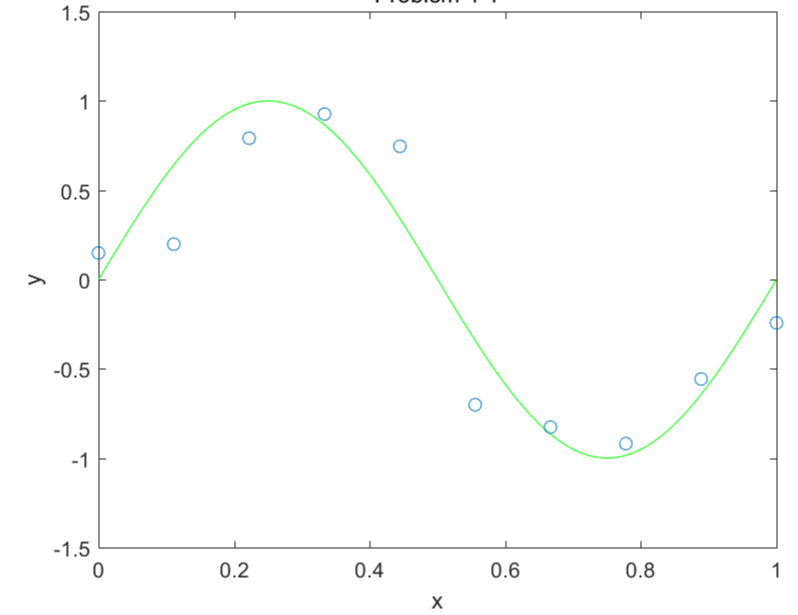
Assignment 1

I used MATLAB to solve the assignment. High version(maybe >=2017a) of MATLAB and some toolboxes are required to use functions in my scripts.

# Question 1

Plot 10 samples, spaced uniformly in range [0, 1], with the function sin(2) with Gaussian noise like below.



## CODE

% set figure option

figure(1), ylim([-1.5, 1.5]), title('Question 1'), xlabel('x'), ylabel('y'), hold on;

% plot original signal y = sin(2 \* pi \* x)

plot(linspace(0, 1), sin(2 \* pi \* linspace(0, 1)), 'g'), hold on;

% generate gaussian noise samples with range = [0, 1], number of samples = 10

gaussianNoise\_samples\_10 = GaussianNoise\_generator([0, 1], 10);

% plot gaussian noise samples

plot(gaussianNoise\_samples\_10.x, gaussianNoise\_samples\_10.y, 'o'), hold on;

Code 1 / Question1.m

Simply plot sine graph on given range [0, 1], and generate given number of samples with gaussian noise added coordinates.

Here is my defined function GaussianNoise\_generator.m

function [ gaussianNoise ] = GaussianNoise\_generator( range, numberOf\_samples )

%GAUSSIANNOISE\_GENERATOR: generates given number of samples coordinates that spaced uniformly in given range, with the function sin(2 \* pi \* x) added gaussian noise. returning coordinates are table. 'x' means x-coordinates, and 'y' means y-coordinates.

x = linspace(range(1), range(2), numberOf\_samples);

y = awgn(sin(2 \* pi \* x), 10);

gaussianNoise = table(x, y);

end

Code 2 / GaussianNoise\_generator.m

out = awgn(in,snr) adds white Gaussian noise to the vector signal in. This syntax assumes that the power of in is 0 dBW.

## RESULT

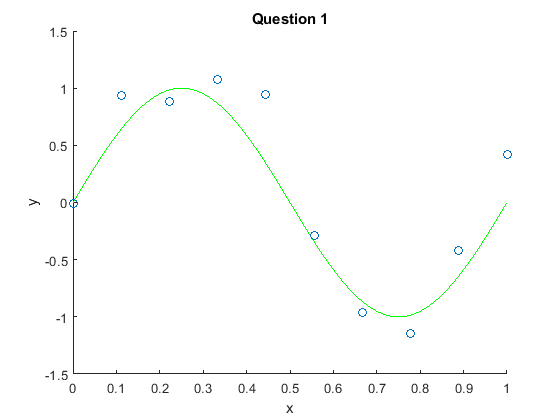


Figure 1 / Result of Question 1

# Question 2

Generate regression lines with polynomial basis function with order 1, 3, 5, and 9.

