MCSE values
##                    ICER
## No Treatment
## Treatment         60542
## * are MCSE values
```

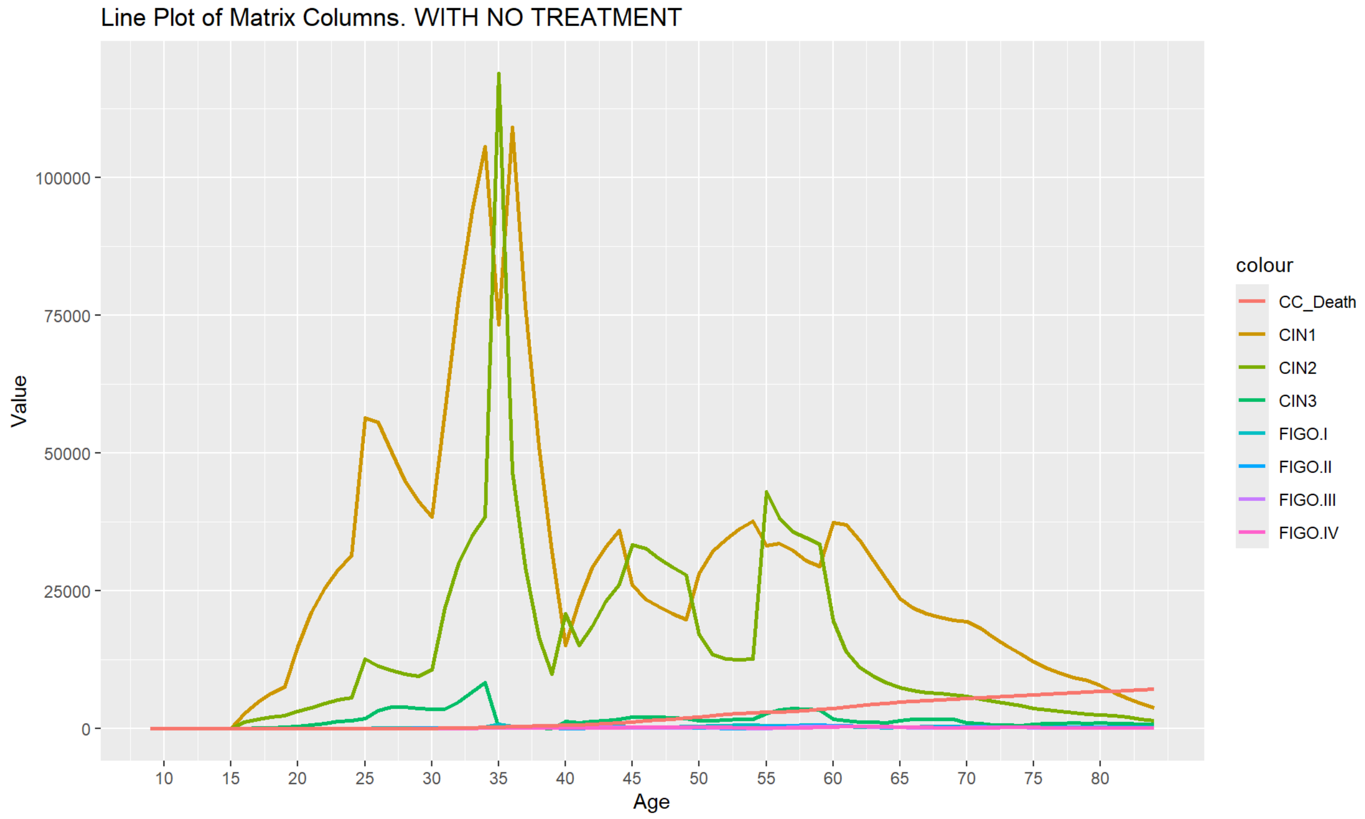
Computing incidence:

