Chua HongWei (A0074741E)
Homework 2

Q1a)

Rosenbrock's Valley Function

$$f(x,y) = (1-x)^2 + 100(y-x^2)^2$$

At global minimum,

$$\nabla f(x,y) = 0$$

Differentiate f w.r.t x

$$\frac{\partial f}{\partial x} = 0$$

$$\frac{\partial f}{\partial x} = 2(1-x)(-1) + 2(100)(y-x^2)(-2x) = 0$$

$$2x - 2 - (400x)(y-x^2) = 0 \text{ --- (1)}$$

Differentiate f w.r.t y

$$\frac{\partial f}{\partial y} = 0$$

$$\frac{\partial f}{\partial y} = 2(100)(y-x^2) = 0$$

$$y = x^2 \text{ --- (2)}$$

Subs (2) into (1)

$$2x - 2 = 0$$
$$x = 1$$

Thus,

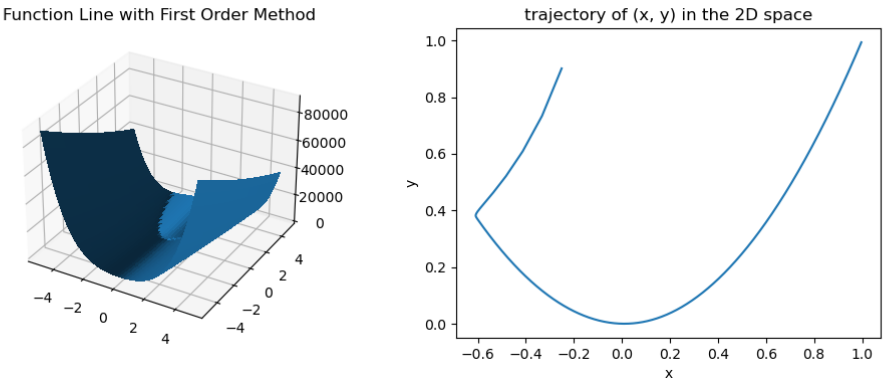
$$y = x^2 = 1$$

This shows that above Rosenbrock's Valley Function has a global minimum at (x,y) = (1,1).

Q1b)

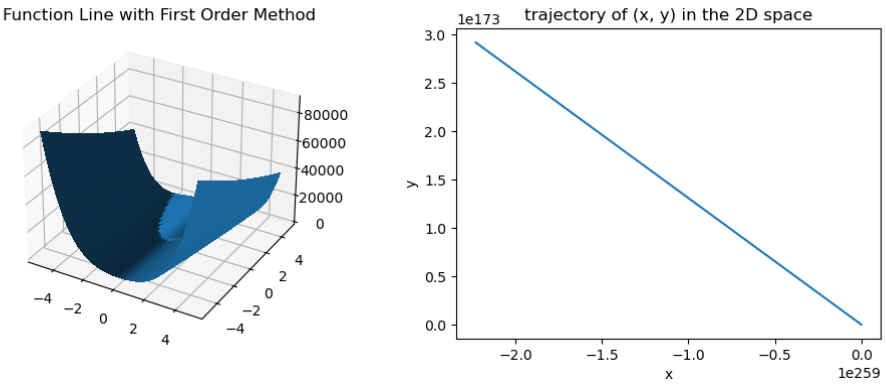
For learning rate, η = 0.001, the following result is obtained with steepest gradient descent method. f(x,y) converge after 11777 epochs, with the x,y value shown below.

```
The function converge after 11777 epochs
The x value after convergence is 0.996840692417849
The y value after convergence is 0.9936786997335835
```



The plot of the function and x,y trajectory can be seen from the figure above.

For learning rate, η = 1.0, the following result is obtained with steepest gradient descent method. Steepest gradient method fails to converge, and overflow error is encountered. The graph can be seen in the figure below.

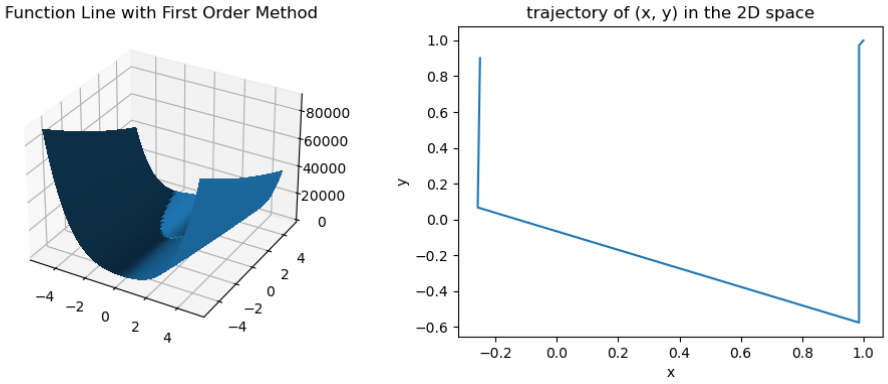


Q1c)

