zonghao li 1003843146

Assignment 4

Tuesday, March 24, 2020 8:25 PM

5.8].

<a> $f_0(X) = tr(x^T W x) = tr(x x^T W)$

let: $X = x x^T$, $rank(X) = 1$, $X \geq 0$

$x_i^2 = 1 \rightarrow X_{ii} = 1$

$\therefore$ minimize $tr(XW)$

subject to $X \geq 0$, $rank(X) = 1$ (neglected), $X_{ii} = 1 \quad i = 1, \dots n$ } <a>

<b> since we relax the constraint $rank(X) = 1$, we therefore enlarge the feasible set, therefore it gives a lower bound compared to the original question. If indeed $rank(X^*) = 1$, then this optimal value is the same as the optimal value from the original question

<c> For <a>:

$$L(x, v, z) = tr(WX) + v^T \begin{bmatrix} x_{11} - 1 \\ x_{22} - 1 \\ \vdots \\ x_{ii} - 1 \end{bmatrix} - z^T x$$

$$= tr(WX) + v^T \begin{bmatrix} x_{11} \\ x_{22} \\ \vdots \\ x_{ii} \end{bmatrix} - v^T \mathbf{1} - tr(xz^T)$$

$$= tr(WX) + tr(X\, diag(v)) - tr(xz^T) - v^T \mathbf{1}$$

$$= tr(WX + X\, diag(v) - xz^T) - v^T \mathbf{1}$$

$$= tr(X(W + diag(v) - zz^T)) - v^T \mathbf{1}$$

$$g(v, z) = \begin{cases} -v^T \mathbf{1} & W + diag(v) = z^T \quad (z \text{ is } S_+^n) \\ -\infty & \text{else} \end{cases}$$

$\therefore$ dual problem: minimize $-v^T \mathbf{1}$, subject to $W + diag(v) = z^T \geq 0$ } <d>

Therefore, for (5.119) in textbook, it is the same as <b>, so both (5.117) & (5.118) give the same lower bound to the prime problem

---

11.23

	SDP Bound	Optimum	11.23(b)	11.23(c)	11.23(d-a)	11.23(d-b)	11.23(d-c)
small	5.334	5.334	5.334	5.334	5.334	5.334	5.334
medium	-42.23	-38.37	-33.42	-38.37	-27.459	-38.37	-38.37
large	66.086	×	679.61	462.18	604.637	344.734	314.095

