NEURAL NETWORKS PROGRAMMING TASKS REPORT

Prepared by:

Chamila Asanka Ariyaratne (Student number:- 801010-P171)

Subash Nemani (Student number:- 861104-9613)

Note:- MATLAB codes are included in the appendix at the end of the report

Programming task 1

Approximation and prediction

with feed forward neural networks

Task 1A

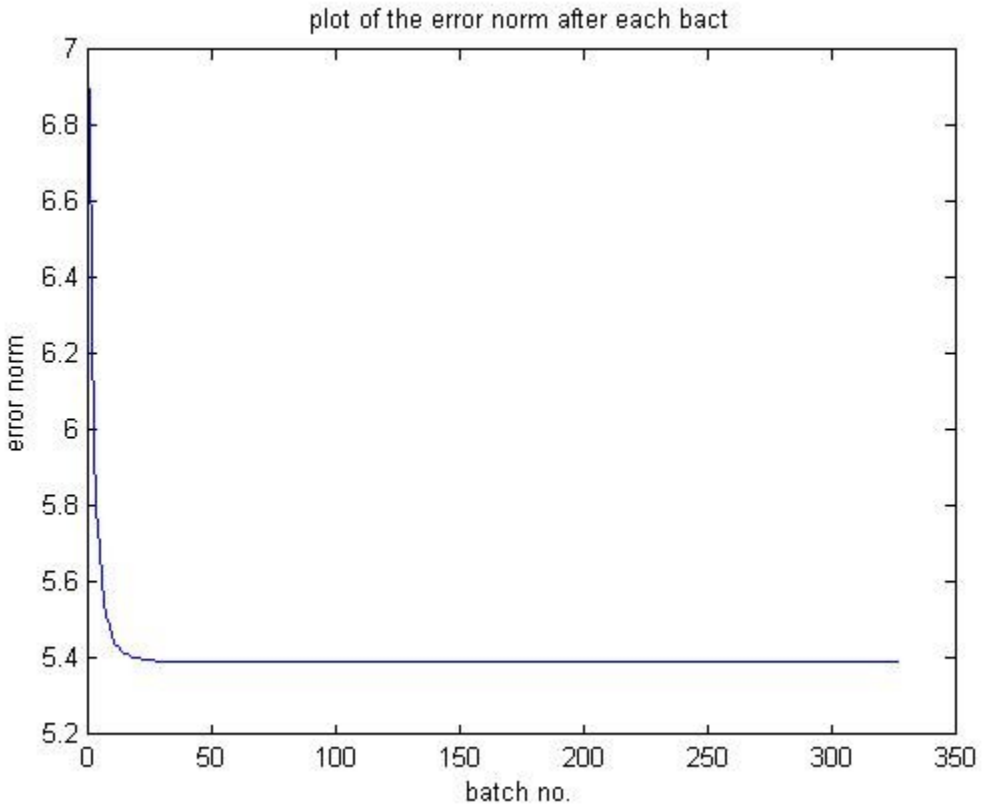
The aim of this task is to solve a set of linear equations using a single layer feed forward linear neural network. The equation set is given by **Ax=b**, where **A** is matrix of size m-by-k, **x** is a vector consisting of unknowns and of size k-by-1 and **b** is a vector of size m-by-1. The role of the neural network is to approximate the pseudo-inverse of **A**.

- m=30 , k=10
- Entries of **A** and **b** are randomly generated to be Gaussian distributed with zero mean and variance 1.
- Training sequence consists of rows of A (or columns of transpose of A) as the inputs and corresponding columns of a m-by-my (30-by-30) identity matrix as the targets. Thus, 30 neurons with each neuron has 10 weights, resulting in a weight matrix **W** of size 30-by-10.
- After convergence, transpose of the weight matrix **W** approximates the pseudo-inverse of **A**.
- The network was trained using MIMO-LMS with a leakage factor, since all the neurons are linear. But the leakage factor lambda was finally chosen as 0 after experimenting with different values since this achieved the lowest possible error after convergence.
- Step size alpha was initialized to 0.01, but reduced by 1.0 % after each batch of training. One batch includes running through all 30 input-target pairs in random order. Function “perm” was used to permutate the input-target pairs in random order. The code for this function is listed below.
- Stopping criteria:
    - After each batch, an error defined as **norm of Ax-b** where $\mathbf{x=W^Tb}$ is calculated. If the absolute value of the difference between above defined error after the current batch and the batch before remains less than 0.000001 for 100 consecutive batches, the training stops and the network is assumed to have converged. The transpose of the weight matrix **W** at this point is assumed to be the approximation for the pseudo-inverse of matrix **A**.

Task 1A (continued..)

Results:

- Plot of **norm of error Ax-b**,



- Error norm using Matlab pinv, (upto 4 decimal places as displayed on Matlab)

from_matlab_pinv =

5.3925

- Error norm using W^T, (upto 4 decimal places as displayed on Matlab)

from_neural_network =

5.3925

Task 1A (continued..)

- Difference between the above two errors,

diff_between_pinv_neural =

9.6338e-007

- Norm of the difference between the final weights W^T and Matlab pinv,

norm_diff_weights_pinv =

0.0012

Task 1B

In this task a 3 layer feed forward neural network is used to approximate the non-linear function,

$$f(x_1, x_2) = \sin\left(\frac{x_1^2}{4} + \frac{x_2^3}{2}\right), \; x_1 \in [-1,3], x_2 \in [-3,3].$$

Network configuration:

3 layer neural network.

2 inputs.

5 neurons in the first hidden layer.

4 neurons in the second hidden layer.

1 output neuron.

All neurons should have a bipolar sigmoid function.

All layers should have a bias term together with the inputs.

Networked to be trained using backpropagation algorithm.

Training sequence:

x_1 was sampled at 30 equally spaced points in the interval [-1 , 3] and x_2 was sampled at 30 equally spaced points in the interval [-3, 3] to give 900 combinations of input points. This was used on the given function to create 900 output points. The 900 combination of x_1 and x_2 together with a bias term of 1 formed the input of the training sequence and the 900 output points formed the targets of the training sequence.

Stopping criteria:

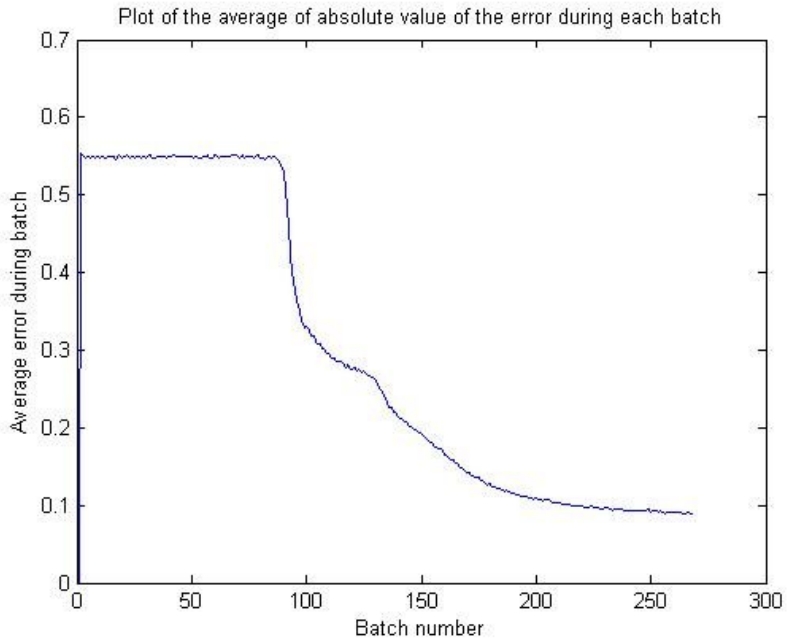
The training stops if the average error during each batch remains less than 0.1 for 50 consecutive batches. A batch consists of training the network using all 900 input-target pairs selected in randomly permutated order.

Comments on the step size:

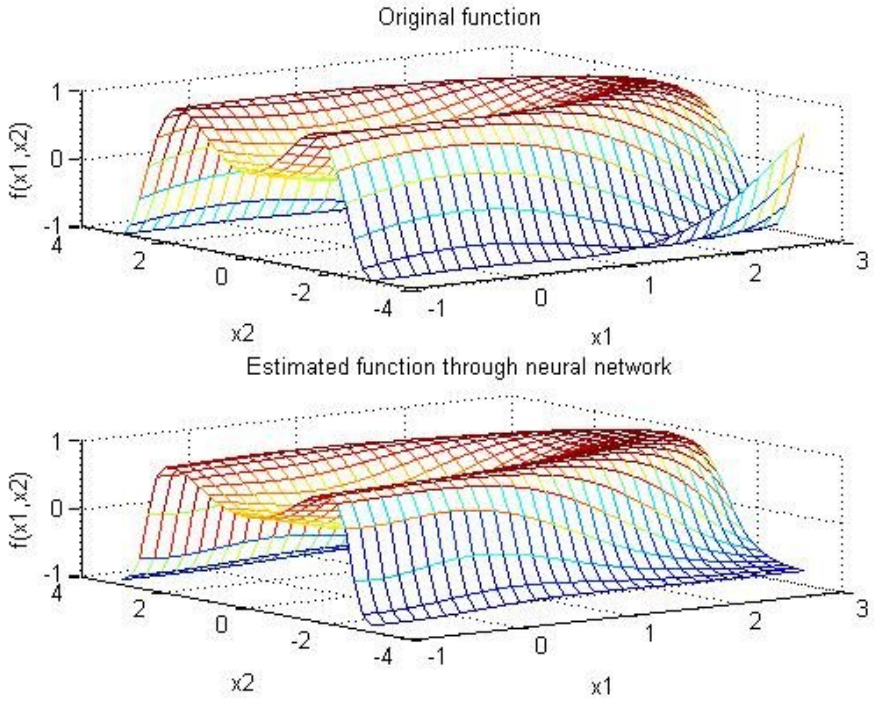
Step size was initiated to 0.01 and decreased by 0.1 % after each batch. When higher values of step sizes were used, the average error was dropping faster at the beginning but resulted in high error flucutuations which made it difficult to fulfill the chosen stopping criteria of error being less than 0.1 for 50 consecutive batches. Hence the training time was longer. When too low step sizes were chosen, it was taking far longer for the error to drop which also resulted in longer training. Therefore a step size initiated at 0.01 with a decrease of 0.1 % after each batch seemed to be a reasonable choice.

Results (without momentum term):

Plot of error without using the momentum term:



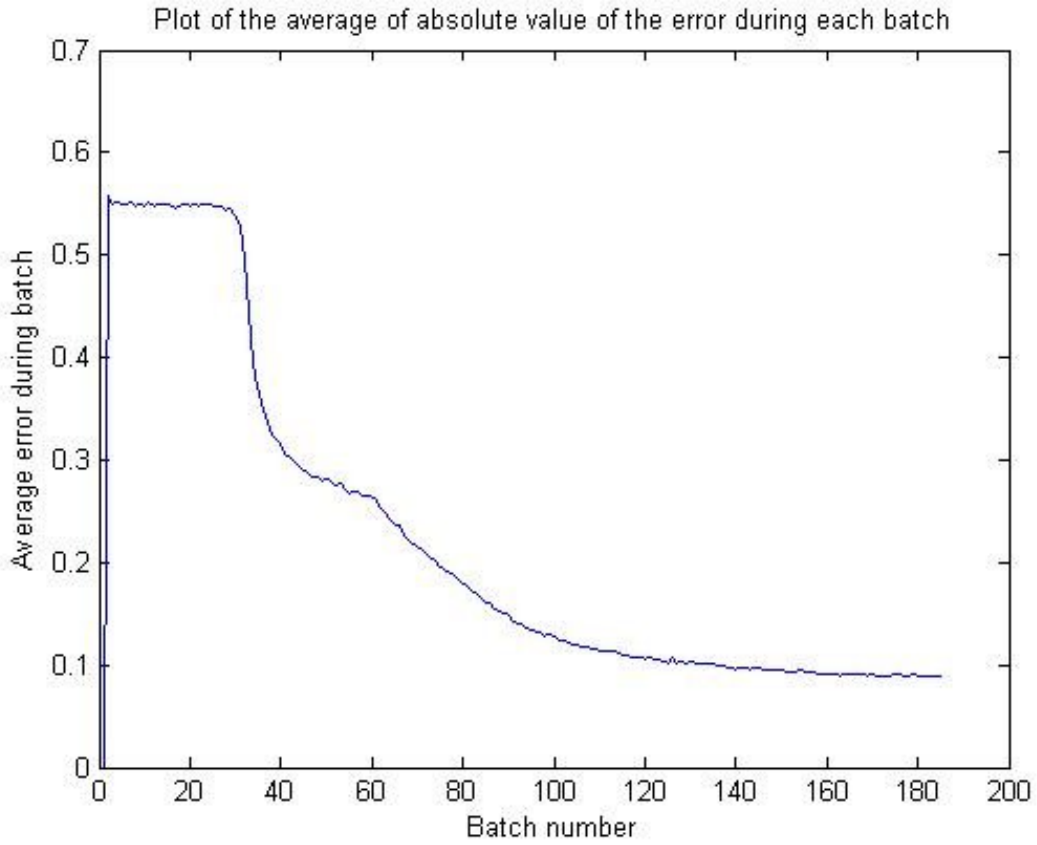
Plot of the approximated function output without the momentum term:



Results (with momentum term):

Momentum term beta was initiated to 0.3 with a decrease of 1% after each batch. As evident from the error plot below the training time was reduced with the momentum term. Without the momentum term, the training required approximately 270 batches as seen in the error plot in the previous page. With the momentum term the training required approximately 190 batches as shown in the error plot below.

Plot of error with the momentum term:



Task 1C

This tasks concerns predicting the non-linear system given by the below equation.

$$y(n) = \sqrt{0.1x(n) + 0.5x(n-1)} + 0.3x(n-2)^2 + \sin\big(x(n-3)\big)$$

Prediction of next output, *y(n+1)* is to be done using current and 3 past inputs *x(n), x(n-1), x(n-2), x(n-2)* and current and 3 past outputs *y(n), y(n-1), y(n-2), y(n-2).*

Network configuration:

3 layer neural network.

8 inputs.

10 neurons with bipolar sigmoid units in the first hidden layer.

5 neurons with bipolar sigmoid units in the second hidden layer.

1 linear neuron in the output layer.

All layers should have a bias terms.

Network to be trained using backpropagation.

Step size initialized to 0.01 and momentum factor beta initialized to 0.3.

Step size was decreased by 0.01% and beta was decreased by 0.1% after each batch.