The following graph is obtained when the function is run with second order method. The function converges within 4 epochs which is much faster when compared to gradient descent method.

```
The function converge after 4 epochs
The x value after convergence is 0.9999999943145675
The y value after convergence is 0.9998048119844937
```

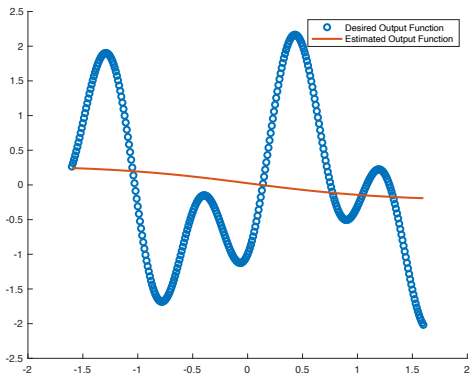
The plot of the function and x,y trajectory can be seen from the figure below.



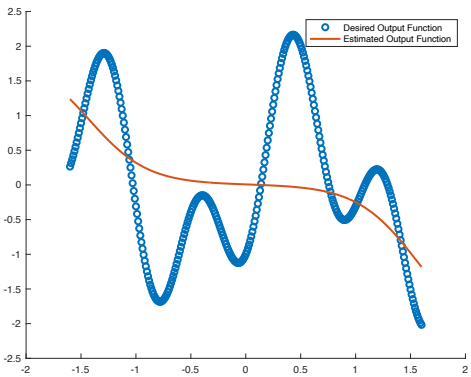
Q2a)

In this solution, the neural network is built with the hidden layer value n = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 50 & 100. The learning rate chosen is 0.1 with the epoch of 1,000 for this sequential learning problem. The optimizer used for this problem is Levenberg-Marquardt algorithms. The plots of desired function vs predicted function are listed as follows. The plot in red is the predicted function and the plot in blue is the desired function.

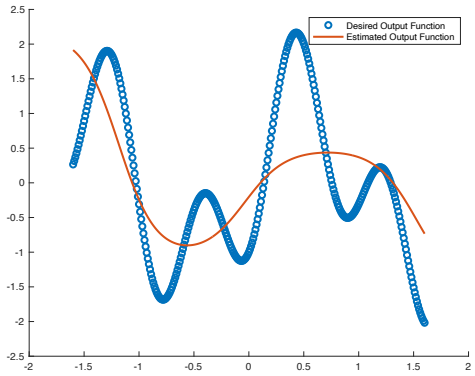
n = 1 (1 hidden layer)



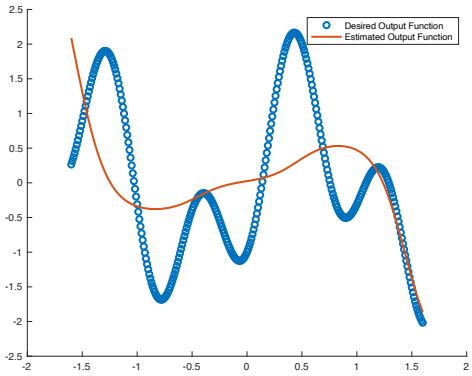
n = 2 (2 hidden layers)



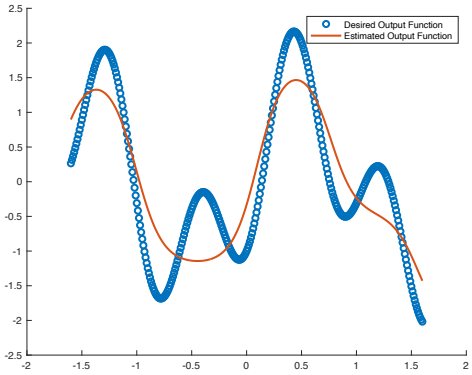
n = 3 (3 hidden layers)



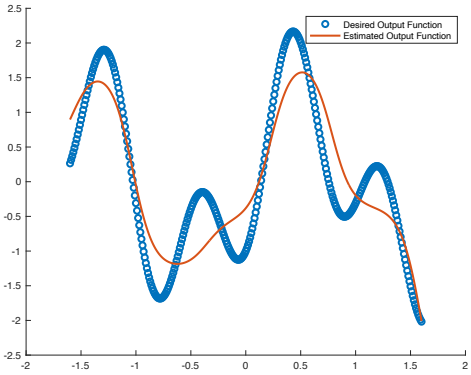
n = 4 (4 hidden layers)



