# Calibrating a 3-state cancer model

## Random search using Latin-Hypercube Sampling

The DARTH workgroup

Developed by the Decision Analysis in R for Technologies in Health (DARTH) workgroup:

Fernando Alarid-Escudero, PhD (1)

Eva A. Enns, MS, PhD (2)

M.G. Myriam Hunink, MD, PhD (3,4)

Hawre J. Jalal, MD, PhD (5)

Eline M. Krijkamp, MSc (3)

Petros Pechlivanoglou, PhD (6)

Alan Yang, MSc (7)

In collaboration of:

1. Drug Policy Program, Center for Research and Teaching in Economics (CIDE) - CONACyT, Aguascalientes, Mexico
2. University of Minnesota School of Public Health, Minneapolis, MN, USA
3. Erasmus MC, Rotterdam, The Netherlands
4. Harvard T.H. Chan School of Public Health, Boston, USA
5. University of Pittsburgh Graduate School of Public Health, Pittsburgh, PA, USA
6. The Hospital for Sick Children, Toronto and University of Toronto, Toronto ON, Canada
7. The Hospital for Sick Children, Toronto ON, Canada

Please cite our publications when using this code:

- Alarid-Escudero F, Maclehose RF, Peralta Y, Kuntz KM, Enns EA. Non-identifiability in model calibration and implications for medical decision making. Med Decis Making. 2018; 38(7):810-821.

- Jalal H, Pechlivanoglou P, Krijkamp E, Alarid-Escudero F, Enns E, Hunink MG. An Overview of R in Health Decision Sciences. Med Decis Making. 2017; 37(3): 735-746. https://journals.sagepub.com/doi/abs/10.1177/0272989X16686559

A walkthrough of the code could be found in the follwing link: - https://darth-git.github.io/calibSMDM2018-materials/

Change `eval` to `TRUE` if you want to knit this document.

```
rm(list = ls())      # clear memory (removes all the variables from the workspace)
```

# 00 Calibration Specifications

Model: 3-State Cancer Relative Survival (CRS) Markov Model

Inputs to be calibrated: p_Mets, p_DieMets

Targets: Surv

Calibration method: Random search using Latin-Hypercube Sampling

Goodness-of-fit measure: Sum of Log-Likelihood

# 01 Load packages

```
if (!require('pacman')) install.packages('pacman'); library(pacman) # use this package to conveniently
# load (install if required) packages from CRAN
p_load("lhs", "plotrix", "psych")
```

# 02 Load target data

```
load("CRS_CalibTargets.RData")
lst_targets <- CRS_targets

# Plot the targets

# TARGET 1: Survival ("Surv")
plotrix::plotCI(x = lst_targets$Surv$time, y = lst_targets$Surv$value,
                ui = lst_targets$Surv$ub,
                li = lst_targets$Surv$lb,
                ylim = c(0, 1),
                xlab = "Time", ylab = "Pr Survive")

# TARGET 2: (if you had more...)
# plotrix::plotCI(x = lst_targets$Target2$time, y = lst_targets$Target2$value,
#                  ui = lst_targets$Target2$ub,
#                  li = lst_targets$Target2$lb,
#                  ylim = c(0, 1),
#                  xlab = "Time", ylab = "Target 2")
```

# 03 Load model as a function

```
# - inputs are parameters to be estimated through calibration
# - outputs correspond to the target data

source("CRS_MarkovModel_Function.R") # creates the function run_crs_markov()

# Check that it works
v_params_test <- c(p_Mets = 0.10, p_DieMets = 0.05)
run_crs_markov(v_params_test) # It works!
```

# 04 Specify calibration parameters

```
# Specify seed (for reproducible sequence of random numbers)
set.seed(072218)

# number of random samples
n_samp <- 1000

# names and number of input parameters to be calibrated
v_param_names <- c("p_Mets","p_DieMets")
n_param <- length(v_param_names)

# range on input search space
lb <- c(p_Mets = 0.04, p_DieMets = 0.04) # lower bound
ub <- c(p_Mets = 0.16, p_DieMets = 0.16) # upper bound

# number of calibration targets
v_target_names <- c("Surv")
n_target <- length(v_target_names)
```

