Supplemntary material - code

Léa Orsini

2023-10-23

```
library(tidyverse)
library(survival)
library(pseudo)
library(geepack)
library(MASS)
library(spBayesSurv)
library(rstan)
rstan_options(auto_write = T)
```

Data generation

```
set.seed(1)
n = 500 #sample size

betab <- -0.3 # treatment effect
Zb<-rep(c(0,1), each = n/2) # treatment variable

#Weibull parameters
shape = 0.6 #must be >0
scale = exp(-betab*Zb/shape) #must be >0

#Uniform parameter
theta<-8.543879 # To have ~20% of censored patients

T_tilde <- rweibull(n, shape = shape, scale = scale) # Weibull distribution for event times

C <- runif(n, 0, theta) # Uniform distribution for censoring times

time <- pmin(T_tilde, C)
event <- as.numeric(time ==T_tilde)

simu <- data.frame(patID = 1:n, # patient ID
                   betab = betab,
                   Zb = Zb,
                   time = time,
                   event = event)
```

Data Analysis

Cox model

```
### COX model
res <- summary(survival::coxph(formula = survival::Surv(time, event) ~Zb, data = simu))

COX_beta_hat <- res$coefficients[1,1];COX_beta_hat
```

```
## [1] -0.4428448
```

```
COX_se_hat <- res$coefficients[1,3]; COX_se_hat
```

```
## [1] 0.1023121
```

Generalized Estimating Equations based on pseudo-observations

```
### Defining the time points used to compute pseudo-observations
K = 5 # Number of time points


event_times <- simu$time[simu$event == 1]
# take the quantile of the event time
cutoffs <- quantile(event_times, seq(0, 1, length.out = K+2)[2:(K+1)])

### Compute the pseudo-observations
pseudo <- pseudo::pseudosurv(simu$time, simu$event, tmax=cutoffs)

#arrange the data
b <- NULL
for(it in 1:length(pseudo$time)){
  b <- rbind(b,cbind(simu,pseudo = pseudo$pseudo[,it],
                     tpseudo = pseudo$time[it],id=1:nrow(simu)))
}

b <- b[order(b $id),]

b$ipseudo <- 1-b$pseudo # needed because cloglog in geese is defined as log(-log(1-y))
b$event<- as.factor(b$event)

### Fit a GEE model
fit_GEE <- geepack::geese(ipseudo ~ as.factor(Zb) + as.factor(tpseudo),
                          id = id, data = b, scale.fix=TRUE, family=gaussian, jack = TRUE,
                          mean.link = "cloglog", corstr="independence")

as.data.frame(cbind(mean = fit_GEE$beta,
                    SE_ajust = sqrt(diag(fit_GEE$vbeta.ajs)),
                    SE_sandwich = sqrt(diag(fit_GEE$vbeta))))
```

```
##                                        mean   SE_ajust SE_sandwich
## (Intercept)                      -1.7928849 0.13119262  0.13174281
## as.factor(Zb)1                   -0.4015238 0.11558334  0.11593787
## as.factor(tpseudo)0.187811938275434  0.7963982 0.09646799  0.09692905
## as.factor(tpseudo)0.479508800946491  1.3188706 0.11215453  0.11269243
## as.factor(tpseudo)0.94444250554837   1.7687568 0.12056400  0.12114560
```

```
## as.factor(tpseudo)1.74276093976556   2.2041333 0.12672607  0.12734317
```

Frequentist generalized method of moments based on pseudo-observations

```
### R function to implement the freq GMM with a cloglog link function
gmm_cloglog <- function (formula = formula(data), id = id, data = parent.frame(), b = NULL,
