ELEC5470 - Convex Optimization, Fall 2018-19

Homework Set #4

Name: Tingyuan LIANG **Major:** MPhil in ECE **E-mail:** tliang@ust.hk

1) **Solution:** The original problem is to:

$$\min_{\beta\in\mathbf{R}^p} \|\mathbf{y}-\mathbf{X}\beta\|_2^2+\lambda\|\beta\|_1 \tag{1}$$

We can re-formulate the problem as below:

$$\min_{\beta,\mathbf{t}\in\mathbf{R}^p} (\mathbf{y}-\mathbf{X}\beta)^T(\mathbf{y}-\mathbf{X}\beta)+\lambda\mathbf{1}^T\mathbf{t} \tag{2}$$

subject to:

$$\beta-\mathbf{t}\leq\mathbf{0} \tag{3}$$

$$-\mathbf{t}-\beta\leq\mathbf{0} \tag{4}$$

In order to eliminate the linear inequality constraints, we can introduce barrier functions and transform the original problem into anther one approximate as below:

$$\min_{\beta,\mathbf{t}\in\mathbf{R}^p} (\mathbf{y}-\mathbf{X}\beta)^T(\mathbf{y}-\mathbf{X}\beta)+\lambda\mathbf{1}^T\mathbf{t}-\frac{1}{\delta}[\sum_{i=1}^{p} log(-(\beta_i-t_i))+\sum_{i=1}^{p} log(-(-\beta_i-t_i))] \tag{5}$$

Let

$$f(\beta,\mathbf{t})=(\mathbf{y}-\mathbf{X}\beta)^T(\mathbf{y}-\mathbf{X}\beta)+\lambda\mathbf{1}^T\mathbf{t}-\frac{1}{\delta}\sum_{i=1}^{p} log(t_i^2-\beta_i^2) \tag{6}$$

where

$$\delta>0 \tag{7}$$

when $\delta\to\infty$, (5) is equivalent to (1).

Then we can get it gradient:

$$\nabla_\beta f=2\mathbf{X}^T(\mathbf{X}\beta-\mathbf{y})+\frac{1}{\delta}[\frac{2\beta_1}{t_1^2-\beta_1^2}\ ,\ \dots\ ,\ \frac{2\beta_p}{t_p^2-\beta_p^2}]^T \tag{8}$$

$$\nabla_{\mathbf{t}} f=\lambda\mathbf{1}-\frac{1}{\delta}[\frac{2t_1}{t_1^2-\beta_1^2}\ ,\ \dots\ ,\ \frac{2t_p}{t_p^2-\beta_p^2}]^T \tag{9}$$

Therefore, the vector of the gradient of f can be described as below:

$$\mathbf{g}=\nabla f=[\nabla_\beta f\ ,\ \nabla_{\mathbf{t}} f]^T \tag{10}$$

Further, the Hessian can be obtained:

$$\frac{\partial^2 f}{\partial\beta_i^2}=[2\mathbf{X}^T\mathbf{X}]_{i,i}+\frac{2}{\delta}\frac{t_i^2+\beta_i^2}{(t_i^2-\beta_i^2)^2} \tag{11}$$

$$\frac{\partial^2 f}{\partial\beta_i\partial\beta_j}=[2\mathbf{X}^T\mathbf{X}]_{i,j},\qquad (i\neq j) \tag{12}$$

$$\frac{\partial^2 f}{\partial t_i^2}=\frac{2}{\delta}\frac{t_i^2+\beta_i^2}{(t_i^2-\beta_i^2)^2} \tag{13}$$

$$\frac{\partial^2 f}{\partial t_i \partial t_j} = 0, \qquad (i \neq j) \tag{14}$$

$$\frac{\partial^2 f}{\partial \beta_i \partial t_i} = \frac{\partial^2 f}{\partial t_i \partial \beta_i} = -\frac{4}{\delta} \frac{\beta_i t_i}{(t_i^2 - \beta_i^2)^2} \tag{15}$$

$$\frac{\partial^2 f}{\partial \beta_i \partial t_j} = \frac{\partial^2 f}{\partial t_j \partial \beta_i} = 0, \qquad (i \neq j) \tag{16}$$

