Homework 02

Problem 1

Consider using a gradient algorithm to minimize the function . With the initial guess as $x = [-0.8, 0.25]^T$

a. To initialize the line-search apply bracketing procedure along the line starting at in the direction of negative gradient. Use = 0.075
b. Apply the golden section method to reduce the width of uncertainty region to 0.01. Organize results of your computation in table format.
c. Repeat the above using Fibonacci method.

Solution:

Matlab code – problem1.m

```
clc; clear all; close all;
% Declare the symbolic variables
syms x1 x2 alph;
% Declare function in terms of x1 x2
f = @(x1, x2) x1.^2+x2.^2+x1.*x2;
X_init = [0.8,-.25];
epsilon = 0.075;
k = 1;

[a0, b0] = Bracketing(f, X_init, epsilon); %run bracketing on phia to find points
[s,t, dat] = GoldenSection(a0,b0,f);
[s1,t1, dat1] = FibonacciSeq(a0,b0,f);
xlswrite('GoldenSearch.xlsx',dat);
xlswrite('Fibonacci.xlsx', dat1);
```

Matlab function – Brackting.m

```
function [a0,b0] = Bracketing(f, x_init, epsilon)
%BRACKETING Summary of this function goes here
%   Detailed explanation goes here
   x0 = x_init; x1 = x0+epsilon; x2 = x1+2*epsilon;
    n = 1;
    %bracketing procedure
    if((f(x0(1), x0(2))>f(x1(1), x1(2))) && (f(x1(1), x1(2))<f(x2(1), x2(2))))
            a0 = x0;
            b0 = x2;
    else
        while not((f(x0(1), x0(2))>f(x1(1), x1(2))) && (f(x1(1), x1(2))<f(x2(1),
x2(2))))
            if((f(x0(1), x0(2))>f(x1(1), x1(2))) && (f(x1(1), x1(2))>f(x2(1),
x2(2))))
                x0 = x1;
                x1 = x2;
                x2 = x2+(2^(n+1))*epsilon;
            elseif((f(x0(1), x0(2))<f(x2(1), x2(2))) && (f(x1(1), x2(2))<f(x2(1),
x2(2))))
                x2 = x1;
                x1 = x0;
                x0 = x0-(2^(n+1))*epsilon;
            end
            n = n+1;
        end
```

```
        a0 = x0;
        b0 = x2;
    end
end
```

Matlab code – GoldenSection.m

```
function [s,t, dat] = GoldenSection(a0, b0, fcn)
%GOLDENSECTION Summary of this function goes here
%   Detailed explanation goes here
%golden section
    epsi = 0.075;
    N = ceil(log(0.01/norm(b0-a0))/log(0.6180));
    rho = 0.382;
    a = a0;
    b = b0;
    s = a+rho*(b-a);
    t = a+(1-rho)*(b-a);
    f1= fcn(s(1), s(2));
    f2= fcn(t(1), t(2));
    dat = {'Iteration','rhok','ak','bk','f(ak)','f(bk)','new int'};
    for n = 1:N
        if fcn(s(1), s(2))< fcn(t(1), t(2))
           b = t;
           t = s;
           s = a+rho*(b-a);
           f2 = f1;
           f1 = fcn(s(1), s(2));
           dat{n+1,5} = f1;
           dat{n+1,6} = f2;
        elseif fcn(s(1), s(2))> fcn(t(1), t(2))
           a = s;
           s = t;
           t = a+(1-rho)*(b-a);
           f1 = f2;
           f2 = fcn(t(1), t(2));
           dat{n+1,5} = f1;
           dat{n+1,6} = f2;
        else
            break;
        end
%         dat{n,:} = {n,rho,s,t,f1,f2,[s,t]};
        dat{n+1,1} = n;
        dat{n+1,2} = rho;
        dat{n+1,3} = mat2str(s);
        dat{n+1,4} = mat2str(t);
        dat{n+1,7} = mat2str([s;t]);
    end
end
```

Matlab code – Fibonacci.m

```
function [s,t, dat] = FibonacciSeq(a0, b0, fcn)
%FIBONACCISEQ Summary of this function goes here
%   Detailed explanation goes here
%     rho =
    N = ceil(1+(2*0.1)/(0.01/norm(b0-a0)));
    n=1;
    while (fibonacci(n)<N)
        fiboNum(n) = fibonacci(n+1);
        n=n+1;
    end
    for i = 1:length(fiboNum)-1
        rho(i) = 1- (fiboNum(length(fiboNum)-i)/fiboNum(length(fiboNum)-i+1));
    end
    a = a0;
    b = b0;
    s = a+rho(1)*(b-a);
    t = a+(1-rho(1))*(b-a);
    f1= fcn(s(1), s(2));
    f2= fcn(t(1), t(2));
    dat = {'Iteration','rhok','ak','bk','f(ak)','f(bk)','new int'};
    for n = 1:length(fiboNum)-1
        if fcn(s(1), s(2))< fcn(t(1), t(2))
           b = t;
           t = s;
           s = a+rho(n)*(b-a);
           f2 = f1;
           f1 = fcn(s(1), s(2));
        else
           a = s;
           s = t;
           t = a+(1-rho(n))*(b-a);
           f1 = f2;
           f2 = fcn(t(1), t(2));
        end
%          dat{n,:} = {n,rho,s,t,f1,f2,[s,t]};
        dat{n+1,1} = n;
        dat{n+1,2} = rho(n);
        dat{n+1,3} = mat2str(s);
        dat{n+1,4} = mat2str(t);
        dat{n+1,5} = f1;
        dat{n+1,6} = f2;
        dat{n+1,7} = mat2str([s;t]);
    end
end
```