                         tol = 1e-8, maxiter = 1000, family = gaussian, corstr = "independence"){
  call <- match.call()
  m <- match.call(expand.dots = FALSE)
  m$b <- m$tol <- m$maxiter <- m$link <- m$varfun <- m$corstr <- m$family <- m$invfun <- NULL

  if (is.null(m$id))
    m$id <- as.name("id")

  m[[1]] <- as.name("model.frame")
  m <- eval(m, parent.frame())
  Terms <- attr(m, "terms")
  y <- as.matrix(model.extract(m, "response"))
  x <- model.matrix(Terms, m)

  # get class id
  id <- model.extract(m, id)

  # get number of observe persons
  nobs <- nrow(x)

  # get number of parameters, include (intercept)
  np <- ncol(x)

  # get the name of X matrix colomn (variables)
  xnames <- dimnames(x)[[2]]

  if (is.character(family))
    family <- get(family)
  if (is.function(family))
    family <- family()

  # Starting parameters: if b is not NULL then sign b to beta; otherwise, use gml to get initial beta
  if (!is.null(b)) {
    beta <- matrix(as.double(b), ncol = 1)
    if (nrow(beta) != np) {
      stop("Dim beta != ncol(x)")
    }
  }
  else {
    # message("\n","running glm to get initial regression estimate")
    mm <- match.call(expand.dots = FALSE)
    mm$b <- mm$tol <- mm$maxiter <- mm$link <- mm$varfun <- mm$corstr <- mm$id <- mm$invfun <- NULL
    mm[[1]] <- as.name("glm")
    beta <- eval(mm, parent.frame())$coef

    beta <- as.numeric(beta)
  }
  if (length(id) != length(y))
```

```
  stop("Id and y not same length")

#get the maximum number of iteration
maxiter <- as.integer(maxiter)

# Correlation structure variable CORR must be one of IND, EXCH or AR-1
corstrs <- c("independence", "exchangeable", "AR-1")

dist <- family$family

### start sign value to calculation ###
#### define:
# x- X matrix,
# y- response matrix
# beta - coefficients of X variables
# id- id list
# uid- unique id
# nobs- number of rep time
# nsub- number of unique id
# np - number of parameter
# m0 - m0 matrix, nobs*nobs matrix, depend on correlation structure
# m1 - m1 matrix, nobs*nobs matrix, depend on correlation structure

# U - np*1 matrix
# C - np*np matrix
# U - np*1 matrix
# C - np*np matrix
# ui - np*1 matrix for i-th id
# Ufirstdev - np*np matrix for first derivative of U
# firstdev - np*np matrix of first derivative

# for each id:
# xi- i-th x
# yi - i-th y
# ni- number of row of xi

# mui- i-th mu
# mui_dev - derivate of mui
# vui - marginal variance for i-th id

### Calculation

y <- as.matrix(y)
x <- as.matrix(x)

obs <- lapply(split(id, id), "length")
nobs <- as.numeric(obs)
nsub <- length(nobs)
np <- dim(x)[[2]]

#################  GMM iteration #######
betadiff <- 1
iteration <- 0
```

```
betanew <- beta

# GMM iteration
while(betadiff > tol && iteration < maxiter)
{

  # initial value
  beta <- betanew
  if (corstr == "independence") {
    U <- matrix(rep(0,np),nrow=np)
    C <- matrix(rep(0,np*np),nrow=np)
    ui <- matrix(rep(0,np),nrow=np)
    Ufirstdev <- matrix(rep(0,np*np),nrow=np)
    firstdev <- matrix(rep(0,np*np),nrow=np)
  }
  else {
    U <- matrix(rep(0,2*np),nrow=2*np)
    C <- matrix(rep(0,2*np*2*np),nrow=2*np)
    ui <- matrix(rep(0,2*np),nrow=2*np)
