A brief summary of the QPADM-slack algorithm

Nan Lin
Department of Mathematics and Statistics
Washington University in St. Louis
nlin@wustl.edu
& $YeFan$
Department of Statistics
Capital University of Economics and Business
fanye0319@outlook.com

The QPADM-slack algorithm finds the coefficient estimates of penalized quantile regression models in distributed big data. We implemented this algorithm in Rcpp. The main function is named `paraQPADMslackcpp()`. The estimates are a p-dimensional vector, called `beta` in the `paraQPADMslackcpp()` function.

1 Input Data

The input data are assumed from n observations consisting of two components, a response variable and p covariates. In our code, the response variable is a length-n vector `y`, and the covariates a $n \times p$ matrix `x`. Both are input to the function `paraQPADMslackcpp()`. Inside this function, we partition the data into K equal-size partitions, and the kth ($k = 1, 2, \cdots, K$) partition are denoted as `yk` and `xk` as in the following.

```
nk = n/K
yk = y.subvec(k*nk, k*nk+nk-1)
xk = x.rows(k*nk, k*nk+nk-1)
```

The Rcpp functions `.subvec()` and `.rows()` extract the sub-vector of `y` from the `(k*nk+1)`th to the `(k*nk+nk)`th element, and the sub-matrix of `x` from the `(k*nk+1)`th to the `(k*nk+nk)`th row. Note that, in Rcpp, the index starts from 0.

2 What does each iteration do?

QPADM-slack is an iterative algorithm. In the $(s+1)$th iteration, using the global estimate from the previous sth iteration, it first updates some local quantities $\boldsymbol{\xi}_k^{s+1}$, $\boldsymbol{\eta}_k^{s+1}$, $\boldsymbol{\beta}_k^{s+1}$, $\boldsymbol{u}_k^{s+1}$ and $\boldsymbol{v}_k^{s+1}$ on every partition $k = 1, 2, \cdots, K$. The iteration index s is called `iteration` in the `paraQPADMslackcpp()` function. **This is the part we would like to make the computation in parallel.** Next, a global estimate $\boldsymbol{\beta}^{s+1}$ is updated based on the local quantities. So, in total, each iteration involves updating $(1 + 5K)$ quantities. Below we give the corresponding names of these variables used in the `paraQPADMslackcpp()` function.

- $\boldsymbol{\beta}^{s+1}$ (length-p vector): `beta`;
- $\boldsymbol{\xi}_k^{s+1}$ (length-nk vector): The long vector `xi` (length-n) includes all of $\boldsymbol{\xi}_1^{s+1}, \boldsymbol{\xi}_2^{s+1}, \cdots, \boldsymbol{\xi}_K^{s+1}$. Each local quantity $\boldsymbol{\xi}_k^{s+1}$ is the sub-vector `xi.subvec(k*nk, k*nk+nk-1)`;
- $\boldsymbol{\eta}_k^{s+1}$ (length-nk vector): The long vector `eta` (length-n) includes all of $\boldsymbol{\eta}_1^{s+1}, \boldsymbol{\eta}_2^{s+1}, \cdots, \boldsymbol{\eta}_K^{s+1}$. Each local quantity $\boldsymbol{\eta}_k^{s+1}$ is the sub-vector `eta.subvec(k*nk, k*nk+nk-1)`. We also use `etaini` to denote the value of `eta` from the previous iteration, i.e., $\boldsymbol{\eta}_1^{s}, \boldsymbol{\eta}_2^{s}, \cdots, \boldsymbol{\eta}_K^{s}$.
- $\boldsymbol{\beta}_k^{s+1}$ (length-p vector): We use a $(p \times K)$-dimensional matrix `z` to store all of the local parameters $\boldsymbol{\beta}_k^{s+1}$ with its kth column, i.e., `z.col(k)`, storing $\boldsymbol{\beta}_k^{s+1}$. Similarly, `zini` is the value of `z` from the previous iteration. Besides, `zmean` (length-p vector) is the average of `zini` over all partitions.
- $\boldsymbol{u}_k^{s+1}$ (length-p vector): We use a $(p \times K)$-dimensional matrix `u` to store all of them with its kth column, i.e., `u.col(k)`, storing $\boldsymbol{u}_k^{s+1}$. Similarly, `uini` is the value of `u` from the previous iteration. Besides, `umean` (length-p vector) is the average of `uini` over all partitions.