Training sequence

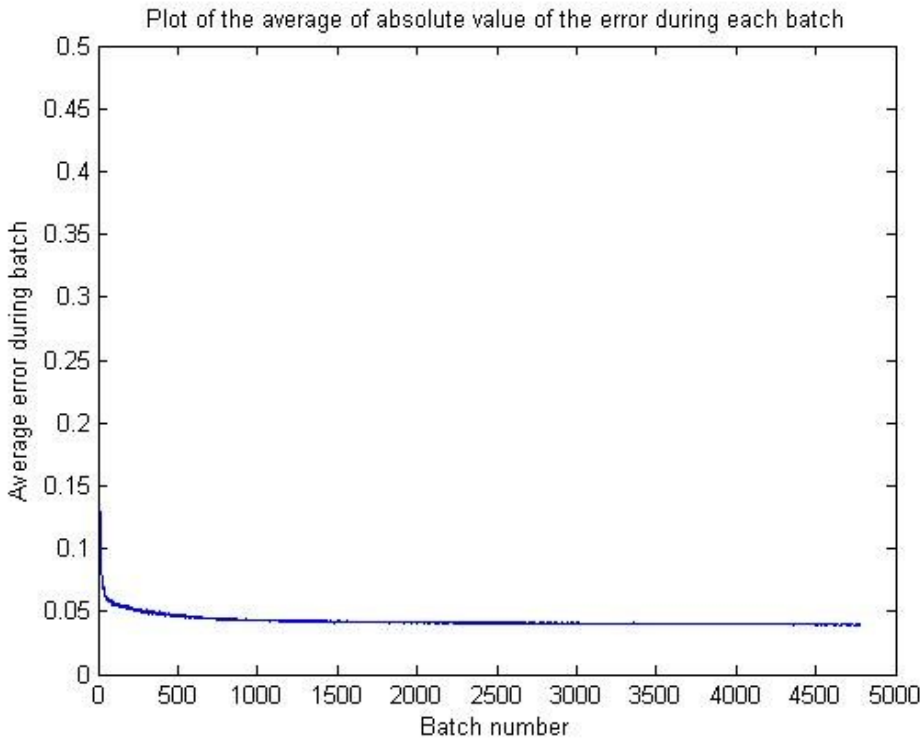
X(n) was generated as a vector of length 1001 of Gaussian distributed random numbers of mean zero and variance 1. X(n-1), X(n-2) and X(n-3) was created by delaying this series using delay filters [0 1], [0 0 1] and [0 0 0 1]. Y(n) was generated using the above generated vectors in the given formula. Y(n) was also filtered as above to get Y(n-1), Y(n-2) and Y(n-3). The input sequence during training was the first 1000 samples of [X(n) X(n-1) X(n-2) X(n-3) Y(n) Y(n-1) Y(n-2) Y(n-3)] together with the bias term of 1. The target sequence was the last 1000 samples of Y(n) which is for $2 \leq n \leq 1001$.

Stopping criteria:

The training stops if the average batch error remains less than 0.04 for 50 consecutive batches.

Results:

Plot of the average error during training:



The weights after convergence (Each row represents weights for each neuron):

Layer 1

-0,091442	1,045	-0,53427	0,033467	0,13824	-0,15187	0,016231	-0,028143	-2,2469
0,54391	-0,14525	0,77146	0,044364	-0,019506	0,31985	0,15901	-0,18622	-0,51821
-0,13859	1,2143	-0,34898	-0,042766	0,097082	-0,075933	0,06168	-0,0524	-1,0932
-0,32225	0,0025885	-1,2681	-0,11008	0,16796	-0,046517	-0,027688	0,09066	1,2535
0,8436	-0,21716	-0,7353	-0,35416	0,052739	0,15503	-0,074639	0,0063572	1,1054
-0,14	-0,19303	-1,3114	0,028692	0,010911	0,036237	0,039431	-0,16548	0,59176
0,53189	0,095353	-1,0447	-0,016905	0,1805	-0,011213	0,047347	0,091207	0,9356
0,16162	0,1537	-0,087607	-0,18018	-0,15619	0,72118	0,21578	-0,040408	-0,33963
0,50581	0,33815	-0,042839	-0,055452	-0,26981	0,078373	-0,13962	0,14935	-0,58663
-0,28579	0,23792	-0,44627	0,099763	-0,18498	0,22393	0,013855	-0,089982	1,464

Layer 2

0,19056	0,52295	0,21177	-0,5781	0,68362	-0,5314	0,22597	-0,24325	0,10648	0,33556	0,263
-0,70559	0,12239	-0,381	0,49067	0,18498	0,39713	-0,16993	-0,27204	-0,24837	0,11044	0,98591
-0,046538	-0,070418	-0,023718	0,10124	0,0090106	0,11766	-0,028377	-0,029856	-0,074608	0,052302	-0,010705
0,043883	0,63273	0,063341	-0,30278	0,67889	-0,30092	0,34738	-0,62976	-0,067492	1,004	0,26977
0,55345	0,22873	0,47309	-0,48038	-0,001959	-0,53515	0,5152	-0,23883	0,087601	0,10488	-0,12761

Layer3

0,60571	-1,8992	-0,10893	1,0359	0,58434	1,928

Programming Task 2

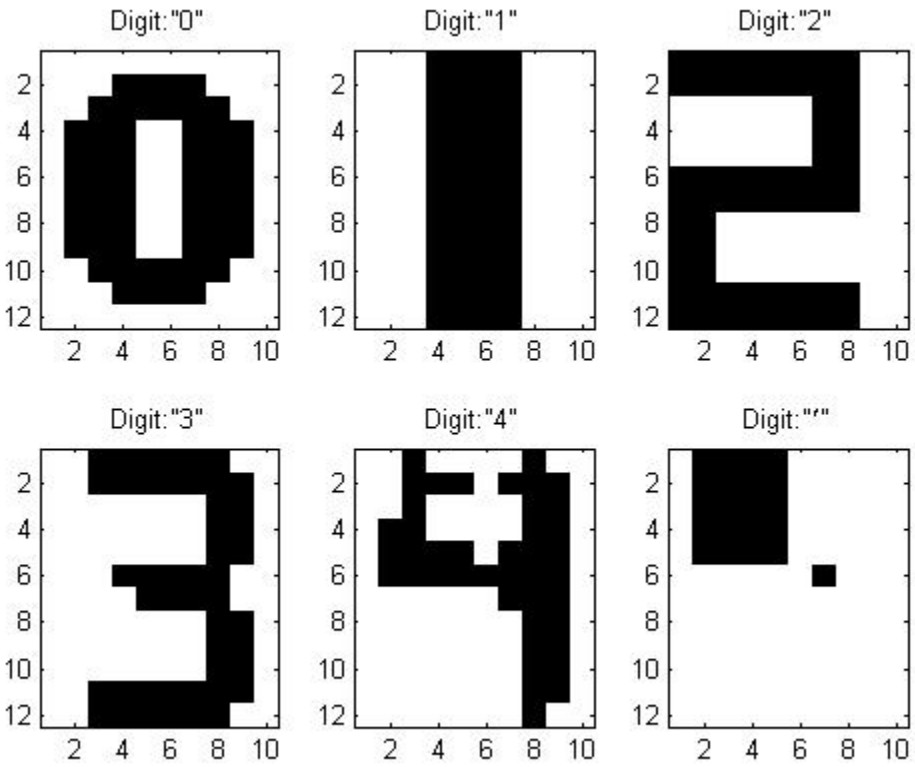
Character recognition with associative networks

Task 2A

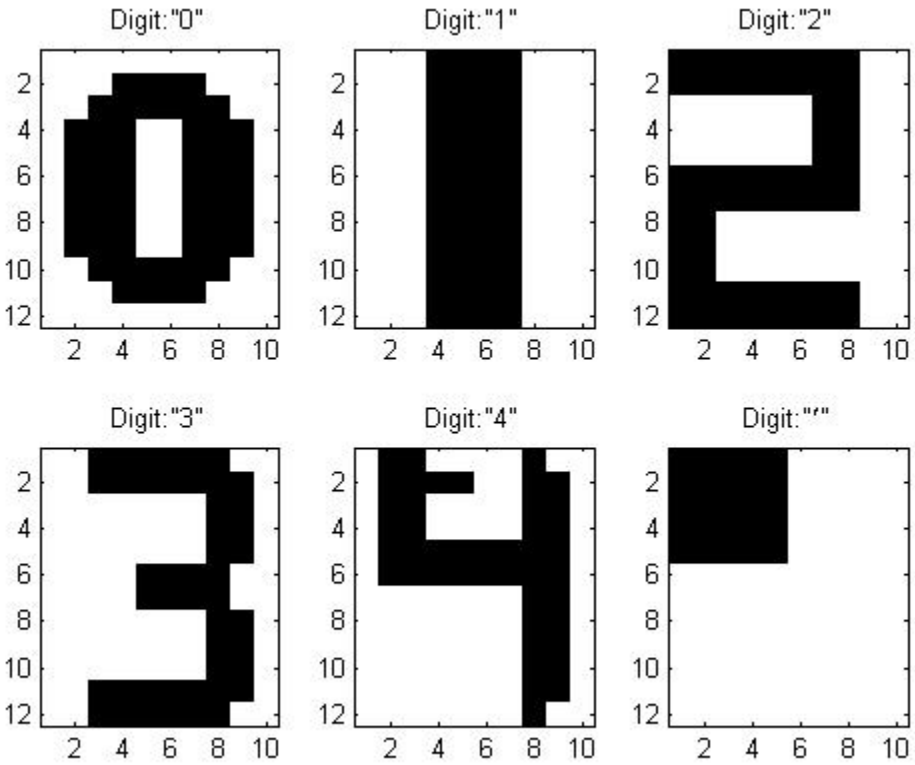
In this task a given set of distorted characters is recognized using an associative network using a single layer feedforward neural network of 12 neurons with each neuron having 12 weights. Auto associative memory is created first based on Hebbian paradigm and then using pseudo-inverse and the performance is compared.

Results (Feedforward network):

Characters recognized using Hebbian paradigm based auto-associative memory and presenting distorted characters.



Characters recognized using **pseudo-inverse** based auto-associative memory and presenting distorted characters.



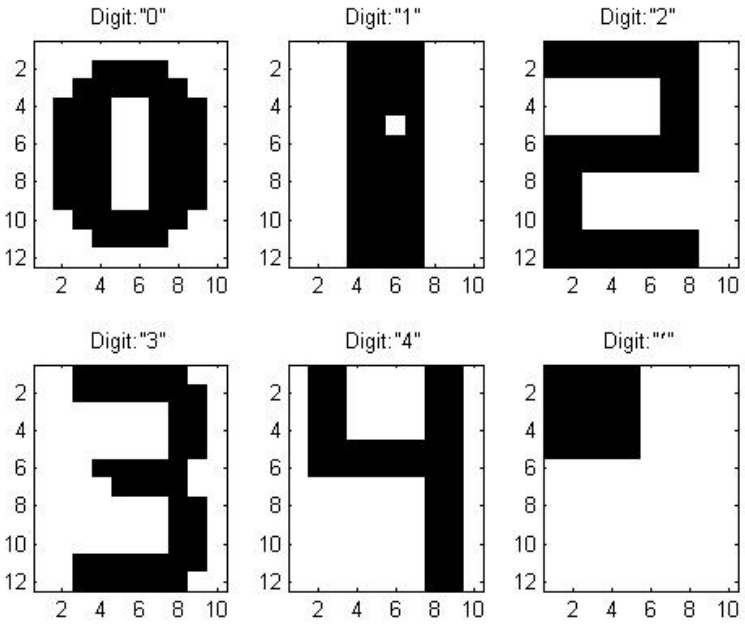
As evident from the above two figures, the pseudo-inverse based auto-associative memory has performed better in memory recovery. It is apparent from the recovery of the three last digits in the above two figures..

Task 2B

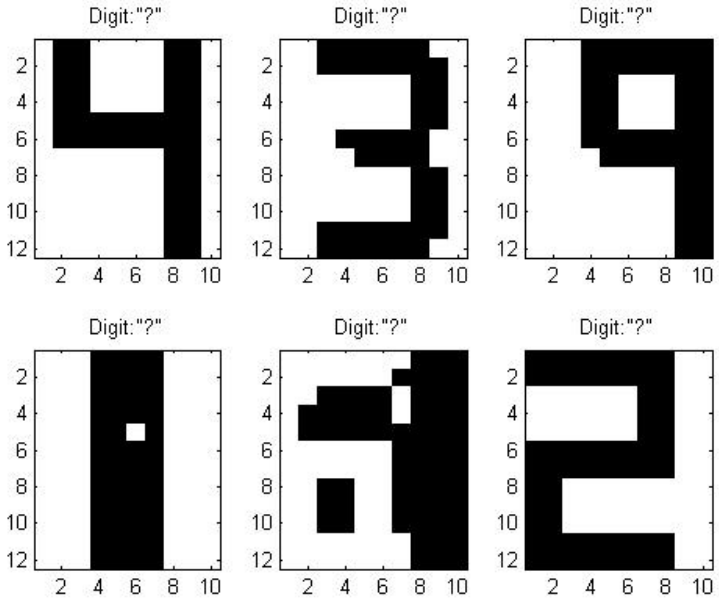
In this task a Hopfield network is used instead of a feedforward network to recognize the characters. The network is presented with a set of distorted digits as well as a set of heavily distorted digits.

Results (Hopfield network):

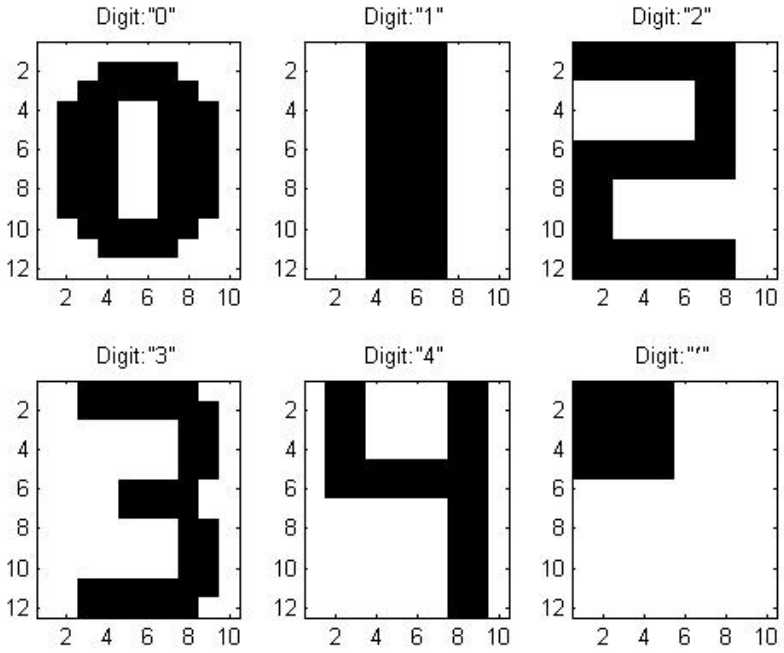
Characters recognized using Hebbian paradigm based auto-associative memory and presenting distorted characters.



