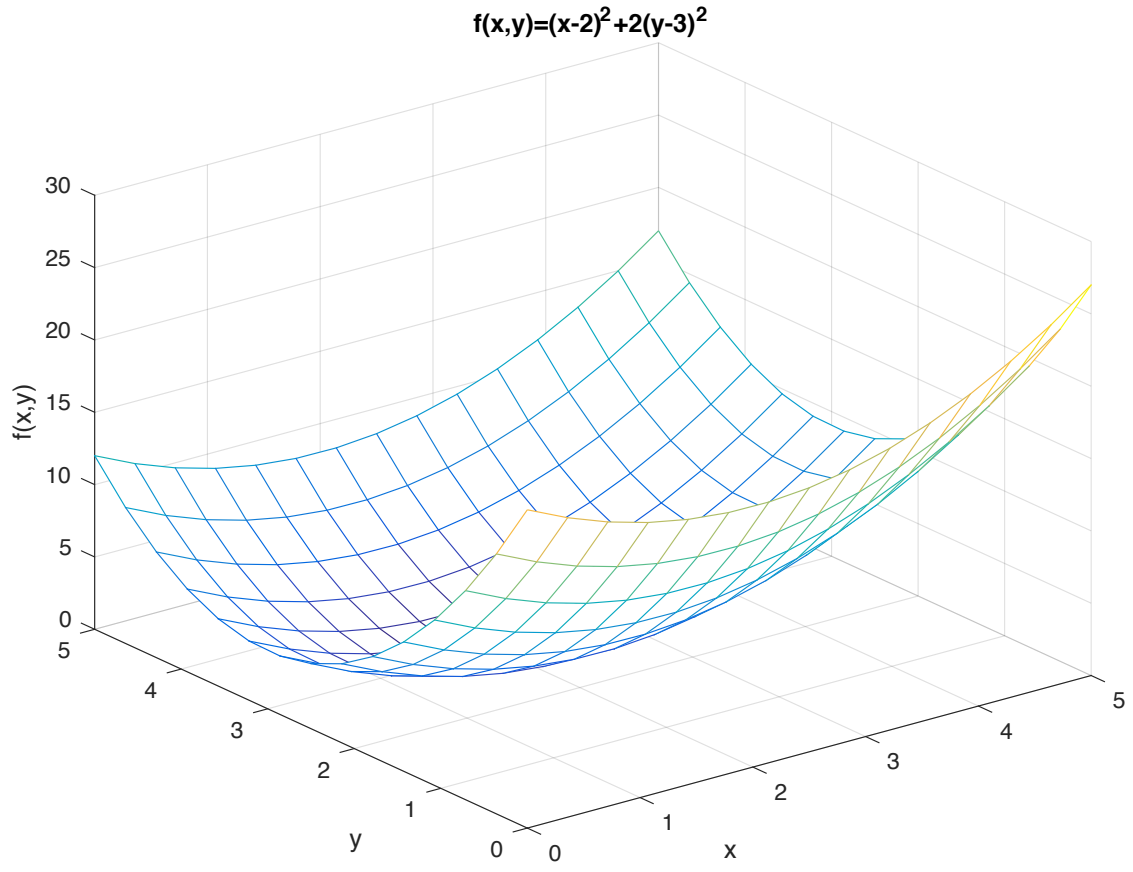
Question 1 Gradient Descent

Output, Commentary and Code **Question 1 Part 1**
Below is an outline of the results and discussion as well as a copy of the code. The same code can also be found in the sections to follow, published and in a more legible format

```
%==========Gradient=Descent==========

%produce a simple visualisation of a gradient descent algorithm. For the
%function f(x,y)=(x-2)^2+2*(y-3)^2

%commence by initialising space for the function in the X and Y dimensions
%corresponding to variables X and Y. The 3rd dimension 'z' will correspond
%to f(x,y).

[X,Y] = meshgrid(linspace(0,5,15),linspace(0,5,15));

%create matrix of f(x,y) call this f
f=((X-2).^2)+(2*(Y-3).^2);

figure;
mesh(X,Y,f) %creates a 3D meshgrid plot of X, Y and f(x,y)
xlabel('x');
ylabel('y');
zlabel('f(x,y)');
title('f(x,y)=(x-2)^2+2(y-3)^2');
grid on;
```

