question 1.

In the example 1.1, we can know that the objective function is

$$f'(x) = (130c - 2 - 2cx)e^{cx} - 0.45.$$

Then we can use One-Variable Newton's Method to solve this optimal problem. We can find the sensitivity of x to c.

c	x
0. 023	14. 5747342001552
0. 0235	15. 8503144188681
0. 024	17. 1201578827516
0. 0245	18. 3386913132619
0. 025	19. 540333762464
0. 0255	20. 5629246411526
0. 026	21. 6216894846366
0. 0265	22. 5997079283244
0. 027	23. 5664300977403

Table 1: Sensitivity of best time to sell x to growth rate parameter c for the pig problem with nonlinear weight model.

Next is the codes to achieve the result using MATLAB.

```
clc,clear
syms x;
res=zeros(9,1);
c=(0.023:0.0005:0.027)';
i=1;
for n=0.023:0.0005:0.027
    f=(130*n-2-2*n*x)*(exp(1)^(n*x))-0.45;      %the objective function
    x0=0;
    eps=0.1;
    res(i,1)=Newton_1(f,x,x0,eps,100);
    i=i+1;
end
table(c,res,'VariableNames',{'c' 'x'})          %draw a table of c and x

```

Figure 1: Main codes 1 in MATLAB

```
function res=Newton_1(f,x,x0,eps,N)
df=diff(f);
for i=1:N
    x1=x0-subs(f,x,x0)/subs(df,x,x0);   %solution after iteration
    if abs(x1-x0)<eps
        break;                          %solution good enough
    end
    x0=x1;
end
res=double(x0);
end
```

Figure 2: Function codes 1 in MATLAB

question 2.

In the example 2.1, we need to find the best location for the new facility. We can find different solution in different iteration times N. We can determine if the solution approaching to the real solution when N becoming larger.

```
clc,clear

Num=0:100:2000;
avge=zeros(1,21);
i=1;
for N=0:100:2000                %different times N
    r=RandomS(0,6,0,6,N);
    rr=[1.66,2.73,6.46];        %real solution
    error=abs(r-rr);
    avge(1,i)=sum(error)/3;     %mean error
    i=i+1;
end
plot(Num,avge);
```

Figure 3: Main codes 2 in MATLAB

I set a for loop to determine the answer after different N's iteration. Then I plot a graph which x-axis means N, and y-axis are **the mean difference to the real answer** (which is $[1.616749, 2.765987, 6.46298]$). Seeing Figure 3. This solution is found since I let the iteration times to become 10^9.

There is also some wave when $N \to \infty$ in Figure 4. That is because random search method is not accurate absolutely. There must be some points which is more optimal than founded point, since it takes random points.

We can find that the mean difference is converging to 0 when N becoming larger and larger. **So we can claim that there is a negetive ralationship between error and iteration times N.**

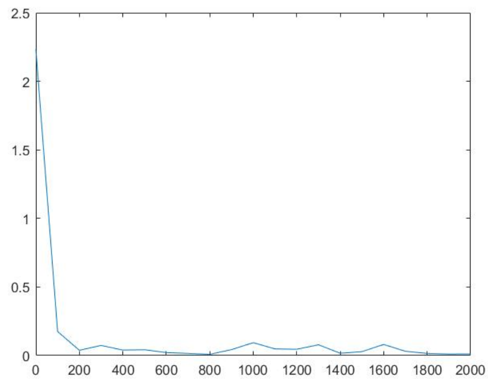

Figure 4: Graph of N and mean difference

Here is the codes to achieve random search method.

The objective function is the function z_m in Figure 5..

```
HW3_1.m  ×  Newton_1.m  ×  RandomS.m  ×  +
function r=RandomS(a, b, c, d, N)
xm=a+(b-a)*rand(1);                          %random number of x, y, z
ym=c+(d-c)*rand(1);

zm=3.2+1.7*(6*(((xm-1)^2+(ym-5)^2)^(1/2))^0.91+8*(((xm-3)^2+(ym-5)^2)^(1/2))^0.91+...
    8*(((xm-5)^2+(ym-5)^2)^(1/2))^0.91+21*(((xm-1)^2+(ym-3)^2)^(1/2))^0.91+6*(((xm-3)^2+...
    (ym-3)^2)^(1/2))^0.91+3*(((xm-5)^2+(ym-3)^2)^(1/2))^0.91+18*(((xm-1)^2+...
    (ym-1)^2)^(1/2))^0.91+8*(((xm-3)^2+(ym-1)^2)^(1/2))^0.91+6*(((xm-5)^2+(ym-1)^2)^(1/2))^0.91)/84;
for n=1:N                                    %zm is the objective funtion
    x=a+(b-a)*rand(1);
    y=c+(d-c)*rand(1);

    z=3.2+1.7*(6*(((x-1)^2+(y-5)^2)^(1/2))^0.91+8*(((x-3)^2+(y-5)^2)^(1/2))^0.91+...
        8*(((x-5)^2+(y-5)^2)^(1/2))^0.91+21*(((x-1)^2+(y-3)^2)^(1/2))^0.91+6*(((x-3)^2+...
        (y-3)^2)^(1/2))^0.91+3*(((x-5)^2+(y-3)^2)^(1/2))^0.91+18*(((x-1)^2+...
        (y-1)^2)^(1/2))^0.91+8*(((x-3)^2+(y-1)^2)^(1/2))^0.91+6*(((x-5)^2+(y-1)^2)^(1/2))^0.91)/84;
    if z<zm                                  %whether better or not
        xm=x;
        ym=y;
        zm=z;
    end
end
r=[xm, ym, zm];                              %output the solution
end
```

Figure 5: Function codes 2 in MATLAB

question 3.

