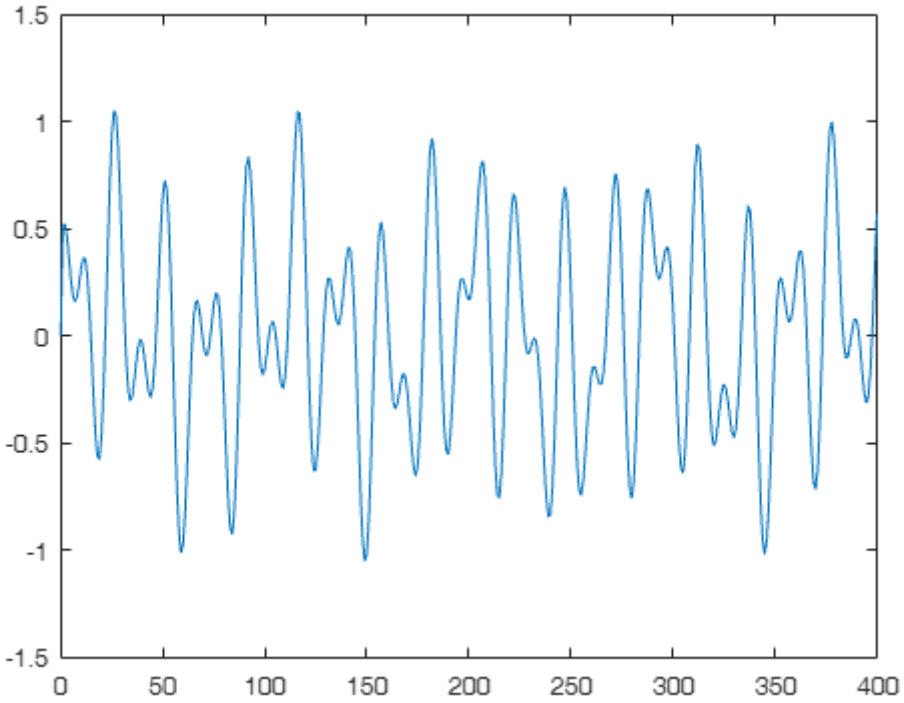
آرین حاجی زاده ۹۹۴۱۱۲۸۱

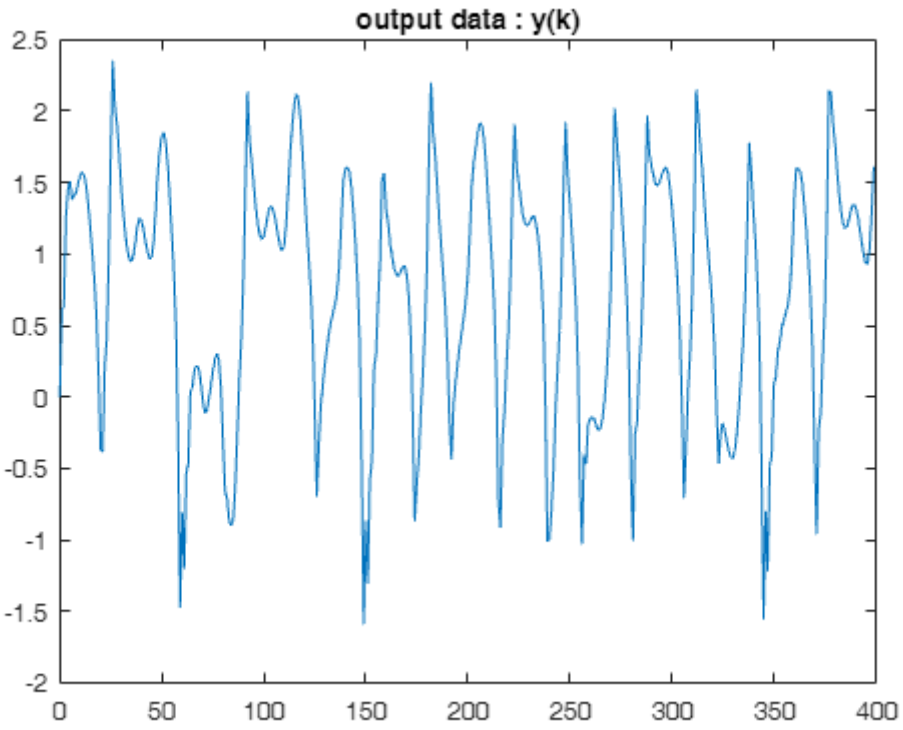
Given non-linear function

```
% input signal :
clear;
flag = 0;
u(1) = 0;
for k = 2:401
    u(k) = 0.5*sin(pi*(k-1)/11) + 0.4*cos(pi*(k-1)/6.5)+0.2*sin(pi*(k-1)/45);
end
figure();
plot(0:k-1,u);
```




```
alpha = 1.2;
beta = [1.1 1.5];


% y_nlf -> y non-linear function
y_nlf(1,:) = [0,0];
y_nlf(2,:) = alpha*u(2)*ones(1,2);
X(1,:) = [u(1) y_nlf(1,1) 0];
X(2,:) = [u(2) y_nlf(2,1) y_nlf(1,1)];
```

```
for k = 3:400
   y_nlf(k,:) = alpha*((y_nlf(k-1)*y_nlf(k-2)*(y_nlf(k-2) +
beta))/(1+(y_nlf(k-2)^2).*(y_nlf(k-1)^2)) + u(k));
   X(k,:) = [u(k), y_nlf(k-1,1),y_nlf(k-2)];
end


for i = 1:size(X,2)
    X(:,i) = X(:,i)./max(X(:,i));
end


tmp = 1;
tmp2 = 1;


% Cross varidation (train: 70%, test: 30%)
cv = cvpartition(size(X,1),'HoldOut',0.2);


idx = cv.test;


% Separate to training and test data
x_train = X(~idx,:);
y_train = y_nlf(~idx,:);


x_test = X(idx,:);
y_test = y_nlf(idx,:)./max(y_nlf(idx,:));


%normalizing
