EE364 HW6

Wentao Jiang
wentao@stanford

9.23 (a) $f(x) = -\sum_i^m \log(b_i - a_i^T x)$. For $x = x_0 + t\Delta x$

$$f(x) = -\sum_i^m \log[b_i - a_i^T x_0 - t\cdot(a_i^T \Delta x)] = g(t)$$
$$= -\sum_i^m \log[\alpha_i - t\cdot\beta_i]$$

We can pre-calculate α_i and β_i, the cost is $2mn$.
Then the cost of evaluating g and g' is $3m \sim O(m)$
Without pre-computation, the cost is $O(m\cdot n)$.

(b) Very similar to (a), $f(x) = \log(\sum_i^m \exp[a_i^T x_0 + b_i + t(a_i^T \Delta x)])$
$$= \log(\sum_i^m \exp(\alpha_i + t\cdot\beta_i)) = g(t)$$

Pre-computation costs $2mn$, then evaluating g cost $O(m)$.
Without pre-computation, the cost is $O(mn)$.

(c) $f(x) = [Ax_0 - b + t(A\Delta x)]^T \begin{pmatrix} P_0 + (x_0)_1 P_1 + \cdots + (x_0)_n P_n \\ + t((\Delta x)_1 P_1 + \cdots + (\Delta x)_n P_n \end{pmatrix}^{-1} [Ax_0 - b + t(A\Delta x)]$

Let's look at $P(x) = P(x_0) + t\cdot \sum_i^n (\Delta x)_i P_i = D + t\cdot B$

We know D and $B \in S^m$, $\therefore$ We could decompose $D = LL^T$, and diagonize $B = U\Lambda U^T$

$\therefore f(x) = g(t) = (y+tz)^T (LL^T + t\cdot U\Lambda U^T)^{-1} (y+tz)$
$$= (y+tz)^T L^{-T} (I + t L^{-1} B L^{-T})^{-1} L^{-1} (y+tz)$$

$\therefore$ We should actually directly diagonize $L^{-1} B L^{-T} = U\Lambda U^T$,

Then $f(x) = g(t) = (U^{-1} L^{-1} (y+tz))^T (I + t\Lambda)^{-1} U^{-1} L^{-1} (y+tz)$

Redefine $y = U^{-1} L^{-1} (Ax_0 - b)$, $z = U^{-1} L^{-1} A\Delta x$

Then $f(x) = g(t) = (y+tz)^T (I + t\Lambda)^{-1} (y+tz) = \sum_i^m \frac{(y_i + tz_i)^2}{1 + t\Lambda_i}$

The complexity of decomposing D and digonizing $L^{-1}BL^{-T}$ ~~takes~~ are m^3,
Computing $D = P(x_0)$ cost $m^2(n+1)$, computing $B = \sum_i^m (\Delta x_i) P_i$ cost $m^2 n$.

$\therefore$ In total $O(m^3 + m^2 n)$ for pre-computation.

After $g(t) = \sum_i^m \frac{(y_i + t z_i)^2}{1 + t\lambda_i}$, then it only cost $O(m)$.

With out pre-computation, we need $2mn$ for $Ax - b$, $m^2 n$ for $P(x)$,
$\frac{1}{3} m^3$ for $P(x)^{-1}(Ax-b)$, then $2m$ for finally $(Ax-b)^T P(x)^{-1} (Ax-b)$

$\therefore$ In total $O(m^3 + m^2 n)$.

9.30 First of all, $(\nabla f(x))_i = \sum_j (a_j)_i \frac{1}{1 - a_j^T x} + \frac{2x_i}{1 - x_i^2}$

$$= \sum_j \frac{(a_j)_i}{1 - a_j^T x} + \frac{1}{1 - x_i} - \frac{1}{1 + x_i}$$

If we write $A = \begin{pmatrix} a_1^T \\ \vdots \\ a_m^T \end{pmatrix}$, then $\nabla f(x) = A^T 1./(1 - Ax)$

See the attached codes and plots.

For Newton's method $(\nabla^2 f(x))_{ij} = \partial_j \left(\sum_k \frac{(a_k)_i}{1 - (a_k^T x)} + \frac{1}{1 - x_i} - \frac{1}{1 + x_i} \right)$

$$= \frac{(a_k)_i (a_k)_j}{(1 - a_k^T x)^2} + \frac{1}{(1 - x_i)^2} \delta_{ij} + \frac{1}{(1 + x_i)^2} \delta_{ij}$$

See the attached codes and plots.

9.31 See the attached codes.

10.2 (a) Compare equation $\begin{bmatrix} I & A^T \\ A & 0 \end{bmatrix}\begin{pmatrix} v \\ w \end{pmatrix} = \begin{pmatrix} -\nabla f(x) \\ 0 \end{pmatrix} \Rightarrow \begin{cases} v + A^T w + \nabla f(x) = 0 \\ Av = 0 \end{cases}$

to optimality conditions $\begin{cases} \nabla f(x) + A^T v = 0 \\ Ax = b \end{cases}$

$x \Rightarrow v$, $b = 0$, $v \Rightarrow w$, $\nabla f(x) \Rightarrow \nabla f(x) + v$

$\therefore$ $\begin{bmatrix} I & A^T \\ A & 0 \end{bmatrix}\begin{pmatrix} v \\ w \end{pmatrix} = \begin{pmatrix} -\nabla f(x) \\ 0 \end{pmatrix}$ are the optimality conditions of

the problem minimize $\|\nabla f(x) + v\|_2^2$ over v

subject to $Av = 0$

$\therefore$ i.e., $v = \underset{Au=0}{\operatorname{argmin}} \|-\nabla f(x) - u\|_2 = \Delta x_{pg}$

As for w, $v + A^T w = -\nabla f(x)$ and we know $Av = 0$

i.e., $v^T A^T w = 0$, $A^T w$ is perpendicular to v, i.e., $A^T w$ has to be the component of $\nabla f(x)$ in the subspace perpendicular to $\mathcal{N}(A)$.

i.e., $w = \underset{y}{\operatorname{argmin}} \|\nabla f(x) + A^T y\|_2$

(b) The reduced problem is: minimize $f(Fz + \hat{x})$

The gradient of this problem is $\nabla f(Fz + \hat{x}) \cdot F$

~~Since range of F is $\mathcal{N}(A)$,~~ here f is still a function of x.