Therefore, the Hessian matrix of $f(\beta, \mathbf{t})$ can be described as below:

$$\mathbf{H} = \begin{bmatrix} 2\mathbf{X}^T\mathbf{X} + \mathbf{P_1} & \mathbf{P_2} \\ \mathbf{P_2} & \mathbf{P_1} \end{bmatrix}_{2p \times 2p} \tag{17}$$

where

$$\mathbf{P_1} = \mathrm{diag}\left(\frac{2}{\delta} \frac{t_1^2 + \beta_1^2}{(t_1^2 - \beta_1^2)^2} \ , \ \ldots \ , \ \frac{2}{\delta} \frac{t_p^2 + \beta_p^2}{(t_p^2 - \beta_p^2)^2} \right) \tag{18}$$

$$\mathbf{P_2} = \mathrm{diag}\left(-\frac{4}{\delta} \frac{\beta_1 t_1}{(t_1^2 - \beta_1^2)^2} \ , \ \ldots \ , \ -\frac{4}{\delta} \frac{\beta_p t_p}{(t_p^2 - \beta_p^2)^2} \right) \tag{19}$$

According to the gradient and the Hessian, with MATLAB, we can implement the barrier method based on Newton method. We first show the results below, including the result based on $\mu = 10$ and $\mu = 100$. Please note that we initialize $\beta = \mathbf{1}$.

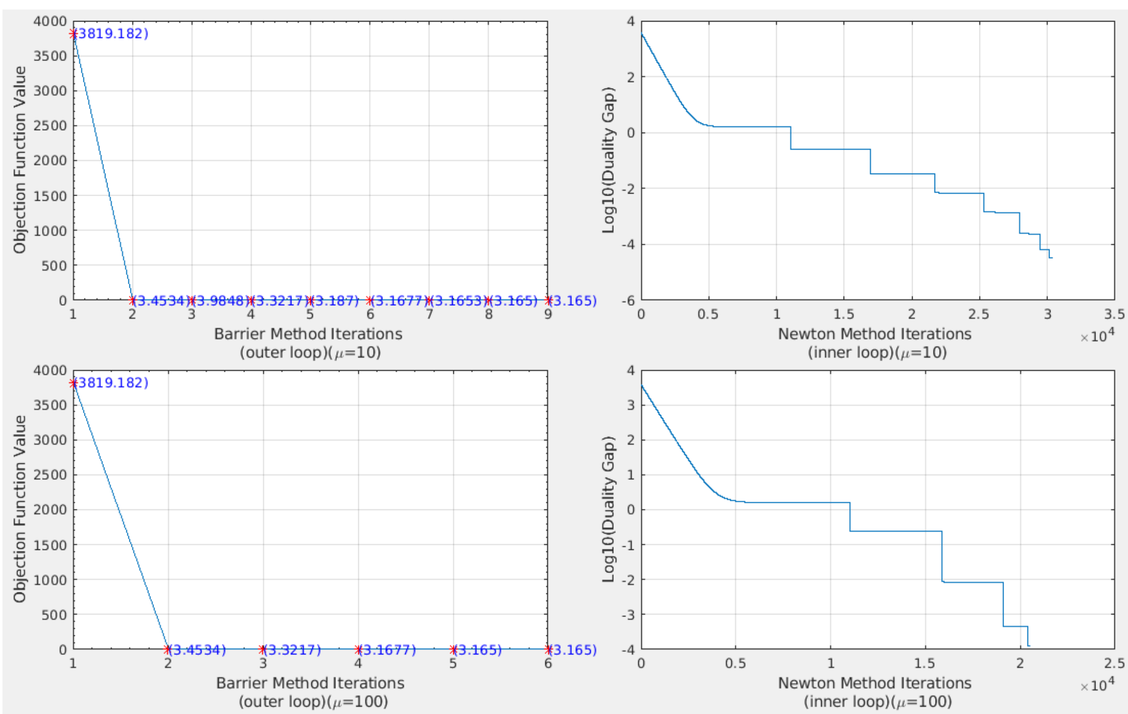

Figure 1. result

Based on the given data, the optimal value of object function is 3.1650 and

the optimal solution β is $[0.0004\ ,\ 0.0028\ ,\ 0.9962\ ,\ 0.0004\ ,\ 7.0136\ ,\ 0.0031\ ,\ 0.0119\ ,\ 0.0011\ ,\ 0.0021\ ,\ 3.0018]^T$