We see that for small size of W, all methods have very good effectiveness in terms of finding the minimum value. When the size of W is medium, the randomized method in (c) and greedy heuristic refinement in (db) and (dc) have similar effectiveness, and the eigenvector heuristic approach in (b) is decent, and greedy heuristic refinement in (da) with initial vector x = 1 has the worst effectiveness. When the size of W is large, greedy heuristic refinement approach improves the effectiveness from (da) to (dc). Comparing (b) and (c), the randomized method in (c) yields a better result than the eigenvector heuristic method in (b).
From this assignment, W is not PSD and the constraint is discrete, therefore the original question is not convex, therefore by solving the dual (SDP) of the original problem, it gives a lower bound of the original

```
close all
clear all

%% 11.23(b) %%
load('hw4data.mat');
%W = W5;   % small case
W = W10;  % medium case
%W = W50;  % large case
[n,n] = size(W);
cvx_begin sdp
    variable X(n,n) symmetric
    minimize ( trace(W*X) )
    subject to
        diag(X) == 1;
        X >= 0;
cvx_end

% SDP lower bound d_prime
opt_value = trace(W*X)

% V is the eigenvector, diag(D) is the eigenvalue
[V,D] = eig(X);
% return the max eigenvalue and its index
[val,idx] = max(diag(D));
% return the eigenvector that corresponds to the max eigenvalue
v = V(:,idx);

x_hat = sign(v);
% heuristic value
heuristic_b = x_hat.'*W*x_hat
% compute the difference between heuristic and SDP lower bound d*
difference = heuristic_b - opt_value;
% small case: -2.2013e-07
% medium case: 8.8061
% large case: 613.5235

%% 11.23(c) %%
% generate randomized sample x with zero mean and X as the covariance
K = 100;
mu = zeros(n,1);
sigma = X;
rng('default')  % For reproducibility
x = mvnrnd(mu,sigma,K); % K rows, n column, so covariance is n*n
x_hat = sign(x);
for i = 1:1:K
    heuristic_c(i,:) = x_hat(i,:)*W*x_hat(i,:)';
end
min(heuristic_c)

%% 11.23(da) %%
% initialize vector x
x = ones(n,1);
iteration = 20;   % iteration
heuristic_da = zeros(iteration,1);
for i = 1:1:iteration
    heuristic_da(:,) = x.'*W*x;
    for j = 1:1:n
        x(j,) = -x(j,);
        buffer(j,) = x.'*W*x; % buffer of W to see which one returns the minimum objective value
        x(j,) = -x(j,);    % flip it back before the next run
    end
    [val,idx] = min(buffer);  % return the minimum value from buffer value and the corrresponding index
    x(idx,) = -x(idx,);
end
min(heuristic_da)

%% 11.23(db) %%
% initialize vector x with uniform distribution (-1,1)
K = 100;
x = -2*round(rand(K,n))+ones(K,n);
iteration = 20;   % iteration
heuristic_db = zeros(iteration,1);
for k = 1:1:K
    for i = 1:1:iteration
        heuristic_db(i,k) = x(k,:)*W*x(k,:)';
        for j = 1:1:n
            x(k,j) = -x(k,j);
            buffer(j,) = x(k,:)*W*x(k,:)'; % buffer of W to see which one returns the minimum objective
value
            x(k,j) = -x(k,j);    % flip it back before the next run
        end
        [val,idx] = min(buffer);  % return the minimum value from buffer value and the corrresponding
index
        x(k,idx) = -x(k,idx);
    end
```

```
end
[val,idx] = min(min(heuristic_db))
x_best = x(idx,:)

%% 11.23(dc) %%
% initialize vector x
K = 100;
mu = zeros(n,1);
sigma = X;
rng('default')  % For reproducibility
x = mvnrnd(mu,sigma,K); % K rows, n column, so covariance is n*n
x = sign(x);
iteration = 20;    % iteration
heuristic_dc = zeros(iteration,1);
for k = 1:1:K
    for i = 1:1:iteration
        heuristic_dc(i,k) = x(k,:)*W*x(k,:)';
        for j = 1:1:n
            x(k,j) = -x(k,j);
            buffer(j,:) = x(k,:)*W*x(k,:)';  % buffer of W to see which one returns the minimum objective
value
            x(k,j) = -x(k,j);    % flip it back before the next run
        end
        [val,idx] = min(buffer);  % return the minimum value from buffer value and the corresponding
index
        x(k,idx) = -x(k,idx);
    end
end
[val,idx] = min(min(heuristic_dc))
x_best = x(idx,:)
```

<a> let: $f(x) = \frac{x}{c-x}$

$$\nabla_x f(x) = [x(c-x)^{-1}]'$$
$$= (c-x)^{-1} + x(-1)(c-x)^{-2}(-1)$$
$$= \frac{1}{c-x} + x\frac{1}{(c-x)^2} = \frac{c}{(c-x)^2}$$

$$\nabla_x^2 f(x) = [c(c-x)^{-2}]'$$
$$= c(-2)(c-x)^{-3}(-1)$$
$$= \frac{2c}{(c-x)^3}$$

since: $c_i \geq x_i \geq 0$ for $\forall i$,

$\therefore \nabla_x^2 f(x) \geq 0$

$\therefore f(x)$ is convex. And the sum of convex function is also convex. Since all inequality constraints are convex, equality constraints are affine. Therefore, the problem is a convex opt. problem

<b> We first use log-barrier to get rid of the inequality constraints

$$F(x) = \sum_{i=1}^{13} \frac{x_i}{c_i - x_i} + \left[-\left(\frac{1}{t}\right)\log\left[-(x_4 + x_6 - 1)\right]\right]$$
$$+ \left[-\left(\frac{1}{t}\right)\log\left[-(x_5 + x_8 - 1)\right]\right]$$
$$+ \left[-\left(\frac{1}{t}\right)\log\left[-(x_9 + x_{10} + x_{12} - 1)\right]\right]$$
$$+ \left[-\left(\frac{1}{t}\right)\log(x_1)\right] + \left[-\left(\frac{1}{t}\right)\log(x_2)\right] + \cdots + \left[-\left(\frac{1}{t}\right)\log(x_{13})\right]$$
$$+ \left[-\left(\frac{1}{t}\right)\log\left[-(x_1 - 1)\right]\right] + \cdots + \left[-\left(\frac{1}{t}\right)\log\left[-(x_{13} - 1)\right]\right]$$

