```
########################## Simulation 1 ##########################

setwd("~/research/Rcodes")
library(foreach)
library(doParallel)
library(doRNG)

cl=makeCluster(24, outfile="")
registerDoParallel(cl)

library(MASS)
library(Matrix)
library(glmnet)
library(compositions)


library(scout)
library(softImpute)
library(grplasso)
library(robustbase)


source('pQIFmp_scad25_par.R')
source('imputeglm.predict.R')
source("mySCAD5.R")
source('DISCOM.R')

n=700        ##### Only changed here compared to Sim133.R
n1=30
n2=220
n3=220
n4=230
p=40
q=14   ##### 4+4+4+2
n_pat=4
Im=10
Im2=13

p1=12
p2=12
p3=12
p4=4

nalpha=10
nlambda=15
alpha.all=10^seq(0,-3,length=nalpha)
lambda.all=10^(seq(log10(2),-3,length=nlambda))

n.tun=10                  ### tuning dataset


a=c(0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9)
beta=c(0.8, 1, 1.5, 2, 2.5, 3, 3.5, 4, 5, 6, 7, 8, 9, 10)
```

```
index2=matrix(TRUE,p,Im)              # take which derivative in
estimating equations
index2[25:36,2]=FALSE
index2[13:36,3]=FALSE
index2[c(1:12,25:36),4]=FALSE
index2[13:24,5]=FALSE
index2[13:36,6]=FALSE
index2[1:24,7]=FALSE
index2[1:12,8]=FALSE
index2[c(1:12,25:36),9]=FALSE
index2[1:24,10]=FALSE

impest=rep(0,9)

missid=matrix(0,3,12)
missid[1,]=c(1:12)
missid[2,]=c(13:24)
missid[3,]=25:36

T=50
nmethod=14

table_b=matrix(0,45,(4+6*nmethod+Im-2))

result_b= array(0, c(T, nmethod*(p+2)+Im-1, 11, 9))
set.seed(1)
seed_each=sample.int(5000000, 100000)
seed_count=1
count=1
IX= array(TRUE, dim=c(T, 11, 9))
for (s1 in 4:8) {
  for (b in c(11,9,7,5)) {

    sigma1=matrix(0,p,p)
    sigma1[1:p,1:p]=diag(1-a[s1],p,p)+matrix(a[s1],p,p)

    true=c(beta[b]*rep(1,4), rep(0,8), beta[b+1]*rep(1,4),
rep(0,8),beta[b+2]*rep(1,4), rep(0,8), beta[b+3]*rep(1,2), rep(0,2))
    signal_id=c(1:4,13:16,25:28,37:38)

    start.time = proc.time()
    ii=1
    while (ii<=T) {
      set.seed(seed_each[seed_count])
      x00=mvrnorm(n,rep(0,p),sigma1)          ## original data
      x=x00
      noise=rnorm(n,0,1)

y=noise+beta[b]*apply(x[,1:4],1,sum)+beta[b+1]*apply(x[,13:16],1,sum)+
beta[b+2]*apply(x[,25:28],1,sum)+beta[b+3]*apply(x[,37:38],1,sum)
```

```
      prob0=exp(-10*apply(x[,37:40],1,sum))
      if (sum(prob0==Inf)>0){
        prob0[prob0==Inf]=max(prob0[!prob0==Inf])
      }
      group1=sample.int(n,n1, prob = prob0)
      group2=sample.int(n-n1,n2)
      group3=sample.int(n-n1-n2,n3)

      x1=x
      for (k1 in 1:n) {
        if (sum((1:n)[-group1][group2]==k1)>0) {
          x1[k1,25:36]=NA
        } else if (sum((1:n)[-group1][-group2][group3]==k1)>0) {
          x1[k1,13:24]=NA
        } else if (sum((1:n)[-group1][-group2][-group3]==k1)>0) {
          x1[k1,1:12]=NA
        }
      }


      x4=na.omit(x1)
      y4=y[apply(is.na(x1),1,sum)==0]

      x0=x
      x2=x1
      centerx=rep(0,p)
      scalex=rep(1,p)
      centery=mean(y)
      y2=y-centery
      for (j in 1:p) {
        centerx[j]=mean(x1[,j], na.rm=TRUE)
        scalex[j]=sd(x1[,j], na.rm = TRUE)
        x2[,j]=(x1[,j]-centerx[j])/scalex[j]
        x0[,j]=(x[,j]-centerx[j])/scalex[j]
      }

      miss=is.na(x2)
      miss_num=apply(miss,1,function(x) unbinary(paste(as.numeric(x),
collapse="")))
      x5=x2[order(miss_num),]
      x6=x1[order(miss_num),]             ###### Don't standardize for
DISCOM

      bi_miss=is.na(x5)[,c(1,13,25)]
      id=apply(bi_miss,1,function(x) unbinary(paste(as.numeric(x),
collapse="")))
      n_pat=length(unique(id))
      pat_start=rep(1,n_pat)
      table_pat=table(id)
      count_tmp=1
      for (l in 2:n_pat){
        count_tmp=count_tmp+table_pat[l-1]
        pat_start[l]=count_tmp
```

```
      }

      y5=y2[order(miss_num)]
      x.ori=x0[order(miss_num),]

      x3=array(0,dim=c(n,p,Im))
      y3=matrix(0,n,Im)
      index1=matrix(FALSE,n,Im)            # use which sample
      index1[1:(pat_start[2]-1),1]=TRUE
      pat=rep(0,Im)
      for (j in 1:Im) {
        y3[,j]=y5
        if (j==1) {
          x3[pat_start[1]:(pat_start[2]-
1),,j]=x5[pat_start[1]:(pat_start[2]-1),]
          pat[j]=1
        }
        if (j==2|j==3|j==4) {
          x3[pat_start[2]:(pat_start[3]-
1),,j]=x5[pat_start[2]:(pat_start[3]-1),]
          index1[pat_start[2]:(pat_start[3]-1),j]=TRUE
          pat[j]=2
        } else if (j==5|j==6|j==7) {
          x3[pat_start[3]:(pat_start[4]-
1),,j]=x5[pat_start[3]:(pat_start[4]-1),]
          index1[pat_start[3]:(pat_start[4]-1),j]=TRUE
          pat[j]=3
        } else if (j==8|j==9|j==10) {
          x3[pat_start[4]:n,,j]=x5[pat_start[4]:n,]
          index1[pat_start[4]:n,j]=TRUE
          pat[j]=4
        }
      }

      miss2=is.na(x5)
      PCA_m=rep(0,Im)
      PCA_m[1]=1
      for (k in 2:Im) {
        if (k==2|k==5|k==8) {
          miss_tmp=is.na(x3[which.max(index1[,k]),,k])
          ind_y=(1:p)[miss_tmp]
          ind_tmp=ind_y
          x3[index1[,k]==TRUE,ind_y,k]=imputeglm.predict(X=x5,
ind_y=ind_y, miss=miss2, newdata = x3[index1[,k]==TRUE,-
ind_tmp,k])$PRED
          PCA_m[k]=1
        } else if (k==3|k==6) {
          miss_tmp=is.na(x3[which.max(index1[,k]),,k])
          ind_y=(1:p)[miss_tmp]
          ind_tmp=13:36
          x3[index1[,k]==TRUE,ind_y,k]=imputeglm.predict(X=x5,
ind_y=ind_y, ind_x = c(1:12,37:40), miss=miss2, newdata =
x3[index1[,k]==TRUE,-ind_tmp,k])$PRED
```

```
          PCA_m[k]=2
        } else if (k==4|k==9) {
          miss_tmp=is.na(x3[which.max(index1[,k]),,k])
          ind_y=(1:p)[miss_tmp]
          ind_tmp=c(1:12,25:36)
          x3[index1[,k]==TRUE,ind_y,k]=imputeglm.predict(X=x5,
ind_y=ind_y, ind_x = c(13:24,37:40), miss=miss2, newdata =
x3[index1[,k]==TRUE,-ind_tmp,k])$PRED
          PCA_m[k]=2
        } else if (k==7|k==10) {
          miss_tmp=is.na(x3[which.max(index1[,k]),,k])