d_k = y_train(:,1)/max(y_train(:,1));


figure;
plot(0:399,y_nlf(:,1));
title("output data : y(k)")
```



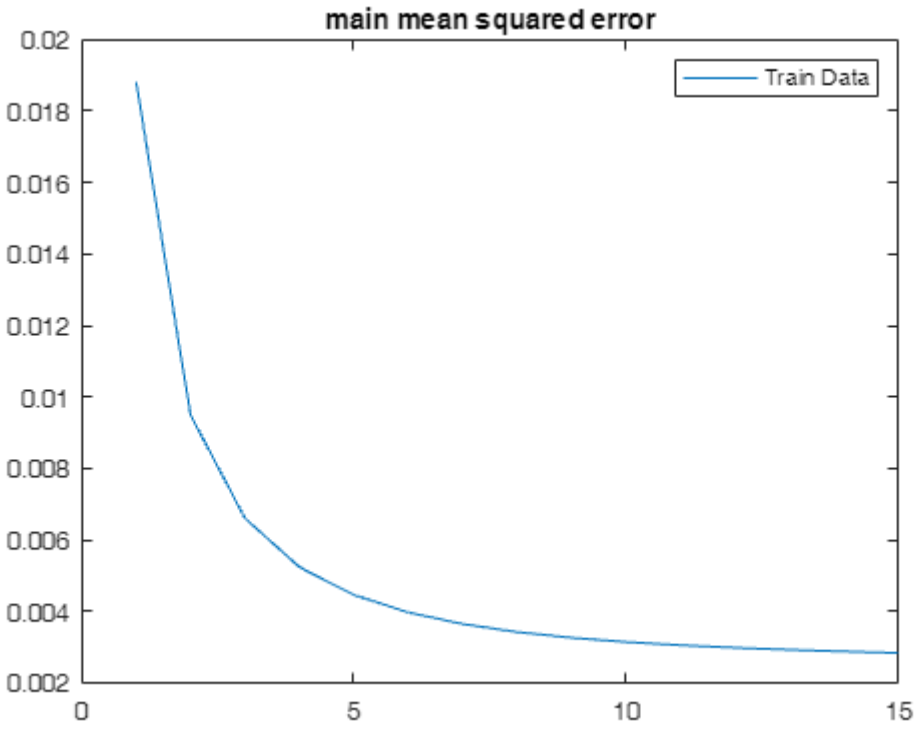

```
k = 40;



% [index,centers] = kmeans(x_train,k);
centers = myKmeans(x_train,k);
max_d = 0;
for q = 1:k
    for e = 1:k
        if(norm(centers(e)-centers(q)) > max_d)
            max_d = norm(centers(e)-centers(q));
        end
    end
end


% getting sigmas according to handouts and the book with K-means and RLS
sigma = max_d*max(max(d_k))/sqrt(2*k)  ;


% this function is defined at the end of the file
[output, Phi] = rbf_HL(x_train, centers, sigma);
[output_test, Phi_test] = rbf_HL(x_test,centers,sigma);
```

```
% adding bias to the weights
Phi = [Phi , ones(size(x_train,1),1)];
Phi_test = [Phi_test , ones(size(x_test,1),1)];



lambda = 0.08;
%this should be a small number


P{1} = lambda * eye(k+1);


r(:,1) = Phi(1,:)' * d_k(1);


% same as MLP
tmp_validation = 1;
loss_validation = 1;% initialization
for epochs = 1:100
    if epochs == 1
        w(:,1) = zeros(k+1,1);