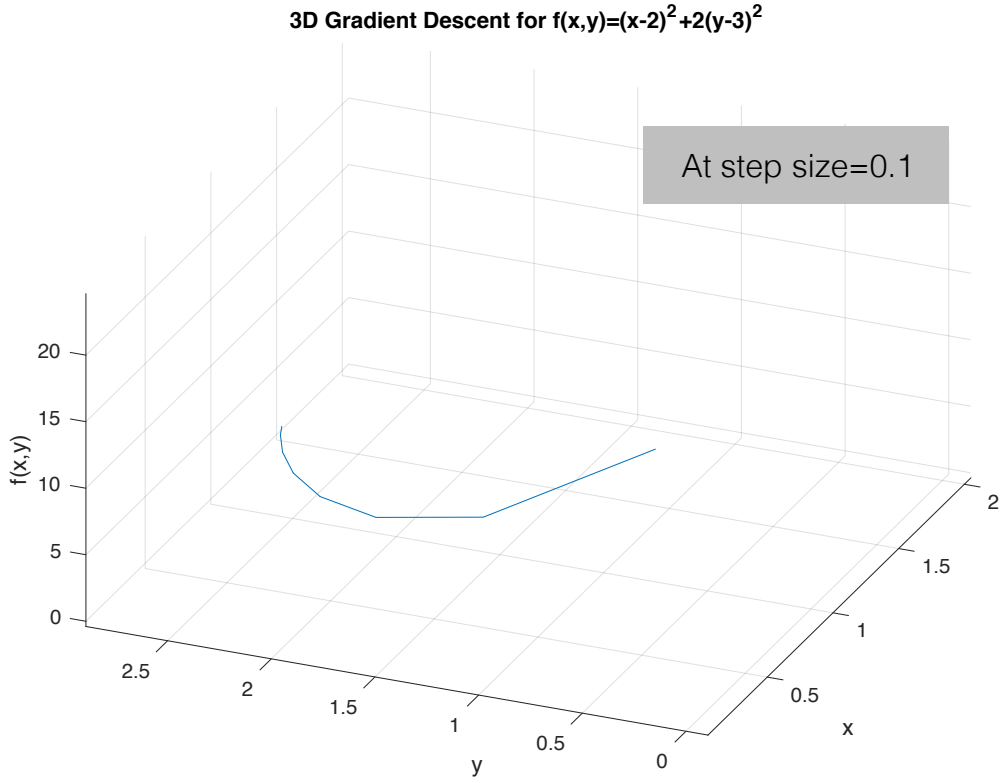


%Find the minimum of function using modified function graddesc.m.

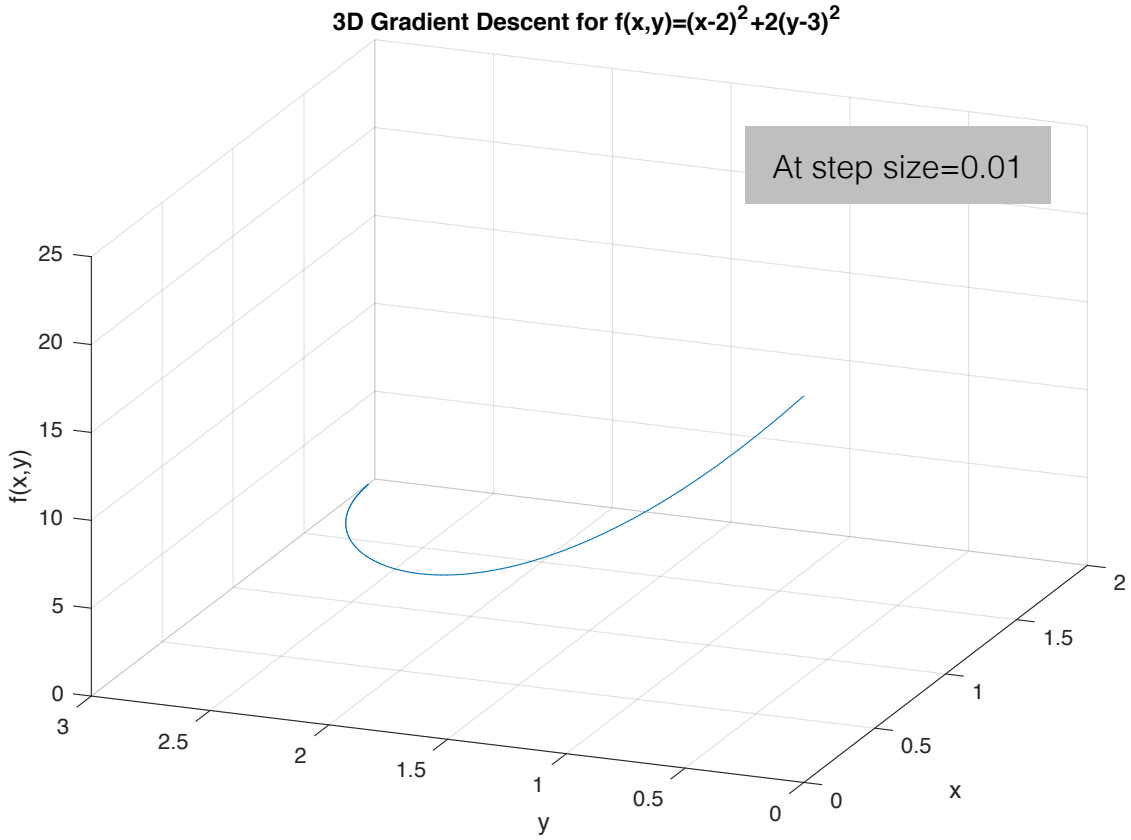
In my version of the graddesc.m function multiple path outputs are available corresponding to different representations of the gradient descent path itself. Path is a cell array where each row corresponds to the x and y coordinates (1st column) and the z coordinate (2nd column), path2 is a vector listing all the points (x,y,z) one after the other, and path3x,path3y,path3z effectively store only the x,y and z component of each point traversed by the gradient descent algorithm. Path3x,path3y,path3z are used for plotting. Path 2 is used to provide desired printed output.

```
[min,path,path2,path3x,path3y,path3z]=graddesc(@fc,@dfc,[0,0],0.1,0.1)

figure;
plot3(path3x,path3y,path3z)
xlabel('x');
ylabel('y');
zlabel('f(x,y)');
title('3D Gradient Descent for f(x,y)=(x-2)^2+2(y-3)^2');
view(-37.5,40);
grid on ;

%recreates path almost the same as the one printed in Homework sheet
j=1;
while j<size(path2,2)
fprintf('(%.1f,%.1f,%.1f)',path2(1,j),path2(1,j+1),path2(1,j+2))
j=j+3;
end
```

