R Packages GPQR and GPER for High Dimensional Quantile and Expectile regression

2022-08-12

Introduction

This vignette describes two R packages, GPQR and GPER, which implement a family of new high dimensional penalized regression methods under three asymmetric loss functions (Quantile regression, Expectile regression and coupled expectile regression). In this thesis, we have dealt with the situation where the groups among the variables are known a priori. So, we have implemented our proposed methods for three penalties described in Chapters 2 and 3: Group Lasso, Group Scad and group MCP. In the GPQR framework, we have combined the idea of the approximation of the quantile objective check function, which is not differentiable everywhere, by a modified check function which is differentiable at zero. Furthermore, we have implemented algorithms which combine the MM principle and the coordinate descent trick to update each group of coefficients simply and efficiently. For the GPER and COGPER, we have used the MM principle and the coordinate descent algorithm directly without approximation of the expectile loss function since the lather is differentiable everywhere.

Of note, both R packages have complete vignettes' documentations, which give more details about how each method can be fitted with some running examples. The vignettes are written in English to reach large scientific audience, and they are publicly available
via Github (https://github.com/ouhourane/These_vignette). In this chapter, we present a brief overview of these two R packages.

GPQR and GPER solve the following problem

$$\hat{\boldsymbol{\beta}} = argmin_{\boldsymbol{\beta}} \left(\frac{1}{n} \sum_{i=1}^{n} \rho_\tau(y_i - \boldsymbol{x}_i^\top \boldsymbol{\beta}) + \sum_{k=1}^{K} P_\lambda(\|\boldsymbol{\beta}_k\|_2) \right)$$

over a grid of values of λ covering its entire range, where $\rho_\tau(u) = |\tau - \infty_{(u<0)}||u|$ for the quanile regression (GPQR) and $\rho_\tau(u) = |\tau - \infty_{(u<0)}|u^2$ for the expectile regression (GPER).

GPER R package solves also the coupled expectile regression described in Section 3.3 in this thesis. which is defined by

$$(\hat{\boldsymbol{\beta}}, \hat{\boldsymbol{\phi}}) = argmin_{(\boldsymbol{\beta},\boldsymbol{\phi})} \frac{\nu}{2n} \sum_{i=1}^{n} \rho_{0.5}(y_i - \boldsymbol{x}_i^\top \boldsymbol{\beta}) + \frac{1}{n} \sum_{i=1}^{n} \rho_\tau(y_i - \boldsymbol{x}_i^\top \boldsymbol{\beta} - \boldsymbol{x}_i^\top \boldsymbol{\phi}) + P_\lambda(\boldsymbol{\beta}) + P_\lambda(\boldsymbol{\phi}),$$

where $\rho_\tau(u) := |\tau - \infty_{(u<0)}|u^2$. In these packages, we consider the group Lasso (GLasso), group MCP (GMCP) and group SCAD (GSCAD) penalties, which are defined in (2), (3) and (4) in our paper.

R is an interpreted programming language, i.e. the R code is not directly run by the machine. Then, R is know to be slower than another compiled language like Fortran, C, etc. Especially for iterative code, the loops are slower in R than in compiled languages . A way to get all the speed advantages of Fortran language with the advantages of R is to code the inner loops in Fortran and call them from R. For speeding up our algorithms in both packages, the core code of GPER and GPQR are written as Fortran subroutines, which

makes a substantial saving of the running time. Several auxiliary functions in both packages (cv, predict, print, coef, etc.) are based on functions taken from the {gglasso} and {sales} packages.

The GPQR and GPER approaches use cyclical block coordinate descent algorithms, which iteratively update a block of variables, with others fixed. Our packages can calculate the path solution very fast, because they make use of some techniques such as the strong rule for efficient update of the active set and warm starts trick, more details are given in Section 2.3.4.