The results were recorded in a excel sheet for every iteration in the process for Golden section and Fibonacci methods as mentioned in the problem.

Iteration	rhok	ak	bk	f(ak)	f(bk)	new int
1	0.382	[0.4562 -0.5938]	[0.5875316 -0.4624684]	0.28982532	0.287355603	[0.4562 -0.5938;0.5875316 -0.4624684]
2	0.382	[0.5875316 -0.4624684]	[0.6686684 -0.3813316]	0.287355603	0.337546827	[0.5875316 -0.4624684;0.6686684 -0.3813316]
3	0.382	[0.5373629288 -0.5126370712]	[0.5875316 -0.4624684]	0.276083526	0.287355603	[0.5373629288 -0.5126370712;0.5875316 -0.4624684]
4	0.382	[0.5063686712 -0.5436313288]	[0.5373629288 -0.5126370712]	0.276666379	0.276083526	[0.5063686712 -0.5436313288;0.5373629288 -0.5126370712]
5	0.382	[0.5373629288 -0.5126370712]	[0.5565273611984 -0.4934726388016]	0.276083526	0.278606924	[0.5373629288 -0.5126370712;0.5565273611984 -0.4934726388016]
6	0.382	[0.525529290779389 -0.524470709220611]	[0.5373629288 -0.5126370712]	0.27562584	0.276083526	[0.525529290779389 -0.524470709220611;0.5373629288 -0.5126370712]
7	0.382	[0.5182084776032 -0.5317915223968]	[0.525529290779389 -0.524470709220611]	0.275763374	0.27562584	[0.5182084776032 -0.5317915223968;0.525529290779389 -0.524470709220611]
8	0.382	[0.525529290779389 -0.524470709220611]	[0.530045928442823 -0.519954071557178]	0.27562584	0.275701384	[0.525529290779389 -0.524470709220611;0.530045928442823 -0.519954071557178]
9	0.382	[0.522730383823936 -0.527269616176064]	[0.525529290779389 -0.524470709220611]	0.275640453	0.27562584	[0.522730383823936 -0.527269616176064;0.525529290779389 -0.524470709220611]
10	0.382	[0.525529290779389 -0.524470709220611]	[0.527251390398408 -0.522748609601592]	0.27562584	0.275640206	[0.525529290779389 -0.524470709220611;0.527251390398408 -0.522748609601592]
11	0.382	[0.524457408335384 -0.525542591664616]	[0.525529290779389 -0.524470709220611]	0.275625883	0.27562584	[0.524457408335384 -0.525542591664616;0.525529290779389 -0.524470709220611]

Table 1. Results for Golden Search Method. First iteration is the first improvement over the points given by Bracketing

Iteration	rhok	ak	bk	f(ak)	f(bk)	new int
1	0.382352941	[0.455882352941177 -0.594117647058823]	[0.587456747404844 -0.462543252595156]	0.289956747	0.287327536	[0.455882352941177 -0.594117647058823;0.587456747404844 -0.462543252595156]
2	0.380952381	[0.587456747404844 -0.462543252595156]	[0.66890756302521 -0.38109243697479]	0.287327536	0.33775316	[0.587456747404844 -0.462543252595156;0.66890756302521 -0.38109243697479]
3	0.384615385	[0.53781512605042 -0.51218487394958]	[0.587456747404844 -0.462543252595156]	0.276117682	0.287327536	[0.53781512605042 -0.51218487394958;0.587456747404844 -0.462543252595156]
4	0.375	[0.505222750865052 -0.544777249134948]	[0.53781512605042 -0.51218487394958]	0.276798419	0.276117682	[0.505222750865052 -0.544777249134948;0.53781512605042 -0.51218487394958]
5	0.4	[0.53781512605042 -0.51218487394958]	[0.554563148788927 -0.495436851211073]	0.276117682	0.278246939	[0.53781512605042 -0.51218487394958;0.554563148788927 -0.495436851211073]
6	0.333333333	[0.52166955017301 -0.52833044982699]	[0.53781512605042 -0.51218487394958]	0.275658276	0.276117682	[0.52166955017301 -0.52833044982699;0.53781512605042 -0.51218487394958]
7	0.5	[0.521518938457736 -0.528481061542264]	[0.52166955017301 -0.52833044982699]	0.275661353	0.275658276	[0.521518938457736 -0.528481061542264;0.52166955017301 -0.52833044982699]

