Supplementary material of "Modeling the population mean outcome trajectory in observational studies with varying time to intervention"

HYUNKEUN RYAN CHO and SEONJIN KIM

University of Iowa and Miami University

Newton-Raphson algorithm

We provide how to obtain $\widehat{\boldsymbol{\beta}}(s) = \text{argmin}_{\boldsymbol{\beta}(s)} Q\{\boldsymbol{\beta}(s)\}$. Remark that minimizing $Q\{\boldsymbol{\beta}(s)\}$ is equivalent to solving $\mathbf{S}\{\boldsymbol{\beta}(s)\} = \dot{\mathbf{G}}\{\boldsymbol{\beta}(s)\}^\top \mathbf{V}\{\boldsymbol{\beta}(s)\}^{-1}\mathbf{G}\{\boldsymbol{\beta}(s)\} = 0$, where $\dot{\mathbf{G}}\{\boldsymbol{\beta}(s)\} = \partial \mathbf{G}\{\boldsymbol{\beta}(s)\}/\partial \boldsymbol{\beta}(s) = \sum_{i=1}^n \dot{\mathbf{h}}_i\{\boldsymbol{\beta}(s)\}K_i(s)/\{f(s|\mathbf{Z}_i)nb(s)\}$ with

$$\dot{\mathbf{h}}_i\{\boldsymbol{\beta}(s)\} = \begin{pmatrix} \dot{\boldsymbol{\mu}}_i^\top \mathbf{A}_i^{-1/2}\mathbf{B}_{i1}\mathbf{A}_i^{-1/2}\dot{\boldsymbol{\mu}}_i \\ \vdots \\ \dot{\boldsymbol{\mu}}_i^\top \mathbf{A}_i^{-1/2}\mathbf{B}_{iD}\mathbf{A}_i^{-1/2}\dot{\boldsymbol{\mu}}_i \end{pmatrix}.$$

Thus, we apply the Newton-Raphson algorithm to solve $\mathbf{S}\{\boldsymbol{\beta}(s)\} = 0$. Specifically, given the estimator at the kth iteration $\widehat{\boldsymbol{\beta}}(s)^{(k)}$, $\widehat{\boldsymbol{\beta}}(s)^{(k+1)}$ is obtained as

$$\widehat{\boldsymbol{\beta}}(s)^{(k+1)} = \widehat{\boldsymbol{\beta}}(s)^{(k)} - [\dot{\mathbf{S}}\{\widehat{\boldsymbol{\beta}}(s)^{(k)}\}]^{-1}\mathbf{S}\{\widehat{\boldsymbol{\beta}}(s)^{(k)}\},$$

where $\dot{\mathbf{S}}\{\widehat{\boldsymbol{\beta}}(s)^{(k)}\} = \dot{\mathbf{G}}\{\widehat{\boldsymbol{\beta}}(s)^{(k)}\}^{\top}\mathbf{V}\{\widehat{\boldsymbol{\beta}}(s)^{(k)}\}^{-1}\dot{\mathbf{G}}\{\widehat{\boldsymbol{\beta}}(s)^{(k)}\}$. We iterate the above process to convergence, that is, $||\widehat{\boldsymbol{\beta}}(s)^{(k+1)} - \widehat{\boldsymbol{\beta}}(s)^{(k)}|| < 10^{-8}$.