Installation

The two packages GPQR and GPER are not published yet on the Comprehensive R Archive Network (CRAN), however they are available via GitHub as many other R packages. To install GPQR and GPER from GitHub, we run the following code lines in R console:

```
library(devtools)
devtools::install_github("https://github.com/ouhourane/GPQR.git")
devtools::install_github("https://github.com/ouhourane/GPER.git")
```

In this vignette, we demonstrate how to use GPQR(.) and GPER(.), the main functions in the GPQR and GPER packages to fit the regularization path of quantile/expectile regression with grouped penalties. Other functions such as, predict(), coef(), cv.predict, cv.coef, etc., are derived from the {gglasso} and {sales} packages with some modifications.

Example 1 of our paper

In this example, our goal is to illustrate to end-users how the proposed can be run in R.

For illustration the toy data from Scenario 1 of Chapiter 2. More precisely, we set the sample size to $n = 100$ observations and $p = 20$ predictors. The predictors $X_j, j = 1 \dots 20$, were generated as follows:

- We generated $Z_j, j = 1, \dots, 11$, following the standard normal distribution;
- We set $X_j = Z_1 + \epsilon_j^x, j = 1, \dots, 4, \epsilon_j^x \sim N(0, 0.1)$;
- $X_j = Z_2 + \epsilon_j^x, j = 5, \dots, 8, \epsilon_j^x \sim N(0, 0.1)$;
- $X_j = Z_3 + \epsilon_j^x, j = 9, \dots, 12, \epsilon_j^x \sim N(0, 0.1)$;
- $X_j = Z_{j-9}, j = 13, \dots, 20$.

The following code describes how to generate the matrix of predictors $\boldsymbol{X}$ (of dimension nxp) from the multivariate normal distribution matrix $\mathbf{Z}$ (of dimension nxK) with the mean vector of the variables is $\mu_z = 0_{11}$ and the covariance matrix of the variables is $\boldsymbol{\Sigma}$, with $\Sigma_{jk} = 0.5^{|j-k|}$.

```
library("MASS")
n = 100
K = 11
p = 20
MuVec<-rep(0,K)
v<-rep(1,K); SigmaMat<-diag(v)

for(j in 1:K)
    for(k in 1:K)
        SigmaMat[j,k] <- 0.5^abs(j-k)
```

```
  Z <- mvrnorm(n,MuVec,SigmaMat,tol = 1e-6, empirical = FALSE)

  X = NULL
  for (h in 1:3)
      for (k in 1:4)
          X=cbind(X,Z[,h]+rnorm(n,0,0.1))

  X = cbind(X,Z[,4:K])
```

Thus, we set the predictors' effects to be

$$\boldsymbol{\beta} = (\underbrace{3,3,3,3}_{G_1},\underbrace{2,2,2,2}_{G_2},\underbrace{-1,-1,-1,-1}_{G_3},\underbrace{0,\ldots,0}_{G_4-G_{11}})^\top$$

In total we have 11 groups: $G_1, G_2, \ldots, G_{11}$

```
# A vector of consecutive integers describing the
# grouping of the coefficients
group=c(1,1,1,1,2,2,2,2,3,3,3,3,4:K)
```

The response Y is generated from the following location-scale linear regression model

$$Y = \sum_{j=1}^{20} \beta_j X_j + \Phi(X_{20})\epsilon, \quad \epsilon \sim N(0,3),$$

where $\Phi(.)$ is the cumulative distribution function of the standard normal distribution.

```
beta = c(rep(3,4),rep(2,4),rep(-1,4),rep(0,p-12))
Y<-X%*%beta+pnorm(X[,p])*rnorm(n,0,3)
```