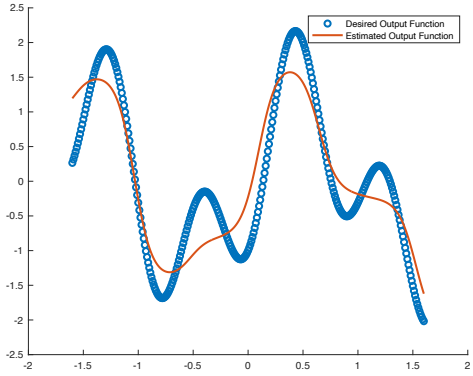
n = 5 (5 hidden layers)



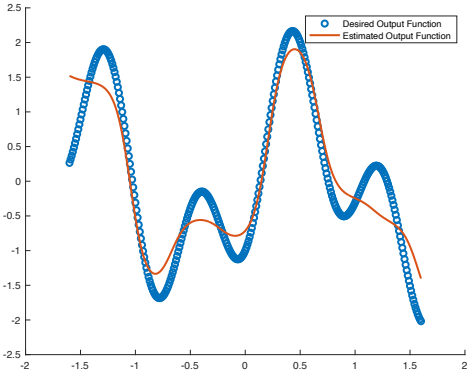
n = 6 (6 hidden layers)



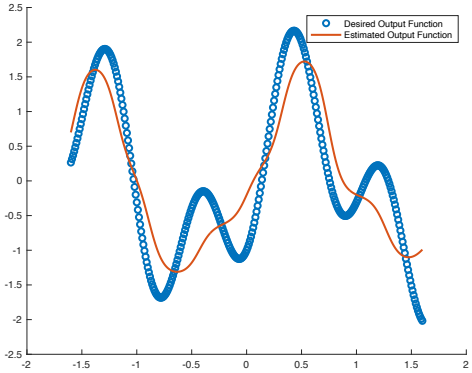
n = 7 (7 hidden layers)



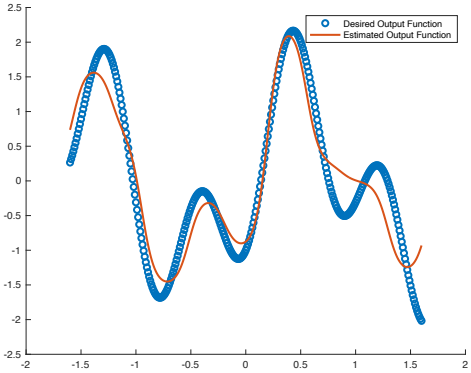
n = 8 (8 hidden layers)



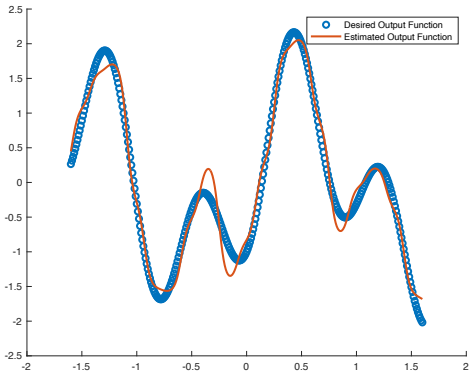
n = 9 (9 hidden layers)



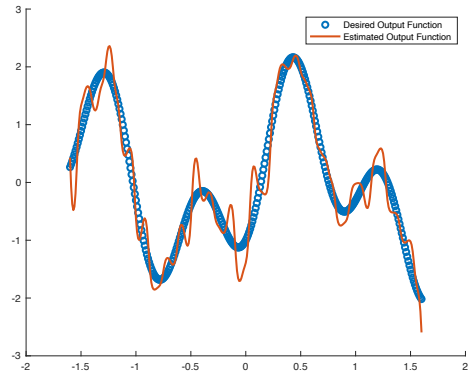
n = 10 (10 hidden layers)



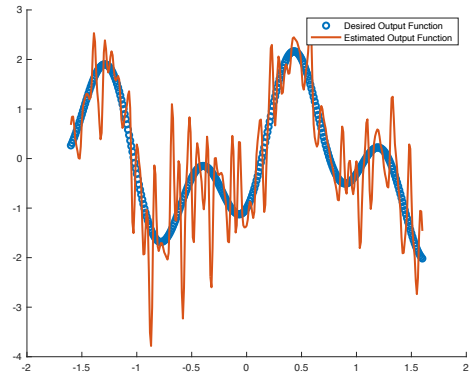
n = 20 (20 hidden layers)



n = 50 (50 hidden layers)



n = 100 (100 hidden layers)



From the plot above, network with 10 hidden layers is a good fit to approximate the function. Any lesser hidden layer will produce a underfitting curve. When the number of hidden layers is huge, in the case of 50 and 100, the function approximation produced a overfit curve, judging from the behavior of the curve trying to fit into all the data points.

