Soft Computing Assignment 1

Mustafa Caner Sezer — 504191123

20/21 Spring Term

1 Question 1

In the first question, it was asked to find the minimum of the following cost function with 3 different techniques.

$$\min_x f(x) = (x-1)^2(x-2)(x-3)$$
$$0 \leq x \leq 4$$

However, before the start, it is beneficial to draw the cost function to gain knowledge about what we're going to face and get a better insight. The cost function in the corresponding interval can be seen as follows. It

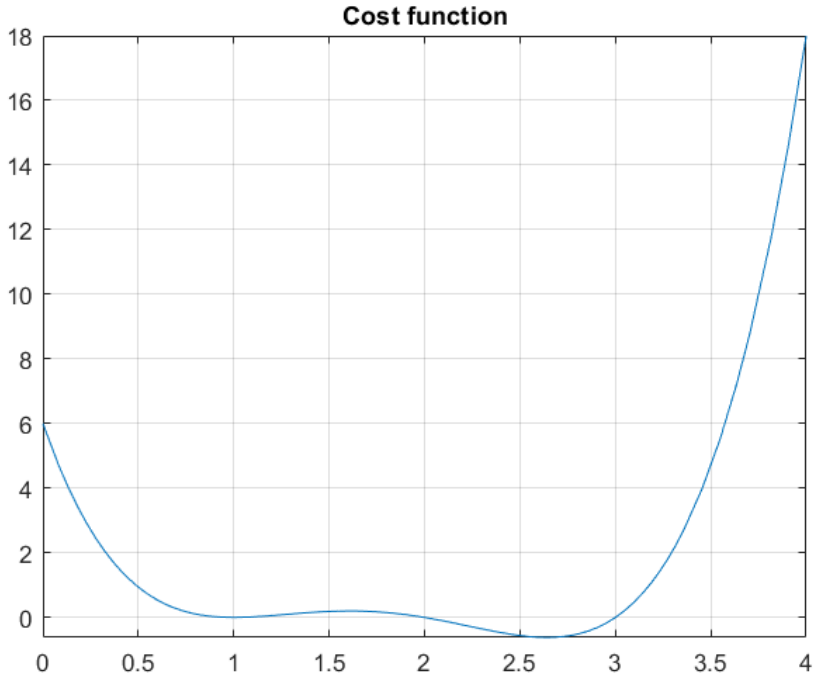

Figure 1: Cost Function within the given interval

is clear that within the interval, there is one local minimum and one global minimum. Therefore, the initial point will be important to reach to the global optimum point. The codes can be found in the attachment.

1.1 Newton-Raphson Method

The first method to find the optimum point will be Newton- Raphson method. In this method, the first and the second derivative of the cost function are taking into account. To start the algorithm, the tolerance was defined and the initial points 0.2, 0.5 and 2.5 were used.

1.1.1 Simulation for $x_0 = 0.2$

In the first initial point, it is clear that we are closer to the local minimum than the global minimum. Therefore, The algorithm converges to the local minimum which is 1. The cost function reaches to zero as expected and the gradient also reaches to zero. The algorithm converges to the local minimum in 6 steps.The simulations can be seen in the figure 2.

