Convex Optimization Homework 5, Assignment 2

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 gradient method for the smoothed primal problem

Because the l1 norm is not differentiable, we use gradient methods toward the smoothed version of object function, where we use

$$\phi_\lambda(x) = \begin{cases} x^2/(2\lambda) & |x| \le \lambda \\ |x| - \lambda/2 & |x| > \lambda \end{cases} \tag{2}$$

to substitute l1 norm. Then the gradient is:

$$grad = A^T(Ax-b) + \mu \begin{cases} x/\lambda & |x| \le \lambda \\ sign(x) & |x| > \lambda \end{cases} \tag{3}$$

The code is showed as follows:

```
function [x, out] = l1_grad_smooth(x0, A, b, mu, opts)
% use smooth() to replace 1-norm
% smooth(x) = x^2/(2*lambda)  if |x|<lambda
%           = |x|-lambda/2    if |x|>=lambda
[n, ~] = size(x0);
AtA = A' * A;
Atb = A' * b;
pseudomu = 10000*mu;
lambda = opts(1);
alpha0 = 1/max((eig(AtA))); % 3.3888e-04
maxIter = 700;
tol = 1e-6;
x = x0;

%continuation trick
while pseudomu >= mu
    alpha = alpha0; %
    iter = 1;
    grad = ones(n,1);
    while norm(grad) > tol && iter < maxIter
        grad = AtA * x - Atb + pseudomu * grad_smooth(x, lambda);
        x = x - alpha * grad;
        if norm(x0 - x) < tol
            break
        end
        iter = iter + 1;
    end
    pseudomu = pseudomu / 10;
end
out = 0.5 * sum_square(A*x-b) + mu * norm(x, 1);
end

function ret = grad_smooth(x, lambda)
mask = abs(x) < lambda;
x(mask) = x(mask) / lambda;
x(~mask) = sign(x(~mask));
ret = x;
end
```

3 Solve the problem using fast gradient method for the smoothed primal problem

Based on the former section, we use a respective accelerating algorithm of the smoothed method, the code is showed as follows:

```
function [x, out] = l1_fast_grad_smooth(x0, A, b, mu, opts)
% use smooth() to replace 1-norm
% smooth(x) = x^2/(2*lambda)  if |x|<lambda
%           = |x|-lambda/2    if |x|>=lambda
[n, ~] = size(x0);
AtA = A' * A;
Atb = A' * b;
pseudomu = 10000*mu;
lambda = opts(1);
alpha0 = 1/max((eig(AtA))); %
maxIter = 90;
tol = 1e-6;
x = x0;

%continuation trick
while pseudomu >= mu
    alpha = alpha0; %
    iter = 0;
    grad = ones(n,1);
    v = x;
    while norm(grad) > tol && iter < maxIter
        if mod(iter, 200) == 0
            alpha = alpha * 0.8;
        end

        x_old = x;
        theta = 2/(iter + 1);
        y = (1 - theta)*x + theta*v;
        grad = AtA * y - Atb + pseudomu * grad_smooth(y, lambda);
        x = y - alpha*grad;
        v = x + (1/theta)*(x - x_old);

        iter = iter + 1;
    end
    pseudomu = pseudomu / 10;
```

```
end
out = 0.5 * sum_square(A*x-b) + mu * norm(x, 1);
end

function ret = grad_smooth(x, lambda)
mask = abs(x) < lambda;
x(mask) = x(mask) / lambda;
x(~mask) = sign(x(~mask));
ret = x;
end
```

4 Solve the problem with proximal gradient method for the primal problem

The proximal operator for l1 norm is:

$$h(x) = \|x\|_1, \quad prox_h(x)_i = \begin{cases} x_i - 1 & x_i > 1 \\ 0 & -1 \le x_i \le 1 \\ x_i + 1 & x_i < -1 \end{cases}$$

For an optimization problem of

$$\min_x f(x) = g(x) + h(x), \tag{4}$$

the proximal gradient method is:

$$x \leftarrow prox_{th}(x - t\nabla g(x)) \tag{5}$$

The code is showed as follows:

```
function [x, out] = l1_proximal_grad(x0, A, b, mu, opts)
AtA = A' * A;
Atb = A' * b;
pseudomu = 10000*mu;
alpha0 = 1/max((eig(AtA)));
maxIter = 425;
x = x0;

proximal = @(x, t) sign(x).*max(abs(x) - t, 0);

%continuation trick
```

```
while pseudomu >= mu
    alpha = alpha0;
    iter = 1;
    while iter < maxIter
        grad = AtA * x - Atb;
        x = x - alpha * grad;
        x = proximal(x, alpha*pseudomu);

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

5 Solve the problem with fast proximal gradient method for the primal problem (FISTA)

The algorithm for FISTA is:

Algorithm 1: Fast Proximal Gradient Method

Input: Initial vector x_0,tolerance error ϵ and initial learning rate α_0

Output: solution x

initializing velocity $v_0 := x_0$
while $k < N$ **do**
    $\theta_k = \frac{2}{k+1}$
    $y = (1 - \theta_k)x_{k-1} + \theta_k v_{k-1}$
    $x_k := prox_{\alpha_k h}(x_{k-1} - \alpha_k \nabla g(x_{k-1}))$
    $v_k = x_{k-1} + \frac{1}{\theta_k}(x_k - x_{k-1})$
    $k := k + 1$
end
return $x = x_k$;

The code is showed as follows:

```
function [x, out] = l1_fast_proximal_grad(x0, A, b, mu, opts)
AtA = A' * A;
Atb = A' * b;
pseudomu = 10000*mu;
alpha0 = 1/max((eig(AtA)));
maxIter = 100;
```

```
x = x0;

proximal = @(x, t) sign(x).*max(abs(x) - t, 0);

%continuation trick
while pseudomu >= mu
    alpha = alpha0;
    iter = 0;
    v = x;
    while iter < maxIter
        x_old = x;
        theta = 2/(iter + 1);
        y = (1 - theta)*x + theta*v;
        grad = AtA * y - Atb;
        x = proximal(y - alpha * grad, alpha*pseudomu);
        v = x + (1/theta)*(x - x_old);

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

6 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	7.2844399533e-02
smooth gradient	0.82	3.86e-6	7.2844322774e-02
fast smooth gradient	0.17	3.60e-6	7.2844349700e-02
proximal gradient	0.49	3.04e-6	7.2844266059e-02
fast proximal gradient	0.17	2.82e-6	7.2844266584e-02

It's obvious that the FISTA is the strongest algorithm among these. With nestorov acceleration and proximal iteration, it achieves amazing effects. Also we can notice that methods with acceleration, which are two order algorithms, are much faster than their original algorithm, which are one order algorithms.