In the example 2.2, there are 2 variables in order to solve. By using Multi-Variables Newton's Method we can finally find the best solution in N times iteration, and this solution would have limited error to the real solution.

Here is the codes in MATLAB.

```
clc,clear

f1=@(x1,x2) 15.5*(x1.^(-0.5))-8+1.3*(x2.^(-0.2))- 0.064*x2*(x1.^(-1.08));   %2 objective function
f2=@(x1,x2) 9*(x2.^(-0.4))-5+0.8*(x1.^(-0.08))-0.26*x1*(x2.^(-1.2));

df1_1=@(x1,x2) 0.06912*x2*(x1^(-2.08))-7.75*(x1^(-1.5));   %4 partial function
df1_2=@(x1,x2) -0.064*(x1^(-1.08))-0.26*(x2^(-1.2));        % of variable x and y
df2_1=@(x1,x2) -0.26*(x2^(-1.2))-0.064*(x1*(-1.08));
df2_2=@(x1,x2) 0.312*x1*(x2^(-2.2))-3.6*(x2^(-1.4));

x1=(0.5:0.5:8.5)';
step=zeros(17,1);
N=10000;                                                     %max iteration times
j=1;
for i=0.5:0.5:8.5
    x0=[i;3.1];                                              %initial point
    [x,n]=Newton_2(x0,df1_1,df1_2,df2_1,df2_2,f1,f2,N);      %Newton Method
    step(j,1)=n;
    j=j+1;
end
table(x1,step,'VariableNames',{'x1' 'n'})
plot(x1,step)
axis([0,9,6,11]);
xlabel('initial number x_1');
ylabel('iteration times');
```

Figure 6: Main codes 3 in MATLAB

This program needs to input the partial function respected to variables x_1 and x_2. And N is the max iteration times while n is the real iteration times. In order to determine the relationship between initial point and iteration times, we must use different initial point. Obviously, we can find it between x_1 and n, and we can extend it to x_2. In my codes, I calculate x_1.

HW3_1.m | Newton_1.m | HW3_2.m | RandomS.m | HW3_3.m | Newton_2.m

```
1  function [x,n] = Newton_2(x0, df1_1, df1_2, df2_1, df2_2, f1, f2, N)
2      x=x0;
3      n=1;
4      eps=0.00001;                                  %error setting
5
6      for i = 1 : N
7          q=df1_1(x(1, 1), x(2, 1));
8          r=df1_2(x(1, 1), x(2, 1));
9          s=df2_1(x(1, 1), x(2, 1));
10         t=df2_2(x(1, 1), x(2, 1));
11         u=-f1(x(1, 1), x(2, 1));
12         v=-f2(x(1, 1), x(2, 1));
13         D=q*t-r*s;
14         if(abs(x(1,1)-x(1,1)+(u*t-v*r)./D)<eps && abs(x(2,1)-x(2,1)+(q*v-s*u)./D)<eps)
15             break;                                %solution good enough
16         end
17
18         x(1,1)=x(1,1)+(u*t-v*r)./D;
19         x(2,1)=x(2,1)+(q*v-s*u)./D;
20         n=n+1;                                    %record the iteration times
21     end
22 end
```

Figure 7: Function codes 3 in MATLAB

In this codes, I set the error requirment to be $eps = 10^{-5}$. Seeing line 4.

We want to find differnt initial point is how to effect the iteration times n. In the main codes I'd drawn a $x_1 - n$ picture and a table which is

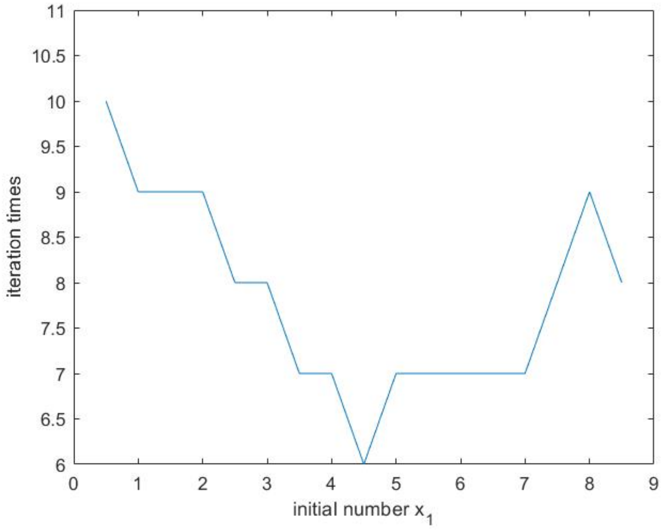

Figure 8: $x_1 - n$ picture using by MATLAB

x1	n
0.5	10
1	9
1.5	9
2	9
2.5	8
3	8
3.5	7
4	7
4.5	6
5	7
5.5	7
6	7
6.5	7
7	7
7.5	8
8	9
8.5	8

Table 2: Sensitivity of initial point x_1 to iteration times n for the question 3.

Through the program we can find the optimal point is approaching to

$$x_1^* = 4.68959, x_2^* = 5.85199$$

Then in the Table 2, we can find that if x_2 is a fixede initial point, the iteration times n is going to be larger when x_1 is far away from the optimal solution $x_1^* = 4.68959$. Since the iteration times n is going to lower between $x_1 \in [0, 8]$. This trend also fixed for x_2.

Also, there is a strange point when $x_1 = 8.5$, we can find that $n = 8 < 9$. After analysis I find that this leads to a wrong answer which is $x_1^* = 6.36335, x_2^* = 0.16817$. This is because the Hessian matrix is not positive definite. In other words, we need to find another matrix to approach Hessian matrix, then we can avoid this mistake. (Quasi-Newton Methods is better).

So we can claim that iteration times n become smaller when initial point is closer to the real solution.