          ind_y=(1:p)[miss_tmp]
          ind_tmp=1:24
          x3[index1[,k]==TRUE,ind_y,k]=imputeglm.predict(X=x5,
ind_y=ind_y, ind_x = c(25:36,37:40), miss=miss2, newdata =
x3[index1[,k]==TRUE,-ind_tmp,k])$PRED
          PCA_m[k]=2
        }
      }

      impest[1]=mean((x3[pat_start[2]:(pat_start[3]-1),
miss2[pat_start[2],], 2]-x.ori[pat_start[2]:(pat_start[3]-1),
miss2[pat_start[2],]])^2)
      impest[2]=mean((x3[pat_start[3]:(pat_start[4]-1),
miss2[pat_start[3],], 5]-x.ori[pat_start[3]:(pat_start[4]-1),
miss2[pat_start[3],]])^2)
      impest[3]=mean((x3[pat_start[4]:n, miss2[pat_start[4],], 8]-
x.ori[pat_start[4]:n, miss2[pat_start[4],]])^2)
      impest[4]=mean((x3[pat_start[2]:(pat_start[3]-1),
miss2[pat_start[2],], 3]-x.ori[pat_start[2]:(pat_start[3]-1),
miss2[pat_start[2],]])^2)
      impest[5]=mean((x3[pat_start[3]:(pat_start[4]-1),
miss2[pat_start[3],], 6]-x.ori[pat_start[3]:(pat_start[4]-1),
miss2[pat_start[3],]])^2)
      impest[6]=mean((x3[pat_start[4]:n, miss2[pat_start[4],], 9]-
x.ori[pat_start[4]:n, miss2[pat_start[4],]])^2)
      impest[7]=mean((x3[pat_start[2]:(pat_start[3]-1),
miss2[pat_start[2],], 4]-x.ori[pat_start[2]:(pat_start[3]-1),
miss2[pat_start[2],]])^2)
      impest[8]=mean((x3[pat_start[3]:(pat_start[4]-1),
miss2[pat_start[3],], 7]-x.ori[pat_start[3]:(pat_start[4]-1),
miss2[pat_start[3],]])^2)
      impest[9]=mean((x3[pat_start[4]:n, miss2[pat_start[4],], 10]-
x.ori[pat_start[4]:n, miss2[pat_start[4],]])^2)



      x5[pat_start[2]:(pat_start[3]-
1),]=x3[pat_start[2]:(pat_start[3]-1),,2]
      x5[pat_start[3]:(pat_start[4]-
1),]=x3[pat_start[3]:(pat_start[4]-1),,5]
      x5[pat_start[4]:n,]=x3[pat_start[4]:n,,8]
```

```
      model1=pQIFmp1(X=x3, y=y3, index1=index1, index2=index2, nlam =
15, index0 = c(1,2,5,8), missid=missid, eps2 = 1e-3, max.iter = 5000,
centerx=centerx, scalex=scalex, centery=centery, lambda.min =
0.000001, pat=pat, PCA_m = PCA_m, h1=2^(-(8:20)))
      theta1=model1$beta
      bic1=model1$bic      ###### changed from biq to bic
      best1=which.min(bic1[bic1!=0])
      tmp1=theta1[best1,]
      notcon1=model1$notcon
      if (length(best1)==0) {
        best1=NA
      }
      if (length(tmp1)==0) {
        tmp1=rep(NA,p)
      }

      bic2=model1$bic1      ###### changed from biq to bic
      best2=which.min(bic2[bic2!=0])
      tmp2=theta1[best2,]
      notcon2=model1$notcon
      if (length(best2)==0) {
        best2=NA
      }
      if (length(tmp2)==0) {
        tmp2=rep(NA,p)
      }

      bic5=model1$ebic1      ###### changed from biq to bic
      best5=which.min(bic5[bic5!=0])
      tmp5=theta1[best5,]
      notcon5=model1$notcon
      if (length(best5)==0) {
        best5=NA
      }
      if (length(tmp5)==0) {
        tmp5=rep(NA,p)
      }

      bic6=model1$ebic11      ###### changed from biq to bic
      best6=which.min(bic6[bic6!=0])
      tmp6=theta1[best6,]
      notcon6=model1$notcon
      if (length(best6)==0) {
        best6=NA
      }
      if (length(tmp6)==0) {
        tmp6=rep(NA,p)
      }


      bic9=model1$ebicM      ###### changed from biq to bic
      best9=which.min(bic9[bic9!=0])
      tmp9=theta1[best9,]
```

```
notcon9=model1$notcon
if (length(best9)==0) {
  best9=NA
}
if (length(tmp9)==0) {
  tmp9=rep(NA,p)
}

bic10=model1$ebicM1      ###### changed from biq to bic
best10=which.min(bic10[bic10!=0])
tmp10=theta1[best10,]
notcon10=model1$notcon
if (length(best10)==0) {
  best10=NA
}
if (length(tmp10)==0) {
  tmp10=rep(NA,p)
}

model3=mySCAD(x4, y4, standardize=TRUE, max.iter = 5000, nlam =
15)
theta3=model3$beta
bic3=model3$bic
best3=which.min(bic3)
tmp3=theta3[best3,]
notcon3=model3$notcon

bic7=model3$ebic1
best7=which.min(bic7)
tmp7=theta3[best7,]
notcon7=model3$notcon

bic11=model3$ebicM
best11=which.min(bic11)
tmp11=theta3[best11,]
notcon11=model3$notcon


object4=mySCAD(x5, y5, standardize=TRUE, max.iter = 5000, nlam =
15)
theta4=object4$beta
bic4=object4$bic
best4=which.min(bic4)
tmp4=theta4[best4,]/scalex
notcon4=object4$notcon

bic8=object4$ebic1
best8=which.min(bic8)
tmp8=theta4[best8,]/scalex
notcon8=object4$notcon

bic12=object4$ebicM
best12=which.min(bic12)
```

```
      tmp12=theta4[best12,]/scalex
      notcon12=object4$notcon


      ## DISCOM
      X.all=x6[(n.tun+1):n,]
      Y=y5[(n.tun+1):n]
      X.tun=x6[1:n.tun,]
      Y.tun=y5[1:n.tun]

      DISCOM.tun.error=array(NA,dim=c(nalpha,nalpha,nlambda))

      for(code in 1:2)
      {
        if(code==1){
          xtx.raw=compute.xtx(X.all)
          xty=compute.xty(X.all,Y)
        }
        else{
          xtx.raw=compute.xtx(X.all,1,1)
          xty=compute.xty(X.all,Y,1,1)
        }


xtx.raw.I=as.matrix(bdiag(xtx.raw[1:p1,1:p1],xtx.raw[(p1+1):(p1+p2),(p
1+1):(p1+p2)],

xtx.raw[(p1+p2+1):(p1+p2+p3),(p1+p2+1):(p1+p2+p3)],

xtx.raw[(p1+p2+p3+1):(p1+p2+p3+p4),(p1+p2+p3+1):(p1+p2+p3+p4)]))
        xtx.raw.C=xtx.raw-xtx.raw.I
        shrink.target=sum(diag(xtx.raw))/p

        for(i in 1:nalpha){
          for(j in 1:nalpha){
            alpha1=alpha.all[i]
            alpha2=alpha.all[j]
            xtx=alpha1*xtx.raw.I+alpha2*xtx.raw.C+(1-
alpha1)*shrink.target*diag(p)
            if(min(eigen(xtx)$values)<0){
              DISCOM.tun.error[i,j,]=10^8
            } else{
              beta.initial=rep(0,p)
              for(k in 1:nlambda){

beta.cov=as.vector(crossProdLasso(xtx,xty,lambda.all[k],beta.init=beta
.initial)$beta)
                betal.initial=beta.cov

DISCOM.tun.values=as.vector(as.matrix(X.tun)%*%beta.cov)
                DISCOM.tun.error[i,j,k]=mean((Y.tun-
DISCOM.tun.values)^2)
              }
```

```
            }
          }
        }


opt.index=as.vector(which(DISCOM.tun.error==min(DISCOM.tun.error),arr.