According to the gradient and the Hessian, with MATLAB, we can implement the barrier method based on Newton method:

---

given strictly feasible point $\mathbf{x}, \mathbf{t}$, $\delta := \delta^{(0)} > 0$, tolerance $\epsilon > 0$

repeat

1. Centering step. Compute $\mathbf{x}^*(\delta)$ and $\mathbf{t}^*(\delta)$ by minimizing $f = f_0 + \phi/\delta$, with Newton method based on backtracking.
2. Update. $\mathbf{x} = \mathbf{x}^*(\delta)$ and $\mathbf{t} = \mathbf{t}^*(\delta)$
3. Stopping criterion. **quit** if $2p/\delta < \epsilon$. Please note that we have $2p$ constraints according to (3)(4).
4. Increase δ. $\delta := \mu\delta$

To implement the barrier method, the following source code are involved.

a) **The Function to Get Gradient and Hessian**:

```
function [g,H]=g_H_comp(X,y,lambda,p,delta,x)

    beta = x(1:p);
    t = x(p+1:2*p);

    %compute the gradient
    g_b_0 = 2*X'*(X*beta-y);
    g_b_1 = zeros(p,1);
    for i=1:p
        g_b_1(i)=2*beta(i)/(t(i)*t(i)-beta(i)*beta(i));
    end
    g_b_1 = g_b_1/delta;
    g_b = g_b_0 + g_b_1;

    g_t_0 = lambda*ones(p,1);
    g_t_1 = zeros(p,1);
    for i=1:p
        g_t_1(i)=2*t(i)/(t(i)*t(i)-beta(i)*beta(i));
    end
    g_t_1 = -g_t_1/delta;
    g_t = g_t_0 + g_t_1;

    g = [g_b;g_t];

    %compute the hessian matrix
    P1_v = zeros(p,1);
    for i=1:p
        P1_v(i) = 2*(t(i)^2+beta(i)^2)/(t(i)^2-beta(i)^2)^2/delta;
    end
    P1 = diag(P1_v);

    P2_v = zeros(p,1);
    for i=1:p
        P2_v(i) = -4*(t(i)*beta(i))/(t(i)^2-beta(i)^2)^2/delta;
    end
    P2 = diag(P2_v);

    H = [2*X'*X+P1, P2;P2,P1];

end
```

b) **The Function for Barrier Method**:

```
function [opt_x,opt_value]=barrier_lty(X,y,lambda,p,delta0,x,error_tol,mu)

    delta = delta0;
    global obj_val;
    global obj_it;

    cnt = 1;
    while (1)
        % 1.Centering step.Compute x'(t) by minimizing tf+Ï , subject to Ax=b.
        [newx,newvalue] = backtracking_newton(X,y,lambda,p,delta,x,error_tol);

        % 2.Update.x:=x'(t).
        x = newx;  opt_x = x
```

```
        opt_value = newvalue
        obj_val = [obj_val,newvalue];
        cnt = cnt + 1;
        obj_it = [obj_it, cnt];

        % 3.Stopping criterion. quit if m/t < error_tol.
        if (2*p/delta<error_tol)
            break
        end

        % 4.Increase t.t:=ÎŒt
        delta = mu*delta
    end

end
```

