

**Objective\_function\_hw2\_1.m :**

function [f,df,H]=objective\_function\_hw2\_1(x)

f=x(1).^2+2.\*x(1).\*x(2)+4.\*x(1).\*x(3)+3.\*x(2).^2+2.\*x(2).\*x(3)+5.\*x(3).^2;

df=zeros(3,1);

df(1)=2.\*x(1)+2.\*x(2)+4.\*x(3);

df(2)=2.\*x(1)+6.\*x(2)+2.\*x(3);

df(3)=4.\*x(1)+2.\*x(2)+10.\*x(3);

H=[2 2 4; 2 6 2; 4 2 10];

(a)

**hw2\_1\_steepest\_descent\_direction\_method.m :**

x0=[1;1;1];

max\_step=100;

iteration\_x=zeros(3,max\_step);

iteration\_f=zeros(1,max\_step);

iteration\_x(:,1)=x0;

for ii=1:max\_step

[iteration\_f(ii),df,H]=objective\_function\_hw2\_1(iteration\_x(:,ii));

if ii>1 && abs(iteration\_f(ii)-iteration\_f(ii-1))<1e-4 || ii==max\_step

f=iteration\_f(ii);

x=iteration\_x(:,ii);

fprintf("steps = %d \n", ii-1)

fprintf("f = %s \n",f)

fprintf("x1 = %s , x2 = %s , x3 = %s \n",x(1),x(2),x(3))

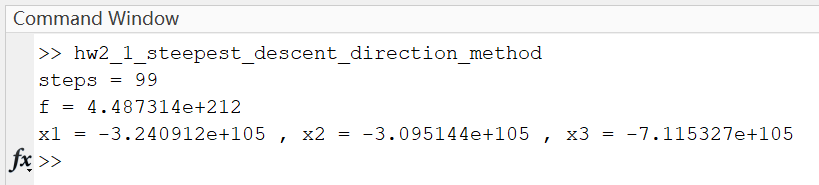
break

end

iteration\_x(:,ii+1)=iteration\_x(:,ii)-df;

end

**result :**



Without the coefficient α, we can’t find the optima in this objective function. Since the gradient is too big to make the step correct.

(b)

**hw2\_1\_Newtons\_method.m :**

x0=[1;1;1];

max\_step=100;

iteration\_x=zeros(3,max\_step);

iteration\_f=zeros(1,max\_step);

iteration\_x(:,1)=x0;

for ii=1:max\_step

[iteration\_f(ii),df,H]=objective\_function\_hw2\_1(iteration\_x(:,ii));

if ii>1 && abs(iteration\_f(ii)-iteration\_f(ii-1))<1e-4 || ii==max\_step

f=iteration\_f(ii);

x=iteration\_x(:,ii);

fprintf("steps = %d \n", ii-1)

fprintf("f = %s \n",f)

fprintf("x1 = %s , x2 = %s , x3 = %s \n",x(1),x(2),x(3))

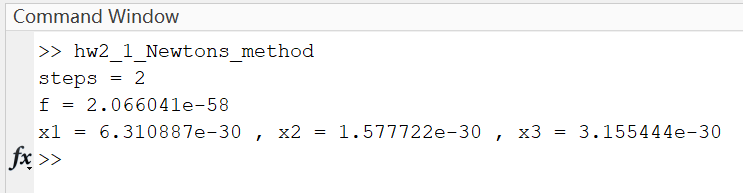
break

end

iteration\_x(:,ii+1)=iteration\_x(:,ii)-df;

