PL-GVCM Algorithm:

```
##############################################################################
# File: PLVCM_algorithm_JASA.pdf
#
# Description:
# Script to fit partly conditional and fully conditional partially
# linear generalized varying coefficient models to a simulated data set.
#
# Author: JPE
# Estes, JP, Nguyen, DV, Dalrymple, LS, Mu Y, Senturk, D (2014)
# Time-Varying Effect Modeling with Longitudinal Data Truncated by
# Death: Conditional Models, Interpretations and Inference.
##############################################################################

library(MASS)
library(mvtnorm)

# ------------------------------------------------------------------------ #
# Read in data generated according to the main simulation model          #
# of Section 4.1 at n=50000 and 80% truncation by death                  #
# for easy implementation of both partly and fully conditional PL-GVCM  #
# ------------------------------------------------------------------------ #
##### Data Description:
# Load as dataset
# Long format
# Dimension: N x 8 where N is the total number of longitudinal response
#            measurements
# column 1: id: subject id
# column 2: t: time response is observed from the start of the study
# column 3: y: binary response variable
# column 4: tsfi: indicator = 1 if response observed after first infection
#           (or event of interest) and indicator = 0 otherwise
# column 5: zi: time since first infection, or event of interest (zero if
#           subject does not experience an infection)
# column 6: x1: covariate
# column 7: x2: covariate
# column 8: si: survival time
dataset <- read.table("mainsim50000.txt",header=TRUE)

# Assign column names and order data set by  id, tsfi, and time
names(dataset) <- c("id","t","y","tsfi","zi","x1","x2","si")
```

```
dataset <- dataset[order(dataset$id,dataset$tsfi, dataset$t), ]

# ------------------------------------------------------------------- #
# DESCRIPTION : Inverse Logit Function                                #
# INPUT                                                               #
# x : Scalar or vector                                                #
#                                                                     #
# OUTPUT                                                              #
# Inverse logit of the scalar or vector (component-wise)              #
# ------------------------------------------------------------------- #
invLogit <- function(x) {
  return(1/(1+exp(-1*x)))
}

# ------------------------------------------------------------------- #
# DESCRIPTION : Get Varying Coefficient Function Estimates at t = t0  #
#               (binary outcome)                                      #
# 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        #
# initEst: Vector 1 x k of initial estimates of all parameters        #
#          to be estimated at time t0                                 #
# betaEst: Vector 1 x 3 of estimates of the constant                  #
#          coefficients in the model. Note: This parameter            #
#          is null in step one of the fitting algorithm               #
# betaCov: Stacked matrix J x 3 with columns zi | x1 | x2. Note: This #
#          parameter is null in step one of the fitting algorithm     #
# t0: Scalar grid point                                               #
# h: Scalar bandwidth                                                 #
# maxIter: Scalar maximum number of iterations allowed                #
# epsilon: Scalar tolerance level used to define convergence          #
#          (typical value .0001)                                      #
#                                                                     #
# OUTPUT                                                              #
# Vector 1 x 2 of real numbered varying coefficient function          #
# estimates at t = t0                                                 #
# ------------------------------------------------------------------- #
getVCFEstimates <- function(vcmData, initEst,
                            betaEst, betaCov, t0, h, maxIter, epsilon) {
  k <- dim(vcmData)[2]
  # Number of parameters to estimate
```

```
numEst <- k-3