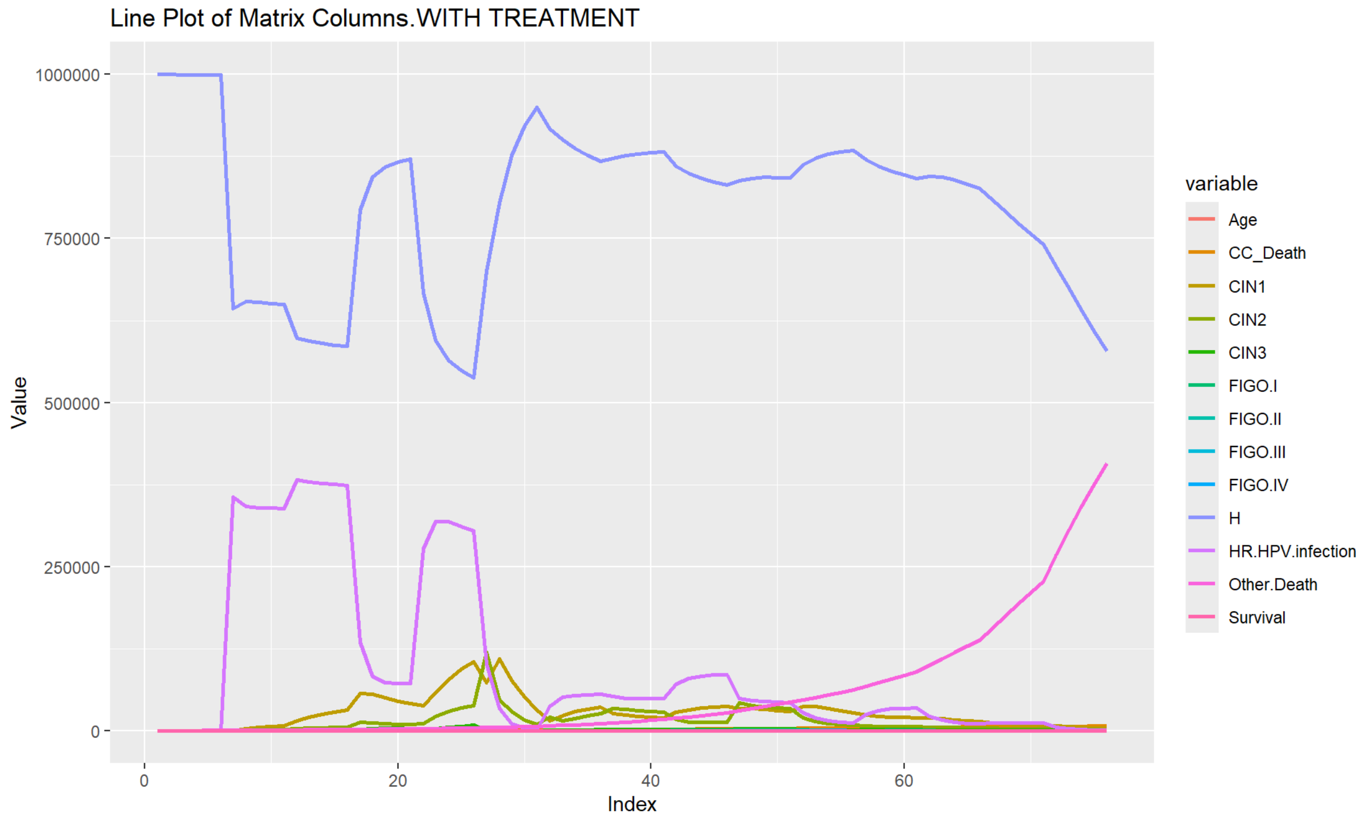
Incidence = (Number of new cases) / (Population size) x (Time period)

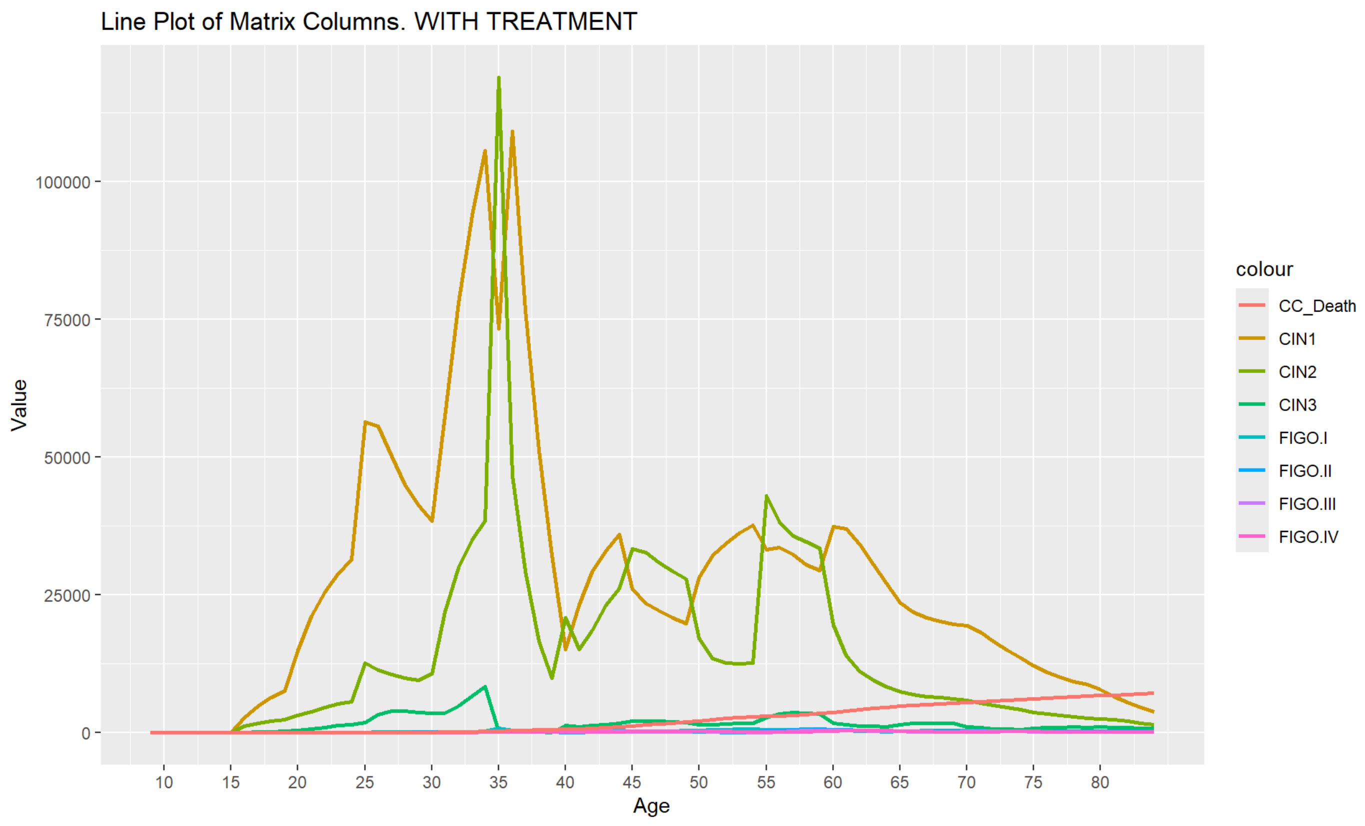
Plotting simulation curves