For the range of x-data, which lies between -1.6 and 1.6, the number of line segment is 8, which is slightly less than the experiment result where 10 hidden neurons is required to obtain a proper fit for the function.

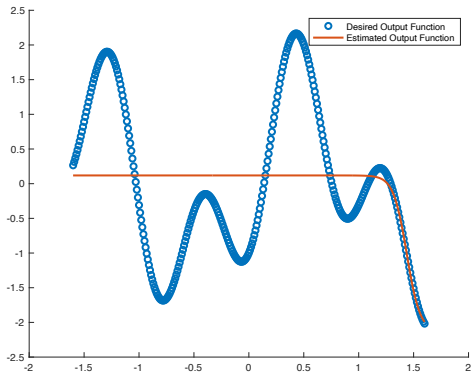
By using 10 hidden layers, the following output is obtained when input of x = 3 and x = -3 is used. The output value from function is different from the output value from the network. This suggest that the network is unable to make a proper prediction outside the domain of limited by the training set.

```
The output value from function for x = 3 : 0.8090
The output value from function for x = -3: 0.8090
The output value from approx function for x = 3  : -1.5865
The output value from approx function for x = -3 : 0.8751
```

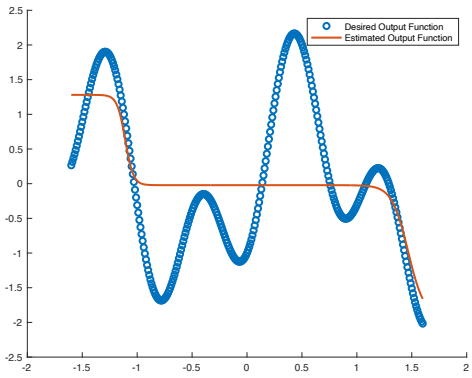
Q2b)

In this solution, the neural network is built with the hidden layer value n = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 50 & 100. The learning rate chosen is 0.1 with the epoch of 500 for this batch training problem. The optimizer used for this problem is Levenberg-Marquardt algorithms (trainlm). The plots of desired function vs predicted function are listed as follows. The plot in red is the predicted function and the plot in blue is the desired function.

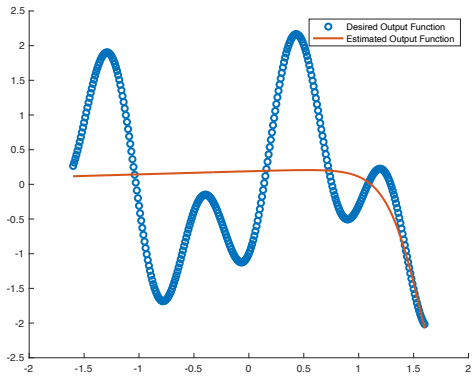
n = 1 (1 hidden layer)



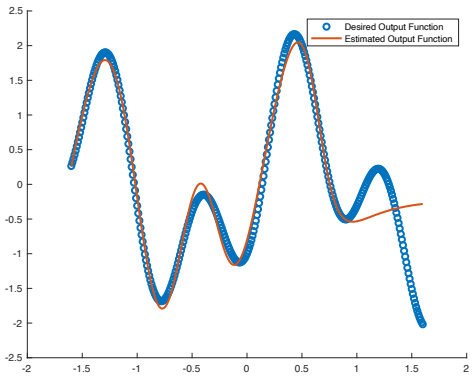
n = 2 (2 hidden layers)



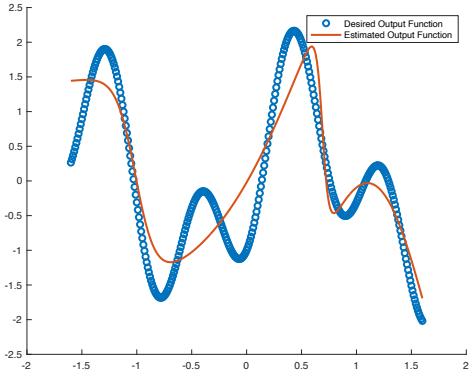
n = 3 (3 hidden layers)



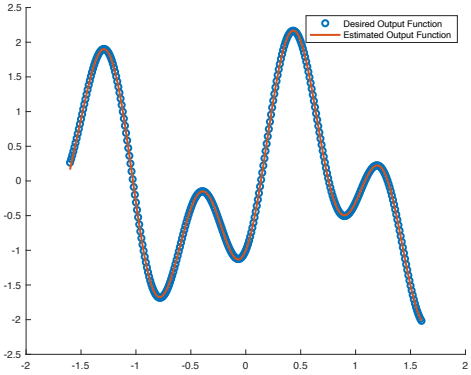
n = 4 (4 hidden layers)



