A brief introduction to the QPADM-slack algorithm

Nan Lin and Ye Fan

Washington University in St. Louis
Capital University of Economics and Business

Distributed big data

- The QPADM-slack algorithm is designed for solving estimation problems in distributed big data. In these estimation problems, the parameter vector to be estimated is $\boldsymbol{\beta} \in \mathbb{R}^p$, i.e., the variable `beta` in our `paraQPADMslackcpp()` function in the code.
- In distributed environments, the full data $(X, \boldsymbol{y})$ is split into K partitions, with each partition stored on a local machine. In our code, $X \in \mathbb{R}^{n \times p}$ and $\boldsymbol{y} \in \mathbb{R}^n$ are respectively denoted as `x` and `y`. They are input variables of `paraQPADMslackcpp()`. The kth $(k = 1, 2, \cdots, K)$ partition of them are respectively denoted as `xk` and `yk` with

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

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

QPADM-slack

QPADM-slack is an iterative algorithm, and in each iteration, it needs to update $(1+5K)$ variables, i.e., the central variable $\boldsymbol{\beta}^{s+1}$ and the local variables $\boldsymbol{\xi}_k^{s+1}, \boldsymbol{\eta}_k^{s+1}, \boldsymbol{\beta}_k^{s+1}, \boldsymbol{u}_k^{s+1}$ and $\boldsymbol{v}_k^{s+1}$ $(k = 1, 2, \cdots, K)$, where s is the number of iterations that the algorithm has already executed (denoted by `iteration` in the code). In the following, we give the corresponding forms of these variables used in our code.

- $\boldsymbol{\beta}^{s+1}$: `beta`;
- $\boldsymbol{\xi}_k^{s+1}$: in the code, we use `xi` to represent the long vector composed of $\boldsymbol{\xi}_1^{s+1}, \boldsymbol{\xi}_2^{s+1}, \cdots, \boldsymbol{\xi}_K^{s+1}$, and its kth subvector, i.e., `xi.subvec(k*nk,k*nk+nk-1)`, is then $\boldsymbol{\xi}_k^{s+1}$;
- $\boldsymbol{\eta}_k^{s+1}$: similarly, we use `eta` to represent the long vector composed of $\boldsymbol{\eta}_1^{s+1}, \boldsymbol{\eta}_2^{s+1}$, $\cdots, \boldsymbol{\eta}_K^{s+1}$, and its kth subvector `eta.subvec(k*nk,k*nk+nk-1)`, is then $\boldsymbol{\eta}_k^{s+1}$. Besides, the long vector composed of $\boldsymbol{\eta}_1^{s}, \boldsymbol{\eta}_2^{s}, \cdots, \boldsymbol{\eta}_K^{s}$ is denoted by `etaini`. That is, `etaini` is the value of `eta` in the previous iteration.

QPADM-slack

- $\boldsymbol{\beta}_k^{s+1}$: in the code, we use a matrix `z` to represent $\boldsymbol{\beta}_1^{s+1}, \boldsymbol{\beta}_2^{s+1}, \cdots, \boldsymbol{\beta}_K^{s+1}$, and the kth column of `z`, i.e., `z.col(k)` is $\boldsymbol{\beta}_k^{s+1}$. Correspondingly, `zini` is the value of `z` in the previous iteration. Besides, `zmean` is the mean of `zini` calculated by rows.
- $\boldsymbol{u}_k^{s+1}$: similarly, we use `u` to represent the matrix composed of $\boldsymbol{u}_1^{s+1}, \boldsymbol{u}_2^{s+1}, \cdots, \boldsymbol{u}_K^{s+1}$, and the kth column of `u`, i.e., `u.col(k)` is $\boldsymbol{u}_k^{s+1}$. Correspondingly, `uini` is the value of `u` in the previous iteration, and `umean` is the mean of `uini`.
- $\boldsymbol{v}_k^{s+1}$: we use `v` to represent the long vector composed of $\boldsymbol{v}_1^{s+1}, \boldsymbol{v}_2^{s+1}, \cdots, \boldsymbol{v}_K^{s+1}$, and its kth subvector `v.subvec(k*nk,k*nk+nk-1)` is $\boldsymbol{v}_k^{s+1}$. `vini` is then the value of `v` in the previous iteration. Besides, for simplicity, we use `vinik` to represent `vini.subvec(nk*k, nk*k+nk-1)` in the code.