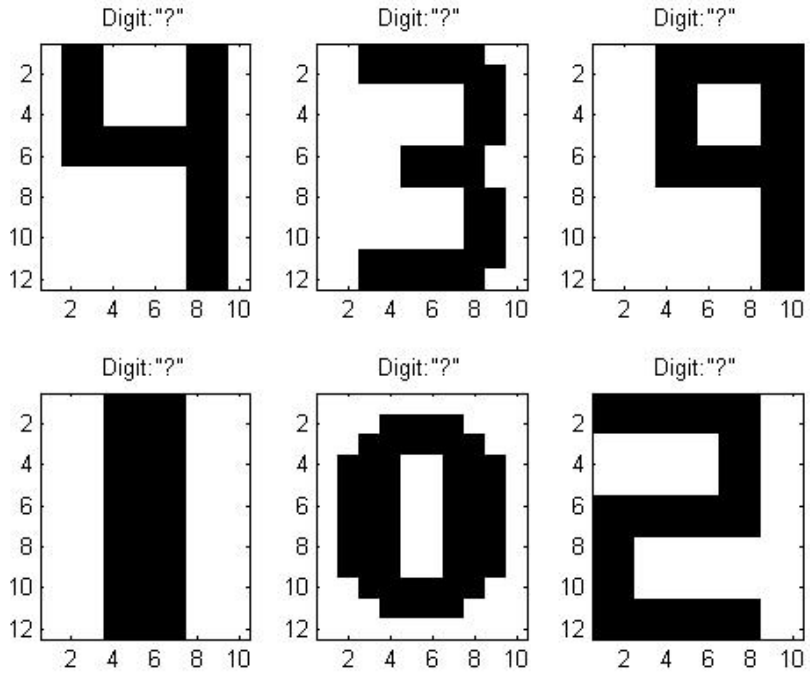
Characters recognized using Hebbian paradigm based auto-associative memory and presenting heavily distorted characters.



Characters recognized using pseudo-inverse based auto-associative memory and presenting distorted characters.



Characters recognized using pseudo-inverse based auto-associative memory and presenting heavily distorted characters.



Comment on the Hopfield network compared to the feedforward network:

By comparing the above images of digits reconstructed using feedforward network and the Hopfield network, it can be said that the Hopfield network is a better candidate as an associative network. Especially, when pseudo inverse memory is used together with the Hopfield network, it has managed to recover the distorted digits perfectly. We cannot comment on the accuracy of the recovery of heavily distorted digits, because we do not have the information as to what these digits actually are. But, it can be said that the Hopfield network together with pseudo-inverse memory has recovered the heavily distorted digits to perfectly match with one of the original digits.

Programming Task 3

Signal compression and Reconstruction with Principal Component Analysis (PCA) using the Generalized Hebbian Algorithm

Task 3A

In this task, an artificially created signal is used to train a network with Generalized Hebbian Algorithm (GHA) to find the principal components. Also the autocorrelation matrix of the signal is calculated mathematically followed by finding the eigenvectors of this matrix using MATLAB. The eigenvectors of the mathematically calculated autocorrelation matrix and the principal components found using the GHA trained network are compared.

The signals are defined as below:

$x_1 = \varepsilon_1$

$x_2 = -0.5x_1 + 0.5\,\varepsilon_2$

$x_3 = 0.4x_1 + 0.2x_2 + 0.1\,\varepsilon_3$

Where ε_1, ε_2 and ε_3 are **independent** Gaussian random variables with mean 0 and variance 1.

Let vector x be defined as,

$x = [x_1\ x_2\ x_3]^T$

Then the autocorrelation matrix of x , R_{xx} can be defined as,

$$R_{xx} = E\{\mathbf{x}\mathbf{x}^T\} = E\left\{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}\right\} = E\left\{\begin{bmatrix} x_1^2 & x_1x_2 & x_1x_3 \\ x_2x_1 & x_2^2 & x_2x_3 \\ x_3x_1 & x_3x_2 & x_3^2 \end{bmatrix}\right\}$$

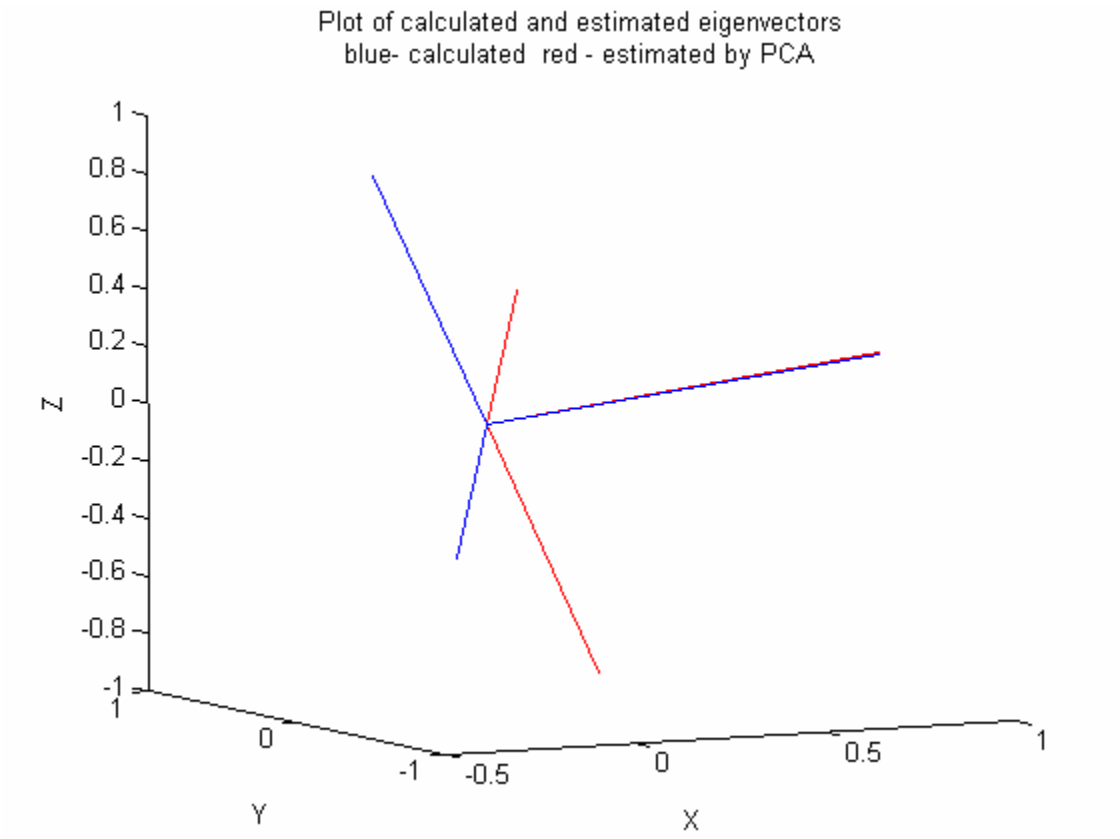
Solving the above expectations by using the statistical independence property of ε_1, ε_2 and ε_3 , R_{xx} was found to be,

$$R_{xx} = \begin{bmatrix} 1 & -0.5 & 0.3 \\ -0.5 & 0.5 & -0.1 \\ 0.3 & -0.1 & 0.11 \end{bmatrix}$$

MATLAB program was written to calculate the eigenvectors of the above autocorrelation matrix and also to train a neural network using GHA to find the principal components of the data. The first principal component should resemble the eigenvector of the autocorrelation matrix that relates to the largest eigenvalue, the second to the eigenvector with the second largest eigenvalue and the third to the eigenvector with the smallest eigenvalue.

Results:

Below plot shows the calculated and estimated eigenvectors. Lines in blue refer to the calculated eigenvectors through the autocorrelation matrix and lines in red refer to the estimated eigenvectors through PCA.



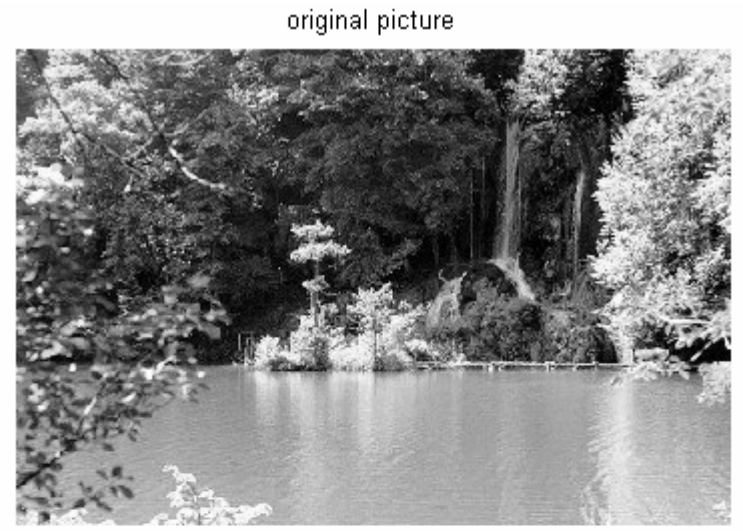
As shown from above plot the PCA has accurately estimated the eigenvectors although two vectors have reversed the sign of direction.

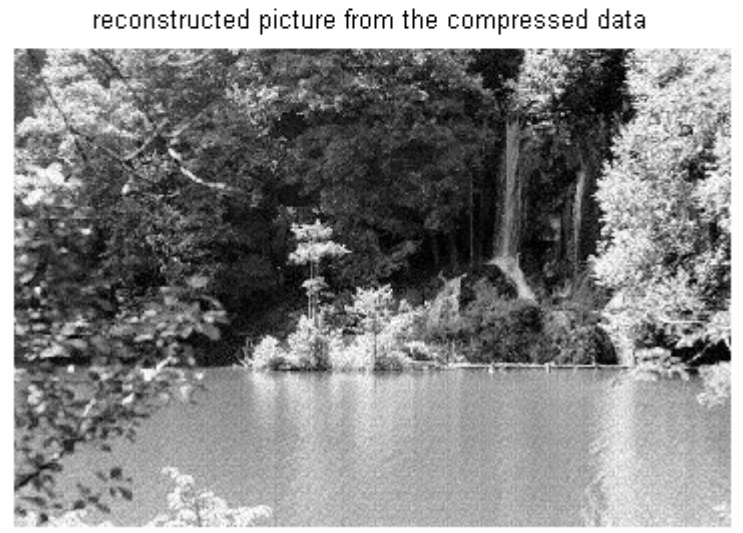
Below are numerical values from MATLAB. ‘V’ is a matrix whose columns are eigenvectors of the autocorrelation matrix calculated mathematically. ‘D’ is a diagonal matrix whose diagonal entries are the eigenvalues of the calculated autocorrelation matrix and ‘eig_estimated’ is the transpose of the final weight vectors of the network trained by GHA. The columns of this matrix refer to the estimated eigenvectors through PCA.

```
V =

   -0.3724   -0.4068    0.8342
   -0.1941   -0.8448   -0.4987
    0.9075   -0.3476    0.2356

>> D

D =

    0.0083         0         0
         0    0.2181         0
         0         0    1.3836

>> eig_estimated

eig_estimated =

    0.8366    0.4103    0.3700
   -0.4952    0.8491    0.1896
    0.2368    0.3397   -0.9101
```

Task 3B

In this task Principal component analysis is used to compress a given picture of size 280-by-420 pixels. Each pixel has a real value in [0,1]. In the task, picture was divided into 588 blocks of size 10-by-20. The pixel values in each block were aligned into a vector to give 588 vectors of size 200-by-1. These 200 dimensional vectors were used as the training sequence for the network that is trained by Generalized Hebbian algorithm. In this task a compression ratio of 3:4 was used requiring 150 principal components to project the 200 dimensional vectors. Therefore the network was a single layer network with 150 linear neurons with each neuron having 200 weights. Below figure shows the original image on top and the image reconstructed from the compressed data at bottom.





As can be seen from above figure, the reconstructed image looks very similar to the original except for a slight reduction in the sharpness of details.

Programming Task 4

Blind Signal Separation with

Independent Component Analysis (ICA)

by means of neural training

Task 4A

In this task two Laplacian distributed sequences of length 10000 with mean zero and variance 2 were created and mixed using an arbitrarily chosen 2-by-2 mixing matrix A to give two mixed sequences. Then a neural network was trained using Independent Component Analysis (ICA) algorithm. The training sequence was the 10000 vectors with 2 entries which are the sample values of the two mixed sequences. After training, the final weights W should act as the separating matrix. During training the condition number of the performance matrix P=W*A was calculated and plotted. The condition number of P should ideally go towards 1 because P should be a rescaled and permuted identity matrix. The task was repeated using 3 arbitrarily chosen mixing matrices given below,

$$A = \begin{bmatrix} 0.5 & 0.75 \\ 0.6 & 0.25 \end{bmatrix} \quad A = \begin{bmatrix} 0.64 & 0.1 \\ 0.9 & 0.4 \end{bmatrix} \quad A = \begin{bmatrix} 0.12 & 0.6 \\ 0.8 & 0.15 \end{bmatrix}$$

Plots of the condition number and the final performance matrices are given below for the 3 cases. On the plots the datatips shows the starting and ending values of the condition number.

The step size used during training is 0.0003.

Results:

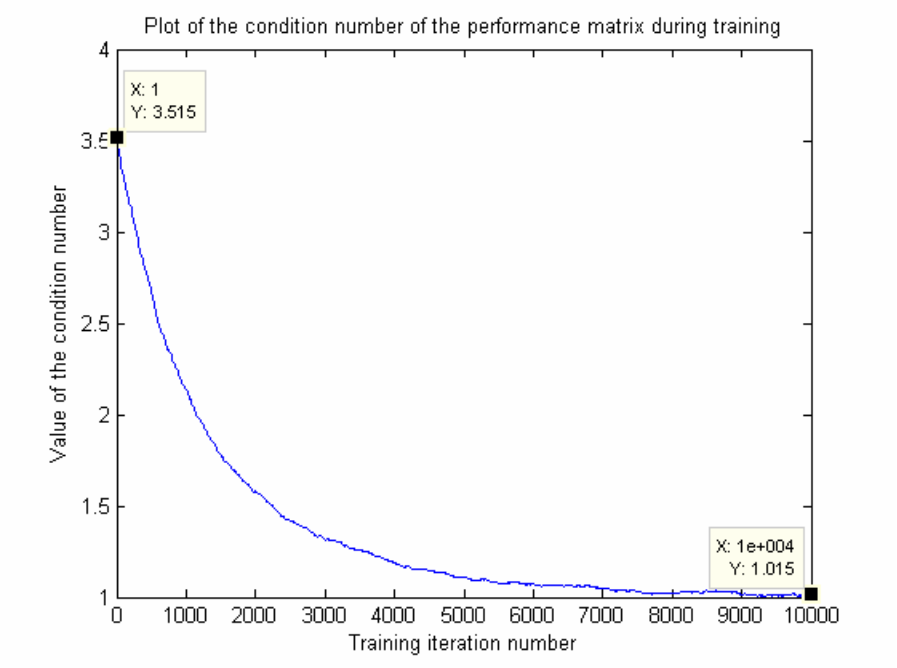
For the mixing matrix 1,

Final performance matrix

```
>> W*A
```

ans =

0.1136 0.6425
0.6573 -0.0955



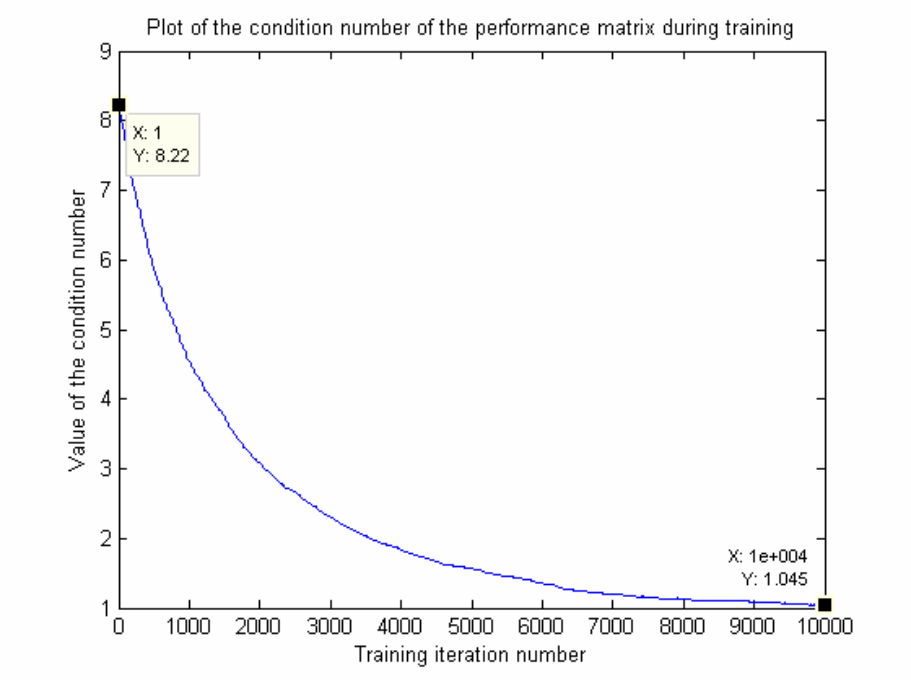
For the mixing matrix 2,

Final performance matrix

>> W*A

ans =

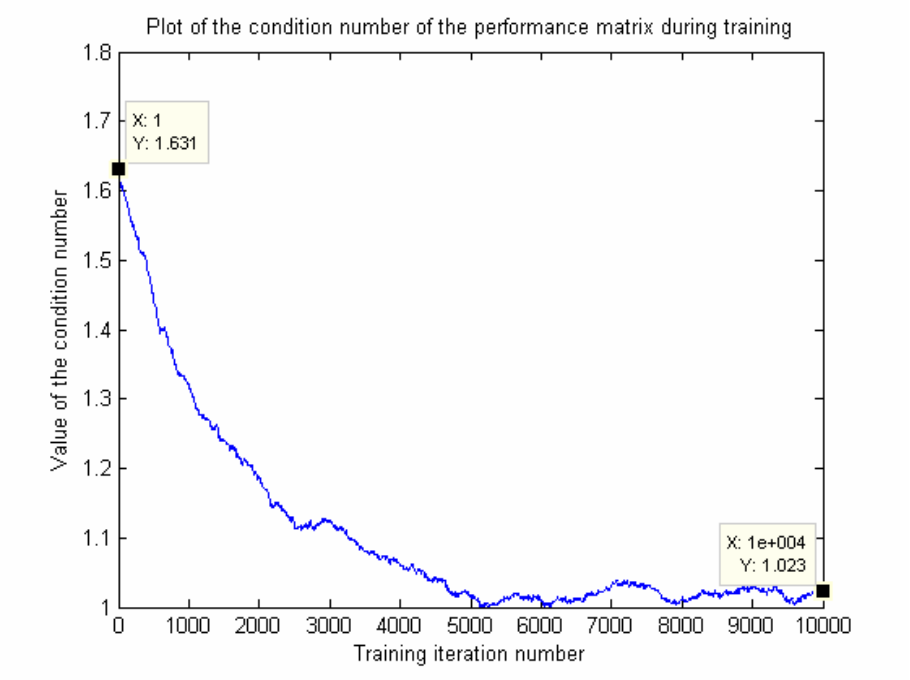
0.4359 -0.4566
0.4848 0.4345



For the mixing matrix 3,

Final performance matrix

```
>> W*A

ans =

   -0.0184    0.6783
    0.6760    0.0035
```


Plot of the condition number of the performance matrix during training
Value of the condition number
Training iteration number
X: 1
Y: 1.631
X: 1e+004
Y: 1.023
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000

Task 4B

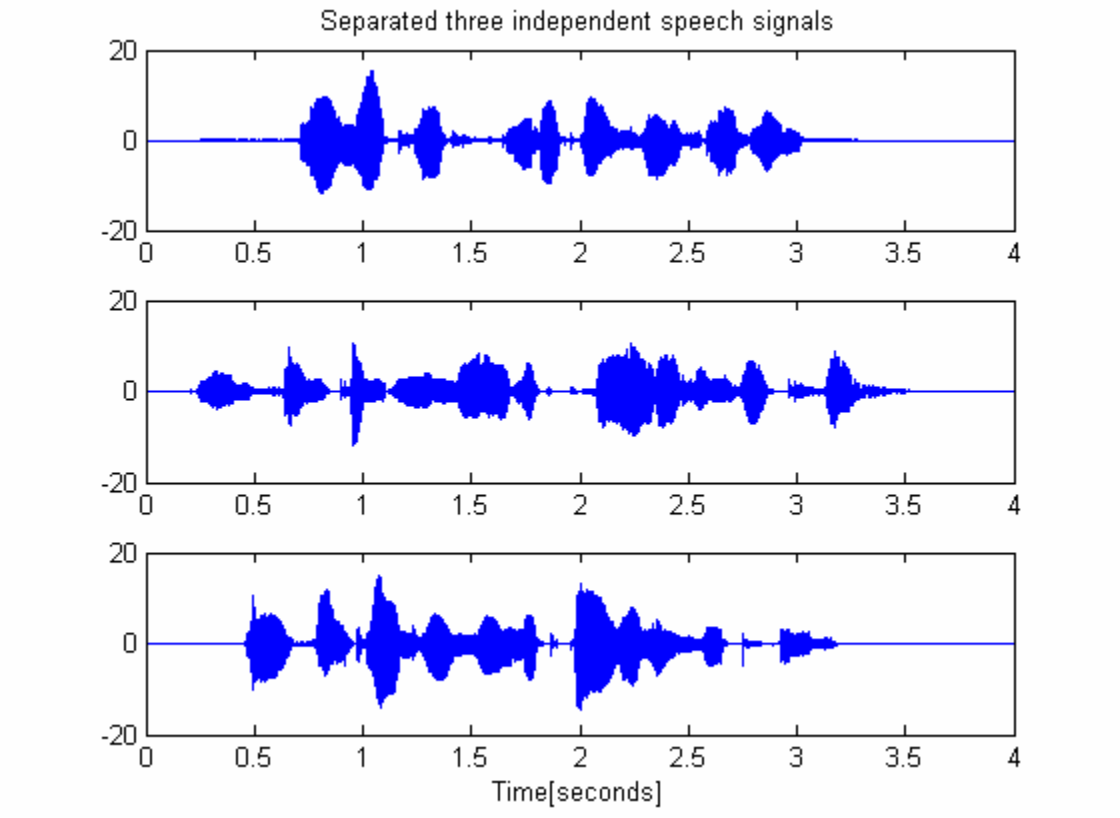
In this task a neural network trained by ICA algorithm is used to separate 3 speech signals which are given as 3 mixtures whose mixing matrix is unknown. Each mixture is 4 seconds long with a sampling frequency of 12000 Hz.

Results:

Final weight matrix,

```
>> W


W =


   13.0338   33.0890  -38.9726

  -21.3810   44.2250  -26.3172

  -27.6649   24.0339    8.7730
```

Plot of the three separated signals,



The separated signals converted to .wav format,

separated1.wav

separated2.wav

separated3.wav

As can be listened from the above files, the signals are quite well separated and can be distinguished from each other.

Programming Task 5

Radial Basis Function Networks and Genetic Algorithms for

System Identification and Filter Design

Task 5A

In this task a Radial Basis Function (RBF) network is used to approximate the same function which was used in task 1B. The function is given as,

$$f(x_1,x_2) = \sin\left(\frac{x_1^2}{4} + \frac{x_2^2}{2}\right), \; x_1 \in [-1,3], x_2 \in [-3,3]$$

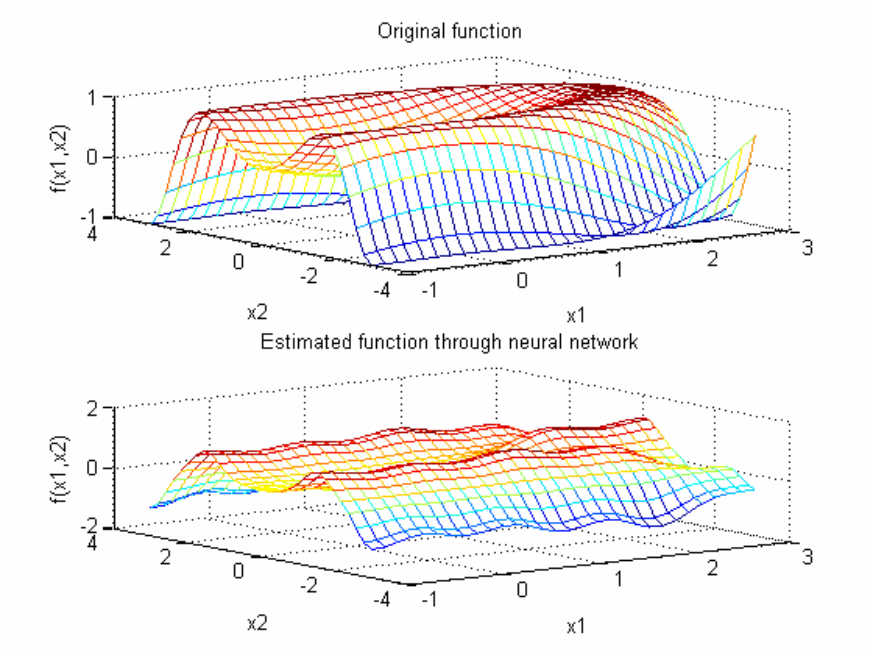
The training sequence was reused from task 1B.

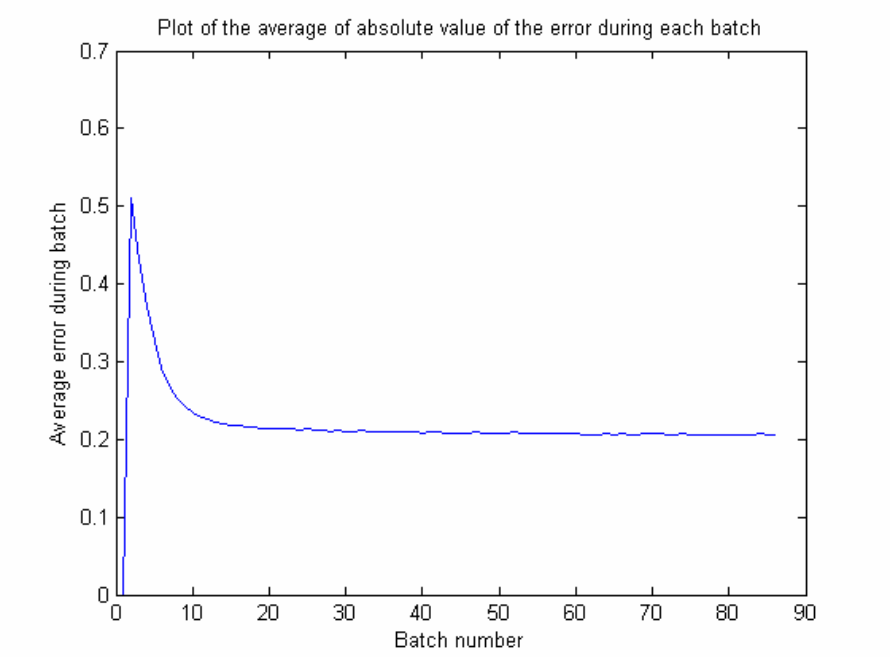
Weights of the Kohonen network were initialized to span uniformly in the input space mentioned above and kept constant while the linear layer of neurons with the bias term were trained by MIMO-LMS.

Three networks were designed with Kohonen network having 25, 100 and 400 neurons respectively. Lastly, in network 3 with 400 neurons, the weights of the last layer of linear neurons were calculated from the normal equation rather than using training.

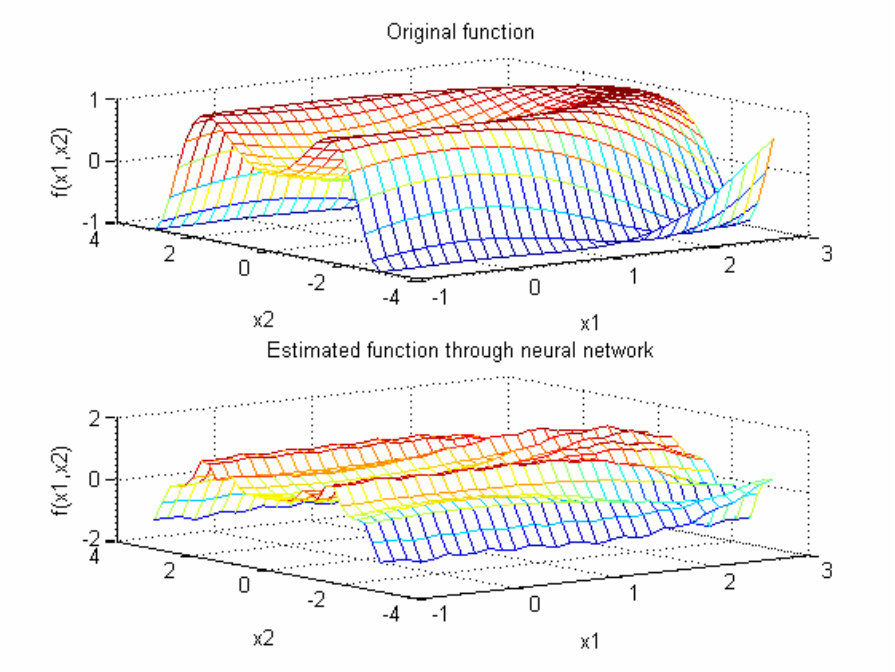
Results:

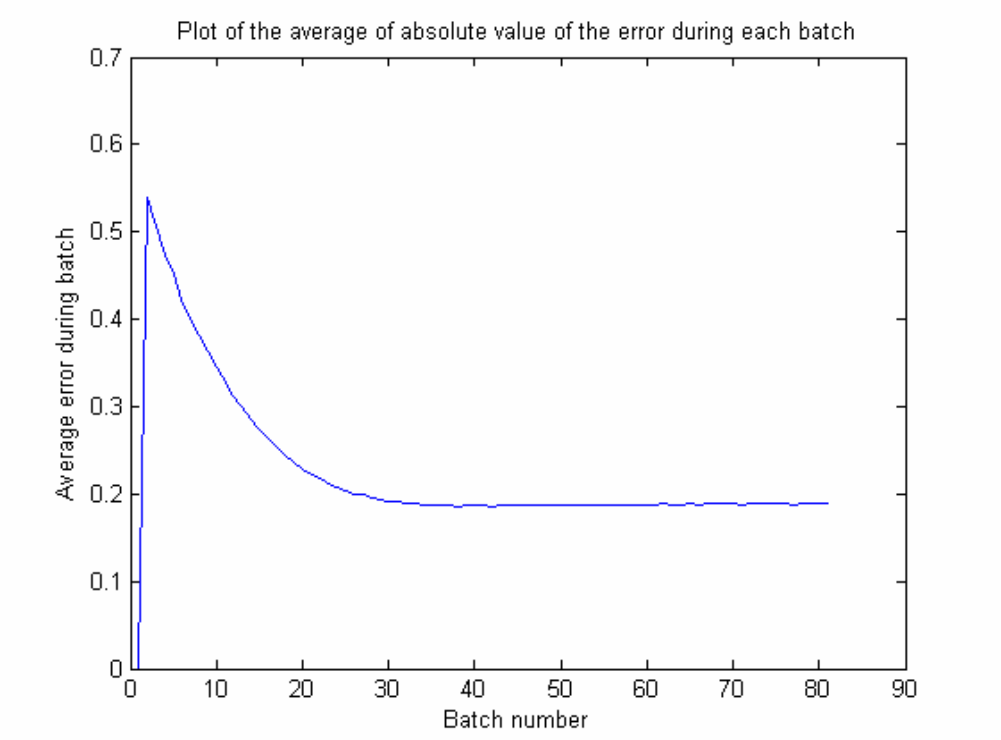
Network 1, number of neurons in the Kohonen network=25.