ind=T)[1,])
        opt.alpha1=alpha.all[opt.index[1]]
        opt.alpha2=alpha.all[opt.index[2]]
        opt.lambda=lambda.all[opt.index[3]]
        xtx=opt.alpha1*xtx.raw.I+opt.alpha2*xtx.raw.C+(1-
opt.alpha1)*shrink.target*diag(p)
        beta.cov=as.vector(crossProdLasso(xtx,xty,opt.lambda)$beta)

        if(code==1){
          tmp13=beta.cov/scalex
          best13=opt.lambda
          notcon13=0
        } else{
          tmp14=beta.cov/scalex
          best14=opt.lambda
          notcon14=0
        }
      }


      for (j in 1:nmethod) {
        result_b[ii, ((j-1)*(p+2)+1):((j-1)*(p+2)+p), b,
s1]=get(paste('tmp',j, sep = ''))
        result_b[ii, (j-1)*(p+2)+p+1, b, s1]=get(paste('notcon',j, sep
= ''))
        result_b[ii, (j-1)*(p+2)+p+2, b, s1]=get(paste('best',j, sep =
''))
      }
      result_b[ii, (nmethod*(p+2)+1):(nmethod*(p+2)+Im-1), b,
s1]=impest

      save.image("~/research/Sim_pQIF_simple_est136.RData")
      ii=ii+1
      seed_count=seed_count+1
    }
    time = proc.time() - start.time

    FN=rep(0,nmethod)
    FP=rep(0,nmethod)
    mpe=rep(0,nmethod)
    variance=rep(0,nmethod)
    mean=rep(0,nmethod)
    notcon=rep(0,T)
    for (i in 1:nmethod) {
      notcon=notcon+result_b[,((i-1)*(p+2)+p+1),b,s1]
    }
    miss_u=apply(result_b[,,b,s1],1,sum)
```

```
    index_u=(notcon==0) & (!is.na(miss_u)) & IX[, b, s1]
    for (i in 1:nmethod) {
      FN[i]=1-mean(result_b[index_u,((i-1)*(p+2)+1):((i-
1)*(p+2)+p),b,s1][,signal_id]!=0)
      FP[i]=mean(result_b[index_u,((i-1)*(p+2)+1):((i-
1)*(p+2)+p),b,s1][,-signal_id]!=0)

      temp=t(apply(result_b[index_u,((i-1)*(p+2)+1):((i-
1)*(p+2)+p),b,s1], 1, function(x) x-true))
      mpe[i]=mean(apply(temp,2,function(x) mean(x^2)))
      variance[i]=mean(apply(temp,2,sd))
      mean[i]=mean(temp)
    }

    table_b[count,1]=n
    table_b[count,2]=a[s1]
    table_b[count,3]=beta[b]
    table_b[count,4:(4+nmethod-1)]=FN
    table_b[count,(4+nmethod):(4+2*nmethod-1)]=FP
    table_b[count,(4+2*nmethod):(4+3*nmethod-1)]=FN+FP
    table_b[count,(4+3*nmethod):(4+4*nmethod-1)]=mpe
    table_b[count,(4+4*nmethod):(4+5*nmethod-1)]=variance
    table_b[count,(4+5*nmethod):(4+6*nmethod-1)]=mean
    table_b[count,(4+6*nmethod):(4+6*nmethod+Im-
2)]=apply(result_b[,(nmethod*(p+2)+1):(nmethod*(p+2)+Im-
1),b,s1],2,mean)

    print(count)

    count=count+1

  }
}
save.image("~/research/Sim_pQIF_simple_est136.RData")

stopCluster(cl)
```

```
#####################################################################
#### 'pQIFmp_scad25_par.R': main function for the proposed method ####

library(pryr)
pQIFmp1 <- function (X, y, index1, index2, index0, missid, centerx,
scalex, centery=0, lambda=NULL, eps1 = 1e-3, eps2 = 1e-7, eps3=1e-8,
max.iter = 1000, lambda.min=ifelse(n>p,.001,.05), nlam=100,
beta0=NULL, C0=NULL, a=3.7, PCA_m, pat, gamma.ebic=0.5,
alpha1=0.5^(0:12), h1=2^(-(8:30)), ratio=1) {
  dims=dim(X)
  n=dims[1]
  p=dims[2]
  m=dims[3]      # Number of imputations
  #m0=length(index0)     #Number of original patterns
  nm=dim(missid)[1]     #Number of missing sources
  n_pat=length(unique(pat))     #Number of partterns

  XX=X
  yy=y
  vary=rep(1,m)
  for (i in 1:m) {
    vary[i]=var(y[index1[,i],i])
  }


  if (is.null(beta0)) {
    if (!is.null(lambda)) {
      if (lambda==0) {
        if (sum(index1[,1])>p) {
          beta0=lm(yy[index1[,1],1]~XX[index1[,1],,1])$coef[-1]
        } else {
          cvfit <- cv.glmnet(XX[index1[,1],,1], yy[index1[,1],1],
nfolds=3)
          fit <- cvfit$glmnet.fit
          beta0=t(as.numeric(coef(fit, s=cvfit$lambda.min))[-1])
        }
      } else {
        model_0=pQIFmp1(X=X, y=y, index1=index1, index2=index2,
index0=index0, missid=missid,  centerx=centerx, scalex=scalex,
centery=centery, lambda=0, eps1 = eps1, eps2 = eps2, eps3=eps3,
max.iter = max.iter, lambda.min=lambda.min, nlam=nlam, beta0=beta0,
C0=C0, a=a, PCA_m=PCA_m, pat=pat, gamma.ebic=gamma.ebic,
alpha1=alpha1, h1=h1, ratio = ratio)
        beta0=model_0$beta0
      }
    } else {
      model_0=pQIFmp1(X=X, y=y, index1=index1, index2=index2,
index0=index0, missid=missid,  centerx=centerx, scalex=scalex,
centery=centery, lambda=0, eps1 = eps1, eps2 = eps2, eps3=eps3,
max.iter = max.iter, lambda.min=lambda.min, nlam=nlam, beta0=beta0,
C0=C0, a=a, PCA_m=PCA_m, pat=pat, gamma.ebic=gamma.ebic,
alpha1=alpha1, h1=h1, ratio = ratio)
      beta0=model_0$beta0
```

```
    }
  }

  N=sum(index2)
  N0=sum(index2[,index0])
  numequ=apply(index2,2,sum)
  nn=apply(index1,2,sum)
  nnn=sqrt(apply(index1,2,sum))
  g=rep(0,N)
  fq=rep(0,p)
  PCA_ind=array(NA,dim=c(n_pat,2,N))
  PCA_ind0=matrix(0,n_pat,2)
  ind_C=rep(1,n_pat+1)
  ind_pat=rep(1,n_pat+1)
  U_ind=array(NA,dim=c(n_pat,2,N))



  Z2=matrix(0,n,N)
  tempequ1=rep(0,m+1)
  tempequ1[1]=1
  for (s in 1:m) {
    tempequ1[s+1]=tempequ1[s]+numequ[s]
    if(PCA_m[s]==1) {

PCA_ind[pat[s],1,(PCA_ind0[pat[s],1]+1):(PCA_ind0[pat[s],1]+numequ[s])
]=tempequ1[s]:(tempequ1[s+1]-1)
      PCA_ind0[pat[s],1]=PCA_ind0[pat[s],1]+numequ[s]
    } else if (PCA_m[s]==2) {

PCA_ind[pat[s],2,(PCA_ind0[pat[s],2]+1):(PCA_ind0[pat[s],2]+numequ[s])
]=tempequ1[s]:(tempequ1[s+1]-1)
      PCA_ind0[pat[s],2]=PCA_ind0[pat[s],2]+numequ[s]
    }
  }

  size1=matrix(1,N,N)
  for (s1 in 1:m) {
    for (s2 in 1:m) {
      n_tmp=sum(index1[,s1]*index1[,s2])
      if (n_tmp*sum(abs(index1[,s1]-index1[,s2]))!=0) {
        size1[tempequ1[s1]:(tempequ1[s1+1]-
1),tempequ1[s2]:(tempequ1[s2+1]-1)]=nnn[s1]*nnn[s2]/n_tmp
      }
    }
  }

  res=matrix(0,n,m)
  resvar=rep(1,m)

  for (i in 1:n_pat) {
    ind_C[i+1]=ind_C[i]+sum(PCA_ind0[i,])
    ind_pat[i+1]=ind_pat[i]+sum(index1[,min(which(pat==i))])
```

```
  }

  if (is.null(lambda)) {
    for (s in 1:m) {
      res[index1[,s],s]=yy[index1[,s],s]-
XX[index1[,s],,s]%*%(rep(0.01,p))
      g[tempequ1[s]:(tempequ1[s+1]-
1)]=t(XX[index1[,s],index2[,s],s])%*%(res[index1[,s],s])/(resvar[s]*nn
[s])
      Z2[,tempequ1[s]:(tempequ1[s+1]-1)]=apply(XX[,index2[,s],s], 2,
function(x) x*res[,s])/(resvar[s]*nnn[s])
    }


    if (is.null(C0)) {
      #start.time = proc.time()
      C=matrix(0,N,N)
      for (ii in 1:n_pat) {
        C[ind_C[ii]:(ind_C[ii+1]-1),ind_C[ii]:(ind_C[ii+1]-
1)]=t(Z2[ind_pat[ii]:(ind_pat[ii+1]-1),ind_C[ii]:(ind_C[ii+1]-1),
drop=FALSE])%*%Z2[ind_pat[ii]:(ind_pat[ii+1]-
