Math 404 Report2
Interior Point Methods for Linear Programming

Abdelateef Khaled Abdelateef 202001344

Department of Engineering, University of Science and Technology at Zewail City, Egypt

Contents

1 Example 1:

$$
\begin{aligned}
\text{Max}\, Z = 2x_1 + 3x_2 \\
\text{s.t.} \quad 2x_1 + x_2 \leq 4 \\
x_1 + 2x_2 \leq 5 \\
x_1, x_2 \geq 0
\end{aligned}
$$

The standard LP form $(Z = -z)$:

$$
\begin{aligned}
&\text{Min} \quad z = -2x_1 - 3x_2 \\
&\qquad\qquad 2x_1 + x_2 + x_3 \qquad = 4 \\
&\text{s.t.} \qquad\quad x_1 + 2x_2 + \qquad x_4 = 5 \quad \text{(where } x_3 \text{ and } x_4 \text{ are slack)} \\
&\quad x_1, x_2, x_3, x_4 \geq 0
\end{aligned}
$$

Hence, $n = 4$, $m = 2$.

$$
\begin{aligned}
X &= \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix} \\
C &= \begin{bmatrix} -2 & -3 & 0 & 0 \end{bmatrix} \\
A &= \begin{bmatrix} 2 & 1 & 1 & 0 \\ 1 & 2 & 0 & 1 \end{bmatrix} \\
\boldsymbol{b} &= \begin{bmatrix} 4 \\ 5 \end{bmatrix}
\end{aligned}
$$

Used $\sigma = 0.3$, $\alpha = 0.8$, and the stopping value be $\text{Tol} = 0.01$ for all method

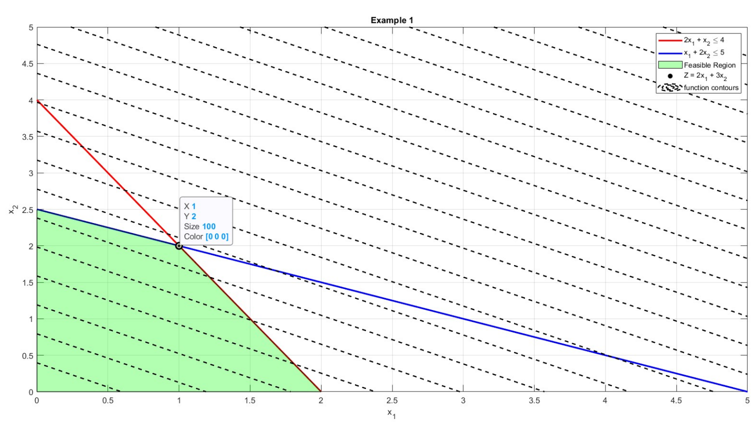

Figure 1: The Graphical Representation of Example 1

1.1 Central Path with fixed step size

The Fixed Central Path method takes 8 iterations with ($\alpha = 0.8$, $\sigma = 0.3$)