$$= \sum_{i=1}^{13} \frac{x_i}{c_i - x_i} + \left(-\frac{1}{t}\right)\log\left[(-1)^3 \prod_{i=1}^{3} \hat{b_i}\right] \quad \left(\text{where } Bx - b = \hat{B} = \begin{bmatrix} \hat{b_1} \\ \hat{b_2} \\ \hat{b_3} \end{bmatrix}\right)$$
$$+ \left(-\frac{1}{t}\right)\log\left[\prod_{i=1}^{13} x_i\right]$$
$$+ \left(-\frac{1}{t}\right)\log\left[(-1)^{13} \prod_{i=1}^{13} (x_i - 1)\right]$$

$\therefore \min_x F(x)$

s.t. $A^T x = s$

Apply infeasible starting point Newton's method:

$$\begin{bmatrix} \nabla^2 F(x) & A^T \\ A^T & 0 \end{bmatrix} \begin{bmatrix} \Delta x_{nt} \\ \Delta v_{nt} \end{bmatrix} = -\begin{bmatrix} \nabla F(x) + A^T v \\ A^T x - s \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} \Delta x_{nt} \\ \Delta v_{nt} \end{bmatrix} = -\begin{bmatrix} \nabla^2 F(x) & A^T \\ A^T & 0 \end{bmatrix}^{-1} \begin{bmatrix} \nabla F(x) + A^T v \\ A^T x - s \end{bmatrix}$$

- $r_{dual} = \nabla F(x) + A^T v$
- $r_{primal} = A^T x - s$
- $\nabla F(x) =$

$$\begin{bmatrix} x1/(x1-1)^2 - 1/(x1-1) - 1/(x1*t)) - 1/(con(t)*(x1-1)) \end{bmatrix}$$

- $r_{primal} = A' x - S$

- $\nabla F(x) =$

```
[ x1/(x1 - 1)^2 - 1/(x1 - 1) - 1/(x1*(t)) - 1/(conj(t)*(x1 - 1))
  x2/(x2 - 1)^2 - 1/(x2 - 1) - 1/(x2*conj(t)) - 1/(conj(t)*(x2 - 1))
  x3/(x3 - 1)^2 - 1/(x3 - 1) - 1/(x3*conj(t)) - 1/(conj(t)*(x3 - 1))
  x4/(x4 - 1)^2 - 1/(x4 - 1) - 1/(conj(t)*(x4 + x6 - 1)) - 1/(x4*conj(t)) - 1/(conj(t)*(x4 - 1))
  x5/(x5 - 1)^2 - 1/(x5 - 1) - 1/(conj(t)*(x5 + x8 - 1)) - 1/(x5*conj(t)) - 1/(conj(t)*(x5 - 1))
  x6/(x6 - 1)^2 - 1/(x6 - 1) - 1/(conj(t)*(x4 + x6 - 1)) - 1/(x6*conj(t)) - 1/(conj(t)*(x6 - 1))
  x7/(x7 - 1)^2 - 1/(x7 - 1) - 1/(x7*conj(t)) - 1/(conj(t)*(x7 - 1))
  x8/(x8 - 1)^2 - 1/(x8 - 1) - 1/(conj(t)*(x5 + x8 - 1)) - 1/(x8*conj(t)) - 1/(conj(t)*(x8 - 1))
  x9/(x9 - 1)^2 - 1/(x9 - 1) - 1/(x9*conj(t)) - 1/(conj(t)*(x9 + x10 + x12 - 1)) - 1/(conj(t)*(x9 - 1))
  x10/(x10 - 1)^2 - 1/(x10 - 1) - 1/(x10*conj(t)) - 1/(conj(t)*(x9 + x10 + x12 - 1)) - 1/(conj(t)*(x10 - 1))
  x11/(x11 - 1)^2 - 1/(x11 - 1) - 1/(x11*conj(t)) - 1/(conj(t)*(x11 - 1))
  x12/(x12 - 1)^2 - 1/(x12 - 1) - 1/(x12*conj(t)) - 1/(conj(t)*(x9 + x10 + x12 - 1)) - 1/(conj(t)*(x12 - 1))
  x13/(x13 - 1)^2 - 1/(x13 - 1) - 1/(x13*conj(t)) - 1/(conj(t)*(x13 - 1)) ]
```