        g(:,1) = zeros(k+1,1);
    else
        w(:,1) = w(:,size(x_train,1));
    end

    for n = 2:size(x_train,1)
        P{n} = P{n-1} - (P{n-1}*Phi(n,:)'*Phi(n,:)*P{n-1})/(1+Phi(n,:)*P{n-
1}*Phi(n,:)');
        g(:,n)= P{n} * Phi(n,:)';
        %prior estimation error
        pre(n) = d_k(n) - Phi(n,:)*w(:,n-1);
        w(:,n) = w(:,n-1) + g(:,n) * pre(n);
    end
    y_pred = Phi * w(:,end);


    mse(epochs) = mean((d_k - y_pred).^2);
    if mod(epochs, 1) == 0
        fprintf('Epoch %d, Loss: %f\n', epochs, mse(epochs));
        y_test_pred = Phi_test * w(:,end);
        loss_validation(tmp_validation) = mean((y_test(:,1) -
y_test_pred).^2);
        loss_compare(tmp_validation) = mse(epochs);
        fprintf('Epoch %d, Loss validation: %f\n\n', epochs,
loss_validation(tmp_validation));
        tmp_validation = tmp_validation + 1;
    end
    if(tmp_validation > 2)
```

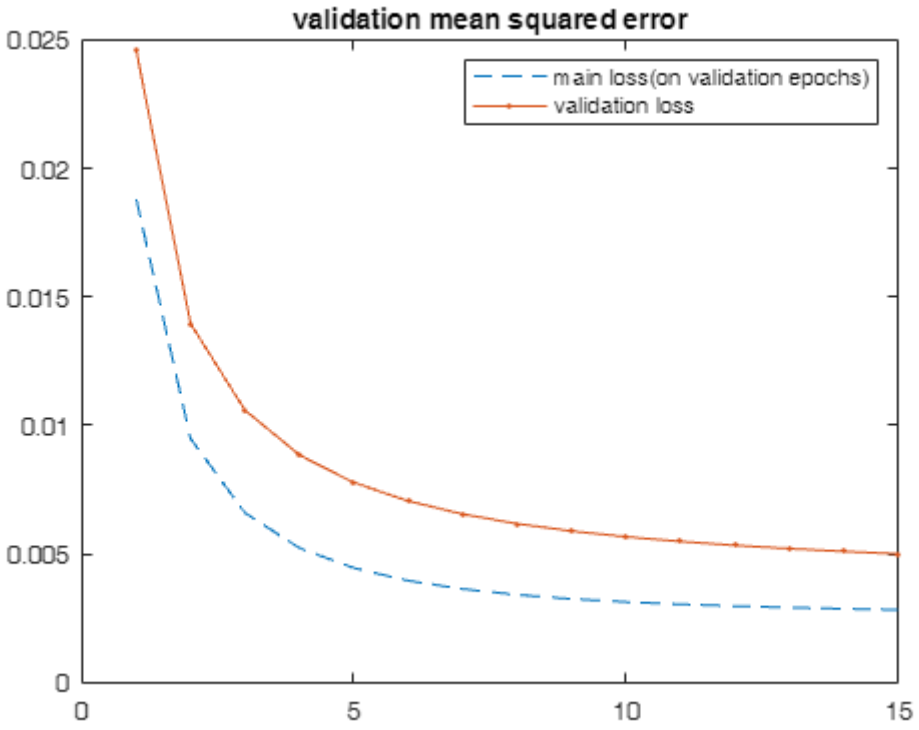
```
        if(abs(loss_validation(tmp_validation -1) -
loss_validation(tmp_validation -2)) < 0.0001 && ...
