Convex Optimization Homework 5, Assignment 3

Dinghuai Zhang, School of Mathematical Sciences, 1600013525

1 Problem Settings

$$\min_x \frac{1}{2}\|Ax - b\|_2^2 + \mu\|x\|_1 \tag{1}$$

It's a standard LASSO problem without constraints.

1.1 Data Building

```
n = 1024;
m = 512;
A = randn(m,n);
u = sprandn(n,1,0.1);
b = A*u;
mu = 1e-3;
x0 = rand(n,1);
```

2 Solve the dual problem using Augmented Lagrangian method

The dual problem of lasso is:

$$\min -b^T y + \frac{1}{2}||y||_2^2 \tag{2}$$

$$s.t. ||A^T y||_\infty \leq \mu \tag{3}$$

which can be formulated as:

$$\min\ f(y) = -b^T y + \frac{1}{2}||y||_2^2 + \mathrm{I}_{\{||z||_\infty \leq \mu\}} \tag{4}$$

$$s.t.\ z = A^T y \tag{5}$$

Impose augmented lagrangian method to this dual problem we solve

$$\begin{aligned}
\left(y^{+}, z^{+}\right) &= \underset{y,z}{\operatorname{argmin}} \; -b^T y + \frac{1}{2}||y||_2^2 + x^T(A^T y - z) + \frac{\beta}{2}||A^T y - z||_2^2 && (6)\\
&= \underset{y,z}{\operatorname{argmin}} \; -b^T y + \frac{1}{2}||y||_2^2 + \frac{\beta}{2}||(A^T y + \frac{x}{\beta}) - z||_2^2 && (7)\\
\text{s.t.} &\quad ||z||_\infty \leq \mu && (8)
\end{aligned}$$

with:

Algorithm 1: Augmented Lagrangian method for the dual problem

Input: γ, β, y, x, N, z

Output: solution $f(y)$

initializing $k = 1$

while $k < N$ **do**

  $y \leftarrow \underset{y}{\operatorname{argmin}}\{-b^T y + \frac{1}{2}||y||_2^2 + \frac{\beta}{2}||(A^T y + \frac{x}{\beta}) - \phi(A^T y + \frac{x}{\beta})||_2^2\}$

  $z \leftarrow \phi(A^T y + \frac{x}{\beta})$

  $x \leftarrow x - \gamma(A^T y - z)$

  $k \leftarrow k + 1$

end

return $f(y)$

where $(\phi(u))_i = \begin{cases} \mu & u_i \geqslant \mu \\ u_i & \text{otherwise} \\ -\mu & u_i \leqslant -\mu \end{cases}$ is the projection on a ∞-norm ball.

(It's easy to see that $\psi(u) = u - \phi(u)$ is soft-threshholding.)

For the y-sub problem, we can simply do gradient descent (or newton method), the gradient of y is $\text{grad} = y - b + \beta A((A^T y + \frac{x}{\beta}) - \phi(A^T y + \frac{x}{\beta}))$, so we simply do $y \leftarrow y - 0.1 * \text{grad}$.

The code is showed as follows:

```
function [x, out] = l1_auglagrange_dual(x0, A, b, mu, opts)
[m, n] = size(A);
pseudomu = 10000*mu;
beta = opts(1);
gamma = opts(2);
maxIter1 = opts(3);
maxIter2 = opts(4);
tol = 1e-6;

x = zeros(n, 1);
y = zeros(m, 1);
```

```
%continuation trick
while pseudomu >= mu
    iter1 = 1;
    while iter1 < maxIter1
        % y sub-problem: gradient descent
        iter2 = 1;
        while iter2 < maxIter2
            temp = softThreshold(A'*y+x/beta, pseudomu);
            grad = y - b + beta*A*temp;
            y = y - 0.1 * grad;

            iter2 = iter2 + 1;
         end

        temp=x/beta+A'*y;
        z = phi(temp, pseudomu);
        x = x + gamma*(A'*y-z);

        iter1 = iter1 + 1;
    end
pseudomu = pseudomu / 10;
end
out = 0.5 * sum_square(A*x-b) + mu * norm(x, 1);
end

function [x] = softThreshold(x, mu)
    x = sign(x) .* max(abs(x) - mu, 0);
end

function ret = phi(input, mu)
input = min(input, mu);
input = max(input, -mu);
ret = input;
end
```