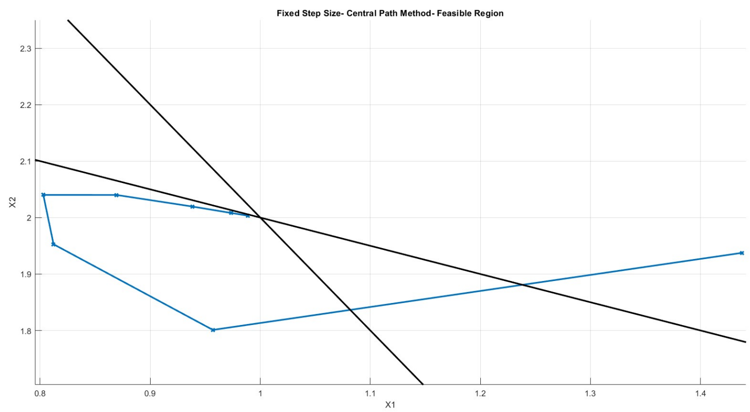

Figure 2: Central Path Method- Feasible Region

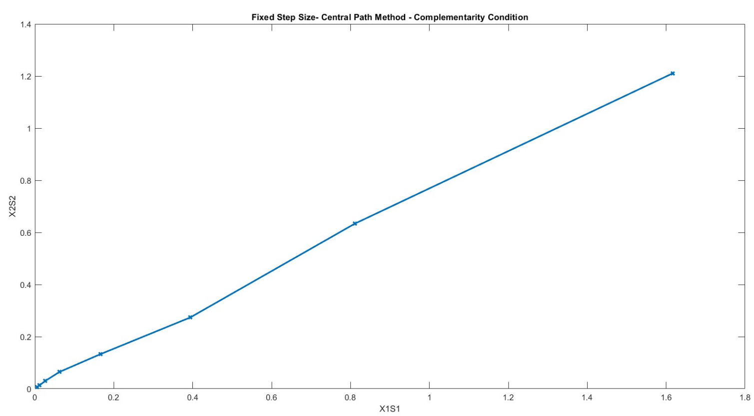

Figure 3: Central Path Method - Complementarity Condition

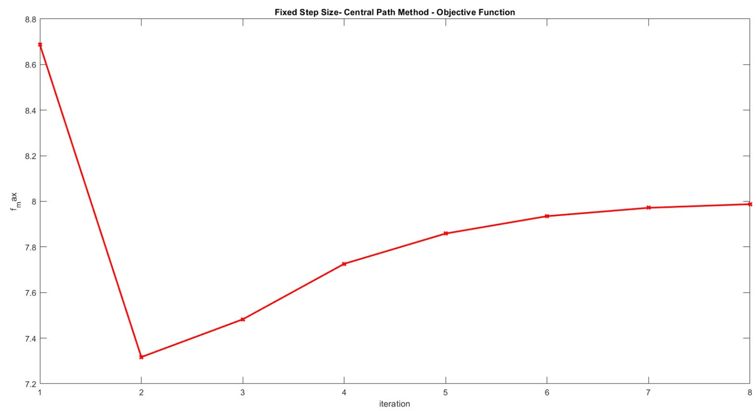

Figure 4: Central Path Method - Objective Function

1.2 Central Path with Adaptive step size

The Adaptive Central Path method takes 6 iterations with ($\alpha = 0.8$, $\sigma = 0.3$)

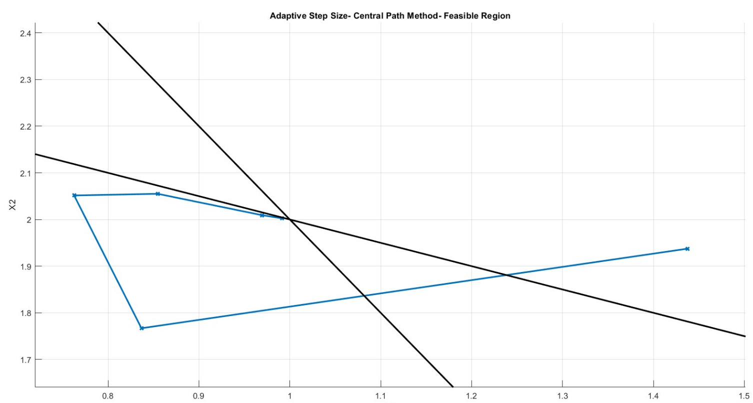

Figure 5: Central Path Method- Feasible Region

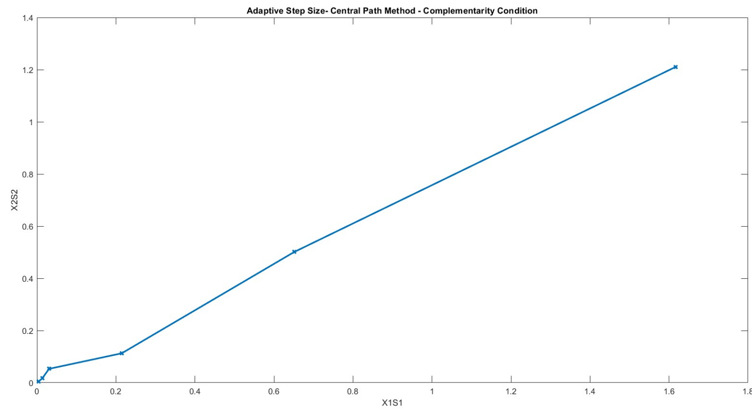

Figure 6: Central Path Method - Complementarity Condition

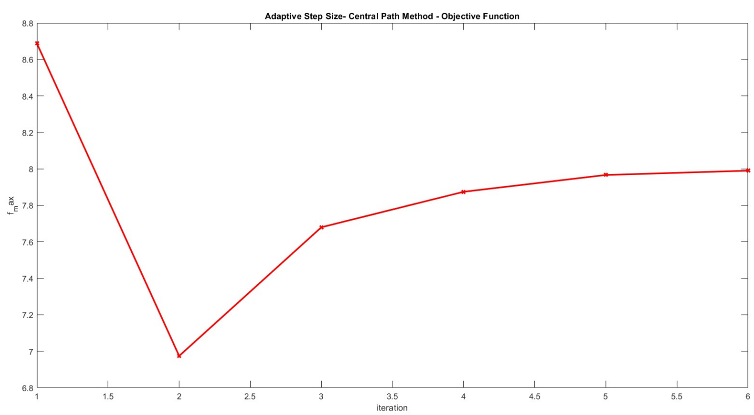

Figure 7: Central Path Method - Objective Function

1.3 Mehrotra Predictor-Corrector

The Fixed Central Path method takes 4 iterations with ($\alpha = 0.8$, $\sigma = 0.3$)

