```
#########################################################################
#########################################################################
########          Sparse Multiple Regression Analysis          ########
########                                                       ########
########          author : Naoto Yamashita                     ########
########     which minimizes f(W) = ||Y - XW||^2 over W        ########
########     subject to W has perfect cluster structure        ########
########       (having only one non-zero element in each row)  ########
########     where  X : matrix of independent variables        ########
########            Y : matrix of dependent variables          ########
########            W : matrix of regression coefficients      ########
########     the following codes are available for S-plus/R    ########
#########################################################################
#########################################################################


##Note that the names of independent and dependent variables matrices
## must be X and Y.


### the main function of Sparse Multiple Regression

SMR <- function(X,Y){

        N <- nrow(X)
        P <- ncol(X)
        Q <- ncol(Y)




        ### the sub function of SMR for multiple-starts approach
        SMRsub <- function(X,Y){

                N <- nrow(X)
                P <- ncol(X)
                Q <- ncol(Y)



                ##Objective function to be minimized f(W) = ||Y - XW||^2
                f <- function(W){
                        sum((Y - X%*%W)^2)
                }



                ##Update functions


                ###Update (p,theta(p))element of W with fixed theta(p)=k

                Updater <- function(W,p,k){

                        N <- nrow(X)
                        P <- ncol(X)
                        Q <- ncol(Y)

                        sums <- 0
                        for(n in 1:N){
                                Xp <- c(X[n,-p]) #vector without pth element
                                Wp <- c(W[-p,k]) #vector without kth element
                                sums <- sums + X[n,p] * t(Xp) %*% Wp
                        }

                        W[p,k] <- (X[,p] %*% Y[,k] - sums) / sum((X[,p])^2)
                        W[p,k]

                }
```

```
###Update column-number theta(p)
###step-by-step computation to evaluate the "risk" of Wik = nonz
ero
whichnonzero <- function(W,p){

        N <- nrow(X)
        P <- ncol(X)
        Q <- ncol(Y)


        f <- function(W){
                sum((Y - X%*%W)^2)
        }



        Updater <- function(W,p,k){
                N <- nrow(X)
                P <- ncol(X)
                Q <- ncol(Y)
                sums <- 0
                for(n in 1:N){
                        Xp <- c(X[n,-p]) #vector without pth ele
ment
                        Wp <- c(W[-p,k]) #vector without kth ele
ment
                        sums <- sums + X[n,p] * t(Xp) %*% Wp
                }

                 W[p,k] <- (X[,p] %*% Y[,k] - sums) / sum((X[,p]
)^2)
                 W[p,k]
        }


         fval <- c()  ##vector contains function values for pth
row
         for(k in 1:Q){

                Wkari <- W
                Wkari[p,] <- rep(0,Q)
                Wkari[p,k] <- Updater(W,p,k)
                fval[k] <- f(Wkari)

        }

        list(fval,which.min(fval))

}




##initialize random start
W <- matrix(rnorm(P*Q),P,Q) #initial value of W



##Start Iteration (max 100 runs)
history <- c() #vector of function value
history[1] <- f(W)


##iteration start

for(ite in 1:100){
```

```
                        for(i in 1:P){
                                Wnew <- W
                                Wnew[i,] <- rep(0,Q)
                                nonzerocol <- whichnonzero(W,i)[[2]]  ##theta-st
ep ("position" of nonzero element)

                                Wnew[i,nonzerocol] <- Updater(W,i,nonzerocol)  #
#W-step ("value" of nonzer element)

                                W <- Wnew

                        }


                        history[ite+1] <- f(W) ##evaluate the current function v
alue


                        ##stopping rule

                        if((history[ite] - history[ite+1]) < 0.00001){
                                break
                        }



                } ##iteration end



                ##ordinal least squares estimate (for comparison)
                Wols <- solve(t(X)%*%X) %*% t(X) %*% Y


                ##return
                list(history,W,Wols)

        }
        ##sub-function ends







        ##multiple-starts approach for finding two best/equivalent solutions

        flist <- c()  ##vector of minimal function values
        reslist <- list() ##list of results
        lmcount <- 0 ##local-minimum counter
        finalsol <- NULL ##final-solution

        f <- function(W){
