## 1.1 Polynomial Regression

### Matlab-Code

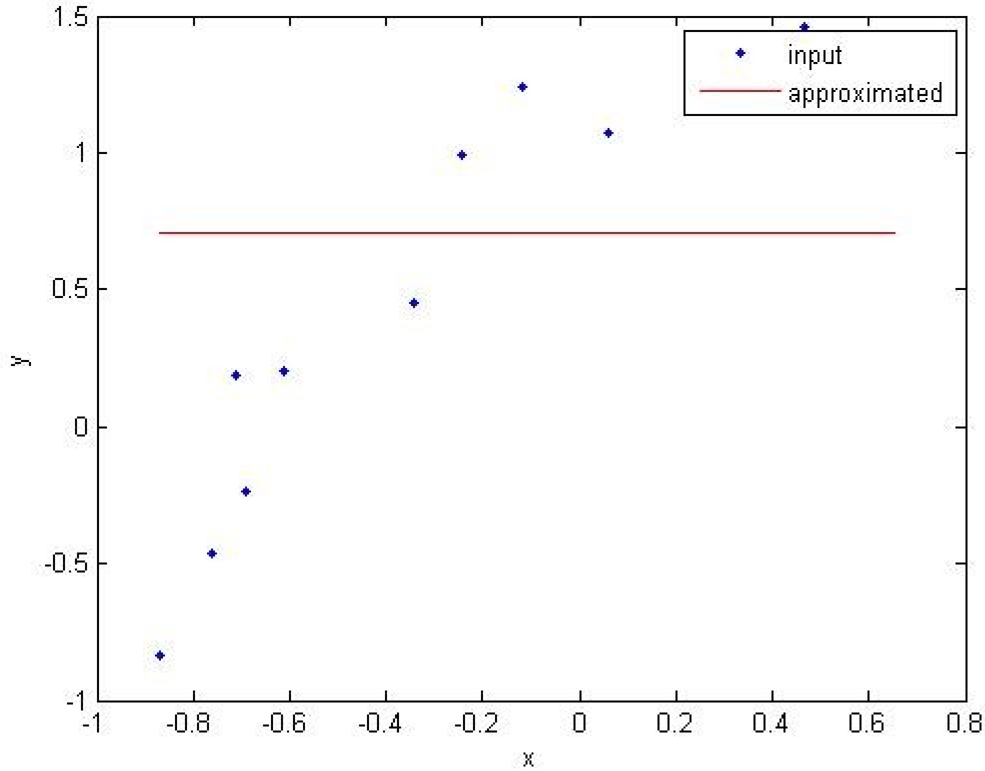
```
clear all;
close all;

load('linearregression_homework.mat');

l = 0:6;

calculate_plot( l,x_train,y_train );
calculate_plot( l,x_test,y_test,y_test_nonoise );


function calculate_plot( l,x,y,y_nonoise )
    x_length = length(x);
    mse = l;
    for degree = l
        X_matrix = ones(x_length, degree+1);
        for exponent = 0:degree
           X_matrix(:,exponent+1) = x .^ exponent;
        end
        W = (X_matrix' * X_matrix)^-1 * X_matrix' * y;
        y_calc = X_matrix * W;
        mse(degree+1) = sum((y_calc - y) .^ 2)  / x_length

       figure;
       plot(x, y, '.b');
       hold all;
       plot(x, y_calc, '-r');

       titles = ['input', 'approximated'];
       if nargin == 4
           plot(x, y_nonoise, '-g');
           legend('input', 'approximated', 'input without noise');
       else
           legend('input', 'approximated');
       end
       xlabel('x');
       ylabel('y');
    end
    figure;
    plot(l,mse);
    title('Mean squared error as a function of the degree.');

end
```

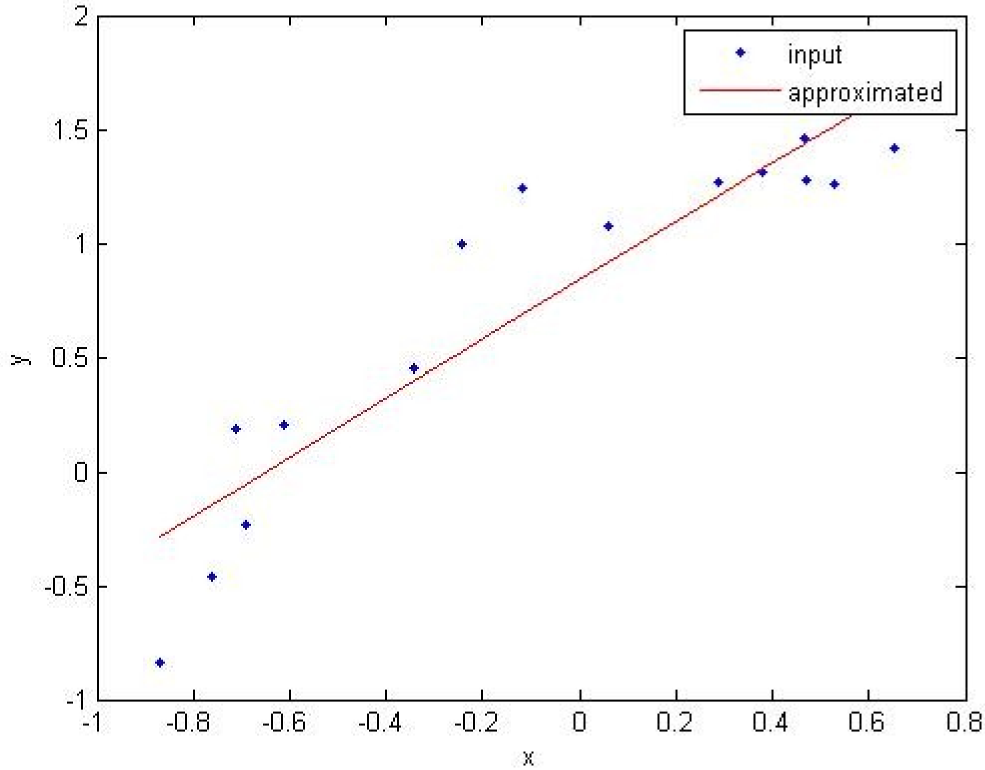
### Which value of l would you choose?