# 05 Calibrate!

```
# record start time of calibration
t_init <- Sys.time()

###  Generate a random sample of input values  ###

# Sample unit Latin Hypercube
m_lhs_unit <- randomLHS(n_samp, n_param)

# Rescale to min/max of each parameter
m_param_samp <- matrix(nrow=n_samp,ncol=n_param)
for (i in 1:n_param){
  m_param_samp[,i] <- qunif(m_lhs_unit[,i],
                           min = lb[i],
                           max = ub[i])
}
colnames(m_param_samp) <- v_param_names
```

```r
# view resulting parameter set samples
pairs.panels(m_param_samp)


###  Run the model for each set of input values ###

# initialize goodness-of-fit vector
m_GOF <- matrix(nrow = n_samp, ncol = n_target)
colnames(m_GOF) <- paste0(v_target_names, "_fit")

# loop through sampled sets of input values
for (j in 1:n_samp){

  ###  Run model for a given parameter set  ###
  model_res <- run_crs_markov(v_params = m_param_samp[j, ])


  ###  Calculate goodness-of-fit of model outputs to targets  ###

  # TARGET 1: Survival ("Surv")
  # log likelihood
  m_GOF[j,1] <- sum(dnorm(x = lst_targets$Surv$value,
                       mean = model_res$Surv,
                       sd = lst_targets$Surv$se,
                       log = T))

  # weighted sum of squared errors (alternative to log likelihood)
  # w <- 1/(lst_targets$Surv$se^2)
  # m_GOF[j,1] <- -sum(w*(lst_targets$Surv$value - v_res)^2)


  # TARGET 2: (if you had more...)
  # log likelihood
  # m_GOF[j,2] <- sum(dnorm(x = lst_targets$Target2$value,
  #                         mean = model_res$Target2,
  #                         sd = lst_targets$Target2$se,
  #                         log = T))


} # End loop over sampled parameter sets


###  Combine fits to the different targets into single GOF  ###
# can give different targets different weights
v_weights <- matrix(1, nrow = n_target, ncol = 1)
# matrix multiplication to calculate weight sum of each GOF matrix row
v_GOF_overall <- c(m_GOF%*%v_weights)
# Store in GOF matrix with column name "Overall"
m_GOF <- cbind(m_GOF,Overall_fit=v_GOF_overall)

# Calculate computation time
comp_time <- Sys.time() - t_init
```

## 06 Exploring best-fitting input sets

```
# Arrange parameter sets in order of fit
m_calib_res <- cbind(m_param_samp,m_GOF)
m_calib_res <- m_calib_res[order(-m_calib_res[,"Overall_fit"]),]

# Examine the top 10 best-fitting sets
m_calib_res[1:10,]

# Plot the top 100 (top 10%)
plot(m_calib_res[1:100,1],m_calib_res[1:100,2],
     xlim=c(lb[1],ub[1]),ylim=c(lb[2],ub[2]),
     xlab = colnames(m_calib_res)[1],ylab = colnames(m_calib_res)[2])

# Pairwise comparison of top 100 sets
pairs.panels(m_calib_res[1:100,v_param_names])

### Plot model-predicted output at best set vs targets ###
v_out_best <- run_crs_markov(m_calib_res[1,])

# TARGET 1: Survival ("Surv")
plotrix::plotCI(x = lst_targets$Surv$time, y = lst_targets$Surv$value,
                ui = lst_targets$Surv$ub,
                li = lst_targets$Surv$lb,
                ylim = c(0, 1),
                xlab = "Time", ylab = "Pr Survive")
points(x = lst_targets$Surv$time,
       y = v_out_best$Surv,
       pch = 8, col = "red")
legend("topright",
       legend = c("Target", "Model-predicted output"),
       col = c("black", "red"), pch = c(1, 8))

# TARGET 2: (if you had more...)
# plotrix::plotCI(x = lst_targets$Target2$time, y = lst_targets$Target2$value,
#                 ui = lst_targets$Target2$ub,
#                 li = lst_targets$Target2$lb,
#                 ylim = c(0, 1),
#                 xlab = "Time", ylab = "Target 2")
# points(x = lst_targets$Target2$time,
#        y = v_out_best$Target2,
#        pch = 8, col = "red")
# legend("topright",
#        legend = c("Target", "Model-predicted output"),
#        col = c("black", "red"), pch = c(1, 8))
```