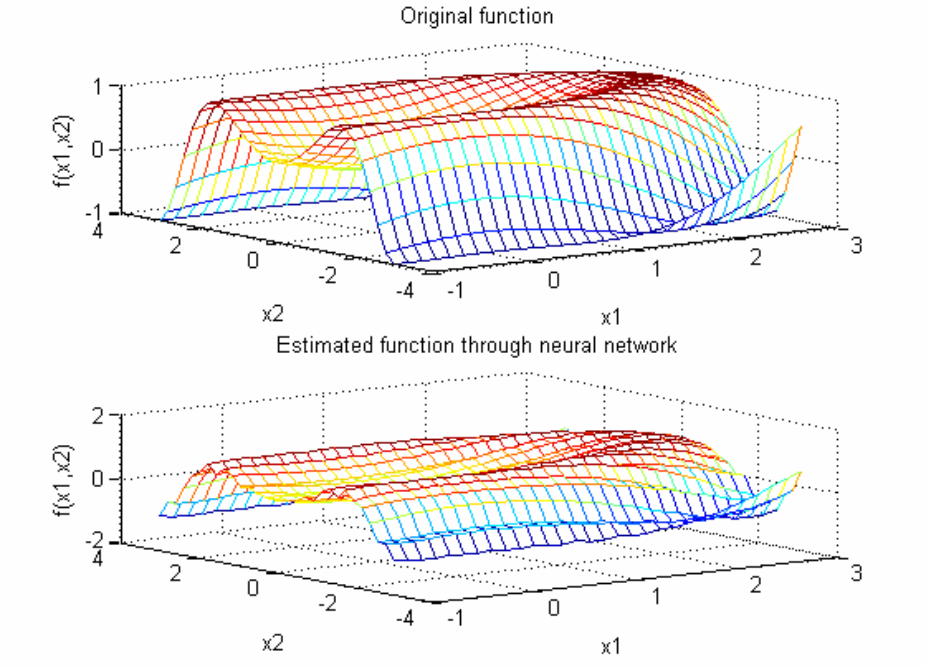


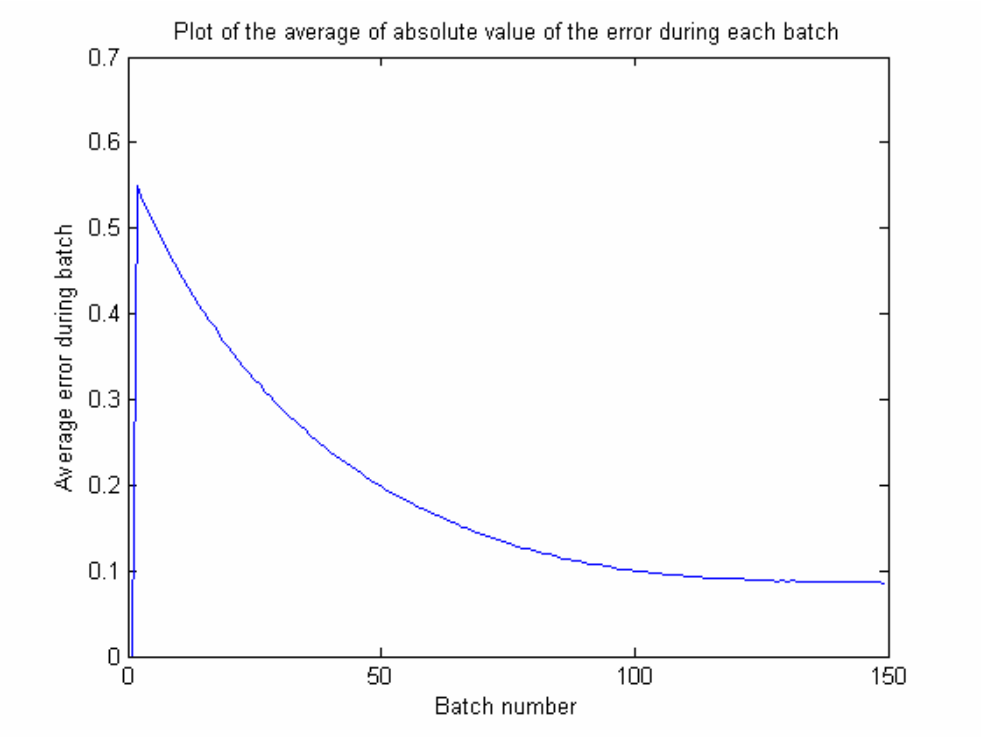
Network 2, number of neurons in the Kohonen network=100.


Original function
f(x1,x2)
1
0
-1
4
2
0
-2
-4
x2
-1
0
1
2
3
x1
Estimated function through neural network
f(x1,x2)
2
0
-2
4
2
0
-2
-4
x2
-1
0
1
2
3
x1

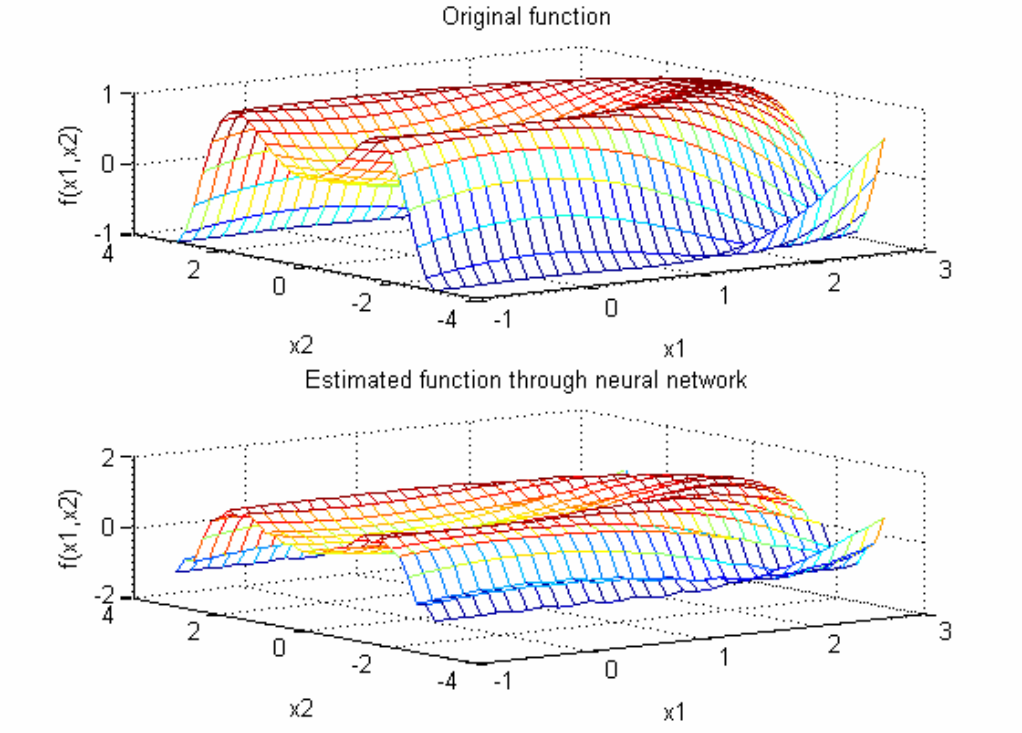


Network 3, number of neurons in the Kohonen network=400.





Network 3, number of neurons in the Kohonen network=400 and the weights of the last layer found from the normal equation.



Answers to the questions:

What impact does the number of neurons in the first layer have on the convergence rate?

Convergence rate was much higher (faster convergence) when the number of neurons in the first layer is low. There was not much difference when the networks with 25 and 100 neurons were compared, but the convergence of the network with 400 neurons was much slower requiring around 150 batches where as the first two networks converged within around 85 batches.

But the lowest achievable error was much lower (better) in the network with higher neurons in the first layer. In the first network the average batch error converged to around 0.21 and was slightly fluctuating around that value without dropping any further. For the second network this number was around 0.19 and for the third network, this number was around 0.09.

Is the convergence rate higher with RBF than with the backpropagation algorithm?

Yes. Backpropagation in task1b required around 190 batches using the momentum term where as the RBF network with 400 neurons converged in around 150 batches.

What kind of impact does the number of neurons in the first layer have on the final approximation performance?

As can be seen from the above plots, higher the number of neurons in the first layer, better is the approximation performance. Also as mentioned in the answer to the first question, the final achievable error was lower when the number of neurons in the first layer was higher.

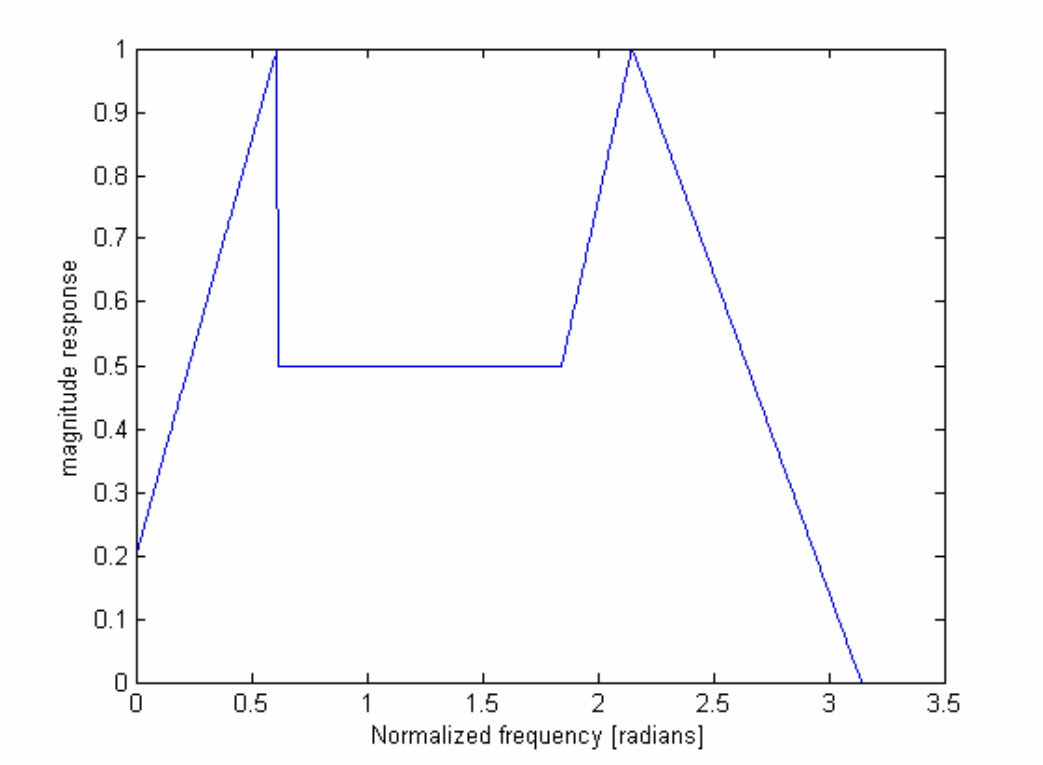
Is the approximation performance of the network whose weights were found by the equation is better than that of the one found by training?

According to the above plots, it does not seem to provide much significant improvement in the approximation. But if the input sequence was created by sampling the input space uniformly at exactly the same number of points as the number of neurons in the Kohonen network (in this case 400 points), this network gives excellent performance because in this case the neurons in the Kohonen network perfectly identifies the region that the input lies. But we have sampled the input space at 900 points rather than 400.

Task 5B

In this task genetic algorithms are used to design an odd length linear phase FIR filter approximating a given desired frequency response using both least square and minimax error criteria. Filters were designed using filter lengths 129, 65 and 33. In the genetic algorithm, the parent gives birth to 10 children as the new population at each iteration step. When the minimax error criterion was used, 5 samples before and after every transition zone was omitted when calculating the error.

Finally, a filter was designed using least square error criterion and with length 129, to approximate an arbitrarily chosen frequency response shown in the figure below,



Termination criteria:

The iteration terminates if the parent does not change during 2000 consecutive batches.

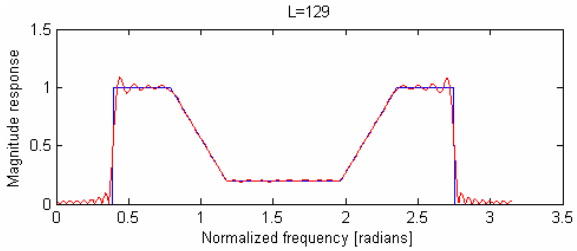
Step size and probability of mutations:

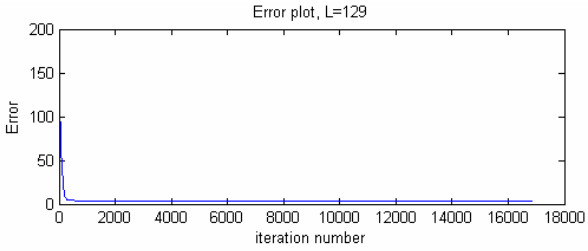
The step size was initialized to 0.01 and decreased by 0.01 % after each iteration during which a change of parent occurred.

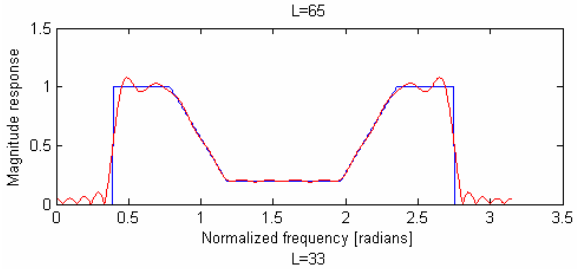
The probability p_i,which is the probability of a mutation in each entry in the parent may occur, was initialized to 0.1 and decreased by 0.01 % after each iteration during which a change of parent occurred.