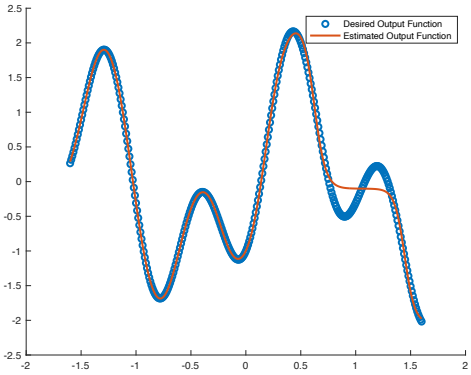
n = 5 (5 hidden layers)



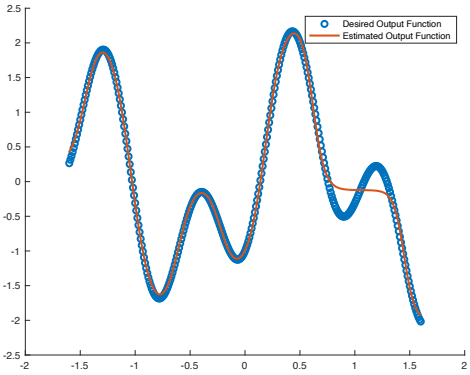
n = 6 (6 hidden layers)



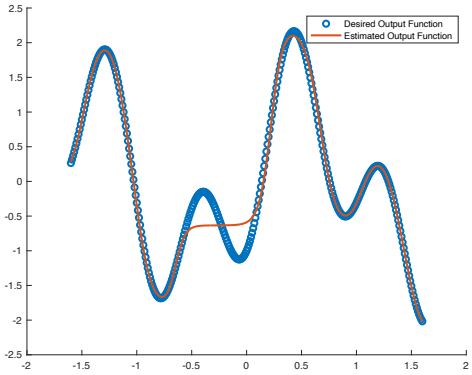
n = 7 (7 hidden layers)



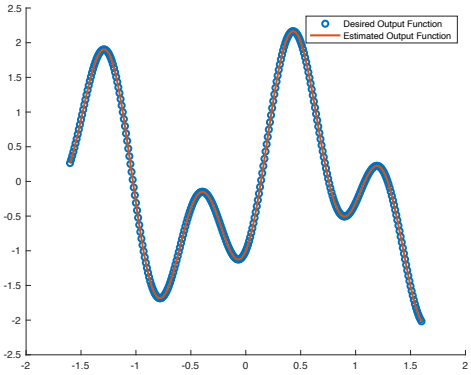
n = 8 (8 hidden layers)



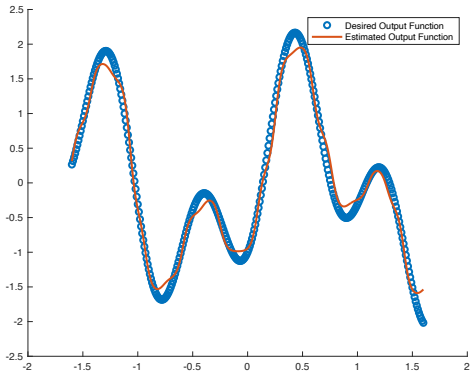
n = 9 (9 hidden layers)



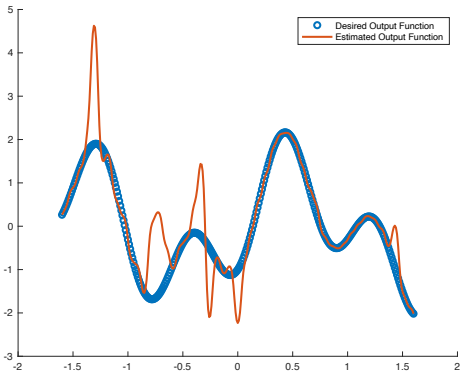
n = 10 (10 hidden layers)



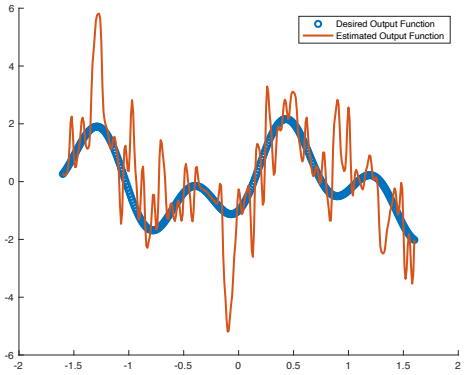
n = 20 (20 hidden layers)



n = 50 (50 hidden layers)



n = 100 (100 hidden layers)



For batch training, judging from the plot above, network with 6 hidden layers is a good fit to approximate the function. Any lesser hidden layer will produce a underfitting curve. The network produces proper fit even with 20 hidden layers. When the number of hidden layers is huge, in the case of 50 and 100, the function approximation produced an overfit curve, judging from the behavior of the curve trying to fit into all the data points.