For $g(z) = f(Fz + \hat{x})$, then $\nabla g = [(\nabla f)^T F]^T = F^T \cdot \nabla f$

The projected gradient Δx_{pg} can also be expressed as

$\Delta x_{pg} = F \cdot \underset{z}{\operatorname{argmin}} \|\nabla f(x) + Fz\|_2$, $F^T F = I$

$\therefore$ It's equivalent to $\Delta x_{pg} = F \cdot \underset{z}{\operatorname{argmin}} \|F^T(\nabla f(x) + Fz\|_2$

$= F \cdot \underset{z}{\operatorname{argmin}} \|F^T \nabla f(x) + z\|_2$

$= F \cdot -\nabla g$

$\therefore$ $\Delta x_{pg} = -F\nabla g$, where $g(z) = f(Fz + \hat{x})$.

(c) For the reduced problem, $\Delta x = F\Delta z$. assuming $F^TF=I$

$$\therefore \Delta z = F^TF\Delta z = F^T\Delta x = -F^TF\nabla g = -\nabla g$$

i.e., it's a usual gradient descent method on the reduced problem.

$\therefore$ Condition: Sublevel set $S = \{x \mid f(x) \le f(x_0),\ Ax=b\}$ is closed.

For unique optimal solution, $g(z) = f(Fz+\vec{x})$ is strongly convex.

A6.5 (a) First follow the assumption $y_1 \le y_2 \le \dots \le y_m$

$$\therefore \alpha \le \phi' \le \beta \Rightarrow \alpha \le \frac{y_{i+1}-y_i}{z_{i+1}-z_i} \le \beta \qquad \alpha > 0, \quad \therefore y_{i+1} \ne y_i$$

$\therefore$ It's equivalent to $\dfrac{y_{i+1}-y_i}{\beta} \le z_{i+1}-z_i \le \dfrac{y_{i+1}-y_i}{\alpha}$

$$P(v_i) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-(z_i - a_i^Tx)^2/2\sigma^2\right)$$

$\therefore$ The log likelihood $\quad l(x,z) = -\dfrac{1}{2\sigma^2}\sum_1^m (z_i - a_i^Tx)^2 - \dfrac{m}{2}\log(2\pi\sigma^2)$

i.e., the max-likelihood problem is

$$\begin{array}{ll} \text{minimize} & \sum_1^m (z_i - a_i^Tx)^2 \\ \text{subject to} & \dfrac{y_{i+1}-y_i}{\beta} \le z_{i+1}-z_i \le \dfrac{y_{i+1}-y_i}{\alpha} \end{array}$$

which is convex.

(b) See attachments.

A9.5 (a) ~~Newton~~ Newton Step is given by $\begin{pmatrix} \nabla^2 f & A^T \\ A & 0 \end{pmatrix}\begin{pmatrix} v \\ w \end{pmatrix} = \begin{pmatrix} -\nabla f \\ 0 \end{pmatrix}$

$\therefore$ For the centering problem,

they are

$$\begin{pmatrix} 1/x_1^2 & & 0 & \\ & \ddots & & A^T \\ 0 & & 1/x_n^2 & \\ & A & & 0 \end{pmatrix}\begin{pmatrix} v \\ \\ w \end{pmatrix} = \begin{pmatrix} -c_1 + 1/x_1 \\ \vdots \\ -c_n + 1/x_n \\ \vec{0} \end{pmatrix}$$

~~Block elimination~~ $\cancel{\Rightarrow A(1/x_i^2)^{-1}A^T}$

Denote $X = \begin{pmatrix} 1/x_1^2 & & \\ & \ddots & \\ & & 1/x_n^2 \end{pmatrix}$, $B = -c + \begin{pmatrix} 1/x_1 \\ \vdots \\ 1/x_n \end{pmatrix} = -\nabla f$

$\therefore$ Block elimination: $AX^{-1}(Xv + A^T w) = AX^{-1}B$

$$\Rightarrow \quad AX^{-1}A^T w = AX^{-1}B$$

Once w is solved from above,

$$\Delta x_{newton} = X^{-1}(B - A^T w).$$

KKT optimality conditions are $\nabla f(x^*) + A^T v^* = 0$

i.e., $v^* = w$ when the iteration is nearly converged.

Remaining of (a), and (b), (c) are attached.

In (c), for phase I, in order to express $Ax = b$ in terms of $z = x + (t-1)\vec{1}$, notice

$$Ax = b \Leftrightarrow Az = b + A(t-1)\vec{1} \Leftrightarrow Az - A \cdot t \cdot \vec{1} = b - A \cdot \vec{1}$$

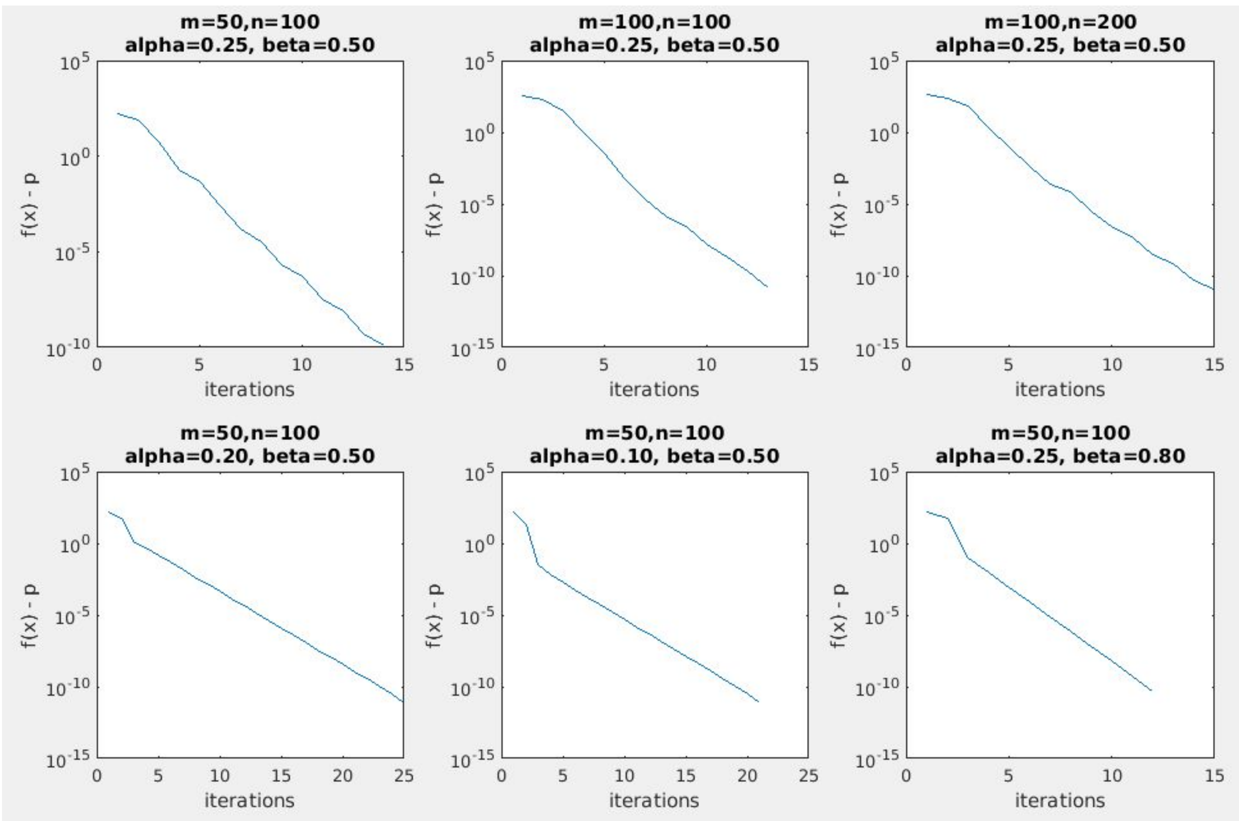
$$\Leftrightarrow (A, -A \cdot \vec{1}) \begin{pmatrix} z \\ t \end{pmatrix} = b - A \cdot \vec{1}$$