    Ufirstdev <- matrix(rep(0,2*np*np),nrow=2*np)
    firstdev <- matrix(rep(0,2*np*np),nrow=2*np)
  }
  # one iteration




  # loc1 - start location for row in xi
  # loc2 - end location for row in xi
  loc1 <- 0
  loc2 <- 0
  for (i in 1:nsub)
  {
    # set start location for next xi
    loc1 <- loc2+1
    loc2 <- loc1+nobs[i]-1

    yi <- as.matrix(y[loc1:loc2,])
    xi <- x[loc1:loc2,]
    ni <- nrow(yi)

    # set m0, m1
    m0 <- diag(ni)
    # set m1 by corr structure
    if (corstr == "independence") {
      m1 <- matrix(rep(0,ni*ni),ni)
    }
    else if (corstr == "exchangeable") {
      m1 <- matrix(rep(1,ni*ni),ni) - m0
    }
    else if (corstr == "AR-1") {
      m1 <- matrix(rep(0,ni*ni),ni)
      for (k in 1:ni) {
```

```
      for (l in 1:ni) {
        if (abs(k-l)==1) m1[k,l] <-1
      }
    }
  }

  # change ui, mui_dev, vui depending on distribution
  if (dist == "gaussian") {
    mui <- exp(-exp(xi %*% beta)) # using a cloglog link function
    mui_dev <- diag(as.vector(-exp(xi%*%beta)))%*%diag(as.vector(mui))
    vui <- diag(ni)     # identity matrix for marginal variance (gaussian case)
  }

  # calculate mui, wi, zi, c, C, U, di, firstdev, Ufirstdev
  # depending on corr structure
  if (corstr == "independence") {
    wi <- t(xi) %*% mui_dev %*% vui %*% m0 %*% vui
    ui0 <- (1/nsub)*wi %*% (yi-mui)
    ui[1:np,] <- ui0
    C <- C + ui %*% t(ui)
    U <- U + ui

    di0 <- -(1/nsub) * wi %*% mui_dev %*% xi
    firstdev[1:np,] <- di0
    Ufirstdev <- Ufirstdev + firstdev
  }


  else {
    wi <- t(xi) %*% mui_dev %*% vui %*% m0 %*% vui
    zi <- t(xi) %*% mui_dev %*% vui %*% m1 %*% vui

    ui0 <- (1/nsub)*wi %*% (yi-mui)
    ui1 <- (1/nsub)*zi %*% (yi-mui)

    ui[1:np,] <- ui0
    ui[(np+1):(2*np),] <- ui1

    C <- C + ui %*% t(ui)
    U <- U + ui

    if (is.na(C[1,1])) {
      print(iteration) # Commented out printing outside of print() methods
      print(ui)
      print(C)
    }

    di0 <- -(1/nsub) * wi %*% mui_dev %*% xi
    di1 <- -(1/nsub) * zi %*% mui_dev %*% xi

    firstdev[1:np,] <- di0
    firstdev[(np+1):(2*np),] <- di1
    Ufirstdev <- Ufirstdev + firstdev
```

```
      }
    }

    # calculate Q, betanew,
    Cinv=ginv(C)

    Q <- t(U) %*% Cinv %*% U

    arqif1dev <- t(Ufirstdev) %*% Cinv %*% U
    arqif2dev <- t(Ufirstdev) %*% Cinv %*% Ufirstdev

    invarqif2dev <- ginv(arqif2dev)

    betanew <- beta - invarqif2dev %*% arqif1dev
    betadiff <- abs(sum(betanew - beta))
    iteration <- iteration +1
  }


  ################  GMM end   ###########

  ################ Output
  fit <- list()
  fit$terms <- Terms
  fit$formula <- as.vector(attr(Terms, "formula"))
  fit$call <- call
  fit$nobs <- nobs
  fit$iteration <- iteration
  fit$coefficients <- as.vector(beta)
  names(fit$coefficients) <- xnames
  fit$family <- family
  fit$y <- y
  fit$x <- x
  fit$id <- unique(id)
  fit$max.id <- max(nobs)
  fit$xnames <- xnames

  # covariance matrix
  dimnames(invarqif2dev)[[1]] <- xnames
  dimnames(invarqif2dev)[[2]] <- xnames
  fit$covariance <- invarqif2dev
  fit
}
```

```
### Defining the time points used to compute pseudo-observations
K = 5 # Number of time points