                sum((Y - X%*%W)^2)
        }




        ##the function for evaluating congruence of two matrices (Adachi, 2011)
        ##used for evaluating equivalence of two solutions

        SS <- function(X,Xt){

                #Xt <- scale(as.matrix(Xt))
                #X <- scale(as.matrix(X))
```

```
        m <- nrow(X)
        p <- ncol(X)
        unitm <- c(rep(1,m))
        unitp <- c(rep(1,p))
        Ub <- unitm %*% t(unitm) %*% (X + Xt) %*% unitp %*% t(unitp) / (
2*m*p)

        1 - (0.5*sum((X-Xt)^2) / (sum((X - Ub)^2) + sum((Xt - Ub)^2)))
}



##step1 : intial M runs
M <- 3
for(m in 1:M){

        reslist[[m]] <- SMRsub(X,Y)
        flist[m] <- min(reslist[[m]][[1]])

}


##step 2 : finding the best solutions within Mini solutions
tmin <- which.min(flist)
reshat <- reslist[[tmin]]
What <- reslist[[tmin]][[2]] ##current "best solution"


##step 3  finding two best solutions within M sets of solutions
for(m in 1:M){
        if(m != tmin){
                if(SS(What, reslist[[m]][[2]]) > 0.999999){
                        finalsol <- reslist[[m]]
                        lmcount <- m-1
                        break
                }
        }
}


##repat until 2 equivalently best solutions are found
##Max 10 runs
Mmax <- 10

for(m in (M+1):Mmax){

        if(is.null(finalsol) == TRUE){

                reslist[[m]] <- SMRsub(X,Y)  ##Step 4
                restilda <- reslist[[m]]
                Wtilda <- restilda[[2]]


                if(round(f(What),12) >= round(f(Wtilda),12)){  ##Step 4
                        tmin <- m

                        if(SS(What,Wtilda) > 0.00001){  ##Step 5
                                finalsol <- reslist[[tmin]]
                                lmcount <- m - 1
                        }

                }


                if(m == Mmax){   ##when maximal iterations reached
                        finalsol <- reslist[[tmin]]
                        lmcount <- Mmax
                }
```

```
                }

        }


        What <- finalsol[[2]]
        Wols <- finalsol[[3]]



        ##Decompose What to D(diag) and R(binary) with What = DR
        one <- rep(1,Q)
        D <- diag(c(What%*%one))
        R <- matrix(0,P,Q)

        for(p in 1:P){

                 for(q in 1:Q){

                        if(What[p,q] != 0){
                        R[p,q] <- 1}
                 }
        }




        list(coefficients=What,
             D=D,
             R=R,
             fanction.value=min(flist),
             local.minimum=lmcount,
             ordinal.least.squares.estimates=Wols)



}##SMR ends
```

```
#########################################################################
#########################################################################
########                Simulation codes for                     ########
########         Sparse Multiple Regression Analysis             ########
#########################################################################
#########################################################################



##data description
N <- 30  ##number of observations
P <- 10  ##number of independent variables
Q <- 5   ##number of dependent variables


##rate of variance explained
rho <- 0.70



###data generation
Datagen <- function(rho){

        ##Matrix of indepent variables
        X <- matrix(rnorm(N*P),N,P)


        ##Matrix of Error
        E <- matrix(rnorm(N*Q),N,Q)



        ##True value Wt = Dt%*%Rt

        Dt <- diag(rnorm(P)) ##diagonal matrix indicating "values" of nonzero el
ements
        Rt <- matrix(0,P,Q) ##binary matrix indicating "potitions" of nonzero el
ements
        for(p in 1:P){
                Rt[p,round(runif(1,0.5,Q+0.4999))] <- 1
        }



        Wt <- Dt %*% Rt    ##Construct Wt from Dt and Rt




        #the function which controls the variance explained
        theta <- function(Xhat,E,rho){
                sqrt(((1-rho)/rho) * (sum(Xhat^2)/sum(E^2)))
        }



        #Data construction
        Y <- X %*% Wt + theta(X%*%Wt,E,rho)*E



        #Decompose Wt to D and R
        one <- rep(1,Q)
        Dt <- diag(c(Wt%*%one))  ##value
        Rt <- matrix(0,P,Q)  ##position

        for(p in 1:P){
                for(q in 1:Q){
```

```
                        if(Wt[p,q] != 0){
                        Rt[p,q] <- 1}
                }
        }

        list(X,Y,Wt,Dt,Rt)

}



##For evaluation

##||vec(D) - vec(Dt)||^2
ESSD <- function(D,Dt){

        vecD <- c(diag(D))
        vecDt <- c(diag(Dt))
        sum((vecD - vecDt)^2)

}


##trRtR'/P
EP <- function(R,Rt){
  P <- nrow(R)
  sum(diag(Rt %*% t(R))) / P
}



#standardized similarity
SS <- function(X,Xt){

        m <- nrow(X)
        p <- ncol(X)
        unitm <- c(rep(1,m))
        unitp <- c(rep(1,p))
        Ub <- unitm %*% t(unitm) %*% (X + Xt) %*% unitp %*% t(unitp) / (2*m*p)

