ASSIGNMENT ON EE602

SUBMITTED BY-

Yogesh Kumar Chauhan(234102508)

Isha Rani Das(234102504)

Mtech, SCA, 2nd sem

1. Consider the dynamical system described by equation (1).

$$y(k+1) = \frac{y(k)}{1+y^2(k)} + u^3(k) \tag{1}$$

Identify the system using a radial basis function network with Hybrid learning scheme. Show the initial and final distributions of RBFN centers. Generate 1000 data pairs, by choosing input randomly between ± 1, for training. Use a different data set for testing.

Solution:

The Matlab code is attached below:

```
clc;
clear;
close all;
%% Data Generation
Nt = 1000;  % Training data count
rng('shuffle');
U_train = 4 * rand(Nt - 1, 1) - 2;  % Input data
between ±1
Yd_train = [rand; zeros(Nt, 1)];
for i = 1:Nt - 1
    Yd_train(i + 1) = Yd_train(i) / (1 +
Yd_train(i)^2) + U_train(i)^3;  % Generate
output data based on the given dynamical system
end
inputToNet = [U_train Yd_train(1:Nt-1)];
%% Radial Basis Function Network (RBFN) Training
num_centers = 100;  % Number of RBFN centers
centers = datasample(inputToNet, num_centers, 1,
'Replace', false);  % Randomly select centers
from the training data
initial_centers = centers;
initial_num_centers = num_centers;
% Initialize widths for each RBFN center
width = 0.2 * ones(num_centers, 1);  % Initial
width value for all centers
% Initialize weights for the output layer
weights = rand(num_centers, 1);
% Plot initial and final center positions
for i=1:num_centers
    plot(initial_centers(:,1),
initial_centers(:,2), 'go', 'MarkerSize',
40*width(i));  % Initial center positions
    hold on;
end
% Hyperparameters for training
eta = 0.1;  % Learning rate
```

```
max_epoch = 3000;  % Maximum number of training epochs
mse = 0;
MSE_t=zeros(max_epoch, 1);
epoch_count = 0;
% Train the RBFN by adjusting the weights and updating the centers for each input
data point
Y_pred_train = zeros(Nt - 1, 1);  % Predictions for training data
% for count = 1:max_epoch
%     for i = 1:Nt - 1
%         for j = 1:num_centers
%             % Update centers based on the unsupervised approach
%             alpha = 0.1;  % Learning rate for center update
%             closest_index = find_closest_center(inputToNet(i,:), centers);
%             centers(closest_index,:) = centers(closest_index,:) + alpha *
(inputToNet(i,:) - centers(closest_index,:));
%         end
%     end
%     count
% end
%% K-means clustering
[idx, centers] = kmeans(inputToNet, num_centers);  % Apply k-means clustering
% Calculate max distance between centers
%max_distance = max(pdist(centers));
% % Update the width using the formula: width = max_distance /
(sqrt(2*num_centers))
% width = (max_distance / sqrt(2*num_centers)) * ones(num_centers, 1);
% Initialize velocities for weights and widths
velocity_weights = zeros(num_centers, 1);
velocity_widths = zeros(num_centers, 1);
% Hyperparameters for velocity update
alpha_v = 0.2; % Momentum parameter
for epoch = 1:max_epoch
    del_widths=zeros(Nt-1, num_centers);
    for i = 1:Nt - 1
        % Forward pass: calculate activations for the current input data point
        activations = zeros(num_centers, 1);
        for j = 1:num_centers
            % Calculate activation for each RBFN center using Gaussian activation
function
            activations(j) = exp(-((U_train(i) - centers(j,1))^2 + (Yd_train(i) -
centers(j,2))^2) / (2 * width(j)^2));
        end
% Compute the output of the RBFN as a weighted sum of the activations
        output = activations' * weights;

        % Compute error
        error = Yd_train(i + 1) - output;

        % Update width using gradient descent
        for j=1:num_centers
            del_widths(i, j) = eta * error*weights(j)'*activations(j)*((U_train(i)
- centers(j,1))^2 + (Yd_train(i) - centers(j,2))^2)/(width(j)^3);
        end

        % Update weights using gradient descent with velocity
        velocity_weights = alpha_v * velocity_weights + eta * error * activations;
        weights = weights + velocity_weights;
        % Store predictions for training data
        Y_pred_train(i) = output;
```

```
end
    % Update width using gradient descent with velocity
    for j=1:num_centers
        velocity_widths(j) = alpha_v * velocity_widths(j) + 0.5 *
mean(del_widths(:,j));
        width(j) = width(j) + velocity_widths(j);
    end
    mse = (norm(Y_pred_train-output))/Nt;
    epoch_count = epoch_count + 1;
    MSE_t(epoch)=mse;
end
%% Testing
Nt_test = 100;  % Number of testing data points
U_test = zeros(100, 1);
for i = 1:100
    U_test(i) = sin(0.1 * i);    % Input for testing data
end
% Compute predictions for testing data
Y_pred_test = zeros(Nt_test, 1);
for i = 1:Nt_test-1
    % Compute activations for each RBFN center using Gaussian activation function
    activations = exp(-((U_test(i) - centers(:,1)).^2 + (Y_pred_test(i) -
centers(:,2)).^2) ./ (2 * width.^2));
    % Compute output of RBFN as a weighted sum of the activations
    Y_pred_test(i+1) = activations' * weights;
end
% Actual results for testing data based on the given dynamical system
Yd_test = zeros(Nt_test, 1);
for i = 1:Nt_test - 1
    Yd_test(i + 1) = Yd_test(i) / (1 + Yd_test(i)^2) + U_test(i)^3;
end
mse_test=norm(Y_pred_test-Yd_test)/Nt_test;
fprintf("Training error= %f, test error= %f", mse, mse_test);
% Plot final center positions
for i=1:num_centers
    plot(centers(i,1), centers(i,2), 'ro', 'MarkerSize', width(i)*40);  % Final
center positions
end
plot(U_train, Yd_train(1:999), 'b.', 'MarkerSize', 8);  % Final center positions
xlabel('Input (U)');
ylabel('Output (Y)');
title('Initial and Final Center Positions');
legend('Initial Centers', 'Final Centers');
grid on;
fprintf("Center count=%d", length(centers));
hold off;
%% Plotting
% Training data
figure;
plot(1:Nt-1, Yd_train(2:Nt), 'b-', 'MarkerSize', 10);  % Actual results for
training data
hold on;
plot(1:Nt-1, Y_pred_train, 'r-', 'LineWidth', 2);  % Predicted results for
training data
xlabel('Iteration (k)');
ylabel('Output (Y)');
title('Training Data');
legend('Actual Result', 'Predicted Result');
grid on;
```

```
hold off;
% Testing data
figure;
plot(1:Nt_test, Yd_test, 'b-', 'MarkerSize', 10);  % Actual results for testing data
hold on;
plot(1:Nt_test, Y_pred_test, 'r-', 'LineWidth', 2);  % Predicted results for testing data
xlabel('Iteration (k)');
ylabel('Output (Y)');
title('Testing Data');
legend('Actual Result', 'Predicted Result');
grid on;
hold off;
figure
plot(1:max_epoch, MSE_t);
function closest_index = find_closest_center(x, centers)
    distances = sqrt(sum((x - centers).^2, 2));
    [~, closest_index] = min(distances);
end
```

```
Training error= 0.006616, test error= 0.005322Center count=100
```