- $\boldsymbol{v}_k^{s+1}$ (length-nk vector): We use a vector `v` (length-n) to store $\boldsymbol{v}_1^{s+1}, \boldsymbol{v}_2^{s+1}, \cdots, \boldsymbol{v}_K^{s+1}$, and its kth sub-vector `v.subvec(k*nk, k*nk+nk-1)` gives $\boldsymbol{v}_k^{s+1}$. `vini` is the value of `v` from the previous iteration. We also use `vinik` to represent `vini.subvec(k*nk, k*nk+nk-1)`.

2.1 Update the global quantity $\boldsymbol{\beta}^{s+1}$

Update of the global quantity $\boldsymbol{\beta}^{s+1}$ is done elementwisely. When updating the jth element β_j^{s+1}, we evaluate an objective function

$$\frac{1}{2}\left(x-\psi_j^s\right)^2+\frac{1}{\rho K}p_\lambda\left(x\right),$$

at a few possible values, among which the one minimizes the objective function is then β_j^{s+1}. Here, $\psi_j^s = z_j^s + u_j^s/\rho$, where z_j^s and u_j^s respectively are the jth element of `zmean` and `umean`. The specific implementation depends on the choice of the penalty function $p_\lambda\left(x\right)$ specified in the `gpenalty()` function. Currently, we only support two penalty functions, SCAD and MCP.

- Under the SCAD penalty, the objective function is evaluated at four values, 0, x_1, x_2, x_3, where

$$\begin{aligned}
x_1 &= \text{sign}\left(\psi_j^k\right)\min\left(\lambda,\ \max\left(0, |\psi_j^k| - \lambda/(\rho K)\right)\right),\\
x_2 &= \text{sign}\left(\psi_j^k\right)\min\left(a\lambda,\ \max\left(\lambda, \frac{\rho K|\psi_j^k|\left(a-1\right)-a\lambda}{\rho K\left(a-1\right)-1}\right)\right),\\
x_3 &= \text{sign}\left(\psi_j^k\right)\max\left(a\lambda,\ |\psi_j^k|\right).
\end{aligned}$$

  The corresponding codes for this case are as follows.

```
line 184: if(penalty == "scad"){
            if(intercept){
              beta(0) = zmean(0)+umean(0)/pho;
              for(int j = 1; j < p; j++){
                ...
                ...
              }
            }
            else{
```

```
              for(int j = 0; j < p; j++){
                ...
                ...
              }
            }
line 211: }
```

Note: if the model contains an intercept, i.e., the input logic variable `intercept == TRUE`, we should update the first element of $\boldsymbol{\beta}^{s+1}$ by `beta(0) = zmean(0)+umean(0)/pho`.

- Under the MCP penalty, the objective function is evaluated at three values, 0, x_1, x_2, where

$$\begin{aligned} x_1 &= \text{sign}\left(\psi_j^k\right) \min\left(a\lambda,\ \max\left(0, \frac{a\left(\rho K|\psi_j^k| - \lambda\right)}{a\rho K - 1}\right)\right), \\ x_2 &= \text{sign}\left(\psi_j^k\right) \max\left(a\lambda,\ |\psi_j^k|\right). \end{aligned}$$

The codes for implementing this case are

```
line 212: else{
            if(intercept){
              beta(0) = zmean(0)+umean(0)/pho;
              for(int j = 1; j < p; j++){
                ...
                ...
               }
             }
            else{
              for(int j = 0; j < p; j++){
                ...
                ...
              }
            }
line 236: }
```

2.2 Update the local quantities

The update for the kth ($k = 1, 2, \cdots, K$) set of local quantities $\left(\boldsymbol{\xi}_k^{s+1}, \boldsymbol{\eta}_k^{s+1}, \boldsymbol{\beta}_k^{s+1}, \boldsymbol{u}_k^{s+1}, \boldsymbol{v}_k^{s+1}\right)$ only depends on the partition `xk` and `yk`. Thus, these K updates can be completed simultaneously in reality. However, in our code, they are implemented in a sequential way through a loop.

```
line 234: for(int k = 0; k < K; k++){
            arma::mat xk = x.rows(k*nk, k*nk+nk-1);
            arma::vec yk = y.subvec(k*nk, k*nk+nk-1);
            ...
            ...
line 272:  }
```

Therefore, we would like to make this part in parallel.

