```
.libPaths("C:/R Code")
source("func_express.R") # symbolic regression function processing
source("LiftingSchemeJPEG.R") # JPEG package
source("PenalizedJPEG.R") # Penalized JPEG package

library(MASS)
library(stringr)
require(splines)
library(minpack.lm)
library(gmm)
library(Deriv)
library(np)

emptylist<-function(q,init=NA){
  mylist.names<-paste0("c",1:q,sep="")
  if (length(init)>=q) {
    mylist <- as.list(init[1:q])
  }
  else {
    mylist <- as.list(c(init,rep(0,q-length(init))))
  }
  names(mylist)<-mylist.names
  return(mylist)
}

#----------- GMM:Global.penalization choice -------------
Penalization.JPEG.GMMReg<-function(myformula, zformula, data,b0,
Level=NaN, UpToLevel=NaN, gamma=Inf, lambda = NaN) {

  #-------------- PREPARING THE INPUT SPACE ----------------------
  data$index = 1:nrow(data)
  data=data[with(data, order(t)),]
  out<-exp.f(myformula,data)
  Y = as.matrix(data[,names(out$y)])
  X = as.matrix(data[,names(out$x)])
  t = as.matrix(data[,names(out$m[[1]])])

  # Projection forumla [Z*inv(Z'Z)*t(Z)] * [Signal-yhat]
  # Z = [z, m(t)]; p = Signal-yhat
  Projection = paste0("p~",Reduce(paste,
deparse(zformula[[3]])),sep="",collapse="")
  print(sprintf("projection=%s",Projection))
  Projection = as.formula(Projection)

  z.out<-exp.f(zformula,data)
  Z = as.matrix(data[,names(z.out$x)])
  z.names=names(z.out$x)[!(names(z.out$x)%in%names(out$x))]
  sub.Z = as.matrix(data[,z.names])

  #-------------- INITIALIZING THE PARAMETERS ----------------------
  alpha = 0.001
  n=nrow(data)

  if (is.nan(Level)) {
```

```
    Level = ceiling(log(n/log(2)))
  }

  if (is.nan(UpToLevel)){
    UpToLevel = min(Level-1, 7+(n>=2000))
  }

  b=b0

  X.Grid = t
  Signal = as.matrix(Y-X%*%as.matrix(b))

  temp = main.filter.X.Grid(y=Signal, X.Grid=X.Grid, Level)
  my.X.Grid = temp$X.Grid

  coeff=temp$y

  #-------------------------------------
  iter = 0
  gap = Inf
  n = length(Signal)
  new.alpha=NULL
  max.coeff=max(abs(coeff))
  min.coeff=min(abs(coeff))
  print(sprintf("max.coeff=%f, lambda/alpha=%f,
gamma*lambda=%f",max.coeff, lambda/alpha, gamma*lambda))

  #  print(coeff)

  min.gamma = min.coeff / lambda

  OK = TRUE
  gap = Inf
  while ((iter < 1000) & OK & (gap>0.0001)) {
    # print("inner loop")
    gap = Inf
    Xt.res = NULL
    old.Xt.res = NULL
    Linear.Xt.res = NULL
    old.Linear.Xt.res = NULL
    iter.inner=0

    #-------------------------------------------------------------------
-------------------------
    # yhat is the prediction obtained as the inverse-transform of the
coefficient vector (coeff)
    # by using the irregular grid (my.X.Grid)
    yhat = inv.main.filter.X.Grid(coeff, my.X.Grid, Level)$y
    #-------------------------------------------------------------------
-------------------------
    residual = Signal - yhat

    Z.residual=sub.Z%*%inv(t(sub.Z)%*%sub.Z)%*%t(sub.Z)%*%residual
```

```
    #------------------Equation (3.10) ---------------------------------
----------------------------------------
    # Let Psi_I and t(Psi_I) be the JPEG-inverse trasform operator and
its transpose, respectively.