The response plots are shown below:

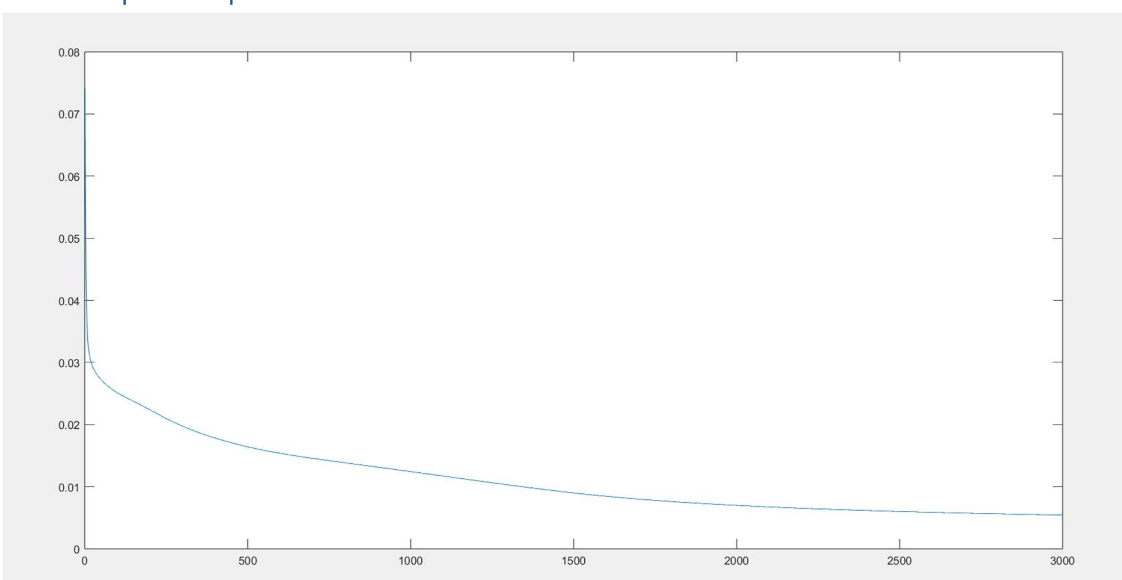

Fig: showing error is reducing with no of epoch increasing

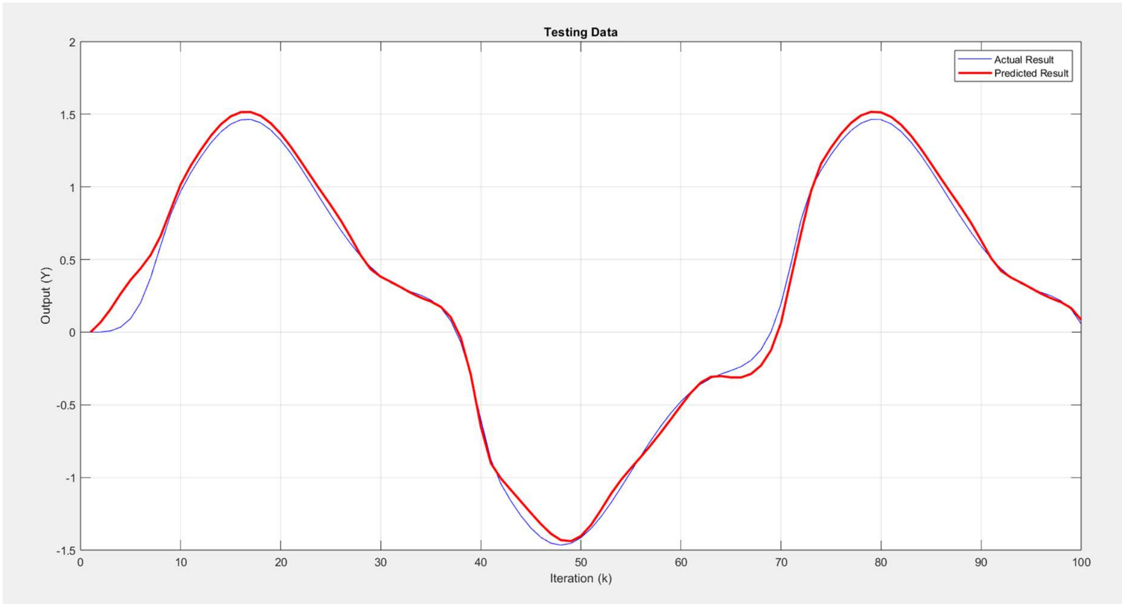

Fig: showing convergence of actual result to the predicted result

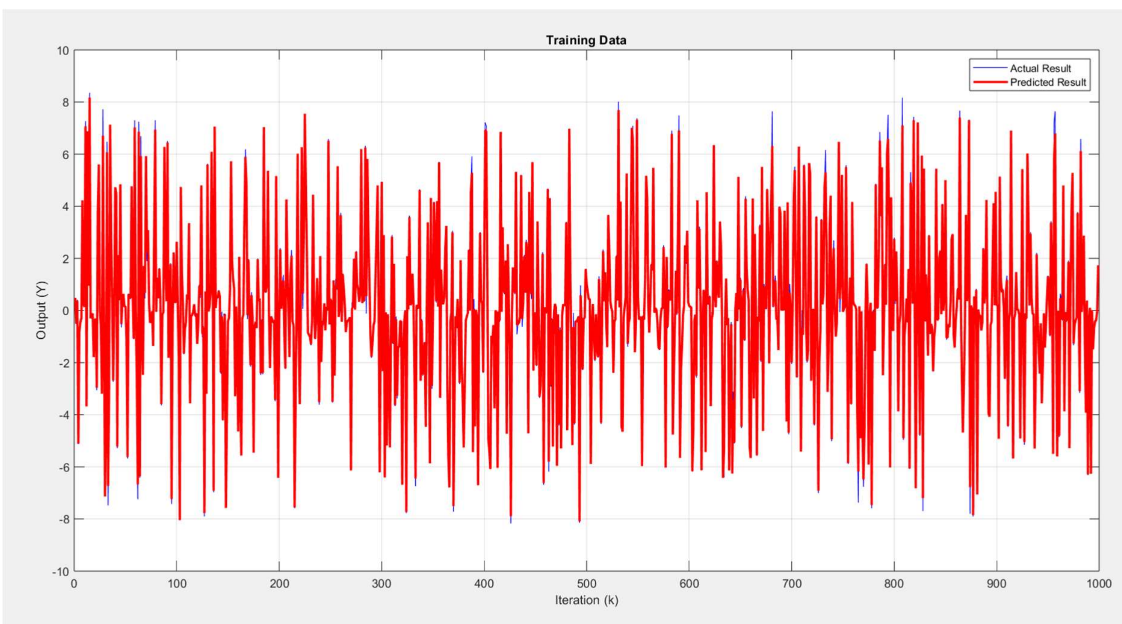

Fig: showing convergence of actual result to the predicted result

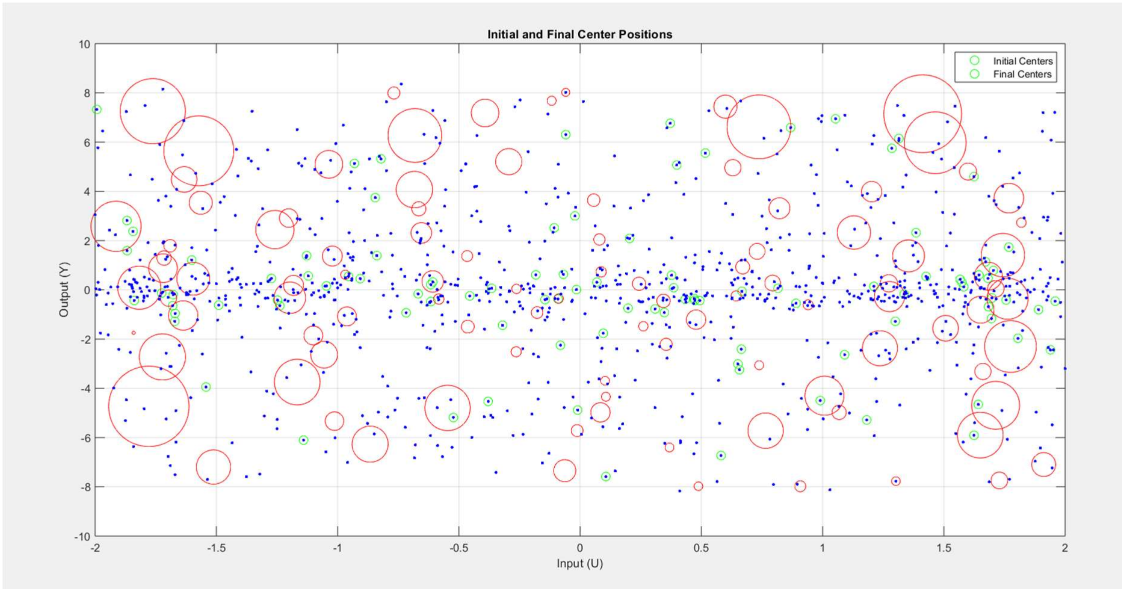

Fig: initial and final center positions

In this problem we have used gradient descent and K-means clustering for data training. The RBFN is trained using a gradient descent approach to adjust weights and update centers based on the input output data. K-means clustering is utilized to determine centers for the RBFN.

2. Consider the following dynamics of a nonlinear SISO system

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= f(\boldsymbol{x}) + g(\boldsymbol{x})u \\ y &= x_1 \end{aligned} \tag{2}$$

where $\boldsymbol{x} = [x_1 \; x_2]^T$ and

$$\begin{aligned} f(\boldsymbol{x}) &= 4\left(\frac{\sin(4\pi x_1)}{\pi x_1}\right)\left(\frac{\sin(\pi x_2)}{\pi x_2}\right)^2 \\ g(\boldsymbol{x}) &= 2 + \sin(3\pi(x_1 - 0.5)) \end{aligned}$$

Design an adaptive controller using neural network

(a) when $f(\boldsymbol{x})$ is unknown but $g(\boldsymbol{x})$ is known

(b) when $f(\boldsymbol{x})$ and $g(\boldsymbol{x})$ both are unknown

to track a sinusoidal trajectory of unit amplitude and 1 Hz frequency.

Solution:

The matlab code is attached below:

Part1:

```
clc;

clear;

close all;

nor_fact=10;

%% Data Generation

Nt = 1000;  % Training data count

rng('shuffle');

U_train = [3 * rand(Nt - 1, 1) - 1.5, 3 * rand(Nt - 1, 1) - 1.5];  % Input data between ±1

Yd_train = [rand; zeros(Nt-1, 1)];

for i = 1:Nt - 1

    Yd_train(i) = F_fun([U_train(i, 1), U_train(i, 2)])/nor_fact;  % Generate output data based on the given dynamical system

end

inputToNet = U_train;  % Use all training data as input

%% Radial Basis Function Network (RBFN) Training

num_centers = 50;  % Number of RBFN centers

centers = datasample(inputToNet, num_centers, 1, 'Replace', false);  % Randomly select centers from the training data

initial_centers = centers;

initial_num_centers = num_centers;

% Initialize widths for each RBFN center

width = 0.2 * ones(num_centers, 1);  % Initial width value for all centers

% Initialize weights for the output layer

weights = rand(num_centers, 1);

% Plot initial and final center positions
```

```
for i=1:num_centers
    plot(initial_centers(:,1), initial_centers(:,2), 'go', 'MarkerSize', 40*width(i));  % Initial center positions
    hold on;
end
% Hyperparameters for training
eta = 0.1;  % Learning rate
alpha_v = 0; % Momentum parameter
max_epoch = 1000;  % Maximum number of training epochs
mse = 1000;
MSE_t=zeros(max_epoch, 1);
epoch_count = 0;
% Train the RBFN by adjusting the weights and updating the centers for each input data point
Y_pred_train = zeros(Nt - 1, 1);  % Predictions for training data
% Initialize velocities for weights and widths
velocity_weights = zeros(num_centers, 1);
velocity_widths = zeros(num_centers, 1);
%% K-means clustering
opts = statset('Display','final');
% [idx,C] = kmeans(inputToNet,2,'Distance','cityblock',...
%     'Replicates',5,'Options',opts);
[idx, centers] = kmeans(inputToNet, num_centers);  % Apply k-means clustering
%% training
for epoch = 1:max_epoch
    del_widths=zeros(Nt-1, num_centers);
    for i = 1:Nt - 1
```

```
        % Forward pass: calculate activations for the current input data point

        activations = zeros(num_centers, 1);

        for j = 1:num_centers

            % Calculate activation for each RBFN center using Gaussian activation function

            activations(j) = exp(-((inputToNet(i, 1) - centers(j,1))^2 + (inputToNet(i, 2) - centers(j,2))^2) / (2 * width(j)^2));