1),ind_C[ii]:(ind_C[ii+1]-1), drop=FALSE]
      }
      #time = proc.time() - start.time
      C=C*size1


      fq=Qfirstdev1(h1=h1, beta_pre=rep(0.01,p), XX=XX, yy=yy,
index1=index1, index2=index2, numequ=numequ, tempequ1=tempequ1,
PCA_ind=PCA_ind, PCA_ind0=PCA_ind0, n_pat=n_pat, eps3=eps3, pat=pat,
ind_C=ind_C, ind_pat=ind_pat, ratio = ratio)$fq_star
      print("Get first derivative 0")
      time=proc.time()
      print(time)
      #time = proc.time() - start.time
      lambda.max=max(abs(fq))+0.5

lambda=exp(seq(log(lambda.min*lambda.max),log(lambda.max),len=nlam))
    } else {
      fq=2*t(fg_temp)%*%solve(C0,g)
      sq=2*t(fg_temp)%*%solve(C0,fg_temp)

      lambda.max=5*max(abs(fq)+abs(sq%*%rep(10,p)))

lambda=exp(seq(log(lambda.min*lambda.max),log(lambda.max),len=nlam))
    }
  }

  L=length(lambda)
  beta=matrix(0,L,p)
  beta_pre=beta0
  res=matrix(0,n,m)
```

```
  qq=rep(0,L)
  biq=rep(0,L)
  aiq=rep(0,L)
  bic=rep(0,L)
  ebic1=rep(0,L)
  ebic2=rep(0,L)
  ebic11=rep(0,L)
  ebic21=rep(0,L)
  ebicM=rep(0,L)
  ebicM1=rep(0,L)
  rss=rep(0,L)
  bic1=rep(0,L)
  rss1=rep(0,L)
  beta_out=matrix(0,L,p)
  intercept=rep(0,L)
  objective=matrix(0,L,2)

  notcon=0

  ind=rep(TRUE, p)
  p1=p

  end=FALSE

  beta_pre2=beta_pre+0.05

  print('Start')            #### Add print
  time=proc.time()
  print(time)

  for (l in 1:L){
    reset=FALSE
    conjugate=FALSE
    alpha=alpha1
    lalpha=length(alpha)
    ff=rep(0,lalpha)
    # if (l==9) {
    #   print(l)
    # }

    if (l!=1) {
      beta_pre=beta[l-1,]
    }
    fq2=Qfirstdev1(h1=h1, beta_pre=beta_pre2, XX=XX, yy=yy,
index1=index1, index2=index2, numequ=numequ, tempequ1=tempequ1,
PCA_ind=PCA_ind, PCA_ind0=PCA_ind0, n_pat=n_pat, eps3=eps3, pat=pat,
ind_C=ind_C, ind_pat=ind_pat, ratio = ratio)$fq_star
    print("Get first derivative 1")
    time=proc.time()
    print(time)
    pe2=rep(0,p)
    if (lambda[l]!=0) {
      for (j in 1:p) {
```

```
        u=abs(beta_pre2[j])
        if (u<=lambda[l]+1e-15) {
          if (u==0) {
            if (fq2[j]>lambda[l]) {
              pe2[j]=-lambda[l]
            } else if (fq2[j]<-lambda[l]) {
              pe2[j]=lambda[l]
            } else {
              pe2[j]=-fq2[j]
            }
          } else {
            pe2[j]=lambda[l]
          }
        } else if (u<=a*lambda[l]) {
          pe2[j]=(a*lambda[l]-u)/(a-1)
        } else {
          pe2[j]=0
        }
      }
    }
    firstdev2=fq2+pe2
    for (i in 1:max.iter) {
      if (lambda[l]!=0) {
        p1=sum(beta_pre!=0)
        ind=(beta_pre!=0)
      } else {
      }

      #if (sum(apply(missid,1,function(x) sum(ind[x], na.rm =
T))!=0)<nm) {
      if (sum(ind)==0) {
        end=TRUE
        break
        # print(beta[l,])
      }


      for (s in 1:m) {
        res[index1[,s],s]=yy[index1[,s],s]-
XX[index1[,s],,s]%*%(as.numeric(beta_pre))
        # resvar[s]=var(res[index1[,s],s])
        g[tempequ1[s]:(tempequ1[s+1]-
1)]=t(XX[index1[,s],index2[,s],s])%*%(res[index1[,s],s])/(resvar[s]*nn
[s])
        Z2[,tempequ1[s]:(tempequ1[s+1]-1)]=apply(XX[,index2[,s],s], 2,
function(x) x*res[,s])/(resvar[s]*nnn[s])
      }

      if (is.null(C0)) {
        C=matrix(0,N,N)
        for (ii in 1:n_pat) {
          C[ind_C[ii]:(ind_C[ii+1]-1),ind_C[ii]:(ind_C[ii+1]-
1)]=t(Z2[ind_pat[ii]:(ind_pat[ii+1]-1),ind_C[ii]:(ind_C[ii+1]-1),
```

```
drop=FALSE])%*%Z2[ind_pat[ii]:(ind_pat[ii+1]-
1),ind_C[ii]:(ind_C[ii+1]-1), drop=FALSE]
        }
        #C=t(Z2)%*%Z2
        C=C*size1


        fq=Qfirstdev1(h1=h1, beta_pre=beta_pre, XX=XX, yy=yy,
index1=index1, index2=index2, numequ=numequ, tempequ1=tempequ1,
PCA_ind=PCA_ind, PCA_ind0=PCA_ind0, n_pat=n_pat, eps3=eps3, pat=pat,
ind_C=ind_C, ind_pat=ind_pat, ratio = ratio)$fq_star
        print("Get first derivative 2")
        time=proc.time()
        print(time)

      } else {
        fq=2*t(fg_temp)%*%solve(C0,g)
        sq=2*t(fg_temp)%*%solve(C0,fg_temp)
      }

      pe=rep(0,p)
      if (lambda[l]!=0) {
        for (j in 1:p) {
          u=abs(beta_pre[j])
          if (u<=lambda[l]+1e-15) {
            if (u==0) {
              if (fq[j]>lambda[l]) {
                pe[j]=-lambda[l]
              } else if (fq[j]<-lambda[l]) {
                pe[j]=lambda[l]
              } else {
                pe[j]=-fq[j]
              }
            } else {
              pe[j]=lambda[l]
            }
          } else if (u<=a*lambda[l]) {
            pe[j]=(a*lambda[l]-u)/(a-1)
          } else {
            pe[j]=0
          }
          pe[j]=pe[j]*sign(beta_pre[j])
        }
      }

      firstdev=fq+pe
      if (conjugate) {
        if (firstdev[ind]%*%conj2[ind]>=0) {
          reset=TRUE
        }
      }
      if (i==1 | reset) {
        reset=FALSE
```

```
        print('before')
        ff = foreach (j = 1:lalpha, .combine = rbind, .packages =
c("MASS", "Matrix"), .export='Qfun1') %dopar% {
          beta_t0=beta_pre[ind]-alpha[j]*firstdev[ind]
          beta_t=beta_pre
          beta_t[ind]=beta_t0

          for (s in 1:m) {
            res[index1[,s],s]=yy[index1[,s],s]-
XX[index1[,s],,s]%*%(as.numeric(beta_t))
            g[tempequ1[s]:(tempequ1[s+1]-
1)]=t(XX[index1[,s],index2[,s],s])%*%(res[index1[,s],s])/(resvar[s]*nn
[s])
            Z2[,tempequ1[s]:(tempequ1[s+1]-
1)]=apply(XX[,index2[,s],s], 2, function(x)
x*res[,s])/(resvar[s]*nnn[s])
          }
          C=matrix(0,N,N)
          for (ii in 1:n_pat) {
            C[ind_C[ii]:(ind_C[ii+1]-1),ind_C[ii]:(ind_C[ii+1]-
1)]=t(Z2[ind_pat[ii]:(ind_pat[ii+1]-1),ind_C[ii]:(ind_C[ii+1]-1),
drop=FALSE])%*%Z2[ind_pat[ii]:(ind_pat[ii+1]-
1),ind_C[ii]:(ind_C[ii+1]-1), drop=FALSE]
          }
          #C=t(Z2)%*%Z2
          C=C*size1
          pe_t=0
          for (jj in 1:p) {
            u=abs(beta_t[jj])
            if (u<=lambda[l]) {
              pe_t=pe_t+lambda[l]*u
            } else if (u<=a*lambda[l]) {
              pe_t=pe_t-(u^2-2*a*lambda[l]*u+lambda[l]^2)/(2*(a-1))
            } else {
              pe_t=pe_t+(a+1)*lambda[l]^2/2
            }
          }
          ff_temp=Qfun1(C=C, g=g, PCA_ind=PCA_ind, PCA_ind0=PCA_ind0,
N=N, n_pat=n_pat, eps3 = eps3, index1=index1, pat=pat, res=res,
numequ=numequ, tempequ1=tempequ1, ind_C=ind_C)$q_star+pe_t
          return(ff_temp)
        }
        print('after')

        beta_tmp=beta_pre[ind]-alpha[which.min(ff)]*firstdev[ind]