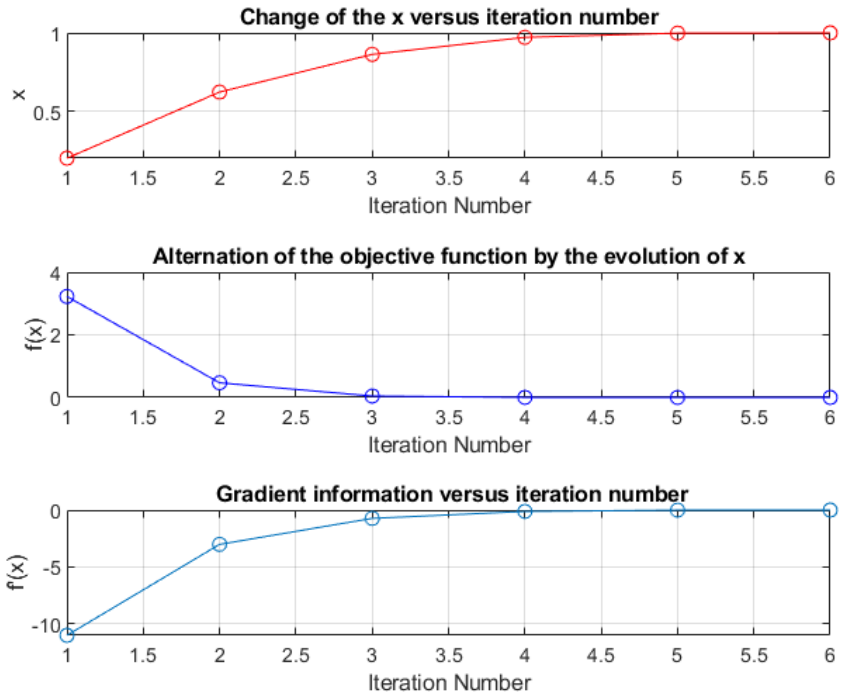

Figure 2: Results for $x_0 = 0.2$

1.1.2 Simulation for $x_0 = 0.5$

In the second initial point, we are still left hand side of the local minimum point. Thus, it means that it can be expected from algorithm to converge to the local minimum.In the figure 3, it is obvious that the algorithm converges to the local minimum while the cost function and the gradient reaches to zero. The algorithm converges in 6 steps.

1.1.3 Simulation for $x_0 = 2.5$

In the last initial point, we are now on the right hand side of the cost function which means we are closer to the global minimum than the local minimum.The x starts from 2.5 and ends up in the global minimum which is a bit greater than 2.6. For some point, the algorithm misses the global point but then turns the sign of the gradient to positive and reaches to the global minimum. The algorithm converges in 5 steps. The results can be seen in the figure 4.

1.2 Bisection Method

In the bisection method, the same cost function was taken into account. The parameters were chosen as $\alpha_a = 1.8$ and $\alpha_b = 3$. This means that the algorithm will search the optimum point within this interval. Since the corresponding interval consists the global minimum, it can be expected from algorithm to converge to the global minimum. In the figure 5 the results can be seen. The algorithm converges in 15 steps which makes it a bit slower than the previous ones.

1.3 Golden Section Method

The golden-section search is a technique for finding an extremum of a function inside a specified interval like bisection algorithm. If there is only one extremum in the interval, it will find this extremum. However, when there are multiple extremum points it converges one of them or if there is no extremum point in the

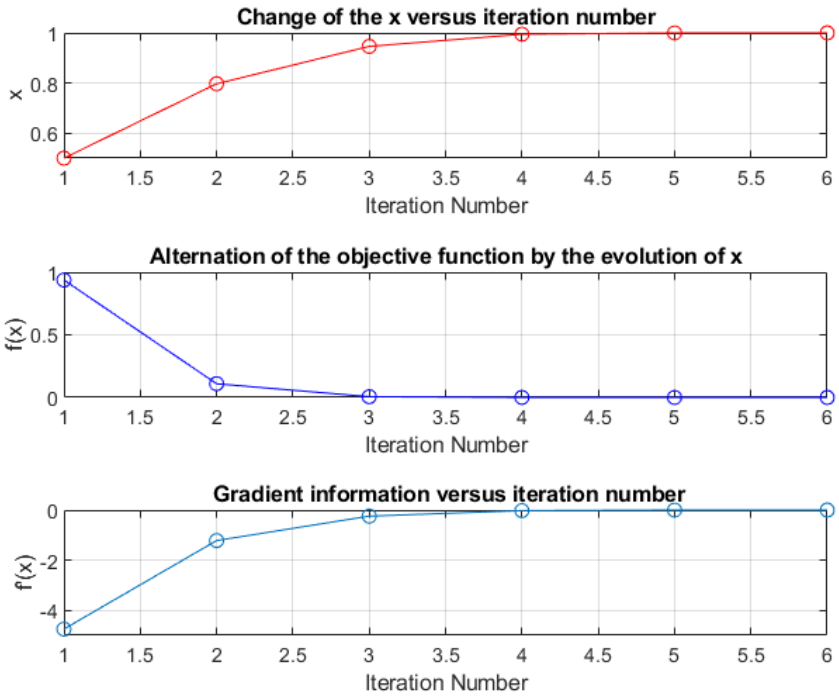

Figure 3: Results for $x_0 = 0.5$

given interval the algorithm can not converge.