i.e., $A_1 = (A, -A \cdot \vec{1})$, $z_1 = \begin{pmatrix} z \\ t \end{pmatrix}$, $b_1 = b - A \cdot \vec{1}$

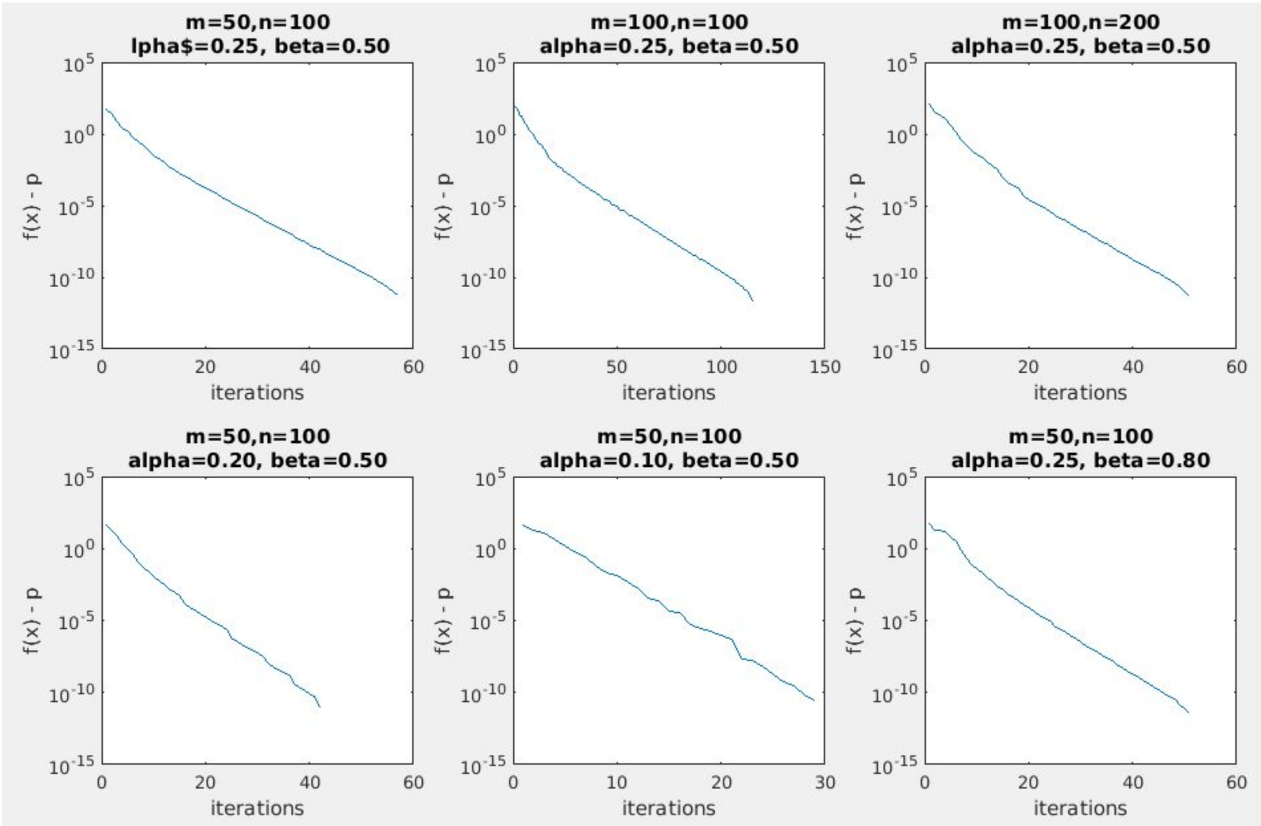
9.30

(a)

Different parameters for m, n, alpha and beta.



Same parameters as above, but re-generate a_i vectors.



