

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

<b>1</b>	<b>Example 1:</b>	<b>3</b>
1.1	Central Path with fixed step size . . . . .	4
1.2	Central Path with Adaptive step size . . . . .	5
1.3	Mehrotra Predictor-Corrector . . . . .	7
<b>2</b>	<b>Example 2:</b>	<b>9</b>
2.1	Central Path with fixed step size . . . . .	10
2.2	Central Path with Adaptive step size . . . . .	11
2.3	Mehrotra Predictor-Corrector . . . . .	13
<b>3</b>	<b>Comparison</b>	<b>15</b>
3.1	Example 1 . . . . .	15
3.2	Example 2 . . . . .	15
<b>4</b>	<b>Implementation</b>	<b>16</b>
4.1	Initial Points . . . . .	16
4.2	Central Path Method . . . . .	16
4.3	Mehrotra Predictor-Corrector . . . . .	19
4.4	Matlab Test Cases Scenario For Validation . . . . .	21

## 1 Example 1:

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

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

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

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

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

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

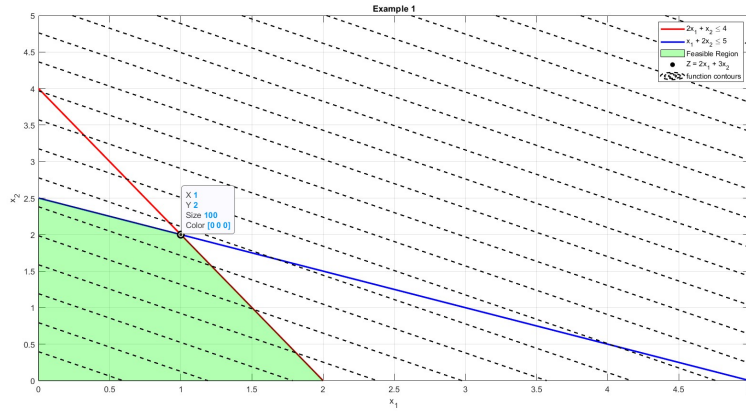