- $\nabla^2 F(x) =$

```
[ 1/(t*conj(x1)^2) + 2/(conj(x1) - 1)^2 + 1/(t*(conj(x1) - 1)^2) - (2*conj(x1))/(conj(x1) - 1)^3, 0, 0, 0, 0, 0, 0, 0,
[ 0, 1/(t*conj(x2)^2) + 2/(conj(x2) - 1)^2 + 1/(t*(conj(x2) - 1)^2) - (2*conj(x2))/(conj(x2) - 1)^3, 0, 0, 0, 0, 0, 0,
[ 0, 0, 1/(t*conj(x3)^2) + 2/(conj(x3) - 1)^2 + 1/(t*(conj(x3) - 1)^2) - (2*conj(x3))/(conj(x3) - 1)^3, 0, 0, 0, 0, 0,
[ 0, 0, 0, 1/(t*conj(x4)^2) + 2/(conj(x4) - 1)^2 + 1/(t*(conj(x4) + conj(x6) - 1)^2) + 1/(t*(conj(x4) - 1)^2) - (2*conj(x4))/(conj(x4) - 1)^3, 0, 1/(t*(conj(x4) + conj(x6) - 1)^2), 0,
[ 0, 0, 0, 0, 1/(t*conj(x5)^2) + 2/(conj(x5) - 1)^2 + 1/(t*(conj(x5) + conj(x8) - 1)^2) + 1/(t*(conj(x5) - 1)^2) - (2*conj(x5))/(conj(x5) - 1)^3, 0, 0,
[ 0, 0, 0, 1/(t*(conj(x4) + conj(x6) - 1)^2), 0, 1/(t*conj(x6)^2) + 2/(conj(x6) - 1)^2 + 1/(t*(conj(x4) + conj(x6) - 1)^2) + 1/(t*(conj(x6) - 1)^2) - (2*conj(x6))/(conj(x6) - 1)^3, 0,
[ 0, 0, 0, 0, 0, 0, 1/(t*conj(x7)^2) + 2/(conj(x7) - 1)^2 + 1/(t*(conj(x7) - 1)^2) - (2*conj(x7))/(conj(x7) - 1)^3,
[ 0, 0, 0, 0, 1/(t*(conj(x5) + conj(x8) - 1)^2), 0, 0, 1/(t*conj(x8)^2) + 2/(conj(x8) - 1)^2 + 1/(t*(conj(x5) + conj(x8) - 1)^2) +
[ 0, 0, 0, 0, 0, 0, 0, 0, 1/(t*con
[ 0, 0, 0, 0, 0, 0, 0, 0,
[ 0, 0, 0, 0, 0, 0, 0, 0,
[ 0, 0, 0, 0, 0, 0, 0, 0,
[ 0, 0, 0, 0, 0, 0, 0, 0,
```

(Matlab by default treat variable as the complex number; since all variable in this question are real, "conj" does not do anything)

- Update equations and rules:

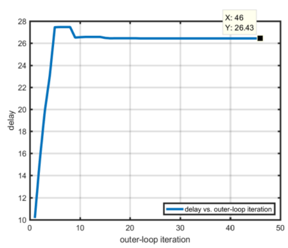
Algorithm 10.2 *Infeasible start Newton method.*

given starting point $x \in \mathbf{dom}\, f$, ν, tolerance $\epsilon > 0$, $\alpha \in (0, 1/2)$, $\beta \in (0, 1)$.
repeat
1. Compute primal and dual Newton steps Δx_{nt}, $\Delta \nu_{nt}$.
2. *Backtracking line search on* $\|r\|_2$.
   $t := 1$.
   while $\|r(x + t\Delta x_{nt}, \nu + t\Delta \nu_{nt})\|_2 > (1 - \alpha t)\|r(x, \nu)\|_2$, $\quad t := \beta t$.
3. *Update.* $x := x + t\Delta x_{nt}$, $\nu := \nu + t\Delta \nu_{nt}$. , also $t_{barrier} = \mu t_{barrier}$

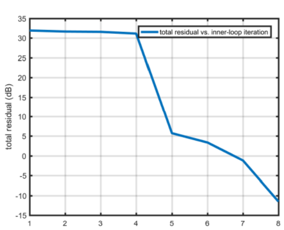
until $Ax = b$ and $\|r(x, \nu)\|_2 \le \epsilon$.