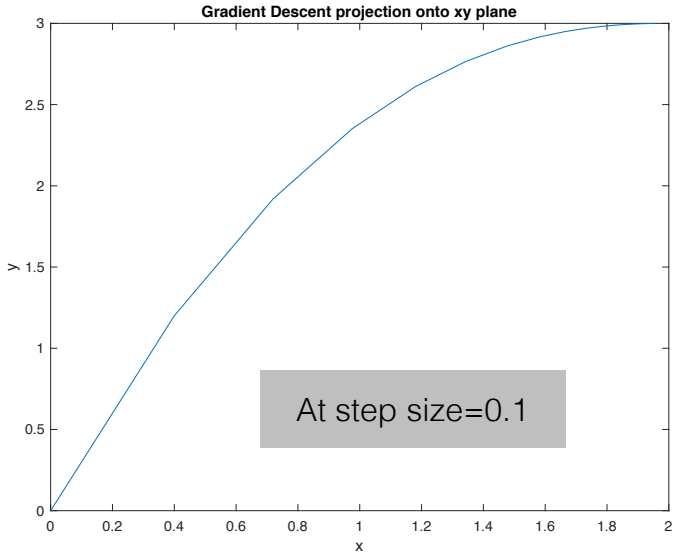
Output:
(0.0,0.0,22.0)(0.4,1.2,9.0)(0.7,1.9,4.0)(1.0,2.4,1.9)(1.2,2.6,1.0)(1.3,2.8,0.5)(1.5,2.9,0.3)(1.6,2.9,0.2)
(1.7,2.9,0.1)(1.7,3.0,0.1)(1.8,3.0,0.0)(1.8,3.0,0.0)(1.9,3.0,0.0)(1.9,3.0,0.0)(1.9,3.0,0.0)(1.9,3.0,0.0)
(1.9,3.0,0.0)(2.0,3.0,0.0)

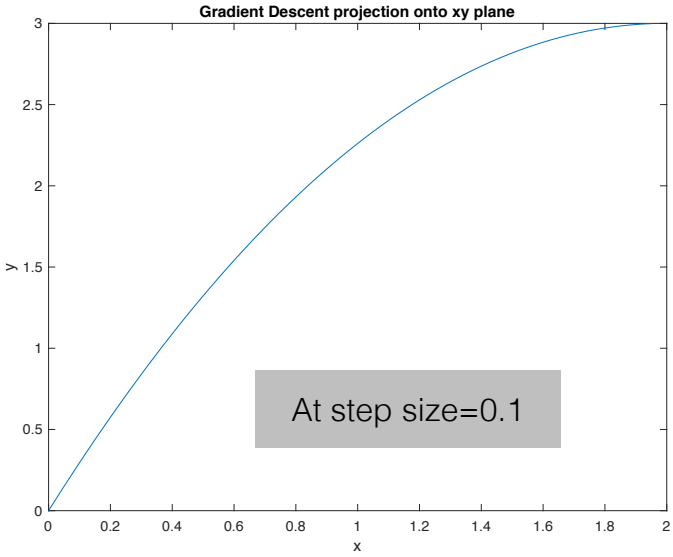


To make smoother can run function with different step size, like shown on figures below. However this will also have a substantial impact on the amount of ‘steps’ the gradient descent has to make before converging to a solution. The correspond path is therefore much longer for step size=0.01.




```
figure;
plot(path3x,path3y)
xlabel('x');
ylabel('y');
title('Gradient Descent projection onto xy plane')
```





Output, Commentary and Code **Question 1 Part 2**

%==========Least Squares Error estimation by Gradient=Descent==========

%======PART=A ====
function written called myLSE.m which computes the least squares solution by gradient descent. Code for this function can be found in Appendix: Functions.

function takes inputs defined below and computes Least Square error solution by gradient descent. Solution is that combination of X's which minimises sum of error squared, where error is defined as: e=Ax-b

%------INPUTS-----
%A is coefficients matrix of size m*n
%b is the corresponding solutions vector of size m*1
%guess is a vector of initial guess of values of x (size m by 1)
%step is the step size of the gradient descent
%tol is the tolerance and dictates when function will stop iterating
%i.e. what difference between predicted and observed values are we satisfied with?

%=======PART=B ====
test myLSE.m by solving system of equations provided. The system can be represented in matrix form as:
A=[1 -1; 1 1; 1 2];
b=[1;1;3];

setting the remaining inputs for myLSE:
guess=[1;1];
step=0.05;
tol=0.01;

note: both the tolerance and the step size needed reduction from original assumptions to converge at solution to 2 decimal points. Achieving larger precision would require many more iterations, thus is computational expensive.