1) Among these updates, the updates for

$$\boldsymbol{\xi}_k^{s+1} = \max\left(\mathbf{0},\ \boldsymbol{y}_k - X_k\boldsymbol{\beta}_k^s + \boldsymbol{\eta}_k^s + \boldsymbol{v}_k^s/\rho - \tau\mathbf{1}/\rho\right)$$

and

$$\boldsymbol{\eta}_k^{s+1} = \max\left(\mathbf{0},\ -\boldsymbol{y}_k + X_k\boldsymbol{\beta}_k^s + \boldsymbol{\xi}_k^{s+1} - \boldsymbol{v}_k^s/\rho - (1-\tau)\,\mathbf{1}/\rho\right)$$

are implemented in a similar way. We first calculate the nk-dimensional vectors $\left(\boldsymbol{y}_k - X_k\boldsymbol{\beta}_k^s + \boldsymbol{\eta}_k^s + \boldsymbol{v}_k^s/\rho - \tau\mathbf{1}/\rho\right)$ and $\left(-\boldsymbol{y}_k + X_k\boldsymbol{\beta}_k^s + \boldsymbol{\xi}_k^{s+1} - \boldsymbol{v}_k^s/\rho - (1-\tau)\,\mathbf{1}/\rho\right)$, and then set their negative elements as 0. See the codes on lines 249-254.

2) For the update of $\boldsymbol{\beta}_k^{s+1}$, i.e.,

$$\boldsymbol{\beta}_k^{s+1} = \left(I_p + X_kX_k\right)^{-1}\left[\boldsymbol{\beta}^{s+1} - \boldsymbol{u}_k^s/\rho + X_k\left(\boldsymbol{y}_k - \boldsymbol{\xi}_k^{s+1} + \boldsymbol{\eta}_k^{s+1} + \boldsymbol{v}_k^s/\rho\right)\right],$$

we decomposed it into two steps.

- Matrix inversion (i.e., $\left(I_p + X_kX_k\right)^{-1}$):

```
line 156: arma::cube tmp = arma::zeros<arma::cube>(p,p,K);
          for(int k = 0; k < K; k++){
            arma::mat tmp2, xk = x.rows(k*nk, k*nk+nk-1);
            ...
            ...
            tmp.slice(k) = tmp2;
line 172: }
```

As this part in $\boldsymbol{\beta}_k^{s+1}$ needs not to be updated, we conduct this computation before the iterative process.

- Matrix product:

```
line 263: z.col(k) = tmp.slice(k)*(beta-uini.col(k)/pho+xk.t()*
(yk-xi.subvec(k*nk,k*nk+nk-1)+eta.subvec(k*nk,k*nk+nk-1)+vinik/pho))
```

3) The updates for $\boldsymbol{u}_k^{s+1}$ and $\boldsymbol{v}_k^{s+1}$ directly follow the rules:

$$\boldsymbol{u}_k^{s+1} = \boldsymbol{u}_k^s + \rho\left(\boldsymbol{\beta}_k^{s+1} - \boldsymbol{\beta}^{s+1}\right),$$

and

$$\boldsymbol{v}_k^{s+1} = \boldsymbol{v}_k^s + \rho\left(\boldsymbol{y}_k - X_k\boldsymbol{\beta}_k^{s+1} - \boldsymbol{\xi}_k^{s+1} + \boldsymbol{\eta}_k^{s+1}\right),$$

and the corresponding codes respectively are

```
line 265: u.col(k) = uini.col(k)+pho*(z.col(k)-beta)
```

and

```
line 268: v.subvec(k*nk,k*nk+nk-1) = vinik+pho* (yx.subvec(k*nk,k*nk+nk-1)
-xi.subvec(k*nk,k*nk+nk-1)+eta.subvec(k*nk,k*nk+nk-1))
```

Note: in the updates for $\boldsymbol{\xi}_k^{s+1}$, $\boldsymbol{\eta}_k^{s+1}$ and $\boldsymbol{v}_k^{s+1}$, the local error $\left(\boldsymbol{y}_k - X_k\boldsymbol{\beta}_k^{s+1}\right)$ or $(\boldsymbol{y}_k - X_k\boldsymbol{\beta}_k^s)$ is a common variable. To save computational cost, we use a long vector `yx` (length-n) to store all these K local errors, and its kth sub-vector `yx.subvec(k*nk, k*nk+nk-1)` is thus $\left(\boldsymbol{y}_k - X_k\boldsymbol{\beta}_k^{s+1}\right)$ or $(\boldsymbol{y}_k - X_k\boldsymbol{\beta}_k^s)$. See the code on line 267.