For batch training with trainlm, the number of hidden layers required is less than the guideline consistent in the lecture notes where there are 8 line segments forming the desired output function. Another observation from this experiment is that for batch training, the algorithm takes fewer epoch before convergence occur.

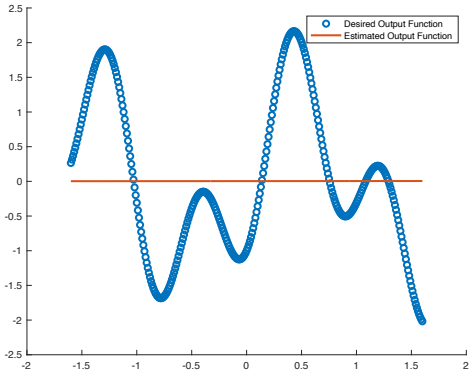
For this question, 6 hidden layers is used for the prediction of output value for x = -3 and 3. The predicted output value deviates from the actual output value which shows that the trained network is unable to make a proper prediction outside the domain of limited by the training set.

```
The output value from function for x = 3 : 0.8090
The output value from function for x = -3: 0.8090
The output value from approx function for x = 3  : -2.0485
The output value from approx function for x = -3 : 0.1620
```

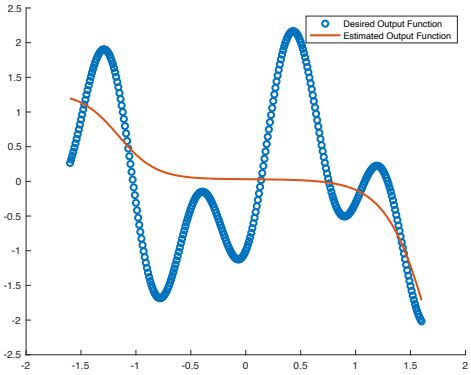
Q2c)

In this solution, the neural network is built with the hidden layer value n = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 50 & 100. The learning rate chosen is 0.1 with the epoch of 500 for this batch training problem. The optimizer used for this problem is Bayesian regularization backpropagation (trainbr). The plots of desired function vs predicted function are listed as follows. The plot in red is the predicted function and the plot in blue is the desired function.

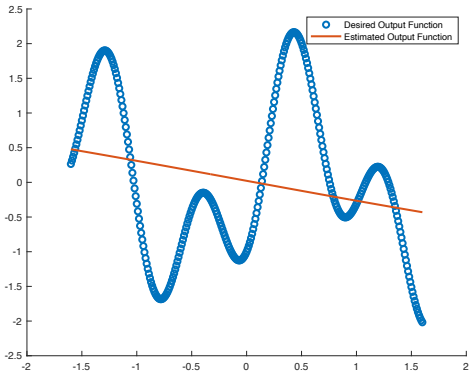
n = 1 (1 hidden layer)



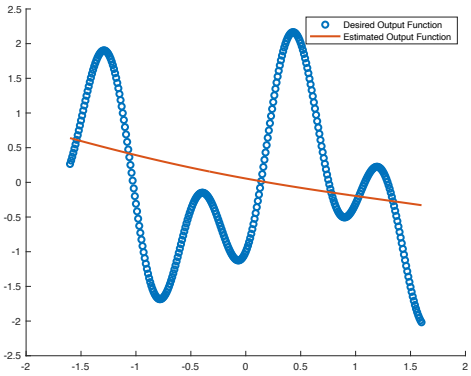
n = 2 (2 hidden layers)



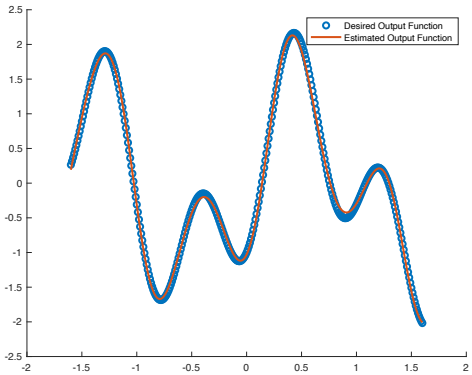
n = 3 (3 hidden layers)



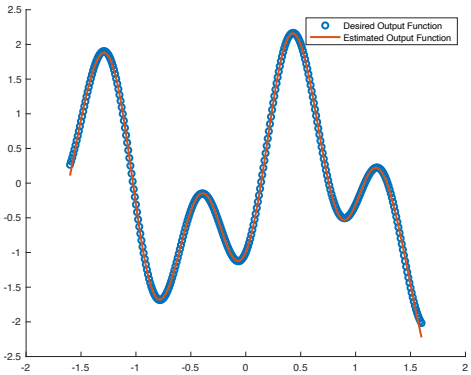
n = 4 (4 hidden layers)