Table 1. Results for Fibonacci Method. First iteration is the first improvement over the points given by Bracketing

Problem 2

For the function

$f(x1, x2) = (x2 - x1)4 + 12x1x2 - x1 + x2 - 3$,

a. Use MATLAB's commands meshgrid and mesh to generate its 3D plot. The range of x1 and x2 is the same and it should be equal to [-1, 1]. Set the box on.

b. Use the command contour to generate 20 contours. Use the same range for x1 and x2 as in (a)

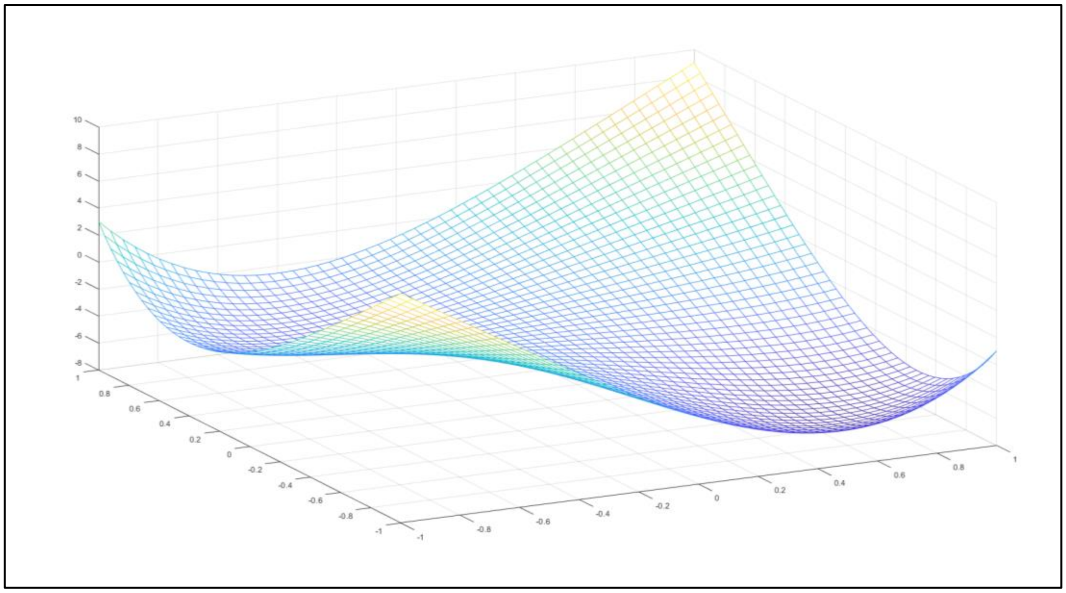

Fig 1. 3D plot of the function using the mesh function in MATLAB

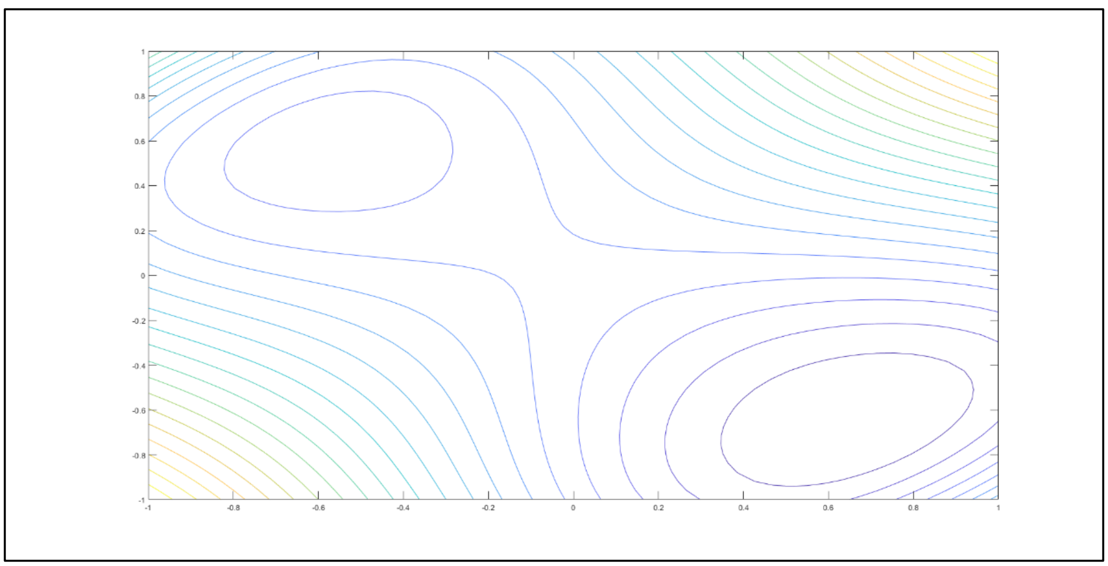

Fig 2. Contours of the function generated in MATLAB

MATLAB Code – Problem2.m

```
%problem 2
clc; close all;
%set range of x from -1 to 1
x = linspace(-1,1,50);
y = x;
% create a meshgrid for plotting the function
[x1,x2] = meshgrid(x,y);
% define the given function for 3d plotting
f = (x2-x1).^4+12.*x1.*x2-x1+x2-3;
box on; mesh(x1,x2,f); %plot the function
figure;
% plot contours of the function
contour(x1,x2,f, 20)
```