                abs(loss_validation(tmp_validation -1) -
loss_compare(tmp_validation -1)) < 0.01)
            tmp_epoch = epochs;
            break;
        end
    end
    tmp_epoch = epochs;
end
```

```
Epoch 1, Loss: 0.018819
Epoch 1, Loss validation: 0.024600
Epoch 2, Loss: 0.009494
Epoch 2, Loss validation: 0.013958
Epoch 3, Loss: 0.006606
Epoch 3, Loss validation: 0.010596
Epoch 4, Loss: 0.005240
Epoch 4, Loss validation: 0.008859
Epoch 5, Loss: 0.004461
Epoch 5, Loss validation: 0.007786
Epoch 6, Loss: 0.003971
Epoch 6, Loss validation: 0.007067
Epoch 7, Loss: 0.003646
Epoch 7, Loss validation: 0.006558
Epoch 8, Loss: 0.003421
Epoch 8, Loss validation: 0.006183
Epoch 9, Loss: 0.003261
Epoch 9, Loss validation: 0.005898
Epoch 10, Loss: 0.003142
Epoch 10, Loss validation: 0.005674
Epoch 11, Loss: 0.003052
Epoch 11, Loss validation: 0.005493
Epoch 12, Loss: 0.002982
Epoch 12, Loss validation: 0.005343
Epoch 13, Loss: 0.002924
Epoch 13, Loss validation: 0.005216
Epoch 14, Loss: 0.002877
Epoch 14, Loss validation: 0.005107
Epoch 15, Loss: 0.002836
Epoch 15, Loss validation: 0.005011
```

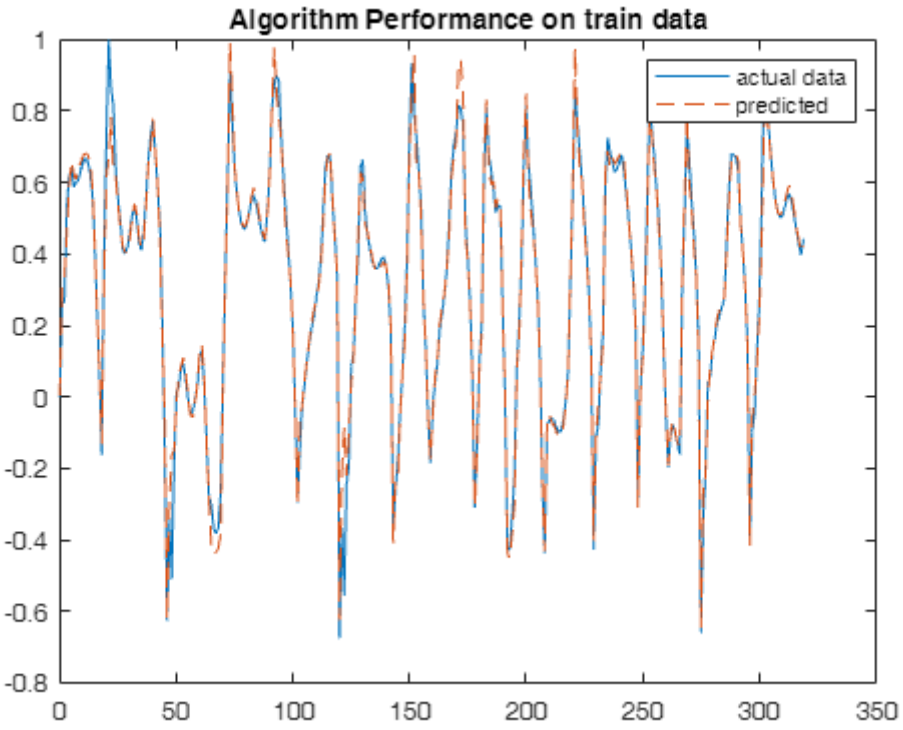
```
figure();
plot([1:tmp_epoch],mse);
title("main mean squared error");
legend("Train Data")
```



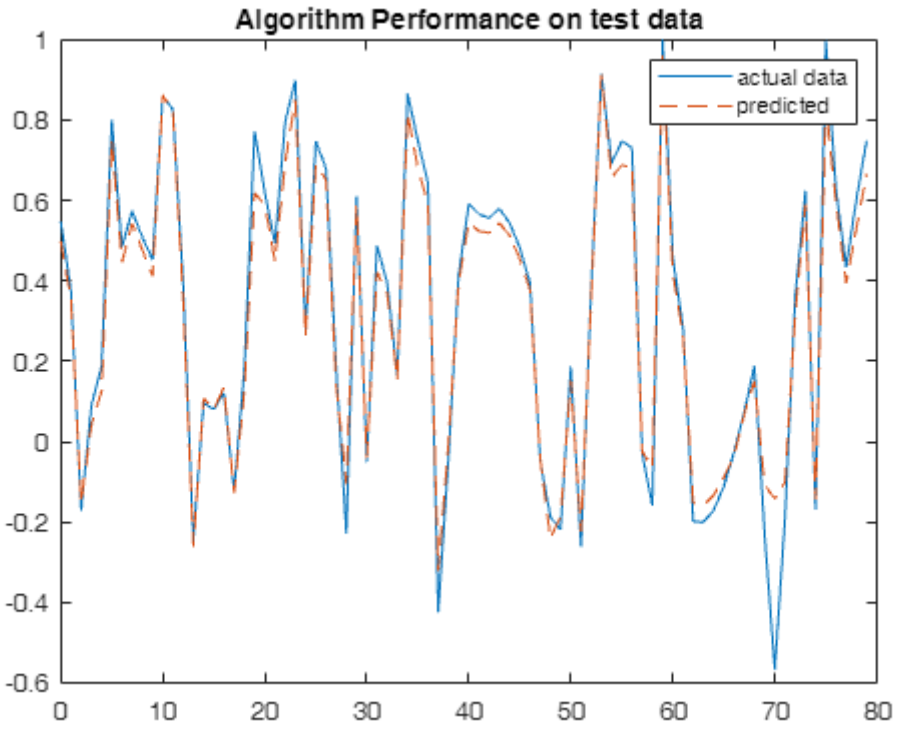

```
figure();
plot([1:tmp_validation-1],loss_compare,'--');
hold on
plot([1:tmp_validation-1],loss_validation,'.-');
title("validation mean squared error")
legend(["main loss(on validation epochs)","validation loss"])
```




```
figure;
plot(0:size(d_k,1)-1,d_k);
hold on
plot(0:size(d_k,1)-1,y_pred,"--");
title("Algorithm Performance on train data");
legend(["actual data","predicted"])
```




```
figure;