Figure 1: The Graphical Representation of Example 1

## 1.1 Central Path with fixed step size

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

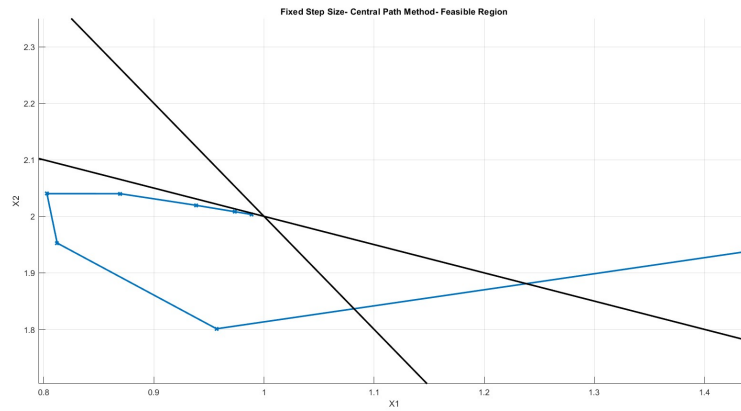


Figure 2: Central Path Method- Feasible Region

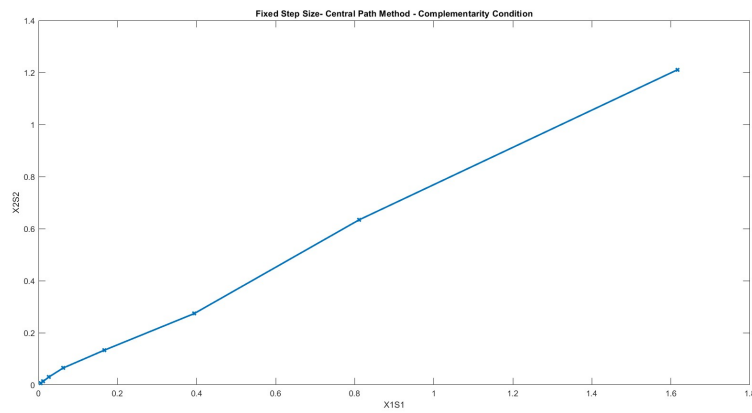


Figure 3: Central Path Method - Complementarity Condition

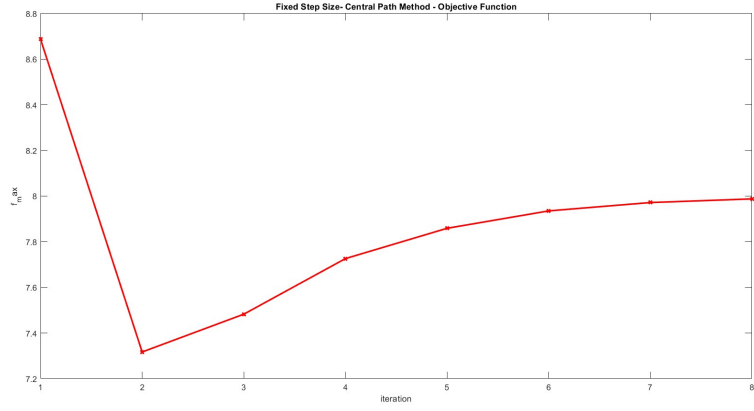


Figure 4: Central Path Method - Objective Function

## 1.2 Central Path with Adaptive step size

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

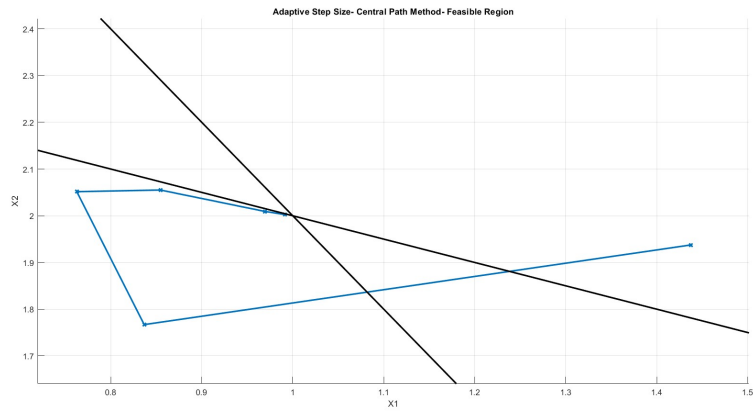


Figure 5: Central Path Method- Feasible Region

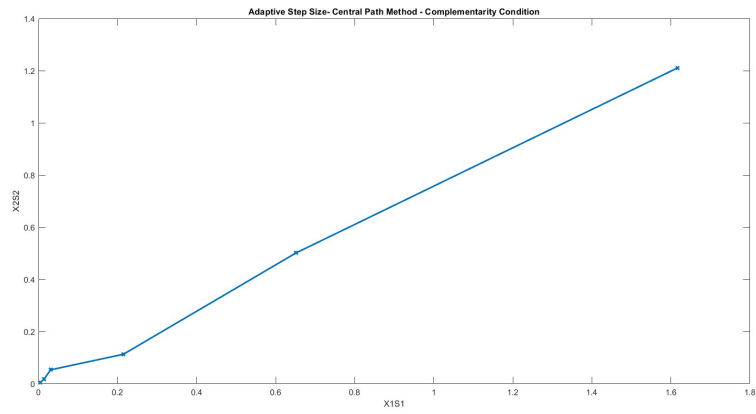


Figure 6: Central Path Method - Complementarity Condition

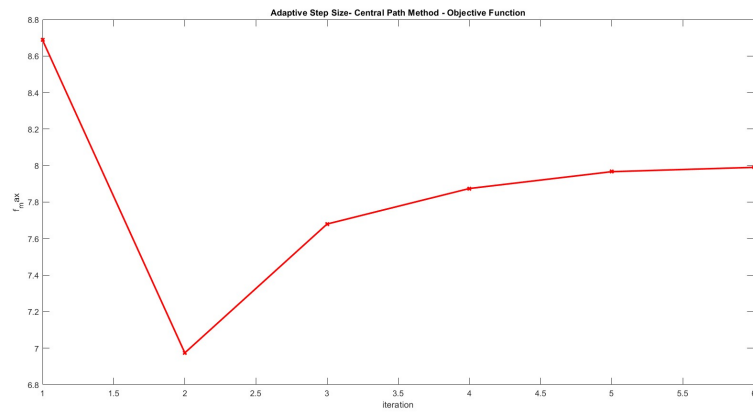


Figure 7: Central Path Method - Objective Function

### 1.3 Mehrotra Predictor-Corrector

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

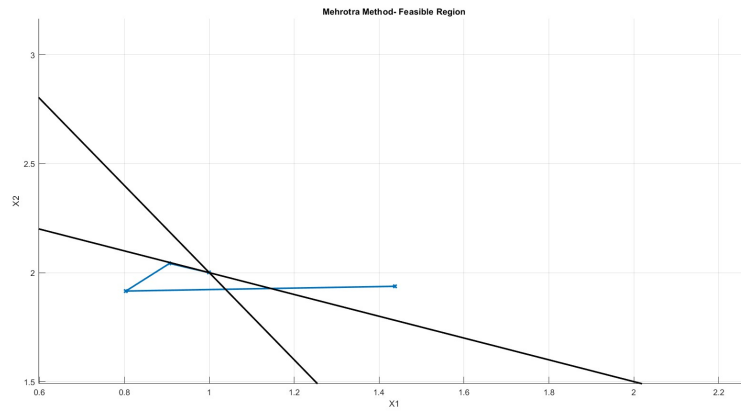


Figure 8: Mehrotra Method- Feasible Region

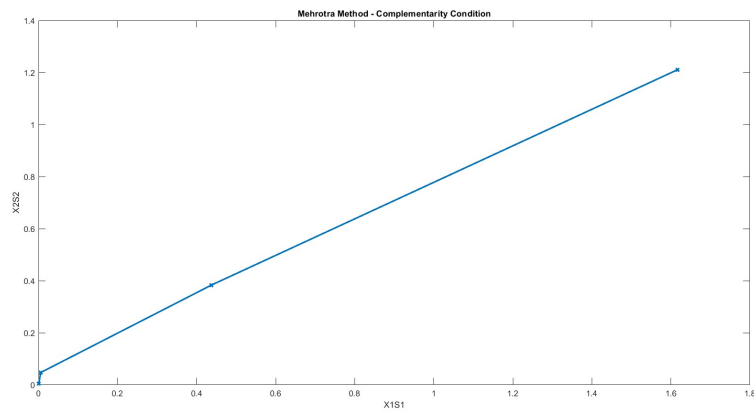


Figure 9: Mehrotra Method - Complementarity Condition