        end



        % Compute the output of the RBFN as a weighted sum of the activations

        output = activations' * weights;



        % Compute error

        error = Yd_train(i) - output;



        %Update width using gradient descent

        for j=1:num_centers

            del_widths(i, j) = eta * error*weights(j)'*activations(j)*((inputToNet(i, 1) - centers(j,1))^2 + (inputToNet(i, 2) - centers(j,2))^2)/(width(j)^3);

        end



        % Update weights using gradient descent with velocity

        velocity_weights = alpha_v * velocity_weights + eta * error * activations;

        weights = weights + velocity_weights;

        % Store predictions for training data

        Y_pred_train(i) = output;

    end
```

```
    %Update width using gradient descent with velocity
    for j=1:num_centers
        velocity_widths(j) = alpha_v * velocity_widths(j) + 0.5 * mean(del_widths(:,j));
        width(j) = width(j) + velocity_widths(j);
    end
    mse = (norm(Yd_train(1:999)-Y_pred_train))/Nt;
    epoch_count = epoch_count + 1
    MSE_t(epoch)=mse
    if isnan(sum(width))
        break
    end
end
%% Testing
Nt_test = 100;  % Number of testing data points
inputToNet_test = [zeros(100, 1), zeros(100, 1)];
for i = 1:100
    inputToNet_test(i,:) = [sin(0.1 * i), cos(0.1*i)];    % Input for testing data
end
% Compute predictions for testing data
Y_pred_test = zeros(Nt_test, 1);
for i = 1:Nt_test-1
    % Compute activations for each RBFN center using Gaussian activation function
    for j = 1:num_centers
        activations(j) = exp(-((inputToNet_test(i, 1) - centers(j,1))^2 + (inputToNet_test(i, 2) -
centers(j,2))^2) / (2 * width(j)^2));
    end
```

```
    % Compute output of RBFN as a weighted sum of the activations
    Y_pred_test(i) = activations' * weights;
end
% Actual results for testing data based on the given dynamical system
Yd_test = zeros(Nt_test, 1);
for i = 1:Nt_test - 1
    Yd_test(i) = F_fun(inputToNet_test(i, :));
end
mse_test=norm(Y_pred_test-Yd_test)/Nt_test;
fprintf("Training error= %f, test error= %f", mse, mse_test);
% Plot final center positions
for i=1:num_centers
    plot(centers(i,1), centers(i,2), 'ro', 'MarkerSize', width(i)*40);  % Final center positions
end
plot(U_train, Yd_train(1:999), 'b.', 'MarkerSize', 8);  % Final center positions
xlabel('Input (U)');
ylabel('Output (Y)');
title('Initial and Final Center Positions');
legend('Initial Centers', 'Final Centers');
grid on;
fprintf("Center count=%d", length(centers));
hold off;
%% Plotting
% Training data
figure;
```

```
plot(1:Nt-1, Yd_train(1:Nt-1), 'b-', 'MarkerSize', 10);  % Actual results for training data
hold on;
plot(1:Nt-1, Y_pred_train, 'r-', 'LineWidth', 2);  % Predicted results for training data
xlabel('Iteration (k)');
ylabel('Output (Y)');
title('Training Data');
legend('Actual Result', 'Predicted Result');
grid on;
hold off;
% Testing data
figure;
plot(1:Nt_test, Yd_test, 'b-', 'MarkerSize', 10);  % Actual results for testing data
hold on;
plot(1:Nt_test, Y_pred_test*nor_fact, 'r-', 'LineWidth', 2);  % Predicted results for testing data
xlabel('Iteration (k)');
ylabel('Output (Y)');
title('Testing Data');
legend('Actual Result', 'Predicted Result');
grid on;
hold off;
figure
plot(1:max_epoch, MSE_t);
%% Function definition for f(x)
%F_fun([1 1])
function y = F_fun(z)
```

```
    % Function definition: f(x) = 4 * (sin(4*pi*x(1)) / (pi*x(1))) * (sin(pi*x(2)) / (pi*x(2)))^2
    T=0.01;
    x=[z(1); (z(2)-z(1))/T];
    if x(1)~=0
        term1 = sin(4*pi*x(1)) / (pi*x(1));
    else
        term1=4;
    end
    if x(2)~=0
        term2 = sin(pi*x(2)) / (pi*x(2));
    else
        term2=1;
    end
    y = T*4 * term1 * term2^2+x(2);   %Last term due to discretization
end
```