if(numEst != length(initEst)) {
  stop("Incompatible dimensions")
}
# Initial varying coefficient function estimates at t0
betaMatrix <- matrix(data=initEst,nrow=numEst,ncol=1)
# 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),nrow=numEst,ncol=1)
  # 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
    if(!is.null(betaEst) & !is.null(betaCov)) {
      lpred <- lpred+
        betaCov[startIndex:endIndex,]%*%matrix(data=betaEst,nrow=3,ncol=1)
    }
    pijs <- invLogit(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
      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[c(1,3)])
}

# ------------------------------------------------------------------- #
# DESCRIPTION : Partially Linear Generalized Varying Coefficient Model  #
#               Fitting Algorithm (binary outcome)                      #
# INPUT                                                                 #
# dataset: Stacked Augmented matrix J x (4 + k) in the form             #
#          id|y|t|chi|si where J is the total number of longitudinal    #
#          measurements and k is the number of columns in chi           #
# initVCFEst: Vector 4 x nGrid of initial estimates of the varying      #
#             coefficient functions and their first derivative at       #
#             time t0 where nGrid is the length of the grid             #
# initBetaEst: Vector 1 x 3 of initial estimates of the constant        #
#              coefficient functions                                    #
# gridAlpha: The grid over which the varying coefficient functions      #
#            will be estimated. Note: If the grid points don't coincide #
#            with the measurement time points, then step 2 will need    #
#            to be slightly modified                                    #
# h: Scalar bandwidth                                                   #
# maxIter: Scalar maximum number of iterations allowed                  #
# epsilon: Scalar tolerance level used to define convergence            #
#          (typical value .0001)                                        #
#                                                                       #
# OUTPUT                                                                #
```

```
# estList: List of coefficient estimates                                  #
# List item 1: Estimate of the first varying coefficient function         #
# List item 2: Estimate of the second varying coefficient function        #
# List item 3: 1x3 vector of the estimates of the constant coefficients   #
# ------------------------------------------------------------------------ #
getVCMEstimates <- function(dataset, initVCFEst,
                            initBetaEst, gridAlpha, h, maxIter, epsilon) {

  nGrid <- length(gridAlpha)
  # Fit the PL-GVCM: Step 1
  alpha0hat <- c()
  alpha1hat <- c()
  for(k in 1:nGrid) {
    t0 <- gridAlpha[k]
    # Ignore all data that is not within one bandwidth from the time point t0
    datasetTemp <- subset(dataset, abs(dataset$t - t0) < h)
    firstCol <- (datasetTemp$tsfi == 0)*1
    tDiff <- datasetTemp$t - t0
    secondCol <- firstCol*tDiff
    thirdCol <- datasetTemp$tsfi
    fourthCol <- thirdCol*tDiff
    # Construct the data set to feed into the VCM algorithm
    vcmData <- cbind(datasetTemp$id, datasetTemp$y, datasetTemp$t, firstCol,
                     secondCol, thirdCol, fourthCol, datasetTemp$zi, datasetTemp$x1,
                     datasetTemp$x2)
    est <- getVCFEstimates(vcmData, c(initVCFEst[,k],initBetaEst), NULL, NULL,
           t0, h, maxIter, epsilon)
    # Concatenate grid point estimates into a vector 1 x nGrid
    alpha0hat <- c(alpha0hat, est[1])
    alpha1hat <- c(alpha1hat, est[2])
  }

  # Fit PL-GVCM Step 2
  dataSplit <- split(dataset, dataset$id)
  n <- length(dataSplit)

  diff <- 1
  estimates <- matrix(data=initBetaEst,nrow=3,ncol=1)
  counter <- 0
  while(diff > epsilon) {
    counter <- counter + 1
    H <- matrix(rep(0,9),nrow=3,ncol=3)
```

```
  V <- matrix(rep(0,3), nrow=3, ncol=1)

  for(i in 1:n) {
    dataTemp <- dataSplit[[i]]
    xi <- cbind(dataTemp$zi,dataTemp$x1,dataTemp$x2)
    yi <- dataTemp$y
    # Note: This assumes that the grid points are 3 months apart