        conj2=-firstdev
      } else {
        conjugate=TRUE

threshold=5*sqrt(sum(beta_pre[ind]^2))/(max(alpha)*sqrt(sum(conj2[ind]
^2)))
        if ((conj2[ind]%*%(firstdev2[ind]-firstdev[ind]))==0) {
```

```
          gamma=threshold
        } else {
          gamma=max(0,-
(firstdev[ind]%*%(firstdev[ind]))/(conj2[ind]%*%(firstdev2[ind]-
firstdev[ind])))
          if (gamma>threshold) {
            gamma=threshold
          }
        }
        conj=-firstdev+gamma*conj2        ##### conjugate direction

        print('before')
        ff = foreach (j = 1:lalpha, .combine = rbind, .packages =
c("MASS", "Matrix"), .export='Qfun1') %dopar% {
          beta_t0=beta_pre[ind]+alpha[j]*conj[ind]
          beta_t=beta_pre
          beta_t[ind]=beta_t0
          for (s in 1:m) {
            res[index1[,s],s]=yy[index1[,s],s]-
XX[index1[,s],,s]%*%(as.numeric(beta_t))
            # resvar[s]=var(res[index1[,s],s])
            g[tempequ1[s]:(tempequ1[s+1]-
1)]=t(XX[index1[,s],index2[,s],s])%*%(res[index1[,s],s])/(resvar[s]*nn
[s])
            Z2[,tempequ1[s]:(tempequ1[s+1]-
1)]=apply(XX[,index2[,s],s], 2, function(x)
x*res[,s])/(resvar[s]*nnn[s])
          }
          C=matrix(0,N,N)
          for (ii in 1:n_pat) {
            C[ind_C[ii]:(ind_C[ii+1]-1),ind_C[ii]:(ind_C[ii+1]-
1)]=t(Z2[ind_pat[ii]:(ind_pat[ii+1]-1),ind_C[ii]:(ind_C[ii+1]-1),
drop=FALSE])%*%Z2[ind_pat[ii]:(ind_pat[ii+1]-
1),ind_C[ii]:(ind_C[ii+1]-1), drop=FALSE]
          }
          #C=t(Z2)%*%Z2
          C=C*size1
          pe_t=0
          for (jj in 1:p) {
            u=abs(beta_t[jj])
            if (u<=lambda[l]) {
              pe_t=pe_t+lambda[l]*u
            } else if (u<=a*lambda[l]) {
              pe_t=pe_t-(u^2-2*a*lambda[l]*u+lambda[l]^2)/(2*(a-1))
            } else {
              pe_t=pe_t+(a+1)*lambda[l]^2/2
            }
          }
          ff_temp=Qfun1(C=C, g=g, PCA_ind=PCA_ind, PCA_ind0=PCA_ind0,
N=N, n_pat=n_pat, eps3 = eps3, index1=index1, pat=pat, res=res,
numequ=numequ, tempequ1=tempequ1, ind_C=ind_C)$q_star+pe_t
          return(ff_temp)
        }
```

```
        print('after')

        beta_tmp=beta_pre[ind]+alpha[which.min(ff)]*conj[ind]
        conj2=conj
      }
      print(ff[which.min(ff)])

      if (lambda[l]!=0) {
        beta_tmp[abs(beta_tmp)<eps1]=0
        if (sum(abs(beta_tmp)<eps1)>0) {
          reset=TRUE
        }
      }
      beta[l,ind]=beta_tmp
      firstdev2=firstdev
      if (i>11) {
        diff1=max(abs(beta[l,]-beta_pre2), abs(beta_pre-beta_pre3))
      }
      if (i>10) {
        beta_pre3=beta_pre2
      }
      beta_pre2=beta_pre
      beta_pre=beta[l,]
      if (diff<eps2) {
        break
      } else {
        if (i>11) {
          if (diff1<eps2) {
            alpha=alpha1/max(abs(conj2))
          }
        }
      }
      time=proc.time()
      print(time)
      print(mem_used())
    }
    if (i==max.iter) {
      converge=FALSE
      notcon=notcon+1
      print(diff)
      print(l)
      #break     #test
    } else {
      converge=TRUE
    }
    if (end) {
      break
    }
    # print(l)
    for (s in 1:m) {
      res[index1[,s],s]=yy[index1[,s],s]-
XX[index1[,s],,s]%*%(as.numeric(beta[l,]))
      # resvar[s]=var(res[index1[,s],s])
```

```
      g[tempequ1[s]:(tempequ1[s+1]-
1)]=t(XX[index1[,s],index2[,s],s])%*%(res[index1[,s],s])/(resvar[s]*nn
[s])
      Z2[,tempequ1[s]:(tempequ1[s+1]-1)]=apply(XX[,index2[,s],s], 2,
function(x) x*res[,s])/(resvar[s]*nnn[s])
    }

    if (is.null(C0)) {
      C=matrix(0,N,N)
      for (ii in 1:n_pat) {
        C[ind_C[ii]:(ind_C[ii+1]-1),ind_C[ii]:(ind_C[ii+1]-
1)]=t(Z2[ind_pat[ii]:(ind_pat[ii+1]-1),ind_C[ii]:(ind_C[ii+1]-1),
drop=FALSE])%*%Z2[ind_pat[ii]:(ind_pat[ii+1]-
1),ind_C[ii]:(ind_C[ii+1]-1), drop=FALSE]
      }
      #C=t(Z2)%*%Z2
      C=C*size1

      Q0=Qfun1(C=C, g=g, PCA_ind=PCA_ind, PCA_ind0=PCA_ind0, N=N,
n_pat=n_pat, eps3 = eps3, index1=index1, pat=pat, res=res,
numequ=numequ, tempequ1=tempequ1, ind_C=ind_C)
      print("Get Q function value 4")
      time=proc.time()
      print(time)

      TT=Q0$T2[1:(Q0$count-1),, drop=FALSE]%*%Q0$T1
      g_star=TT%*%g
      C_star=TT%*%C%*%t(TT)
      invCg_star=solve(C_star,g_star)
    } else {
      invCg=solve(C0,g)
    }

    qq[l]=t(g_star)%*%invCg_star
    rss[l]=sum(res[,index0]^2)
    for (k1 in 1:n_pat) {
      rss1[l]=rss1[l]+sum(res[,pat==k1]^2)/sum(pat==k1)
    }
    biq[l]=qq[l]+sum(beta[l,]!=0)*log(n)
    aiq[l]=qq[l]+sum(beta[l,]!=0)*2
    bic[l]=n*log(rss[l])+sum(beta[l,]!=0)*log(n)
    bic1[l]=n*log(rss1[l])+sum(beta[l,]!=0)*log(n)

ebic1[l]=n*log(rss[l])+(log(n)+2*gamma.ebic*log(p))*sum(beta[l,]!=0)

ebic2[l]=n*log(rss[l])+log(n)*sum(beta[l,]!=0)+2*gamma.ebic*log(choose
(p,sum(beta[l,]!=0)))

ebic11[l]=n*log(rss1[l])+(log(n)+2*gamma.ebic*log(p))*sum(beta[l,]!=0)

ebic21[l]=n*log(rss1[l])+log(n)*sum(beta[l,]!=0)+2*gamma.ebic*log(choo
se(p,sum(beta[l,]!=0)))
    ebicM[l]=n*log(rss[l])+(log(n)+2*log(p))*sum(beta[l,]!=0)
```

```
    ebicM1[l]=n*log(rss1[l])+(log(n)+2*log(p))*sum(beta[l,]!=0)
    objective[l,1]=n*log(rss[l])
    objective[l,2]=log(n)*sum(beta[l,]!=0)
    beta_out[l,]=beta[l,]/scalex
    intercept[l] = centery-crossprod(beta_out[l,],as.numeric(centerx))
  }
  returnlist=list("beta"=beta_out, 'biq'=biq, 'aiq'=aiq, 'bic'=bic,
'bic1'=bic1, "ebic1"=ebic1, "ebic2"=ebic2, "ebic11"=ebic11,
"ebic21"=ebic21, "ebicM"=ebicM, "ebicM1"=ebicM1,
'objective'=objective, 'rss'=rss, 'rss1'=rss1, 'res'=res,
'resvar'=resvar, "lambda"=lambda, "notcon"=notcon,
"intercept"=intercept, "beta0"=beta, "XX"=XX, "yy"=yy,
'centerx'=centerx,'scalex'=scalex, 'qq'=qq, 'C'=C, 'C0'=C0)#, 'T1'=T1,
'T2'=T2)
  return(returnlist)
}



Qfun1 <- function (C, g, PCA_ind, PCA_ind0, N, n_pat, eps3=1e-8,
index1, pat, res, numequ, tempequ1, maxeigen=NULL, ind_C) {
  U_ind=array(NA,dim=c(n_pat,2,N))
  U_ind0=matrix(0,n_pat,2)
  T1=diag(1,N,N)
  T2=matrix(0,N,N)

  if (is.null(maxeigen)){
    maxeigen=base::norm((C+t(C))/2, type = '2')
  }

  count=1
  for (k1 in 1:n_pat) {
    id_im=which(pat==k1)
    n_im=length(id_im)
    if (n_im>1) {
      PCA_ind[k1,2,]=NA
      PCA_ind0[k1,2]=0
      for (i in 2:n_im) {
        if (sum(abs(res[,id_im[1]]-res[,id_im[i]]))==0) {     ##### If
residuals of other layers are the same as the main layer in the same
group?