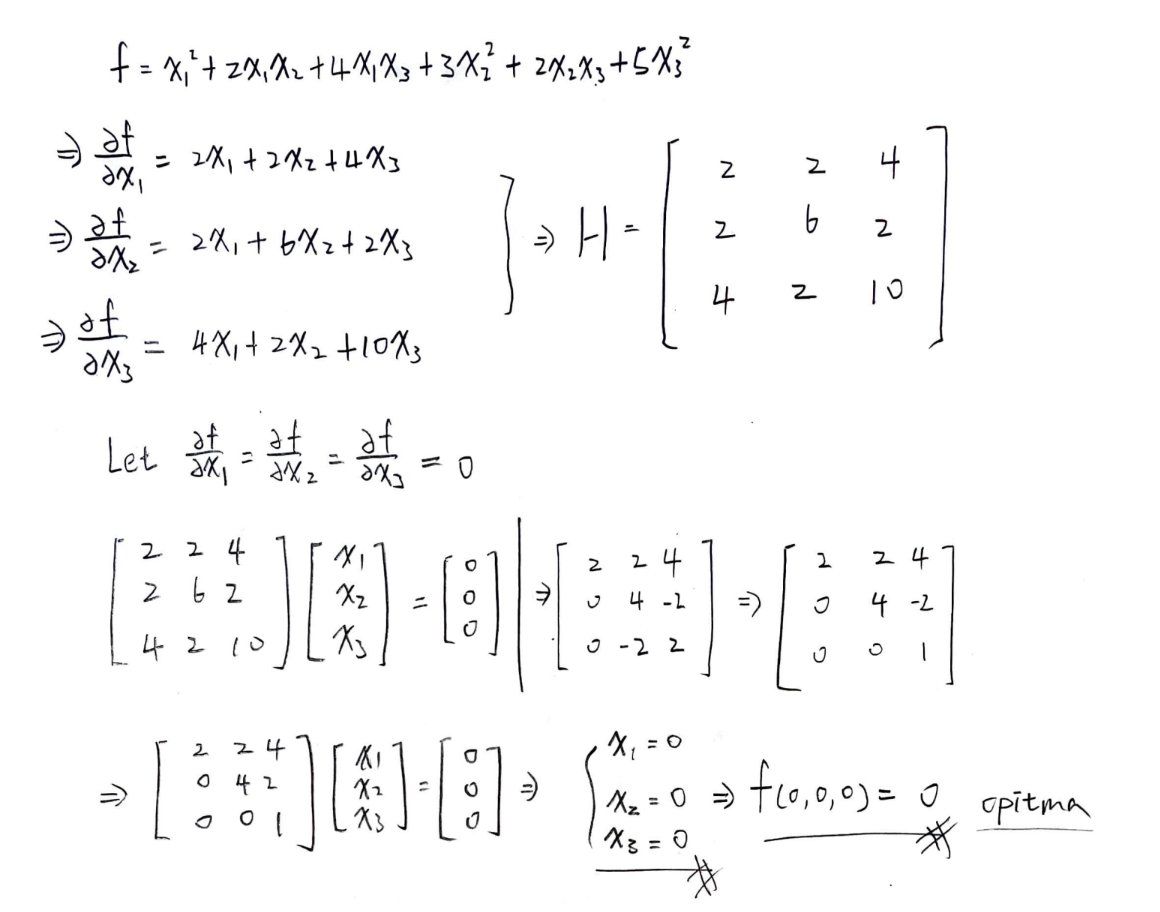
end

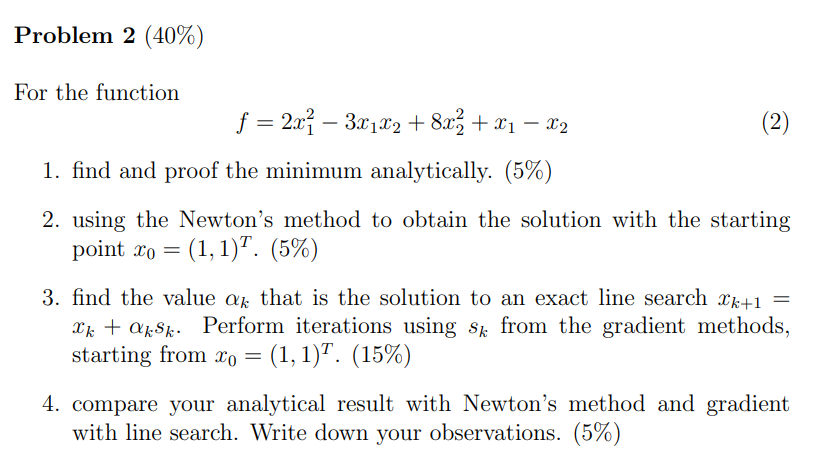
**result :**



Since the objective function is quadratic, Newton’s method use only two steps to find the optima.

(c)





**objective\_function\_hw2\_2.m :**

function [f,df,H]=objective\_function\_hw2\_2(x)

f=2\*x(1)^2-3\*x(1)\*x(2)+8\*x(2)^2+x(1)-x(2);

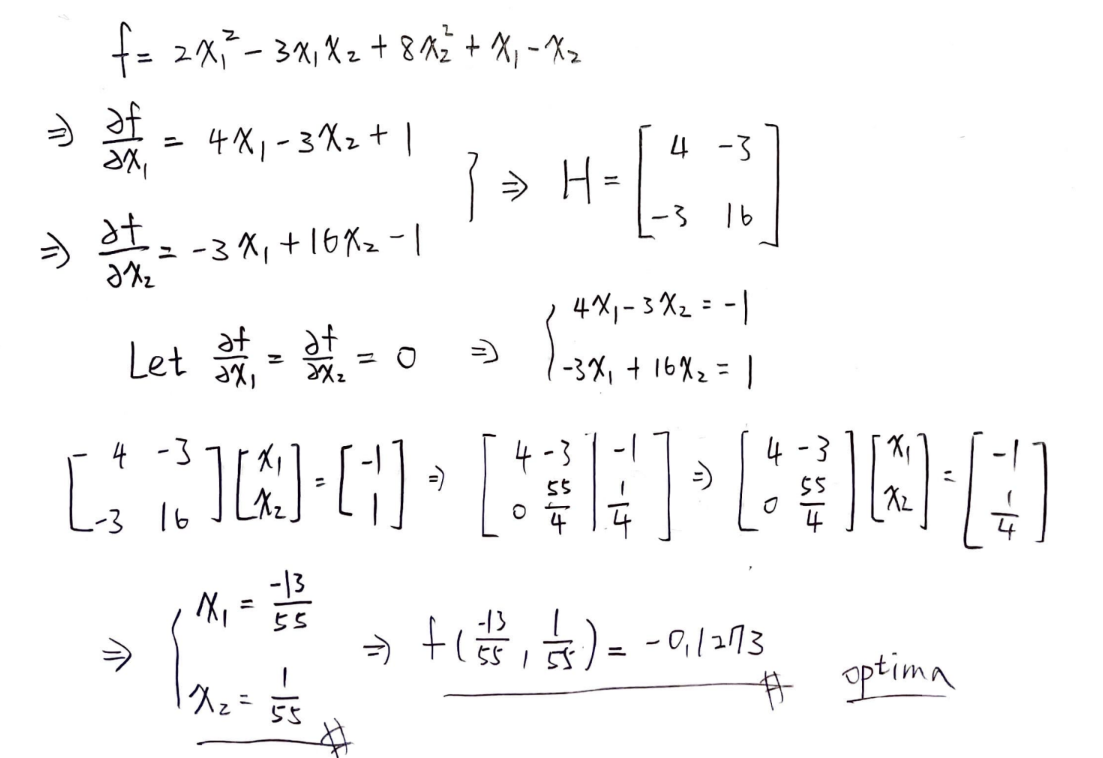
df=zeros(2,1);