## CODE

% set figure option

figure(2), ylim([-1.5, 1.5]), title('Question 2'), xlabel('x'), ylabel('y'), hold on;

% plot original signal y = sin(2 \* pi \* x)

plot(linspace(0, 1), sin(2 \* pi \* linspace(0, 1)), 'g'), hold on;

% plot already made samples.

plot(gaussianNoise\_samples\_10.x, gaussianNoise\_samples\_10.y, 'o'), hold on;

% generate regression lines with given order and plot them

for order = [1, 3, 5, 9]

regression\_line = RegressionLine\_generator([0, 1], gaussianNoise\_samples\_10, order);

plot(regression\_line.x, regression\_line.y, '--'), hold on;

clear regression\_line; clear order;

end

% show legend

legend('original signal', 'gaussian noise samples: 10', '1', '3', '5', '9');

Code 3 / Question2.m

And Here is my defined function RegressionLine\_generator.m

function [ regression\_line ] = RegressionLine\_generator( range, sample, order )

%REGRESSIONLINE\_GENERATOR generates regression line with given order from given samples in given range

x = linspace(range(1), range(2));

polynomial = polyfit(sample.x, sample.y, order);

y = polyval(polynomial, x);

regression\_line = table(x, y);

end

Code 4 / RegressionLine\_generator.m

p = polyfit(x,y,n) returns the coefficients for a polynomial p(x) of degree n that is a best fit (in a least-squares sense) for the data in y. The coefficients in p are in descending powers, and the length of p is n+1.

[y](https://kr.mathworks.com/help/matlab/ref/polyval.html?lang=en#d117e1058013) = polyval([p](https://kr.mathworks.com/help/matlab/ref/polyval.html?lang=en" \l "f93-999699_sep_shared-p),[x](https://kr.mathworks.com/help/matlab/ref/polyval.html?lang=en#d117e1057829)) evaluates the polynomial p at each point in x. The argument p is a vector of length n+1 whose elements are the coefficients (in descending powers) of an nth-degree polynomial:

*p*(*x*)=*p*1*xn*+*p*2*xn*−1+...+*pnx*+*pn*+1.

## RESULT

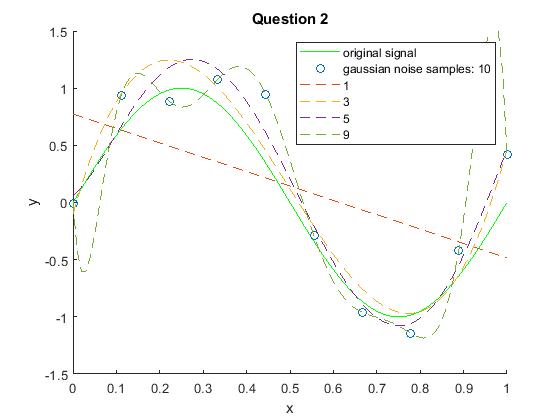


Figure 2 / Result of Question 2

Polynomial with order 1 is underfitted because of too few variables. Order 3 and 5 predict quite similar with original signal. But order 9 seems overfitted because of too many variables.

# Question 3

Plot 100 samples and then generate the regression lines

## CODE

% set figure option

figure(3), ylim([-1.5, 1.5]), title('Question 3'), xlabel('x'), ylabel('y'), hold on;

% plot original signal y = sin(2 \* pi \* x)

plot(linspace(0, 1), sin(2 \* pi \* linspace(0, 1)), 'g'), hold on;

% generate gaussian noise samples with range = [0, 1], number of samples = 100

gaussianNoise\_samples\_100 = GaussianNoise\_generator([0, 1], 100);

% plot gaussian noise samples

plot(gaussianNoise\_samples\_100.x, gaussianNoise\_samples\_100.y, 'o'), hold on;

% generate regression lines with given order and plot them

for order = [1, 3, 5, 9]

regression\_line = RegressionLine\_generator([0, 1], gaussianNoise\_samples\_100, order);

plot(regression\_line.x, regression\_line.y, '--'), hold on;

end

% show legend

legend('original signal', 'gaussian noise samples: 100', '1', '3', '5', '9');

Code 5 / Question3.m

Similar as Question2.m, because I re-used the defined functions. I only changed the number of samples.

## RESULT

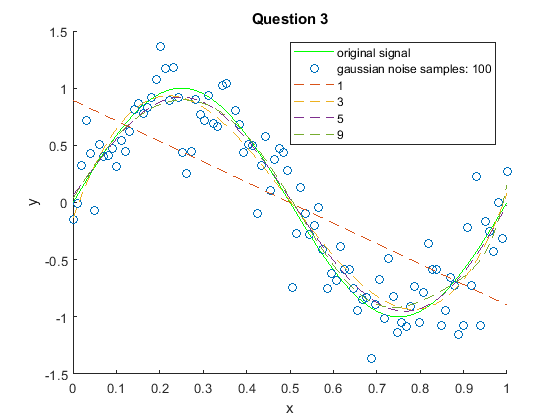


Figure 3 / Result of Question 3

One of way to prevent overfitting in Machine Learning is getting more data. In this Question, we obtained 10x data(100 samples) than previous question so the polynomials whose order are lower than number of samples quite predict well.

# Question 4

Generate the regression lines with a quadratic regularization term with order 1, 3, 5, and 9. Show how the lines are changed with respect to .