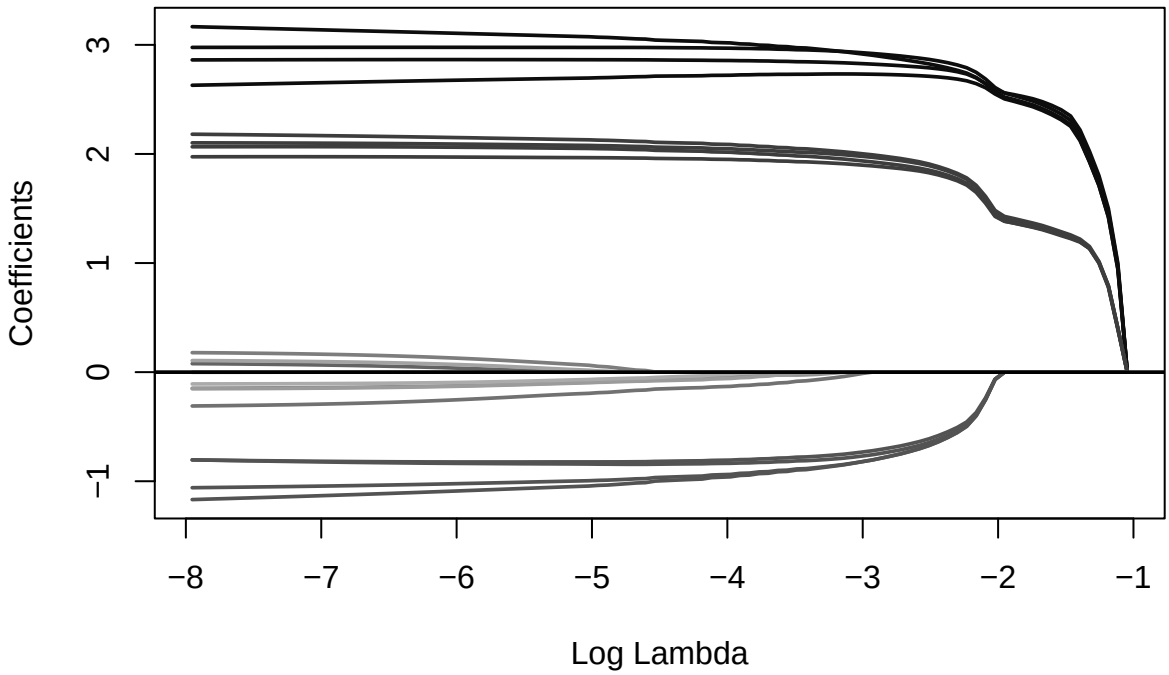
Introduction to the GPQR package

We fit the model using the most basic call of GPQR() function with many optional input arguments. In the folowing code line, we run GPQR() for group Lasso penaly (method = "GLasso"), the parameter $\tau = 0.5$ (taux = 0.5), and the pseudo quantile loss function $\Psi^{(1)}_{\tau,\delta}(.)$ (check = "f1") given in (5).

```
library("GPQR")
fit <- GPQR(x=X,y=Y,group=group, method="GLasso", check="f1", taux=0.5)
```

