Convex Optimization Homework 5

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 problem using CVX calling Gurobi and Mosek

2.1 Solving the problem using Mosek in CVX

The code is showed as follows:

```
function [ x1,out1 ] = l1_cvx_mosek(x0, A, b, mu, opts1)
[m,n] = size(A);
cvx_solver mosek
cvx_begin
    variable x(n)
    minimize (0.5*square_pos(norm(A*x-b))+mu*norm(x,1))
cvx_end
x1 = x;
out1 = cvx_optval;
end
```

The result is

```
Optimizer summary
  Optimizer                    -                          time: 0.59
    Interior-point             - iterations : 8           time: 0.54
Optimal value (cvx_optval): +0.0746955
```

2.2 Solving the problem using Gurobi in CVX

The code is showed as follows:

```
function [x, cvx_optval]=l1_cvx_gurobi(x0, A, b, mu, opts2)
[n, ~] = size(x0);
cvx_solver gurobi
cvx_begin
    variable x(n)
    minimize (mu*norm(x,1) + 0.5*norm(A*x-b))
cvx_end
end
```

The result is

```
Barrier solved model in 13 iterations and 1.04 seconds
Optimal objective 7.46954736e-02
```

3 Solve the problem by directly calling Gurobi and Mosek

The original problem can be reformulated as:

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

$$s.t. x \preceq t \tag{3}$$

$$-t \preceq x \tag{4}$$

which is (when $b = Au$):

$$\min \frac{1}{2}\|Ay\|_2^2 + \mu 1^T t \tag{5}$$

$$s.t. y + u \preceq t \tag{6}$$

$$-t \preceq y + u \tag{7}$$

In this way we can call mosek and gurobi directly.

3.1 Solve the problem by directly calling Mosek

The code is showed as follows:

```
function [x, cvx_optval]=l1_mosek(x0, A, b, mu, opts3)
[~,n] = size(A);
q = [0.5*(A'*A), zeros(n,n);zeros(n,n),zeros(n,n)];
c = [zeros(n,1);mu*ones(n,1)];
a = [eye(n),eye(n);-eye(n),eye(n)];
v = A\b;
blc = [-v;v];
res = mskqpopt(q,c,a,blc,[],[],[],[],'minimize');
x = res.sol.itr.xx(1:1024)+v;
cvx_optval = res.sol.itr.pobjval;
end
```

The result is

```
Optimizer                    -                         time: 1.31
   Interior-point            - iterations : 10          time: 1.25
Optimal value (cvx_optval): +0.0746955
```

3.2 Solve the problem by directly calling Gurobi

The code is showed as follows:

```
function [x, cvx_optval]=l1_gurobi(x0, A, b, mu, opts4)
[~,n] = size(A);
At = [A -A];
model.Q = 0.5*sparse(At'*At);
c = mu*ones(2*n,1)- (b'*At)';
model.obj = c;
P = eye(2*n);
model.A = sparse(P);
l1 = zeros(2*n,1);
model.rhs = full(l1);
model.sense = '>';
params.method = 2;
res = gurobi(model, params);
x = res.x(1:n,1)-res.x(n+1:2*n,1);
cvx_optval = 0.5 * sum_square(A * x - b) + mu * norm(x, 1);
end
```

The result is

```
Barrier solved model in 9 iterations and 1.52 seconds
Optimal objective 7.4696220351e-02
```

4 Solve the problem with Projection Gradient Algorithm

The core iteration step of projection gradient algorithm is:

$$x^{+} = P_C(x - t\nabla g(x)) \tag{8}$$

We separate variable x into two parts of x_1, x_2 *s.t.* $x_1 \geq 0, x_2 \geq 0$ and define C as $\mathbb{R}^{+}{}_{2n}$. In the implementation we start from a large μ to ensure its convergence. The code is showed as follows:

```
x0 = [max(x0,0); max(-x0,0)];
[~,n] = size(A);
pseodumu = mu * 1e5;
tol = 1e-4;
alpha_l = 1e-6;
alpha_u = 1;
alpha = 1e-4;
iter = 200;
Atb = A' * b;
AtA = A' * A;
while pseodumu >= mu
    cur = pseodumu * ones(2*n,1) - [Atb; -Atb];
    AtAx0 = AtA* (x0(1:n)-x0((n+1):2*n));
    g0 = [AtAx0;-AtAx0] + cur;
    x = x0;
    g = g0;
    k = 1;
    while k < iter
        x0 = x;
        x = max(x - alpha*g, 0);
        g0 = g;
        AAz = AtA* (x(1:n)-x((n+1):2*n));
        g = [AAz;-AAz] + cur;
        y = g - g0;
        s = x - x0;
        BB = (s'*s) / (s'*y);
```

```
        if s'*y <=0
            alpha = alpha_u;
        else
            alpha = max(alpha_l, min(BB, alpha_u));
        end
        k = k + 1;
        if  norm(max(x-g, 0) - x) < tol
            break
        end
    end
    pseodumu = pseodumu / 10;
end
x =  x(1:n)-x((n+1):2*n);
cvx_optval = 0.5*sum_square(A*x - b) + mu*norm(x,1);
end
```

The result is **better** and **faster** than mosek's 7.4695473580e-02 in 1.16s:

```
sub-gradient: cpu:  0.46, cvx_optval: 7.4695410841e-02
```

5 Solve the problem with Subgradient Algorithm

The sub-gradient algorithm is showed as follows:

$$x^{(k)} = x^{(k-1)} - t_k g^{(k-1)} \tag{9}$$

where $g^{(k-1)}$ is the sub gradient of object function when $x = x^{(k-1)}$. In the implementation we start from a large μ to ensure its convergence as above. What's more, we use continuation method for step length α. The code is showed as follows:

```
function [x, cvx_optval]=l1_Subgradient(x0, A, b, mu, opts5)
pseudomu = 100;
iter = 500;
tol = 1e-7;
x = x0;

while pseudomu >= mu
    k = 1;
    alpha = 3e-4;
    while k < iter
        x0 = x;
        g = A' * (A * x - b)  + pseudomu * sign(x);
```

```
        if mod(k,50)==0
            alpha = alpha * 0.9;
        end
        x = x - alpha * g;
        if norm(x0-x) < tol
            break;
        end
        k = k + 1;
    end
    pseudomu = pseudomu / 10;
end
cvx_optval = 0.5*sum_square(A*x - b) + mu * norm(x,1);
end
```

The result is **better** and **faster** than mosek's 7.4695473580e-02 in 1.16s:

```
projectgradient: cpu:  0.11, cvx_optval: 7.4695365639e-02
```