[x,path]=myLSE(A,b,guess,step,tol); ====> yields ====>

SOLUTION TO Part B is

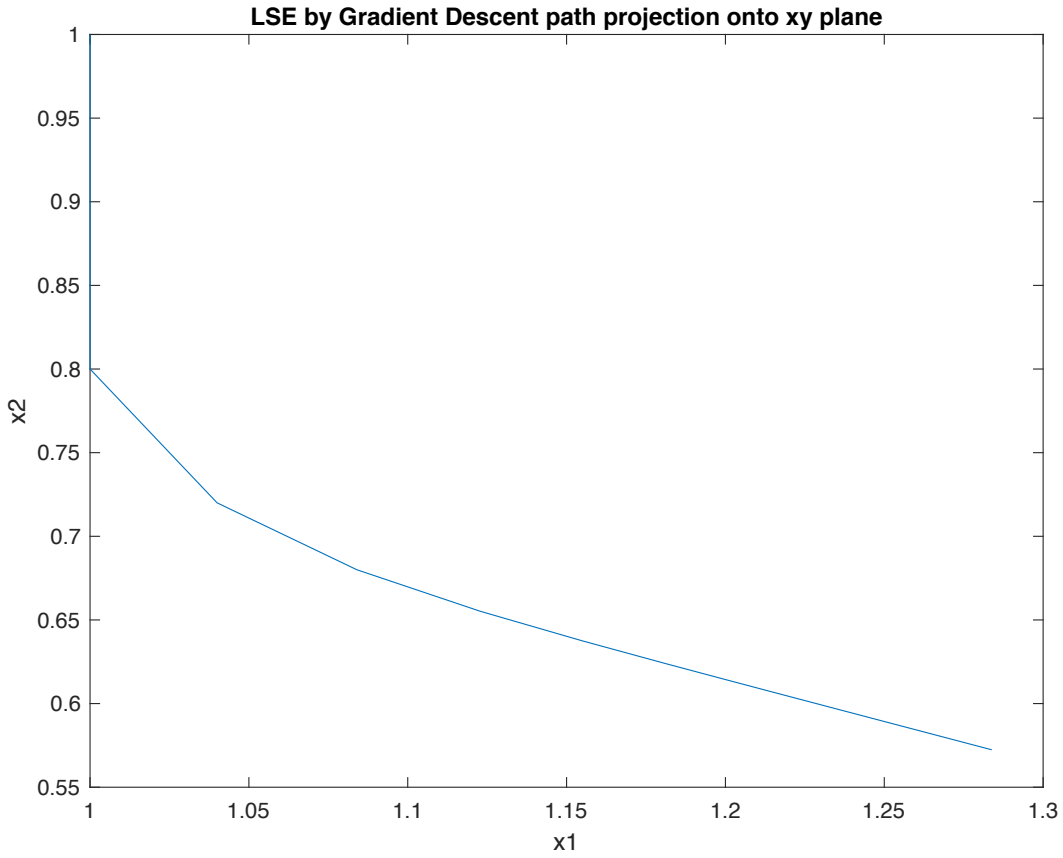
x1=1.28
x2=0.57

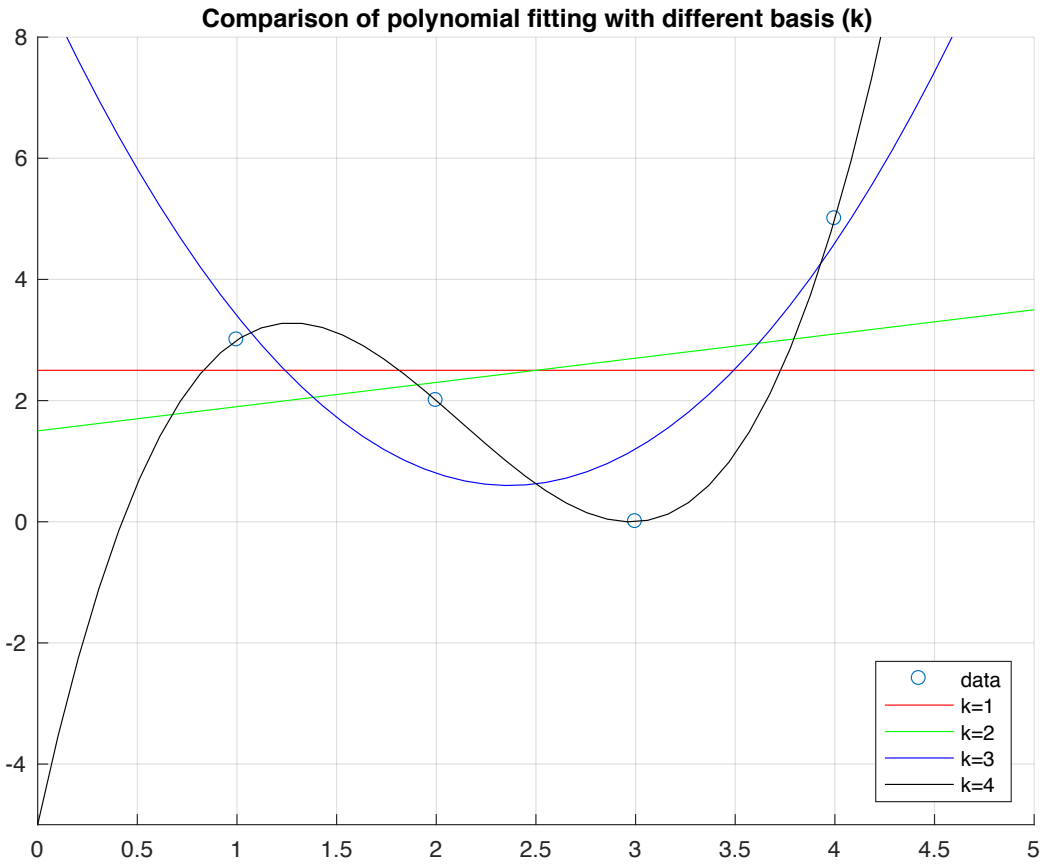
%Additional test
Aa=[1 -1 2; 1 2 1; 1 2 2];
bb=[1;1;3];
%setting the remaining inputs for myLSE:

guessg=[-2;0;1];
tol2=0.05; %needs decreasing so that solution converses

[x2,path2]=myLSE(Aa,bb,guessg,step,tol2);
%solution produced is equivalent to matlab Aa\bb therefore correct code