Problem 3

Minimize the above function using the method of the gradient descent when α = 0.02 and locate these points on the level sets of f. Connect the successive points with lines or lines with arrows to show clearly the progression of the optimization process. Use two staring points,

and

Obtain the sequence of points using the steepest descent method and locate these points on the level sets of f

Solution: *MATLAB Code – Problem3.m*

```
clc; clear all; close all;

%Gradient descent
syms x1 x2;
f = @(x1, x2) (x2-x1).^4+12.*x1.*x2-x1+x2-3; %declare the function in terms of
x1, x2
%Initializations for the sequence of points
X_init_array1{1} = [0.55, 0.7]; %starting point 1
X_init_array1{2} = [0,0];
X_init_array2{1} = [-0.9 -0.5]; %starting point 2
X_init_array2{2} = [0,0];
%Gradient descent from the two starting points
X_init_array1 = gradDesc(f, X_init_array1);
X_init_array2 = gradDesc(f, X_init_array2);

%Steepest gradient method
X_init_array3{1} = [0.55, 0.7]; %starting point 1
X_init_array3{2} = [0,0];
X_init_array3 = steepestGrad(f, X_init_array3);

X_init_array4{1} = [-0.9,-0.5]; %starting point 1
X_init_array4{2} = [0,0];
X_init_array4 = steepestGrad(f, X_init_array4);

%plotting the contours/ level sets for the function to plot sequence later
x = linspace(-1,1,50);
y = x;
[x1,x2] = meshgrid(x,y);
f = (x2-x1).^4+12.*x1.*x2-x1+x2-3;
% box on; mesh(x1,x2,f); %plotting the function
```

```
contour(x1,x2,f, 20); hold on;
for i = 1:length(X_init_array1)
    px(i) = X_init_array1{i}(1);
    py(i) = X_init_array1{i}(2);
end
hold on;
plot(px, py, '^-'); %plot sequence of points starting from starting point1

figure;
contour(x1,x2,f, 20); hold on;
for i = 1:length(X_init_array2)
    px1(i) = X_init_array2{i}(1);
    py1(i) = X_init_array2{i}(2);
end
hold on;
plot(px1, py1, 'x-'); %plot sequence of points starting from starting point2

figure;
contour(x1,x2,f, 20); hold on;
for i = 1:length(X_init_array3)
    px2(i) = X_init_array3{i}(1);
    py2(i) = X_init_array3{i}(2);
end
hold on;
plot(px2, py2, 'x-'); %plot sequence of points starting from starting point1
using sd

figure;
contour(x1,x2,f, 20); hold on;
for i = 1:length(X_init_array4)
    px3(i) = X_init_array4{i}(1);
    py3(i) = X_init_array4{i}(2);
end
hold on;
plot(px3, py3, 'x-'); %plot sequence of points starting from starting point1
using sd
```

MATLAB Code – gradDescent.m

```
function [X_init_array1] = gradDesc(f, X_init_array1)
%GRADDESC Summary of this function goes here
%   Detailed explanation goes here
    syms x1 x2;
    grad = {};
    alpha = 0.02;
    epsilon = .001;
%     Initialize first gradients and 2nd element
    grad{1} = (double(subs(gradient(f(x1, x2), [x1 x2]), {x1,x2},
X_init_array1(1))))';
    X_init_array1{2}= X_init_array1{1}-alpha*grad{1};
    k = 2;
%     Iterate till the norm of consecutive points becomes less than epsilon
    while(norm((X_init_array1{k-1}-X_init_array1{k}),2)>epsilon)
        grad{k} = (double(subs(gradient(f(x1, x2), [x1 x2]), {x1,x2},
X_init_array1(k))))';
        X_init_array1{k+1}= X_init_array1{k}-alpha*grad{k};
        k = k+1;
```

```
    end
end
```