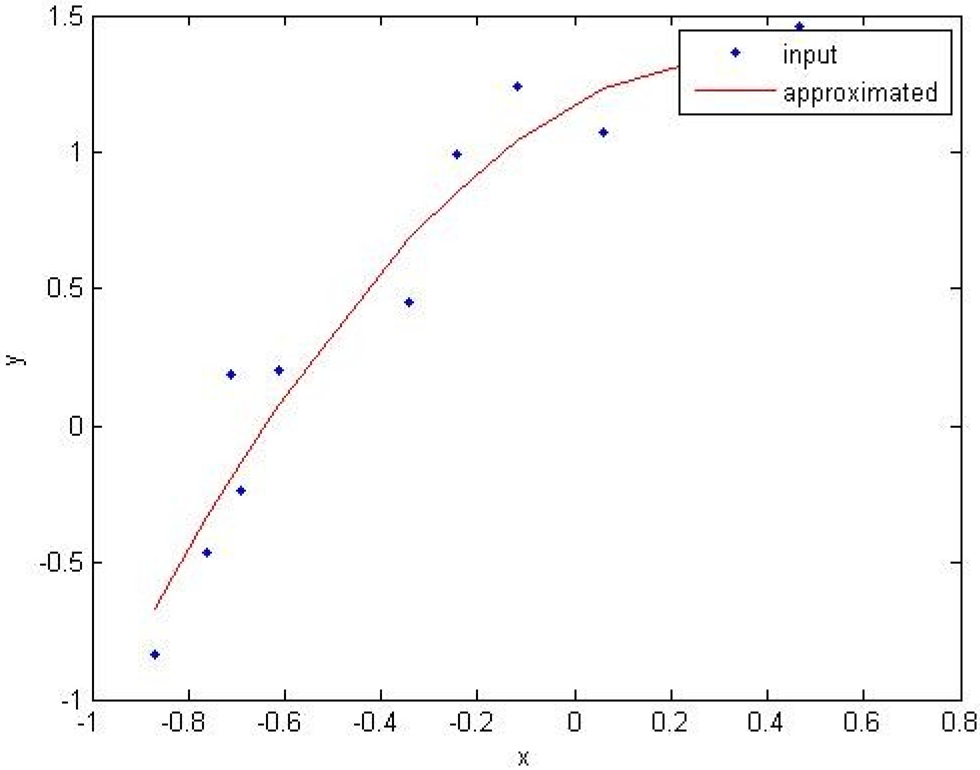
We will choose the value 2 for l.