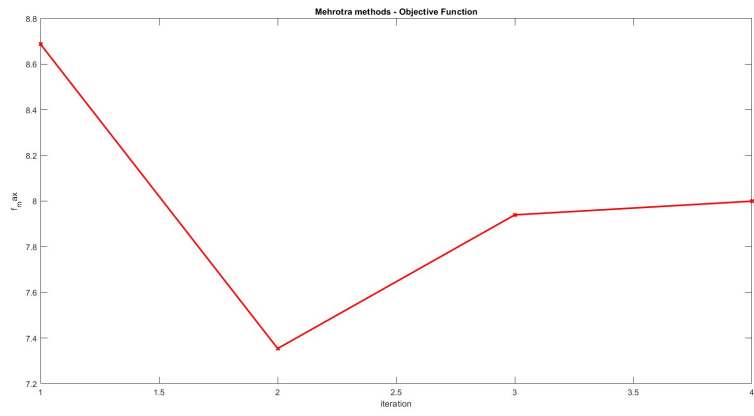


Figure 10: Methorta Method - Objective Function



## 2 Example 2:

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

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

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

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

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

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

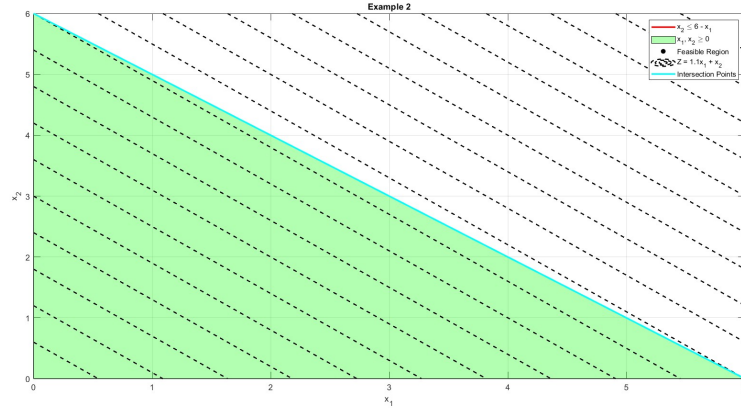


Figure 11: The Graphical Representation of Example 2

## 2.1 Central Path with fixed step size

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

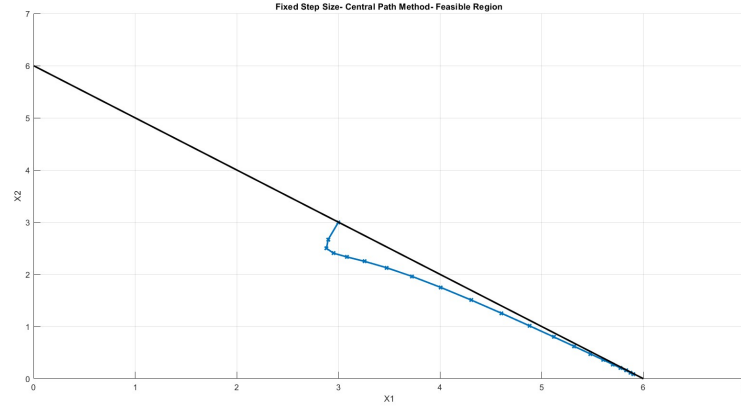


Figure 12: Central Path Method- Feasible Region

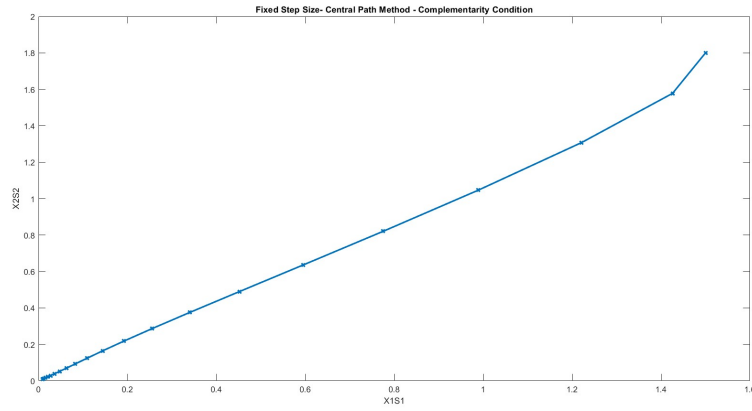


Figure 13: Central Path Method - Complementarity Condition

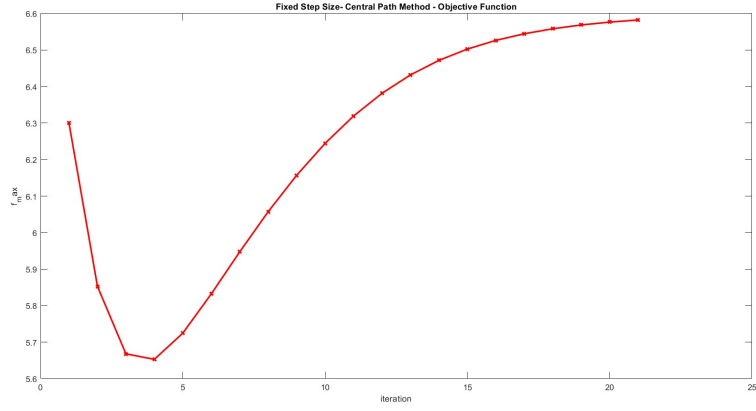


Figure 14: Central Path Method - Objective Function

## 2.2 Central Path with Adaptive step size

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

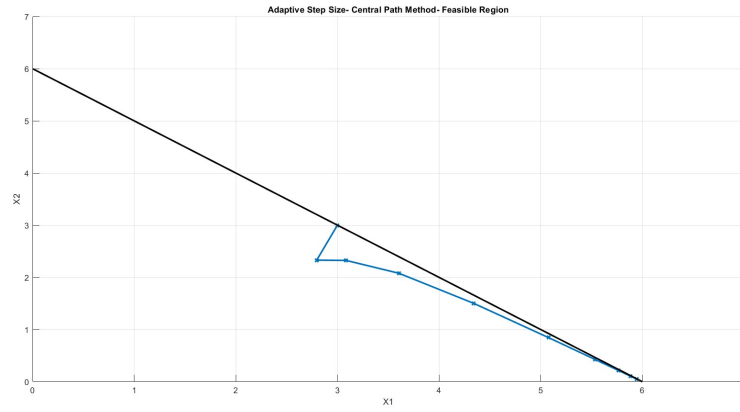