    # and coincide with the measurement times
    ind <- dataTemp$t/.25+1
    lpred <- xi%*%estimates+alpha0hat[ind]*(dataTemp$tsfi==0)+
             alpha1hat[ind]*(dataTemp$tsfi==1)
    pi <- invLogit(lpred)
    k <- length(pi)
    wi <- as.matrix(diag(as.vector(pi*(1-pi)),nrow=k,ncol=k))
    H <- H + t(xi)%*%wi%*%xi
    V <- V + t(xi)%*%(yi-pi)
  }

  lastEstimates <- estimates
  estimates <- lastEstimates + solve(H)%*%V
  diff <- max(abs(estimates - lastEstimates))

}

# Fit PL-GVCM Step 3
alpha0hat <- c()
alpha1hat <- c()
for(k in 1:nGrid) {
  t0 <- gridAlpha[k]
  # Ignore all data that is not within one bandwidth from the time point t0
  datasetTemp <- subset(dataset, abs(dataset$t - t0) < h)
  firstCol <- (datasetTemp$tsfi == 0)*1
  tDiff <- datasetTemp$t - t0
  secondCol <- firstCol*tDiff
  thirdCol <- datasetTemp$tsfi
  fourthCol <- thirdCol*tDiff
  # Construct the data set to feed into the VCM algorithm
  vcmData <- cbind(datasetTemp$id, datasetTemp$y, datasetTemp$t, firstCol,
             secondCol, thirdCol, fourthCol)
  betaCov <- cbind(datasetTemp$zi, datasetTemp$x1, datasetTemp$x2)
  est <- getVCFEstimates(vcmData, initVCFEst[,k], estimates, betaCov, t0, h,
         maxIter, epsilon)
```

```
    # Concatenate grid point estimates into a vector 1 x nGrid
    alpha0hat <- c(alpha0hat, est[1])
    alpha1hat <- c(alpha1hat, est[2])
  }

  estList <- list()
  estList[[1]] <- alpha0hat
  estList[[2]] <- alpha1hat
  estList[[3]] <- estimates

  return(estList)

}

# ------------------------------------------------------------------- #
# Partly Conditional PL-GVCM Fit                                      #
# ------------------------------------------------------------------- #

# Define the grid to estimate the varying coefficient functions
gridAlpha <- seq(0,5,.25)
numGrid <- length(gridAlpha)
# Get the coefficient estimates in the PL-GVCM
vcmEstimates <- getVCMEstimates(dataset, matrix(rep(0,4*numGrid),nrow=4,
                ncol=numGrid),rep(0,3), gridAlpha, 1.5, 12, .00001)


alpha0hat <- vcmEstimates[[1]]
alpha1hat <- vcmEstimates[[2]]
betaEstimates <- round(vcmEstimates[[3]],4)

# ------------------------------------------------------------------- #
# Graph the varying coefficient function estimates                    #
# ------------------------------------------------------------------- #
ytemp <- c(alpha0hat,alpha1hat)
par(mfrow=c(1,1),mar=c(4,5,3,1), oma=c(0,0,0,2))
plot(gridAlpha, alpha0hat, ylim=c(min(ytemp),max(ytemp)), type="n",
main="Varying Coefficient Function Estimates", xlab=expression(paste(t[0],
" or ",t[1])), ylab=expression(paste(hat(alpha["0,P"])(t[0])," or ",
hat(alpha)["1,P"])(t[1])), xaxs="i",cex.lab=1.2, cex.axis=1,
cex.main=1.4, cex=1)
lines(gridAlpha, alpha0hat)
lines(gridAlpha, alpha1hat, col="grey")
```

```
legend(.05,-1.75,c("Before infection", "After infection"), cex=1.2,
col=c("black","grey"), lty=c(1,1),lwd=1, bty="n");

#Estimates of the constant coefficients:
print(paste("Beta 0: ",betaEstimates[1]))
print(paste("Beta 1: ",betaEstimates[2]))
print(paste("Beta 2: ",betaEstimates[3]))


# ------------------------------------------------------------------ #
# Fully Conditional PL-GVCM Fit                                      #
# ------------------------------------------------------------------ #

# ------------------------------------------------------------------ #
# Construct four subsets by conditioning on survival times           #
# lying in three month intervals with left endpoints 2 year,         #
# 3 years, 4 years, and 5 years respectively.                        #
# ------------------------------------------------------------------ #

datasetF2 <- subset(dataset, (dataset$si > 2 & dataset$si < 2.25))
datasetF3 <- subset(dataset, (dataset$si > 3 & dataset$si < 3.25))
datasetF4 <- subset(dataset, (dataset$si > 4 & dataset$si < 4.25))
datasetF5 <- subset(dataset, (dataset$si > 5 & dataset$si < 5.25))

gridAlphaF2 <- seq(0,2,.25)
nGridF2 <- length(gridAlphaF2)
vcmEstimatesF2 <- getVCMEstimates(datasetF2, matrix(rep(0,4*nGridF2),nrow=4,
ncol=nGridF2), rep(0,3), gridAlphaF2, .75, 12, .00001)
alpha0hatF2 <- vcmEstimatesF2[[1]]
alpha1hatF2 <- vcmEstimatesF2[[2]][-1*nGridF2]