Training error= 0.007126, test error= 8.842917Center count=50

The responses:

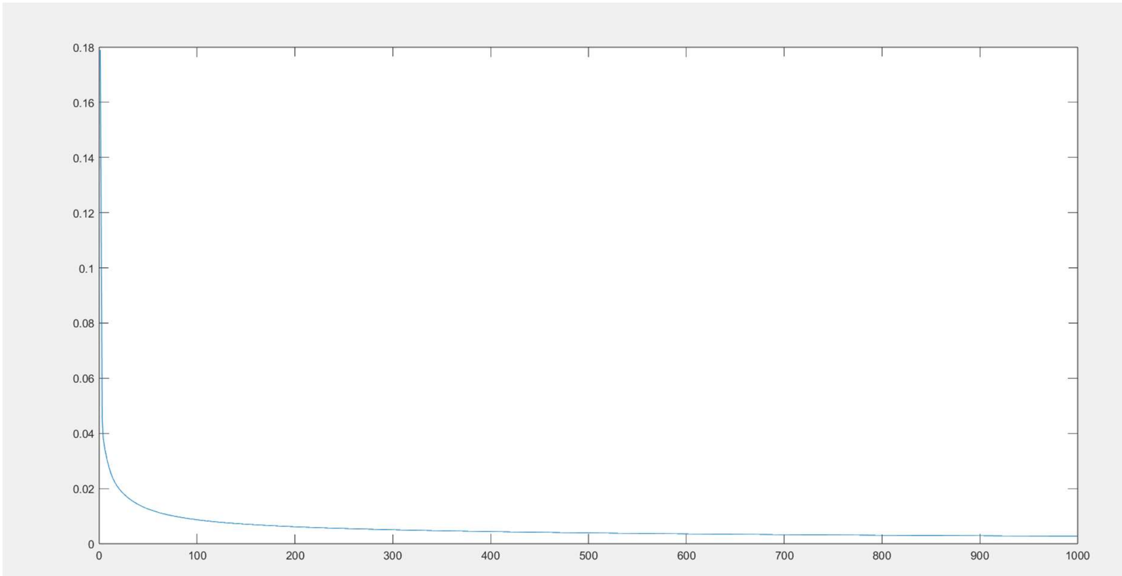

Fig: no of epoch vs error graph

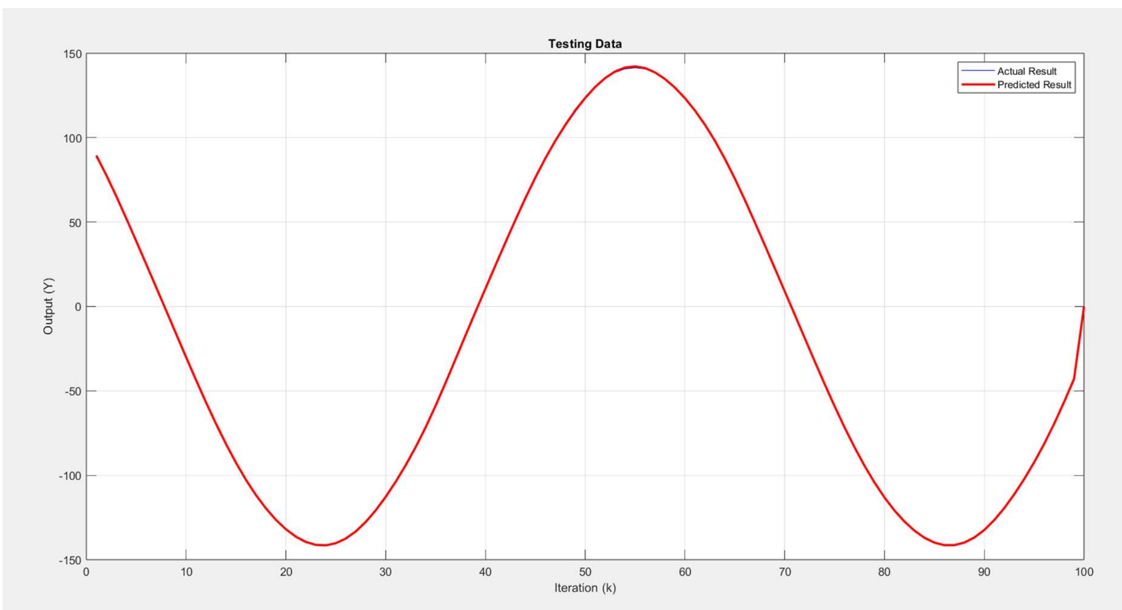

Fig: showing the convergence of actual result to the predicted result

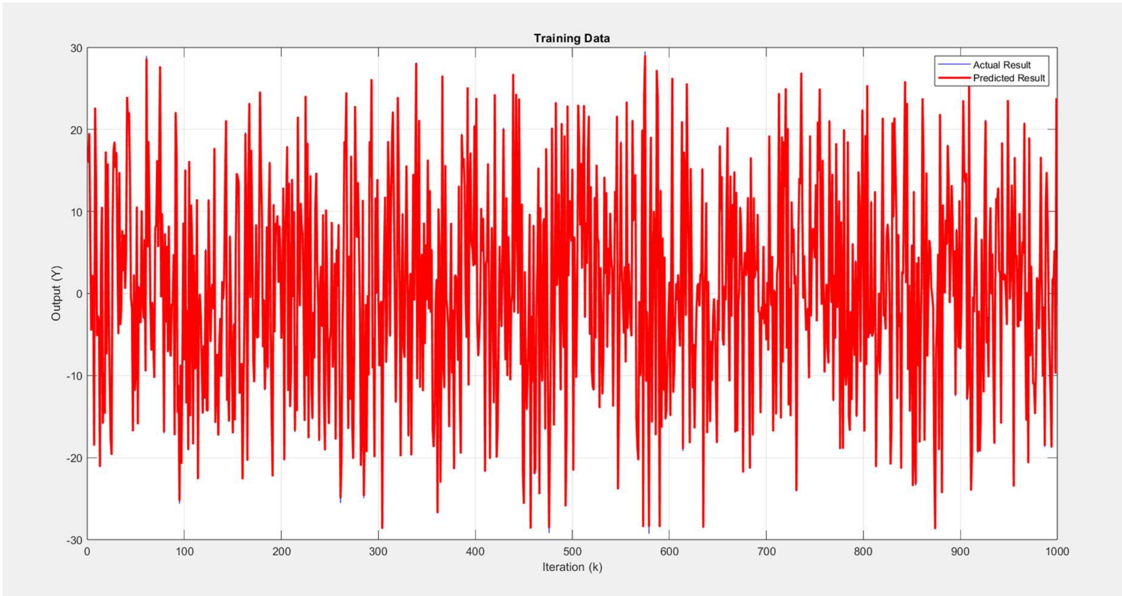

Fig: iteration vs output graph for actual result and predicted result

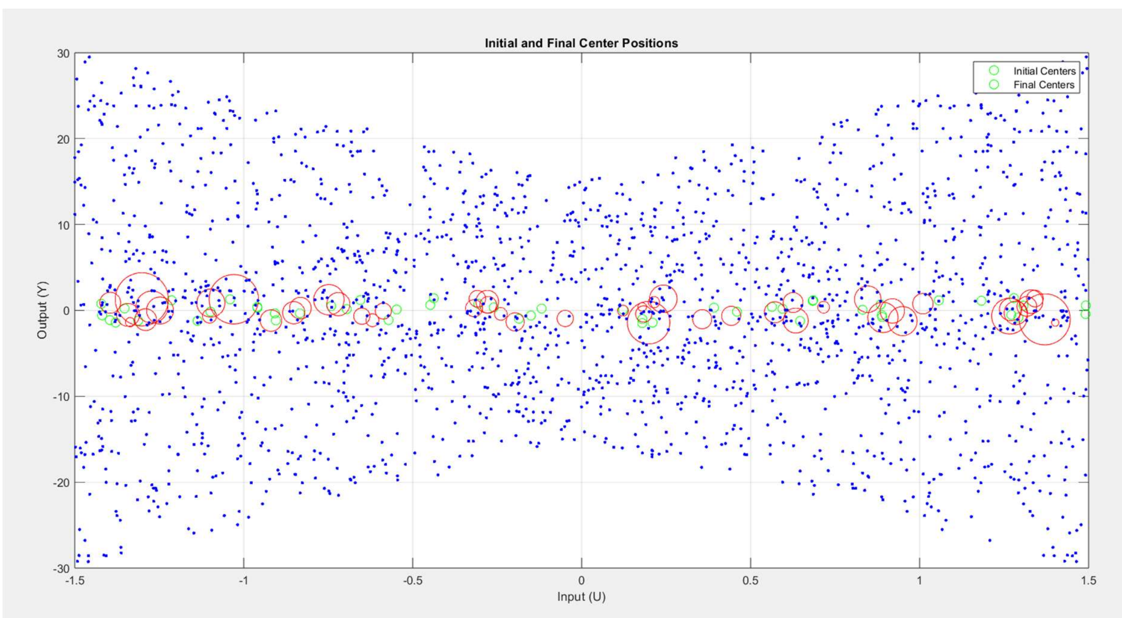

Fig: initial and final center positions

We have tried to design the adapter controller but not getting useful result , here we are attaching the code although the neural network is working quiet well.

```
clc;

clear;

close all;
```

```
nor_fact=10;
T=0.01;
max_k = 1000;  % Maximum time horizon/T
Yd=0;
x1d=sin(T*(1:max_k));       %Output is x1
x2d=zeros(max_k, 1);
for k=2:max_k
    x2d(k)=my_F_fun([x1d(k-1) x2d(k-1)]);
end
x1=0;
x2=0;
X=zeros(max_k, 2);
z1=0;
z2=0;
kv=1;
Nt=100      %width update after each Nt
%% Data Generation
rng('shuffle');
inputToNet = 0;  % Use all training data as input
error=0;
%% Radial Basis Function Network (RBFN) Training
num_centers = 50;  % Number of RBFN centers