The interval boundaries were chosen as $x^{low} = 0$,$x^{up} = 2$ and $\Delta x_{final} = 10^{-10}$. The algorithm was implemented in MATLAB and the results can be observed in the figure 6. It is clear that the algorithm converge to 1 and the value of the objective function reaches to zero.

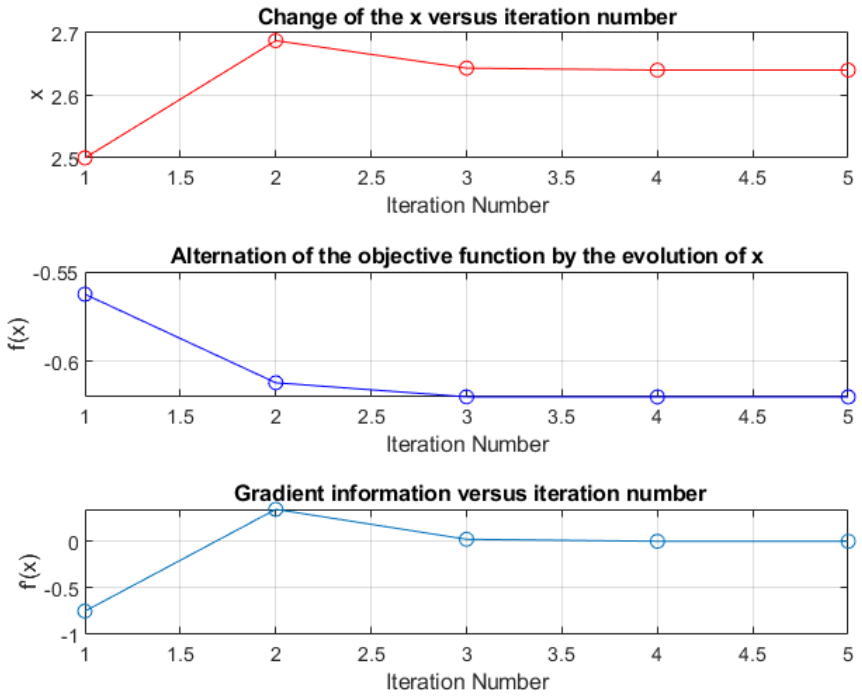

Figure 4: Results for $x_0 = 2.5$

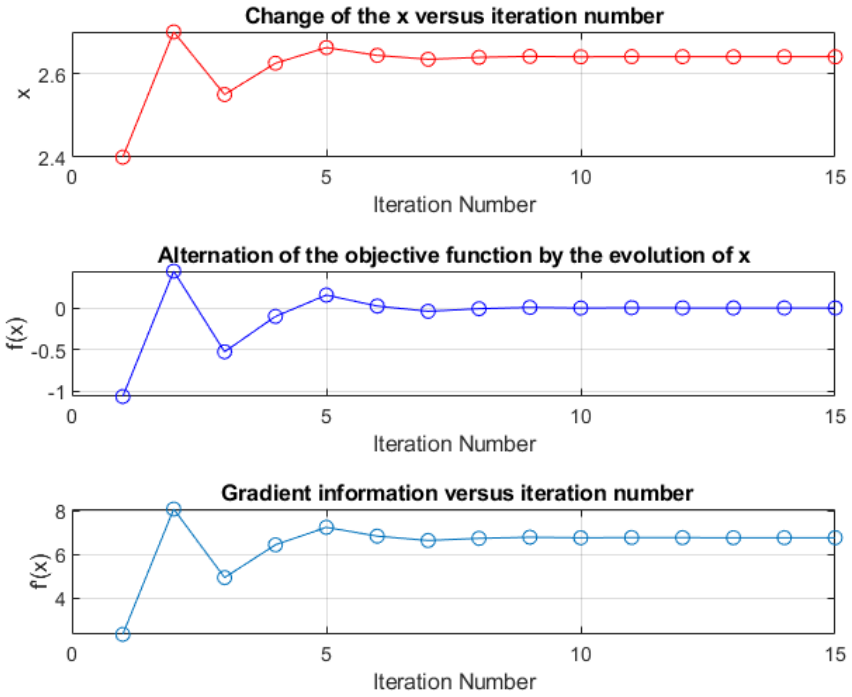

Figure 5: Results for the bisection algorithm

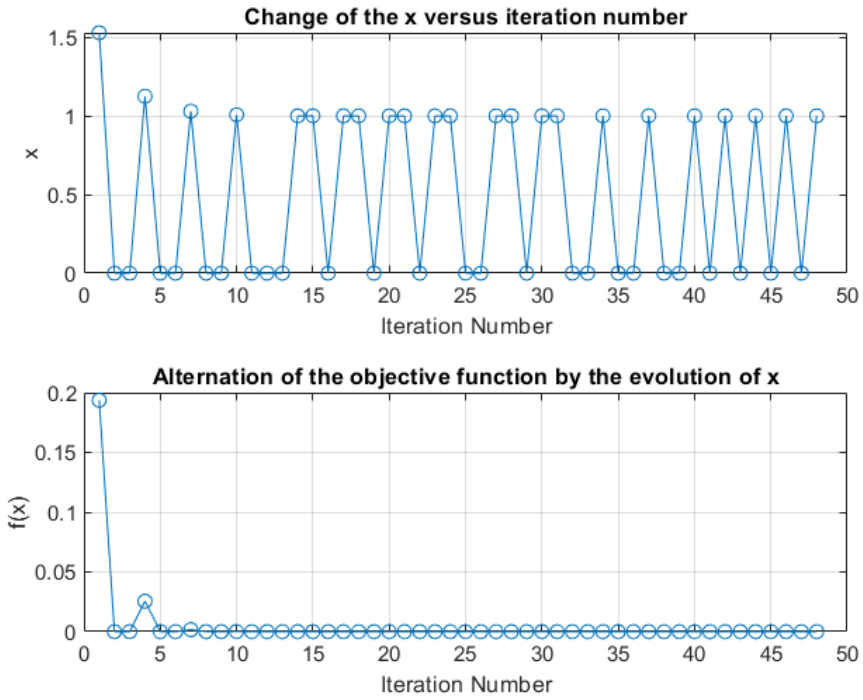

Figure 6: Results for the golden selection algorithm

2 Question 2

In the second question, eye lenses of rabbits were demonstrated as a function of age. To model the mentioned data, the following function were used.

$$y = 233.846(1 - \exp(-0.006042x)) + \epsilon$$

2.1 Comparison by using nftool

To improve the modelling and get a better MSE, we can use neural network structures. The data was taken and different types of neural networks created in the MATLAB.

For a comparison, a neural network with 5 hidden layers, another one with 10 hidden layers and the last one with 20 hidden layers were created. The MSE values of them and the first equation can be seen as follows.

First eqn	NN w 5HL	NN w 10HL	NN w 20HL
76.11	60.6	45.14	170.7

It is also important to note that since the biases and weights of the neural network converges to a different point in every run, the RMS values vary in every run either. In the figure 7 it can be observed that, the neural