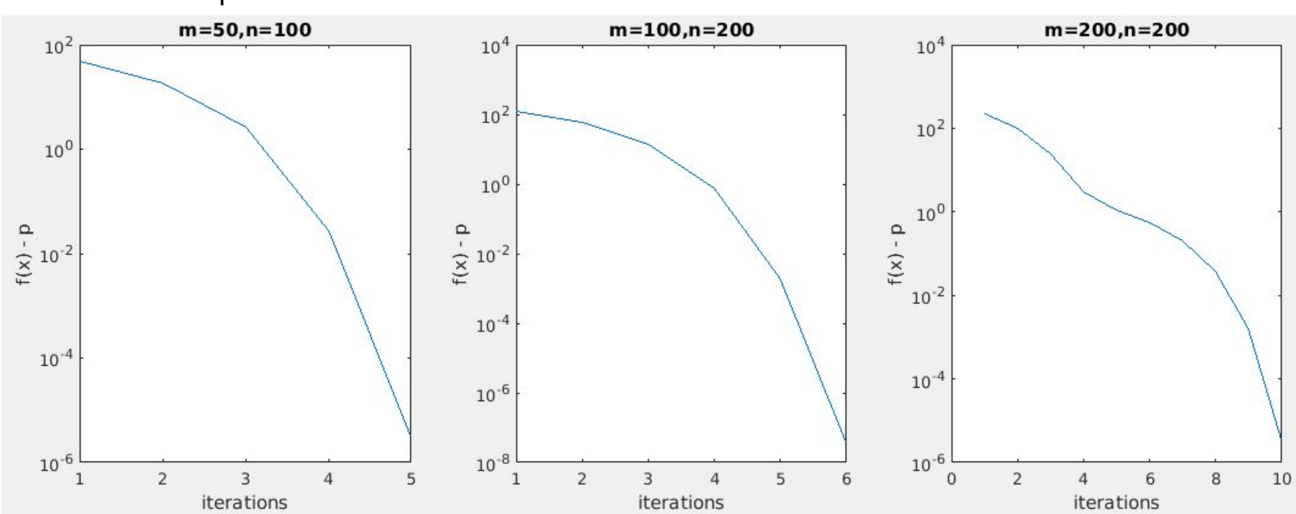
MATLAB function:

```
function [p, fx_all] = solve_HW6_9_30(m, n, alpha, beta)
% alpha = 0.25;
% beta = 0.5;
% m = 50;
% n = 100;
A = 2*rand(m, n) - ones(m, n);
tol = 1e-5;
n_maxiter = 500;
% initial x
x = zeros(n,1);
fx_all = NaN(1, n_maxiter);
x_all = NaN(n, n_maxiter);
ps = NaN(1, n_maxiter);
p = 1;
for ii = 1:n_maxiter
   fx = - sum(log(1-A*x)) - sum(log(1-x.^2));
   fx_all(ii) = fx;
   x_all(:,ii) = x;
   ps(ii) = p;
   df = A' * (1./(1 - A*x)) - 1./(1+x) + 1./(1-x);
```

```
    if norm(df) < tol
        p = fx;
        break;
    end
    if fx < p
        p = fx;
    end
    dx = -df;
    t = 1;
    while (max(A*(x + t*dx) ) >= 1) || (max(abs((x + t*dx))) >= 1)
        t = t * beta;
    end
    while (- sum(log(1-A*(x + t*dx))) - sum(log(1-(x + t*dx).^2)) ) >...
            (fx + alpha * t * df'*dx )
        t = t * beta;
    end
    x = x + t * dx;
end

end
```

(b)

solve the problem for different size:



MATLAB function:

```
function [p, fx_all] = solve_HW6_9_30_b(m, n)
% alpha = 0.25;
% beta = 0.5;
```

```
% m = 50;
% n = 100;
A = 2*rand(m, n) - ones(m, n);
tol = 1e-5;
n_maxiter = 500;
% initial x
x = zeros(n,1);
fx_all = NaN(1, n_maxiter);
x_all = NaN(n, n_maxiter);
ps = NaN(1, n_maxiter);
p = 1;
for ii = 1:n_maxiter
    fx = - sum(log(1-A*x)) - sum(log(1-x.^2));
    fx_all(ii) = fx;
    x_all(:,ii) = x;
    ps(ii) = p;
    df = A' * (1./(1 - A*x)) - 1./(1+x) + 1./(1-x);
    ddf = A' * diag((1./(1 - A*x)).^2) * A + diag(1./(1+x).^2 + 1./(1-x).^2);
    if sqrt(df'*(ddf\df)) < tol
        p = fx;
        break;
    end
    if fx < p
        p = fx;
    end

    dx = -ddf\df;
    x = x + dx;
end

end
```