Yd_train = [rand; zeros(Nt-1, 1)];
centers = [0.363967494316839, -0.557835021948100;
           0.586519722269397,  0.965702097230501;
```

-1.27030986839960, -0.935959112483098;

-0.439854961171384, 0.568446957505851;

-1.39452376290622, 0.0185461875063542;

0.135760675540234, -0.798966808734080;

-1.14178919055393, 0.444739146808625;

-0.948561709518276, -0.602960726688085;

0.819363698309447, -0.769327330704365;

0.243467003713836, 0.0505718250110133;

0.00377236843119735,1.40097889840455;

0.971466830529498, 1.30074029221752;

1.36929681422331, -0.196450914326569;

0.729530252225835, 0.121435363447677;

-0.222160494309628, -0.645705377966331;

1.40138040412387, 0.364383172480505;

-1.02065420559128, -0.0988553902133012;

1.40525159038584, 1.29862519535170;

0.431768502816382, -1.30477469675911;

-1.32145285016814, -0.519706995067396;

-1.22566394926170, 1.29250978575360;

-0.372813791648497, 1.36901872322582;

0.952337549428624, -1.30553699216156;

0.000117541570946073,1.02808277758760;

-0.919668991055883, 0.722360820804287;

-0.684306688084754, -1.32701838556854;