QPADM-slack

Next, we give the update rules for these variables, and their specific implementation in the code.

$\boldsymbol{\beta}^{s+1}$ (a p-dimensional vector) is updated by element. Denote the jth element of $\boldsymbol{\beta}^{s+1}$ as β_j^{s+1}. In our algorithm, β_j^{s+1} is updated as follows.

Under SCAD penalty:

$$\beta_j^{s+1} = \underset{x \in \{x_1, x_2, x_3, 0\}}{\arg\min} \frac{1}{\rho K} p_\lambda^{\mathrm{SCAD}}(x) + \frac{1}{2}\left(x - \psi_j^s\right)^2,$$

where

$$p_\lambda^{\mathrm{SCAD}}(x) = \lambda|x|\mathbb{I}(0 \le |x| \le \lambda) + \frac{-x^2 + 2a\lambda|x| - \lambda^2}{2(a-1)}\mathbb{I}(\lambda < |x| \le a\lambda) + \frac{(a+1)\lambda^2}{2}\mathbb{I}(|x| > a\lambda)$$

is the SCAD penalty function (implemented by `gpenalty()` in our code), and $\psi_j^s = z_j^s + u_j^s/\rho$, where z_j^s and u_j^s respectively are the jth element of `zmean` and `umean`. More specifically, β_j^{s+1} is one of the four values x_1, x_2, x_3 and 0, such that the function $\frac{1}{\rho K} p_\lambda^{\mathrm{SCAD}}(x) + \frac{1}{2}\left(x - \psi_j^s\right)^2$ reaches its minimum. The definitions of x_1, x_2 and x_3 are given in the next page.

QPADM-slack

$$x_1 = \mathrm{sign}\left(\psi_j^k\right) \min\left(\lambda, \max\left(0, |\psi_j^k| - \lambda/(\rho K)\right)\right),$$

$$x_2 = \mathrm{sign}\left(\psi_j^k\right) \min\left(a\lambda, \max\left(\lambda, \frac{\rho K |\psi_j^k| (a-1) - a\lambda}{\rho K (a-1) - 1}\right)\right),$$

$$x_3 = \mathrm{sign}\left(\psi_j^k\right) \max\left(a\lambda, |\psi_j^k|\right).$$

QPADM-slack

Code:

```
zmean = mean(zini, 1);
umean = mean(uini, 1);
for(int j = 0; j < p; j++){
phi = zmean(j)+umean(j)/pho;
//calculate x1
xscad(0) = sign(phi)*min(lambda, max(0.0, abs(phi)-lambda/(pho*K)));
//calculate x2
xscad(1) = sign(phi)*min(a*lambda, max(lambda, (pho*K*(a-1)*abs(phi)-a*lambda)/(pho*K*(a-1)-1)));
//calculate x3
xscad(2) = sign(phi)*max(a*lambda, abs(phi));
arma::vec hscad(4, fill::zeros);
//calculate the function value under x1, x2, x3 and 0
for(int i = 0; i < 4; i++){
hscad(i) = 0.5*(xscad(i)-phi)*(xscad(i)-phi)+gpenalty(xscad(i), a, lambda, penalty)/(pho*K); }
beta(j) = xscad(hscad.indexmin());
}
```

