

Calibrating a 3-state cancer model

Incremental mixture importance sampling (IMIS)

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. Division of Public Administration, Center for Research and Teaching in Economics (CIDE), 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. <https://pubmed.ncbi.nlm.nih.gov/30248276/>
- 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 following link: - <https://darth-git.github.io/calibSMDM2018-materials/>

Copyright 2017, THE HOSPITAL FOR SICK CHILDREN AND THE COLLABORATING INSTITUTIONS. All rights reserved in Canada, the United States and worldwide. Copyright, trademarks, trade names and any and all associated intellectual property are exclusively owned by THE HOSPITAL FOR SICK CHILDREN and the collaborating institutions. These materials may be used, reproduced, modified, distributed and adapted with proper attribution.

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: Incremental mixture importance sampling (IMIS)

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", "IMIS", "matrixStats", "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$Surv$sub,  
                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$Target2$sub,  
#                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_resamp <- 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 Calibration functions

```

# Write function to sample from prior
sample_prior <- function(n_samp){
  m_lhs_unit <- randomLHS(n = n_samp, k = n_param)
  m_param_samp <- matrix(nrow = n_samp, ncol = n_param)
  colnames(m_param_samp) <- v_param_names
  for (i in 1:n_param){
    m_param_samp[, i] <- qunif(m_lhs_unit[,i],
                             min = lb[i],
                             max = ub[i])
    # ALTERNATIVE prior using beta (or other) distributions
    # m_param_samp[, i] <- qbeta(m_lhs_unit[,i],
    #                             shape1 = 1,
    #                             shape2 = 1)
  }
  return(m_param_samp)
}

```

```

# view resulting parameter set samples
pairs.panels(sample_prior(1000))

### PRIOR ###
# Write functions to evaluate log-prior and prior

# function that calculates the log-prior
calc_log_prior <- function(v_params){
  if(is.null(dim(v_params))) { # If vector, change to matrix
    v_params <- t(v_params)
  }
  n_samp <- nrow(v_params)
  colnames(v_params) <- v_param_names
  lprior <- rep(0, n_samp)
  for (i in 1:n_param){
    lprior <- lprior + dunif(v_params[, i],
                           min = lb[i],
                           max = ub[i],
                           log = T)
    # ALTERNATIVE prior using beta distributions
    # lprior <- lprior + dbeta(v_params[, i],
    #                          shape1 = 1,
    #                          shape2 = 1,
    #                          log = T)
  }
  return(lprior)
}
calc_log_prior(v_params = v_params_test)
calc_log_prior(v_params = sample_prior(10))

# function that calculates the (non-log) prior
calc_prior <- function(v_params) {
  exp(calc_log_prior(v_params))
}
calc_prior(v_params = v_params_test)
calc_prior(v_params = sample_prior(10))

### LIKELIHOOD ###
# Write functions to evaluate log-likelihood and likelihood

# function to calculate the log-likelihood
calc_log_lik <- function(v_params){
  # par_vector: a vector (or matrix) of model parameters
  if(is.null(dim(v_params))) { # If vector, change to matrix
    v_params <- t(v_params)
  }
  n_samp <- nrow(v_params)
  v_llik <- matrix(0, nrow = n_samp, ncol = n_target)
  llik_overall <- numeric(n_samp)
  for(j in 1:n_samp) { # j=1

```

```

jj <- tryCatch( {
  ### Run model for parametr set "v_params" ###
  model_res <- run_crs_markov(v_params[j, ])

  ### Calculate log-likelihood of model outputs to targets ###
  # TARGET 1: Survival ("Surv")
  # log likelihood
  v_llik[j, 1] <- sum(dnorm(x = lst_targets$Surv$value,
                           mean = model_res$Surv,
                           sd = lst_targets$Surv$se,
                           log = T))

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

  # OVERALL
  llik_overall[j] <- sum(v_llik[j, ])
}, error = function(e) NA)
if(is.na(jj)) { llik_overall <- -Inf }
} # End loop over sampled parameter sets
# return LLIK
return(llik_overall)
}
calc_log_lik(v_params = v_params_test)
calc_log_lik(v_params = sample_prior(10))

# function to calculate the (non-log) likelihood
calc_likelihood <- function(v_params){
  exp(calc_log_lik(v_params))
}
calc_likelihood(v_params = v_params_test)
calc_likelihood(v_params = sample_prior(10))

### POSTERIOR ###
# Write functions to evaluate log-posterior and posterior

# function that calculates the log-posterior
calc_log_post <- function(v_params) {
  lpost <- calc_log_prior(v_params) + calc_log_lik(v_params)
  return(lpost)
}
calc_log_post(v_params = v_params_test)
calc_log_post(v_params = sample_prior(10))

# function that calculates the (non-log) posterior
calc_post <- function(v_params) {