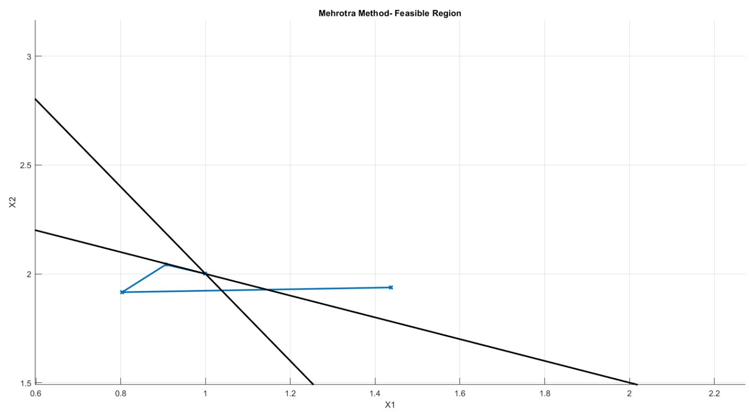

Figure 8: Mehrotra Method- Feasible Region

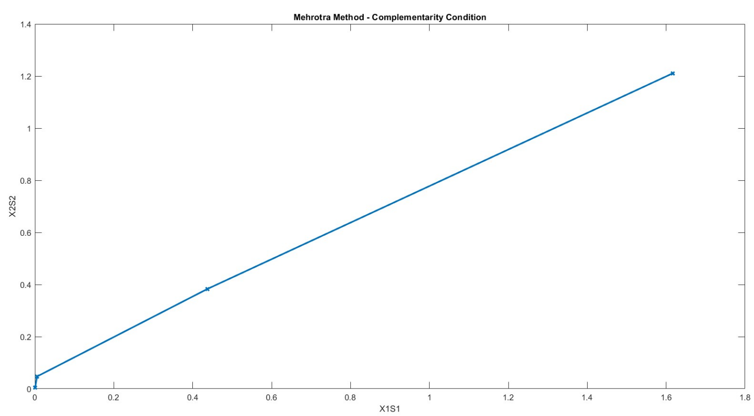

Figure 9: Mehrotra Method - Complementarity Condition

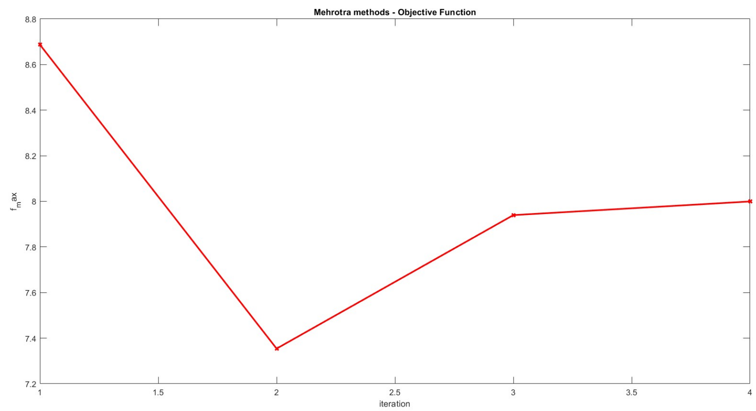

Figure 10: Methorta Method - Objective Function

2 Example 2:

$$
\begin{aligned}
\text{Max}\, Z &= 1.1x_1 + x_2 \\
\text{s.t.} \quad x_1 + x_2 &\leq 6 \\
x_1, x_2 &\geq 0
\end{aligned}
$$

The standard LP form $(Z = -z)$:

$$
\begin{aligned}
&\text{Min} \quad z = -1.1x_1 - x_2 \\
&\text{s.t.} \quad \begin{array}{l} x_1 + x_2 + x_3 \quad = 6 \\ x_1, x_2, x_3, \geq 0 \end{array} \quad \text{(where } x_3 \text{ are slack)}
\end{aligned}
$$

Hence, $n = 3$, $m = 1$.

$$
\begin{aligned}
X &= \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \\
C &= \begin{bmatrix} -1.1 & -1 & 0 \end{bmatrix} \\
A &= \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \\
\boldsymbol{b} &= \begin{bmatrix} 6 \end{bmatrix}
\end{aligned}
$$

