Calibrating a 3-state cancer model

Directed search using Nelder-mead

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/

```r
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: Directed search using Nelder-mead

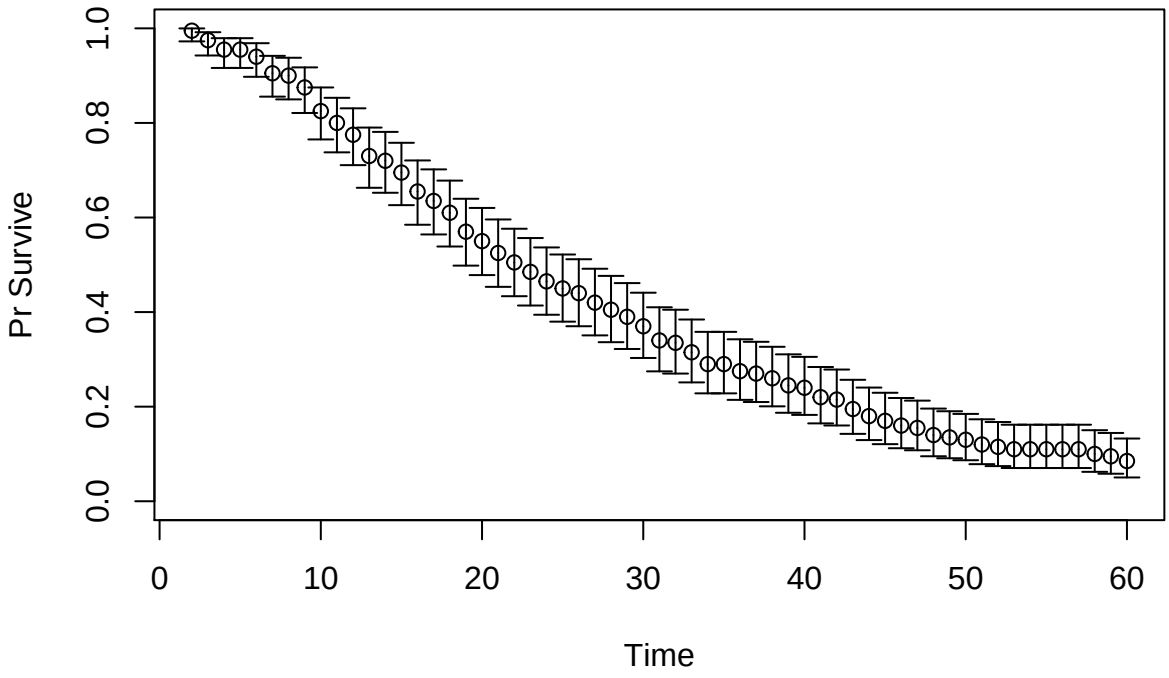
Goodness-of-fit measure: Sum of Log-Likelihood

01 Load packages

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

02 Load target data

```r
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!
```

```
## $Surv
##          2          3          4          5          6          7          8
## 0.99500000 0.98575000 0.97291250 0.95707188 0.93874278 0.91837769 0.89637365
##          9         10         11         12         13         14         15
## 0.87307833 0.84879544 0.82378959 0.79829064 0.77249758 0.74658203 0.72069133
##         16         17         18         19         20         21         22
## 0.69495132 0.66946885 0.64433400 0.61962203 0.59539519 0.57170426 0.54859000
```

```
##         23         24         25         26         27         28         29
## 0.52608435 0.50421161 0.48298935 0.46242937 0.44253844 0.42331901 0.40476980
##         30         31         32         33         34         35         36
## 0.38688637 0.36966161 0.35308613 0.33714867 0.32183639 0.30713521 0.29303003
##         37         38         39         40         41         42         43
## 0.27950495 0.26654348 0.25412871 0.24224343 0.23087030 0.21999193 0.20959096
##         44         45         46         47         48         49         50
## 0.19965017 0.19015255 0.18108132 0.17242001 0.16415250 0.15626300 0.14873618
##         51         52         53         54         55         56         57
## 0.14155706 0.13471112 0.12818429 0.12196293 0.11603386 0.11038433 0.10500206
##         58         59         60
## 0.09987521 0.09499237 0.09034259
```

04 Specify calibration parameters

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

# number of initial starting points
n_init <- 100

# 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