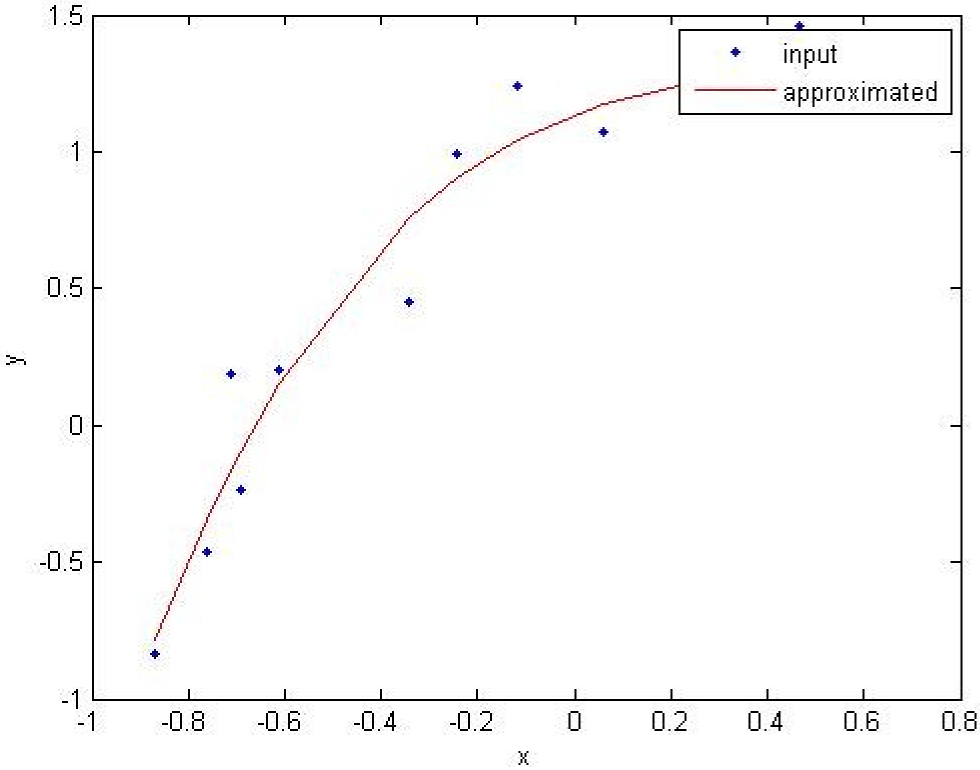
## Plots

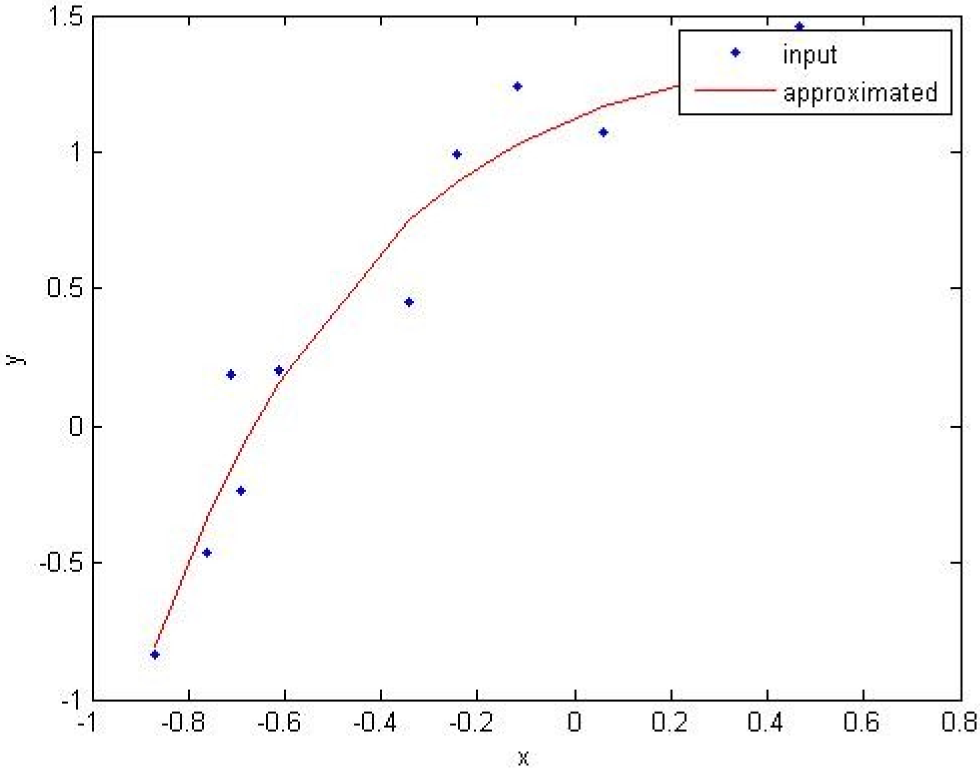
### Training data-points



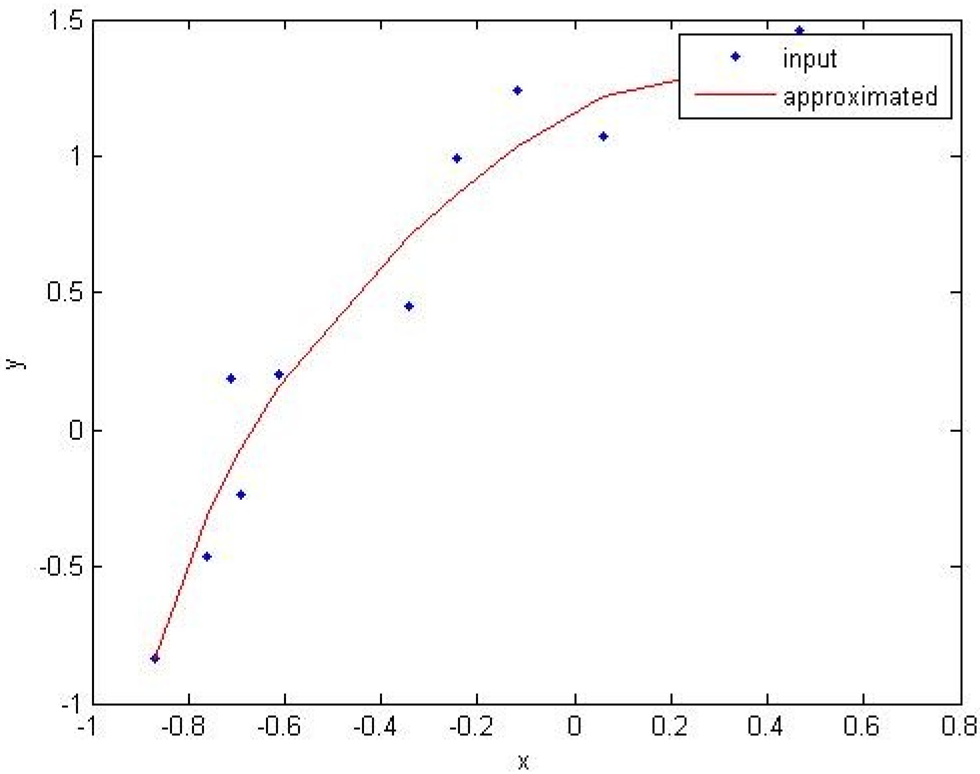




input
approximated
x
y

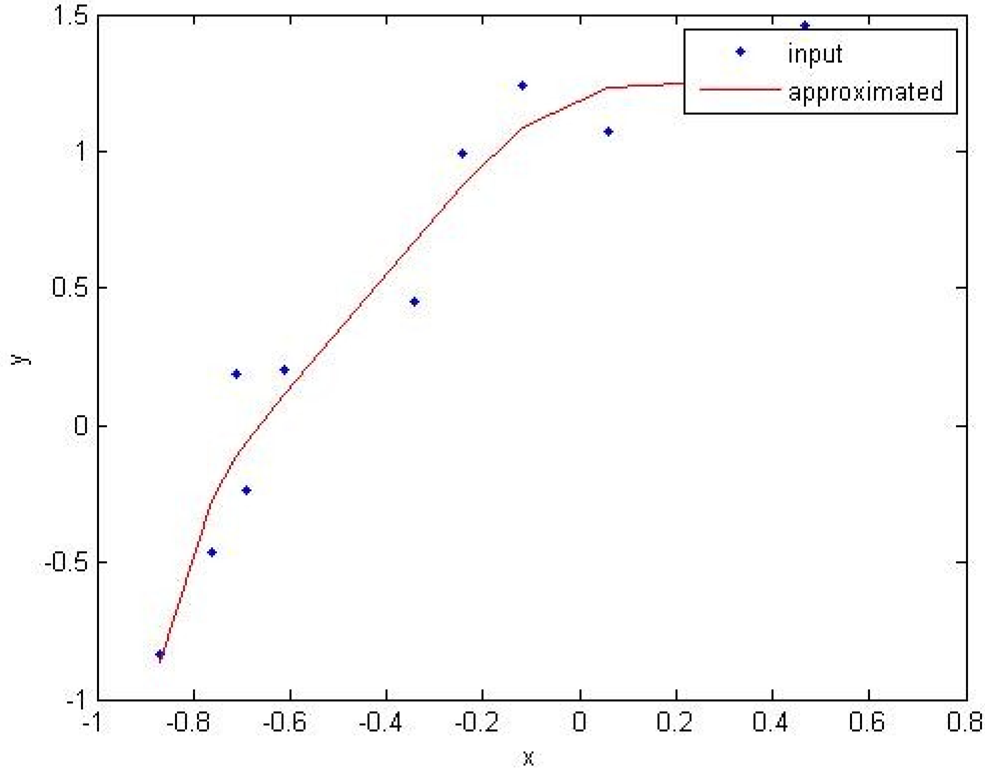

input
approximated
x
y

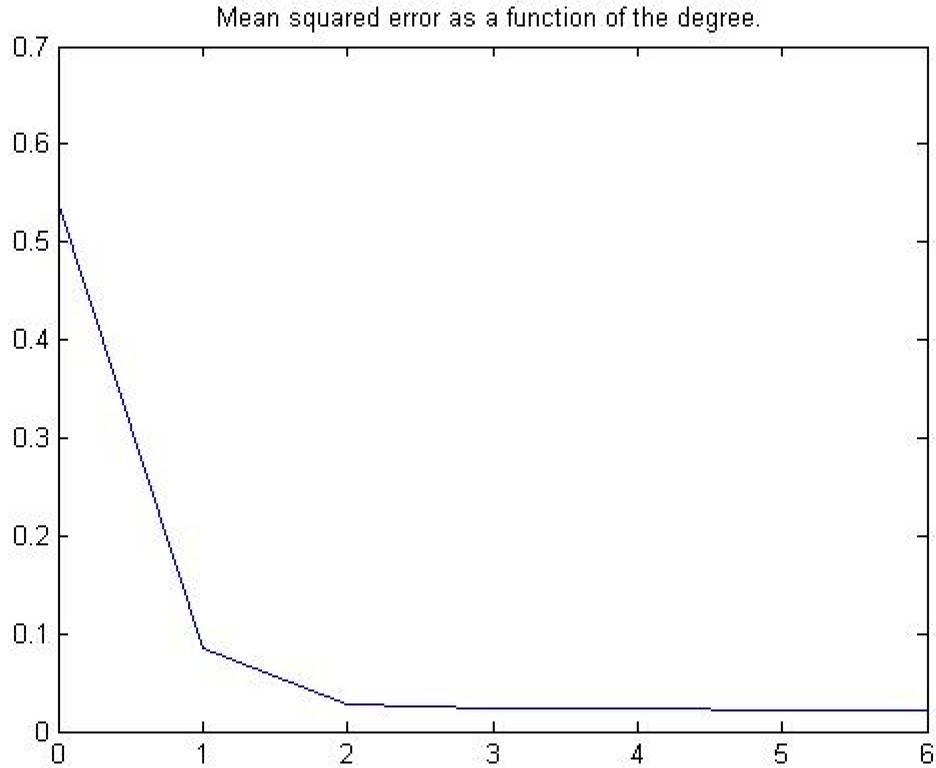


input
approximated
y
x
1.5
1
0.5
0
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0.2
0.4
0.6
0.8