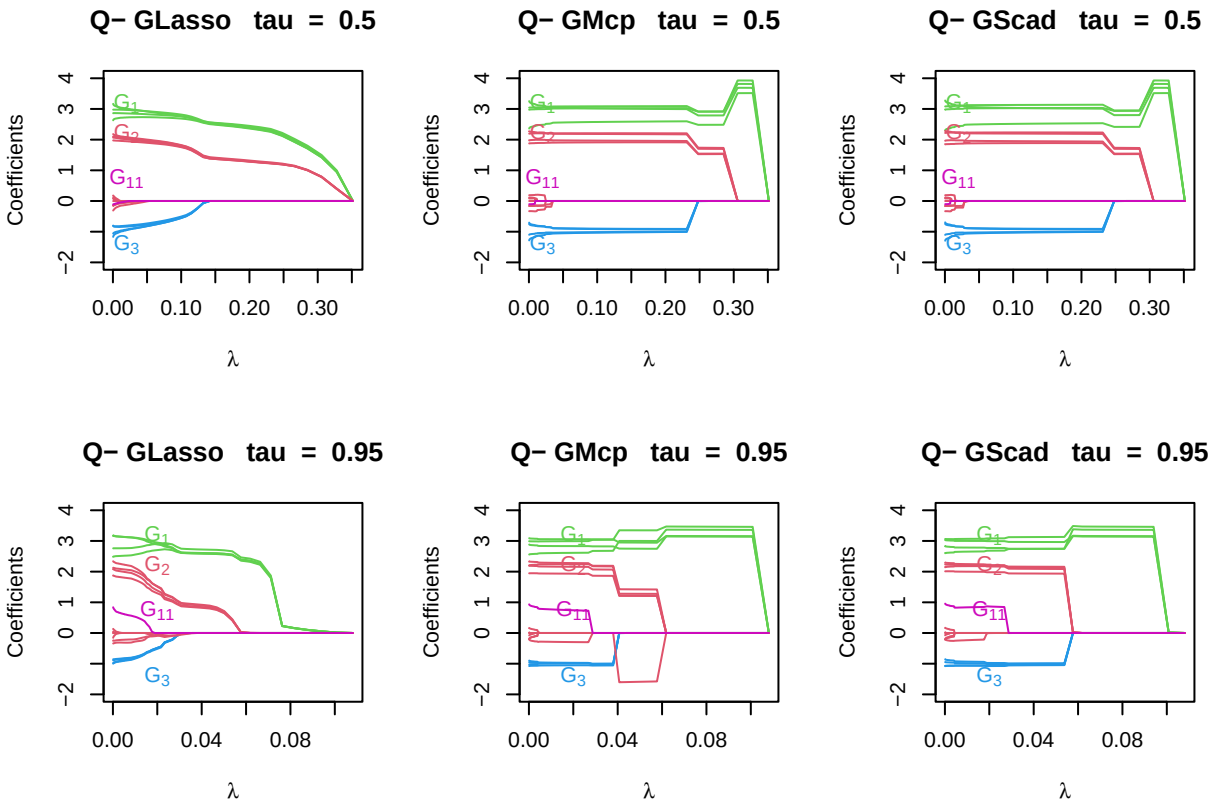
We use the function {plot} to produce a coefficient profile plot of the coefficient paths, for the fitted GPQR object, fit.

```
plot(fit)
```



In the Figure above, we reproduce a part of Figure 2 shown in our paper, which illustrates how the GPQR approache can be useful for detecting heteroskedastic groups of variables. We use a custom function {GPQR_illustration} to highlight paths of each group with a different colour for better visual presentation. This lead the following figure. The code of this function is given in Appendix A

```
par(mfrow = c(2, 3))
GPQR_illustration("GLasso", 0.5)
GPQR_illustration("GMcp", 0.5)
GPQR_illustration("GScad", 0.5)
GPQR_illustration("GLasso", 0.95)
GPQR_illustration("GMcp", 0.95)
GPQR_illustration("GScad", 0.95)
```



Introduction to the GPER package: GPER approach

We fit the GPER model with the penalty GMcp using the most call of GPER() function with $\tau = 0.85$.This is given by

```
library("GPER")
```

```
## Le chargement a nécessité le package : Matrix
```

```
fit <- gper(x=X,y=Y,group=group, method="GMcp", tau = 0.85)
```

The object {fit} is a list containing all the relevant information of the fitted model. Users can explore this object by directly looking at its elements, which are summarized as a list. Various functions are provided to extract information from the GPER object such as plot, print, coef and predict functions, which enable us to execute several tasks easily.

We can obtain the actual coefficients a specific value (or several values) of λ within the range of the sequence $(\lambda_{\min}, \lambda_{\max})$:

```
coef(fit, s = 1)
```

```
## 21 x 1 Matrix of class "dgeMatrix"
##                      1
## (Intercept)  1.4794644
## V1           2.0938797
## V2           3.7848164
## V3           7.0350721
## V4          -1.5177053
## V5           3.3744109
```