event_times <- simu$time[simu$event == 1]
# take the quantile of the event time
cutoffs <- quantile(event_times, seq(0, 1, length.out = K+2)[2:(K+1)])

### Compute the pseudo-observations
pseudo <- pseudo::pseudosurv(simu$time, simu$event, tmax=cutoffs)
```

```
#arrange the data
b <- NULL
for(it in 1:length(pseudo$time)){
  b <- rbind(b,cbind(simu,pseudo = pseudo$pseudo[,it],
                     tpseudo = pseudo$time[it],id=1:nrow(simu)))
}
b <- b[order(b $id),]
b$event<- as.factor(b$event)

### Fit a GMM model

fit_GMM <- gmm_cloglog(pseudo ~ as.factor(Zb)+ as.factor(tpseudo),
                       id = id, data = b, family=gaussian, corstr="independence")

as.data.frame(cbind(mean = fit_GMM$coefficients,
                    SE = sqrt(diag(fit_GMM$covariance))))
```

```
##                                          mean         SE
## (Intercept)                          -1.7928851 0.13174283
## as.factor(Zb)1                       -0.4015239 0.11593787
## as.factor(tpseudo)0.187811938275434   0.7963983 0.09692906
## as.factor(tpseudo)0.479508800946491   1.3188708 0.11269244
## as.factor(tpseudo)0.94444250554837    1.7687569 0.12114562
## as.factor(tpseudo)1.74276093976556    2.2041334 0.12734318
```

Bayesian piecewise exponential model

```
##Interval Murray :
r = sum(as.numeric(simu$event)) #event number
M = floor(max(5, min(r/8, 20)))


res <- summary(spBayesSurv::indeptCoxph(formula = survival::Surv(time, event) ~Zb, data = simu,
                                        prior=list(M=M, r0=1, beta0 = 0, S0 = 10000),
                                        mcmc=list(nburn=2000,
                                                  nsave=8000,
                                                  nskip=0,
                                                  ndisplay=1000)))
```

```
## scan = 1000
## scan = 2000
## scan = 3000
## scan = 4000
## scan = 5000
## scan = 6000
## scan = 7000
## scan = 8000
```

```
PEM_beta_hat <- res$coef[1,1];PEM_beta_hat
```

```
## [1] -0.4590899
```

```
PEM_se_hat <- res$coef[1,3]; PEM_se_hat
```

```
## [1] 0.1043122
```

Bayesian generalized method of moments based on pseudo-observations

```
### Defining the time points used to compute pseudo-observations
K = 5 # Number of time points
np = K+1
n = length(simu$patID)

event_times <- simu$time[simu$event == 1]
# take the quantile of the event time
cutoffs <- quantile(event_times, seq(0, 1, length.out = K+2)[2:(K+1)])

### Compute the pseudo-observations
pseudo <- pseudo::pseudosurv(simu$time, simu$event, tmax=cutoffs)

#arrange the data
b <- NULL
for(it in 1:length(pseudo$time)){
  b <- rbind(b,cbind(simu,pseudo = pseudo$pseudo[,it],
                     tpseudo = pseudo$time[it],id=1:nrow(simu)))
}
b <- b[order(b $id),]
b$event<- as.factor(b$event)



b$tpseudo1 <- 1*(b$tpseudo==pseudo$time[[1]])
b$tpseudo2 <- 1*(b$tpseudo==pseudo$time[[2]])
b$tpseudo3 <- 1*(b$tpseudo==pseudo$time[[3]])
b$tpseudo4 <- 1*(b$tpseudo==pseudo$time[[4]])
b$tpseudo5 <- 1*(b$tpseudo==pseudo$time[[5]])




X = matrix(c(rep(1,n*K), rep(simu$Zb, each = K), tpseudo2 = b$tpseudo2, tpseudo3 = b$tpseudo3,tspeudo4 =
             nrow = n*K, ncol = np)