```r
# Write goodness-of-fit function to pass to Nelder-Mead algorithm
f_gof <- function(v_params){

  # Run model for parametr set "v_params"
  model_res <- run_crs_markov(v_params)

  # Calculate goodness-of-fit of model outputs to targets
  v_GOF <- numeric(n_target)
  # TARGET 1: Survival ("Surv")
  # log likelihood
  v_GOF[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_GOF[2] <- sum(dnorm(x = lst_targets$Target2$value,
  #                       mean = model_res$Target2,
  #                       sd = lst_targets$Target2$se,
  #                       log = T))

  # OVERALL
  # can give different targets different weights
  v_weights <- rep(1,n_target)
  # weighted sum
  GOF_overall <- sum(v_GOF[1:n_target] * v_weights)

  # return GOF
  return(GOF_overall)
}
```

06 Calibrate!

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

###  Sample multiple random starting values for Nelder-Mead  ###
v_params_init <- matrix(nrow=n_init, ncol=n_param)
for (i in 1:n_param){
  v_params_init[,i] <- runif(n_init,min=lb[i],max=ub[i])
}
colnames(v_params_init) <- v_param_names

###  Run Nelder-Mead for each starting point  ###
m_calib_res <- matrix(nrow = n_init, ncol = n_param+1)
colnames(m_calib_res) <- c(v_param_names, "Overall_fit")
for (j in 1:n_init){

  ### use optim() as Nelder-Mead ###
  fit_nm <- optim(v_params_init[j,], f_gof,
                  control = list(fnscale = -1, # switches from minimization to maximization
                                 maxit = 1000), hessian = T)
  m_calib_res[j,] <- c(fit_nm$par,fit_nm$value)

  ### to use a simulated annealing instead ###
  # fit_sa <- optim(v_params_init[j,], f_gof,
  #                 method = c("SANN"),  # switches to using simulated annealing
  #                 control = list(temp = 10, tmax = 10, # algorithm tuning parameters
  #                                fnscale = -1, maxit = 1000),
  #                 hessian = T)
  # m_calib_res[j,] = c(fit_sa$par,fit_sa$value)

  ### to use a genetic algorithm instead ###
  # library(DEoptim)
  # f_fitness <- function(params){
  #   names(params) = v_param_names
  #   return(-f_gof(params))}
  # fit_ga = DEoptim(f_fitness, lower=lb, upper=ub)
```