Used $\sigma = 0.5$, $\alpha = 0.5$, and the stopping value be Tol $= 0.01$ for all method

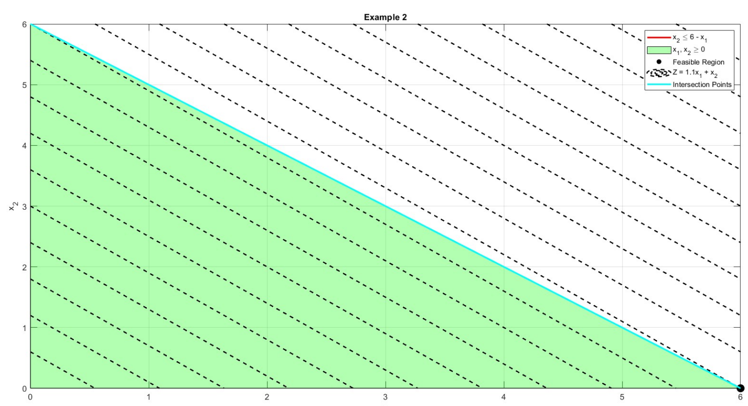

Figure 11: The Graphical Representation of Example 2

2.1 Central Path with fixed step size

The Fixed Central Path method takes 21 iterations with ($\alpha = 0.5$, $\sigma = 0.5$)

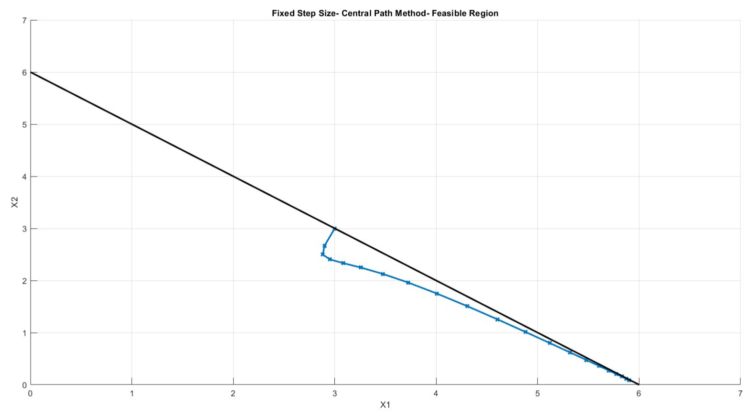

Figure 12: Central Path Method- Feasible Region

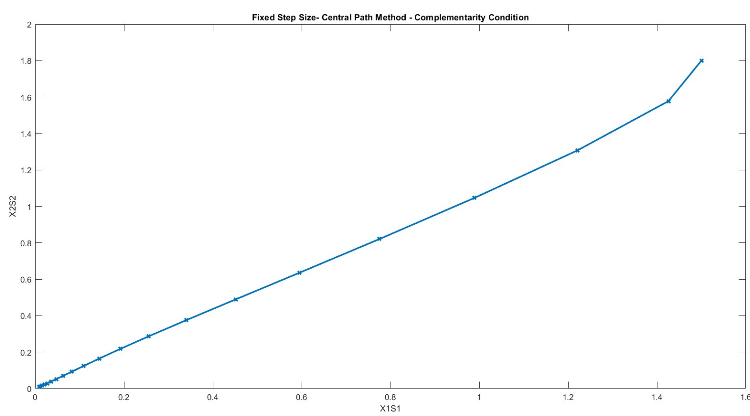

Figure 13: Central Path Method - Complementarity Condition

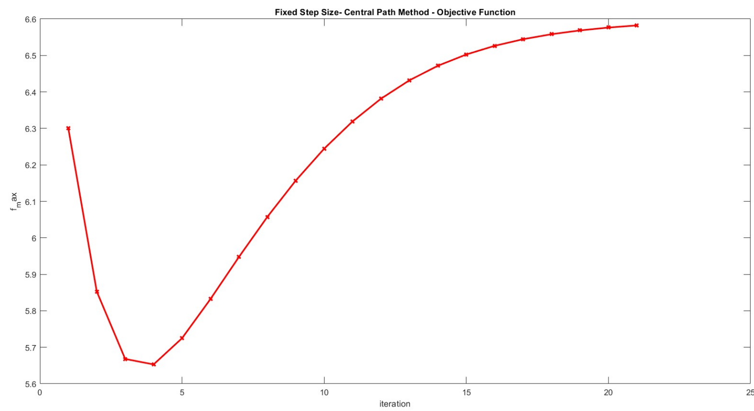

Figure 14: Central Path Method - Objective Function

2.2 Central Path with Adaptive step size

The Adaptive Central Path method takes 10 iterations with ($\alpha = 0.5$, $\sigma = 0.5$)