        1 - (0.5*sum((X-Xt)^2) / (sum((X - Ub)^2) + sum((Xt - Ub)^2)))
}



##Congruence coefficient
CC <- function(X,Xt){
        sum(diag(t(X)%*%Xt)) / sqrt(sum(X^2) * sum(Xt^2))
}



##vectors of results
resultsSS <- c()
resultsCC <- c()
resultsD <- c()
resultsR <- c()
resultsminima <- c()




##Simulation Start

for(ite in 1:100){
        res <- Datagen(rho)
        X <- res[[1]]
        Y <- res[[2]]
        Wt <- res[[3]]
```

```
        Dt <- matrix(diag(res[[4]]),P,1)
        Rt <- res[[5]]

        ##apply SMR to X and Y
        res <- SMR(X,Y)

        W <- res$coefficients
        D <- matrix(diag(res$D),P,1)
        R <- res$R

        ##Evaluation
        resultsSS[ite] <- SS(Wt,W)
        resultsCC[ite] <- CC(Wt,W)
        resultsD[ite] <- ESSD(D,Dt)
        resultsR[ite] <- EP(R,Rt)
        resultsminima[ite] <- res[[5]]

  print(ite)

}
```

```
##########################################################################
##########################################################################
########          Sparse Principal Component Analysis           ########
########    as an extention of Sparse Multiple Regression       ########
########                                                        ########
########          author : Naoto Yamashita                      ########
########   which minimizes f(F,A) = ||X - FA'||^2 over F and A  ########
########     subject to A has perfect cluster structure         ########
########       (having only one non-zero element in each row)   ########
########   where  X : N(sample) times P(variables) data matrix  ########
########          F : matrix of principal component scores      ########
########          A : matrix of loadings                        ########
########     the following codes are available for S-plus/R     ########
##########################################################################
##########################################################################



##Note that the name of input dataset must be "Mat"


SPCA <- function(Mat,r){


        N <- nrow(Mat)
        P <- ncol(Mat)


        ##Mat : N times P data matrix
        ##r : number of principal components




        ##the sub function of SPCA for multiple-starts approach

        SPCAsub <- function(Mat,r){

                ## a modification of Sparse Multiple Regression for SPCA
                ## for obtaining the solution having only one non-zero element i
n each COLUMN


                SMR.mod.for.SPCA <- function(X,Y){


                        N <- nrow(X)
                        P <- ncol(X)
                        Q <- ncol(Y)



                        ##Objective function to be minimized f(W) = ||Y - XW||^2

                        f <- function(W,X,Y){
                                sum((Y - X%*%W)^2)
                        }



                        ##Update functions


                        ###Update (p,theta(p))element of W with fixed theta(p)=k


                        Updater <- function(W,p,k,X,Y){

                                N <- nrow(X)
```

```
                P <- ncol(X)
                Q <- ncol(Y)

                sums <- 0
                for(n in 1:N){
                        Xp <- c(X[n,-p]) #vector without pth ele
ment
                        Wp <- c(W[-p,k]) #vector without kth ele
ment
                        sums <- sums + X[n,p] * t(Xp) %*% Wp
                }

                 W[p,k] <- (X[,p] %*% Y[,k] - sums) / sum((X[,p]
)^2)
                 W[p,k]
        }


        ###Update row-number of nonzero-element
        ###step-by-step computation to evaluate the "risk" of Wp
q = nonzero
        whichnonzero <- function(W,q,X,Y){

                P <- nrow(W)


                f <- function(W,X,Y){
                        sum((Y - X%*%W)^2)
                }



                Updater <- function(W,p,k,X,Y){

                        N <- nrow(X)
                        P <- ncol(X)
                        Q <- ncol(Y)

                        sums <- 0
                        for(n in 1:N){
                                Xp <- c(X[n,-p]) #vector without
 pth element
                                Wp <- c(W[-p,k]) #vector without
 kth element
                                sums <- sums + X[n,p] * t(Xp) %*
% Wp
                        }

                         W[p,k] <- (X[,p] %*% Y[,k] - sums) / su
m((X[,p])^2)
                         W[p,k]
                }


                fval <- c()  ##vector contains function values f
or pth row
                 for(p in 1:P){

                        Wkari <- W
                        Wkari[,q] <- rep(0,P)
                        Wkari[p,q] <- Updater(W,p,q,X,Y)
                        fval[p] <- f(Wkari,X,Y)

                }

                list(fval,which.min(fval))

        }
```

```
                        ##initialize random start
                        W <- matrix(rnorm(P*Q),P,Q) #initial value of W