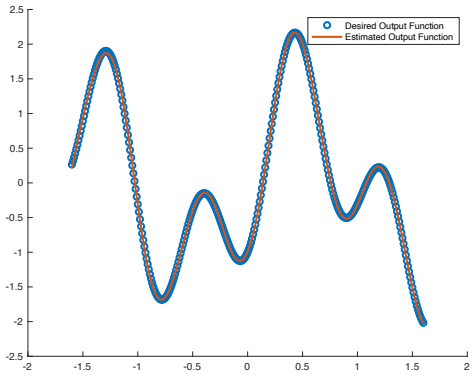
n = 5 (5 hidden layers)



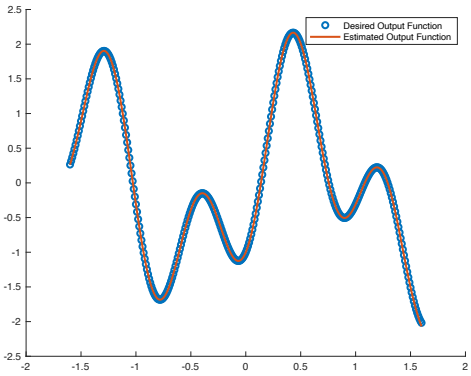
n = 6 (6 hidden layers)



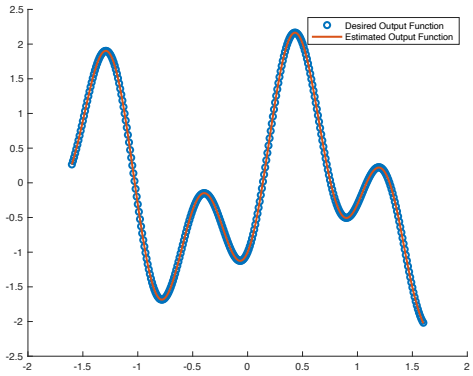
n = 7 (7 hidden layers)



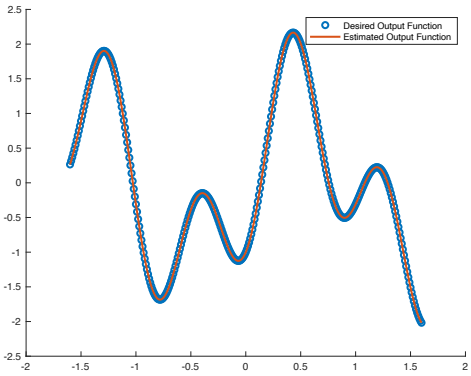
n = 8 (8 hidden layers)



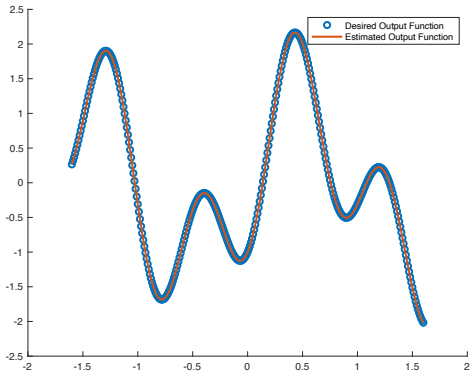
n = 9 (9 hidden layers)



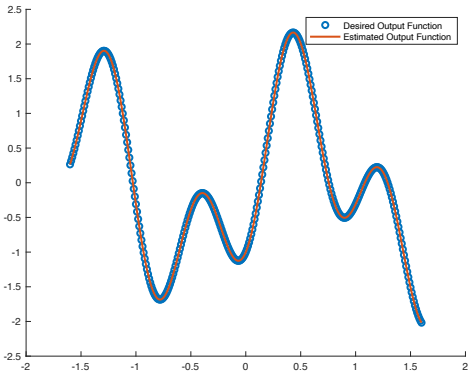
n = 10 (10 hidden layers)



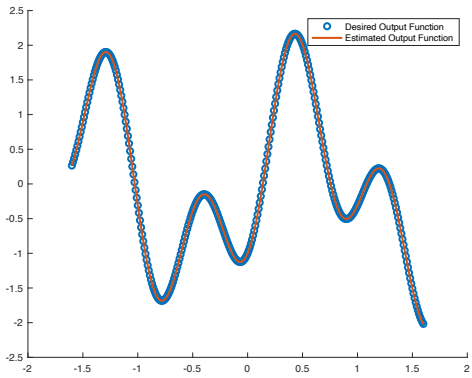
n = 20 (20 hidden layers)



n = 50 (50 hidden layers)



n = 100 (100 hidden layers)



For batch training with trainbr, judging from the plot above, network with 5 hidden layers is a good fit to approximate the function. Any lesser hidden layer will produce a underfitting curve. The network produces proper fit even with large neural network such as 20, 50 and 100 hidden layers.