R codes of Section 6. Simulation studies

```
# n subjects with 5 repeated measures in 1000 simulated datasets
n=1000; ni=5; sim=1000; ID.f=rep(1:n,each=ni)


# Epanechnikov kernel function
epan=function(x,b){3/4*(1-(x/b)^2)*(abs(x)<b)}


b.in.obs=b.in.ipw=array(0,dim=c(3,sim))
b.ar.obs=b.ar.ipw=array(0,dim=c(3,sim))
b.cs.obs=b.cs.ipw=array(0,dim=c(3,sim))
se.in.obs=se.in.ipw=array(0,dim=c(3,sim))
se.ar.obs=se.ar.ipw=array(0,dim=c(3,sim))
se.cs.obs=se.cs.ipw=array(0,dim=c(3,sim))
qhat.ar.obs=qhat.cs.obs=qhat.ar.ipw=qhat.cs.ipw=array(0,dim=c(sim))


for(s in 1:sim){


# Generate TTI S, measurement time points T, and design matrix X in (6.2)
```

```
si=runif(n,0.5,2.5)
age=cbind(0,runif(n),1+runif(n),2+runif(n),3+runif(n)); Age=c(t(age))
s.rep=rep(si,each=5)
spline1=Age-s.rep;spline1[spline1<0]=0
X1.f=cbind(Age,spline1)

# Generate confounders Z, error e, and five repeated outcomes Y
z1=rnorm(n,0,0.5); z2=rnorm(n,0,0.5); z3=rnorm(n,0,0.5)
epsilon=rep(z1,each=5)+rep(z2,each=5)*Age
+rep(z3,each=5)*spline1+rnorm(n*ni,0,0.5)
y.f=exp(0.1*s.rep-0.15)+(cos(pi*s.rep)/25-1)*Age
+(0.03-0.02*s.rep)*spline1+epsilon

# Process of simulating missing data
miss=array(1,dim=c(n,ni))
miss[,2:ni]=rbinom(n*(ni-1), 1, prob=0.8)
X1=X1.f[c(t(miss))==1,]
y=y.f[c(t(miss))==1]
ID=ID.f[c(t(miss))==1]

# Process of building generalized propensity score (GPS)
```

```
prob=exp(0.2*z1+0.3*z2-0.5*z3)/(1+exp(0.2*z1+0.3*z2-0.5*z3))
M=rbinom(n, 1, prob=prob)
M.ipw=M
M.ipw[M==1]=1/predict(glm(M~z1+z2+z3,family=binomial(link=logit)),
type='response')[which(M==1)]

# Process of finding the optimal bandwidth
mse=mse1=rep(0,10);pj=1
for (b in 0.1*1:10){
for(i in which(M==1)){
T=si[i]
MM=M
MM[i]=0
kernel=epan(si[which(MM==1)]-T,b)
wgs1=rep(kernel, each=5)
wfit=lm(y.f[5*rep(which(MM==1),each=5)-4:0]~
X1.f[5*rep(which(MM==1),each=5)-4:0,],weights=wgs1)
mse[j]=mse[j]+sum(c((y.f[(5*(i-1)+1):(5*i)]-
cbind(1,X1.f[(5*(i-1)+1):(5*i),])%*%wfit$coefficient)^2))*epan(si[i]-1.5,b)
mse1[j]=mse1[j]+5*epan(si[i]-1.5,b)
}
```

```
j=j+1
}
bhat=which(min(mse/mse1)==mse/mse1)*0.1*sum(M==1)^(-1/20)

# Process of building kernel density
kernel.f <- epan(s.rep-1.5,bhat)
kernel.i <- epan(si-1.5,bhat)
kernel <- kernel.f[c(t(miss))==1]

# Estimation of coefficients under an independent correlation structure
# Using the kernel-weighted estimation procedure
b.in.obs[,s]=
summary(lm(y~X1, weights = kernel*rep(M, table(ID))))$coef[,1]
se.in.obs[,s]=
summary(lm(y~X1, weights = kernel*rep(M, table(ID))))$coef[,2]
# Using the double-weighted estimation procedure
b.in.ipw[,s]=
summary(lm(y~X1, weights = kernel*rep(M.ipw, table(ID))))$coef[,1]
se.in.ipw[,s]=
summary(lm(y~X1, weights = kernel*rep(M.ipw, table(ID))))$coef[,2]
```

```
# Estimation of coefficients under the AR(1) structure
# Using the kernel-weighted estimation procedure
fit.ar.obs=
WQIF(y,cbind(1,X1),b.in.obs[,s],table(ID),'ar1','cont',kernel.i*M)
b.ar.obs[,s]=fit.ar.obs$coefficient[,1]
se.ar.obs[,s]=fit.ar.obs$coefficient[,2]
qhat.ar.obs[s]=fit.ar.obs$Q
# Using the double-weighted estimation procedure
fit.ar.ipw=
WQIF(y,cbind(1,X1),b.in.ipw[,s],table(ID),'ar1','cont',kernel.i*M.ipw)
b.ar.ipw[,s]=fit.ar.ipw$coefficient[,1]
se.ar.ipw[,s]=fit.ar.ipw$coefficient[,2]
qhat.ar.ipw[s]=fit.ar.ipw$Q


# Estimation of coefficients under the compound symmetry structure
# Using the kernel-weighted estimation procedure
fit.cs.obs=
WQIF(y,cbind(1,X1),b.in.obs[,s],table(ID),'exch','cont',kernel.i*M)
b.cs.obs[,s]=fit.cs.obs$coefficient[,1]
se.cs.obs[,s]=fit.cs.obs$coefficient[,2]
qhat.cs.obs[s]=fit.cs.obs$Q
```

```
# Using the double-weighted estimation procedure

fit.cs.ipw=

WQIF(y,cbind(1,X1),b.in.ipw[,s],table(ID),'exch','cont',kernel.i*M.ipw)

b.cs.ipw[,s]=fit.cs.ipw$coefficient[,1]

se.cs.ipw[,s]=fit.cs.ipw$coefficient[,2]

qhat.cs.ipw[s]=fit.cs.ipw$Q