Line Plot of Matrix Columns. WITH NO TREATMENT
Value
Age
100000
75000
50000
25000
0
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
colour
CC_Death
CIN1
CIN2
CIN3
FIGO.I
FIGO.II
FIGO.III
FIGO.IV


Line Plot of Matrix Columns.WITH TREATMENT
Value
Index
1000000
750000
500000
250000
0
0
20
40
60
variable
Age
CC_Death
CIN1
CIN2
CIN3
FIGO.I
FIGO.II
FIGO.III
FIGO.IV
H
HR.HPV.infection
Other.Death
Survival

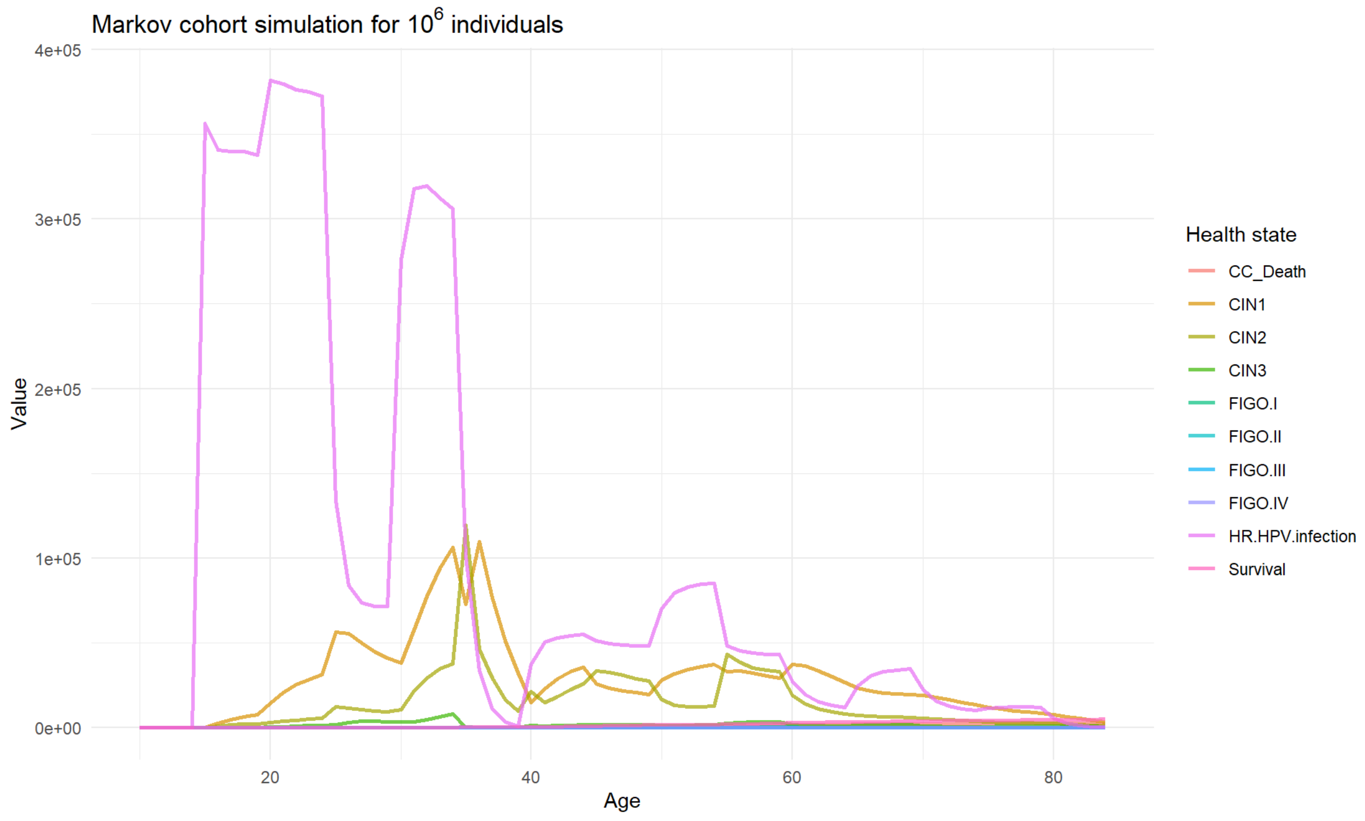


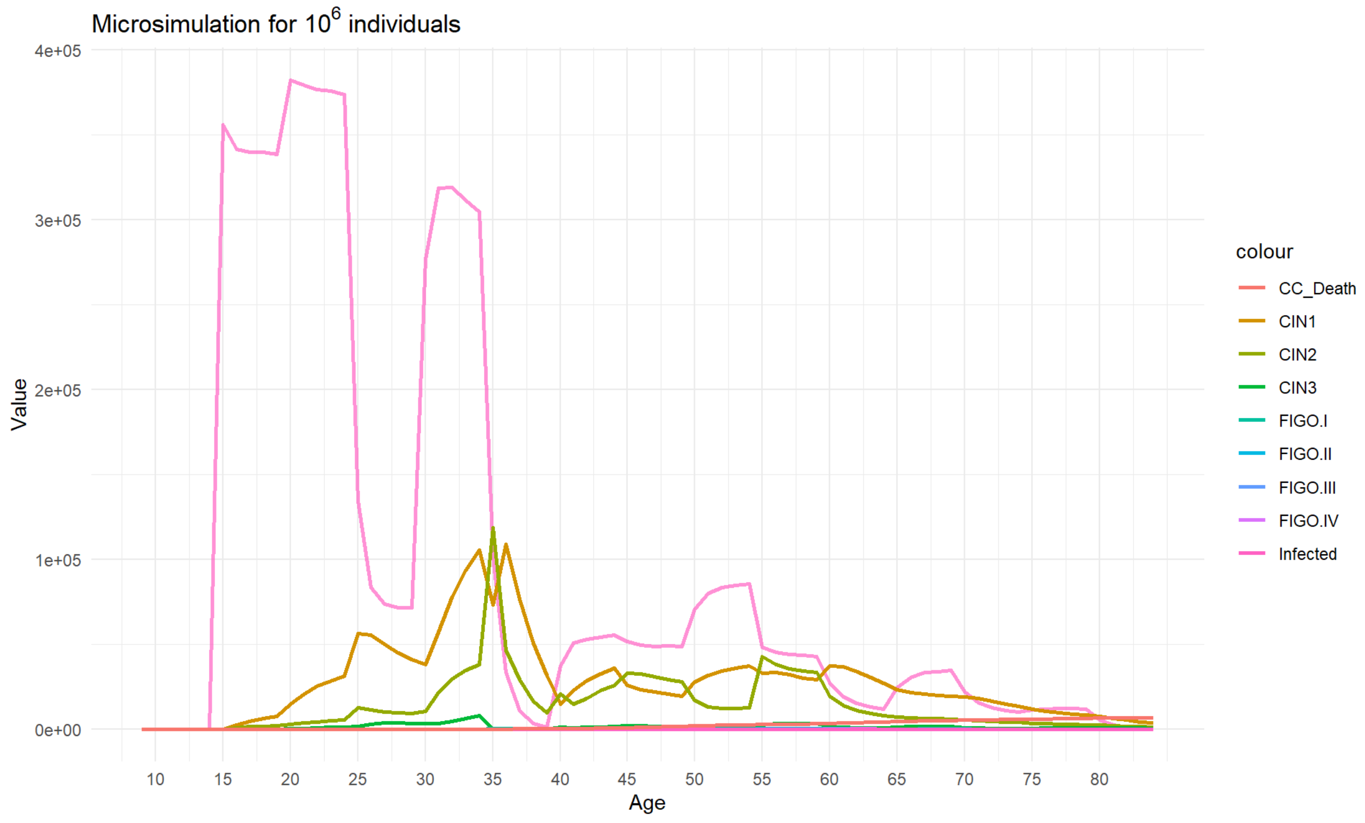
Comparing microsimulation with Markov cohort simulations.