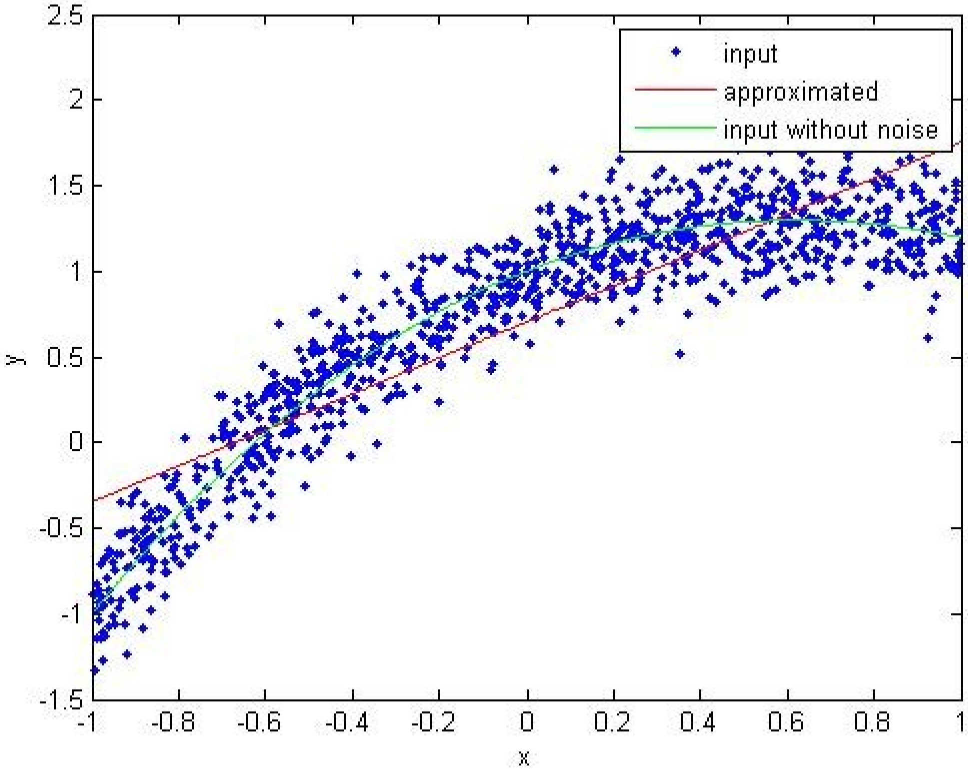



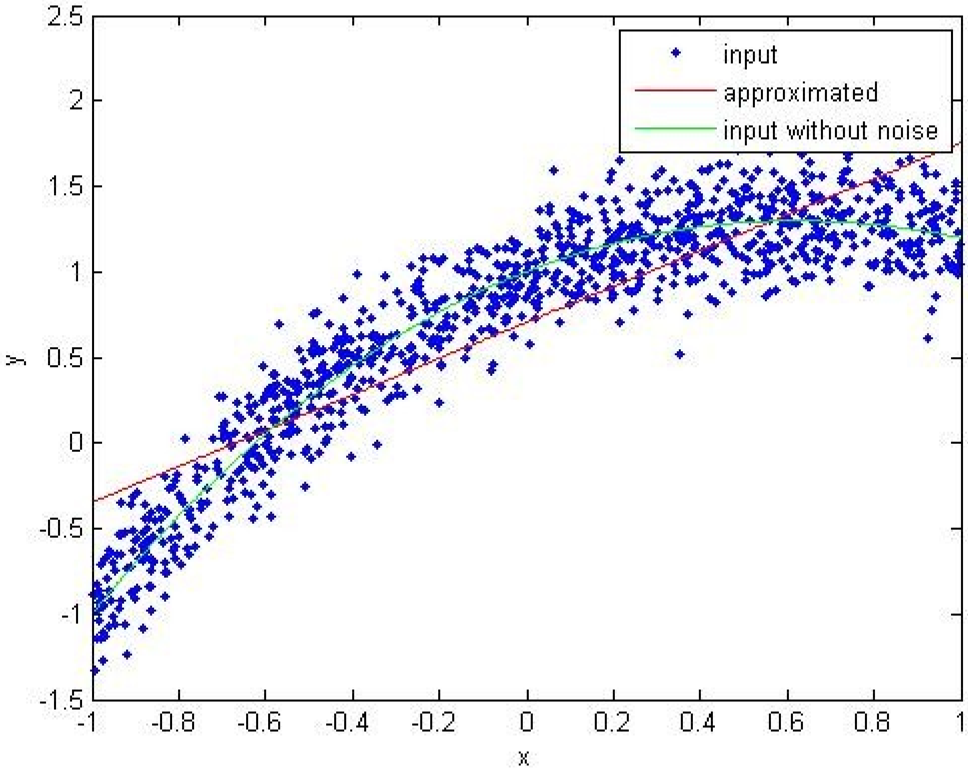
input
approximated
y
x
1.5
1
0.5
0
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0.2
0.4
0.6
0.8

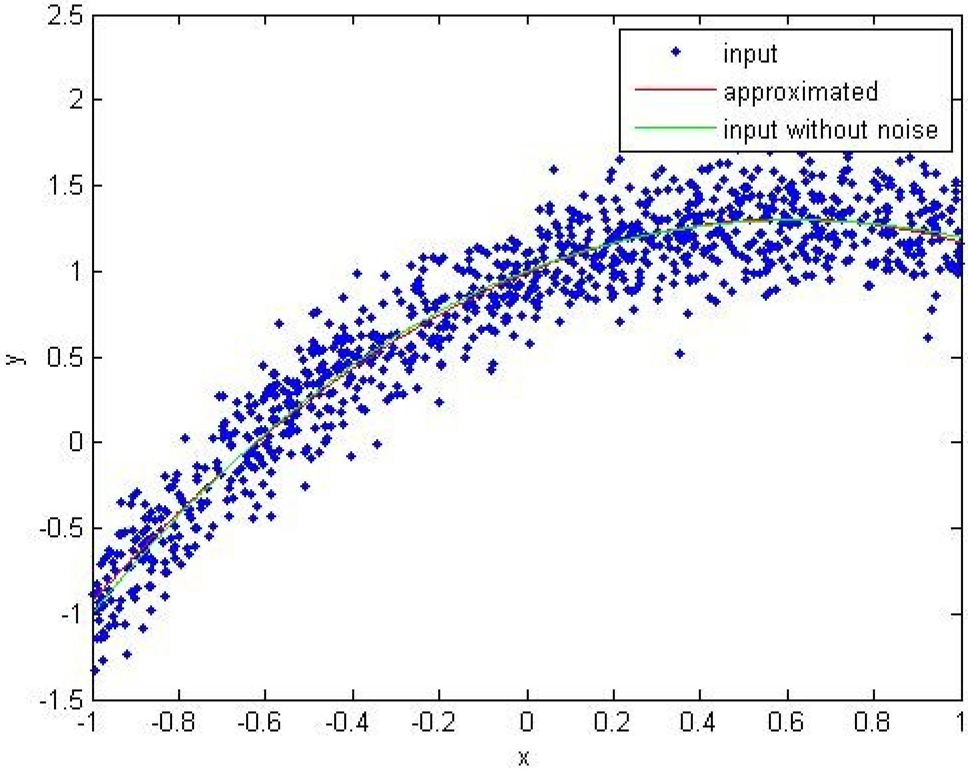

input
approximated
y
x
1.5
1
0.5
0
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8


Mean squared error as a function of the degree.
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1
2
3
4
5
6