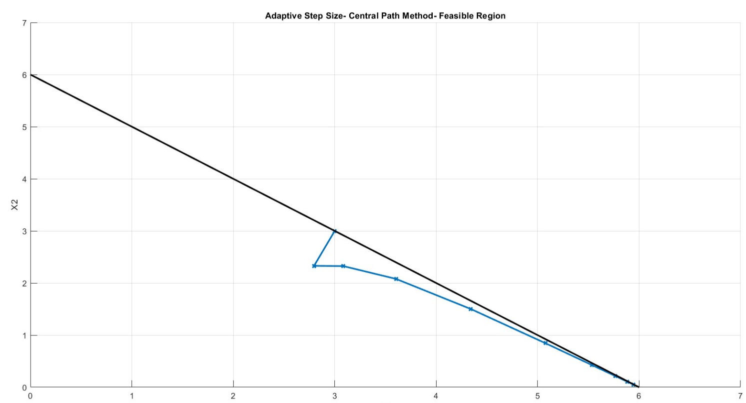

Figure 15: Central Path Method- Feasible Region

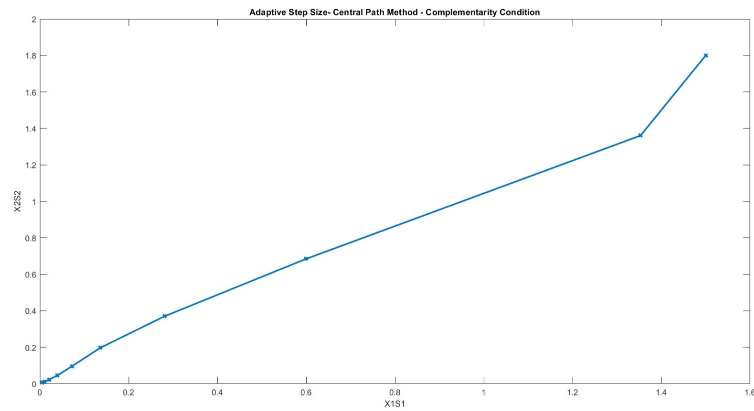

Figure 16: Central Path Method - Complementarity Condition

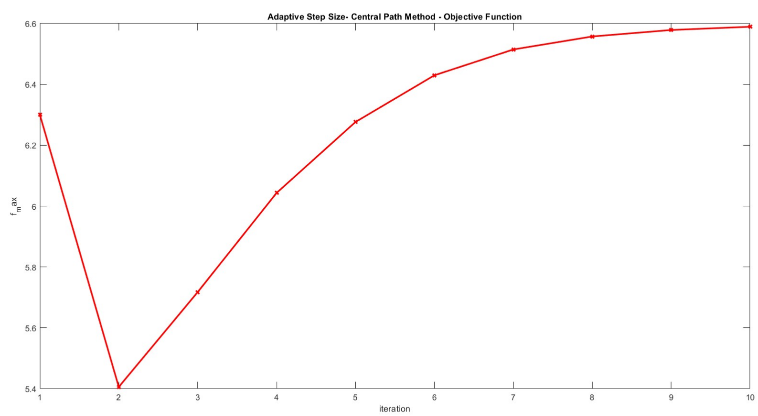

Figure 17: Central Path Method - Objective Function

2.3 Mehrotra Predictor-Corrector

The Fixed Central Path method takes 5 iterations with ($\alpha = 0.5$, $\sigma = 0.5$)