MATLAB Code – steepestGrad.m

```
function [X_init_array1] = steepestGrad(f, X_init_array1)
%Steepest Gradient - Used to calculate the minima using the steepest
%gradient method
%   The function takes f - The objective function and the Initial point
%   from the sequence as the input and gives out an array of sequence of
%   points to reach the minima as output. At each step Line search is used
%   to find the optimum step size
    syms x1 x2;
    epsilon = .001;
%     Initialize first gradients and 2nd element
    grad{1} = double(subs(gradient(f(x1, x2), [x1 x2]), {x1,x2},
X_init_array1(1)))';
    syms alph;
    searchDir = -gradient(f(x1, x2), [x1 x2]); %search dir in direction of
negative grad
    fx_ag = f(x1+alph*searchDir(1), x2+alph*searchDir(2)); %f(x-alpha*g)
    f_alph = subs(fx_ag, [x1, x2], {X_init_array1(1)}); %f(x1-alph*g1)
    f_dash_alph = diff(f_alph, alph); %f'(x1-alph*g1)
    f_ddash_alph = diff(f_dash_alph, alph); %f''(x1-alph*g1)
    %initialize alpha values to arbitrary levels
    alpha{1} = 0.02;
    alpha{2} = alpha{1}-double(subs((f_dash_alph/f_ddash_alph), alph, alpha{1}));
    %find 2nd point based on initial alpha
    X_init_array1{2}= X_init_array1{1}-alpha{1}*grad{1};
    k = 2;
    %     Iterate till the norm of consecutive points becomes less than epsilon
    while(k<100&& norm((X_init_array1{k}-X_init_array1{k-1}),2)>epsilon)
        syms alph;
        %find the kth gradient, f(xk-alph*gk), f'(xk-alph*gk),
        %f''(xk-alph*gk)
        grad{k} = (double(subs(gradient(f(x1, x2), [x1 x2]), {x1,x2},
X_init_array1(k))))';
        f_alph = subs(fx_ag, [x1, x2], {X_init_array1(k)});
        f_dash_alph = diff(f_alph, alph);
        f_ddash_alph = diff(f_dash_alph, alph);
        %initialize alpha for line search using NM
        alpha_new{1} = alpha{k-1};
        alpha_new{2} = alpha{k};
        i = 2;
%         while loop to evaluate optimum alpha to minimize f(x-alpha*g)
%         using Newtons method (Line search)
        while(norm(alpha_new{i}-alpha_new{i-1})>0.001)
%         this while loop can be removed and the statement below can be
%         used to get alpha after one iteration of NM. And the point
%         converges to minima using this approach. Not sure if it is the
%         correct way.
%             alpha{k} = alpha{k-1}- double(subs((f_dash_alph/f_ddash_alph),
alph, alpha{k-1}));
            alpha_new{i+1} = alpha_new{i}-
double(subs((f_dash_alph/f_ddash_alph), alph, alpha_new{i}));
            i = i+1;
        end
        alpha{k+1} = alpha_new{i}; %update alpha with the one found using NM
```

```
        X_init_array1{k+1}= X_init_array1{k}-alpha{k+1}*grad{k}; %update point
using the new found alpha
        k = k+1;
    end
end
```

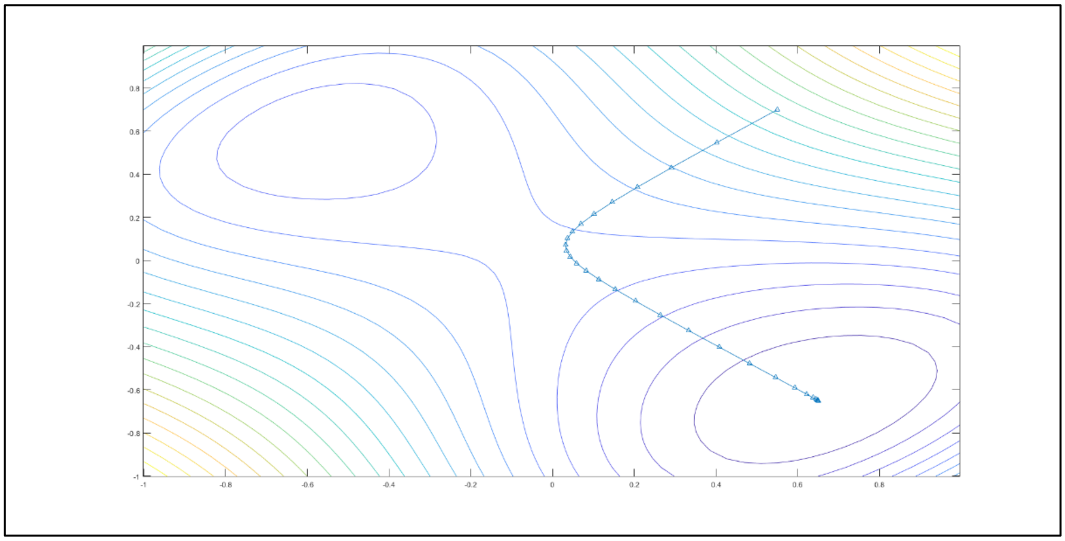

Fig 3. The sequence of points plotted on level set of the function for starting point $[0.55, 0.7]^T$