Figure 15: Central Path Method- Feasible Region

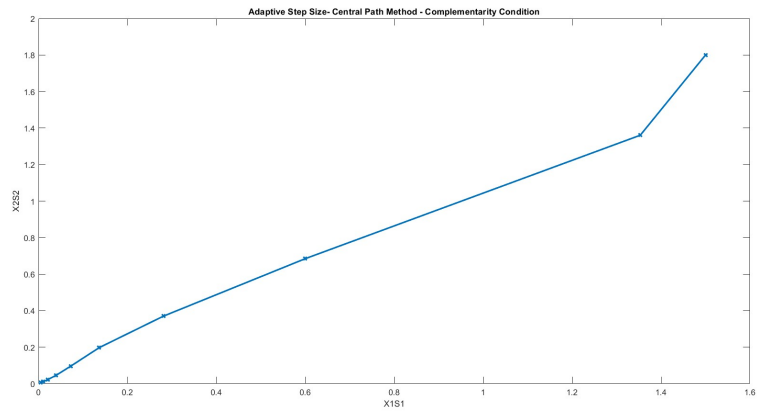


Figure 16: Central Path Method - Complementarity Condition

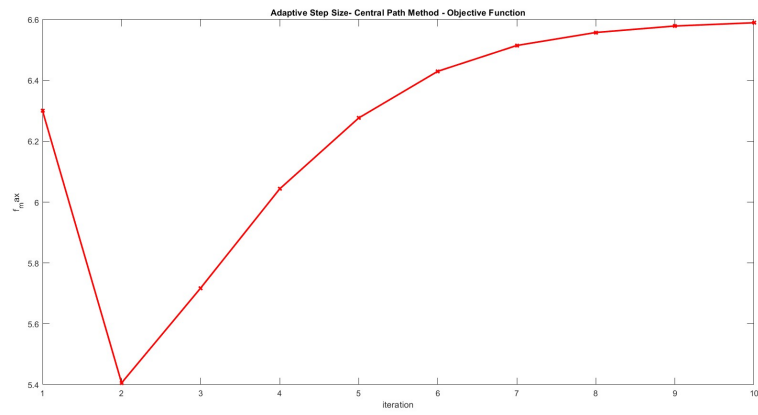


Figure 17: Central Path Method - Objective Function

## 2.3 Mehrotra Predictor-Corrector

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

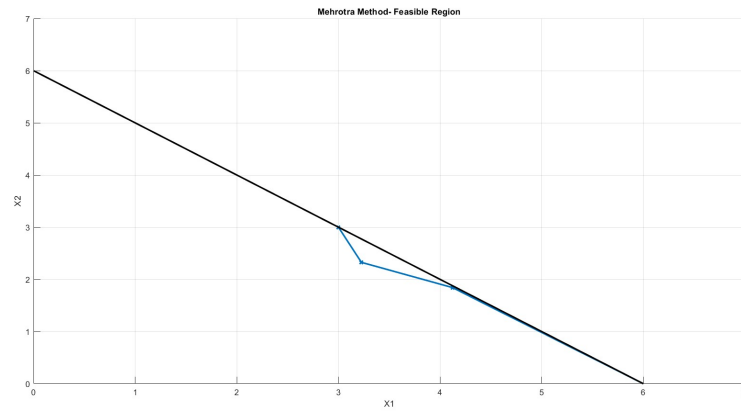


Figure 18: Mehrotra Method- Feasible Region

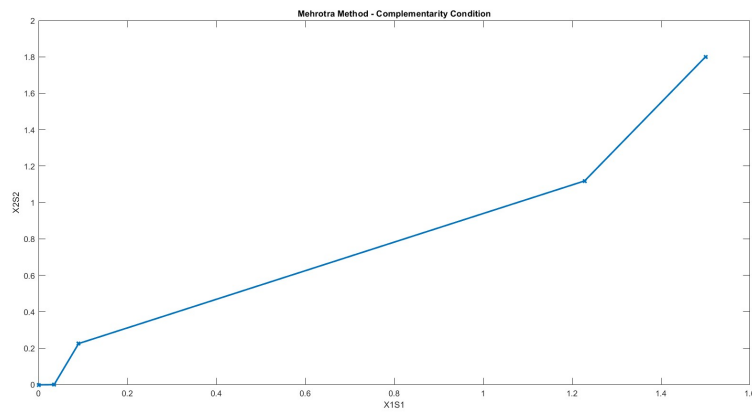


Figure 19: Mehrotra Method - Complementarity Condition

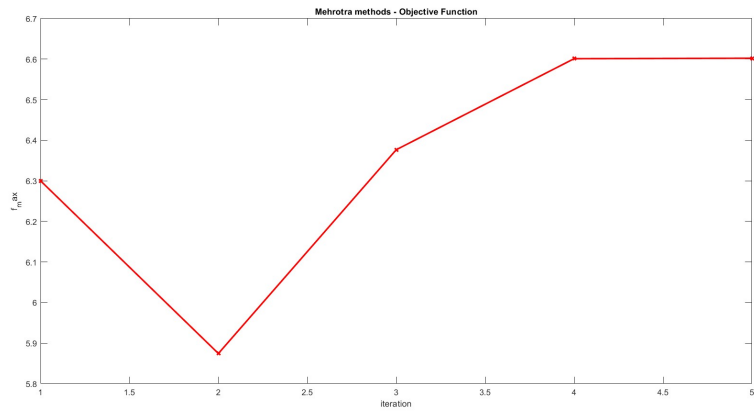


Figure 20: Methorta Method - Objective Function

## 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

```
1 function [x0,y0,s0]=Initial_Points(A,b,c)
2 % intial values for the intial points
3 At=transpose(A);
4 AAt_inv=inv(A*At);
5 x_hat=At*AAt_inv*b;
6 lamda_hat=AAt_inv*A*c;
7 s_hat=c-At*lamda_hat;
8 % eleminate the nonpositive components
9 sgma_x1=max((-3/2)*min(x_hat),0);
10 sgma_s1=max((-3/2)*min(s_hat),0);
11 e=ones(size(x_hat));
12 et=transpose(e);
13 x_hat1=x_hat+sgma_x1*e;
14 s_hat1=s_hat+sgma_s1*e;
15 %check that (xhat,shat)>=0