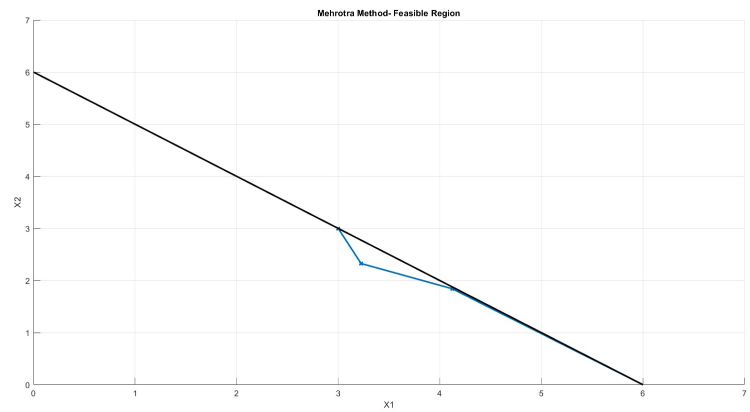

Figure 18: Mehrotra Method- Feasible Region

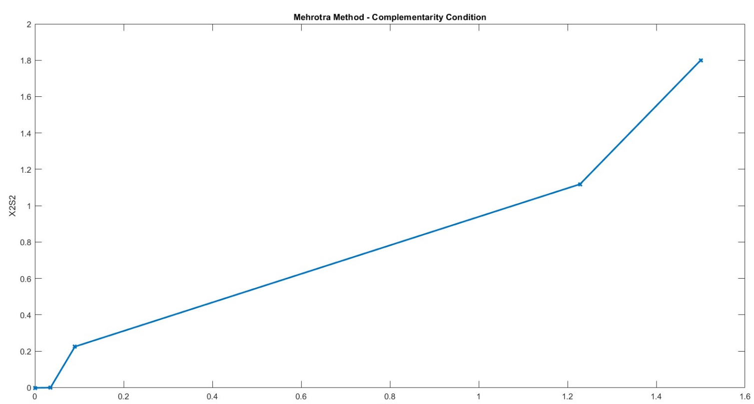

Figure 19: Mehrotra Method - Complementarity Condition

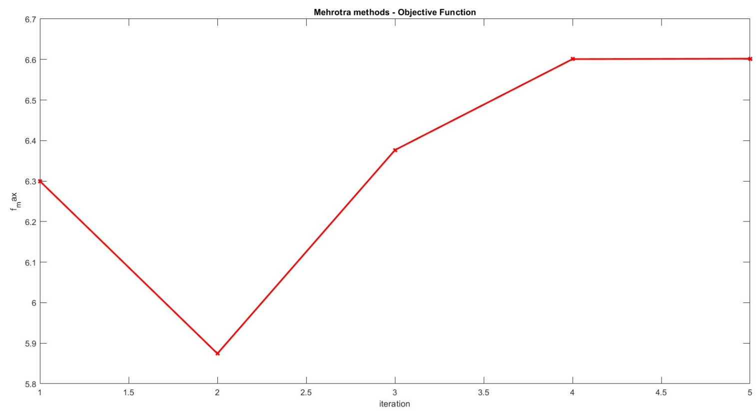

Figure 20: Methorta Method - Objective Function

3 Comparison

3.1 Example 1

In Example 1:

Fixed Central Path Method: Takes 8 iterations to converge.
Adaptive Central Path Method: Takes 6 iterations to converge.
Mehrotra Predictor-Corrector Method: Takes 4 iterations to converge.

Comparison: The Adaptive Central Path method outperforms the Fixed Central Path method in terms of convergence, taking fewer iterations (6 vs. 8). Additionally, the Mehrotra Predictor-Corrector method performs even better, converging in the fewest iterations (4). This suggests that the Mehrotra Predictor-Corrector method is the most efficient in terms of convergence in Example 1.

3.2 Example 2

In Example 2:

Fixed Central Path Method: Takes 21 iterations to converge.
Adaptive Central Path Method: Takes 10 iterations to converge.
Mehrotra Predictor-Corrector Method: Takes 5 iterations to converge.

Comparison: In Example 2, similar to Example 1, the Adaptive Central Path method demonstrates superior convergence performance compared to the Fixed Central Path method. It converges in 10 iterations, while the Fixed Central Path method takes a longer time with 21 iterations. The Mehrotra Predictor-Corrector method also shows efficient convergence, taking only 5 iterations. This emphasizes the effectiveness of adaptability in achieving convergence more rapidly, with the Mehrotra method being the most efficient in this case.

4 Implementation

4.1 Initial Points

```
function [x0,y0,s0]=Initial_Points(A,b,c)
% intial values for the intial points
At=transpose(A);
AAt_inv=inv(A*At);
x_hat=At*AAt_inv*b;
lamda_hat=AAt_inv*A*c;
s_hat=c-At*lamda_hat;
% eleminate the nonpositive components
sgma_x1=max((-3/2)*min(x_hat),0);
sgma_s1=max((-3/2)*min(s_hat),0);
e=ones(size(x_hat));
et=transpose(e);
x_hat1=x_hat+sgma_x1*e;
s_hat1=s_hat+sgma_s1*e;
%check that (xhat,shat)>=0
xhat1t=transpose(x_hat1);
sgma_x2=.5*((xhat1t*s_hat1)/(et*s_hat1));
sgma_s2=.5*((xhat1t*s_hat1)/(et*x_hat1));
%final values for the intial points
x0=x_hat1+sgma_x2*e;
y0=lamda_hat;
s0=s_hat1+sgma_s2*e;
end
```