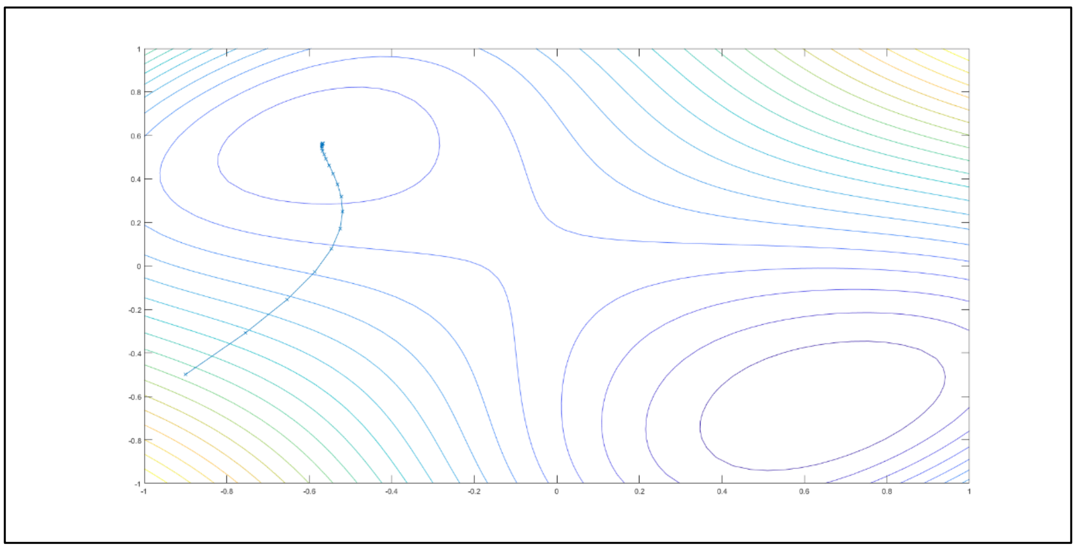

Fig 4. The sequence of points plotted on level set of the function for starting point $[-0.9,-0.5]^T$

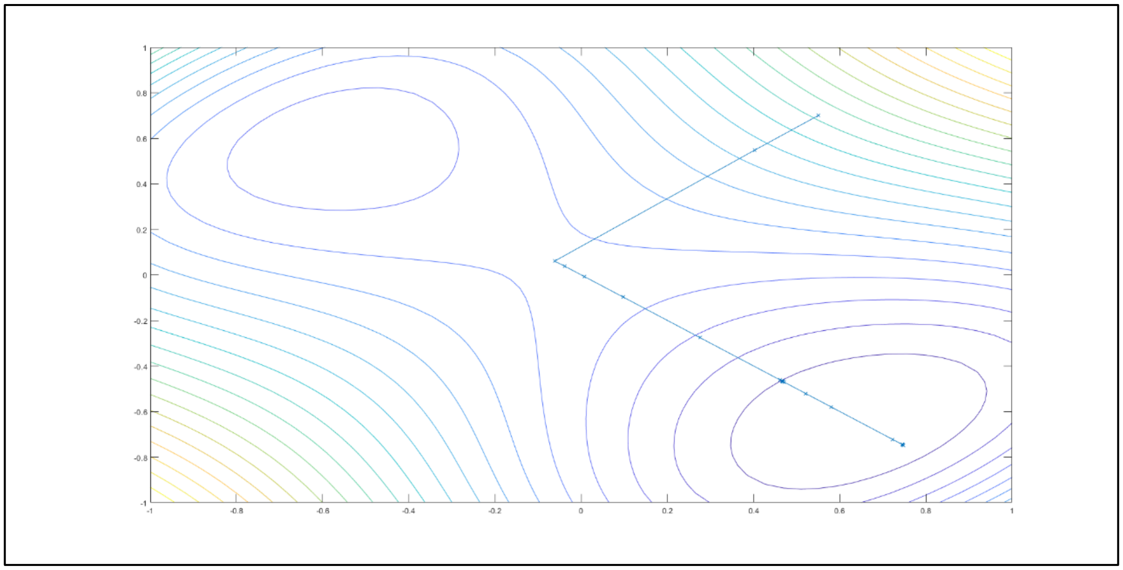

Fig 5. The sequence of points plotted on level set of the function for starting point $[.55,.7]^T$ using steepest descent