    # By using the transposed-inverse-transform (inv.t), we calculate
Xt.res=t(Psi_I)*residual
    Xt.res = inv.t(residual, X.Grid=X.Grid, Level = Level)
    #------------------------------------------------------------------
------------------------

    Linear.Xt.res = t(X)%*%Z.residual

    GMMObjective<-function(residual){
      g1 = 1/n*as.matrix(t(Z)%*%residual)
      return(n*t(g1)%*%pinv(1/n*t(Z)%*%Z)%*%g1)
    }

    # ------------ EVALUATION OF THE PENALIZED gmm OBJECTIVE FUNCTION ---
-----------------
    object.fun = 1/(2*n)*GMMObjective(residual) +
sum(MCP.penalty(abs(coeff),lambda, alpha, gamma))

    print(sprintf("obj=%f,gmmobj=%f",object.fun,GMMObjective(residual)))

    old.object.fun = object.fun
    old.coeff = coeff
    old.b = b
    old.Signal = Signal
    old.Xt.res = Xt.res
    old.Linear.Xt.res = Linear.Xt.res
    old.yhat = yhat

    inner.iter = 0
    if (iter==0){
      alpha = alpha
    } else{
      alpha = sqrt(new.alpha)
    }

    alpha = max(alpha,1/gamma+0.000001)

    repeat {
      Update = coeff+rep(1/(alpha*n),length(Xt.res))*Xt.res
      Update.b = b+rep(1/(alpha*n),length(Linear.Xt.res))*Linear.Xt.res

      # -------------- BEGIN OF JPEG coefficients update (MCP
thresholding)
      coeff = S.alpha.Threshold(Update, lambda, alpha, gamma)
      # -------------- END OF JPEG coefficients update ----------------
--

      b = Update.b
      yhat = inv.main.filter.X.Grid(coeff, my.X.Grid, Level)$y
```

```
      b=inv(t(Z)%*%X)%*%t(Z)%*%as.matrix(Y-yhat)

      Signal = as.matrix(Y-X%*%as.matrix(b))

      residual = Signal - yhat

      object.fun = 1/(2*n)*GMMObjective(residual) +
sum(MCP.penalty(abs(coeff),lambda, alpha, gamma))

      d=c(coeff-old.coeff, b-old.b)
      gap = max(abs(d))

      iter.inner = iter.inner + 1

      OK = (object.fun < old.object.fun)
      if ((inner.iter > 100)| OK) {
        Z.residual=sub.Z%*%inv(t(sub.Z)%*%sub.Z)%*%t(sub.Z)%*%residual
        Xt.res = inv.t(residual, X.Grid=X.Grid, Level = Level)
        Linear.Xt.res = t(X)%*%Z.residual
        break
      } else {
        inner.iter = inner.iter + 1
        alpha = alpha * 1.2;
        coeff = old.coeff
        object.fun = old.object.fun
        Signal = old.Signal
        b = old.b
        Xt.res = old.Xt.res
        Linear.Xt.res = old.Linear.Xt.res
        yhat = old.yhat
      }
    }

    if (OK){
      g.a=(-1/n*Xt.res)-(-1/n*old.Xt.res)
      g.b=(-1/n*Linear.Xt.res)-(-1/n*old.Linear.Xt.res)
      g=c(g.a, g.b)
      d=c(coeff-old.coeff, b-old.b)

new.alpha=(t(as.matrix(g))%*%as.matrix(d))/(t(as.matrix(d))%*%as.matrix(d
))
    } else {
      new.alpha = NULL
    }

    iter = iter + 1
  } # ouer while: alpha

  print(sprintf("min.gamma=%f, min.coeff=%f", min.gamma, min.coeff))
  return(list(b=old.b, coeff=old.coeff, X.Grid=my.X.Grid,
obj.fun=old.object.fun, yhat=old.yhat))
}
```

```
#----------------------------  JPEG IV EXAMPLE CODE: BEGIN SIMULATION ---
-----------------
  N=5000 # number of observations

# Generate a trivariate vector of correlated disturbances
  v = mvrnorm(N,c(0,0,0),Sigma=matrix(c(1,0.5,-0.25,0.5,1,0.4, -0.25,
0.4, 1),3,3), empirical = TRUE)
  eps = rnorm(N,0,1)
  ro=0.8; # the correlation of the endogenous covariate with the random
disturbance (in the substantive equation)
  # Constructing the covariate t for the bias term function:
  t=(1*v[,1]-3*v[,3])/4

  # generate an instrumental variable covariate correlated with the bias
term
  ro.z=-0.25
  z = ro.z*t + sqrt(1-ro.z^2)*rnorm(N,0,1)

  # introducing a correlation between the endogenous covariate x2 (which
is v[,2]), Z and epsilon
  v[,2]=ro*eps+sqrt(1-ro^2)*v[,2]#*rnorm(N,0,1)

  # generating a correlation between x2 and t
  v[,2] = 0.5*v[,2] + 0.5*z+tanh(t)

  # print all correlations:
  cor(v[,2],z)
  cor(v[,2],eps)
  cor(v[,2],t)
  cor(z,eps)

  data = data.frame(x1= v[,1], x2=v[,2], x3=v[,3],z=z)
  #t=(1*v[,1]-3*v[,3])/4
  true_b0 = c(1, 1.25)

  # the true (bias-free) outcome variable
  t0=v[,1]*true_b0(1)+v[,2]*true_b0(2)

  # the observed (bias-dependent) outcome variable y
  data$y= t0+ 0.8*exp(t)+eps
  mydata=data; mydata$t = t

  result1 <- tryCatch({ # Store data
    # begin #
    model<-Penalization.JPEG.GMMReg(y~x1+x2+m(t), y~x1+z+m(t), b0=c(12,-
3),data=mydata,lambda=0.001,gamma=Inf)
    model$b
    #i=1
      TRUE
  }, error = function(err) {

    print(paste("Model 1: MY_ERROR:  ",err))
    return(FALSE)
  }, finally = {
```

```
  print("Done")
}) # END tryCatch
```