c) **The Function for Newton Method, which the barrier method is based on**:

```
function [newx,newf_value]=backtracking_newton(X,y,lambda,p,delta,x,error_tol)

    [g,H]=g_H_comp(X,y,lambda,p,delta,x);
    deltax = -inv(H)*g;
    decrement_2 = g'*inv(H)*g;
    t = 0.001;
    newx = x;
    newf_value = eval_obj(X,y,lambda,p,delta,x);

    global newton_vals;

    a = 0.1;
    b = 0.9;

    % refer to paper: https://web.stanford.edu/~boyd/papers/pdf/l1_ls.pdf
    % the dual value can be obtained:
    s = min(lambda./(abs(2*X'*(X*x(1:p)-y))));
    v = 2*s*(X*x(1:p)-y);

    % 2.Stopping criterion. quit if Î»2/2 â   error_tol.
    while (decrement_2/2>error_tol)

        % 3.Line search. Choose step size t by backtracking line search.
        newx = x+t*deltax;
        while (eval_obj(X,y,lambda,p,delta,newx) >= eval_obj(X,y,lambda,p,delta,x) +a*t
            *(g')*deltax)
            t = b*t;
            newx = x+t*deltax;
        end

        % 4a.Update. x:=x+tâ xnt
        x = newx;
        newf_value = eval_obj(X,y,lambda,p,delta,x);

        % 4b. Calculate the duality gap
        newton_vals = [newton_vals,eval_obj_tmp(X,y,lambda,p,delta,x)-G(v,y)];

        % 1.Compute the Newton step and decrement.
        %    â xnt:= â  â  ^(2)f(x)^(â 1)â f(x);  Î»_2:= â f(x)^(T)â  ^(2)f(x)^(â 1)
            â f(x).
        [g,H]=g_H_comp(X,y,lambda,p,delta,x);
        deltax = -inv(H)*g;
        decrement_2 = g'*inv(H)*g;
    end

end

function rs=G(v,y)
    rs = -0.25*v'*v-v'*y;
end
```

d) **The Function to Evaluate the Approximate Object Function**:

```
function rs=eval_obj(X,y,lambda,p,delta,x)

    % object function
    beta = x(1:p);
    t = x(p+1:2*p);

    P=(y-X*beta)'*(y-X*beta);
    Q=lambda*ones(1,p)*t;

    R=0;
    for i=1:p
        R=R+log(t(i)*t(i)-beta(i)*beta(i));
    end
    R=-R/delta;
    rs=P+Q+R;

end
```

e) **The Function to Evaluate the Original Object Function**:

```
function rs=eval_obj_tmp(X,y,lambda,p,delta,x)

    % object function
    beta = x(1:p);
    t = x(p+1:2*p);

    P=(y-X*beta)'*(y-X*beta);
    Q=lambda*ones(1,p)*abs(beta);

    rs=P+Q;

end
```

f) **The Initialization of Input and The Plotting of Figures**:

```
clear all;
clf;
close all;
randn('seed',1);

beta = zeros(10,1);
beta(3) = 1;
beta(5) = 7;
beta(10) = 3;
global newton_vals;
newton_vals = [];
n=100;
p=10;
mu=10;

X=randn(n,p);
y = X*beta + 0.1*randn(n,1);
lambda = 0.2;

beta = ones(10,1);
t=20*ones(p,1);
x=[beta;t];

delta0=1/lambda;

initial_val=eval_obj(X,y,lambda,p,delta0,x)

global obj_val;
global obj_it;

obj_val = [initial_val];
obj_it = [1];

[opt_x,opt_value]=barrier_lty(X,y,lambda,p,delta0,x,1e-6,mu);

opt_x = opt_x(1:p)

subplot(221);
plot(obj_it,obj_val);
set(gca,'XMinorTick','on','YMinorTick','on');
grid on;
hold on;
plot(obj_it,obj_val,'r*');
xlabel({'Barrier Method Iterations ';'(outer loop)(\mu=10)'})
ylabel('Objection Function Value')
for i=1:size(obj_it,2)
    text(obj_it(i),obj_val(i),['(',(num2str(obj_val(i))),')'],'color','b');
end

subplot(222);
plot(log(newton_vals)/log(10));
grid on;
xlabel({'Newton Method Iterations ';'(inner loop)(\mu=10)'})
ylabel('Log10(Duality Gap)')


%clear all;

newton_vals = [];
mu=100;

delta0=1/lambda;

initial_val=eval_obj(X,y,lambda,p,delta0,x)

obj_val = [initial_val];
obj_it = [1];

[opt_x,opt_value]=barrier_lty(X,y,lambda,p,delta0,x,1e-6,mu);

opt_x = opt_x(1:p)

subplot(223);
plot(obj_it,obj_val);
set(gca,'XMinorTick','on','YMinorTick','on');
grid on;
hold on;
plot(obj_it,obj_val,'r*');
xlabel({'Barrier Method Iterations';' (outer loop)(\mu=100)'})
```

```
ylabel('Objection Function Value')
for i=1:size(obj_it,2)
    text(obj_it(i),obj_val(i),['(',(num2str(obj_val(i))),')'],'color','b');
end

subplot(224);
plot(log(newton_vals)/log(10));
grid on;
xlabel({'Newton Method Iterations ';'(inner loop)(\mu=100)'})
ylabel('Log10(Duality Gap)')

set(gcf, 'position', [300 100 1200 800]);
```

.