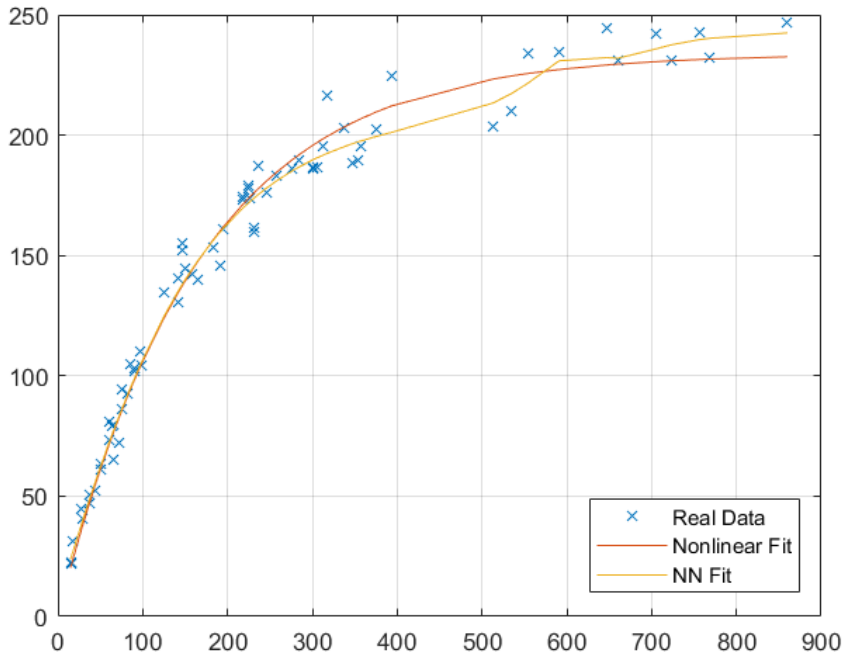

Figure 7: Comparison of single valued function and NN with 5 hidden layer

network structure represents the data better than the first function. However, since the number of neurons are not high, the representation is close. In the figure 8, the data was modelled with a NN model which includes 10 hidden layers. The data was modelled the best with this structure. In the last comparison, the model starts to overfit the data. Even if the model and the graphs changes in every run, it can be said that the more the model gets complicated, the more it is likely to overfit and the less the model is complicated, the more it is likely to underfit.

As a result, it is possible to obtain a better model with neural networks than the single valued function.

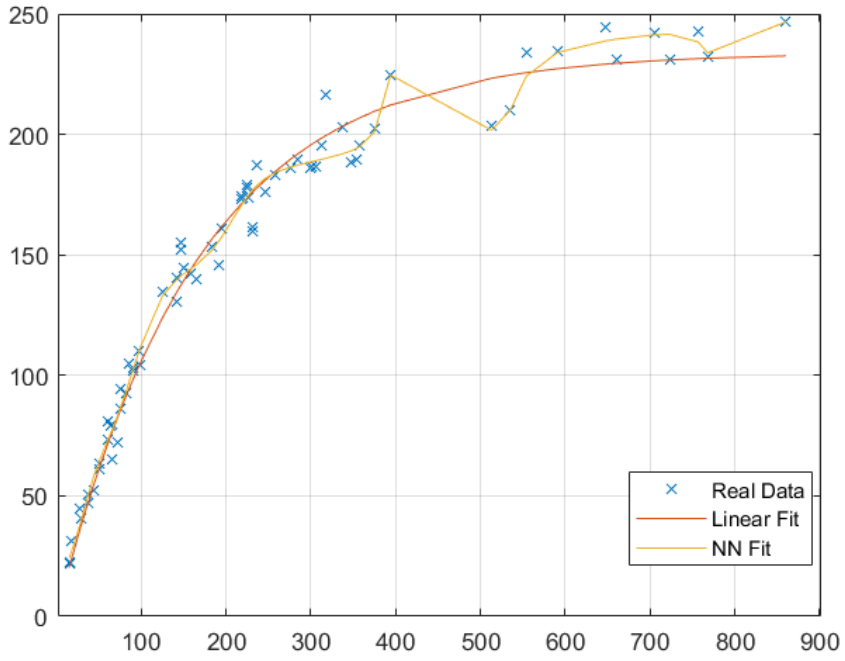

Figure 8: Comparison of single valued function and NN with 10 hidden layer

2.2 Comparison by implementing algorithm from scratch

For another comparison, a neural network structure was implemented in MATLAB. It actually consist of six different steps.

- Get Data
- Preprocess Data
- Initialize weights and biases
- Feedforward Phase
- Calculate Error
- Backpropagation

In the code, the data was taken and preprocessed. In the preprocessing part, firstly i shuffled data to make the train and test data contain samples from every part. Secondly, i normalized data to help the algorithm to reach global minimum faster. Finally, i divided the data into two parts as seventy percent training and thirty percent testing data.

Then it is crucial to initialize the weights properly. I basically tuned the weights by try and error methodology. After that, the feedforward stage was performed by matrix multiplications. Next, i calculated error between what i get and what it should be. Finally, i updated the weights with backpropagation.

For the network structure, i used 1 input layer, 5 hidden layers and 1 output layer. I used sigmoid activation function in the hidden layers and RELU activation function in the output layer. The results can be seen in the following table.