%=======PART=C==== PLOT solution
figure;
plot(path(:,1),path(:,2))
xlabel('x1');
ylabel('x2);
title('LSE by Gradient Descent path projection onto xy plane');


LSE by Gradient Descent path projection onto xy plane
x2
x1
1
0.95
0.9
0.85
0.8
0.75
0.7
0.65
0.6
0.55
1
1.05
1.1
1.15
1.2
1.25
1.3


Comparison of polynomial fitting with different basis (k)
8
6
4
2
0
-2
-4
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
data
k=1
k=2
k=3
k=4

Question 2 Linear Regression

Output and Commentary for **Question 2 Part 1**

Below is an outline of the results and discussion. detailed and exhaustively commented code can be found in the sections to follow.

PART A
The relevant plot is on page above

Part B
Equations of polynomials:

k=1: P1=2.5
k=2: P2=1.5+0.4x
k=3: P3=9.0-7.1x+1.5(x^2)
k=4: P4=-5+15.17-8.5(x^2)+1.33(x^3)

PART C
MSE is the Mean Squared Error denied as MSE=SSE/m where SSE is the sum of squared errors and m is the amount of observations (definition of the mean).

MSE for k=1 is 3.250
MSE for k=2 is 3.050
MSE for k=3 is 0.800
MSE for k=4 is 3.2541e-30

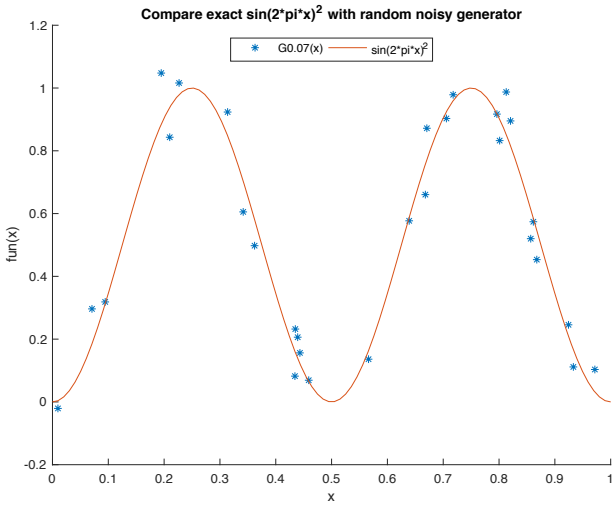
It can be seen that error decreases with degree of polynomial and is lowest for k=4. Having examined what happens for higher order polynomial fits (k=6 and k=9) I observed that MSE is minimum at k=4 and increases very slightly thereafter, however it error remains low. This is due to overfitting a phenomenon explored in more depth in question 2.