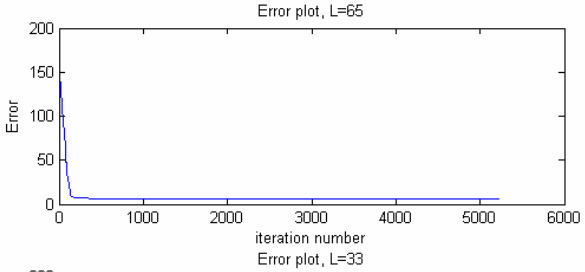
Results:

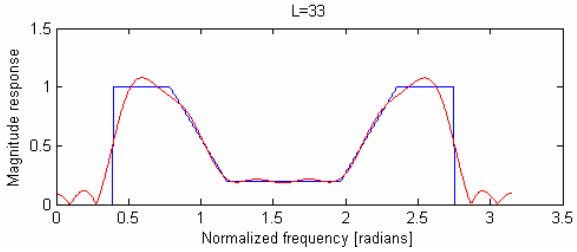
Frequency response and the error plots for filters designed using **least square error criterion** with filter lengths 129, 65 and 33. (Blue – desired; Red – designed filter using GA)

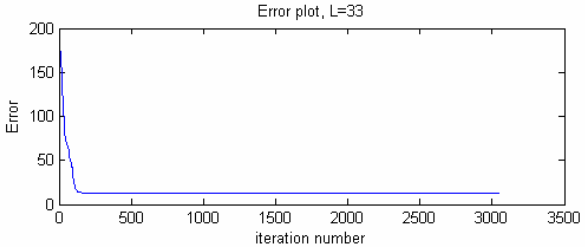




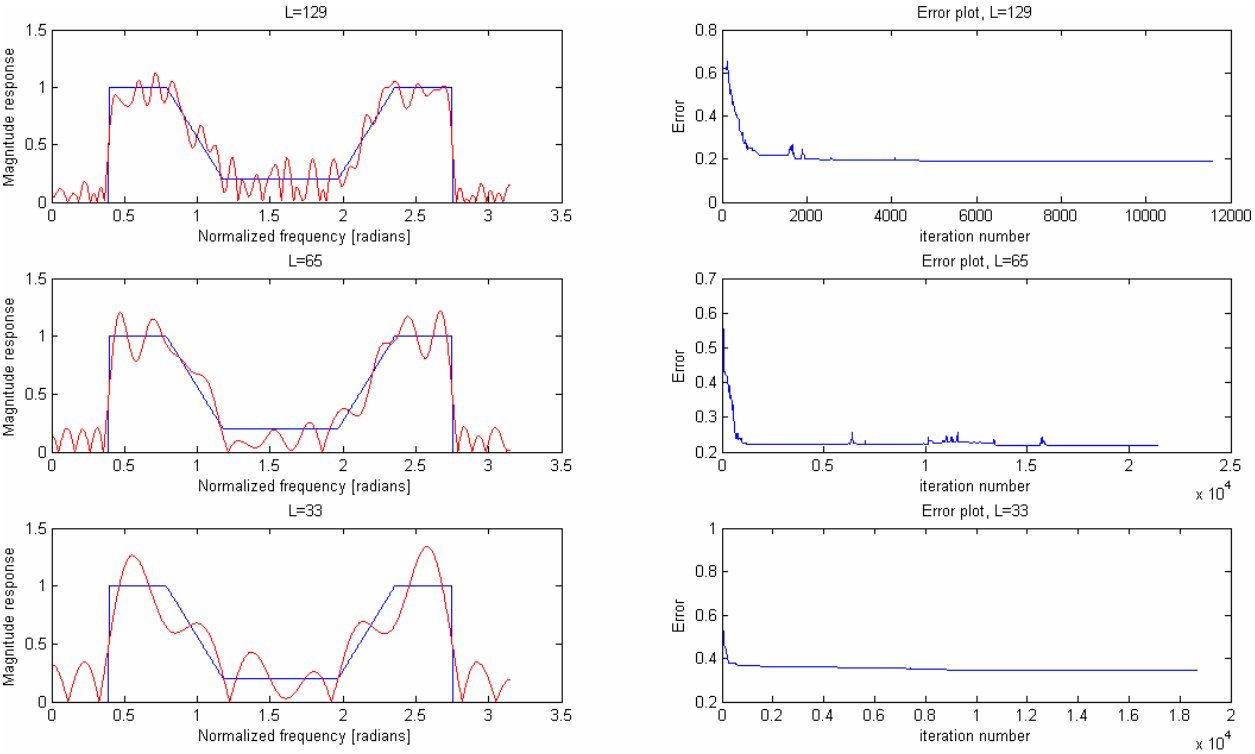




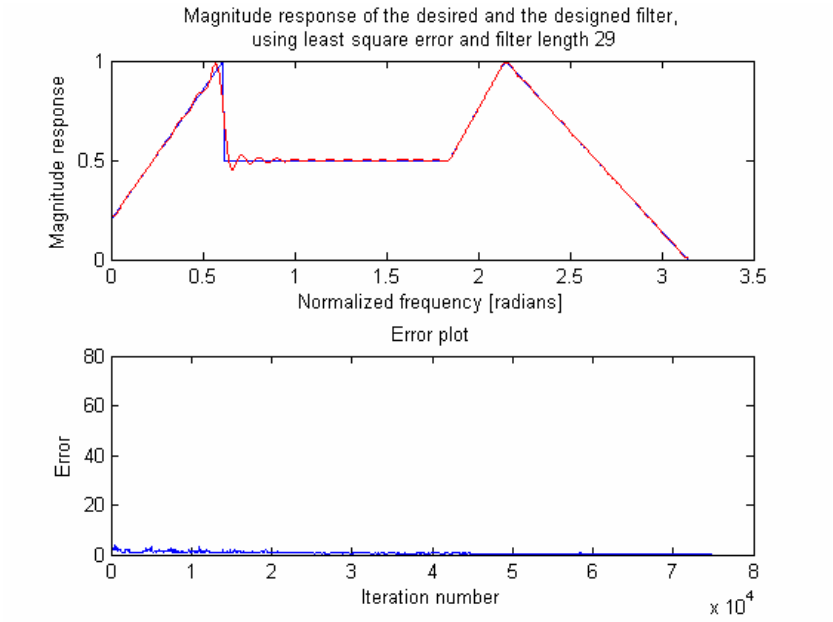




Frequency response and the error plots for filters designed using **minimax error criterion** with filter lengths 129, 65 and 33. (Blue – desired; Red – designed filter using GA)



Frequency response and error plot of the filter designed for the arbitrarily chosen desired frequency respone using least square error criterion and filter length 129. (Blue – desired; Red – designed filter using GA)



Answers to the questions:

What do you think about the convergence rate? Is it acceptable for filter design?

Convergence required few thousand iterations and hence not very acceptable for filter design since there are much more efficient and widely used filter design techniques are available.

Adjust the stepsize and the probability factor and make a comment of the sensitivity of these choices.

Lower stepsizes made the convergence slower. Also it was noted that sometimes the error stabilized around higher values than what is achievable. This may be due to local minimas and lower stepsizes made it difficult to jump out of local minimas. Higher stepsizes made the error to drop faster but made the error to fluctuate more and sometimes tends to diverge, making the convergence difficult.

Probability factor had a similar effect to stepsize. Lower probabilities makes the mutations in the children rare and made it take longer to converge and also made to get stuck in local minimas. Higher probabilities had an effect of error dropping faster but with higher fluctuations, thus making it to diverge sometimes.

Appendix – Matlab codes

Task 1A

Function "perm":

```
function [p]=perm(a)
    len=length(a);
    p=[];
    while (length(p)~=len)
        num = round((len-1)*rand) + 1;
        b=find(p==a(num));
        if (length(b)==0)
            p=[p,a(num)];
        end
    end
end
```

Main program:

```
clear all
close all
clc
%dimensions of the matrix to be inverted
m=30;
k=10;
%generating random matrix and vector b
A=randn(m,k);
b=randn(m,1);
%initializing the weights randomly
W=(rand(m,k)-0.5)/k;
t=eye(m);
%step size chosen as 0.01 and leakage factor chosen
%as zero after experimenting to generate better results.
alpha=0.01;
lambda=0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
terminate=0;
err=0;
count=0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
while (terminate==0)
    %running for each batch
    for i=perm(1:m)
        input=A(i,:)';
        %creating output
        y=W*input;
```

```
        %generating error vector
        target=t(:,i);
        e=target-y;
        %updating weights
        deltaW=alpha*(e*input');
        W=(1-lambda)*W + deltaW;
    end
    %%%%%%%%%%%%%%%%%%%%%%%%
    %reducing alpha by a factor of 0.01 after
    %each batch of iteration
    alpha=alpha*0.99;
    %calculating and storing norm of the error after
    %each batch
    err_norm=norm((A*(W'*b)) - b)
    err=[err err_norm];
    %implementing terminating criteria. Terminating criteria is that
    %absolute value of the difference between the norm of the error
    %after current batch and previous batch is less than 0.000001 for 100
    %consecutive batches.
    if (abs(err(end)-err(end-1))<0.000001)
        count=count+1;
    else
        count=0;
    end
    if (count==100)
        terminate=1;
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
end
%plotting norm of error and displaying the comparison with the
%MATLAB pinv function
figure;plot(err(2:end))
title('plot of the error norm after each bact');
xlabel('batch no.');
ylabel('error norm');
from_matlab_pinv=norm(b-A*(pinv(A)*b))
from_neural_network=norm(b-A*(W'*b))
diff_between_pinv_neural=abs(from_neural_network-from_matlab_pinv)
norm_diff_weights_pinv=norm(W'-pinv(A))
```

Task1B

```
clear all
close all
clc

%step size
alpha=0.01;
%momentum term
beta=0.3;
%beta=0;
%leaky term
gamma=0;

Err=0;
numPoints=30;
terminate=0;
prevDeltaW1=0;
prevDeltaW2=0;
prevDeltaW3=0;
count=0;

numIn=2;  %number of inputs
numL1=5;  %number of neurons in layer 1
numL2=4;  %number of neurons in layer 2
numOut=1; %number of outputs

%creating training sequence
[x1,x2]=meshgrid(linspace(-1,3,numPoints),linspace(-3,3,numPoints));
%aligning all combinations of x1 and x2 into a matrix with the
% bias term of 1
x=[x1(:) x2(:) ones(numPoints^2,1)];
%creating target sequence
f=sin((x1.^2)/4 + (x2.^2)/2);
t=f(:);
%initializing the weights
W1=(rand(numL1,numIn+1)-0.5)/(numIn+1);
W2=(rand(numL2,numL1+1)-0.5)/(numL1+1);
W3=(rand(numOut,numL2+1)-0.5)/(numL2+1);
%iteration begins
while (terminate==0)
    %starting a batch
    batchErr=[];
    for i=perm(1:length(x))
        %calculating the output "y" for the training vector
        u1=W1*x(i,:)';
        o1=[tanh(u1);1];
        u2=W2*o1;
        o2=[tanh(u2);1];
        u3=W3*o2;
        y=tanh(u3);
        %calculating the error
        e=t(i)-y;
```

```
        batchErr=[batchErr e];
        %calculating the weight update
        localErr3=diag(e)* (1-tanh(u3).^2);
        deltaW3=alpha* localErr3* o2';

        localErr2=diag(W3(:,1:end-1)'* localErr3)* (1-tanh(u2).^2);
        deltaW2=alpha* localErr2* o1';

        localErr1=diag(W2(:,1:end-1)'* localErr2)* (1-tanh(u1).^2);
        deltaW1=alpha* localErr1* x(i,:);

        %updating the weights with momentum update
        W3=(1-gamma)*W3 + deltaW3 + beta*prevDeltaW3;
        W2=(1-gamma)*W2 + deltaW2 + beta*prevDeltaW2;
        W1=(1-gamma)*W1 + deltaW1 + beta*prevDeltaW1;

        prevDeltaW1=deltaW1;
        prevDeltaW2=deltaW2;
        prevDeltaW3=deltaW3;
    end
    %ending a batch
    alpha=alpha*0.999;
    beta=beta*0.99;
    avgErr=sum(abs(batchErr))/length(x);
    Err=[Err avgErr];
    avgErr

    if (avgErr<0.1)
        count=count+1;
    else
        count=0;
    end
    if (count==50)
        terminate=1;
    end
end

%using the network to plot
f_network=[];
f_original=t;

for i=1:length(x)
    u1=W1*x(i,:)';
    o1=[tanh(u1);1];
    u2=W2*o1;
    o2=[tanh(u2);1];
    u3=W3*o2;
    y=tanh(u3);
    f_network=[f_network;y];
end

f_network =reshape(f_network,numPoints,numPoints);
f_original=reshape(f_original,numPoints,numPoints);
figure
subplot(2,1,1);
```

```
mesh(linspace(-1,3,numPoints),linspace(-3,3,numPoints),f_original);
title('Original function');
xlabel('x1');
ylabel('x2');
zlabel('f(x1,x2)');

subplot(2,1,2);
mesh(linspace(-1,3,numPoints),linspace(-3,3,numPoints),f_network);
title('Estimated function through neural network');
xlabel('x1');
ylabel('x2');
zlabel('f(x1,x2)');

figure
plot(Err)
title('Plot of the average of absolute value of the error during each batch')
xlabel('Batch number');
ylabel('Average error during batch');
```

Task 1C

```
clear all
 close all
 clc

 N=1001;
 %absolute value was taken to avoid negative value that may result in
 %complex complex values for "y"
 x=abs(randn(N,1));
 x1=filter([0 1],1,x);
 x2=filter([0 0 1],1,x);
 x3=filter([0 0 0 1],1,x);

 y=sqrt(0.1*x+0.5*x1) + 0.3*(x2.^2) + sin(x3);
 y1=filter([0 1],1,y);
 y2=filter([0 0 1],1,y);
 y3=filter([0 0 0 1],1,y);

 %creating sequences of current input and 3 last inputs and  current output
to be presented to
 %the neural network as training sequences. Therefore the training sequence
 %will be [ x(n) x(n-1) x(n-2) x(n-3) y(n) ].
 for i=1:length(x)-1
     X(:,i)=[x(i) x1(i) x2(i) x3(i) y(i) y1(i) y2(i) y3(i)]';
 end

 %creating the target sequence "t" which is the sequence of y(n+)
t=y(2:end);

%step size
alpha=0.01;
%momentum term
beta=0.3;
%leaky term
gamma=0;

Err=0;
terminate=0;
prevDeltaW1=0;
prevDeltaW2=0;
prevDeltaW3=0;
count=0;

numIn=8;  %number of inputs
numL1=10;  %number of neurons in layer 1
numL2=5;  %number of neurons in layer 2
numOut=1; %number of outputs

%initializing the weights
W1=(rand(numL1,numIn+1)-0.5)/(numIn+1);
W2=(rand(numL2,numL1+1)-0.5)/(numL1+1);
```

```
W3=(rand(numOut,numL2+1)-0.5)/(numL2+1);

while (terminate==0)
    %starting a batch
    batchErr=[];
    for i=perm(1:length(x)-1)
        %calculating the output "y" for the training vector
        u1=W1*[X(:,i);1]; %with bias
        o1=[tanh(u1);1];
        u2=W2*o1;
        o2=[tanh(u2);1];
        u3=W3*o2;
        o3=u3; %linear neuron, no bipolar sigmoid function
        %calculating the error
        e=t(i)-o3;
        batchErr=[batchErr e];
        %calculating the weight update
        localErr3=diag(e)* 1; %derivative is one due to linear neuron
        deltaW3=alpha* localErr3* o2';

        localErr2=diag(W3(:,1:end-1)'* localErr3)* (1-tanh(u2).^2);
        deltaW2=alpha* localErr2* o1';

        localErr1=diag(W2(:,1:end-1)'* localErr2)* (1-tanh(u1).^2);
        deltaW1=alpha* localErr1* [X(:,i);1]';

        %updating the weights with momentum update
        W3=(1-gamma)*W3 + deltaW3 + beta*prevDeltaW3;
        W2=(1-gamma)*W2 + deltaW2 + beta*prevDeltaW2;
        W1=(1-gamma)*W1 + deltaW1 + beta*prevDeltaW1;

        prevDeltaW1=deltaW1;
        prevDeltaW2=deltaW2;
        prevDeltaW3=deltaW3;
    end

    beta=beta*0.999;
    alpha=alpha*0.9999;
    avgErr=sum(abs(batchErr))/(length(x)-1);
    Err=[Err avgErr];
    avgErr

    if (avgErr<0.04)
        count=count+1;
    else
        count=0;
    end
    if (count==50)
        terminate=1;
    end
end

figure
plot(Err)
title('Plot of the average of absolute value of the error during each batch')
```

```
xlabel('Batch number');
ylabel('Average error during batch');
```