For this question, 5 hidden layers is used for the prediction of output value for x = -3 and 3. The predicted output value deviates from the actual output value which shows that the trained network is unable to make a proper prediction outside the domain of limited by the training set.

```
The output value from function for x = 3 : 0.8090
The output value from function for x = -3: 0.8090
The output value from approx function for x = 3  : 4.9952
The output value from approx function for x = -3 : -5.0429
```

Q3

The dataset that is assigned to me for this question is Group 2, where the neural network will be trained to classify deer and ship. The MATLAB code for this question is in the file Q3.m.

Q3a)

In the Rosenblatt's perceptron develop for this problem, a learning rate of 0.1 was chosen. The perceptron converges after 38,202 epochs with the perceptron achieving the following classification accuracy.

Training set: 100%
Testing set: 69%

Rosenblatt's perceptron performed well for this problem with the trained perceptron able to make better prediction than wild guess which would yield an accuracy of 50%.

Q3b)

The global mean and standard deviation of the dataset is found to be 123.7647 and 56.3617 respectively. By applying normalization to the dataset to have zero mean and unit standard deviation improve the testing accuracy and improve the training performance of the perceptron. With normalization in place and learning rate set to be same as Q3a, the algorithm only took 600 epochs to converge. This is an improvement of 63 times compare to training with raw dataset. The classification accuracy has also improved as shown below.

Training set: 100%
Testing set: 73%

By shifting the average of inputs towards zero will reduce the bias of perceptron to update the weights towards data with larger magnitude. This improves the performance of perceptron in classification task. Also, by scaling the dataset to similar size allows faster convergence, which is proven above.

Q3c)

MLP with batch training is designed for this question with the following parameters.

1. Training algorithm: Gradient descent w/momentum & adaptive lr backpropagation (traingdx)
2. Epoch : 5,000
3. Gradient before stopping. : 1e-6
4. Number of hidden neurons: 100

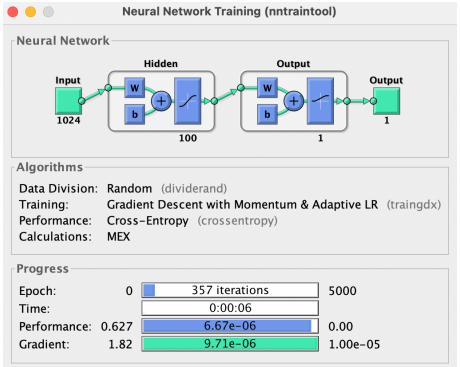
By using MLP, the accuracy for both training and testing sets are as follows.

Training set: 100%
Testing set: 83%

This is an improvement from previous two approaches, which shows MLP's ability to make prediction for a more complex dataset where more parameters involved.

Q3d)

The trained MLP in Q3c is overfitting. The MLP predicts the training set with 100% accuracy but fall short when it predicts the testing set. This shows that the MLP is not generalized. The epoch where the overfitting occurs is at epoch 357.



After applying regularization of 0.25, the MLP is still overfitting with not much improvement to the classification accuracy for testing set. The training epoch increase to 3,174 epochs before convergence.

Training set: 100%
Testing set: 82%

By increasing the regularization to 0.5 doesn't make too much different to the classification accuracy. The MLP is still overfitting.

Training set: 100%
Testing set: 81%

When the regularization is set to 0.75, it gives more weights to the mean square weights. However, it doesn't improve the overfitting issue with the following accuracy recorded. It also failed to converge after 5,000 epochs.

Training set: 100%
Testing set: 82%

Q3e)

For this sequential training problem, the epoch is set at 100. For fair comparison with Q3c, similar training algorithm traingdx and 100 hidden neurons is used. The classification accuracy is shown below.

Training set: 79.67%
Testing set: 82%

The training accuracy for sequential training is worse than the training accuracy for batch training. However, both MLP with sequential training and batch training perform similarly for the testing set.

Both methods produced similar testing accuracy and would work for this image dataset. If the dataset is huge, batch training could be considered as it is a much cheaper compute and would guarantee a convergence. Sequential training might cause weights fluctuations and stall the convergence.

Q3f)

The MLP performance could be improve by feeding image data that contains more pixels. As the images provided are only 32x32 pixels, the data that feeds into the MLP are limited. A larger image data would contain more information for the MLP to perform better classification.

Alternatively, by applying deeper neural network that involves more modern layers such as convolution neural network would improve the performance of the MLP also. Convolution neural network allows certain image features to be extracted and grouped before feeding into MLP for classification.