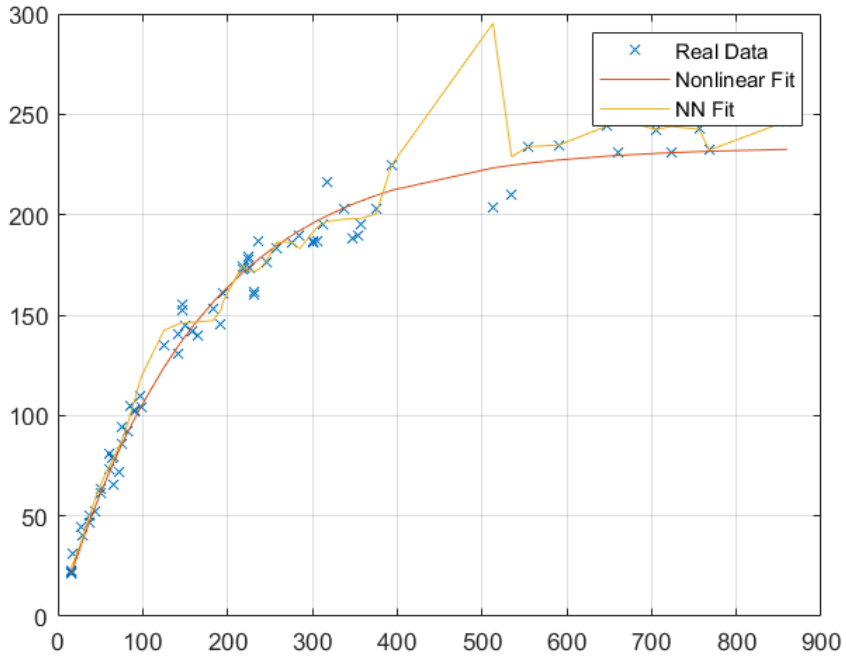

Figure 9: Comparison of single valued function and NN with 20 hidden layer

Train Error	Test Error
0.011	0.031

It is obvious that test error is greater than the train error. It might be about overfitting and can be investigated further. The RMS error is relatively smaller than the nftool due to the normalization of the data. The cost function of the training phase can be seen in the figure 10. The code can be found in the attachment.