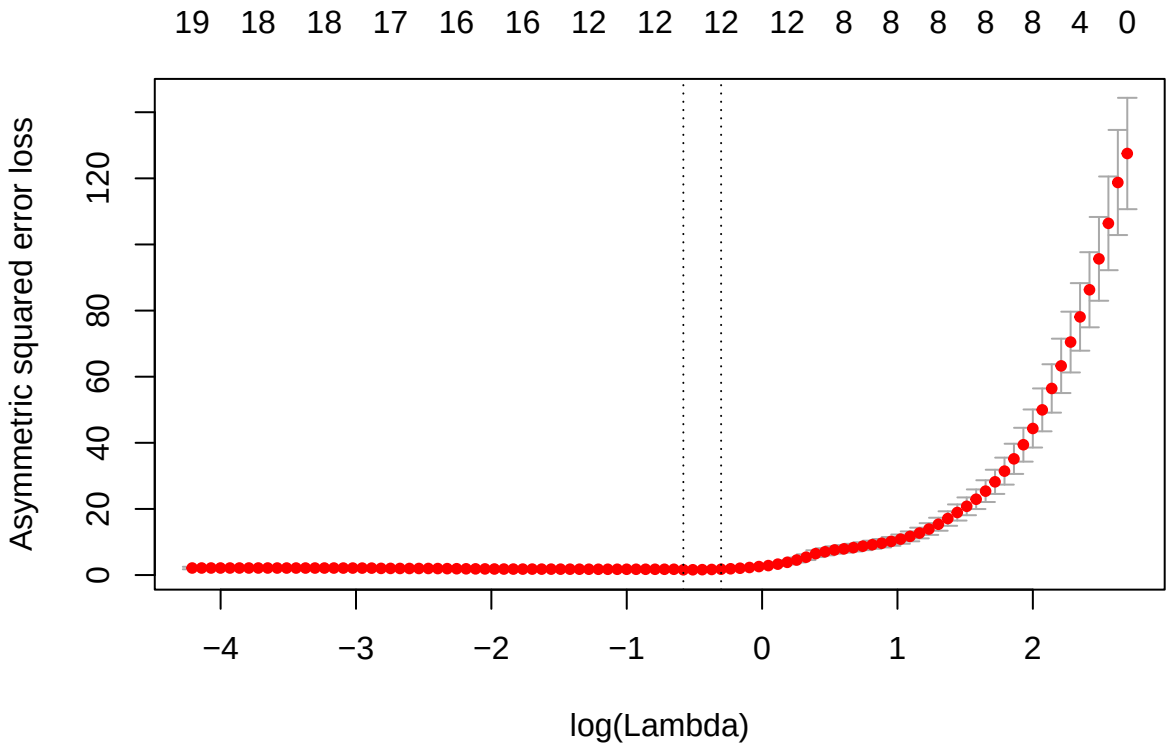
```
## V6           3.5354607
## V7          -1.0957749
## V8           1.7606026
## V9          -0.6370541
## V10         -0.6552879
## V11         -0.6457848
## V12         -0.6762927
## V13          0.0000000
## V14          0.0000000
## V15          0.0000000
## V16          0.0000000
## V17          0.0000000
## V18          0.0000000
## V19          0.0000000
## V20          0.0000000
```

The function {cv.gper} can be used to compute k-fold cross-validation for the GPER model. This function returns a list of outputs that contains a cv.gper object.

```
cvfit <- cv.gper(x=X, y=Y, group=group, method="GScad", tau=0.5)
```

We can plot the cross-validated error by plotting this object as following:

```
plot(cvfit)
```



The optimal value of λ can be obtained by the two vertical dotted lines corresponding to {lambda.min} and {lambda.1se}, where {lambda.min} is the values of λ corresponding to the minimum of cross validation error, and {lambda.1se} largest value of λ such that error is within one standard error of the cross-validated errors for {lambda.min}. for instance, the values of the coefficients, $\hat{\beta}$, corresponding to lambda.1se is given by

```
coef(cvfit, s = "lambda.1se")
```

```
## 21 x 1 Matrix of class "dgeMatrix"
##                      1
## (Intercept)  0.1294323
## V1           2.3709347
## V2           2.2377443
## V3           5.6697511
## V4           1.2791922
## V5           2.1031916
## V6           2.5486275
## V7           0.8019137
## V8           2.5900584
## V9          -0.7718017
## V10         -0.8328294
## V11         -0.7806776
## V12         -0.8213181
## V13          0.0000000
## V14          0.0000000
## V15          0.0000000
## V16          0.0000000
## V17          0.0000000
## V18          0.0000000
## V19          0.0000000
## V20          0.0000000
```