Task 2A

```
%Task 2A
clear all
close all
clc

load digits.mat

%plotting the original (correct) digits
figure; title('original digits');
colormap([1 1 1;0 0 0])
subplot(2,4,1),imagesc(digits(:,:,1));
title('Digit:"0"');
subplot(2,4,2),imagesc(digits(:,:,2));
title('Digit:"1"');
subplot(2,4,3),imagesc(digits(:,:,3));
title('Digit:"2"');
subplot(2,4,4),imagesc(digits(:,:,4));
title('Digit:"3"');
subplot(2,4,5),imagesc(digits(:,:,5));
title('Digit:"4"');
subplot(2,4,6),imagesc(digits(:,:,6));
title('Digit:"6"');
subplot(2,4,7),imagesc(digits(:,:,7));
title('Digit:"\prime"');
subplot(2,4,8),imagesc(digits(:,:,8));
title('Digit:"9"');

%creating the input matrix
X=zeros(120,8);
for i=1:8
    x_temp=digits(:,:,i);
    X(:,i)=x_temp(:);
end

%%%%%%%%%%%%%%%%%%Using Hebbian paradigm%%%%%%%%%%%%%%%%
%creating auto-associative memory using Hebbian paradigm
W=X*(X');

%processing through the neural network for distorted digits
Y=zeros(120,6);
for j=1:6
    x_temp=distorted_digits(:,:,j);
    Y(:,j)=sign(W*x_temp(:));
end
%plotting reconstructed digits
figure;
colormap([1 1 1;0 0 0])
subplot(2,3,1),imagesc(reshape(Y(:,1),12,10));
title('Digit:"0"');
subplot(2,3,2),imagesc(reshape(Y(:,2),12,10));
```

```
title('Digit:"1"');
subplot(2,3,3),imagesc(reshape(Y(:,3),12,10));
title('Digit:"2"');
subplot(2,3,4),imagesc(reshape(Y(:,4),12,10));
title('Digit:"3"');
subplot(2,3,5),imagesc(reshape(Y(:,5),12,10));
title('Digit:"4"');
subplot(2,3,6),imagesc(reshape(Y(:,6),12,10));
title('Digit:"\prime"');

%%%%%%%%%%%%%%%%%%Using pseudoinverse%%%%%%%%%%%%%%%%
%creating auto-associative memory using pseudoinverse
W=X*pinv(X);

%processing through the neural network for distorted digits
Y=zeros(120,6);
for j=1:6
    x_temp=distorted_digits(:,:,j);
    Y(:,j)=sign(W*x_temp(:));
end
%plotting reconstructed digits
figure;
colormap([1 1 1;0 0 0])
subplot(2,3,1),imagesc(reshape(Y(:,1),12,10));
title('Digit:"0"');
subplot(2,3,2),imagesc(reshape(Y(:,2),12,10));
title('Digit:"1"');
subplot(2,3,3),imagesc(reshape(Y(:,3),12,10));
title('Digit:"2"');
subplot(2,3,4),imagesc(reshape(Y(:,4),12,10));
title('Digit:"3"');
subplot(2,3,5),imagesc(reshape(Y(:,5),12,10));
title('Digit:"4"');
subplot(2,3,6),imagesc(reshape(Y(:,6),12,10));
title('Digit:"\prime"');
```

Task 2B

```
%Task 2B
clear all
close all
clc

load digits.mat

%creating the input matrix
X=zeros(120,8);
for i=1:8
    x_temp=digits(:,:,i);
    X(:,i)=x_temp(:);
end

%finding the weight matrix W using hebbian paradigm with modification
%to make the diagonal entries zero to prevent self-feedback

W=X*(X') - 8*eye(120,120);
%uncomment below two lines to use pseudoinverse memory
%W=X*pinv(X);
%W=W-diag(diag(W));

%processing through the neural network for distorted digits
Y=zeros(120,6);
for i=1:6 %looping for each digit (1 to 6)
    s_new=distorted_digits(:,:,i);
    s_new=s_new(:);
    s_old=zeros(length(s_new),1);
    while (sum(abs(s_new-s_old))~=0)
        s_old=s_new;
        %%%%% updating each element of 's' in random order
        for j=perm(1:length(s_new))
            s_new(j)=sign(W(j,:)*s_new);
        end
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    end
    Y(:,i)=s_new;
end

%plotting reconstructed digits
figure;
colormap([1 1 1;0 0 0])
subplot(2,3,1),imagesc(reshape(Y(:,1),12,10));
title('Digit:"0"');
subplot(2,3,2),imagesc(reshape(Y(:,2),12,10));
title('Digit:"1"');
subplot(2,3,3),imagesc(reshape(Y(:,3),12,10));
title('Digit:"2"');
subplot(2,3,4),imagesc(reshape(Y(:,4),12,10));
title('Digit:"3"');
```

```
subplot(2,3,5),imagesc(reshape(Y(:,5),12,10));
title('Digit:"4"');
subplot(2,3,6),imagesc(reshape(Y(:,6),12,10));
title('Digit:"\prime"');
% for i=1:6
%     subplot(2,3,i),imagesc(reshape(Y(:,i),12,10));
%     title('Digit:"?"');
% end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%processing through the neural network for heavily distorted digits
Y=zeros(120,6);
for i=1:6 %looping for each digit (1 to 6)
    s_new=heavily_distorted_digits(:,:,i);
    s_new=s_new(:);
    s_old=zeros(length(s_new),1);
    while (sum(abs(s_new-s_old))~=0)
        s_old=s_new;
        %%%%% updating each element of 's' in random order
        for j=perm(1:length(s_new))
            s_new(j)=sign(W(j,:)*s_new);
        end
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    end
    Y(:,i)=s_new;
end

%plotting reconstructed digits
figure;
colormap([1 1 1;0 0 0])
for i=1:6
    subplot(2,3,i),imagesc(reshape(Y(:,i),12,10));
    title('Digit:"?"');
end
```

Task 3A

```
%task 3a
close all
clear all
clc

Rxx=[1 -0.5 0.3;-0.5 0.5 -0.1;0.3 -0.1 0.11];

%calculating theoritical eigenvectors
[V,D]=eig(Rxx);

%------------using neural network to estimate the eigenvectors-----------
N=5000;
eps1=randn(1,N);
eps2=randn(1,N);
eps3=randn(1,N);

x1 = eps1;
x2 = -0.5*x1 + 0.5*eps2;
x3 = 0.4*x1 + 0.2*x2 + 0.1*eps3;

X=[x1;x2;x3];

w=rand(3,3);

terminate=0;
alpha=0.01;
E=[];
while (terminate==0)
    w_old=w;
    for i=perm(1:N)
        in=X(:,i);
        y=w*in;
        w_new = w + alpha*(y*in' - tril(y*y')*w);
        w=w_new;
    end
    err=norm(w_old-w_new)
    E=[E err];
    if (err < 0.02)
        terminate=1;
    end
end
eig_estimated=w';

x1=[0 V(1,1)];y1=[0 V(2,1)];z1=[0 V(3,1)];
x2=[0 V(1,2)];y2=[0 V(2,2)];z2=[0 V(3,2)];
x3=[0 V(1,3)];y3=[0 V(2,3)];z3=[0 V(3,3)];
figure; plot3(x1,y1,z1);
hold on
plot3(x2,y2,z2);
plot3(x3,y3,z3);
```

```
x1=[0 eig_estimated(1,1)];y1=[0 eig_estimated(2,1)];z1=[0
eig_estimated(3,1)];
x2=[0 eig_estimated(1,2)];y2=[0 eig_estimated(2,2)];z2=[0
eig_estimated(3,2)];
x3=[0 eig_estimated(1,3)];y3=[0 eig_estimated(2,3)];z3=[0
eig_estimated(3,3)];
plot3(x1,y1,z1,'r');
plot3(x2,y2,z2,'r');
plot3(x3,y3,z3,'r');
xlabel('X');
ylabel('Y');
zlabel('Z');
hold off
```

Task 3B

```
%Task 3b
clear all
close all
clc

load picture1

input=[];
for i=1:280/10
    for j=1:420/20
        temp=G(((i-1)*10+1):i*10,((j-1)*20+1):j*20);
        input=[input temp(:)];
    end
end

N=588;

w=0.1*(rand(150,200)-0.5);

terminate=0;
alpha=0.01;
E=[];

while (terminate==0)
    w_old=w;
    for i=perm(1:N)
        in=input(:,i);
        y=w*in;
        w = w + alpha*(y*in' - tril(y*y')*w);
    end
    err=norm(w_old-w)
    E=[E err];
    if (err < 0.5)
        terminate=1;
    end
end

compressed=w*input;

recons1=pinv(w)*compressed;

recons2=zeros(280,420);
k=1;
for i=1:10:280
    for j=1:20:420
        temp=reshape(recons1(:,k),10,20);
        recons2(i:i+10-1,j:j+20-1)=temp;
        k=k+1;
    end
```

```
end
figure;
subplot(2,1,1);imshow(G);title('original picture');
subplot(2,1,2);imshow(recons2);title('reconstructed picture from the
compressed data')
```

Task 4A

```
%Task 4a
close all
clear all
clc

alpha=0.0003;

len=10000;
variance=2;
mean=0;

a=mean;
b=sqrt(variance/2);

X=lapdist(a,b,2,len);

%A=[0.5 0.75;0.6 0.25];
%A=[0.64 0.1;0.9 0.4];
A=[0.12 0.6;0.8 0.15];

U=A*X;

W=eye(2);
n=1;
condition=[];

for i=1:len
    i
    s=W*U(:,i);
    W=W + alpha * (eye(2) - 2*tanh(s)*s')*W;
    condition=[condition cond(W*A)];
end

S=W*U;
figure,plot(condition);
title('Plot of the condition number of the performance matrix during
training');
xlabel('Training iteration number');
ylabel('Value of the condition number');
```

Task 4B

```
%task 4b
close all
clear all
clc

load mixtures.mat
t=linspace(0,4,12000*4);

alpha=0.00005;

len=length(U);

W=eye(3);
n=1;

for j=1:10
    for i=1:len
        s=W*U(:,i);
        W=W + alpha * (eye(3) - 2*tanh(s)*s')*W;
    end
end

S=W*U;
figure;
subplot(3,1,1),plot(t,S(1,:));
title('Separated three independent speech signals');
subplot(3,1,2),plot(t,S(2,:));
subplot(3,1,3),plot(t,S(3,:));
xlabel('Time[seconds]');
```

Task 5A

Matlab code for the RBF network 1 with 25 neurons in the first layer

```
clear all
close all
clc

%step size
alpha=0.01;
%momentum term
beta=0.5;
%leaky term
gamma=0;

Err=0;
numPoints=30;
terminate=0;
count=0;
M=25; %number of neurons in the Kohonen network

%creating training sequence
[x1,x2]=meshgrid(linspace(-1,3,numPoints),linspace(-3,3,numPoints));
%aligning all combinations of x1 and x2 into a matrix
x=[x1(:) x2(:)];
%creating target sequence
f=sin((x1.^2)/4 + (x2.^2)/2);
t=f(:);

%initializing the weights for the Kohonen network
[w1,w2]=meshgrid(linspace(-1,3,sqrt(M)),linspace(-3,3,sqrt(M)));
W1=[w1(:) w2(:)];

%finding distance between the weights in the kohonen network
d=norm(W1(1,:) - W1(end,:));

%initializing the weights for the MIMO LMS
W2=(rand(1,M+1)-0.5)/(M+1);

prevDeltaW2=0;
%iteration begins
while (terminate==0)
    %starting a batch
    batchErr=0;
    for i=perm(1:length(x))
        inp=x(i,:);
        %calculating the output "g1" of the Kohonen network for the selected
input
        g1=zeros(M,1);
        for j=1:M
            g1(j,1)=norm(W1(j,:)-inp);
        end
```

```
        %calculating the output "g2" of the radial basis function
        g2=exp((-M*(g1.^2))/d);
        %adding the bias term to be input to MIMO LMS
        g2=[g2;1];
        %calculating the output "y" of the MIMO LMS
        y=W2*g2;
        %calculating the error
        e=t(i)-y;
        batchErr=batchErr+abs(e);
        %calculating the weight update for MIMO LMS
        deltaW2=alpha*e*g2';
        %updating the weights for the MIMO LMS
        W2=(1-gamma)*W2 + deltaW2 +beta*prevDeltaW2;
        prevDeltaW2=deltaW2;
    end
    %ending a batch
    alpha=alpha*0.999;
    beta=beta*0.99
    avgErr=batchErr/length(x);
    Err=[Err avgErr];
    avgErr

    if (avgErr<0.21)
        count=count+1;
    else
        count=0;
    end
    if (count==50)
        terminate=1;
    end
end

%using the network to plot
f_network=[];
f_original=t;

for i=1:length(x)
    inp=x(i,:);
    g1=zeros(M,1);
    for j=1:M
        g1(j,1)=norm(W1(j,:)-inp);
    end
    g2=exp((-M*(g1.^2))/d);
    g2=[g2;1];
    y=W2*g2;
    f_network=[f_network;y];
end

f_network =reshape(f_network,numPoints,numPoints);
f_original=reshape(f_original,numPoints,numPoints);
figure
subplot(2,1,1);
mesh(linspace(-1,3,numPoints),linspace(-3,3,numPoints),f_original);
title('Original function');
xlabel('x1');
ylabel('x2');
```

```
zlabel('f(x1,x2)');

subplot(2,1,2);
mesh(linspace(-1,3,numPoints),linspace(-3,3,numPoints),f_network);
title('Estimated function through neural network');
xlabel('x1');
ylabel('x2');
zlabel('f(x1,x2)');

figure
plot(Err)
title('Plot of the average of absolute value of the error during each batch')
xlabel('Batch number');
ylabel('Average error during batch');
```