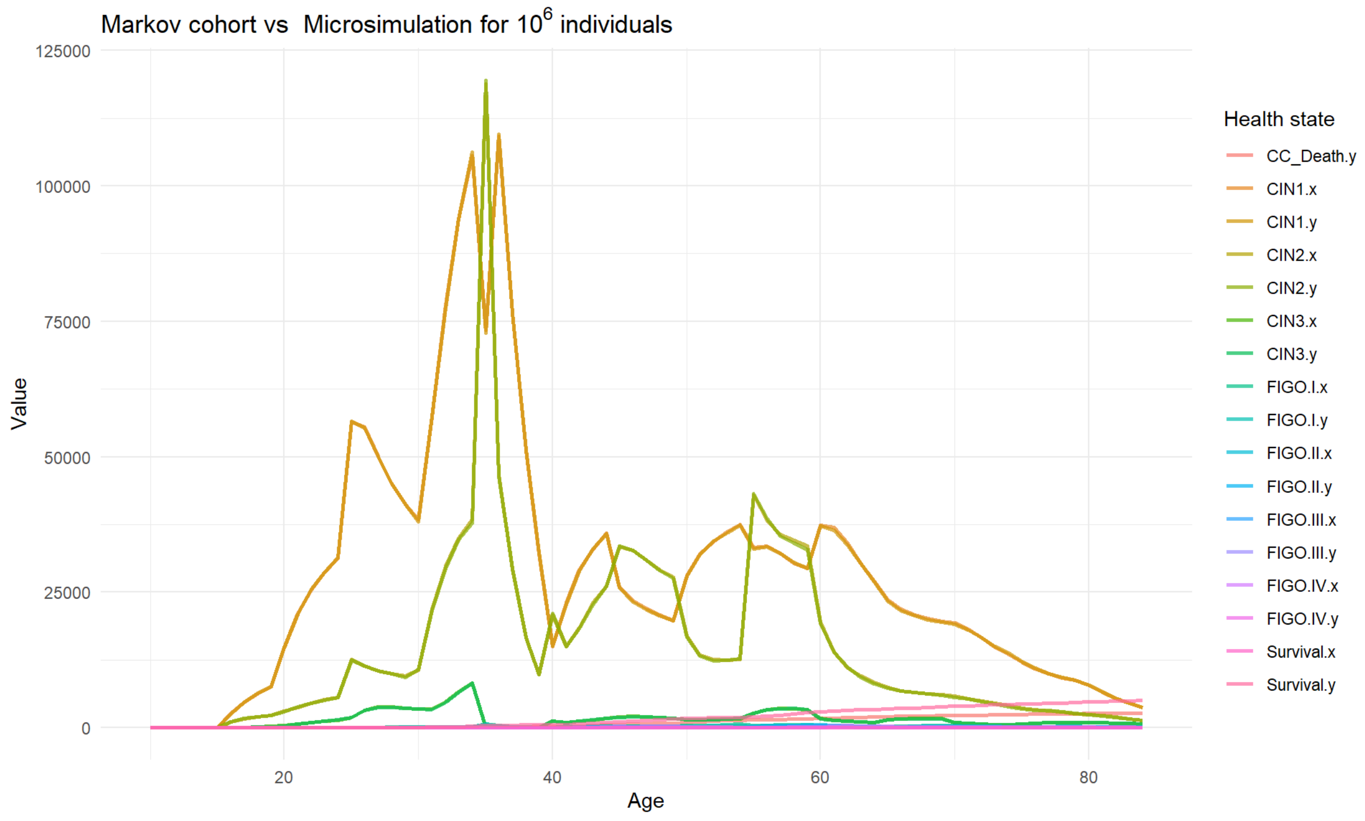
Given a output of a currently used Markov cohort model with the same 12-states structure and 15 age-dependent transition matrices I proceed to compare both simulations by aggregating individuals of the microsimulation and comparing them with the appropriate cohort model results.