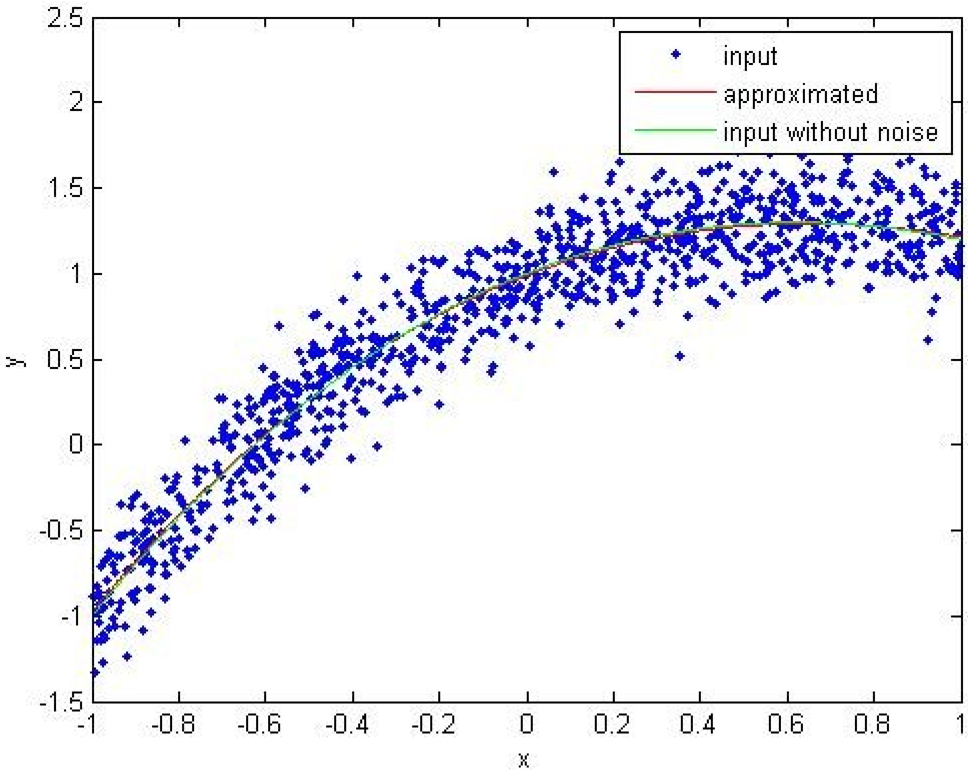
**Test data-points**



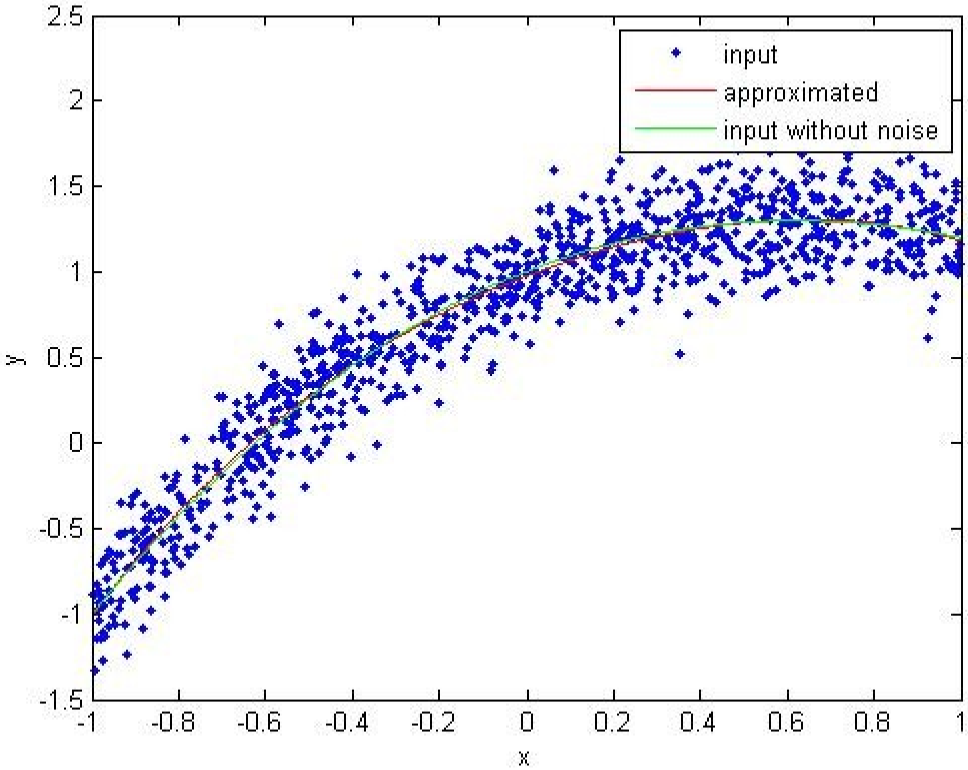



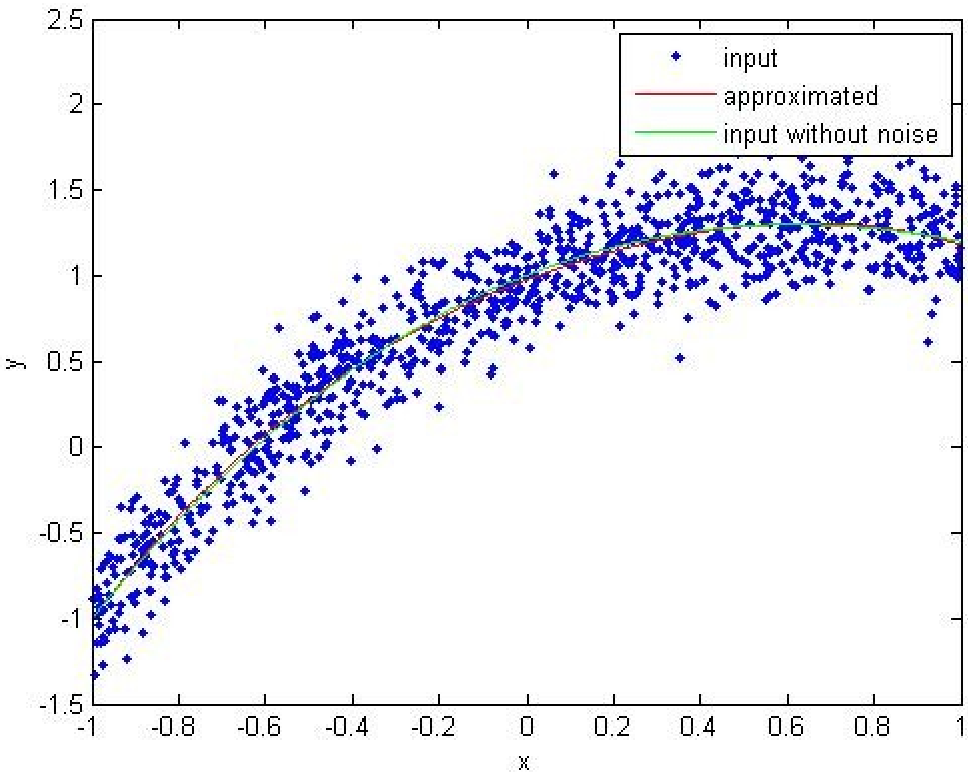


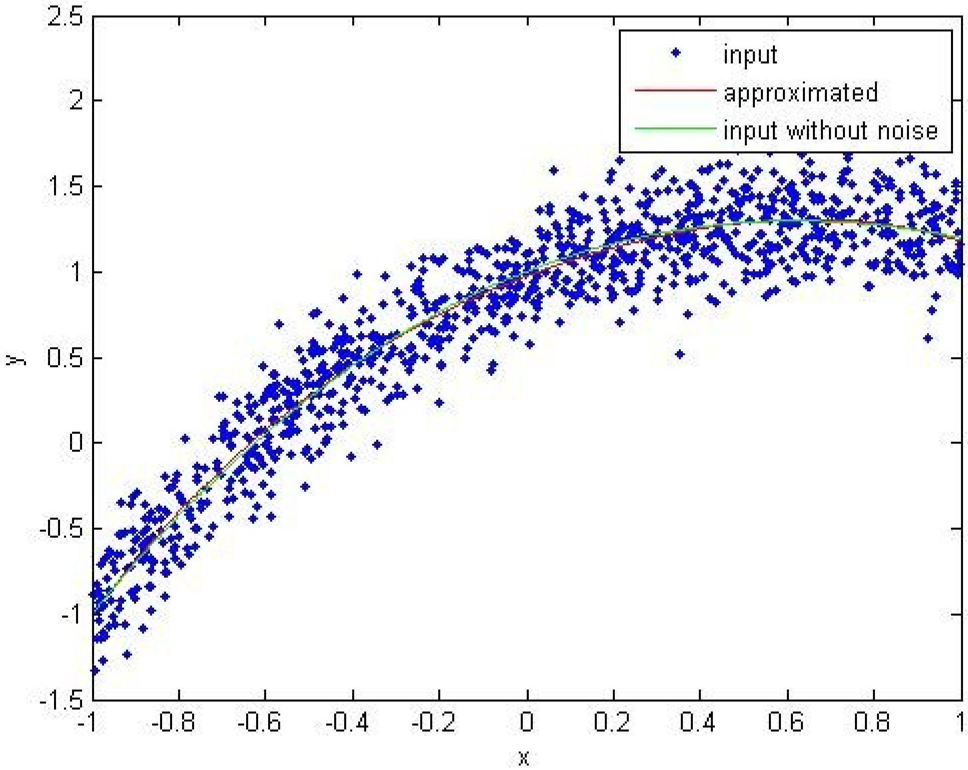
input
approximated
input without noise
y
x
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

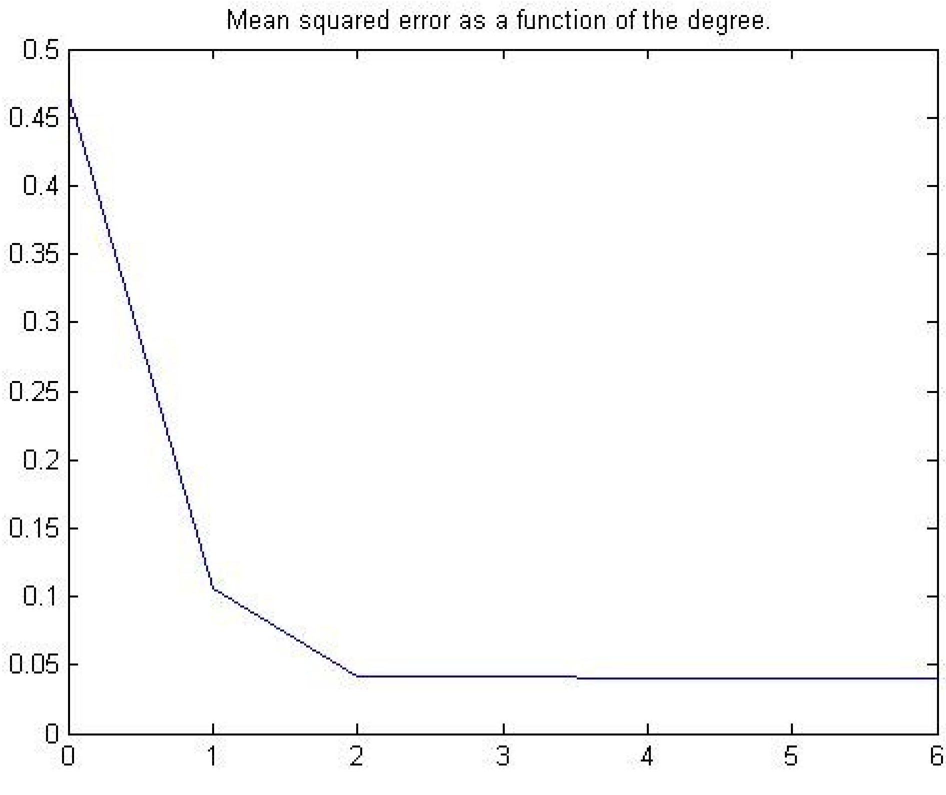



input
approximated
input without noise
y
x
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1


input
approximated
input without noise
y
x
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1


input
approximated
input without noise
y
x
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1


input
approximated
input without noise
x
y


Mean squared error as a function of the degree.

## 1.2 Regularized Polynomial Regression

### Matlab-Code

```
clear all;
close all;

load('linearregression_homework.mat');
degree = 10;
space = logspace(-3, 0, 10);
train_plot_style = 'b-';
test_plot_style =  'm-';

figure;
[calc_y_train, mse_train, w_train] = plot_mse_alpha(x_train, y_train, space,
degree, train_plot_style);
hold on;
[calc_y_test, mse_test, w_test] = plot_mse_alpha(x_test, y_test, space, degree,
test_plot_style);
legend('training data', 'test data');
hold off;


lowest_alpha_index = 1;
highest_alpha_index = length(space);

figure;
hold on;

    [min_mse, best_alpha_index] = min(mse_train);

    handle = plot(x_train, calc_y_train(lowest_alpha_index, :), 'b-');
    set(handle, 'linewidth', 3);
    plot(x_train, calc_y_train(highest_alpha_index, :), 'g-');
    plot(x_train, calc_y_train(best_alpha_index, :), 'r-');

    plot(x_train, y_train, 'co');
    plot(x_test, y_test_nonoise, 'm-');

xlabel('x');
ylabel('y');
legend('y for lowest alpha', 'y for highest alpha', 'y for best alpha',
'training points', 'target function');
title('y for different alphas');