}


# Construction of the AR(1) structure

ar1 = function(rho,n) {

z = outer(1:n,1:n,'-')

z = rho^(abs(z))

diag(z) = 1

z

}


# Construction of the compound symmetry structure

exch = function(para,s) {

z = matrix(para,s,s)

diag(z) = 1

z
```

```
}


# R function for estimating coefficients using QIF
WQIF = function(y,x,beta_initial,nobs,working,response,wgs){
Y = as.matrix(y) # Outcome
X = as.matrix(x) # Design matrix
n = length(nobs) # Sample size
p = dim(x)[[2]] # The column dimension of X
betadiff = 1 # The initial value of beta estimates difference
iteration = 0 # The number of iterations
betanew = as.numeric(beta_initial) # Newly updated estimate
tol = 1e-8 # Stopping rule of the iteration
maxiter = 200 # The maximum number of iterations


# Process of the iterating procedure for estimation of beta
while (betadiff > tol && iteration < maxiter && sum(betanew)!=0) {
beta = betanew
sumg = matrix(rep(0, 2 * p), nrow = 2 * p)
sumc = matrix(rep(0, 2 * p * 2 * p), nrow = 2 * p)
arsumg = matrix(rep(0, 2 * p), nrow = 2 * p)
arsumc = matrix(rep(0, 2 * p * 2 * p), nrow = 2 * p)
```

```
gi = matrix(rep(0, 2 * p), nrow = 2 * p)

arsumgfirstdev = matrix(rep(0, 2 * p * p), nrow = 2 * p)

firstdev = matrix(rep(0, 2 * p * p), nrow = 2 * p)

loc1 = 0; loc2 = 0

for (i in 1:n) {

loc1 = loc2 + 1

loc2 = loc1 + nobs[i] - 1

yi = as.matrix(Y[loc1:loc2, ])

xi = matrix(X[loc1:loc2, ],nrow=nobs[i])

ni = nrow(yi)

m0 = diag(ni)


# Basis matrix under the AR(1)

if(working=="ar1"){

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

}}}
```

```
# Basis matrix under the compound symmetry
if(working=="exch"){
m1 = matrix(rep(1, ni * ni), ni) - m0
}

# For continuous outcome
if (response == "cont") {
if(length(beta)!=1){ui = xi %*% beta}
if(length(beta)==1){ui = t(beta %*% t(xi))}
fui = ui
fui_dev = diag(ni)
vui = diag(ni)
wi = wgs[i] * t(xi) %*% fui_dev %*% vui %*% m0 %*% vui
zi = wgs[i] * t(xi) %*% fui_dev %*% vui %*% m1 %*% vui
}

# For binary outcome
if (response == "binary") {
ui = 1/(1 + exp(-xi %*% beta))
fui = log(ui) - log(1 - ui)
fui_dev = diag(as.vector(ui)) %*% diag(as.vector(1 - ui))
```

```
vui = diag(as.vector(sqrt(1/ui))) %*% diag(as.vector(sqrt(1/(1 - ui))))

wi = wgs[i] * t(xi) %*% fui_dev %*% vui %*% m0 %*% vui

zi = wgs[i] * t(xi) %*% fui_dev %*% vui %*% m1 %*% vui

}


# For count outcome

if (response == "poisson") {

ui = exp(xi %*% beta)

fui = log(ui)

fui_dev = diag(as.vector(ui))

vui = diag(as.vector(sqrt(1/ui)))

wi = wgs[i] * t(xi) %*% fui_dev %*% vui %*% m0 %*% vui

zi = wgs[i] * t(xi) %*% fui_dev %*% vui %*% m1 %*% vui

}


gi0 = (1/n) * wi %*% (yi - ui)

gi1 = (1/n) * zi %*% (yi - ui)

gi[1:p, ] = gi0

gi[(p + 1):(2 * p), ] = gi1

arsumc = arsumc + gi %*% t(gi)

arsumg = arsumg + gi
```

```
di0 = -(1/n) * wi %*% fui_dev %*% xi

di1 = -(1/n) * zi %*% fui_dev %*% xi

firstdev[1:p, ] = di0

firstdev[(p + 1):(2 * p), ] = di1

arsumgfirstdev = arsumgfirstdev + firstdev

}


arcinv = solve(arsumc)

# The value of QIF

Q = t(arsumg) %*% arcinv %*% arsumg

arqif1dev = t(arsumgfirstdev) %*% arcinv %*% arsumg

arqif2dev = t(arsumgfirstdev) %*% arcinv %*% arsumgfirstdev

invarqif2dev = solve(arqif2dev)


# Update a new estimate

betanew = beta - invarqif2dev %*% arqif1dev


# Calculate the difference of two beta estimates

betadiff = abs(sum(betanew - beta)) #t(betanew - beta)%*%(betanew - beta)

iteration = iteration + 1

}
```

```
# Standard error of the final estimate

betase = sqrt(diag(invarqif2dev))

result = cbind(betanew,betase)

return(list(iteration=iteration, coefficient=result, Q=Q))

}
```

Department of Biostatistics, University of Iowa, Iowa City, IA 52242

E-mail: (hyunkeun-cho@uiowa.edu)

Department of Statistics, Miami University, Oxford, OH 45056

E-mail: (kims20@miamioh.edu)