-0.229913613778815, -1.29670111719450;

```
 0.865455947933068,  0.759369969885894;
 1.31287380423666,  -1.32411608565822;
-1.23065321106405,  -1.35840007268155;
-0.785597157975713,  0.321203755627475;
-0.0122411335548145,0.395363475298654;
 0.532489288027987,  0.525183547811018;
 0.364237856869538,  1.33392797100870;
-0.303672857972633,  0.0946856745636213;
 1.21213693983356,   0.716348087780019;
-0.594955622128991, -0.275879471566228;
 1.33921280151247,  -0.608795074481834;
-0.622389922338866, -0.878864414794022;
 0.0758774281673357,-0.299072969504407;
-0.874005643750108,  1.25118564545961;
-0.503839981329220,  0.947880863717196;
 0.208758848722001,  0.656034023620586;
 0.0568382030530952,-1.24528347071097;
-1.35194496484219,   0.866481913786384;
 0.954175601533702, -0.315734297935237;
 1.35616855428414,  -0.944098560639925;
 0.497801477191861, -0.912111213459283;
 1.07015106429649,   0.179469665574748;
 0.622302650945660, -0.269328837311158];
 % Randomly select centers from the training data % Randomly select centers
% initial_centers = centers;
```

```
% initial_num_centers = num_centers;

% Initialize widths for each RBFN center

width = 0.2 * ones(num_centers, 1);  % Initial width value for all centers

% Initialize weights for the output layer

weights = rand(num_centers, 1);

% % Plot initial and final center positions

% for i=1:num_centers

%     plot(initial_centers(i,1), initial_centers(i,2), 'go', 'MarkerSize', 40*width(i));  % Initial center positions

%     hold on;

% end

% Hyperparameters for training

eta = 0.1;  % Learning rate

alpha_v = 0; % Momentum parameter

mse = 1000;

MSE_t=zeros(max_k, 1);

% Train the RBFN by adjusting the weights and updating the centers for each input data point

Nt=100;     %For width update after Nt samples

Y_pred_train = zeros(Nt, 1);  % Predictions for training data

% Initialize velocities for weights and widths

velocity_weights = zeros(num_centers, 1);

velocity_widths = zeros(num_centers, 1);

del_widths=zeros(Nt, num_centers);

z=[0;0];

lambda=1;

Y=0;
```

```
%% training
for k = 2:max_k

    %% Neural Network
    Fd=F_fun(z)
    inputToNet=[z(1), z(2)];
    % Forward pass: calculate activations for the current input data point
    activations = zeros(num_centers, 1);
    for j = 1:num_centers
        % Calculate activation for each RBFN center using Gaussian activation function
        activations(j) = exp(-(norm(inputToNet - centers(j, :))^2) / (2 * width(j)^2));
    end

    % Compute the output of the RBFN as a weighted sum of the activations
    output = activations' * weights

    % Compute error
    error = Fd - output;

    %Update width using gradient descent every Nt
    if mod(k, Nt)~=0
        for j=1:num_centers
            del_widths(mod(k, Nt), j) = eta * error*weights(j)'*activations(j)*(norm(inputToNet - centers(j,:))^2)/(width(j)^3);
        end
        % Store predictions for training data
```

```
    Y_pred_train(mod(k, Nt)) = output;
end


% Update weights using gradient descent with velocity
velocity_weights = alpha_v * velocity_weights + eta * error * activations;
weights = weights + velocity_weights;
%Update width using gradient descent with velocity every Nt
if mod(k, Nt)==0
    for j=1:num_centers
        velocity_widths(j) = alpha_v * velocity_widths(j) + 0.5 * mean(del_widths(:,j));
        width(j) = width(j) + velocity_widths(j);
    end
end
% %% K-means clustering
% [idx, centers] = kmeans(inputToNet, num_centers);  % Apply k-means clustering
mse = (norm(error))/Nt;
MSE_t(k)=mse;
%% Plant dynamics
e1=x1-x1d(k);         %x1 holds previous value
e2=x2-x2d(k);       %x2 holds previous value
r=lambda*e1+e2;
if k~=max_k
    u=(1/G_fun(z(1)))*(-output+kv*r+x2d(k+1)+lambda*e2);
    z1=z2                       %z(k+1)=z(k)
    z2=F_fun(z)+G_fun(z(1))*u    %kth input gives k+1th output
```

```
        x2=(z(2)-z(1))/T
        x1=z(1)
    end
    X(k, 1)=x1;
    X(k, 2)=x2;
    t = k*T + 1
    k=k+1
    if isnan(sum(width))
        break
    end
end
figure
% Plot final center positions
hold on
for i=1:num_centers
    plot(centers(i,1), centers(i,2), 'ro', 'MarkerSize', width(i)*40);  % Final center positions
end
xlabel('Input (U1)');
ylabel('Input (U2)');
title('Initial and Final Center Positions');
legend('Initial Centers', 'Final Centers');
grid on;
fprintf("Center count=%d", length(centers));
hold off;
figure
```

```
%% Plotting

% Plot x2d vs x2

plot(1:max_k, x1d,'r.');

hold on

plot(1:max_k, x2d,'y.');

% Plot x1d vs x1

plot(1:max_k, X(:,1),'g.');

plot(1:max_k, X(:,2),'b.');

xlabel('k');

ylabel('States');

title('States vs k');

legend('x1d', 'x2d', 'x1', 'x2');

%% Plotting

figure

plot(1:max_k, MSE_t);

%% Function definition for f(x)

%F_fun([1 1])

function y = F_fun(z)

    % Function definition: f(x) = 4 * (sin(4*pi*x(1)) / (pi*x(1))) * (sin(pi*x(2)) / (pi*x(2)))^2

    T=0.01;

    x=[z(1); (z(2)-z(1))/T];

    if x(1)~=0

        term1 = sin(4*pi*x(1)) / (pi*x(1));

    else

        term1=4;
```

```
    end
    if x(2)~=0
        term2 = sin(pi*x(2)) / (pi*x(2));
    else
        term2=1;
    end
    y = T*4 * term1 * term2^2+x(2);   %Last term due to discretization
end
function out = G_fun(z)
    % Function definition: f(x) = 4 * (sin(4*pi*x(1)) / (pi*x(1))) * (sin(pi*x(2)) / (pi*x(2)))^2
    T=0.01;
    %x=[z(1); (z(2)-z(1))/T];
    out = T*(2+sin(3*pi*(z-0.5)));   %Last term due to discretization
end
function y=my_F_fun(x)
    T = 0.01;
    if x(1) ~= 0
        term1 = sin(4 * pi * x(1)) / (pi * x(1));
    else
        term1 = 4;
    end
    if x(2) ~= 0
        term2 = sin(pi * x(2)) / (pi * x(2));
    else
        term2 = 1;
```

```
    end

    y = T * 4 * term1 * term2^2 + x(2);   % Last term due to discretization

end
```

