
Web Appendix E: R code

The following code is provided for implementing the proposed GM-IVC model in R. It follows the parametrization described in our paper.

```
#############################################################
# Description:
# Script to fit Generalized multi-index varying coefficient models with alpha_0, 
# alpha_1, gamma_0, gamma_1, beta_1, beta_2 and beta_3.
#
# Author: JPE
# event risk dynamics over time in older patients on dialysis: A generalized multi- 
# index varying coefficient model approach.
#
# R 3.0.1
#############################################################

##### Step 1: Data Description:
# Load as dataset
# Long format
# Dimension: N x 8 where N is the total number of longitudinal measurements
# column 1: id: subject id
# column 2: age: age
# column 3: t: time response is observed from the start of the study
# column 4: y: response variable
# column 5: tsfi: indicator = 1 if response observed after first infection
# (or event of interest) and indicator = 0 otherwise
# column 6: zi: time since first infection, or event of interest
# (zero if subject does not experience an infection)
# column 7: x1: covariate
# column 8: x2: covariate
names(dataset) <- c("id","age","t","y","tsfi","zi","x1","x2")

##### Step 2: Bin subjects by age and fit a generalized varying coefficient 
# model in each bin
# Select age group, we select every two years here.
ageLeftEP <- c(65,67,69,71,73,75,77,79,81,83,85,87)
ageRightEP <- c(67,69,71,73,75,77,79,81,83,85,87,90)
# Get the number of age groups
numAgeGroups <- length(ageLeftEP)

# Initialize a vector for the age stratified estimates of alpha0 and alpha1
alpha0Stratified <- c()
alpha1Stratified <- c()

# Define a grid and bandwidth to estimate alpha0 and alpha1
gridAlpha <- seq(0,5,.25)
nGrid <- length(gridAlpha)
h <- 1.5

for(i in 1:numAgeGroups) {
  # Select ith age group
  ageGroup <- subset(dataset, (dataset$age >= ageLeftEP[i] &
    dataset$age < ageRightEP[i]))
  a0 <- c()
  a1 <- c()
  for(j in 1:nGrid) {
    t0 <- gridAlpha[j]
    # Ignore all data that is not within one bandwidth from the time point t0
    vcmTemp <- subset(ageGroup, abs(ageGroup$t - t0) < h)
    firstCol <- (vcmTemp$tsfi == 0)*1
    tDiff <- vcmTemp$t - t0
    secondCol <- firstCol*tDiff
    thirdCol <- vcmTemp$tsfi
    fourthCol <- thirdCol*tDiff
    # Construct the data set to feed into the VCM algorithm
    vcmData <- cbind(vcmTemp$id, vcmTemp$y, vcmTemp$t, firstCol, secondCol,
                     thirdCol,fourthCol, vcmTemp$zi, vcmTemp$x1, vcmTemp$x2)
    est <- getVCMEstimates(vcmData, rep(0,7), t0, h, 12, .001)
    # Concatenate grid point estimates into a vector 1 x nGrid
    a0 <- c(a0, est[1])
    a1 <- c(a1, est[3])
  }
  # Construct matrix 7 x nGrid for the stratified estimates
  alpha0Stratified <- rbind(alpha0Stratified, a0)
  alpha1Stratified <- rbind(alpha1Stratified, a1)
}

##### Step 3: Normalize varying coefficient estimates
sumEstimatesTSD <- apply(alpha0Stratified,2,sum)
alpha0Estimates <- sumEstimatesTSD / sqrt(getArea(gridAlpha,sumEstimatesTSD^2))
sumEstimatesTSFI <- apply(alpha1Stratified,2,sum)
alpha1Estimates <- sumEstimatesTSFI / sqrt(getArea(gridAlpha,sumEstimatesTSFI^2))

##### Step 4: Fit generalized varying coefficient model in follow-up time
# with fitted $\hat{\alpha}_0(t)$, $\hat{\alpha}_1(t)$

# Get alpha estimates at each time point/record
dataset$alpha <- (dataset$tsfi == 0)*alpha0Estimates[dataset$t/.25+1] +
  (dataset$tsfi == 1)*alpha1Estimates[dataset$t/.25+1]

# Age grid to estimate age varying functions
gridGamma <- seq(65, 90, .50)
nGrid <- length(gridGamma)

# Matrix 5 x nGrid to store age varying function estimates
betaEstimates <- c()

h <- 6
for(j in 1:nGrid) {
  a0 <- gridGamma[j]
  # Ignore all data that is not within one bandwidth from the age point a0
  vcmTemp <- subset(dataset, abs(dataset$age - a0) < h)
  firstCol <- (vcmTemp$tsfi == 0)*vcmTemp$alpha
  tDiff <- vcmTemp$age - a0
  secondCol <- firstCol*tDiff
  thirdCol <- vcmTemp$tsfi*vcmTemp$alpha
  fourthCol <- thirdCol*tDiff
  # Construct the data set to feed into the VCM algorithm
  vcmData <- cbind(vcmTemp$id, vcmTemp$y, vcmTemp$age, firstCol,
                   secondCol, thirdCol, fourthCol, vcmTemp$zi, vcmTemp$zi*tDiff,
                   vcmTemp$x1, vcmTemp$x1*tDiff, vcmTemp$x2, vcmTemp$x2*tDiff)
  est <- getVCMEstimates(vcmData, rep(0,10), a0, h, 12, .001)
  betaEstimates <- cbind(betaEstimates, est[c(1,3,5,7,9)])
}

#### Final estimate vectors are alpha0Estimates (for alpha_0),
# alpha1Estimates (for alpha_1), betaEstimates[1,] (for gamma_0),
# betaEstimates[2,] (for gamma1), betaEstimates[3,] (for beta1),
# betaEstimates[4,] (for beta2) and betaEstimates[5,] (for beta3)