```
## Loading required package: readxl
```

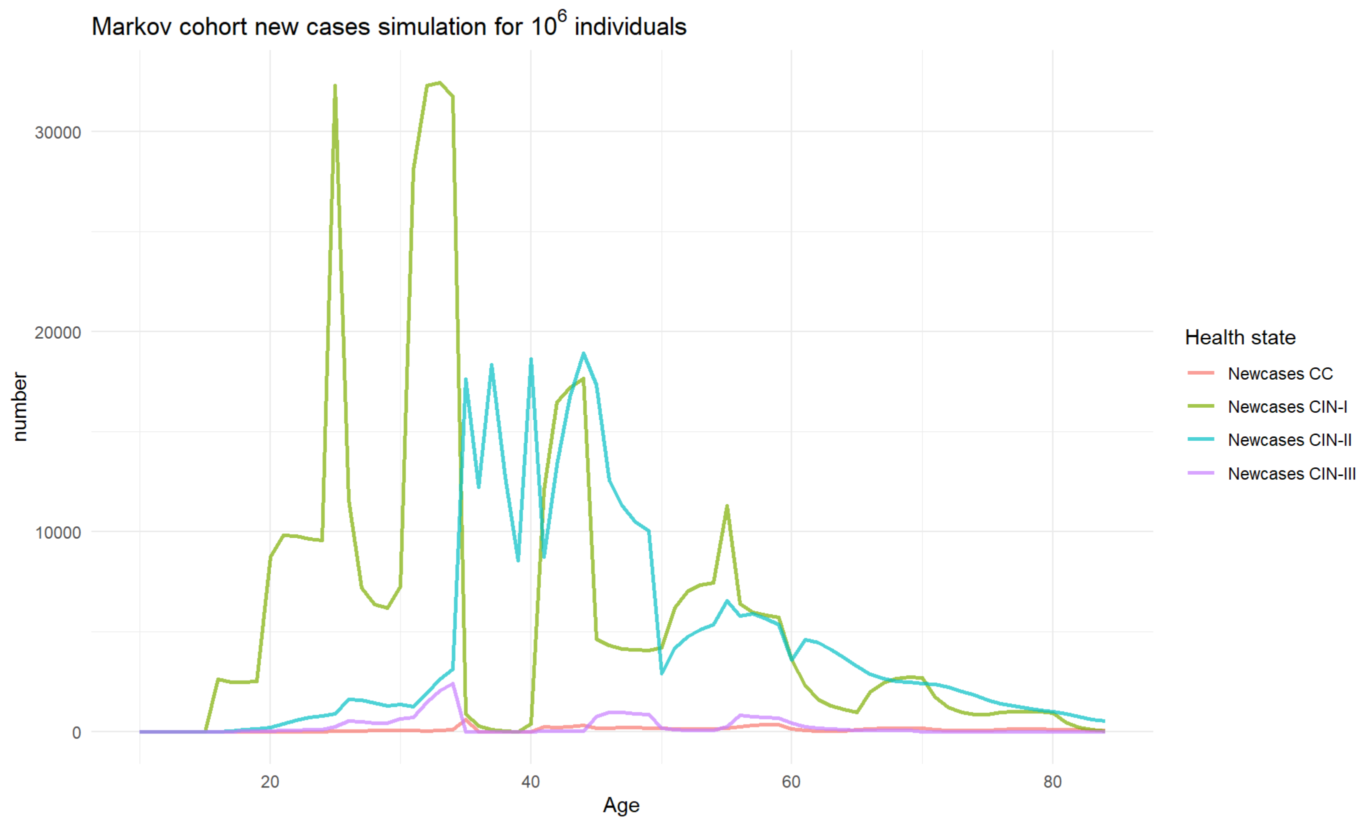
```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## ℹ Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```


Markov cohort simulation for 10^6 individuals
Value
4e+05
3e+05
2e+05
1e+05
0e+00
Age
20
40
60
80
Health state
CC_Death
CIN1
CIN2
CIN3
FIGO.I
FIGO.II
FIGO.III
FIGO.IV
HR.HPV.infection
Survival


Microsimulation for 10^6 individuals
Value
4e+05
3e+05
2e+05
1e+05
0e+00
Age
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
colour
CC_Death
CIN1
CIN2
CIN3
FIGO.I
FIGO.II
FIGO.III
FIGO.IV
Infected



Now I compare the new cases of some microsimulation model states to Markov cohort model with the same probabilistic structure.