4.2 Central Path Method

```
function Central_Path(A,b,c,alpha,beta,Tol,alpha_type)
[x,y,s]=Initial_Points(A,b,c); %initals
X_points=[x];
S_points=[s];
[m,n]=size(A);
%% J matrix componants
J11=zeros(n);
At=transpose(A);
I= eye(n);
J22=zeros(m,n);
J23=zeros(m);
S=diag(s);
J32=zeros(n,m);
X=diag(x);
%% f componants
e=ones(n,1);
Tao=transpose(x)*s;
Tao_array=[Tao];
mu=(Tao/n);
mu_array=[mu];
rc=At*y+s-c;
rb=A*x-b;
rxs=x.*s- mu.*beta.*e;
k =1;
No_iteration=[k];
while((mu)>=Tol)
```

```
I= eye(n);
J=[J11, At, I;
    A, J22, J23;
    S, J32, X];
f=[-rc;-rb;-rxs];
%calculate J by normal inverse
deltas=inv(J)*f;
deltas_x=deltas(1:n);
[nn,mm]=size(y);
deltas_y=deltas(n+1:nn+n);
deltas_s=deltas(nn+n+1:nn+2*n);

%% update
if strcmp(alpha_type, 'Fixed')
    prim_alpha=alpha;
    dual_alpha=alpha;
elseif strcmp(alpha_type, 'Adaptive')
    x_ratio = min(x./abs(deltas_x));
    s_ratio=min(s./abs(deltas_s));
    prim_alpha=min(1,x_ratio);
    dual_alpha=min(1,s_ratio);
else
    disp('you should enter the alpha_type Fixed or Adaptive');
end
x=x+prim_alpha*deltas_x;
s=s+dual_alpha*deltas_s;
if (all(x>=0) && all(s>=0)) %check the entring values of s and x
    are postive
    y=y+dual_alpha*deltas_y;
    S_points=[S_points,s];
    X_points=[X_points,x];
    S=diag(s);
    X=diag(x);
    Tao=transpose(x)*s;
    Tao_array=[Tao_array,Tao];
    mu=(Tao)/n;
    mu_array=[mu_array,mu];
    rc=At*y+s-c;
    rb=A*x-b;
    rxs=x.*s- mu.*beta.*e;
    k =k+1;
    No_iteration=[No_iteration,k];
else
    break;
end

end
%% ploting
%plot fesable regoin
x_coordinates = X_points(1, :);
y_coordinates = X_points(2, :);

% Create a figure 1
figure;
hold on
plot(x_coordinates, y_coordinates, 'x-', 'LineWidth', 2);
% % Plot the linear constraints
```

```
for i = 1:m
    st_array = [A(i, 1), A(i, 2), b(i)];
    x2 = @(x) (st_array(3) - st_array(1) * x) / st_array(2);
    fplot(x2,[0,7],'LineWidth', 2, 'Color', 'black');
end
ylim([0,7]);
xlabel('X1');
ylabel('X2');
title([alpha_type,' Step Size','- Central Path Method','- Feasible
    Region']);
hold off;
grid on;
% Create a figure 2
S1= S_points(1, :);
S2= S_points(2, :);
X1S1=[];
X2S2=[];
[nn,mm]=size(x_coordinates);
for i=1:mm
    X1S1=[X1S1,x_coordinates(1,i)*S1(1,i)];
    X2S2=[X2S2,y_coordinates(1,i)*S2(1,i)];
end
figure;
plot(X1S1, X2S2, 'x-', 'LineWidth', 2);
xlabel('X1S1');
ylabel('X2S2');
title([alpha_type,' Step Size','- Central Path Method',' -
    Complementarity Condition']);

%Create figure 3
f=@(x_1,x_2) c(1)*x_1+c(2)*x_2;
f_values=[];
for i=1:mm
    f_values=[f_values,-1*f(x_coordinates(1,i),y_coordinates(1,i))
    ];
end
figure;
plot(No_iteration, f_values, 'x-', 'LineWidth', 2, 'Color', 'red');
xlabel('iteration');
ylabel('f_max');
title([alpha_type,' Step Size','- Central Path Method',' -
    Objective Function']);
end
```