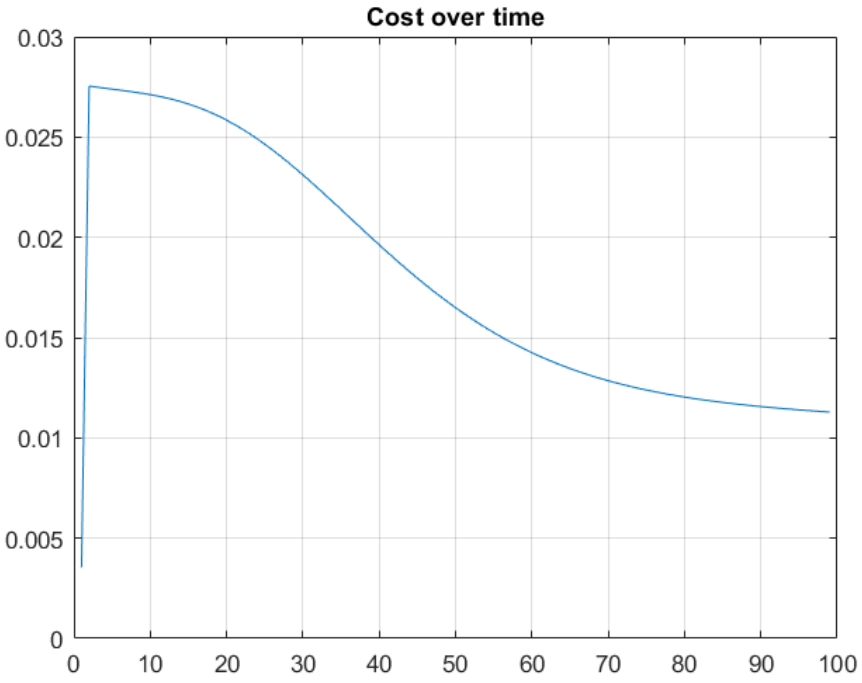

Figure 10: Cost versus iteration in the self implemented training phase

Appendix A MATLAB Codes

```
clc;clear
syms x

func = (x-1)^2*(x-2)*(x-3); %Cost function
fplot(func,[0,4]);

g = diff(func); %First derivative of the cost function
H = diff(func,2); %Second derivative of the cost function (Hessian)

%First step: initialization
x_init = 2.5;
epsilon = 10*10^-10;
deltaX = 0;

for k=1:100
    x_values(k) = x_init;
    cost_func_values(k) = vpa(subs(func,x,x_init));
    derivative_values(k) = vpa(subs(g,x,x_init)); %Calculating the value of function derivative at x_in
    second_derivative = vpa(subs(H,x,x_init)); %Calculating the value of functions second derivative at
    fprintf('Iteration %d: x=%.20f, err=%.20f\n', k, x_values(k), derivative_values(k));
    %Second step: calculating the delta
    deltaX = -derivative_values(k)/second_derivative;
    %Third step: calculating the next x
    x_next = x_init+deltaX;
    %Fourth step: check if the derivative<epsilon
    if(abs(vpa(subs(g,x,x_next)))<epsilon)
       break
    end
    x_init = x_next;
end
```

Listing 1: Code for the Newton-Raphson Method.

```
clc;clear
syms x

func = (x-1)^2*(x-2)*(x-3); %Cost function

g = diff(func); %First derivative of the cost function
H = diff(func,2);
%First step: initialization
epsilon = 10^-4;
alpha_a = 1.8;
alpha_b = 3;

for k=1:100
    alpha_k = alpha_a + (alpha_b-alpha_a)/2;

    alpha_values(k) = alpha_k;
    alpha_derivative_values(k) = vpa(subs(g,x,alpha_k));
    alpha_gradient_values(k) = vpa(subs(H,x,alpha_k));

    if(vpa(subs(g,x,alpha_k)) == 0)
        break
    elseif(alpha_b-alpha_a < epsilon)
        break
    elseif(vpa(subs(g,x,alpha_k))*vpa(subs(g,x,alpha_a))>0)
        alpha_a = alpha_k;
    else
        alpha_b = alpha_k;
    end
end
```

Listing 2: Code for the Bisection Method.

```
clc;clear
syms x

func = (x-1)^2*(x-2)*(x-3); %Cost function

%Step 1: initialize

x_low = 0;
x_up = 2;
deltaX_final = 10^-10;
tau =  0.38197;
epsilon = deltaX_final/(x_up-x_low);
N = -2.078*log(epsilon);
k = 1;

%Step 2

x1 = (1-tau)*x_low+ tau*x_up;
f1 = vpa(subs(func,x,x1));
x2 = (1-tau)*x_up+ tau*x_low;
f2 = vpa(subs(func,x,x2));

%Step 3
for i=1:100
    if(k<N)
       if(f1>f2)
          x_low = x1;
          x1 = x2;
          f1 = f2;
          x2 = tau*x_low + (1-tau)*x_up;
          f2 = vpa(subs(func,x,x2));
          x2_values(k) = x2;
          f2_values(k) = f2;
          k = k + 1;

       elseif(f1<f2)
          x_up = x2;
          x2 = x1;
          f2 = f1;
          x1 = tau*x_up + (1-tau)*x_low;
          f1 = vpa(subs(func,x,x1));
          k = k + 1;
          x1_values(k) = x1;
          f1_values(k) = f1;
       end
    end
i = i+1;
end
```

Listing 3: Code for the Golden Selection Method.

```
clc;clear;
data=readtable('Kitap1.csv');
ages=data{:,1};
weights=data{:,2};
save('mymat.mat','ages','weights')

dim = size(ages);
for i=1:dim(1)
    guess(i) = 233.846*(1-exp(-0.006042*ages(i)));
    MSE_err(i) = weights(i) - guess(i);
end
MSE = mean(MSE_err.^2);
%% Neural Network Part
x = ages';
t = weights';

% Choose a Training Function
trainFcn = 'trainlm';  % Levenberg-Marquardt backpropagation.

% Create a Fitting Network
hiddenLayerSize = 10;
net = fitnet(hiddenLayerSize,trainFcn);

% Setup Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

% Train the Network
[net,tr] = train(net,x,t);

% Test the Network
y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y)
```

Listing 4: Code for the second question.