```
## [1] "HR.HPV.infection->CIN1"
## [1] "CIN2->CIN1"
## [1] "CIN3->CIN1"
## [1] "CIN1->CIN2"
## [1] "HR.HPV.infection->CIN2"
## [1] "CIN3->CIN2"
```

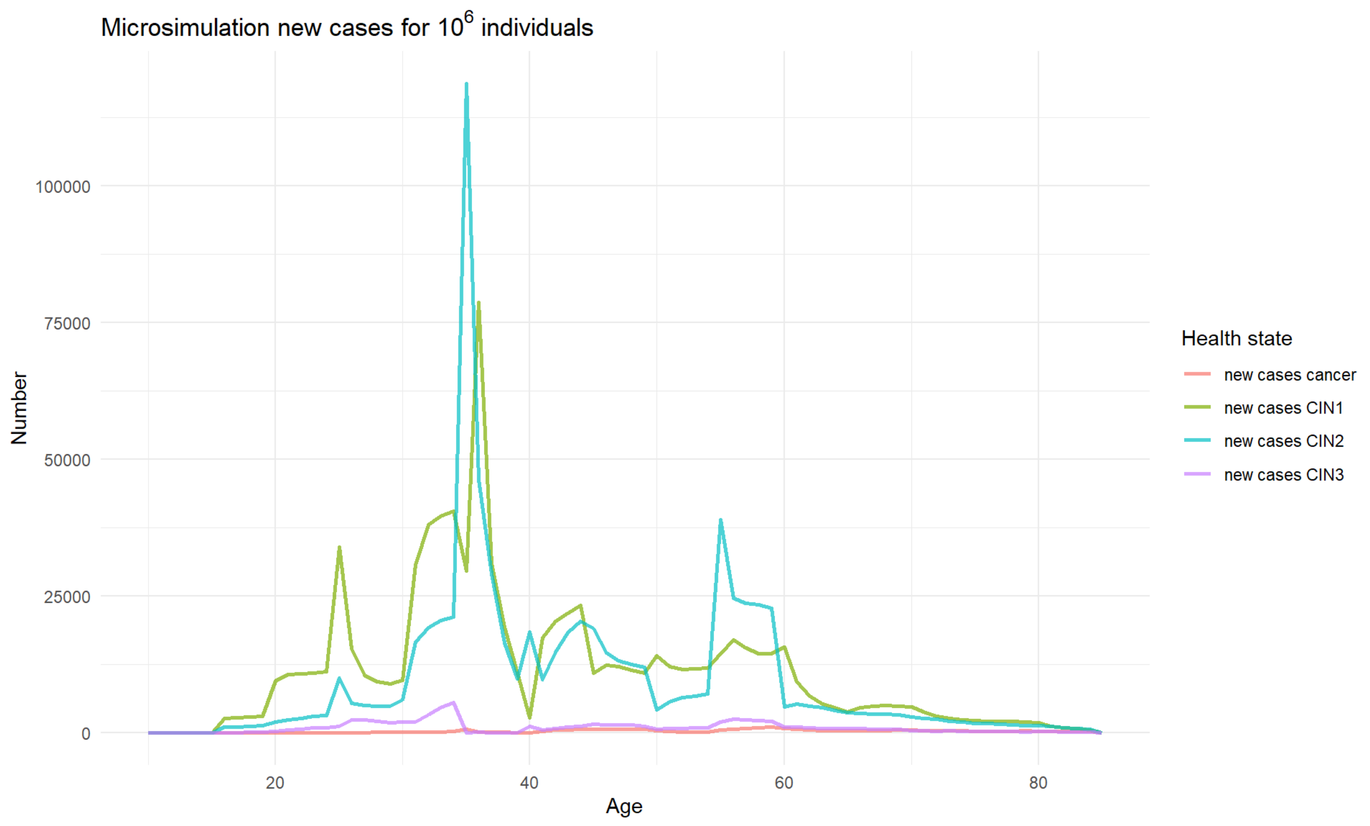
```
## [1] "CIN1->CIN3"
## [1] "CIN2->CIN3"
## [1] "FIGO.I->CIN3"
```

```
## [1] "CIN3->FIGO.I"
```

```
## [1] "FIGO.I->FIGO.II"
## [1] "FIGO.III->FIGO.II"
```

```
## [1] "FIGO.II->FIGO.III"
## [1] "FIGO.I->FIGO.III"
## [1] "FIGO.IV->FIGO.III"
```

```
## [1] "FIGO.III->FIGO.IV"
```


Microsimulation new cases for 10^6 individuals
Number
Age
Health state
new cases cancer
new cases CIN1
new cases CIN2
new cases CIN3