16 xhat1t=transpose(x_hat1);
17 sgma_x2=.5*((xhat1t*s_hat1)/(et*s_hat1));
18 sgma_s2=.5*((xhat1t*s_hat1)/(et*x_hat1));
19 %final values for the intial points
20 x0=x_hat1+sgma_x2*e;
21 y0=lamda_hat;
22 s0=s_hat1+sgma_s2*e;
23 end
```

### 4.2 Central Path Method

```
1 function Central_Path(A,b,c,alpha,beta,Tol,alpha_type)
2 [x,y,s]=Initial_Points(A,b,c); %initials
3 X_points=[x];
4 S_points=[s];
5 [m,n]=size(A);
6 %% J matrix componants
7 J11=zeros(n);
8 At=transpose(A);
9 I= eye(n);
10 J22=zeros(m,n);
11 J23=zeros(m);
12 S=diag(s);
13 J32=zeros(n,m);
14 X=diag(x);
15 %% f componants
16 e=ones(n,1);
17 Tao=transpose(x)*s;
18 Tao_array=[Tao];
19 mu=(Tao/n);
20 mu_array=[mu];
21 rc=At*y+s-c;
22 rb=A*x-b;
23 rxs=x.*s- mu.*beta.*e;
24 k =1;
25 No_iteration=[k];
26 while((mu)>=Tol)
```



```

27 I= eye(n);
28 J=[J11, At, I;
29     A, J22, J23;
30     S, J32, X];
31 f=[-rc;-rb;-rxs];
32 %calculate J by normal inverse
33 deltas=inv(J)*f;
34 deltas_x=deltas(1:n);
35 [nn,mm]=size(y);
36 deltas_y=deltas(n+1:nn+n);
37 deltas_s=deltas(nn+n+1:nn+2*n);
38
39 %% update
40 if strcmp(alpha_type, 'Fixed')