plot(0:size(x_test,1)-1,y_test(:,1));
hold on
plot(0:size(x_test,1)-1,y_test_pred,"--");
title("Algorithm Performance on test data");
legend(["actual data","predicted"]);
```




```
function [output, Phi] = rbf_HL(X, centers, sigma)
    num_data = size(X, 1);
    num_centers = size(centers, 1);
    Phi = zeros(num_data, num_centers);

    for i = 1:num_data
        for j = 1:num_centers
            Phi(i, j) = exp(-norm(X(i,:) - centers(j,:))^2 / (2*sigma^2));
        end
    end

    output = Phi; % Just return Phi if we don't have target values
end


function [centers] = myKmeans(X, k)
    % Randomly initialize the cluster centers
    num_samples = size(X, 1);
    random_indices = randperm(num_samples, k);
    centers = X(random_indices, :);


    % Initialize variables
    cluster_assignment = zeros(num_samples, 1);
```

```
    max_iters = 100;
    iter = 0;

    while iter < max_iters
        iter = iter + 1;

        % Assign each sample to the nearest center
        for i = 1:num_samples
            distances = sum((X(i, :) - centers) .^ 2, 2);
            [~, min_index] = min(distances);
            cluster_assignment(i) = min_index;
        end

        % Update centers
        new_centers = zeros(size(centers));
        for j = 1:k
            cluster_points = X(cluster_assignment == j, :);
            if ~isempty(cluster_points)
                new_centers(j, :) = mean(cluster_points, 1);
            else
                % Reinitialize empty cluster
                new_centers(j, :) = X(randi(num_samples), :);
            end
        end

        % Check for convergence
        if all(new_centers == centers)
            break;
        end

        centers = new_centers;
    end
end
```