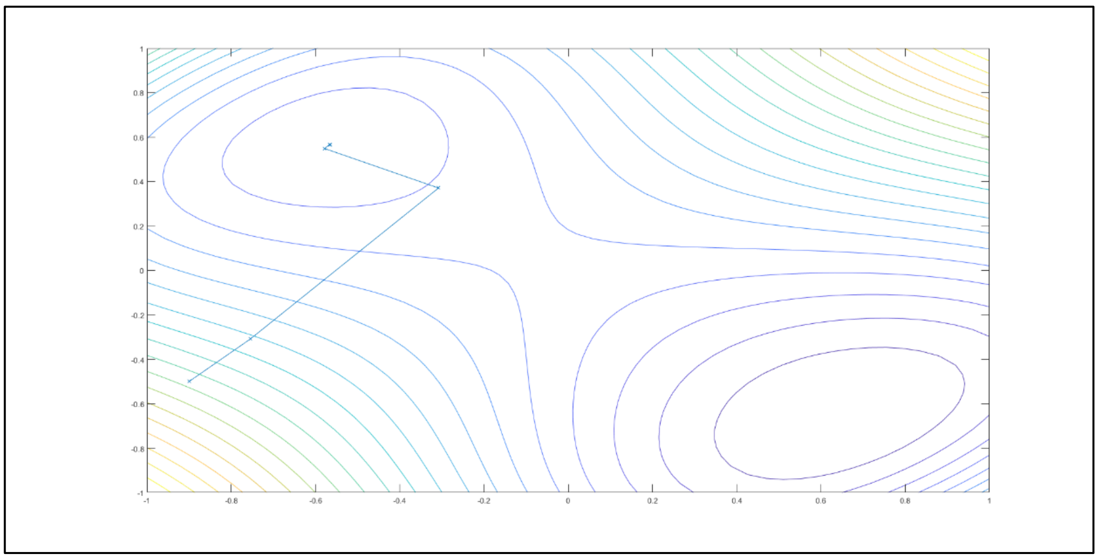

Fig 6. The sequence of points plotted on level set of the function for starting point $[-0.9,-0.25]^T$ using steepest descent

As we can see for the first starting point, while using the Steepest descent method we get caught in a 'narrow valley' region due to which, it becomes difficult to reach the minimizer.
For the second point, due to the optimum step size selection, the function reaches the local minimizer very quickly as we can see in Figure 6.

Problem 4

Minimize the above function using Newton's method. Locate the points on Level sets of f.

The given function has the very first Hessian which is not pd. So, the Newtons method won't converge for this function. The code is written to handle this condition. However, we can still run the code to visualize the sequence of points on the level sets

Solution:

```
% problem4;
clear all; clc; close all;
% Use newtons method to minimize the given function
syms x1 x2;
%define the function in terms of symbolic variables x1 and x2
f = (x2-x1).^4+12.*x1.*x2-x1+x2-3;
X_init_array1{1} = [1,-.8];
X_init_array1{2} = [0,0]; %initialize second point in the sequence
% alpha = 0.02;
epsilon = .001; %set epsilon to some small value
%calculate the gradient and hessian at the initial point x0
grad{1} = (double(subs(gradient(f, [x1 x2]), {x1,x2}, X_init_array1(1))));
hess{1} = double(subs(hessian(f, [x1 x2]), {x1,x2}, X_init_array1(1)));
%The conditions for the positive definiteness can be removed to check
%the updated sequence we get through the newtons method. But as this
%example has no pd matrix the sequence obtained by using the Hessian as the
%step size does not converge to the minimizer.
if (eig(hess{1})>0) %check for the kth Hessian to be pd
    %     update point 2 with hess{1} as step size
    X_init_array1{2}= X_init_array1{1}-(pinv(hess{1})*grad{1})';
    k = 2;
    %loop to update the points while calculating the hessian at every step
    %to set the step size
    while(norm((X_init_array1{k-1}-X_init_array1{k}),2)~=0)
        grad{k} = (double(subs(gradient(f, [x1 x2]), {x1,x2},
X_init_array1(k))));
        hess{k} = double(subs(hessian(f, [x1 x2]), {x1,x2}, X_init_array1(k)));
        if eig(hess{k})>0 %check if hess{k} is pd
            %               update kth point with hess{k} as step size
            X_init_array1{k+1}= X_init_array1{k}-(pinv(hess{k})*grad{k})';
            k = k+1;
        else
            sprintf('Hessian at the %d th is not pd. Descent not guaranteed.',
k);
        end
    end
else
    disp('Hessian at the first point is not pd. Descent not guaranteed.');
end

% plotting the contours/ level sets for the function to plot sequence later
x = linspace(-1,1,50);
y = x;
[x1,x2] = meshgrid(x,y);
f = (x2-x1).^4+12.*x1.*x2-x1+x2-3;
% box on; mesh(x1,x2,f); %plotting the function
contour(x1,x2,f, 20); hold on;
for i = 1:length(X_init_array1)
```

```
    px(i) = X_init_array1{i}(1);
    py(i) = X_init_array1{i}(2);
end
hold on;
plot(px, py, 'x-'); %plot sequence of points starting from starting point1
```