41     prim_alpha=alpha;
42     dual_alpha=alpha;
43 elseif strcmp(alpha_type, 'Adaptive')
44     x_ratio = min(x./abs(deltas_x));
45     s_ratio=min(s./abs(deltas_s));
46     prim_alpha=min(1,x_ratio);
47     dual_alpha=min(1,s_ratio);
48 else
49     disp('you should enter the alpha_type Fixed or Adaptive');
50 end
51 x=x+prim_alpha*deltas_x;
52 s=s+dual_alpha*deltas_s;
53 if (all(x>=0) && all(s>=0)) %check the entering values of s and x
54     are postive
55     y=y+dual_alpha*deltas_y;
56     S_points=[S_points,s];
57     X_points=[X_points,x];
58     S=diag(s);
59     X=diag(x);
60     Tao=transpose(x)*s;
61     Tao_array=[Tao_array,Tao];
62     mu=(Tao)/n;
63     mu_array=[mu_array,mu];
64     rc=At*y+s-c;
65     rb=A*x-b;
66     rxs=x.*s- mu.*beta.*e;
67     k =k+1;
68     No_iteration=[No_iteration,k];
69 else
70     break;
71 end
72
73 %% plotting
74 %plot fesable regoin
75 x_coordinates = X_points(1, :);
76 y_coordinates = X_points(2, :);
77
78 % Create a figure 1
79 figure;
80 hold on
81 plot(x_coordinates, y_coordinates, 'x-', 'LineWidth', 2);
82 %% Plot the linear constraints