- minimum delay $= \sum_{i=1}^{13} \frac{x_i^*}{C_i - x_i^*} = 26.43$

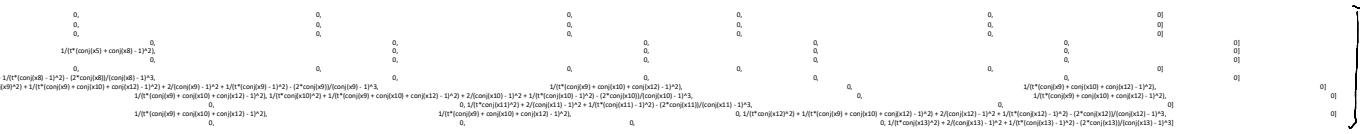
<c>



<d>



The plots above also relevant to the initilization of x. For example, picking x = 0.6 * ones (L,1) takes way more iteration than x = 0.3*ones(L,1). Also, to avoid denominator be zero, x ≠ 0.5 * ones (L,1). In this question, it seems initilizing x < 0.5 * ones (L,1) gives faster convergence. Noticing delay at the begining is very small, it is because we start with infeasible point. The delay increases quick, meaning that the program starts to pickup feasible point; then the delay starts to decrease.



```
close all
clear all

digitsOld = digits(7);

L = 13;
N = 9;
c = ones(L,1); % capacity
s = [1.2;0.6;0.6;zeros(8-3-1,1)];
A = [
    1  1  1  0  0  0  0  0  0  0  0  0  0;
   -1  0  0  1  0  1  0  0  0  0  0  0  0;
    0  0  -1  0  1  0  0  1  0  0  0  0  0;
    0  -1  0  -1  -1  0  1  0  0  0  0  0  0;
    0  0  0  0  0  0  -1  0  1  1  0  1  0;
    0  0  0  0  0  0  0  -1  0  0  -1  0  0;
    0  0  0  0  0  0  0  0  -1  0  0  0  1;
    0  0  0  0  0  0  0  0  0  0  0  -1  -1  -1;
    ];

A_plus = A(1:end-1,:);

B = [
    0  0  0  1  0  1  0  0  0  0  0  0  0;
    0  0  0  0  1  0  0  1  0  0  0  0  0;
    0  0  0  0  0  0  0  0  1  1  0  1  0;
    ];
b = ones(3,1);

syms x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 t
x = [x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13]';   % the input are real anyway, so just to remove conj in
front of each x
f0 = 0;
for i = 1:1:L
    f0 = f0 + x(i,:)./(1-x(i,:));
end

B_hat = B*x - b;

f1 = (-1/t) * log(-1*(size(B,1))*prod(B_hat));
f2 = (-1/t) * log(prod(x));
f3 = (-1/t) * log((-1)^L*prod(x-1));
f = f0 + f1 + f2 + f3;

m = size(b,1) + size(x,1) + size(c, 1);

H = hessian(f,[x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13]);   % Hessian of f
J = jacobian(f,[x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13])';  % Jacobian of f

alpha = 0.01;%0.5*rand;     % alpha in the range of (0,0.5)
beta = 0.5;%rand;           % beta in the range of (0,1)

%% Infeasible start Newton method with log-barrier %%
% tolerance for Newton ietration
tol_int = 1e-7;

% tolerance for log-barrier interior point iteration, controlling the
% duality gap
tol_int = 1e-4;

% tolerance for residual
tol_res = 1e-4;

% for setting the accuracy of log-barrier
t_barrier = 1;
mu = 10;   % used for update t in each iteration

% initialize x
x = 0.3*ones(L,1);  % avoid 0.5 since there will be zero in the denominator

% initialize v
v = zeros(size(A_plus',2),1);

% define how many iterations one wants
iteration = 200;

% more initilizations...for the plot
t_int = zeros(iteration,1);              % t for log-barrier in each iteration
delay = zeros(iteration,1);              % delay in each iteration
residual = zeros(iteration,1);           % total residual in eaech iteration
counter_inner = 0;                       % counter for total amount of inner loop iteration
counter_outer = 0;                       % counter for total amount of outer loop iteration

for k=1:1:iteration

% https://www.mathworks.com/help/symbolic/subs.html
% symbolic substitution for J and H
% initialize Jacobian and Hessian

    Jacobian = subs(J,[x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 t],[x' t_barrier]);