Part2:

```
clc;
clear;
close all;
nor_fact=1/50;
%% Data Generation
Nt = 10000;  % Training data count
rng('shuffle');
U_train = 4 * rand(Nt - 1, 1) - 2;  % Input data between ±1
Yd_train = [rand; zeros(Nt-1, 1)];
for i = 1:Nt - 1
    Yd_train(i) = G_fun(U_train(i, 1))/nor_fact;  % Generate output data based on
the given dynamical system
end
inputToNet = U_train;  % Use all training data as input
%% Radial Basis Function Network (RBFN) Training
num_centers = 25;  % Number of RBFN centers
centers = datasample(inputToNet, num_centers, 1, 'Replace', false);  % Randomly
select centers from the training data
initial_centers = centers;
initial_num_centers = num_centers;
% Initialize widths for each RBFN center
width = 0.2 * ones(num_centers, 1);  % Initial width value for all centers
% Initialize weights for the output layer
weights = rand(num_centers, 1);
% Plot initial and final center positions
for i=1:num_centers
    length(initial_centers(:,1))
    plot(initial_centers(i,1), 0, 'go', 'MarkerSize', 40*width(i));  % Initial
center positions
    hold on;
end
% Hyperparameters for training
eta = 0.1;  % Learning rate
alpha_v = 0; % Momentum parameter
max_epoch = 1000;  % Maximum number of training epochs
mse = 1000;
MSE_t=zeros(max_epoch, 1);
epoch_count = 0;
% Train the RBFN by adjusting the weights and updating the centers for each input
data point
Y_pred_train = zeros(Nt - 1, 1);  % Predictions for training data
% Initialize velocities for weights and widths
velocity_weights = zeros(num_centers, 1);
velocity_widths = zeros(num_centers, 1);
```

```
%% K-means clustering
opts = statset('Display','final');
% [idx,C] = kmeans(inputToNet,2,'Distance','cityblock',...
%     'Replicates',5,'Options',opts);
[idx, centers] = kmeans(inputToNet, num_centers);  % Apply k-means clustering
%% training
for epoch = 1:max_epoch
    del_widths=zeros(Nt-1, num_centers);
    for i = 1:Nt - 1
        % Forward pass: calculate activations for the current input data point
        activations = zeros(num_centers, 1);
        for j = 1:num_centers
            % Calculate activation for each RBFN center using Gaussian activation
function
            activations(j) = exp(-(norm(inputToNet(i, :) - centers(j,:))^2 / (2 *
width(j)^2)));
        end

        % Compute the output of the RBFN as a weighted sum of the activations
        output = activations' * weights;

        % Compute error
        error = Yd_train(i) - output;

        %Update width using gradient descent
        for j=1:num_centers
            del_widths(i, j) = eta *
error*weights(j)'*activations(j)*(norm(inputToNet(i, :) -
centers(j,:))^2)/(width(j)^3);
        end

        % Update weights using gradient descent with velocity
        velocity_weights = alpha_v * velocity_weights + eta * error * activations;
        weights = weights + velocity_weights;
        % Store predictions for training data
        Y_pred_train(i) = output;
    end
    %Update width using gradient descent with velocity
    for j=1:num_centers
        velocity_widths(j) = alpha_v * velocity_widths(j) + 0.5 *
mean(del_widths(:,j));
        width(j) = width(j) + velocity_widths(j);
    end
    mse = (norm(Yd_train(1:Nt-1)-Y_pred_train))/Nt
    epoch_count = epoch_count + 1
    MSE_t(epoch)=mse;
    if isnan(sum(width))
        break
    end
end
%% Testing
Nt_test = 100;  % Number of testing data points
inputToNet_test = [zeros(100, 1)];
for i = 1:100
    inputToNet_test(i,:) = [sin(0.1 * i)];    % Input for testing data
end
% Compute predictions for testing data
Y_pred_test = zeros(Nt_test, 1);
for i = 1:Nt_test-1
```

```
    % Compute activations for each RBFN center using Gaussian activation function
    for j = 1:num_centers
        activations(j) = exp(-(norm(inputToNet_test(i, :) - centers(j,:)) / (2 *
width(j)^2)));
    end
    % Compute output of RBFN as a weighted sum of the activations
    Y_pred_test(i) = activations' * weights;
end
% Actual results for testing data based on the given dynamical system
Yd_test = zeros(Nt_test, 1);
for i = 1:Nt_test - 1
    Yd_test(i) = G_fun(inputToNet_test(i, :));
end
mse_test=norm(Y_pred_test*nor_fact-Yd_test)/Nt_test;
fprintf("Training error= %f, test error= %f", mse, mse_test);
% Plot final center positions
for i=1:num_centers
    plot(centers(i,1), 0, 'ro', 'MarkerSize', width(i)*40);  % Final center
positions
end
plot(U_train(:, 1), zeros(length(U_train(:, 1)), 1), 'b.', 'MarkerSize', 8);  %
Final center positions
xlabel('Input (U1)');
ylabel('Input (U2)');
title('Initial and Final Center Positions');
legend('Initial Centers', 'Final Centers');
grid on;
fprintf("Center count=%d", length(centers));
hold off;
%% Plotting
% Training data
figure;
plot(1:Nt-1, Yd_train(1:Nt-1), 'b-', 'MarkerSize', 10);  % Actual results for
training data
hold on;
plot(1:Nt-1, Y_pred_train, 'r-', 'LineWidth', 2);  % Predicted results for
training data
xlabel('Iteration (k)');
ylabel('Output (Y)');
title('Training Data');
legend('Actual Result', 'Predicted Result');
grid on;
hold off;
% Testing data
figure;
plot(1:Nt_test, Yd_test, 'b-', 'MarkerSize', 10);  % Actual results for testing
data
hold on;
plot(1:Nt_test, Y_pred_test*nor_fact, 'r-', 'LineWidth', 2);  % Predicted results
for testing data
xlabel('Iteration (k)');
ylabel('Output (Y)');
title('Testing Data');
legend('Actual Result', 'Predicted Result');
grid on;
hold off;
figure
plot(1:max_epoch, MSE_t);
%% Function definition for f(x)
```

```
%G_fun([1 1])
function out = G_fun(z)
    % Function definition: f(x) = 4 * (sin(4*pi*x(1)) / (pi*x(1))) * (sin(pi*x(2))
/ (pi*x(2)))^2
    T=0.01;
    %x=[z(1); (z(2)-z(1))/T];
    out = T*(2+sin(3*pi*(z-0.5)));   %Last term due to discretization
end
```

Training error= 0.000010, test error= 0.001437Center count=25

The responses are shown below:

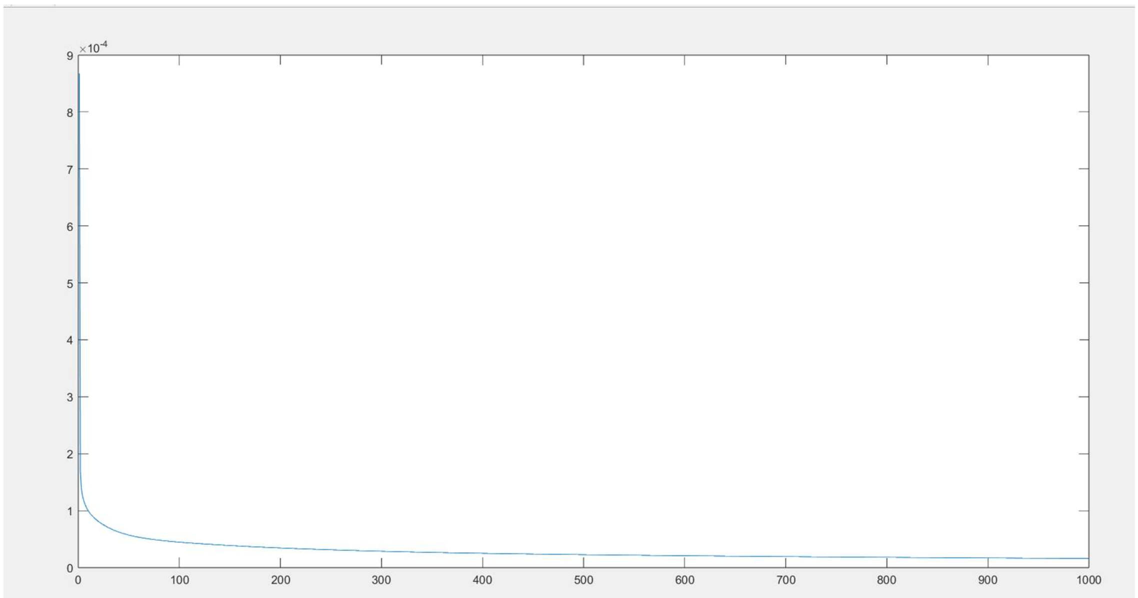

Fig: showing error is reducing with number of epoch increasing

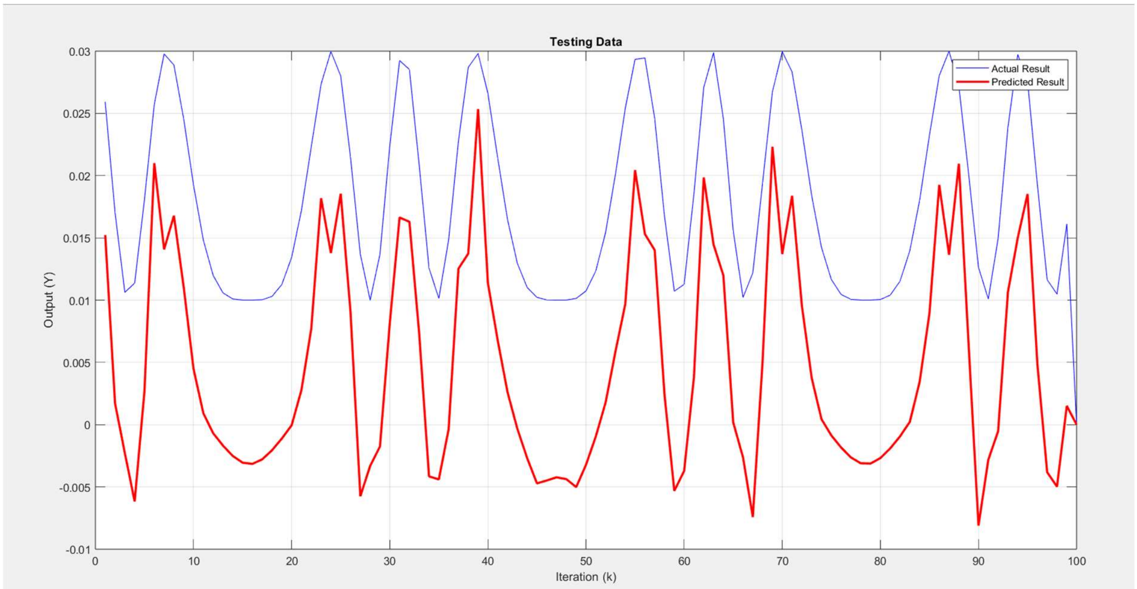

Fig: iteration vs output graph trying to show the convergence of actual result to the predicted result

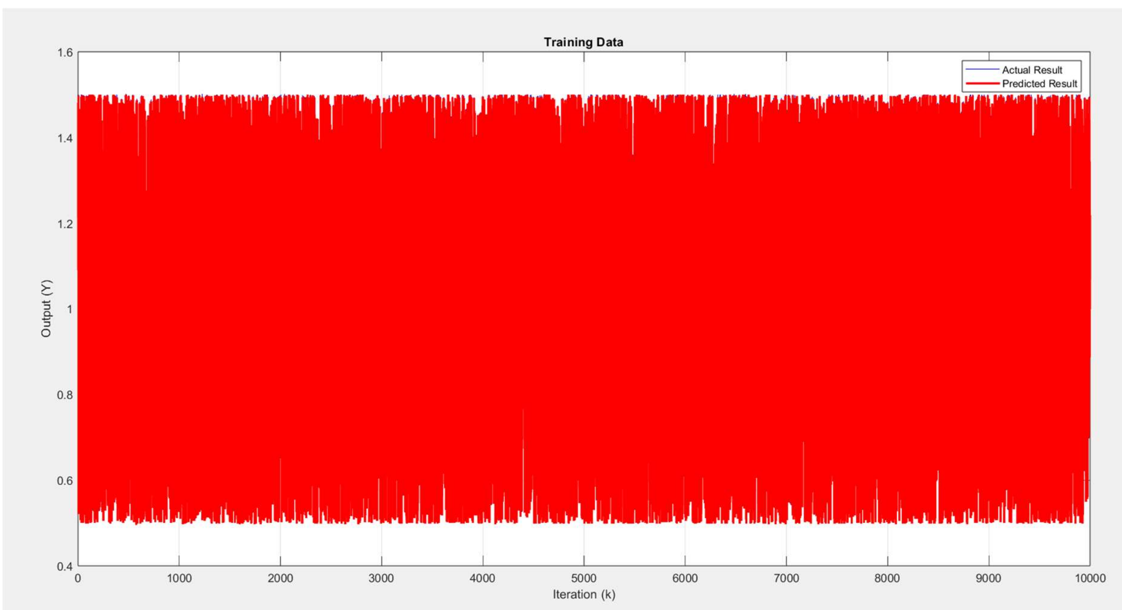

Fig: graph of training data set

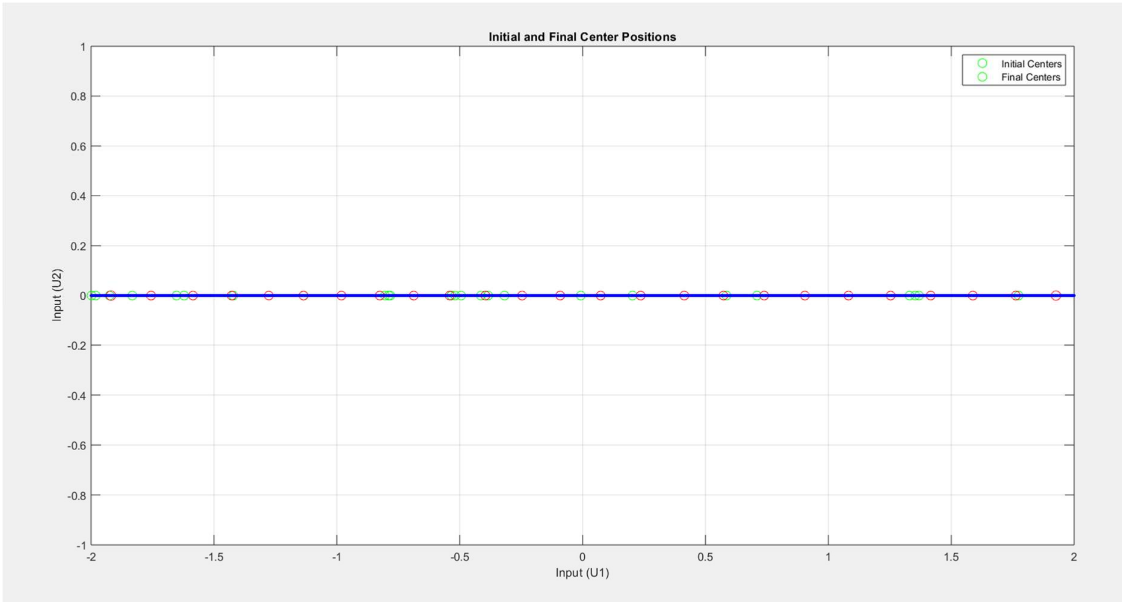

Fig: initial and final positions