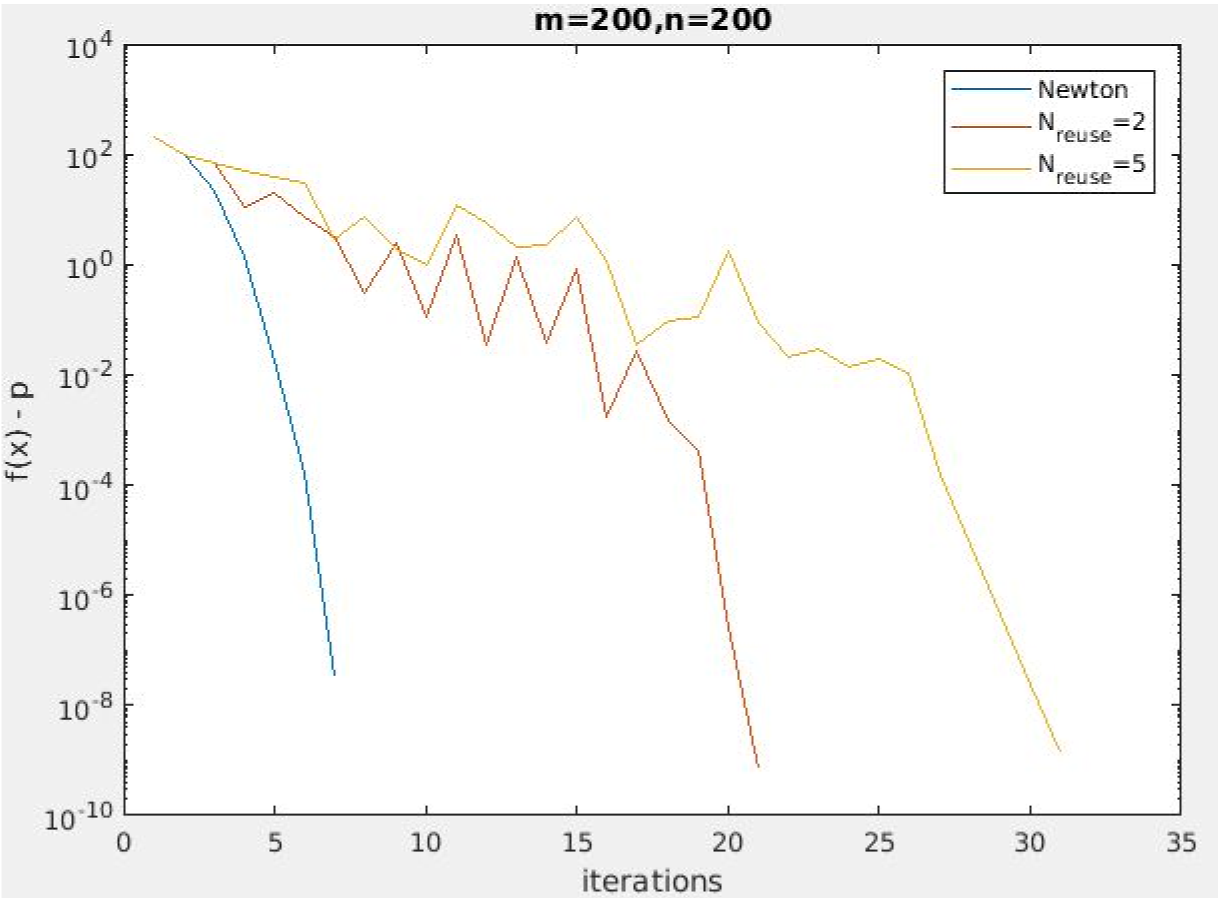
9.31

Code for generating the following plot:

```
%% 9.31
% (a)
```

```
A = 2*rand(m*4, n*2) - ones(m*4, n*2);
[p_1, fx_all_1] = solve_HW6_9_31(A, m*4, n*2, 1);
[p_2, fx_all_2] = solve_HW6_9_31(A, m*4, n*2, 2);
[p_5, fx_all_5] = solve_HW6_9_31(A, m*4, n*2, 5);

figure;
semilogy(fx_all_1-p_1)
hold all
semilogy(fx_all_2-p_2)
semilogy(fx_all_5-p_5)
xlabel('iterations')
ylabel('f(x) - p')
title(sprintf('m=%d,n=%d',m*4,n*2))
legend({'Newton', 'N_{reuse}=2', 'N_{reuse}=5'});


%% (b) compare diagonal approx and Newton

A = 2*rand(m*4, n*2) - ones(m*4, n*2);
[p_1, fx_all_newton] = solve_HW6_9_31(A, m*4, n*2, 1);
[p_2, fx_all_diagApprox] = solve_HW6_9_31_b(A, m*4, n*2);

figure;
semilogy(fx_all_newton-p_1)
hold all
semilogy(fx_all_diagApprox-p_2)
xlabel('iterations')
ylabel('f(x) - p')
title(sprintf('m=%d,n=%d',m*4,n*2))
legend({'Newton', 'Diag. approx.'});
```

(a)



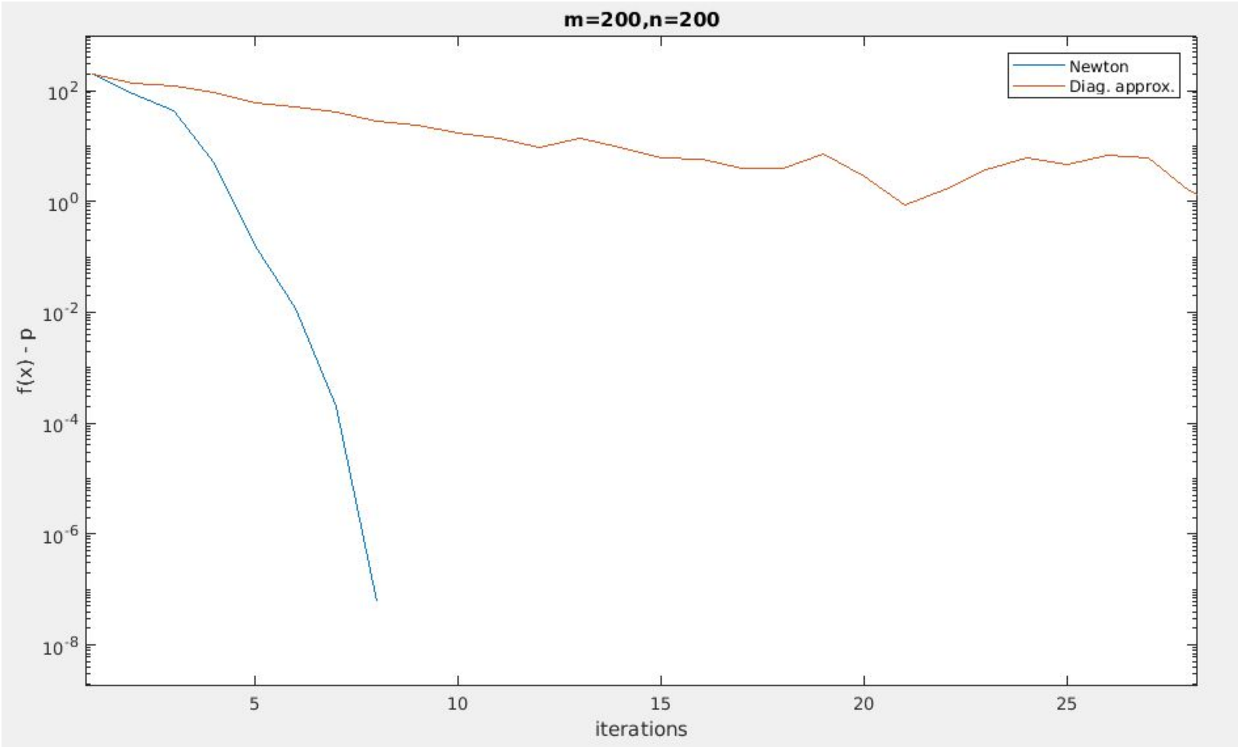
MATLAB code:

```
function [p, fx_all] = solve_HW6_9_31(A, m, n, N_use)
% alpha = 0.25;
beta = 0.5;
% m = 50;
% n = 100;
% A = 2*rand(m, n) - ones(m, n);
tol = 1e-5;
n_maxiter = 500;
% initial x
x = zeros(n,1);
fx_all = NaN(1, n_maxiter);
x_all = NaN(n, n_maxiter);
ps = NaN(1, n_maxiter);
p = 1;
n_used = 1000;
```

```
for ii = 1:n_maxiter
    fx = - sum(log(1-A*x)) - sum(log(1-x.^2));
    fx_all(ii) = fx;
    x_all(:,ii) = x;
    ps(ii) = p;
    df = A' * (1./(1 - A*x)) - 1./(1+x) + 1./(1-x);
    if n_used >= N_use
        ddf = A' * diag((1./(1 - A*x)).^2) * A + diag(1./(1+x).^2 + 1./(1-x).^2);
        n_used = 0;
    end
    if sqrt(df'*(ddf\df)) < tol
        p = fx;
        break;
    end
    if fx < p
        p = fx;
    end

    dx = -ddf\df;
    t = 1;
    while (max(A*(x + t*dx) ) >= 1) || (max(abs((x + t*dx))) >= 1)
        t = t * beta;
    end
    n_used = n_used + 1;
    x = x + t*dx;
end

end
```