          id_im[i]=0
        } else {

PCA_ind[k1,2,(PCA_ind0[k1,2]+1):(PCA_ind0[k1,2]+numequ[id_im[i]])]=tem
pequ1[id_im[i]]:(tempequ1[id_im[i]+1]-1)
          PCA_ind0[k1,2]=PCA_ind0[k1,2]+numequ[id_im[i]]
        }
      }
    }
    for (k2 in 1:2) {
      if (PCA_ind0[k1,k2]!=0) {
        if (k2==1 | U_ind0[k1,1]==0) {
```

```
          cov_temp=C[PCA_ind[k1,k2,1:PCA_ind0[k1,k2]],
PCA_ind[k1,k2,1:PCA_ind0[k1,k2]], drop=FALSE]
        } else {
          temp1=T2[U_ind[k1,1,1:U_ind0[k1,1]],
PCA_ind[k1,1,1:PCA_ind0[k1,1]], drop=FALSE]
          temp2=solve(temp1%*%C[PCA_ind[k1,1,1:PCA_ind0[k1,1]],
PCA_ind[k1,1,1:PCA_ind0[k1,1]], drop=FALSE]%*%t(temp1))
          temp3=C[PCA_ind[k1,k2,1:PCA_ind0[k1,k2]],
PCA_ind[k1,1,1:PCA_ind0[k1,1]], drop=FALSE]%*%t(temp1)
          T1[PCA_ind[k1,k2,1:PCA_ind0[k1,k2]],
PCA_ind[k1,1,1:PCA_ind0[k1,1]]]=-temp3%*%temp2%*%temp1

temp_ind=c(PCA_ind[k1,1,1:PCA_ind0[k1,1]],PCA_ind[k1,k2,1:PCA_ind0[k1,
k2]])
          temp4=T1[temp_ind, temp_ind,
drop=FALSE]%*%C[temp_ind,temp_ind]%*%t(T1[temp_ind,temp_ind])

cov_temp=temp4[(PCA_ind0[k1,1]+1):(PCA_ind0[k1,1]+PCA_ind0[k1,2]),(PCA
_ind0[k1,1]+1):(PCA_ind0[k1,1]+PCA_ind0[k1,2])]
          #cov_temp=C[PCA_ind[k1,k2,1:PCA_ind0[k1,k2]],
PCA_ind[k1,k2,1:PCA_ind0[k1,k2]], drop=FALSE]-temp3%*%temp2%*%t(temp3)
        }
        eigens=eigen((cov_temp+t(cov_temp))/2)
        values=eigens$values
        if (max(values)>maxeigen) {
          maxeigen=max(values)
        }
        # eigenthreshold=max(eps3, maxeigen*1e-10)
        n_tmp0=sum(index1[,min(which(pat==k1)), drop=FALSE])   ###
Same as pQIFmp_scad20.R, except using the BIC in Cho's paper and ask
number of selected PCA less than n
        nr=n_tmp0*PCA_ind0[k1,k2]
        if (k2==1) {                                          ### Ask
number of selected PCA less than n
          tm=n_tmp0-1
        } else {
          tm=n_tmp0-1-U_ind0[k1,1]
        }
        eigenthreshold=max(eps3, maxeigen*1e-10,
sum(values)*log(nr)/(nr))
        rank=rankMatrix(cov_temp)
        if (min(values)>eigenthreshold & rank==min(dim(cov_temp)) &
min(dim(cov_temp))<=tm) {
          U_ind0[k1,k2]=PCA_ind0[k1,k2]
          U_ind[k1,k2,1:U_ind0[k1,k2]]=count:(count+U_ind0[k1,k2]-1)
          #nu_num[count+1]=nu_num[count]+PCA_ind0[k1,k2]
          T2[U_ind[k1,k2,1:U_ind0[k1,k2]],
PCA_ind[k1,k2,1:PCA_ind0[k1,k2]]]=diag(1,PCA_ind0[k1,k2],PCA_ind0[k1,k
2])
        } else if (max(values)>eigenthreshold & rank>0 & tm>0) {
          U_ind0[k1,k2]=max(1,min(apply(index1[,which(pat==k1),
drop=FALSE],2,sum), sum(values>eigenthreshold), rank, tm))
          U_ind[k1,k2,1:U_ind0[k1,k2]]=count:(count+U_ind0[k1,k2]-1)
```

```
          U_temp=as.matrix(eigens$vectors[,1:U_ind0[k1,k2]])
          T2[U_ind[k1,k2,1:U_ind0[k1,k2]],
PCA_ind[k1,k2,1:PCA_ind0[k1,k2]]]=t(U_temp%*%diag(sign(U_temp[dim(U_te
mp)[1],]), length(sign(U_temp[dim(U_temp)[1],])),
length(sign(U_temp[dim(U_temp)[1],]))))
        }
        count=count+U_ind0[k1,k2]
      }
    }
  }

  ind_tT=rep(1,n_pat+1)
  #ind_Cstar=rep(1,n_pat+1)
  tempT=matrix(0,(count-1),N)
  C_star=matrix(0,(count-1),(count-1))
  for (ii in 1:n_pat) {
    ind_tT[ii+1]=ind_tT[ii]+sum(U_ind0[ii,])
    #ind_Cstar[ii+1]=ind_Cstar[ii]+sum(PCA_ind0[ii,])
    tempT[ind_tT[ii]:(ind_tT[ii+1]-1),ind_C[ii]:(ind_C[ii+1]-
1)]=T2[ind_tT[ii]:(ind_tT[ii+1]-1),ind_C[ii]:(ind_C[ii+1]-1),
drop=FALSE]%*%T1[ind_C[ii]:(ind_C[ii+1]-1),ind_C[ii]:(ind_C[ii+1]-1),
drop=FALSE]
    C_star[ind_tT[ii]:(ind_tT[ii+1]-1),ind_tT[ii]:(ind_tT[ii+1]-
1)]=tempT[ind_tT[ii]:(ind_tT[ii+1]-1),ind_C[ii]:(ind_C[ii+1]-1),
drop=FALSE]%*%C[ind_C[ii]:(ind_C[ii+1]-1),ind_C[ii]:(ind_C[ii+1]-1),
drop=FALSE]%*%t(tempT[ind_tT[ii]:(ind_tT[ii+1]-
1),ind_C[ii]:(ind_C[ii+1]-1), drop=FALSE])
  }
  q_star=t(g_star)%*%solve(C_star,g_star)

  # time5 = proc.time() - start.time
  returnlist=list('q_star'=q_star, 'C_star'=C_star, 'g_star'=g_star,
'T1'=T1, 'T2'=T2, 'count'=count, 'tempT'=tempT)
  return(returnlist)
}


Qfirstdev1 <- function (h1=2^(-(8:30)), beta_pre, XX, yy, index1,
index2, numequ, tempequ1, PCA_ind, PCA_ind0, n_pat, eps3, pat, ind_C,
ind_pat, ratio) {
  dims=dim(XX)
  n=dims[1]
  p=dims[2]
  m=dims[3]
  res=matrix(0,n,m)
  res1=matrix(0,n,m)
  res2=matrix(0,n,m)
  resvar=rep(1,m)
  resvar1=rep(1,m)
  resvar2=rep(1,m)
  N=sum(index2)
  Z2=matrix(0,n,N)
  Z21=matrix(0,n,N)
```

```
  Z22=matrix(0,n,N)
  g=rep(0,N)
  g1=rep(0,N)
  g2=rep(0,N)
  nn=apply(index1,2,sum)
  nnn=sqrt(apply(index1,2,sum))


  for (s in 1:m) {
    res[index1[,s],s]=yy[index1[,s],s]-
XX[index1[,s],,s]%*%(as.numeric(beta_pre))
    # resvar1[s]=var(res1[index1[,s],s])
    g[tempequ1[s]:(tempequ1[s+1]-
1)]=t(XX[index1[,s],index2[,s],s])%*%(res[index1[,s],s])/(resvar[s]*nn
[s])
    Z2[,tempequ1[s]:(tempequ1[s+1]-1)]=apply(XX[,index2[,s],s], 2,
function(x) x*res[,s])/(resvar[s]*nnn[s])
  }
  C=matrix(0,N,N)
  for (ii in 1:n_pat) {
    C[ind_C[ii]:(ind_C[ii+1]-1),ind_C[ii]:(ind_C[ii+1]-
1)]=t(Z2[ind_pat[ii]:(ind_pat[ii+1]-1),ind_C[ii]:(ind_C[ii+1]-1),
drop=FALSE])%*%Z2[ind_pat[ii]:(ind_pat[ii+1]-
1),ind_C[ii]:(ind_C[ii+1]-1), drop=FALSE]
  }
  #C=t(Z2)%*%Z2
  maxeigen=base::norm((C+t(C))/2, type = '2')
  #print(maxeigen)

  qfun=Qfun1(C=C, g=g, PCA_ind=PCA_ind, PCA_ind0=PCA_ind0, N=N,
n_pat=n_pat, eps3 = eps3, index1=index1, pat=pat, res=res,
numequ=numequ, tempequ1=tempequ1, maxeigen=maxeigen, ind_C=ind_C)
  tempT=qfun$tempT
  C_star=qfun$C_star
  S_C_star=solve(C_star)


  #fq_star=rep(0,p)
  # start.time = proc.time()
  result = foreach (j1 = 1:p, .combine = rbind, .packages = c("MASS",
"Matrix"), .export='Qfun1') %dopar% {
    h=h1
    if (abs(beta_pre[j1])<1e-2 & abs(beta_pre[j1])>0) {