Matlab code for the RBF network 2 with 100 neurons in the first layer

```
clear all
close all
clc

%step size
alpha=0.01;
%momentum term
beta=0.3;
%leaky term
gamma=0;

Err=0;
numPoints=30;
terminate=0;
count=0;
M=100; %number of neurons in the Kohonen network

%creating training sequence
[x1,x2]=meshgrid(linspace(-1,3,numPoints),linspace(-3,3,numPoints));
%aligning all combinations of x1 and x2 into a matrix
x=[x1(:) x2(:)];
%creating target sequence
f=sin((x1.^2)/4 + (x2.^2)/2);
t=f(:);

%initializing the weights for the Kohonen network
[w1,w2]=meshgrid(linspace(-1,3,sqrt(M)),linspace(-3,3,sqrt(M)));
W1=[w1(:) w2(:)];

%finding distance between the weights in the kohonen network
d=norm(W1(1,:) - W1(end,:));

%initializing the weights for the MIMO LMS
W2=(rand(1,M+1)-0.5)/(M+1);

prevDeltaW2=0;
%iteration begins
while (terminate==0)
    %starting a batch
    batchErr=0;
    for i=perm(1:length(x))
        inp=x(i,:);
        %calculating the output "g1" of the Kohonen network for the selected
input
        g1=zeros(M,1);
        for j=1:M
            g1(j,1)=norm(W1(j,:)-inp);
        end
        %calculating the output "g2" of the radial basis function
        g2=exp((-M*(g1.^2))/d);
        %adding the bias term to be input to MIMO LMS
```

```
        g2=[g2;1];
        %calculating the output "y" of the MIMO LMS
        y=W2*g2;
        %calculating the error
        e=t(i)-y;
        batchErr=batchErr+abs(e);
        %calculating the weight update for MIMO LMS
        deltaW2=alpha*e*g2';
        %updating the weights for the MIMO LMS
        W2=(1-gamma)*W2 + deltaW2 +beta*prevDeltaW2;
        prevDeltaW2=deltaW2;
    end
    %ending a batch
    alpha=alpha*0.999;
    beta=beta*0.99;
    avgErr=batchErr/length(x);
    Err=[Err avgErr];
    avgErr

    if (avgErr<0.19)
        count=count+1;
    else
        count=0;
    end
    if (count==50)
        terminate=1;
    end
end

%using the network to plot
f_network=[];
f_original=t;

for i=1:length(x)
    inp=x(i,:);
    g1=zeros(M,1);
    for j=1:M
        g1(j,1)=norm(W1(j,:)-inp);
    end
    g2=exp((-M*(g1.^2))/d);
    g2=[g2;1];
    y=W2*g2;
    f_network=[f_network;y];
end

f_network =reshape(f_network,numPoints,numPoints);
f_original=reshape(f_original,numPoints,numPoints);
figure
subplot(2,1,1);
mesh(linspace(-1,3,numPoints),linspace(-3,3,numPoints),f_original);
title('Original function');
xlabel('x1');
ylabel('x2');
zlabel('f(x1,x2)');

subplot(2,1,2);
```

```
mesh(linspace(-1,3,numPoints),linspace(-3,3,numPoints),f_network);
title('Estimated function through neural network');
xlabel('x1');
ylabel('x2');
zlabel('f(x1,x2)');

figure
plot(Err)
title('Plot of the average of absolute value of the error during each batch')
xlabel('Batch number');
ylabel('Average error during batch');
```

Matlab code for the RBF network 3 with 400 neurons in the first layer

```
clear all
close all
clc

%step size
alpha=0.01;
%momentum term
beta=0.3;
%leaky term
gamma=0;

Err=0;
numPoints=30;
terminate=0;
count=0;
M=400; %number of neurons in the Kohonen network

%creating training sequence
[x1,x2]=meshgrid(linspace(-1,3,numPoints),linspace(-3,3,numPoints));
%aligning all combinations of x1 and x2 into a matrix
x=[x1(:) x2(:)];
%creating target sequence
f=sin((x1.^2)/4 + (x2.^2)/2);
t=f(:);

%initializing the weights for the Kohonen network
[w1,w2]=meshgrid(linspace(-1,3,sqrt(M)),linspace(-3,3,sqrt(M)));
W1=[w1(:) w2(:)];

%finding distance between the weights in the kohonen network
d=norm(W1(1,:) - W1(end,:));

%initializing the weights for the MIMO LMS
W2=(rand(1,M+1)-0.5)/(M+1);

prevDeltaW2=0;
%iteration begins
while (terminate==0)
    %starting a batch
    batchErr=0;
    for i=perm(1:length(x))
        inp=x(i,:);
        %calculating the output "g1" of the Kohonen network for the selected
input
        g1=zeros(M,1);
        for j=1:M
            g1(j,1)=norm(W1(j,:)-inp);
        end
        %calculating the output "g2" of the radial basis function
        g2=exp((-M*(g1.^2))/d);
        %adding the bias term to be input to MIMO LMS
```

```
        g2=[g2;1];
        %calculating the output "y" of the MIMO LMS
        y=W2*g2;
        %calculating the error
        e=t(i)-y;
        batchErr=batchErr+abs(e);
        %calculating the weight update for MIMO LMS
        deltaW2=alpha*e*g2';
        %updating the weights for the MIMO LMS
        W2=(1-gamma)*W2 + deltaW2 +beta*prevDeltaW2;
        prevDeltaW2=deltaW2;
    end
    %ending a batch
    alpha=alpha*0.999;
    beta=beta*0.99
    avgErr=batchErr/length(x);
    Err=[Err avgErr];
    avgErr

    if (avgErr<0.1)
        count=count+1;
    else
        count=0;
    end
    if (count==50)
        terminate=1;
    end
end

%using the network to plot
f_network=[];
f_original=t;

for i=1:length(x)
    inp=x(i,:);
    g1=zeros(M,1);
    for j=1:M
        g1(j,1)=norm(W1(j,:)-inp);
    end
    g2=exp((-M*(g1.^2))/d);
    g2=[g2;1];
    y=W2*g2;
    f_network=[f_network;y];
end

f_network =reshape(f_network,numPoints,numPoints);
f_original=reshape(f_original,numPoints,numPoints);
figure
subplot(2,1,1);
mesh(linspace(-1,3,numPoints),linspace(-3,3,numPoints),f_original);
title('Original function');
xlabel('x1');
ylabel('x2');
zlabel('f(x1,x2)');

subplot(2,1,2);
```

```
mesh(linspace(-1,3,numPoints),linspace(-3,3,numPoints),f_network);
title('Estimated function through neural network');
xlabel('x1');
ylabel('x2');
zlabel('f(x1,x2)');

figure
plot(Err)
title('Plot of the average of absolute value of the error during each batch')
xlabel('Batch number');
ylabel('Average error during batch');
```

Matlab code for the RBF network 4 with 400 neurons in the first layer and the weights of the second layer found by solving the equation

```
clear all
close all
clc

numPoints=30;
M=400; %number of neurons in the Kohonen network

%creating input sequence
[x1,x2]=meshgrid(linspace(-1,3,numPoints),linspace(-3,3,numPoints));
%aligning all combinations of x1 and x2 into a matrix
x=[x1(:) x2(:)];
%creating target sequence
f=sin((x1.^2)/4 + (x2.^2)/2);
t=f(:);

%initializing the weights for the Kohonen network
[w1,w2]=meshgrid(linspace(-1,3,sqrt(M)),linspace(-3,3,sqrt(M)));
W1=[w1(:) w2(:)];

%finding distance between the weights in the kohonen network
d=norm(W1(1,:) - W1(end,:));
G=[];
for i=1:length(x)
    inp=x(i,:);
    %calculating the output "g1" of the Kohonen network for the selected
input
    g1=zeros(M,1);
    for j=1:M
        g1(j,1)=norm(W1(j,:)-inp);
    end
    %calculating the output "g2" of the radial basis function
    g2=exp((-M*(g1.^2))/d);
    %adding the bias term to be input to MIMO LMS
    g2=[g2;1]';
    %storing the output "g2" to be used as input for the MIMO LMS
    G=[G;g2];
end

%calculating the optimum weights for the MIMO LMS using the normal
%equation
W2=(pinv(G)*t)';

%using the network to plot
f_network=[];
f_original=t;

for i=1:length(x)
    inp=x(i,:);
    g1=zeros(M,1);
```

```
    for j=1:M
        g1(j,1)=norm(W1(j,:)-inp);
    end
    g2=exp((-M*(g1.^2))/d);
    g2=[g2;1];
    y=W2*g2;
    f_network=[f_network;y];
end

f_network =reshape(f_network,numPoints,numPoints);
f_original=reshape(f_original,numPoints,numPoints);
figure
subplot(2,1,1);
mesh(linspace(-1,3,numPoints),linspace(-3,3,numPoints),f_original);
title('Original function');
xlabel('x1');
ylabel('x2');
zlabel('f(x1,x2)');

subplot(2,1,2);
mesh(linspace(-1,3,numPoints),linspace(-3,3,numPoints),f_network);
title('Estimated function through neural network');
xlabel('x1');
ylabel('x2');
zlabel('f(x1,x2)');
```

Task 5B

Matlab code for the function “ga_fir” that runs the genetic algorithm

```
function [hn E]=ga_fir(Hd,L,ec)
%Hd=vector containing desired frequency response
%L=total filter length
%ec=1 for least square error calculation
%ec=2 for minimax error calculation
Hd=Hd(:);
Hd_minimax=[Hd(1:122);Hd(135:250);Hd(262:378);Hd(390:634);Hd(646:762);Hd(774:
890);Hd(903:1024)];
%step size
alpha=0.01;
%alpha=0.005;
p_i=0.1;
%no. of children at each step
p=10;
%initiating the frequency sampling points
freq=linspace(0,pi,length(Hd))';
%initializing first parent
h=zeros((L+1)/2,1);
h(1)=0.5*rand;
runs=2000;
E=[];
%begin iteration
while (runs>0)
    children=zeros(length(h),p);
    for i=1:p
        children(:,i)=h +
alpha*diag(randn(length(h),1))*(rand(length(h),1)<p_i);
    end
    %calculating the error of each child
    err_children=zeros(p,1);
    for i=1:p
        child=children(:,i);
        child=[flipud(child(2:end));child];
        freq_res=abs(fft(child,2048));
        freq_res=freq_res(1:1024);

freq_res_minimax=[freq_res(1:122);freq_res(135:250);freq_res(262:378);freq_re
s(390:634);freq_res(646:762);freq_res(774:890);freq_res(903:1024)];
        if (ec==1)
            err_children(i)=(norm(Hd-freq_res))^2;
        else
            err_children(i)=max(abs(Hd_minimax-freq_res_minimax));
        end
    end
    %calculating the error for parent
    h_temp=[flipud(h(2:end));h];
    freq_res=abs(fft(h_temp,2048));
    freq_res=freq_res(1:1024);
```

```
freq_res_minimax=[freq_res(1:122);freq_res(135:250);freq_res(262:378);freq_re
s(390:634);freq_res(646:762);freq_res(774:890);freq_res(903:1024)];
    if (ec==1)
        err_parent=(norm(Hd-freq_res))^2;
    else
        err_parent=max(abs(Hd_minimax-freq_res_minimax));
    end

    if (ec==1)
        check=find(err_children<err_parent-0.0001);
    else
        check=find(err_children<err_parent-0.000001);
    end
    err_parent
    E=[E err_parent];

    if (length(check)>0)
        [err_children_sorted,sort_ind]=sort(err_children);
        [k ind]=max(diag(rand(p,1))*exp((-1*linspace(0,1,p)')/0.1));
        h=children(:,sort_ind(ind));
        runs=2000;
        if (ec==1)
            alpha=0.9999*alpha;
            p_i=0.9999*p_i;
        else
            alpha=0.9999*alpha;
            p_i=0.9999*p_i;
        end
    else
        runs=runs-1
    end
end
hn=[flipud(h(2:end));h];
end
```

Script to call the function “ga_fir” using least square error criterion

```
close all
clear all
clc

R=[zeros(1,128) ones(1,128) linspace(1,0.2,128) 0.2*ones(1,256)
linspace(0.2,1,128) ones(1,128) zeros(1,128)];
f=linspace(0,pi,length(R));
L=33;
[hn E]=ga_fir(R,L,1);
figure;
subplot(3,2,1);
plot(f,R,'b');
hold on;
HN=abs(fft(hn,2048));
plot(f,HN(1:1024),'r');
hold off
subplot(3,2,2)
plot(E)

L=65;
[hn2 E2]=ga_fir(R,L,1);
subplot(3,2,3);
plot(f,R,'b');
hold on;
HN2=abs(fft(hn2,2048));
plot(f,HN2(1:1024),'r');
hold off
subplot(3,2,4)
plot(E2)

L=129;
[hn3 E3]=ga_fir(R,L,1);
subplot(3,2,5);
plot(f,R,'b');
hold on;
HN3=abs(fft(hn3,2048));
plot(f,HN3(1:1024),'r');
hold off
subplot(3,2,6)
plot(E3)
```

Script to call the function “ga_fir” using minimax error criterion

```
close all
clear all
clc

R=[zeros(1,128) ones(1,128) linspace(1,0.2,128) 0.2*ones(1,256)
linspace(0.2,1,128) ones(1,128) zeros(1,128)];
f=linspace(0,pi,length(R));

L=129;
[hn E]=ga_fir(R,L,2);
figure;
subplot(3,2,1);
plot(f,R,'b');
hold on;
HN=abs(fft(hn,2048));
plot(f,HN(1:1024),'r');
hold off
subplot(3,2,2)
plot(E)

L=65;
[hn2 E2]=ga_fir(R,L,2);
subplot(3,2,3);
plot(f,R,'b');
hold on;
HN2=abs(fft(hn2,2048));
plot(f,HN2(1:1024),'r');
hold off
subplot(3,2,4)
plot(E2)

L=33;
[hn3 E3]=ga_fir(R,L,2);
subplot(3,2,5);
plot(f,R,'b');
hold on;
HN3=abs(fft(hn3,2048));
plot(f,HN3(1:1024),'r');
hold off
subplot(3,2,6)
plot(E3)
```

Script to call the function “ga_fir” for the arbitrarily chosen frequency response

```
close all
clear all
clc

R=[linspace(0.2,1,200) 0.5*ones(1,400) linspace(0.5,1,100)
linspace(1,0,324)];
f=linspace(0,pi,length(R));
L=33;
[hn E]=ga_fir(R,L,1);
figure;
subplot(3,2,1);
plot(f,R,'b');
hold on;
HN=abs(fft(hn,2048));
plot(f,HN(1:1024),'r');
hold off
subplot(3,2,2)
plot(E)

L=65;
[hn2 E2]=ga_fir(R,L,1);
subplot(3,2,3);
plot(f,R,'b');
hold on;
HN2=abs(fft(hn2,2048));
plot(f,HN2(1:1024),'r');
hold off
subplot(3,2,4)
plot(E2)

L=129;
[hn3 E3]=ga_fir(R,L,1);
subplot(3,2,5);
plot(f,R,'b');
hold on;
HN3=abs(fft(hn3,2048));
plot(f,HN3(1:1024),'r');
hold off
subplot(3,2,6)
plot(E3)
```