```

```

83 for i = 1:m
84     st_array = [A(i, 1), A(i, 2), b(i)];
85     x2 = @(x) (st_array(3) - st_array(1) * x) / st_array(2);
86     fplot(x2,[0,7], 'LineWidth', 2, 'Color', 'black');
87 end
88 ylim([0,7]);
89 xlabel('X1');
90 ylabel('X2');
91 title([alpha_type, ' Step Size', '- Central Path Method', '- Feasible
      Region']));
92 hold off;
93 grid on;
94 % Create a figure 2
95 S1= S_points(1, :);
96 S2= S_points(2, :);
97 X1S1=[];
98 X2S2=[];
99 [nn,mm]=size(x_coordinates);
100 for i=1:mm
101     X1S1=[X1S1,x_coordinates(1,i)*S1(1,i)];
102     X2S2=[X2S2,y_coordinates(1,i)*S2(1,i)];
103 end
104 figure;
105 plot(X1S1, X2S2, 'x-', 'LineWidth', 2);
106 xlabel('X1S1');
107 ylabel('X2S2');
108 title([alpha_type, ' Step Size', '- Central Path Method', '-
      Complementarity Condition']));
109
110 %Create figure 3
111 f=@(x_1,x_2) c(1)*x_1+c(2)*x_2;
112 f_values=[];
113 for i=1:mm
114     f_values=[f_values,-1*f(x_coordinates(1,i),y_coordinates(1,i))
115 ];
116 end
117 figure;
118 plot(No_iteration, f_values, 'x-', 'LineWidth', 2, 'Color', 'red');
119 xlabel('iteration');
120 ylabel('f_max');
121 title([alpha_type, ' Step Size', '- Central Path Method', '-
      Objective Function']));
122 end

```

### 4.3 Mehrotra Predictor-Corrector

```

1 function Mehrotra(A,b,c,beta,Tol)
2 [x,y,s]=Initial_Points(A,b,c); %initials
3 X_points=[x];
4 S_points=[s];
5 [m,n]=size(A);
6 %% J matrix componants
7 At=transpose(A);
8 S=diag(s);
9 X=diag(x);
10 %% f componants
11 e=ones(n,1);
12 Tao=transpose(x)*s;
13 Tao_array=[Tao];
14 mu=(Tao/n);
15 mu_array=[mu];
16 rc=At*y+s-c;
17 rb=A*x-b;
18 rxs_aff=x.*s;
19 k =1;
20 No_iteration=[k];
21 while((mu)>=Tol)
22 S_inv=inv(S);
23 D2=S_inv*X;
24 AD2AT_inv=inv(A*D2*At);
25 %%cal aff
26 deltas_y_aff =AD2AT_inv*(-rb-A*S_inv*X*rc+A*S_inv*rxs_aff);
27 deltas_s_aff=-rc-At*deltas_y_aff;
28 deltas_x_aff=-S_inv*rxs_aff-X*S_inv*deltas_s_aff;
29
30 rxs=x.*s + deltas_x_aff.*deltas_s_aff.*e - mu.*beta.*e;
31 deltas_y =AD2AT_inv*(-rb-A*S_inv*X*rc+A*S_inv*rxs);
32 deltas_s=-rc-At*deltas_y;
33 deltas_x=-S_inv*rxs-X*S_inv*deltas_s;
34
35 %cal alpha_prime alpha_daul mu_aff
36 x_ratio_aff= min(x./abs(deltas_x_aff));
37 s_ratio_aff=min(s./abs(deltas_s_aff));
38 prim_alpha_aff=min(1,x_ratio_aff);
39 dual_alpha_aff=min(1,s_ratio_aff);
40 mu_aff= (transpose(x+prim_alpha_aff*deltas_x_aff)*(s+dual_alpha_aff
    *deltas_s_aff))/n;
41 beta=(mu_aff/mu)^3;
42
43 % update
44 x=x+prim_alpha_aff*deltas_x;
45 s=s+dual_alpha_aff*deltas_s;
46 y=y+dual_alpha_aff*deltas_y;
47 S_points=[S_points,s];
48 X_points=[X_points,x];
49 S=diag(s);
50 X=diag(x);
51 Tao=transpose(x)*s;
52 Tao_array=[Tao_array,Tao];
53 mu=(Tao)/n;
54 mu_array=[mu_array,mu];