(b)

Diagonal approximation converge linearly:



MATLAB code:

```
function [p, fx_all] = solve_HW6_9_31_b(A, m, n)
% alpha = 0.25;
beta = 0.5;
% m = 50;
% n = 100;
% A = 2*rand(m, n) - ones(m, n);
tol = 1e-5;
n_maxiter = 500;
% initial x
x = zeros(n,1);
fx_all = NaN(1, n_maxiter);
x_all = NaN(n, n_maxiter);
ps = NaN(1, n_maxiter);
p = 1;
for ii = 1:n_maxiter
    fx = - sum(log(1-A*x)) - sum(log(1-x.^2));
    fx_all(ii) = fx;
    x_all(:,ii) = x;
    ps(ii) = p;
    df = A' * (1./(1 - A*x)) - 1./(1+x) + 1./(1-x);
    ddf = A' * diag((1./(1 - A*x)).^2) * A + diag(1./(1+x).^2 + 1./(1-x).^2);
```

```
    ddf = diag(diag(ddf));
    if sqrt(df'*(ddf\df)) < tol
        p = fx;
        break;
    end
    if fx < p
        p = fx;
    end
    dx = -ddf\df;
    t = 1;
    while (max(A*(x + t*dx) ) >= 1) | (max(abs((x + t*dx))) >= 1)
        t = t * beta;
    end
    x = x + t*dx;
end
end
```

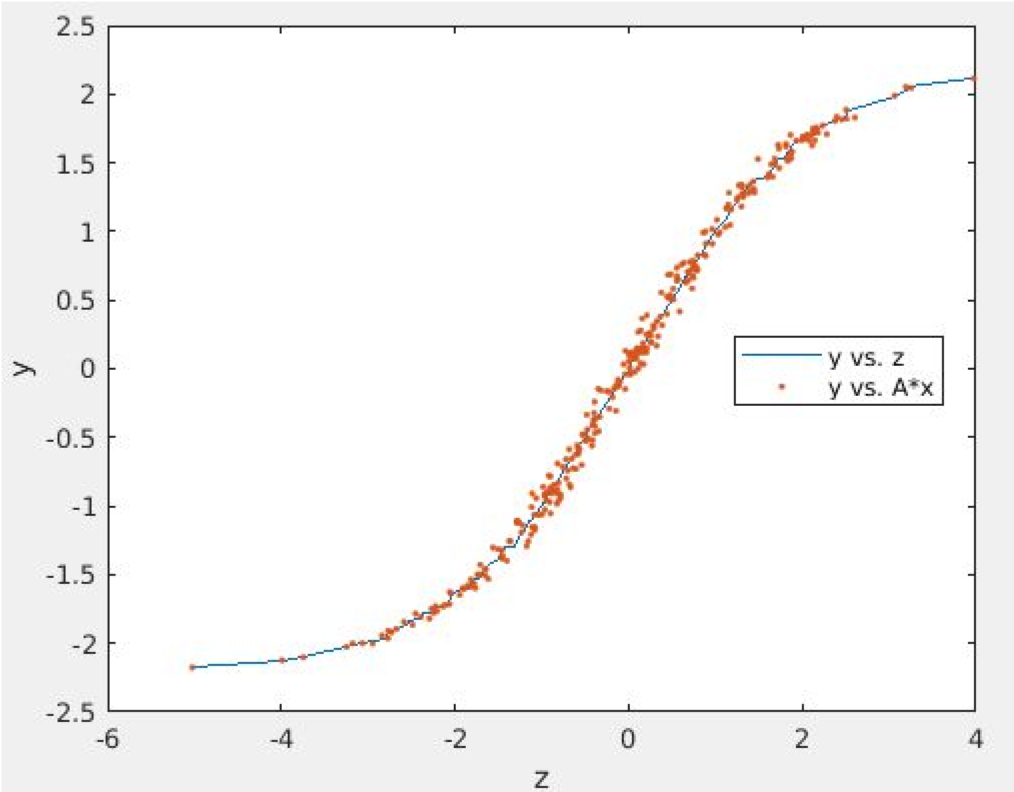
A6.5

(b)
Resulting x:

```
x =

    0.4819
   -0.4657
    0.9364
    0.9297
```

Plot of estimated function:

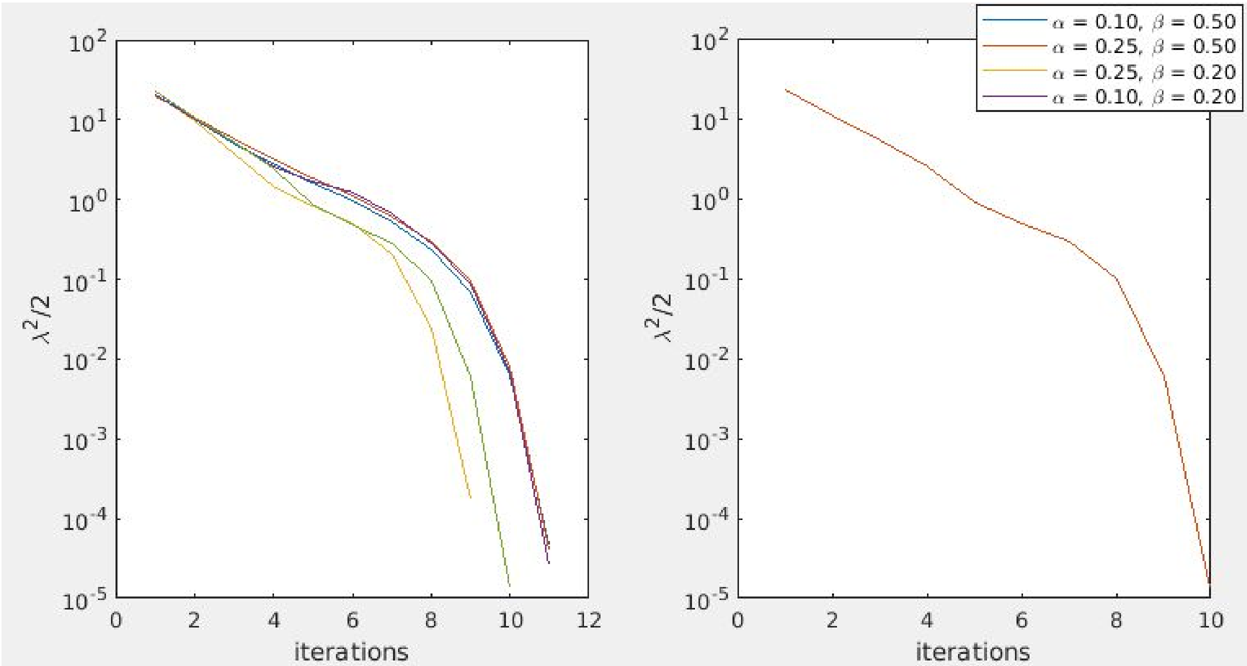
- Blue curve is y-z curve based on max-likelihood
- orange dots are raw data from max-likelihood estimation without the noise.