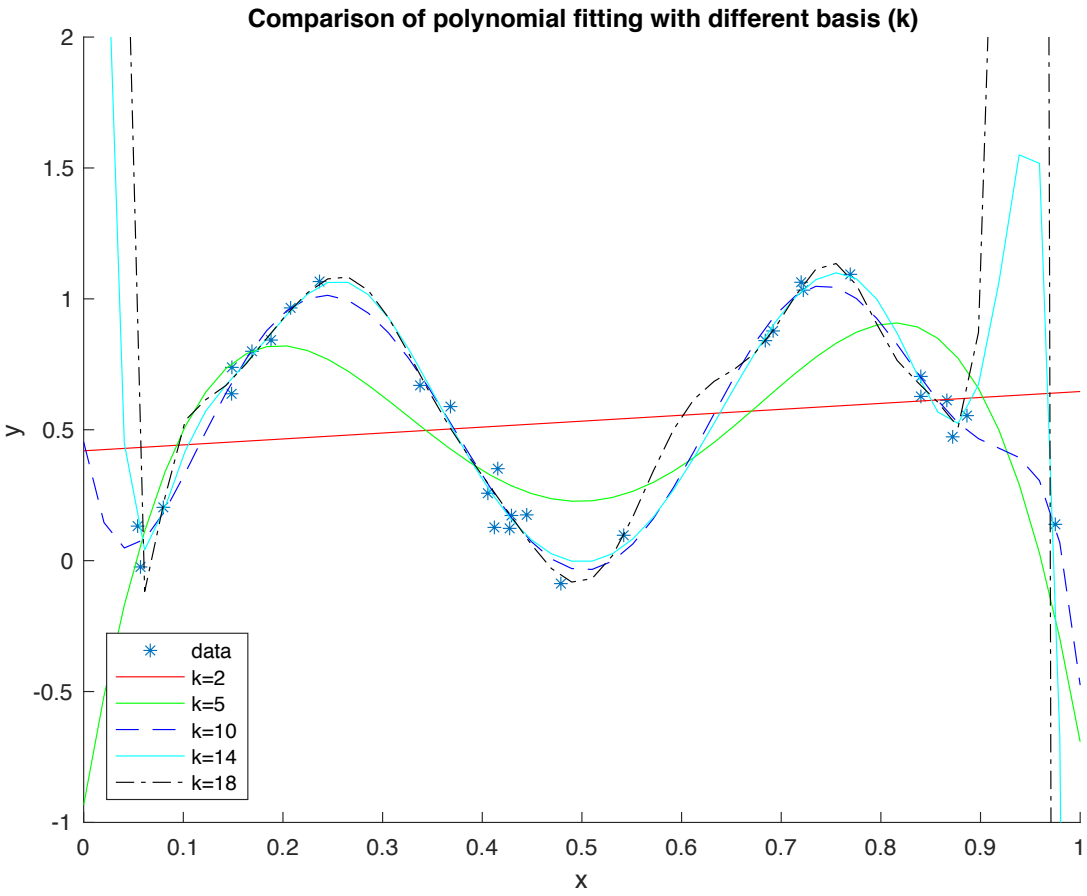
Output for **Question 2 Part 2**

PART A
Function MYrandom.m was defined, for working refer to Appendix called Functions.

PART B

This figure illustrates the exact function sin(2*pi*x) generated over the interval x from 0 to 1.
This is plotted alongside a noisy data sample generated by the function GofX. This noisy sample takes an offset from the exact sin(2*pi*x) value, the offset is generated sampling at random from a gaussian distribution with mean zero and standard deviation 0.07



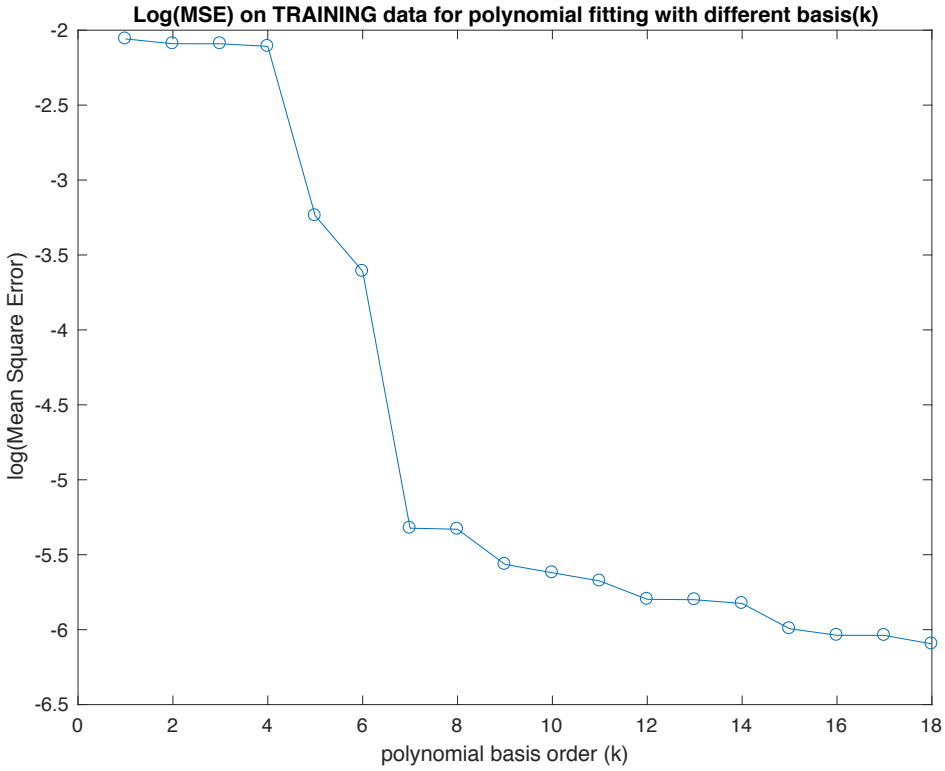


Function GofX.m was defined, for working refer to Appendix called Functions.

Figures Obtained:
In the above figure k is the order of the polynomial fitted to the 30 datapoint generated by taking a random, gaussian distributed ,deviation from the exact value of sin(2pi*x).
From the figure above, it can be seen that for low values of k (up to about k=10) as the degree of the polynomial increases the fit becomes closer to the exact value of the SIN function. However, when polynomials of higher order are fitted the phenomenon of 'overfitting; can be observed, where curves follow the data points exactly with loss of generality. Overfilled polynomials no longer resemble the SIN distribution, which gave rise to the data and are thus bad predictors of the original distribution or of the general physical situation being described