hold off;

w_mean = zeros(length(space), 1);
for index = 1:length(space)
   w = w_train(index, :)
   w_mean(index) = mean(abs(w));
end

figure;
semilogx(space, w_mean);
title('mean absolute weight values');
```

```
function [y_calc, mse, w] = plot_mse_alpha(x, y, space, degree, plot_style)
    x_length = length(x);
    X = ones(x_length, degree+1);
    for exponent = 0:degree
       X(:,exponent+1) = x .^ exponent;
    end

    space_length = length(space);
    mse = zeros(space_length, 1);
    y_calc = zeros(space_length, x_length);
    w = zeros(space_length, degree+1);
    for index = 1:length(space)
        alpha = space(index);
        current_w = calculate_weight( alpha, X, y, degree );

        current_y_calc = X * current_w;

        w(index, :) = current_w;
        y_calc(index, :) = current_y_calc;

        mse(index) = sum((current_y_calc - y) .^ 2)  / x_length;
    end

    semilogx(space, mse, plot_style);
    xlabel('alpha');
    ylabel('mean squared error');
end


function [ w ] = calculate_weight( alpha, X, y, degree )
    N = length(y);

    w = inv(alpha^2*eye(degree+1) + (X' * X) / N) * ((X'*y)/N);
end
```

## 1.2 Regularized Polynomial Regression

### o) Matrix-Form

$$E(w) = \frac{1}{N} \sum_{i=1}^{N} \left( y_i - \sum_{k=0}^{10} x_i^{k} w_k \right)^2 + \alpha^2 \sum_{k=0}^{10} w_k^2$$

$$\alpha^2 \sum_{k=0}^{10} w_k^2 = \alpha^2_{1\times N} \cdot \vec{w}^T \cdot \vec{w}$$

$$E(w) = \frac{1}{N} \overbrace{\left( \underbrace{\vec{y}}_{N\times 1} - \underbrace{X}_{N\times 11} \cdot \underbrace{\vec{w}}_{11\times 1} \right)}^{}{}^{T} \cdot \left( \vec{y} - X \cdot \vec{w} \right)$$

$$X = \begin{pmatrix} x_0^0 & x_0^1 & \dots & x_0^{10} \\ x_1^0 & x_1^1 & \dots & x_1^{10} \\ \vdots & & & \\ x_N^0 & x_N^1 & \dots & x_N^{10} \end{pmatrix}$$