A9.5

(a)

- Left: different instances generated randomly
- right: same instance with different model parameters alpha and beta. They appear to be exactly the same because the code never enters the two while loop to use the two parameters.

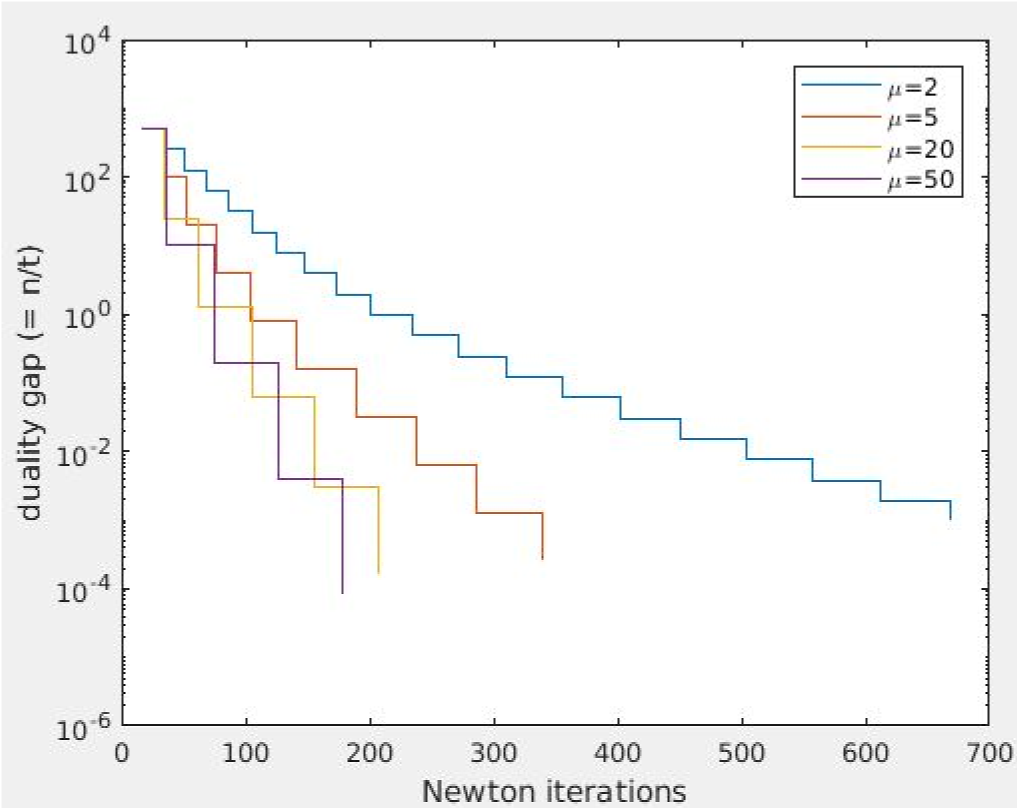


Check KKT condition:

-
- df = c - 1./x_opt;
- norm(df + A'*v_opt)

this gives norm(df + A'*v_opt) = 0.0021, which means x_opt and v_opt nearly satisfy the KKT condition.

(b) Plot of the progress:



Comparing with CVX result:

- disp(norm(x-x_opt)) = 6.4490e-04
- i.e, the same

(c)

Results:

- for infeasible instance, both methods returns infeasible (for the function I wrote, x_opt = NaN)
- for feasible problem, norm(x-x_opt) is typically <~ 0.01

Code used for comparison:

```
m = 100;
n = 500;
% this is likely infeasible
A = 2*rand(m, n) + 2*[diag(ones(1, m )), zeros(m, n-m)];
b = rand(m,1);
c = rand(n,1);

% manually generate a feasible instance:

A = 2*rand(m, n) + 2*[diag(ones(1, m )), zeros(m, n-m)];
```

```
x0 = rand(n, 1);
b = A*x0;
c = rand(n,1);

%% written solver:
x_opt = solve_HW6_LP_full(A,b,c);
%% cvx:
cvx_begin
variable x(n)
minimize(c'*x)
subject to
A*x == b;
x >= 0;
cvx_end

% compare
disp(norm(x-x_opt))
```