data <- list(X = X, Y = b$pseudo, n = n, N = n, K = K, np = np)


### fit a Bayesian GMM with independence working matrix
GMM_ind <- rstan::stan_model("Bayesian_GMM_on_simulated_pseudo_vector.stan") # compile Stan model

# Calculate starting parameters for the 3 chains
b <- b%>%
  mutate(pseudo_cut1 = 0.99*(pseudo>=0.99) + 0.01*(pseudo<=0.01)+ pseudo*(pseudo<0.99 & pseudo>0.01),
         pseudo_cut2 = 0.95*(pseudo>=0.95) + 0.05*(pseudo<=0.05)+ pseudo*(pseudo<0.95 & pseudo>0.05),
         pseudo_cut3 = 0.90*(pseudo>=0.90) + 0.10*(pseudo<=0.10)+ pseudo*(pseudo<0.90 & pseudo>0.10))

fit_LM_cloglog1 <- lm(log(-log(pseudo_cut1)) ~ as.factor(Zb)+as.factor(tpseudo), data = b)
fit_LM_cloglog2 <- lm(log(-log(pseudo_cut2)) ~ as.factor(Zb)+as.factor(tpseudo), data = b)
fit_LM_cloglog3 <- lm(log(-log(pseudo_cut3)) ~ as.factor(Zb)+as.factor(tpseudo), data = b)

fit_BayesGMM_ind <- rstan::sampling(GMM_ind, data = data, chains = 3, iter = 6000, warmup = 1000, thin =
                                    init = list(chain1 = list(beta = as.numeric(fit_LM_cloglog1$coef)),
```

```
                                                chain2 = list(beta = as.numeric(fit_LM_cloglog2$coef)),
                                                chain3 = list(beta = as.numeric(fit_LM_cloglog3$coef)))
                                    save_warmup =T, seed = 1)
summary(fit_BayesGMM_ind)$summary
```

```
##               mean     se_mean         sd       2.5%        25%        50%
## beta[1] -1.8106473 0.002505166 0.13022468 -2.0744456 -1.8947599 -1.8070554
## beta[2] -0.4121645 0.002245937 0.11929763 -0.6507488 -0.4905597 -0.4086695
## beta[3]  0.8037238 0.001961183 0.09907091  0.6203965  0.7350647  0.8015940
## beta[4]  1.3290759 0.002234106 0.11388541  1.1125426  1.2521135  1.3247172
## beta[5]  1.7837428 0.002384662 0.12123238  1.5629914  1.7016599  1.7789634
## beta[6]  2.2229049 0.002522956 0.12855633  1.9861350  2.1320145  2.2157599
## loglik  -3.0394986 0.032944452 1.77482820 -7.4014581 -3.9440599 -2.7162982
## lp__    -3.7429463 0.033151958 1.78415169 -8.0840142 -4.6515945 -3.4220674
##                75%      97.5%    n_eff      Rhat
## beta[1] -1.7198815 -1.5692315 2702.175 1.0003068
## beta[2] -0.3298284 -0.1813578 2821.425 1.0006249
## beta[3]  0.8661016  1.0107248 2551.856 0.9995567
## beta[4]  1.4015931  1.5682677 2598.536 0.9996937
## beta[5]  1.8605605  2.0386332 2584.543 1.0001054
## beta[6]  2.3097974  2.4888851 2596.376 1.0005239
## loglik  -1.7323402 -0.6191548 2902.338 1.0004598
## lp__    -2.4243252 -1.3073313 2896.311 1.0004035
```

```
### fit a Bayesian GMM with exchangeable working matrix
GMM_exch <- rstan::stan_model("Bayesian_GMM_on_simulated_pseudo_exch_wcm_vector.stan") # compile Stan m

fit_BayesGMM_exch <- rstan::sampling(GMM_exch, data = data, chains = 3, iter = 6000, warmup = 1000, thi
                                    init = list(chain1 = list(beta = as.numeric(fit_LM_cloglog1$coef))
                                                chain2 = list(beta = as.numeric(fit_LM_cloglog2$coef))
                                                chain3 = list(beta = as.numeric(fit_LM_cloglog3$coef))