Dim

### o) Weight-Vector

$$E'(w) = 0$$

$$E'(w) = \frac{1}{N} \cdot 2 \left( \vec{y} - X\vec{w} \right)^T \cdot (-X) + \alpha^2 \cdot 2 \cdot \vec{w}^T$$

$$\cancel{2} \cdot \alpha^2 \vec{w}^T = \frac{\cancel{2}}{N} \left( \vec{y} - X\vec{w} \right)^T \cdot X$$

$$\alpha^2 \vec{w} = \frac{1}{N} X^T \left( \vec{y} - X\vec{w} \right)$$

$$\alpha^2 \vec{w} = \frac{1}{N} \left( X^T \cdot \vec{y} - X^T \cdot X \cdot \vec{w} \right)$$

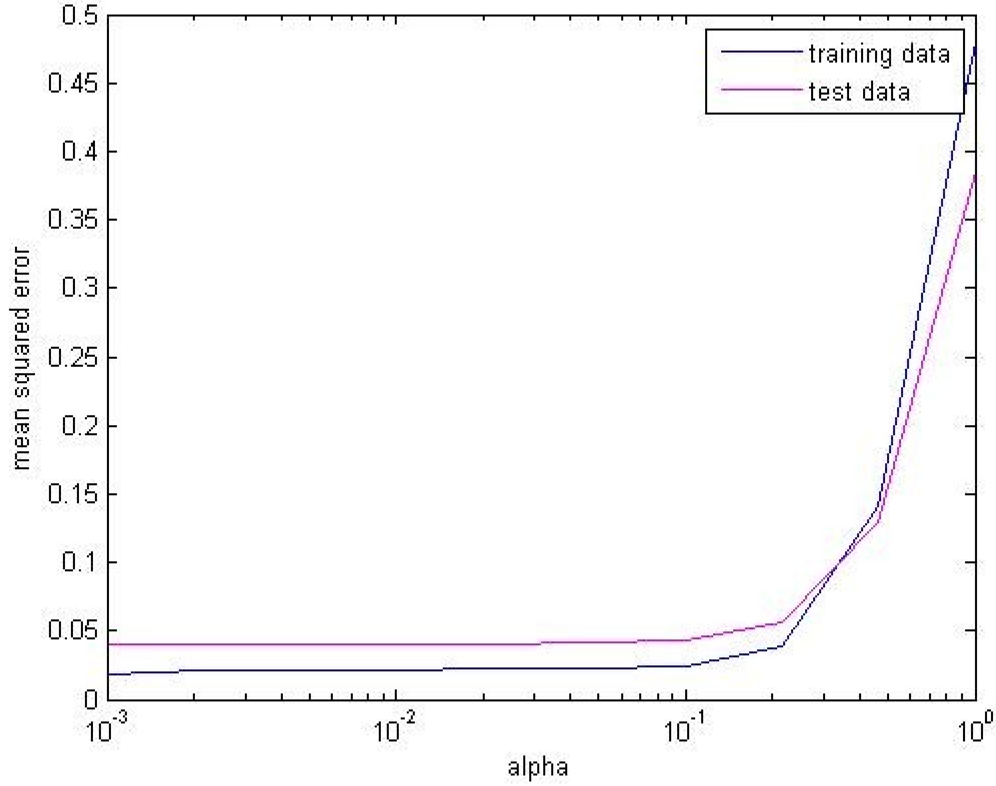
$$\alpha^2 \cdot I \cdot \vec{w} + X^T \cdot X \vec{w} \cdot \frac{1}{N} = \frac{1}{N} X^T \cdot \vec{y}$$

$$\left( \alpha^2 \cdot I + \frac{1}{N} \cdot X^T \cdot X \right) \cdot \vec{w} = \frac{1}{N} \cdot X^T \cdot \vec{y}$$

$$\vec{w} = \left( \alpha^2 \cdot I + \frac{1}{N} \cdot X^T \cdot X \right)^{-1} \cdot \frac{1}{N} \cdot X^T \cdot \vec{y}$$