```

```

    exp(calc_log_post(v_params))
  }
calc_post(v_params = v_params_test)
calc_post(v_params = sample_prior(10))

```

06 Calibrate!

```

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

#### Bayesian calibration using IMIS ####
# define three functions needed by IMIS: prior(x), likelihood(x), sample.prior(n)
prior <- calc_prior
likelihood <- calc_likelihood
sample.prior <- sample_prior

# run IMIS
fit_imis <- IMIS(B = 1000, # the incremental sample size at each iteration of IMIS
                B.re = n_resamp, # the desired posterior sample size
                number_k = 10, # the maximum number of iterations in IMIS
                D = 0)

# obtain draws from posterior
m_calib_res <- fit_imis$resample

# Calculate log-likelihood (overall fit) and posterior probability of each sample
m_calib_res <- cbind(m_calib_res,
                    "Overall_fit" = calc_log_lik(m_calib_res[,v_param_names]),
                    "Posterior_prob" = calc_post(m_calib_res[,v_param_names]))

# normalize posterior probability
m_calib_res[, "Posterior_prob"] <- m_calib_res[, "Posterior_prob"]/sum(m_calib_res[, "Posterior_prob"])

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

```

07 Exploring posterior distribution

```

# Plot the 1000 draws from the posterior
v_post_color <- scales::rescale(m_calib_res[, "Posterior_prob"])
plot(m_calib_res,
     xlim = c(lb[1], ub[1]), ylim = c(lb[2], ub[2]),
     xlab = v_param_names[1], ylab = v_param_names[2],
     col = scales::alpha("black", v_post_color))
# add center of Gaussian components
points(fit_imis$center, col = "red", pch = 8)
legend("topright", c("Draws from posterior", "Center of Gaussian components"),
     col = c("black", "red"), pch = c(1, 8))

```

```

# Plot the 1000 draws from the posterior with marginal histograms
pairs.panels(m_calib_res[,v_param_names])

# Compute posterior mean
v_calib_post_mean <- colMeans(m_calib_res[,v_param_names])
v_calib_post_mean

# Compute posterior median and 95% credible interval
m_calib_res_95cr <- colQuantiles(m_calib_res[,v_param_names], probs = c(0.025, 0.5, 0.975))
m_calib_res_95cr

# Compute maximum-a-posteriori (MAP) parameter set
v_calib_map <- m_calib_res[which.max(m_calib_res[, "Posterior_prob"]),]

### Model-predicted output at mode/MAP ###
v_out_best <- run_crs_markov(v_calib_map[v_param_names])

```

07.1 Model predicted outputs at maximum-a-posteriori (MAP)

```

### Plot model-predicted output at mode vs targets ###
# set plot margins (helps plotting)
par(mar=c(5,4,4,4))
# 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))

```

07.2 Distribution of model-predicted outputs

```
### Distribution of model-predicted output at mode vs targets ###
## Initialize matrix to store outputs
m_out_post <- matrix(NA,
                    nrow = n_resamp,
                    ncol = length(v_out_best$Surv))
## Iterate model over all parameter sets from posterior distribution
for(i in 1:n_resamp){
  l_out <- run_crs_markov(m_calib_res[i, ])
  m_out_post[i, ] <- l_out$Surv

  if(i/(n_resamp/10) == round(i/(n_resamp/10),0)) { # display progress every 10%
    cat('\r', paste(i/n_resamp * 100, "% done", sep = " "))
  }
}

## Compute model-predicted posterior summary statistics
# Model-predicted posterior mean
v_out_post_mean <- colMeans(m_out_post)
# Model-predicted posterior credible interval
m_out_post_intervals <- colQuantiles(m_out_post, probs = c(0.025, 0.975))
```

07.2.1 Plot model-predicted output at mode vs targets

```
# set plot margins (helps plotting)
par(mar=c(5,4,4,4))
# TARGET 1: Survival ("Surv")
plotrix::plotCI(x = lst_targets$Surv$time, y = lst_targets$Surv$value,
               ui = lst_targets$Surv$sub,
               li = lst_targets$Surv$lb,
               ylim = c(0, 1),
               xlab = "Time", ylab = "Pr Survive")
points(x = lst_targets$Surv$time,
       y = v_out_post_mean,
       pch = 8, col = "red")
lines(x = lst_targets$Surv$time,
      y = m_out_post_intervals[, 1],
      col = "blue")
lines(x = lst_targets$Surv$time,
      y = m_out_post_intervals[, 2],
      col = "blue")
legend("topright",
      legend = c("Target",
                  "Model-predicted posterior mean",
                  "Model-predicted 95% posterior CrI"),
      col = c("black", "red", "blue"),
      pch = c(1, 8, NA),
      lty = c(NA, NA, 1))

# 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))

```