Introduction to the GPER package: COGPER approaches

{cv.cogper} is the main function to do cross-validation for the COGPER model. We run this function with $\tau = 0.9$ and the penalty GLasso.

```
cvfit <- cv.cogper(x=X,y=Y,group=group, method="GLasso",tau=0.9)
```

The returned output is an object of class cogper that contains all relevant information of the fitted model for further use. The function plot, coef and predict can be applied to the fitted object to get easily more detailed results in a similar way as it is illustrated earlier.

We can make predictions by applying the predict function. For this, users need to input a design matrix and the value(s) of λ at which predictions are need to be made.

```
predict(cvfit, newx = X[1:3,], s = "lambda.min")
```

```
##              1
## [1,]  15.162606
## [2,] -10.117208
## [3,]   9.725722
```

For instance, the value 13.404298 is the prediction, $\boldsymbol{x}_1^\top \hat{\boldsymbol{\beta}}$, for the first observation from $\boldsymbol{X}$

The following function gives coefficients from a cross-validated COGPER model, using the fitted cv.cogper object, and the optimal value chosen for λ.

```
coef(cvfit, s = "lambda.min")
```

```
## $beta
## 21 x 1 Matrix of class "dgeMatrix"
##                       1
```

```
## (Intercept)  0.01183706
## V1           3.31180381
## V2           2.50733489
## V3           3.65469516
## V4           2.07498498
## V5           1.99845139
## V6           2.19570556
## V7           1.67838861
## V8           2.39000895
## V9          -0.82152466
## V10         -1.15844259
## V11         -0.72713663
## V12         -1.04536019
## V13          0.02877057
## V14         -0.23215591
## V15          0.00000000
## V16         -0.01840737
## V17         -0.06300334
## V18         -0.11087659
## V19          0.00000000
## V20          0.00000000
##
## $phi
## 21 x 1 sparse Matrix of class "dgCMatrix"
##                      1
## (Intercept) 1.3220145
## V1          .
## V2          .
## V3          .
## V4          .
## V5          .
## V6          .
## V7          .
## V8          .
## V9          .
## V10         .
## V11         .
## V12         .
## V13         .
## V14         .
## V15         .
## V16         .
## V17         .
## V18         .
## V19         .
## V20         0.1003674
```

Appendix A

The following function offers a customized version of plot function for the fitted GPQR object.

```
GPQR_illustration <-function(penalty, taux){
fit <- GPQR(x=X, y=Y, group=group, method=penalty, check="f1", taux=taux)
```

```
main_lab = paste("Q-",penalty," ",expression(tau), " = ", taux)
matplot(fit$lambda, t(fit$beta[1:4,]), type = "l", col = 3,lty = 1, ylim=c(-2,4), lwd=1,
        ylab="Coefficients", main=main_lab, xlab = expression(lambda))
matlines(fit$lambda,t(fit$beta[5:8,]), type="l", col=2, lty=1,lwd=1)
matlines(fit$lambda,t(fit$beta[9:12,]), type="l", col=4, lty=1,lwd=1)
matlines(fit$lambda,t(fit$beta[13:19,]), type="l", col=2, lty=1,lwd=1)
matlines(fit$lambda,fit$beta[20,], type="l", col=6, lty=1,lwd=1)
text(0.02,0.75, expression("G"[11]), col=6)
text(0.02,3.2, expression("G"[1]), col=3)
text(0.02,2.2, expression("G"[2]), col=2)
text(0.02,-1.4, expression("G"[3]), col=4)
}
```