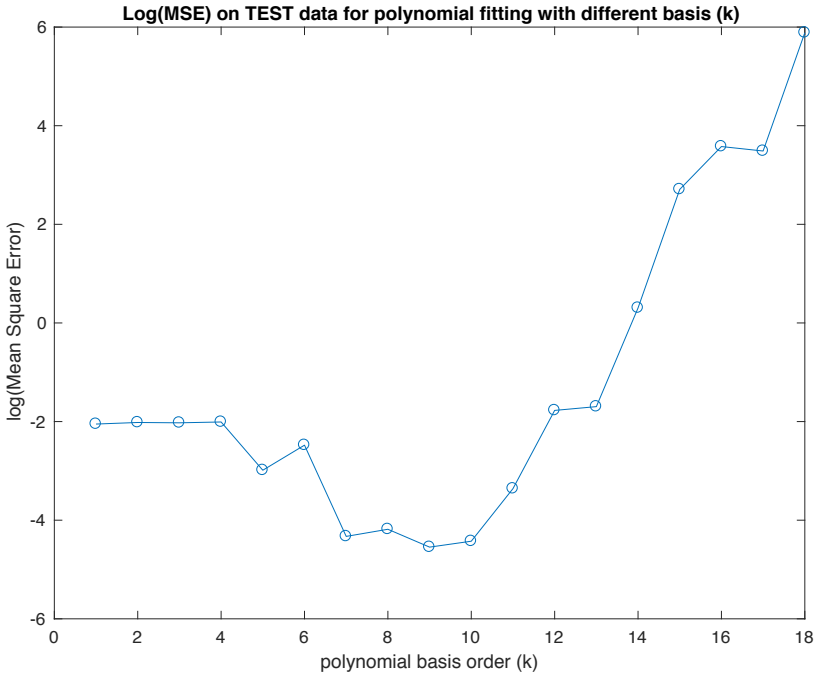
PART C

In this section log(mean squared error) was found for the training set generated thus far (30 points). This was found using he property: MSE=SSE/m (defied earlier).
The log of the training error was plotted versus the fitted polynomials of dimensions range: k=1:18
The graph below illustrates the results; for low k values the MSE drops sharply and remains approximately constant thereafter. This is to be expected as overfitting occurs and higher order polynomials fit to the noise, producing deceptively low MSE's.



PART D

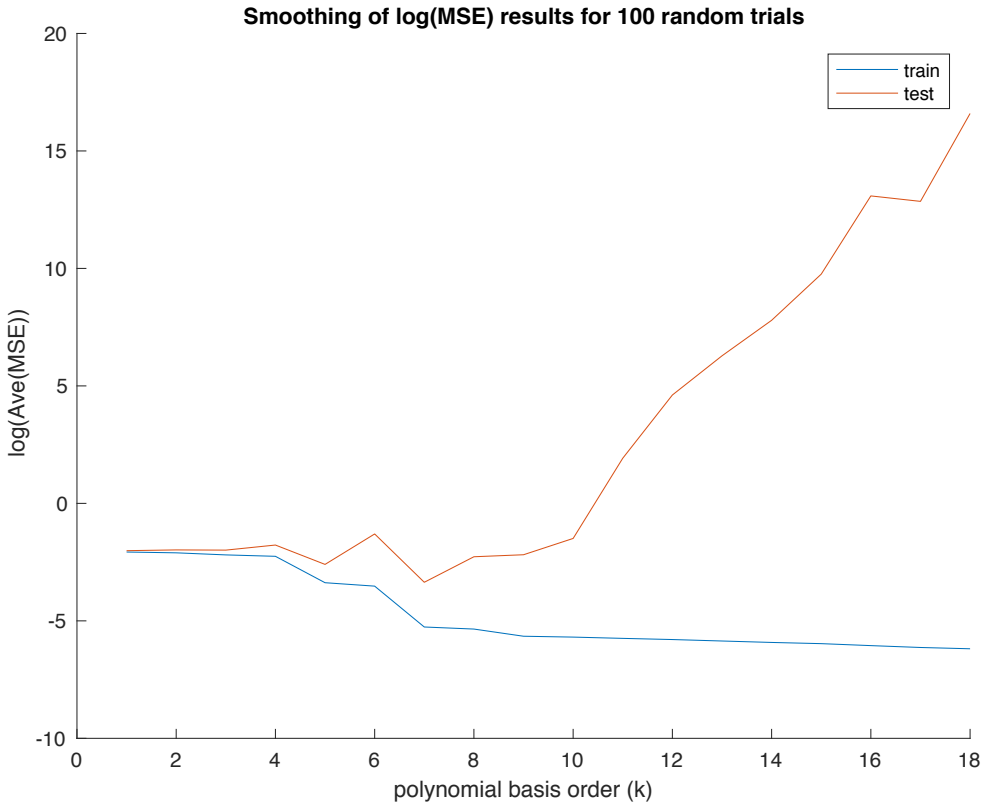
This part is similar to Part C but now log(mean squared error) is found for a newly generated test data, of much larger size (1000). This test data is still evaluated using the model trained on the 30 data points in the earlier sections of this assignment. The results are plotted against the degree of polynomial fitted to the data and the following graph is obtained:



Unsurprisingly, the newly found log(MSE) plot is not a deceasing function. The error on test data increases for high order polynomial fits, since rather than fitting to the 'true' data distribution (here the SIN function) we are fitting to noise.