# R functions used above
# ----------------------------------------------------------------
# DESCRIPTION : Generalized Varying Coefficient Fitting Algorithm
# INPUT
# vcmData : Stacked Augmented matrix J x (3 + k) in the form
#   id|y|t|chi where J is the total number of longitudinal measurements and k is the number of columns in chi
# Note: The stacked augmented matrix must be ordered by id and by time t within id.
# initEst : Vector 1 x k of initial estimates
# t0 : Scalar grid point
# h : Scalar bandwidth
# maxIter : Scalar maximum number of iterations allowed
# epsilon: Scalar tolerance level used to define convergence
#   (typical value .001)
#
# OUTPUT
# Vector 1 x k of real numbered parameter estimates
# ----------------------------------------------------------------
getVCMEstimates <- function(vcmData, initEst, t0, h, maxIter, epsilon) {
  k <- dim(vcmData)[2]
  # Number of parameters to estimate
  numEst <- k-3
  # Initial parater estimates at t0
  betaMatrix <- matrix(initEst,ncol=1,nrow=numEst)
  # This flag is true until we have convergence
  flag = "true"
  # Loop counter
  l <- 0
  while(flag == "true" & l < maxIter) {
    l <- l + 1
    estimates <- betaMatrix
    betaMatrix <- matrix(rep(0,numEst),ncol=1,nrow=numEst)
    # This will give the number of time points for each patient. Note: The table
    # function will order by id.
    nIntervals <- as.numeric(table(vcmData[,1]))
    # Number of unique ids
    numSubjects <- length(nIntervals)
    hmatrices <- matrix(rep(0,numEst^2), nrow=numEst, ncol=numEst)
    vmatrices <- matrix(rep(0,numEst), numEst, 1)
    # Starting index of the first patient
    startIndex <- 1
    for(i in 1:numSubjects) {
      ni <- nIntervals[i]
      endIndex <- startIndex+ni-1
      dataTemp <- matrix(vcmData[startIndex:endIndex,],nrow=ni,ncol=k)
      chi <- matrix(dataTemp[,c(-1,-2,-3)],nrow=ni,ncol=numEst)
      tDiff <- dataTemp[,3] - rep(t0,ni)
      # Linear predictor in the model
      lpred <- chi%*%estimates
      pijs <- 1/(1+exp(-1*lpred))
      kernels <- 0.75*(1-(tDiff/h)^2)
      kernels <- (kernels > 0)*kernels/h
      yijs <- c(dataTemp[,2] - pijs)
      w1<-diag(ni,x=as.vector(kernels*pijs*(1-pijs)))
      Hi <- t(chi)%*%w1%*%chi
      w2<-diag(ni,x=as.vector(kernels))
      Vi <- t(chi)%*%w2%*%yijs
      hi <- hmatrices
      hi[i,i] <- ((pijs*yijs) - t(chi)%*%w1%*%hi[i,])
      hi[i,i] <- ((pijs*yijs) - t(chi)%*%w1%*%hi[i,])
      v[i] <- v[i] - t(chi)%*%w2%*%yijs
    }
  }
}
hmatrices <- hmatrices + Hi
vmatrices <- vmatrices + Vi
startIndex <- startIndex + ni

betaMatrix <- estimates + solve(hmatrices)%*%vmatrices
# Checks that the functional estimates all differ by no more than epsilon
# from the previous estimates.
if(sum((abs(estimates-betaMatrix) >= epsilon)*1) == 0) {
  flag = "false"
} else {
  if(l == maxIter) {
    stop("*** No Convergence ***")
  }
}
}
return(betaMatrix)

# DESCRIPTION : Riemann approximation to the integral using the
# trapezoidal rule.
# INPUT
# x : Vector of the ordered elements of the domain that define the
#     endpoints of the trapezoids.
# y : The corresponding range elements of the function being
#     integrated.
# OUTPUT
# Scalar real numbered Riemann approximation to the integral.
# ----------------------------------------------------------------
getArea <- function(x,y) {
  if(length(x) != length(y)) {
    stop("The number of elements in the domain and range must equal.")
  }
  if(sum(sort(x) != x) > 0) {
    stop("The endpoints of the trapezoids must be ordered.")
  }
  k <- length(x)
  return(1/2*sum((y[-k]+y[-1])*(x[-1]-x[-k])))
}