3 Solve the dual problem using Alternating direction method of multipliers

The problem is still

$$\min\ f(y) = -b^T y + \frac{1}{2}||y||_2^2 + \mathrm{I}_{\{||z||_\infty \le \mu\}} \tag{9}$$

$$s.t.\ z = A^T y \tag{10}$$

In every iteration:

$$y = \underset{y}{\operatorname{argmin}} - b^T y + \frac{1}{2}||y||_2^2 + \mathrm{I}_{\{||z||_\infty \le \mu\}} + \frac{\beta}{2}||(A^T y + \frac{x}{\beta}) - z||_2^2 \tag{11}$$

$$= (I + \beta A A^T)^{-1}(\beta A z + b - Ax) \tag{12}$$

$$z = \underset{z}{\operatorname{argmin}} - b^T y + \frac{1}{2}||y||_2^2 + \mathrm{I}_{\{||z||_\infty \le \mu\}} + \frac{\beta}{2}||(A^T y + \frac{x}{\beta}) - z||_2^2 \tag{13}$$

$$= \phi(A^T y + \frac{x}{\beta}) \tag{14}$$

$$x = x + \gamma(A^T y - z) \tag{15}$$

Algorithm 2: Augmenteddirection method of multipliers for the dual problem

Input: γ, β, y, x, N, z
Output: solution $f(y)$
initializing $k = 1$
while $k < N$ **do**
  $y \leftarrow (I + \beta A A^T)^{-1}(\beta A z + b - Ax)$
  $z \leftarrow \phi(A^T y + \frac{x}{\beta})$
  $x \leftarrow x + \gamma(A^T y - z)$
  $k \leftarrow k + 1$
end
return $f(y)$

The code is showed as follows:

```
function [x, out] = l1_admm_dual(x0, A, b, mu, opts)
[m, n] = size(A);
pseudomu = 10000*mu;
beta = opts(1);
gamma = opts(2);
maxIter = opts(3);
inver = inv(beta*(A*A') + eye(m));
x = zeros(n, 1);
```

```
z = zeros(n, 1);


while pseudomu >= mu

    iter = 0;

    while iter < maxIter
        y = inver * (beta*A*z + b - A*x);
        z = phi(A'*y + x/beta, pseudomu);
        x = x + gamma*(A'*y-z);

        iter = iter + 1;
    end
    pseudomu = pseudomu / 10;
end
out = 0.5 * sum_square(A*x-b) + mu * norm(x, 1);
end

function ret = phi(input, mu)
input = min(input, mu);
input = max(input, -mu);
ret = input;
end
```

4 Solve the dual problem using Alternating direction method of multipliers with linearization

Reformulate lasso as:

$$\min \frac{1}{2}\|Ax - b\|_2^2 + \mu\|z\|_1 \tag{16}$$

$$s.t.\ x - z = 0 \tag{17}$$

The augmented lagrangian is :

$$\frac{1}{2}\|Ax - b\|_2^2 + \mu\|z\|_1 + \frac{\beta}{2}||x - z + u||_2^2 \tag{18}$$

So the admm iteration step is:

$$x \leftarrow (A^TA + \beta I)^{-1}(A^Tb + \beta(z - u)) \tag{19}$$

$$z \leftarrow \text{soft-thresholding}(x + u, \frac{\mu}{\beta}) \tag{20}$$

$$u \leftarrow u + \gamma(x - z) \tag{21}$$

If we adopt linearization, the update for x is changed to:

$$x \leftarrow x - c\,(\nabla f\,(x) + \beta\,(x - z + y)) \tag{22}$$

$$= x - c\left((A^TA + \beta I)x + \beta(u - z) - A^Tb\right) \tag{23}$$

The code is showed as follows:

```
function [x, out] = l1_admm_primal_linear(x0, A, b, mu, opts)
[m, n] = size(A);
pseudomu = 10000*mu;
beta = opts(1);
gamma = opts(2);
maxIter = opts(3);
c = opts(4);
tol = 1e-6;

x = x0;
u = randn(n, 1);
Q = inv(A'*A+beta*eye(n));
Atb = A' * b;

thresh = @(x,th) sign(x).* max(abs(x) - th,0);

while pseudomu >= mu
    for iter = 1:maxIter
        z = thresh(x + u, mu/beta);
        x = x - c*(AtA*x + beta*x + beta*(u-z) - Atb);
        u = u + gamma * (x - z);
    end
    pseudomu = pseudomu / 10;
end
out = 0.5 * sum_square(A*x-b) + mu * norm(x, 1);
end
```

5 Numerical results and interpretations

We can see the cpu time, the optimal value and error with cvx mosek in the tabluation:

Method	cpu time/s	error with cvx-mosek	optimal value
cvx-call-mosek	0.98	0.00e+00	7.2844266239e-02
auglagrange dual	0.19	2.97e-06	7.2844322774e-02
admm dual	0.17	2.75e-06	7.2844268531e-02
admm primal linear	0.65	3.04e-06	7.2844295011e-02

Alternative methods can achieve good results compared to cvx. However when implemented with linearization, the efficiency will be a little worse because of the approximation.