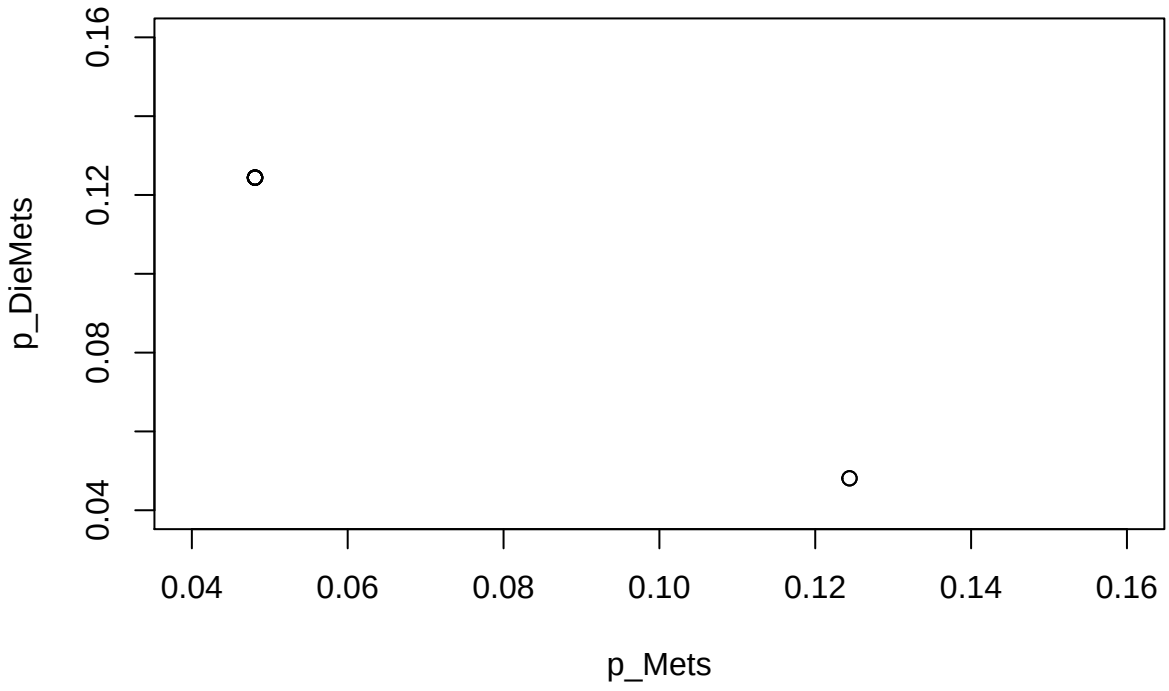
```
  # m_calib_res[j,] = c(fit_ga$optim$bestmem,-1*fit_ga$optim$bestval)

  }

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

07 Exploring best-fitting input sets

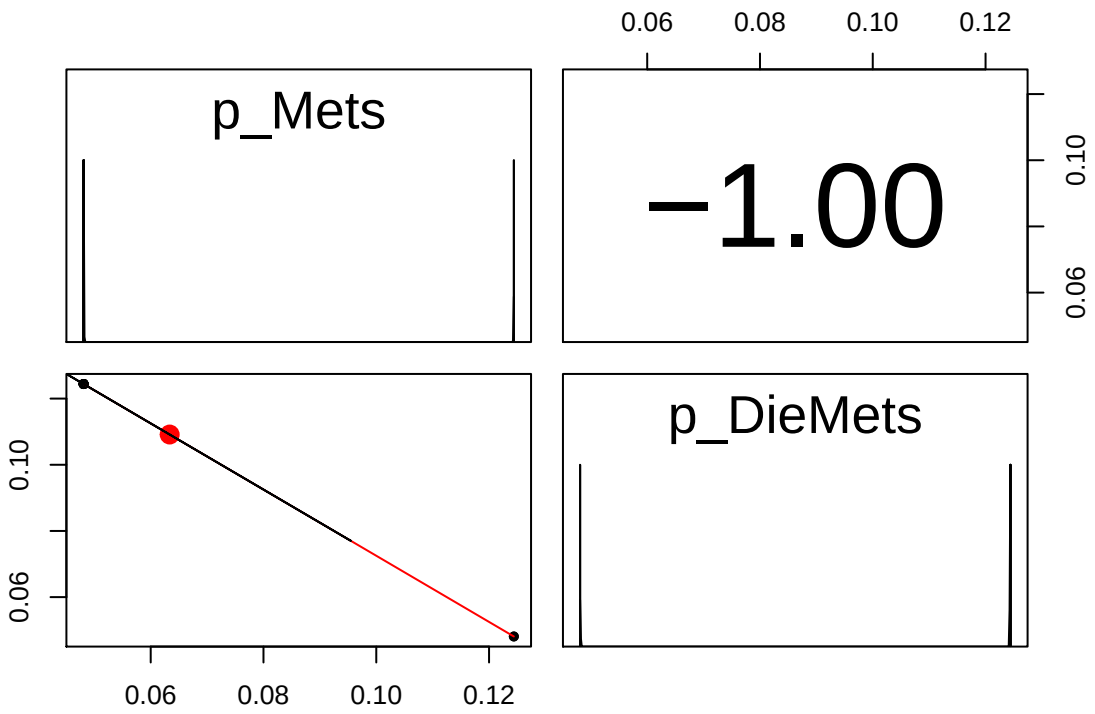
```
# Arrange parameter sets in order of fit
m_calib_res <- m_calib_res[order(-m_calib_res[,"Overall_fit"]),]