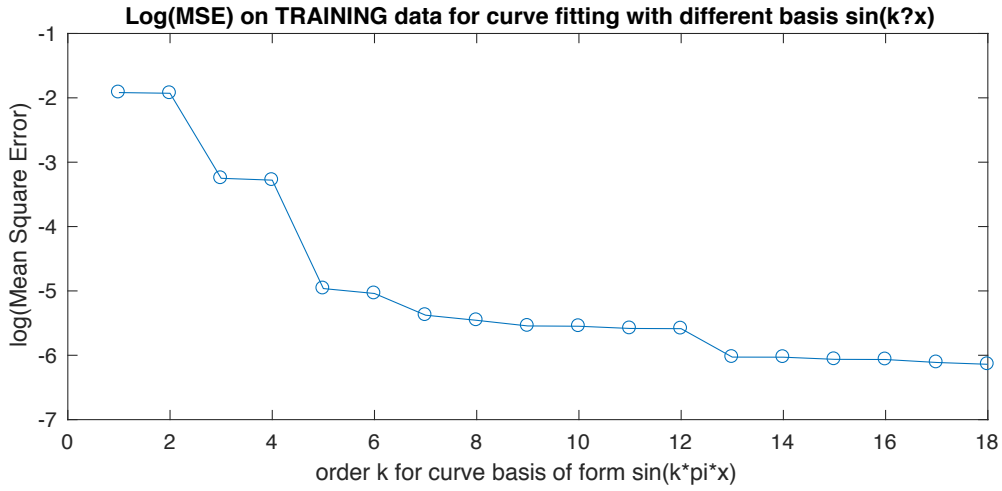
PART E

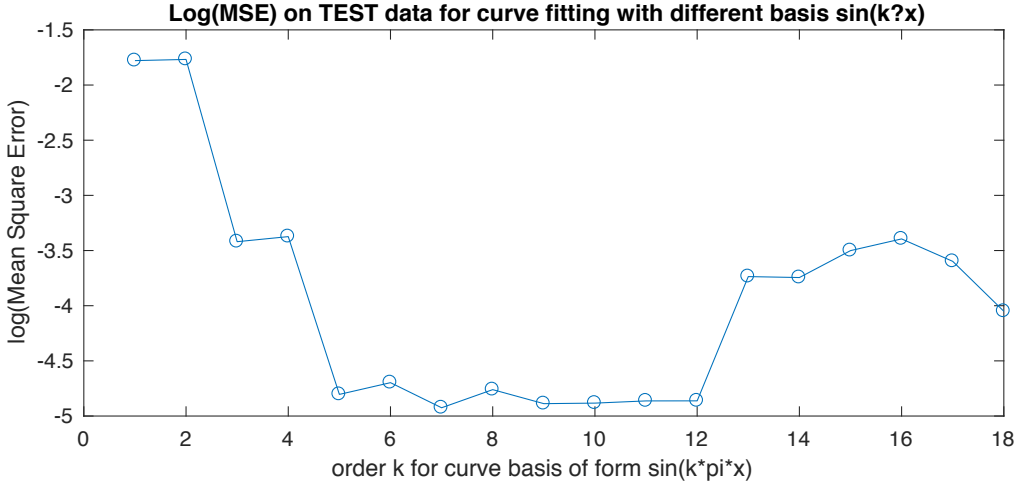
In previous sections different sets of random data were generated at every instance of running the program. Thus, results varied depending on the particular run of the program. To check that the above discoveries hold regardless of the instance of the data generated, and to provide a 'smoothed' average log(MSE), the experiments form part C and D were repeated 100 times. This yelled the smoothed graph below which is a good approximation for the average expected log(MSE) for any instance of data generated in using the same process (defined isn part A and B).

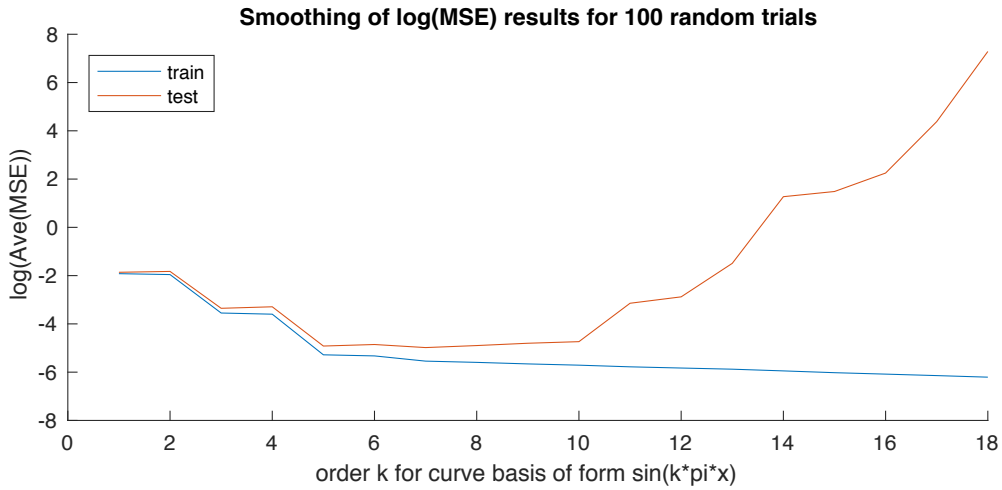


Output and Commentary for **Question 2 Part 3**

In this part Experiments form Section 2 part C to E will repeated using different basis given by: $\sin(1\pi x)$, $\sin(2\pi x)$, $\sin(3\pi x)$, . . . , $\sin(k\pi x)$,where k ranges from 1 to equal 18. In order to do this a new function needs to be defined which returns the coefficients of this fit instead of the previously used polynomial fit. This function is called MySIN2.m and can be found in the Appendix: Functions. The 3 graphs obtained are presented on the next page. similar phenomena as in part two of their question can be observed; log(MSE) decreases with k for the training set and for the test set it is increasing sharply for high values of k. In all instances the log(MSE) is a positive value, as opposed to negatives obtained when fitting polynomials. This signifies that the MSE is significantly higher when fitting using SIN functions as base and therefore polynomial base functions are preferable to obtain a good model.


Log(MSE) on TRAINING data for curve fitting with different basis sin(k?x)
log(Mean Square Error)
order k for curve basis of form sin(k*pi*x)


Log(MSE) on TEST data for curve fitting with different basis sin(k?x)
log(Mean Square Error)
order k for curve basis of form sin(k*pi*x)


Smoothing of log(MSE) results for 100 random trials
train
test
log(Ave(MSE))
order k for curve basis of form sin(k*pi*x)