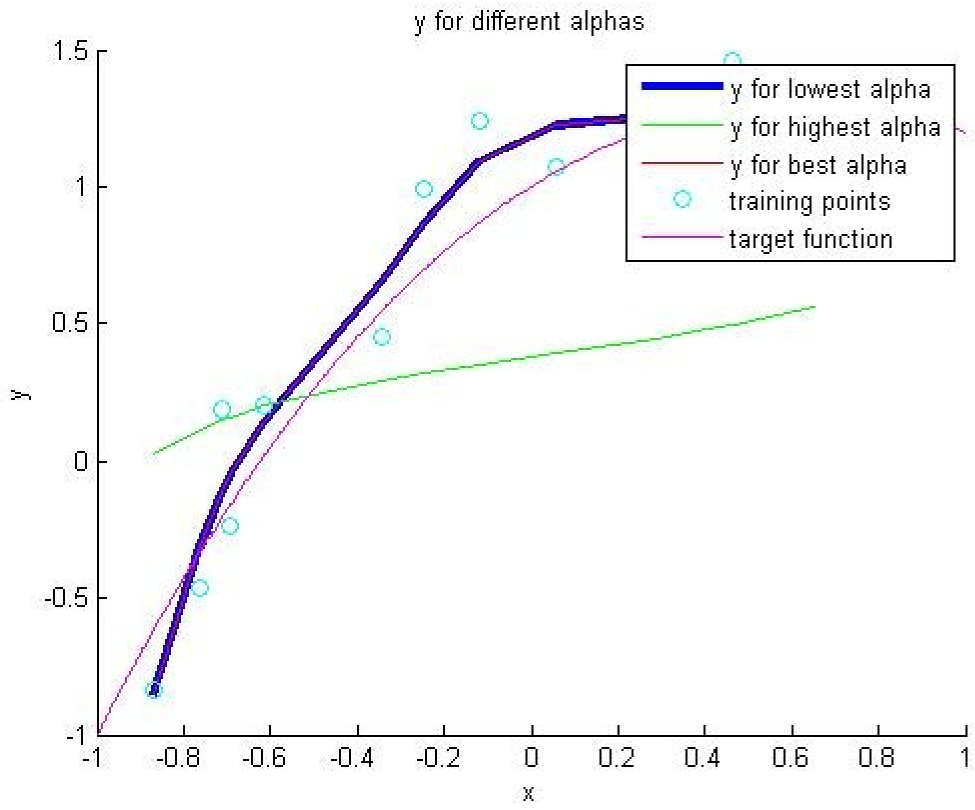
**<u>Plots</u>**

**Mean squared error of the training and of the test set for the given alphas.**

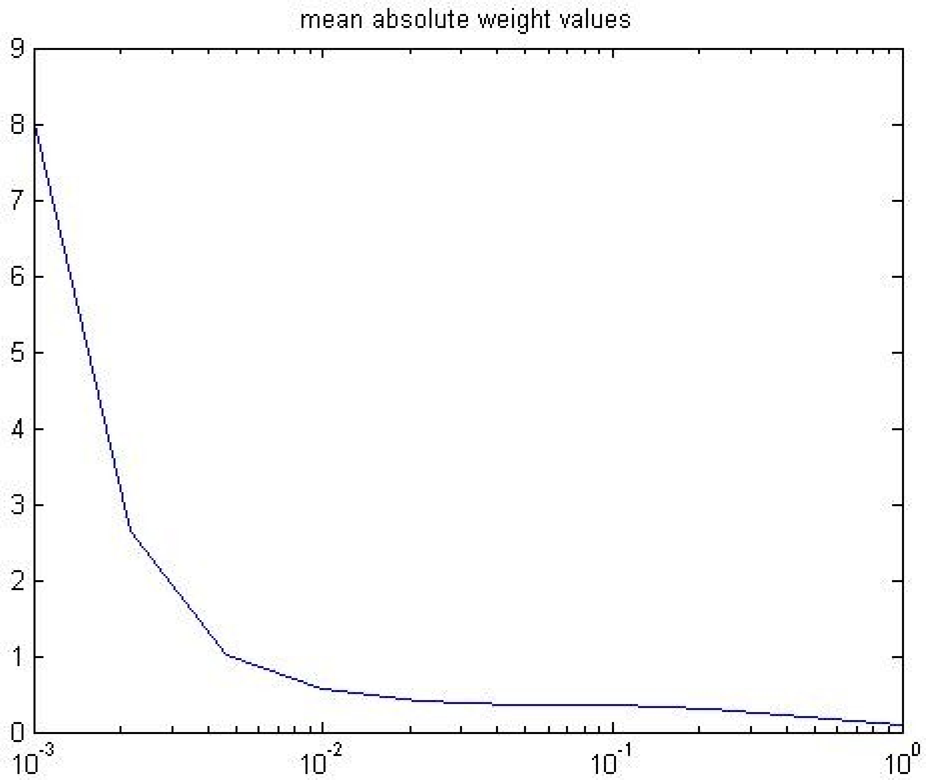



<u>What is the best alpha?</u>

**Learned functions for the lowest, the highest and the best alpha.**



**Mean absolute weight values for the given alpha.**



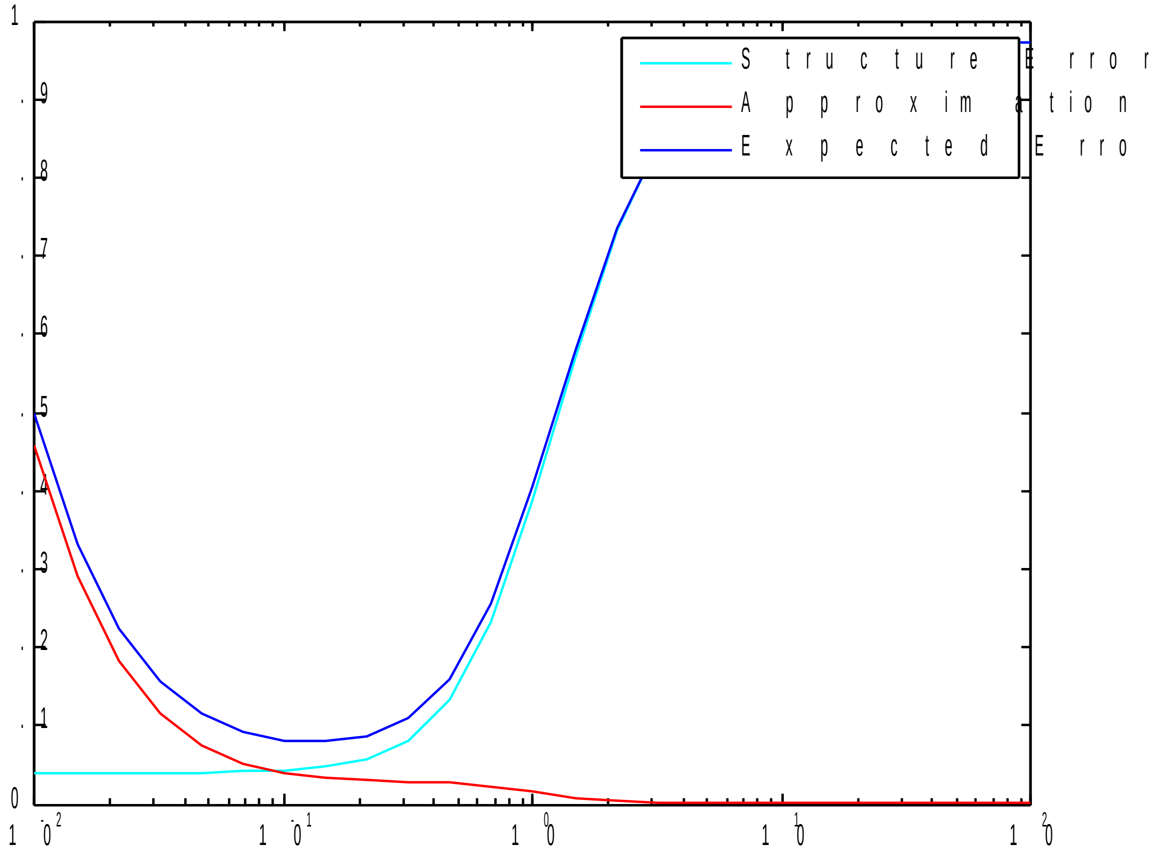

Interpretation:

## 1.3 Decomposition into structural and approximation error

**Matlab-Code:**

```
clear all;
close all;

load('linearregression_homework_10.mat');


nDataSets = length(x_train);

% Calculate optimal w
% This can be calculated with the big dataset (x_true, y_true
degree = 10;
X_true = calculate_matrix(x_true, degree);

nr_alphas = 25;
alphas = logspace(-2, 2, nr_alphas);
structure_errors = zeros(nr_alphas, 1);
approx_errors = zeros(nr_alphas, 1);
for i = 1:nr_alphas
    alpha = alphas(i);
    w_opt = calculate_weight(alpha, X_true, y_true, degree );
    structure_errors(i) = mean((y_true - X_true * w_opt) .^ 2);

    approx_error = zeros(nDataSets, 1);
    for j = 1:nDataSets
        X_train = calculate_matrix(x_train{j}, degree);
        w = calculate_weight(alpha, X_train, y_train{j}, degree);
        approx_error(j) = mean((X_true * w - X_true * w_opt) .^ 2);
    end

    approx_errors(i) = mean(approx_error);

end

expected_errors = structure_errors + approx_errors;

figure;
semilogx(alphas, structure_errors, 'c-');
hold on;
semilogx(alphas, approx_errors, 'r-');
semilogx(alphas, expected_errors, 'b-');
hold off;
legend('Structure Error', 'Approximation Error', 'Expected Error');



function [ w ] = calculate_weight( alpha, X, y, degree )
    N = length(y);
    w = inv(alpha^2*eye(degree+1) + (X' * X) / N) * ((X'*y)/N);
end


function [ X ] = calculate_matrix( x, degree )
    x_length = length(x);
    X = ones(x_length, degree+1);
    for exponent = 0:degree
       X(:,exponent+1) = x .^ exponent;
    end
end
```

**Plots**




Compare your results to the previous example. Is there a difference in the best value? If so, why?