Calibrating the Sick-Sicker 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. 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: Sick-Sicker 4-state Markov Model

Inputs to be calibrated: p_S1S2, hr_S1, hr_S2

Targets: Surv, Prev, PropSick

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", "plotrix", "psych", "scatterplot3d", "IMIS", "matrixStats")
# install_github("DARTH-git/darthtools", force = TRUE) Uncomment if there is a newer version
p_load_gh("DARTH-git/darthtools")
```

02 Load target data

```
load("SickSicker_CalibTargets.RData")
lst_targets <- SickSicker_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: Prevalence ("Prev")
plotrix::plotCI(x = lst_targets$Prev$time, y = lst_targets$Prev$value,
                ui = lst_targets$Prev$ub,
                li = lst_targets$Prev$lb,
                ylim = c(0, 1),
                xlab = "Time", ylab = "Prev")

# TARGET 3: Proportion who are Sick ("PropSick"), among all those afflicted (Sick+Sicker)
plotrix::plotCI(x = lst_targets$PropSick$time, y = lst_targets$PropSick$value,
                ui = lst_targets$PropSick$ub,
```

```
                li = lst_targets$PropSick$lb,
                ylim = c(0, 1),
                xlab = "Time", ylab = "PropSick")
```

03 Load model as a function

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

# creates the function run_sick_sicker_markov()
source("SickSicker_MarkovModel_Function.R")

# Check that it works
v_params_test <- c(p_S1S2 = 0.105, hr_S1 = 3, hr_S2 = 10)
run_sick_sicker_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_S1S2","hr_S1","hr_S2")
n_param <- length(v_param_names)

# range on input search space
lb <- c(p_S1S2 = 0.01, hr_S1 = 1.0, hr_S2 = 5)  # lower bound
ub <- c(p_S1S2 = 0.50, hr_S1 = 4.5, hr_S2 = 15) # upper bound

# number of calibration targets
v_target_names <- c("Surv", "Prev", "PropSick")
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
```

```r
# 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_sick_sicker_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: "Prev"
      # log likelihood
      v_llik[j,2] <- sum(dnorm(x = lst_targets$Prev$value,
                               mean = model_res$Prev,
                               sd = lst_targets$Prev$se,
                               log = T))

      # TARGET 3: "PropSick"
      # log likelihood
      v_llik[j,3] <- sum(dnorm(x = lst_targets$PropSick$value,
                               mean = model_res$PropSick,
                               sd = lst_targets$PropSick$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)
```

```r
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!

```r
# 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"])
```

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

07 Exploring best-fitting input sets

```r
# Plot the 1000 draws from the posterior
v_post_color <- scales::rescale(m_calib_res[,"Posterior_prob"])
s3d <- scatterplot3d(x = m_calib_res[, 1],
                     y = m_calib_res[, 2],
                     z = m_calib_res[, 3],
                     color = scales::alpha("black", v_post_color),
                     xlim = c(lb[1],ub[1]), ylim = c(lb[2],ub[2]), zlim = c(lb[3],ub[3]),
                     xlab = v_param_names[1], ylab = v_param_names[2], zlab = v_param_names[3])
# add center of Gaussian components
s3d$points3d(fit_imis$center, col = "red", pch = 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"]),]


### Plot model-predicted output at best set vs targets ###
v_out_best <- run_sick_sicker_markov(v_calib_map[v_param_names])

# 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: "Prev"
plotrix::plotCI(x = lst_targets$Prev$time, y = lst_targets$Prev$value,
                ui = lst_targets$Prev$ub,
                li = lst_targets$Prev$lb,
```

```
                ylim = c(0, 1),
                xlab = "Time", ylab = "Prev")
points(x = lst_targets$Prev$time,
       y = v_out_best$Prev,
       pch = 8, col = "red")
legend("topright",
       legend = c("Target", "Model-predicted output"),
       col = c("black", "red"), pch = c(1, 8))

# TARGET 3: "PropSick"
plotrix::plotCI(x = lst_targets$PropSick$time, y = lst_targets$PropSick$value,
                ui = lst_targets$PropSick$ub,
                li = lst_targets$PropSick$lb,
                ylim = c(0, 1),
                xlab = "Time", ylab = "PropSick")
points(x = lst_targets$PropSick$time,
       y = v_out_best$PropSick,
       pch = 8, col = "red")
legend("topright",
       legend = c("Target", "Model-predicted output"),
       col = c("black", "red"), pch = c(1, 8))
```