USAGE

InterQR.fn(DAT, tau, method, tstar)

ARGUMENTS

DAT	Matrix of data with (P+2) columns and n rows, where p is the number of predictors, and n is the number of observations. The first column contains all ones corresponding to the intercept, and the last column is the response vector.
tau	Vector of quantiles of interest.
method	"FAL" or "FAS", denoting the Fused Adaptive Lasso and Fused Adaptive Sup-norm methods considered in the paper.
tstar	Tuning parameter controlling the degree of penalization.

VALUE

The estimated coefficient matrix with (p+1) rows and K columns, where K is the number of quantile levels. Each row contains the estimated quantile coefficients for one predictor at K quantile levels.

Tun.Sel(DAT, tstar, coef.matrix, dec=4, tau)

ARGUMENTS

coef.matrix Matrix of estimated quantile coefficients, which is the return value from InterQR.fn.

dec The number of decimal places to round the quantile coefficients when counting the degrees of freedom.

VALUE

AIC (or BIC) value corresponding to one fixed tuning parameter.

Examples

```
#generate data matrix
dat1<- function(n,p,gamma,beta)
{
 X<- cbind(1,matrix(runif(p*n,0,1),byrow=F,nrow=n))  #X includes intercept
 y<- X%*%beta + (1+gamma*X[,p+1])*rnorm(n,0,1)  #last predictor varies
 return(cbind(X,y))  #first 1:(p+1)columns are X, column (p+2) is Y
}


library(quantreg)
n<- 200
p<- 1   #univariate case
# when gamma=0, slope coefficients are constant; gamma=2, slope coefficients vary across quantiles
gamma<- 2
tau<- seq(from=0.1,to=0.9,by=0.1)
beta<- c(1,rep(3,p))   #true intercept and slope coefficients
tgrid<- 100   #grid search points

set.seed(1220901)
DAT<- dat1(n=n,p=p,gamma=gamma,beta=beta) #X:1+p columns, Y:p+2 column

#use FAL method
coef.FAL<- InterQR.fn(DAT, tau, method="FAL", tstar=0.5)
#use FAS method
coef.FAS<- InterQR.fn(DAT, tau, method="FAS", tstar=0.5)
```

```
# Simulations of Example 3 from paper, show FAL method only here, FAS is very similar

# Note: in this example, coefficients for the first predictor always remain constant across quantiles.
#Coefficients for the second predictor may vary, or stay constant across quantiles, depending on
#different choices of gamma.


dat1<- function(n,p,gamma,beta)

{

 X<- cbind(1,matrix(runif(p*n,0,1),byrow=F,nrow=n))  #X includes intercept

 y<- X%*%beta + (1+gamma*X[,p+1])*rnorm(n,0,1)

return(cbind(X,y))            #first 1:(p+1)columns are X, column (p+2) is Y

}


library(quantreg)

n<- 200

p<- 2        #Bivariate case

#gamma=2: coefficients for the first predictor are constant across quantiles, but vary

#across quantiles for the second predictor

 #gamma=0: coefficients are constant for both predictors across quantiles

gamma<- 2

tau<- seq(from=0.1,to=0.9,by=0.1)

k<- length(tau)

beta<- c(1,rep(3,p))


nsim<- 500    #number of simulations

tgrid<- 100   #number of grid search points
```

```
COEF.RQ = COEF.FAL = array(0,dim=c((p+1),k,nsim))
DAT.save = array(0,dim=c(n,p+2,nsim))        #save the simulated data matrix

set.seed(1220901)
for ( i in 1:nsim )
{
  DAT<- dat1(n=n,p=p,gamma=gamma,beta=beta) #X:1+p columns, Y:p+2 column
  X.dat<- DAT[,1:(p+1)]   #design matrix including intercept 1
  y.dat<- DAT[,(p+2)]
  DAT.save[,,i]<- DAT

  #conventional RQ method
  X.rq<- X.dat[,-1]  #don't need intercept for rq regression!
  rq.coef<- coef(rq(y.dat~X.rq,tau=tau))
  COEF.RQ[,,i]<- rq.coef

  #FAL method
  ####tuning parameter
  tmax<- p*(k-1)

  #grid search for t from 0 to tmax
   t<- seq(0,1,length=tgrid)*tmax  #grid search for unadaptive lasso

  aic.FAL<- aic.FAS<- rep(0, tgrid)
  coef.mat<- array(0, dim=c(p+1, k, tgrid))
```

```
 #find t with the smallest AIC
 for ( q in 1:tgrid )
  {
    coef.mat[,,q]<- InterQR.fn(DAT, tau, method="FAL", tstar=t[q])   #return estimated coefficient matrix
    aic.FAL[q]<- Tun.Sel(DAT=DAT, tstar=t[q], coef.matrix=coef.mat[,,q], dec=4, tau=tau )  #return AIC
values
 }

 idx<- which.min(aic.FAL)  #find the index of tuning parameter that returns the smallest AIC value
 t.select<- t[idx]        #select the tuning parameter which returns the smallest AIC value

 COEF.FAL[,,i]<- coef.mat[,,idx]   #save the estimated coefficient matrix

 print(i)
}

#True slope and intercepts
if (p==1) {true.beta<- rbind(beta[1]+qnorm(tau),beta[2]+gamma*qnorm(tau))} else {
true.beta<- rbind(beta[1]+qnorm(tau),
matrix(rep(beta[2:p],each=k),byrow=T,ncol=k),beta[p+1]+gamma*qnorm(tau))}

# Get Integrated MSE
IMSE.FAL = IMSE.RQ = matrix(0,nsim,k)
for ( j in 1:nsim)
{
  X<- DAT.save[,1:(p+1),j]
```

```
  IMSE.FAL[j,]<- apply((X%*%true.beta-X%*%COEF.FAL[,,j])^2,2,mean)  ##why not y.true-?
  IMSE.RQ[j,]<- apply((X%*%true.beta-X%*%COEF.RQ[,,j])^2,2,mean)
}

sfnc.sim.IMSE.FAL<- apply(IMSE.FAL,2,mean)
sfnc.se.IMSE.FAL<- apply(IMSE.FAL,2,sd)/sqrt(nsim)
sfnc.sim.IMSE.RQ<- apply(IMSE.RQ,2,mean)
sfnc.se.IMSE.RQ<- apply(IMSE.RQ,2,sd)/sqrt(nsim)
```