MATLAB codes

Scripts for generating all results:

```
%% A9.5
%% different instances

m = 200;
n = 300;

figure;
subplot(121)
for nn = 1:5
% generate data
A = 2*rand(m, n) + 2*[diag(ones(1, m )), zeros(m, n-m)];
x0 = rand(n, 1);
b = A*x0;
c = rand(n,1);

% solving
[x_opt, v_opt, N_steps, lambdasqs] = solve_HW6_LP_CS(A, b, c, x0);

% plotting
semilogy(1:N_steps, lambdasqs); hold all;
xlabel('iterations');
```

```
ylabel('\lambda^2/2');
end

%% try same instance with different alpha and beta
lgds = {};

subplot(122);
alf = 0.1;
beta = 0.5;
[x_opt, v_opt, N_steps, lambdasqs] = solve_HW6_LP_CS(A, b, c, x0, alf, beta);
semilogy(1:N_steps, lambdasqs); hold all;
lgds{end+1} = sprintf('\\alpha = %.2f, \\beta = %.2f', alf, beta);

alf = 0.25;
beta = 0.5;
[x_opt, v_opt, N_steps, lambdasqs] = solve_HW6_LP_CS(A, b, c, x0, alf, beta);
semilogy(1:N_steps, lambdasqs); hold all;
lgds{end+1} = sprintf('\\alpha = %.2f, \\beta = %.2f', alf, beta);

alf = 0.25;
beta = 0.2;
[x_opt, v_opt, N_steps, lambdasqs] = solve_HW6_LP_CS(A, b, c, x0, alf, beta);
semilogy(1:N_steps, lambdasqs); hold all;
lgds{end+1} = sprintf('\\alpha = %.2f, \\beta = %.2f', alf, beta);

alf = 0.1;
beta = 0.2;
[x_opt, v_opt, N_steps, lambdasqs] = solve_HW6_LP_CS(A, b, c, x0, alf, beta);
semilogy(1:N_steps, lambdasqs); hold all;
lgds{end+1} = sprintf('\\alpha = %.2f, \\beta = %.2f', alf, beta);

xlabel('iterations');
ylabel('\lambda^2/2');
legend(lgds);

% check KKT:
df = c - 1./x_opt;
norm(df + A'*v_opt)


%% (b), barrier

m = 100;
```

```
n = 500;

% generate data
A = 2*rand(m, n) + 2*[diag(ones(1, m )), zeros(m, n-m)];
x0 = rand(n, 1);
b = A*x0;
c = rand(n,1);
% solve
[x_opt, history] = solve_HW6_LP_barrier(A, b, c, x0);

%% plot
figure;
[xx, yy] = stairs(cumsum(history(1,:)),history(2,:));
semilogy(xx,yy);hold all;
lgds = {'\mu=2'};
for mu = [5, 20, 50]
    mu
[x_opt, history] = solve_HW6_LP_barrier(A, b, c, x0, mu);
[xx, yy] = stairs(cumsum(history(1,:)),history(2,:));
semilogy(xx,yy);hold all;
lgds{end+1} = sprintf('\\mu=%d',mu);
end
legend(lgds)
xlabel('Newton iterations');
ylabel('duality gap (= n/t)');

%% check with CVX:
cvx_begin
variable x(n)
minimize(c'*x)
subject to
A*x == b;
x >= 0;
cvx_end

disp(norm(x-x_opt))

%% (c), full LP solver:

m = 100;
n = 500;
% this is likely infeasible
A = 2*rand(m, n) + 2*[diag(ones(1, m )), zeros(m, n-m)];
```

```
b = rand(m,1);
c = rand(n,1);

% manually generate a feasible instance:

A = 2*rand(m, n) + 2*[diag(ones(1, m )), zeros(m, n-m)];
x0 = rand(n, 1);
b = A*x0;
c = rand(n,1);

%% written solver:
x_opt = solve_HW6_LP_full(A,b,c);
%% cvx:
cvx_begin
variable x(n)
minimize(c'*x)
subject to
A*x == b;
x >= 0;
cvx_end

% compare
disp(norm(x-x_opt))
```

function for (a)

```
function [x_opt, v_opt, N_steps, lambdasqs] = solve_HW6_LP_CS(A, b, c, x0, alpha, beta)
% Solve LP with centering step
% Example data:
%
%     m = 200;
%     n = 300;
%     A = 2*rand(m, n) + 2*[diag(ones(1, m )), zeros(m, n-m)];
%     x0 = rand(n, 1);
%     b = A*x0;
%     c = rand(n,1);
%
% WTJ, 20180810
```

```
x = x0;
n_iter = 500;
if nargin < 6
beta = 0.5;
end
if nargin < 5
alpha = 0.25;
end
x_opt = NaN;
v_opt = NaN;
lambdasqs = [];
N_steps = 0;
tol = 1e-3;

if (min(x0) <= 0 || norm(A*x0 - b) > tol)
    return;
end

for ii = 1:n_iter
%     X = diag(1./(x.^2));
    N_steps = ii;
    X_inv = diag(x.^2);
    df = c - 1./x;
    B = -df;
    % w from block elimination
    w = (A*X_inv * A')\(A*X_inv*B);
    dx_nt = X_inv * (B - A' * w);
    lambdasq = -dx_nt' * df;
    lambdasqs(ii) = lambdasq/2;
    if lambdasq < tol
        x_opt = x;
        v_opt = w;
        return;
    end
    % line search
    t = 1;
    while min(x+t*dx_nt) <= 0
        t = beta * t;
    end

    while get_fx(x + t*dx_nt) >= get_fx(x) + alpha * t * (df' * dx_nt)
        t = beta * t;
    end
```

```
    x = x + t*dx_nt;
end

function f = get_fx(x)
    f = c'*x - sum(log(x));
end

end
```

function for (b)

```
function [x_opt, history] = solve_HW6_LP_barrier(A, b, c, x0, mu)
% Solving LP with eq constraint using barrier
%
% WTJ, 20180810
t = 1;
if nargin < 5
mu = 2;
end
n_iter = 500;
x_opt = [];
history = NaN(2, n_iter);
tol_dualgap = 1e-3;
n = length(x0);
for ii = 1:n_iter
    g = n/t;
    [x, v_opt, N_steps, lambdasqs] = solve_HW6_LP_CS(A, b, c*t, x0);
    history(1, ii) = N_steps;
    history(2, ii) = g;
    if g < tol_dualgap
        x_opt = x;
        return
    end
    t = t * mu;
end
end
```

function for (c)

```
function [x_opt] = solve_HW6_LP_full(A,b,c)
% Full LP solver
%
% WTJ, 20180810

[m, n] = size(A);
x_opt = NaN;

%% phase I
isfeasible = false;
x0 = A\b;
x0_min = min(x0);
if x0_min > 0
    isfeasible = true;
else
    t0 = 2 - x0_min;
    z0 = x0 + (t0-1)*ones(n, 1);
    c1 = [zeros(n, 1); 1];
    A1 = [A, -A*ones(n,1)];
    b1 = b - A*ones(n,1);
    z1 = [z0; t0];
    [z_opt] = solve_HW6_LP_barrier(A1, b1, c1, z1);
    if z_opt(end) < 1
        isfeasible = true;
    end
end

if ~ isfeasible
    return;
end

%% phase II
t = z_opt(end);
x0 = z_opt(1:(end-1)) - (t-1)*ones(n,1);
[x_opt] = solve_HW6_LP_barrier(A, b, c, x0);

end
```