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

```
##           p_Mets  p_DieMets Overall_fit
##  [1,] 0.04810740 0.12439706    156.0328
##  [2,] 0.04810703 0.12439834    156.0328
##  [3,] 0.12439619 0.04810733    156.0328
##  [4,] 0.04810708 0.12439961    156.0328
##  [5,] 0.04810765 0.12439450    156.0328
##  [6,] 0.04810695 0.12439797    156.0328
##  [7,] 0.04810655 0.12440104    156.0328
##  [8,] 0.04810789 0.12439263    156.0328
##  [9,] 0.12439411 0.04810754    156.0328
## [10,] 0.04810769 0.12439731    156.0328
```

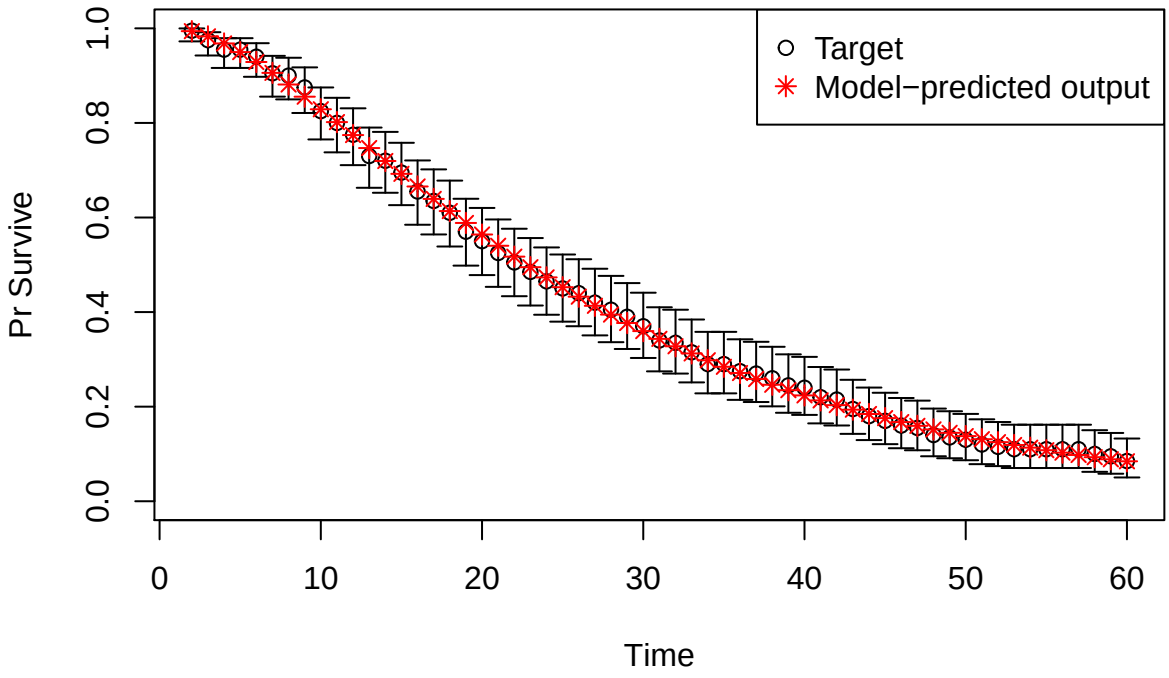
```
# Plot the top 10 (top 10%)
plot(m_calib_res[1:10,1],m_calib_res[1:10,2],
     xlim=c(lb[1],ub[1]),ylim=c(lb[2],ub[2]),
     xlab = colnames(m_calib_res)[1],ylab = colnames(m_calib_res)[2])
```




```
# Pairwise comparison of top 10 sets
pairs.panels(m_calib_res[1:10,v_param_names])
```




```
### Plot model-predicted output at mean 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))
```