    Hessian = subs(H,[x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 t],[x' t_barrier]);

    % old dual residual
    r_dual1 = Jacobian + A_plus' * v;

    % old primal residual
    r_primal1 = A_plus * x - s;

    % calculate del_v
    %w = pinv(A_plus') * (-Jacobian-Hessian*del_x);
    %w = vpa(-inv([A_plus/(Hessian)*A_plus'])*A_plus*inv(Hessian)*Jacobian);
    %del_v = w - v;

    % calculate del_x
    %del_x = pinv(A_plus) * (s - A_plus*x);
    %del_x = vpa((Hessian)\(-A_plus'*w-Jacobian));

    % calculate r

    sol = - [Hessian A_plus'; A_plus zeros(size(v,1),size(v,1))] \ [r_dual1; r_primal1];

    del_x = sol(1:L);

    del_v = sol(L+1:end);

    if max(A_plus'*x-s)<tol_int && norm([r_dual1, r_primal1], 2) < tol_res
        gap = m/t_barrier;
        if (gap < tol_int); break; end;
        t_barrier = mu * t_barrier;
        t_int(k,:) = t_barrier;
    else
        % intialize step size for Newton's method
        t_newton = 1;

        % update variable
        newx = x + t_newton*del_x;
        newv = v + t_newton*del_v;

        while (min([newx; c - newx])<0)
            t_newton = beta * t_newton
            % update variable
            newx = x + t_newton*del_x;
            newv = v + t_newton*del_v;
        end

        % update Jacobian and Hessian
        Jacobian = subs(J,[x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 t],[newx' t_barrier]);
        Hessian = subs(H,[x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 t],[newx' t_barrier]);

        % new dual residual
        r_dual2 = vpa(Jacobian + A_plus' * newv);

        % new primal residual
        r_primal2 = vpa(A_plus * newx - s);

        % inner iteration
        % counter = 0;
        while (norm([r_dual2, r_primal2], 2) > (1 - alpha * t_newton) * norm([r_dual1, r_primal1],2))
            t_newton = beta * t_newton
            % update variable
            newx = x + t_newton*del_x;
            newv = v + t_newton*del_v;
            Jacobian = subs(J,[x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 t],[newx' t_barrier]);
            Hessian = subs(H,[x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 t],[newx' t_barrier]);

            % new dual residual
            r_dual2 = vpa(Jacobian + A_plus' * newv);

            % new primal residual
            r_primal2 = vpa(A_plus * newx - s);

            % counter to count how many iteration it takes
            counter_inner = counter_inner + 1;

            % calculate total residual
            residual(counter_inner_) = norm([r_dual2, r_primal2], 2) + m/t_barrier;
        end
        % update variable
        x = vpa(x + t_newton*del_x)
        v = vpa(v + t_newton*del_v);
    end

    % delay
    delay(k,:) = subs(f0,[x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 t],[x' t_barrier]);

    %increment outer loop counter
    counter_outer = counter_outer + 1;
end

figure(1)
p1 = plot(delay(1:counter_outer,:));
set(gca,'linewidth',2)
set(p1, 'linewidth',3)
xlabel('outer-loop iteration')
ylabel('delay')
legend('delay vs. outer-loop iteration')
grid on

figure(2)
p2 = plot(log10(residual(1:counter_inner,:)));
set(gca,'linewidth',2)
set(p2, 'linewidth',3)
xlabel('inner-loop iteration')
ylabel('total residual')
legend('total residual vs. inner-loop iteration')
grid on
```