df(1)=4\*x(1)-3\*x(2)+1;

df(2)=-3\*x(1)+16\*x(2)-1;

H=[4 -3; -3 16];

(a)



(b)

**hw2\_2\_Newtons\_method.m :**

x0=[1;1];

max\_step=100;

iteration\_x=zeros(2,max\_step);

iteration\_f=zeros(1,max\_step);

iteration\_x(:,1)=x0;

f=0;

x=zeros(3,1);

for ii=1:max\_step

[iteration\_f(ii),df,H]=objective\_function\_hw2\_2(iteration\_x(:,ii));

if ii>1 && abs(iteration\_f(ii)-iteration\_f(ii-1))<1e-9 || ii==max\_step

f=iteration\_f(ii);

x=iteration\_x(:,ii);

fprintf("steps = %d \n", ii-1)

fprintf("f = %s \n",f)

fprintf("x1 = %s , x2 = %s \n",x(1),x(2))

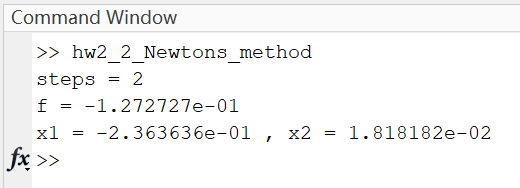
break

end

iteration\_x(:,ii+1)=iteration\_x(:,ii)-H^-1\*df;

end

**result :**



Since the objective function is quadratic, Newton’s method use only two steps to find the optima.

(c)

**hw2\_2\_optimize\_apha\_method.m :**

x0=[1;1];

max\_step=100;

iteration\_x=zeros(2,max\_step);

iteration\_f=zeros(1,max\_step);

iteration\_apha=zeros(2,max\_step);

iteration\_x(:,1)=x0;

a1=linspace(0,1,1000);

a2=linspace(0,1,1000);

n=size(a1,2);

try\_apha=cell(n,n);

for aa=1:n

for bb=1:n

try\_apha{aa,bb}=[a1(aa); a2(bb)];

end

end

for ii=1:max\_step

[iteration\_f(ii),df,H]=objective\_function\_hw2\_2(iteration\_x(:,ii));

if ii>1 && abs(iteration\_f(ii)-iteration\_f(ii-1))<1e-15 || ii==max\_step

f=iteration\_f(ii);

x=iteration\_x(:,ii);

fprintf("steps = %d \n", ii-1)

fprintf("f = %s \n",f)

fprintf("x1 = %s , x2 = %s \n",x(1),x(2))

break

end

for jj=1:n\*n

try\_x=iteration\_x(ii)-try\_apha{jj}.\*df;

try\_f=objective\_function\_hw2\_2(try\_x);

if jj==1

step\_f=try\_f;

iteration\_apha(:,ii)=try\_apha{jj};

elseif jj>1 && try\_f<step\_f

step\_f=try\_f;

iteration\_apha(:,ii)=try\_apha{jj};

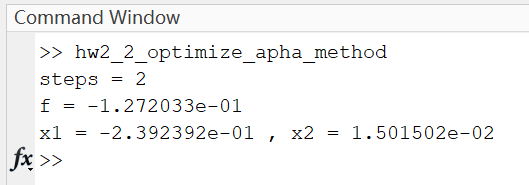
end

end

iteration\_x(:,ii+1)=iteration\_x(:,ii)-iteration\_apha(:,ii).\*df;

end

**result :**



The starting point is really close to the optima point, so it didn’t iteration to many times but try to figure out the correct α to step forward.

(d)

Although initial point [1,1]T is relatively close to the optima point, the gradient at that point is still too big to get to the lower position. In other words, if the apha\_k is [1 1], the first steps will make the algorithm exceed the optima and go to a higher point. Thus, the apha\_k must smaller than 1 in the first step. As the result, the line search method also use only one step to find the optima. If I divide the