      h=h1*max(1e-2, abs(beta_pre[j1]))
    }
    temp_fq=0
    temp_sq=0
    temp_diff1=Inf
    temp_diff2=Inf
    for (hh in 1:length(h)) {
      # start.time = proc.time()
      beta_temp1=beta_pre
      beta_temp2=beta_pre
```

```
      beta_temp1[j1]=beta_pre[j1]+h[hh]
      beta_temp2[j1]=beta_pre[j1]-h[hh]

      for (s in 1:m) {
        res1[index1[,s],s]=yy[index1[,s],s]-
XX[index1[,s],,s]%*%(as.numeric(beta_temp1))
        g1[tempequ1[s]:(tempequ1[s+1]-
1)]=t(XX[index1[,s],index2[,s],s])%*%(res1[index1[,s],s])/(resvar1[s]*
nn[s])
        res2[index1[,s],s]=yy[index1[,s],s]-
XX[index1[,s],,s]%*%(as.numeric(beta_temp2))
        g2[tempequ1[s]:(tempequ1[s+1]-
1)]=t(XX[index1[,s],index2[,s],s])%*%(res2[index1[,s],s])/(resvar2[s]*
nn[s])
      }

      g_star1=tempT%*%g1
      g_star2=tempT%*%g2
      q_star1=t(g_star1)%*%S_C_star%*%g_star1
      q_star2=t(g_star2)%*%S_C_star%*%g_star2


      fq_star_par=(q_star1-q_star2)/(2*h[hh])
      #print(sq_star[j1,j1])
      if (hh==1) {
        fq_pre=fq_star_par
      } else {
        if (fq_diff<temp_diff1) {
          temp_diff1=fq_diff
          temp_fq=fq_star_par
        }
        if (fq_diff<1e-3) {
          break
        } else {
          fq_pre=fq_star_par
        }
      }
      # time3 = proc.time() - start.time
    }
    if (fq_diff>1e-3) {
      fq_star_par=temp_fq
      # print(paste(j1,l,i))
    }
    # print(hh)
    return(c(fq_star_par,q_star1,q_star2))
  }
  # time = proc.time() - start.time
  returnlist=list('fq_star'=result[,1]+rnorm(p,0,0.1),
'q_star1'=result[,2], 'q_star2'=result[,3])
  return(returnlist)
}
```

```
##################### 'imputeglm.predict.R' ####################

library(MASS)
library(glmnetcr)
imputeglm.predict <- function (X, ind_y, ind_x=-ind_y, miss, newdata,
family="gaussian") {
  ny=length(ind_y)
  nx=length(X[1,ind_x])+1
  B=matrix(0,nx,ny)
  PRED=matrix(0,dim(newdata)[1],ny)
  for (l in 1:ny) {
    ind_obs=!miss[,ind_y[l]]

    x.t=X[ind_obs,ind_x]
    y.t=X[ind_obs,ind_y[l]]
    x.train=x.t[apply(miss[ind_obs,ind_x],1,sum)==0,]
    y.train=y.t[apply(miss[ind_obs,ind_x],1,sum)==0]
    data0=data.frame(y.train, x.train)
    newx=as.data.frame(newdata)
    colnames(newx)=colnames(data0)[-1]
    if (family=="ordinal") {
      data0[,1]=as.factor(data0[,1])
      if (dim(x.train)[1]>dim(x.train)[2]) {
        fit=polr(y.train~., data = data0)
        coeff=c(0,fit$coef)                          ##### The intercept
may be wrong
        pred=as.numeric(as.character(predict(fit, type = 'class',
newdata=newx)))
      } else {
        fit <- glmnetcr(data0[,-1], data0[,1], maxit=500)
        select = select.glmnetcr(fit)
        coeff=fit$beta[1:nx,select]                  ##### The intercept
may be wrong
        pred=as.numeric(fitted(fit, newx=newx, s=select)$class)
      }
    } else if (family=="binomial") {
      data0[,1]=as.factor(data0[,1])
      if (dim(x.train)[1]>dim(x.train)[2]) {
        fit=glm(y.train~., family=family, data=data0)
        coeff=fit$coef
        prob=predict(fit,newdata=newx, type = 'response')
        pred=rep(as.numeric(levels(data0$y.train)[1]),dim(newdata)[1])
        pred[prob>0.5]=as.numeric(levels(data0$y.train)[2])
      } else {
        fit=glmnet(x.train, y.train, family=family)
        k <- fit$df
        n <- fit$nobs
        select=which.min(log(n)*k+deviance(fit))
        coeff=t(as.numeric(coef(fit, s=select)))
```

```
        pred=as.numeric(predict(fit, newx=newdata, s=select, type =
'class'))
      }
    } else {
      if (dim(x.train)[1]>dim(x.train)[2]) {
        fit=glm(y.train~., family=family, data=data0)
        coeff=fit$coef
        pred=predict(fit,newdata=newx)
      } else {
        cvfit <- cv.glmnet(x.train, y.train, nfolds=3, family=family)
        fit <- cvfit$glmnet.fit
        coeff=t(as.numeric(coef(fit, s=cvfit$lambda.min)))
        pred=predict(fit, newx=newdata, s=cvfit$lambda.min)
      }
    }
    B[,l]=coeff
    PRED[,l]=pred
  }
  returnlist=list("B"=B, "PRED"=PRED)
  return(returnlist)
}
```

```
################## 'mySCAD5.R' #################
mySCAD <- function (X, y, lambda=NULL, a=3.7, eps = 0.001,
lambda.min=ifelse(n>p,.001,.05), max.iter = 1000, nlam=100,
standardize=TRUE, gamma.ebic=0.5) {   #y should be centered
  n = nrow(X)
  p = ncol(X)
  center=rep(0,p)
  scale=rep(1,p)
  XX=X
  v=rep(n,p)     #inner product of columns of XX
  if (standardize==TRUE) {
    for (i in 1:p) {
      center[i]=mean(X[,i])
      Xtmp=X[,i]-center[i]
      if (sqrt(sum((Xtmp)^2)/n)>0) {
        scale[i]=sqrt(sum((Xtmp)^2)/n)
      }
      XX[,i]=Xtmp/scale[i]
    }
  } else {
    for (i in 1:p) {
      center[i]=mean(X[,i])
      XX[,i]=X[,i]-center[i]
      v[i]=crossprod(XX[,i],XX[,i])
    }
  }
```

```
  yy=y-mean(y)
  m=n               #I think packages 'plus' uses n
  if (a<=(m/min(v)+1)) {
    print(paste("Error: a should be larger than ", m/min(v)+1))
    a=m/min(v)+1.05                #plus 0.05, change a
  }
  user=TRUE
  if (is.null(lambda)) {
    fit.lam <- glm(yy ~ 1, family = "gaussian")
    lambda.max=max(abs(crossprod(XX,fit.lam$residuals)/m))

lambda=exp(seq(log(lambda.max),log(lambda.min*lambda.max),len=nlam))
    user=FALSE
  }
  L=length(lambda)
  beta=matrix(0,L,p)
  beta_out=matrix(0,L,p)
  intercept=rep(0,L)
  rss=rep(0,L)        #residual sum of squares
  gcv=rep(0,L)
  bic=rep(0,L)
  ebic1=rep(0,L)
  ebic2=rep(0,L)
  ebicM=rep(0,L)          ####### With the largest gamma.ebic
  beta_pre=rep(0,p)
  e=rep(0,p)
  res=yy
  notcon=0

  if (user) {
    lstart=1
  } else {
    rss[1]=crossprod(yy,yy)
    gcv[1]=rss[1]/(n*(1-sum(beta[1,]!=0)/n)^2)    #new formula in
paper???