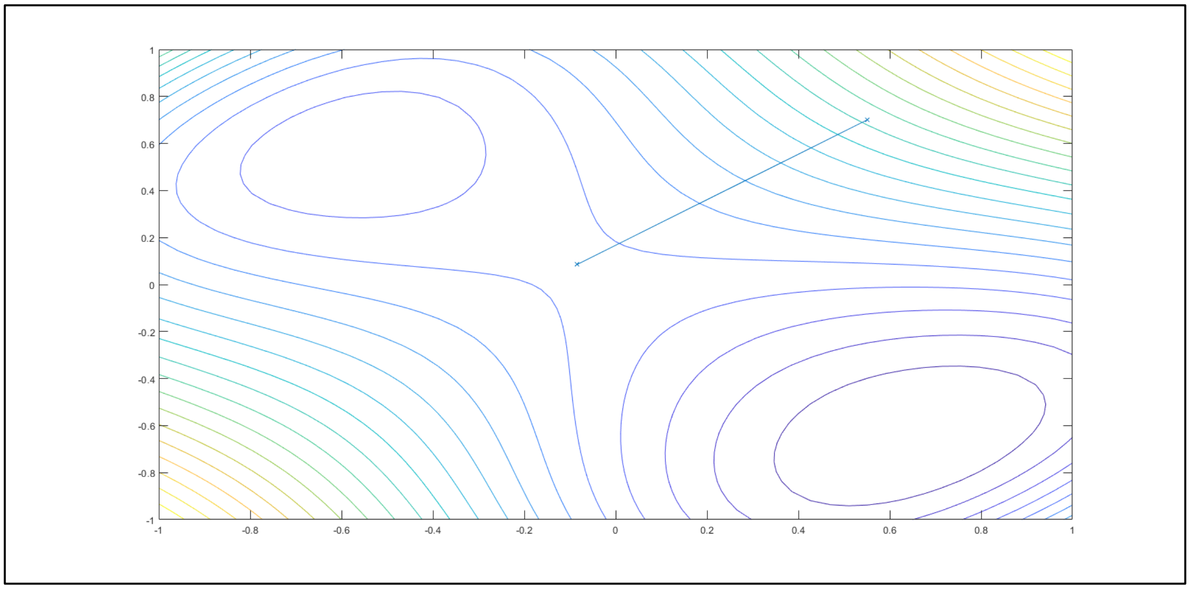

Figure 7. Sequence of point obtained by ignoring the fact that the Hessian is pd.

The plot shown above shows that if we try to apply the Newtons method to a point away from the minimizer and to a function whose Hessian is not positive definite, the method fails to converge to a minimizer. We can see that the method reaches the saddle point and stops there.

However, for sanity check if we check a starting point near to minimizer, the Hessian comes out to be pd and converges to the minima.

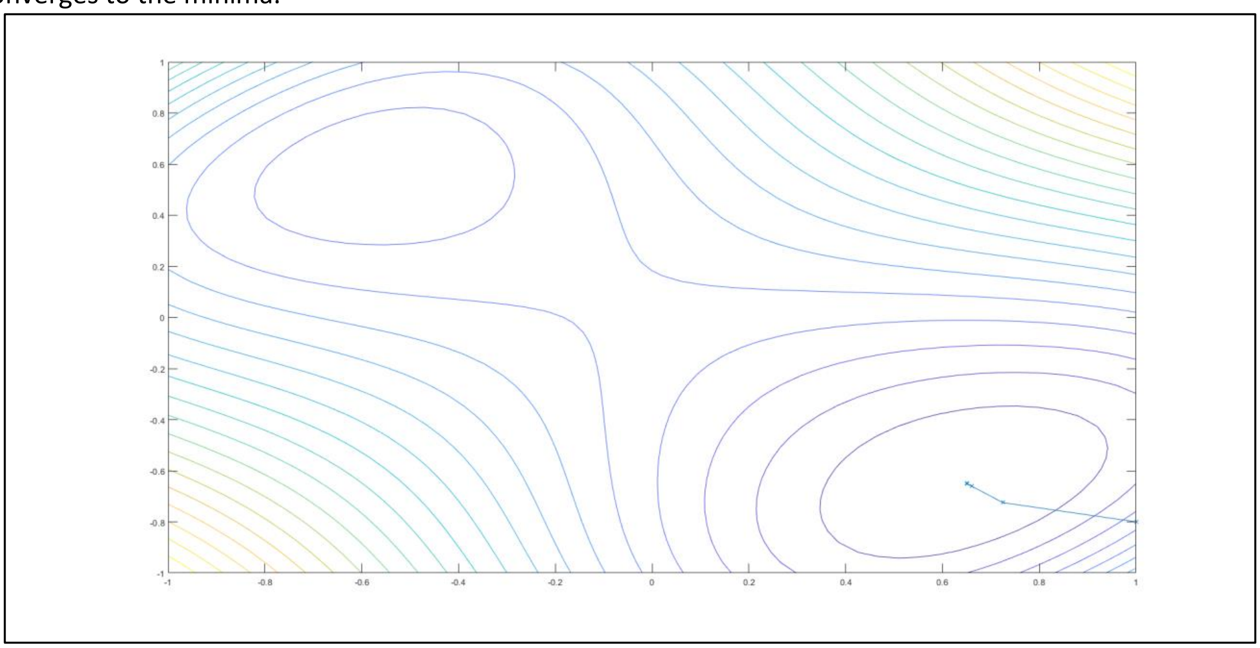

Figure 8. Sequence of points for a point near the minimizer