betaEstimatesF2 <- round(vcmEstimatesF2[[3]],4)

gridAlphaF3 <- seq(0,3,.25)
nGridF3 <- length(gridAlphaF3)
vcmEstimatesF3 <- getVCMEstimates(datasetF3, matrix(rep(0,4*nGridF3),nrow=4,
ncol=nGridF3), rep(0,3), gridAlphaF3, 1, 12, .00001)
alpha0hatF3 <- vcmEstimatesF3[[1]]
alpha1hatF3 <- vcmEstimatesF3[[2]][-1*nGridF3]
betaEstimatesF3 <- round(vcmEstimatesF3[[3]],4)

gridAlphaF4 <- seq(0,4,.25)
nGridF4 <- length(gridAlphaF4)
```

```
vcmEstimatesF4 <- getVCMEstimates(datasetF4, matrix(rep(0,4*nGridF4),nrow=4,
ncol=nGridF4), rep(0,3), gridAlphaF4, 1.25, 12, .00001)
alpha0hatF4 <- vcmEstimatesF4[[1]]
alpha1hatF4 <- vcmEstimatesF4[[2]][-1*nGridF4]
betaEstimatesF4 <- round(vcmEstimatesF4[[3]],4)

gridAlphaF5 <- seq(0,5,.25)
nGridF5 <- length(gridAlphaF5)
vcmEstimatesF5 <- getVCMEstimates(datasetF5, matrix(rep(0,4*nGridF5),nrow=4,
ncol=nGridF5), rep(0,3), gridAlphaF5, 1.5, 12, .00001)
alpha0hatF5 <- vcmEstimatesF5[[1]]
alpha1hatF5 <- vcmEstimatesF5[[2]][-1*nGridF5]
betaEstimatesF5 <- round(vcmEstimatesF5[[3]],4)

# ------------------------------------------------------------------- #
# Graph the varying coefficient function estimates                    #
# ------------------------------------------------------------------- #

yTempF <- c(alpha0hatF2, alpha1hatF2, alpha0hatF3, alpha1hatF3,
alpha0hatF4, alpha1hatF4, alpha0hatF5, alpha1hatF5)
maxY <- max(yTempF)
par(mfrow=c(2,2),mar=c(4,5,3,1), oma=c(0,0,0,2))
plot(gridAlphaF2, alpha0hatF2, type="n", ylim=c(.98*min(yTempF), 1.02*maxY),
main=expression(paste("(a) ",D[j]," Midpoint = 1.125")),
xlab=expression(paste(t[0]," or ",t[1])),
ylab=expression(paste(hat(alpha)["0j,F"](t[0])," or ", hat(alpha)["1j,F"](t[1]))),
xaxs="i", yaxs="i", cex.lab=1.2, cex.axis=1, cex.main=1.2, cex=1)
lines(gridAlphaF2, alpha0hatF2)
lines(gridAlphaF2[-1*nGridF2], alpha1hatF2, col="grey")
legend(.02*2, maxY, c("Before infection", "After infection"), cex=1.2,
col=c("black","grey"), lty=c(1,1), lwd=1, bty="n");
print(paste("Year 2 - Beta 0: ",betaEstimatesF2[1]))
print(paste("Year 2 - Beta 1: ",betaEstimatesF2[2]))
print(paste("Year 2 - Beta 2: ",betaEstimatesF2[3]))

plot(gridAlphaF3, alpha0hatF3, type="n", ylim=c(.98*min(yTempF), 1.02*maxY),
main=expression(paste("(b) ",D[j]," Midpoint = 2.125")),
xlab=expression(paste(t[0]," or ",t[1])),
ylab=expression(paste(hat(alpha)["0j,F"](t[0])," or ", hat(alpha)["1j,F"](t[1]))),
xaxs="i", yaxs="i", cex.lab=1.2, cex.axis=1, cex.main=1.2, cex=1)
lines(gridAlphaF3, alpha0hatF3)
lines(gridAlphaF3[-1*nGridF3], alpha1hatF3, col="grey")
```

```
legend(.02*3, maxY, c("Before infection", "After infection"), cex=1.2,
col=c("black","grey"), lty=c(1,1), lwd=1, bty="n");
print(paste("Year 3 - Beta 0: ",betaEstimatesF3[1]))
print(paste("Year 3 - Beta 1: ",betaEstimatesF3[2]))
print(paste("Year 3 - Beta 2: ",betaEstimatesF3[3]))

plot(gridAlphaF4, alpha0hatF4, type="n", ylim=c(.98*min(yTempF), 1.02*maxY),
main=expression(paste("(c) ",D[j]," Midpoint = 3.125")),
xlab=expression(paste(t[0]," or ",t[1])),
ylab=expression(paste(hat(alpha)["0j,F"](t[0])," or ", hat(alpha)["1j,F"](t[1]))),
xaxs="i", yaxs="i", cex.lab=1.2, cex.axis=1, cex.main=1.2, cex=1)
lines(gridAlphaF4, alpha0hatF4)
lines(gridAlphaF4[-1*nGridF4], alpha1hatF4, col="grey")
legend(.02*4, maxY, c("Before infection", "After infection"), cex=1.2,
col=c("black","grey"), lty=c(1,1), lwd=1, bty="n");
print(paste("Year 4 - Beta 0: ",betaEstimatesF4[1]))
print(paste("Year 4 - Beta 1: ",betaEstimatesF4[2]))
print(paste("Year 4 - Beta 2: ",betaEstimatesF4[3]))

plot(gridAlphaF5, alpha0hatF5, type="n", ylim=c(.98*min(yTempF), 1.02*maxY),
main=expression(paste("(d) ",D[j]," Midpoint = 4.125")),
xlab=expression(paste(t[0]," or ",t[1])),
ylab=expression(paste(hat(alpha)["0j,F"](t[0])," or ", hat(alpha)["1j,F"](t[1]))),
xaxs="i", yaxs="i", cex.lab=1.2, cex.axis=1, cex.main=1.2, cex=1)
lines(gridAlphaF5, alpha0hatF5)
lines(gridAlphaF5[-1*nGridF5], alpha1hatF5, col="grey")
legend(.02*5, maxY, c("Before infection", "After infection"), cex=1.2,
col=c("black","grey"), lty=c(1,1), lwd=1, bty="n");
print(paste("Year 5 - Beta 0: ",betaEstimatesF5[1]))
print(paste("Year 5 - Beta 1: ",betaEstimatesF5[2]))
print(paste("Year 5 - Beta 2: ",betaEstimatesF5[3]))
```