                        ##Start Iteration (max 100 runs)
                        history <- c() #vector of function value
                        history[1] <- f(W,X,Y)


                        ##iteration start

                        for(ite in 1:100){

                                for(q in 1:Q){
                                        Wnew <- W
                                        Wnew[,q] <- rep(0,P)
                                        nonzerorow <- whichnonzero(W,q,X,Y)[[2]]
  ##theta-step ("position" of nonzero element)

                                        Wnew[nonzerorow,q] <- Updater(W,nonzeror
ow,q,X,Y)  ##W-step ("value" of nonzer element)

                                        W <- Wnew

                                }


                                history[ite+1] <- f(W,X,Y) ##evaluate the curren
t function value


                                ##stopping rule

                                if((history[ite] - history[ite+1]) < 0.00001){
                                        break
                                }



                        } ##iteration end



                        ##return
                        list(history,W)


                }
                ##SMR.mod.for.SPCA ends


                N <- nrow(Mat)
                P <- ncol(Mat)



                ##LS criterion to be minimized in SPCA
                f <- function(Fp,A){
                        sum((Mat - Fp%*%t(A))^2)
                }


                ###parameteres to be estimated
                ##Fp : Score matrix
                ##A : Loading matrix having perfect cluster structure
```

```
        ###initial values###
        res <- svd(Mat)
        A <- res$v[,1:r] %*% diag(res$d[1:r])/sqrt(N)



        ###iteration start###
        history <- c()
        for(ite in 1:100){
                res <- svd(t(A)%*%t(Mat))
                Fp <- res$v%*%t(res$u)*sqrt(N)
                X <- Fp
                Y <- Mat

                A <- t(SMR.mod.for.SPCA(X=Fp,Y=Mat)[[2]])

                history[ite] <- f(Fp,A)

                ##stopping rule
                if(ite > 1){
                        if((history[ite-1] - history[ite]) < 0.000001){b
reak}
                }

        }##iteration ends


        ##return
        list(F=Fp,A=A,history=history)

 }
 ##SPCAsub ends







##multiple-starts approach for finding two best/equivalent solutions

flist <- c()  ##vector of minimal function values
reslist <- list() ##list of results
lmcount <- 0 ##local-minimum counter
finalsol <- NULL ##final-solution

f <- function(Fp,A){
        sum((Mat - Fp%*%t(A))^2)
}




##the function for evaluating congruence of two matrices (Adachi, 2011)
##used for evaluating equivalence of two solutions

SS <- function(X,Xt){

        #Xt <- scale(as.matrix(Xt))
        #X <- scale(as.matrix(X))

        m <- nrow(X)
        p <- ncol(X)
        unitm <- c(rep(1,m))
        unitp <- c(rep(1,p))
        Ub <- unitm %*% t(unitm) %*% (X + Xt) %*% unitp %*% t(unitp) / (
```

```
2*m*p)

        1 - (0.5*sum((X-Xt)^2) / (sum((X - Ub)^2) + sum((Xt - Ub)^2)))
}



##step1 : intial M runs
M <- 3
for(m in 1:M){

        reslist[[m]] <- SPCAsub(Mat,r)
        flist[m] <- min(reslist[[m]]$history)

}


##step 2 : finding the best solutions within Mini solutions
tmin <- which.min(flist)
reshat <- reslist[[tmin]]
Fhat <- reslist[[tmin]]$F ##current "best" solution
Ahat <- reslist[[tmin]]$A ##current "best" solution


##step 3  finding two best solutions within M sets of solutions
for(m in 1:M){
        if(m != tmin){
                if(SS(Fhat, reslist[[m]]$F) > 0.999999){
                        finalsol <- reslist[[m]]
                        lmcount <- m-1
                        break
                }
        }
}


##repat until 2 equivalently best solutions are found
##Max 10 runs
Mmax <- 10

for(m in (M+1):Mmax){

        if(is.null(finalsol) == TRUE){

                reslist[[m]] <- SPCAsub(Mat,r)  ##Step 4
                restilda <- reslist[[m]]
                Ftilda <- restilda$F
                Atilda <- restilda$A


                if(round(f(Fhat,Ahat),12) >= round(f(Ftilda,Atilda),12))
{  ##Step 4
                        tmin <- m

                        if(SS(Fhat,Ftilda) > 0.00001){  ##Step 5
                                finalsol <- reslist[[tmin]]
                                lmcount <- m - 1
                        }

                }


                if(m == Mmax){   ##when maximal iterations reached
                        finalsol <- reslist[[tmin]]
                        lmcount <- Mmax
                }

        }
```

```
        }


        Fhat <- finalsol$F
        Ahat <- finalsol$A
        history <- finalsol$history


        list(Scores = Fhat,
             Loadings = Ahat,
             History = history,
             local.minimum = lmcount)

}
```