4.3 Mehrotra Predictor-Corrector

```
function Mehrotra(A,b,c,beta,Tol)
[x,y,s]=Initial_Points(A,b,c); %initals
X_points=[x];
S_points=[s];
[m,n]=size(A);
%% J matrix componants
At=transpose(A);
S=diag(s);
X=diag(x);
%% f componants
e=ones(n,1);
Tao=transpose(x)*s;
Tao_array=[Tao];
mu=(Tao/n);
mu_array=[mu];
rc=At*y+s-c;
rb=A*x-b;
rxs_aff=x.*s;
k =1;
No_iteration=[k];
while((mu)>=Tol)
S_inv=inv(S);
D2=S_inv*X;
AD2AT_inv=inv(A*D2*At);
%%cal aff
deltas_y_aff =AD2AT_inv*(-rb-A*S_inv*X*rc+A*S_inv*rxs_aff);
deltas_s_aff=-rc-At*deltas_y_aff;
deltas_x_aff=-S_inv*rxs_aff-X*S_inv*deltas_s_aff;

rxs=x.*s + deltas_x_aff.*deltas_s_aff.*e - mu.*beta.*e;
deltas_y =AD2AT_inv*(-rb-A*S_inv*X*rc+A*S_inv*rxs);
deltas_s=-rc-At*deltas_y;
deltas_x=-S_inv*rxs-X*S_inv*deltas_s;

%cal alpha_prime alpha_daul mu_aff
x_ratio_aff= min(x./abs(deltas_x_aff));
s_ratio_aff=min(s./abs(deltas_s_aff));
prim_alpha_aff=min(1,x_ratio_aff);
dual_alpha_aff=min(1,s_ratio_aff);
mu_aff= (transpose(x+prim_alpha_aff*deltas_x_aff)*(s+dual_alpha_aff
    *deltas_s_aff))/n;
beta=(mu_aff/mu)^3;

% update
x=x+prim_alpha_aff*deltas_x;
s=s+dual_alpha_aff*deltas_s;
y=y+dual_alpha_aff*deltas_y;
S_points=[S_points,s];
X_points=[X_points,x];
S=diag(s);
X=diag(x);
Tao=transpose(x)*s;
Tao_array=[Tao_array,Tao];
mu=(Tao)/n;
mu_array=[mu_array,mu];
```

```
rc=At*y+s-c;
rb=A*x-b;
rxs_aff=x.*s;
k =k+1;
No_iteration=[No_iteration,k];
end
%% ploting
%plot fesable regoin
x_coordinates = X_points(1, :);
y_coordinates = X_points(2, :);

% Create a figure 1
figure;
hold on
plot(x_coordinates, y_coordinates, 'x-', 'LineWidth', 2);
% % Plot the linear constraints
for i = 1:m
    st_array = [A(i, 1), A(i, 2), b(i)];
    x2 = @(x) (st_array(3) - st_array(1) * x) / st_array(2);
    fplot(x2,[0,7],'LineWidth', 2, 'Color', 'black');
end
ylim([0,7]);
xlabel('X1');
ylabel('X2');
title(['Mehrotra Method','- Feasible Region']);
hold off;
grid on;
% Create a figure 2
S1= S_points(1, :);
S2= S_points(2, :);
X1S1=[];
X2S2=[];
[nn,mm]=size(x_coordinates);
for i=1:mm
    X1S1=[X1S1,x_coordinates(1,i)*S1(1,i)];
    X2S2=[X2S2,y_coordinates(1,i)*S2(1,i)];
end
figure;
plot(X1S1, X2S2, 'x-', 'LineWidth', 2);
xlabel('X1S1');
ylabel('X2S2');
title(['Mehrotra Method',' - Complementarity Condition']);

%Create figure 3
f=@(x_1,x_2) c(1)*x_1+c(2)*x_2;
f_values=[];
for i=1:mm
    f_values=[f_values,-1*f(x_coordinates(1,i),y_coordinates(1,i))
    ];
end
figure;
plot(No_iteration, f_values, 'x-', 'LineWidth', 2, 'Color', 'red');
xlabel('iteration');
ylabel('f_max');
title([ 'Mehrotra methods',' - Objective Function']);
end
```

4.4 Matlab Test Cases Scenario For Validation

```
clear
clc
%% Example 1
c=[-2;-3;0;0];
A=[2 1 1 0;1 2 0 1];
b=[4;5];
alpha=0.8;
beta=0.3;
Tol=0.01;
Central_Path(A,b,c,alpha,beta,Tol,'Fixed');
Central_Path(A,b,c,alpha,beta,Tol,'Adaptive');
Mehrotra(A,b,c,beta,Tol);
%% Example 2
c=[-1.1;-1;0];
A=[1 1 1];
b=[6];
alpha=0.5;
beta=0.5;
Tol=0.01;
Central_Path(A,b,c,alpha,beta,Tol,'Fixed');
Central_Path(A,b,c,alpha,beta,Tol,'Adaptive');
Mehrotra(A,b,c,beta,Tol);
```