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;
```

where `pho` is ρ and `phi` is ψ_j^s.

QPADM-slack

Under MCP penalty:

$$\beta_j^{s+1} = \underset{x\in\{x_1,x_2,0\}}{\arg\min} \frac{1}{\rho K} p_\lambda^{\mathrm{MCP}}(x) + \frac{1}{2}\left(x - \psi_j^k\right)^2,$$

where

$$p_\lambda^{\mathrm{MCP}}(x) = \left(\lambda|x| - \frac{x^2}{2a}\right)\mathbb{I}\left(0 \le |x| \le a\lambda\right) + \frac{a\lambda^2}{2}\mathbb{I}\left(|x| > a\lambda\right)$$

is the MCP penalty function (implemented by `gpenalty()` in our code), and the definition of ψ_j^s is the same as that under the SCAD penalty. Thus, β_j^{s+1} is one of the three values x_1, x_2 and 0, such that the function $\frac{1}{\rho K} p_\lambda^{\mathrm{MCP}}(x) + \frac{1}{2}\left(x - \psi_j^k\right)^2$ reaches its minimum. Here,

$$x_1 = \mathrm{sign}\left(\psi_j^k\right)\min\left(a\lambda, \max\left(0, \frac{a\left(\rho K|\psi_j^k| - \lambda\right)}{a\rho K - 1}\right)\right),$$

$$x_2 = \mathrm{sign}\left(\psi_j^k\right)\max\left(a\lambda, |\psi_j^k|\right).$$

QPADM-slack

Code:

```
for(int j = 0; j < p; j++){
phi = zmean(j)+umean(j)/pho;
//calculate x1
xmcp(0) = sign(phi)*min(a*lambda, max(0.0, a*(pho*K*abs(phi)-lambda)/(pho*K*a-1)));
//calculate x2
xmcp(1) = sign(phi)*max(a*lambda, abs(phi));
//calculate the function value under x1, x2 and 0
for(int i = 0; i < 3; i++){
hmcp(i) = 0.5*(xmcp(i)-phi)*(xmcp(i)-phi)+gpenalty(xmcp(i), a, lambda, penalty)/(pho*K);
}
beta(j) = xmcp(hmcp.indexmin());
}
```

Similarly, when the input logic variable `intercept == TRUE`, we update the first element of $\boldsymbol{\beta}^{s+1}$ by

```
beta(0) = zmean(0)+umean(0)/pho;
```

QPADM-slack

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

```
arma::vec vinik = vini.subvec(nk*k, nk*k+nk-1), etainik = etaini.subvec(k*nk,k*nk+nk-1);
yx.subvec(k*nk,k*nk+nk-1) = yk-xk*zini.col(k);
xi.subvec(k*nk,k*nk+nk-1) = yx.subvec(k*nk,k*nk+nk-1)+etainik+vinik/pho-tau*arma::ones(nk)/(n*pho);
for(int i = k*nk; i<k*nk+nk; i++){
if(xi(i) < 0){
xi(i) = 0;
}
}
```

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

```
eta.subvec(k*nk,k*nk+nk-1) = -yx.subvec(k*nk,k*nk+nk-1)+xi.subvec(k*nk,k*nk+nk-1)-vinik/pho-(1-tau)*arma::ones(n
for(int i = k*nk; i < k*nk+nk; i++){
if(eta(i) < 0){
eta(i) = 0;
}
}
```

QPADM-slack

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

Step 1: calculate $\left(I_p + X_k^{\mathrm{T}} X_k\right)^{-1}$ (this step is completed before the "`while`" loop as the matrix inversion does not need to be updated)

```
arma::mat tmp2, xk = x.rows(k*nk,k*nk+nk-1);
if(nk > p) tmp2 = inv(arma::eye(p,p)+xk.t()*xk);
else tmp2 = arma::eye(p,p)-xk.t()*inv(arma::eye(nk,nk)+xk*xk.t())*xk;
tmp.slice(k) = tmp2;
```

Step 2: calculate $\boldsymbol{\beta}_k^{s+1}$

```
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)+vi
```

QPADM-slack

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

```
u.col(k) = uini.col(k)+pho*(z.col(k)-beta);
```

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

```
yx.subvec(k*nk,k*nk+nk-1) = yk-xk*z.col(k);
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+n
```