    bic[1]=n*log(rss[1])+log(n)*sum(beta[1,]!=0)    #cv??

ebic1[1]=n*log(rss[1])+(log(n)+2*gamma.ebic*log(p))*sum(beta[1,]!=0)

ebic2[1]=n*log(rss[1])+log(n)*sum(beta[1,]!=0)+2*gamma.ebic*log(choose
(p,sum(beta[1,]!=0)))
    ebicM[1]=n*log(rss[1])+(log(n)+2*log(p))*sum(beta[1,]!=0)
####### With the largest gamma.ebic
    beta_out[1,]=beta[1,]/scale
    intercept[1] = mean(y)-crossprod(center, beta_out[1,])
    lstart=2
  }
  for (l in lstart:L) {
    if (l!=1) {
      beta_pre=beta[l-1,]
    }
    i=1
    while (i<max.iter) {
```

```
      while (i<max.iter) {
        i=i+1
        #print(i)
        tmp=0
        for (j in 1:p) {
          if (e[j]!=0){
            u=crossprod(XX[,j],res)/v[j]+beta_pre[j]      #least
squares estimator
            s=0
            if (u>0) s=1
            if (u<0) s=-1
            lamscaled=lambda[l]*m/v[j]
            if (abs(u)<=(lamscaled+1e-15)) {
              beta[l,j]=0
            } else if (abs(u)<=(lambda[l]+lamscaled)) {
              beta[l,j]=s*(abs(u)-lamscaled)
            } else if (abs(u)<=a*lambda[l]) {
              beta[l,j]=s*(abs(u)-a*lamscaled/(a-1))/(1-m/((a-
1)*v[j]))
            } else {
              beta[l,j]=u
            }
            shift=beta[l,j]-beta_pre[j]
            if (shift!=0) {
              res=res-shift*XX[,j]
              if (abs(shift/beta_pre[j])>tmp)
tmp=abs(shift/beta_pre[j])
            }
          }
        }
        beta_pre=beta[l,]
        if (tmp<eps) {
          break
        }
      }
      violations=0;
      for (j in 1:p)
      {
        if (e[j]==0)
        {
          u=crossprod(XX[,j],res)/v[j]+beta_pre[j]      #least squares
estimator
          s=0
          if (u>0) s=1
          if (u<0) s=-1
          lamscaled=lambda[l]*m/v[j]
          if (abs(u)<=(lamscaled+1e-15)) {
            beta[l,j]=0
          } else if (abs(u)<=(lambda[l]+lamscaled)) {
            beta[l,j]=s*(abs(u)-lamscaled)
          } else if (abs(u)<=a*lambda[l]) {
            beta[l,j]=s*(abs(u)-a*lamscaled/(a-1))/(1-m/((a-1)*v[j]))
          } else {
```

```
            beta[l,j]=u
          }

          if (beta[l,j]!=0)
          {
            e[j]=1
            res=res-beta[l,j]*XX[,j]
            beta_pre[j]=beta[l,j]
            violations=violations+1
          }
        }
      }
      if (violations==0) break;
    }
    if (i==max.iter) {
      converge=FALSE
      notcon=notcon+1
      print(l)
    } else {
      converge=TRUE
    }
    rss[l]=crossprod(res,res)
    gcv[l]=rss[l]/(n*(1-sum(beta[l,]!=0)/n)^2)    #new formula in
paper???
    bic[l]=n*log(rss[l])+log(n)*sum(beta[l,]!=0)    #cv??

ebic1[l]=n*log(rss[l])+(log(n)+2*gamma.ebic*log(p))*sum(beta[l,]!=0)

ebic2[l]=n*log(rss[l])+log(n)*sum(beta[l,]!=0)+2*gamma.ebic*log(choose
(p,sum(beta[l,]!=0)))
    ebicM[l]=n*log(rss[l])+(log(n)+2*log(p))*sum(beta[l,]!=0)
####### With the largest gamma.ebic
    beta_out[l,]=beta[l,]/scale
    intercept[l] = mean(y)-crossprod(center, beta_out[l,])
  }
  returnlist=list("beta"=beta_out, "bic"=bic, "ebic1"=ebic1,
"ebic2"=ebic2, "ebicM"=ebicM, "gcv"=gcv, "rss"=rss, "lambda"=lambda,
"notcon"=notcon, "intercept"=intercept, "beta0"=beta, "XX"=XX,
"yy"=yy)
  return(returnlist)
}



#################### 'DISCOM.R' ####################
compute.xtx<-function(x,robust=0,k_value=1.5) # robust=1 for huber
estimate, k_value is used in huber function
{
  p=ncol(x)
  cov.matrix=matrix(NA,p,p)
  for(i in 1:p)
  {
    for(j in 1:p){
```

```
      index=which((!is.na(x[,i]))&(!is.na(x[,j])))
      x1=x[index,i]
      x2=x[index,j]
      if(robust==0){cov.matrix[i,j]=sum((x1-mean(x1))*(x2-
mean(x2)))/(length(index)-1)}
      else {cov.matrix[i,j]=huberM((x1-mean(x1))*(x2-
mean(x2)),k=k_value*sqrt(length(index)/log(p)))$mu}
    }
  }
  cov.matrix
}

compute.xty<-function(x,y,robust=0,k_value=1.5)# robust=1 for huber
estimate, k_value is used in huber function
{
  p=ncol(x)
  cov.vector=rep(NA,p)
  for(i in 1:p){
    index=which((!is.na(x[,i])))
    x1=x[index,i]
    x2=y[index]
    if(robust==0){cov.vector[i]=sum((x1-mean(x1))*(x2-
mean(x2)))/(length(index)-1)}
    else {cov.vector[i]=huberM((x1-mean(x1))*(x2-
mean(x2)),k=k_value*sqrt(length(index)/log(p)))$mu}
  }
  cov.vector
}

compute.mean<-function(x)
{
  p=ncol(x)
  means=rep(NA,p)
  for(i in 1:p){means[i]=mean(x[,i],na.rm=T)}
  means
}

compute.sd<-function(x)
{
  p=ncol(x)
  sds=rep(NA,p)
  for(i in 1:p){sds[i]=sd(x[,i],na.rm=T)}
  sds
}
```