2) **Solution:**

a) According to the observation, we can determine the weights in every layer. From the inputs to the output, there are The three layers and they can be described in a matrix, a vector and a vector respectively.

The input is a vector, $[\mathbf{x}^T, 1]^T$, where $\mathbf{x} = [x_1,\ x_2,\ x_3]^T$.

The weights of the first layers is a matrix, $\mathbf{M}$, shown below:

$$\mathbf{M} = \begin{bmatrix} -a & a/2 & a/2 & 0 \\ a/2 & -a & a/2 & 0 \\ a/2 & a/2 & -a & 0 \end{bmatrix}_{3\times 4} \tag{20}$$

where

$$a > 0 \tag{21}$$

Therefore the output of the first layer is $\mathbf{o_1} = \mathbf{M}[\mathbf{x}^T, 1]^T$.

The input of the second layer is $[\mathbf{o_1^T}, 1]^T$. The weights of the second layers is a vector, $\mathbf{n} = [1,\ 1,\ 1,\ -0.5]^T$, and therefore the output of the second layer is $o_2 = \mathbf{n^T}[\mathbf{o_1}, 1]^T$.

The input of the third layer is $[o_2, 1]^T$. The weight of the third layer is a vector, $\mathbf{r} = [-7.5,\ 2.5]^T$, and therefore the output of the second layer is $o_3 = \mathbf{r^T}[o_2, 1]^T$.

.

b) Suppose

$$\mathbf{x_n} \in \mathbf{R}^p \tag{22}$$

and

$$\mathbf{w} = [w_0, \ \dots \ , \ w_p] \in \mathbf{R}^{p+1} \tag{23}$$

Given $\mathbf{x}_n$, the object function is:

$$g(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(t_n - f(\mathbf{x_n}, \mathbf{w}))^2 \tag{24}$$

in order to solve the following problem:

$$\min_{\mathbf{w}} \quad g(\mathbf{w}) \tag{25}$$

We should first get the gradient of $g(\mathbf{w})$ and the procedure is shown below:

Let

$$\mathbf{z}_i = [1 \ , \ x_{i,1} + x_{i,1}^2 + x_{i,1}^3, \ \dots \ , \ x_{i,p} + x_{i,p}^2 + x_{i,p}^3]^T \in \mathbf{R}^{p+1} \tag{26}$$

and

$$\mathbf{Z} = [\mathbf{z}_1, \dots \ , \ , \mathbf{z}_N]^T \in \mathbf{R}^{N,p+1} \tag{27}$$

$$\mathbf{u} = [f(\mathbf{x}_1, \mathbf{w}), \ \dots \ , \ f(\mathbf{x}_N, \mathbf{w})]^T \in \mathbf{R}^N \tag{28}$$

$$\mathbf{t} = [t_1, \ \dots \ , \ t_N]^T \in \mathbf{R}^N \tag{29}$$

Then we can get:

$$\nabla_{\mathbf{w}} g = -\mathbf{Z}^T(\mathbf{t} - \mathbf{u}) = \mathbf{Z}^T(\mathbf{u} - \mathbf{t}) \tag{30}$$

Based on this result, we can derive a gradient descent training algorithm as below:

---

given a starting point $\mathbf{w} \in \mathbf{R}^{p+1}$

repeat

1. $\Delta\mathbf{w} := -\nabla_{\mathbf{w}} g$
2. Line search. Choose step size t via exact or backtracking line search.
3. Update. $\mathbf{w} := \mathbf{w} + t\Delta\mathbf{w}$

until stopping criterion is satisfied.

.