```

```

55 rc=A*t*y+s-c;
56 rb=A*x-b;
57 rxs_aff=x.*s;
58 k =k+1;
59 No_iteration=[No_iteration,k];
60 end
61 %% plotting
62 %plot fesable regoin
63 x_coordinates = X_points(1, :);
64 y_coordinates = X_points(2, :);
65
66 % Create a figure 1
67 figure;
68 hold on
69 plot(x_coordinates, y_coordinates, 'x-', 'LineWidth', 2);
70 % % Plot the linear constraints
71 for i = 1:m
72     st_array = [A(i, 1), A(i, 2), b(i)];
73     x2 = @(x) (st_array(3) - st_array(1) * x) / st_array(2);
74     fplot(x2,[0,7], 'LineWidth', 2, 'Color', 'black');
75 end
76 ylim([0,7]);
77 xlabel('X1');
78 ylabel('X2');
79 title(['Mehrotra Method',' - Feasible Region']);
80 hold off;
81 grid on;
82 % Create a figure 2
83 S1= S_points(1, :);
84 S2= S_points(2, :);
85 X1S1=[];
86 X2S2=[];
87 [nn,mm]=size(x_coordinates);
88 for i=1:mm
89     X1S1=[X1S1,x_coordinates(1,i)*S1(1,i)];
90     X2S2=[X2S2,y_coordinates(1,i)*S2(1,i)];
91 end
92 figure;
93 plot(X1S1, X2S2, 'x-', 'LineWidth', 2);
94 xlabel('X1S1');
95 ylabel('X2S2');
96 title(['Mehrotra Method',' - Complementarity Condition']);
97
98 %Create figure 3
99 f=@(x_1,x_2) c(1)*x_1+c(2)*x_2;
100 f_values=[];
101 for i=1:mm
102     f_values=[f_values,-1*f(x_coordinates(1,i),y_coordinates(1,i))
103 ];
104 end
105 figure;
106 plot(No_iteration, f_values, 'x-', 'LineWidth', 2, 'Color', 'red');
107 xlabel('iteration');
108 ylabel('f_max');
109 title(['Mehrotra methods',' - Objective Function']);
110 end

```

## 4.4 Matlab Test Cases Scenario For Validation

```
1 clear
2 clc
3 %% Example 1
4 c=[-2;-3;0;0];
5 A=[2 1 1 0;1 2 0 1];
6 b=[4;5];
7 alpha=0.8;
8 beta=0.3;
9 Tol=0.01;
10 Central_Path(A,b,c,alpha,beta,Tol,'Fixed');
11 Central_Path(A,b,c,alpha,beta,Tol,'Adaptive');
12 Mehrotra(A,b,c,beta,Tol);
13 %% Example 2
14 c=[-1.1;-1;0];
15 A=[1 1 1];
16 b=[6];
17 alpha=0.5;
18 beta=0.5;
19 Tol=0.01;
20 Central_Path(A,b,c,alpha,beta,Tol,'Fixed');
21 Central_Path(A,b,c,alpha,beta,Tol,'Adaptive');
22 Mehrotra(A,b,c,beta,Tol);
```