                                    save_warmup =T, seed = 1)
summary(fit_BayesGMM_exch)$summary
```

```
##               mean     se_mean         sd       2.5%        25%        50%
## beta[1] -1.7945652 0.002580091 0.13133854 -2.0675402 -1.8797236 -1.7884254
## beta[2] -0.4247052 0.002118666 0.11501811 -0.6509677 -0.5014264 -0.4226278
## beta[3]  0.7967192 0.001937929 0.09772515  0.6097710  0.7302351  0.7952228
## beta[4]  1.3172161 0.002338050 0.11408430  1.1006080  1.2391120  1.3164202
## beta[5]  1.7673691 0.002515995 0.12393174  1.5377611  1.6813039  1.7630106
## beta[6]  2.2098587 0.002583452 0.12816970  1.9724255  2.1208902  2.2032923
## loglik  -4.4347402 0.032588719 1.74840711 -8.6654683 -5.4014381 -4.0768070
## lp__    -5.1278667 0.032925336 1.75854877 -9.3815071 -6.1073966 -4.7655885
##                75%      97.5%    n_eff      Rhat
## beta[1] -1.7055919 -1.5510831 2591.280 0.9993670
## beta[2] -0.3454826 -0.2075194 2947.185 0.9994729
## beta[3]  0.8605199  0.9972053 2542.945 0.9994443
## beta[4]  1.3891274  1.5542112 2380.917 0.9998886
## beta[5]  1.8478888  2.0255932 2426.305 1.0000432
## beta[6]  2.2935697  2.4732914 2461.331 0.9998579
## loglik  -3.1838568 -1.9641308 2878.396 0.9999580
## lp__    -3.8641338 -2.6588292 2852.649 0.9999117
```

```
### fit a Bayesian GMM with ar-1 working matrix
GMM_ar1 <- rstan::stan_model("Bayesian_GMM_on_simulated_pseudo_ar1_wcm_vector.stan") # compile Stan mod
```

```
fit_BayesGMM_ar1 <- rstan::sampling(GMM_ar1, data = data, chains = 3, iter = 6000, warmup = 1000, thin =
                                    init = list(chain1 = list(beta = as.numeric(fit_LM_cloglog1$coef))
                                                chain2 = list(beta = as.numeric(fit_LM_cloglog2$coef))
                                                chain3 = list(beta = as.numeric(fit_LM_cloglog3$coef))
                                    save_warmup =T, seed = 1)
summary(fit_BayesGMM_ar1)$summary
```

```
##               mean     se_mean         sd        2.5%        25%        50%
## beta[1] -1.7911804 0.002838818 0.13261400  -2.0734076 -1.8726250 -1.7833316
## beta[2] -0.4179598 0.002258256 0.11671148  -0.6439041 -0.4964424 -0.4159164
## beta[3]  0.7974304 0.002147970 0.09828651   0.6191100  0.7288863  0.7915646
## beta[4]  1.3176058 0.002414109 0.11507536   1.1095873  1.2366028  1.3131097
## beta[5]  1.7541750 0.002908219 0.12608368   1.5266350  1.6679469  1.7469261
## beta[6]  2.2005993 0.003226719 0.13254741   1.9598723  2.1063459  2.1959266
## loglik  -5.5630203 0.034319741 1.77682650  -9.7928735 -6.5682087 -5.2163758
## lp__    -6.2511359 0.034742112 1.79228092 -10.5695804 -7.2656885 -5.9030341
##                75%      97.5%    n_eff      Rhat
## beta[1] -1.6996707 -1.5533016 2182.246 1.0002869
## beta[2] -0.3420758 -0.1819908 2671.042 0.9993246
## beta[3]  0.8599610  0.9986182 2093.781 0.9994487
## beta[4]  1.3887780  1.5571872 2272.224 0.9995735
## beta[5]  1.8332496  2.0204992 1879.591 1.0004196
## beta[6]  2.2867808  2.4846594 1687.408 1.0005631
## loglik  -4.2258218 -3.1254815 2680.415 1.0002632
## lp__    -4.9031104 -3.8124902 2661.336 1.0003369
```