Foundation of Machine Learning

Practical 1

En.Roll.: 202211063

Name:Pranshu Parate

Q1Generate 20 real number for the variable X from the uniform distribution U [0,1]

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import *

x_train = np.random.uniform(0,1,20)
x_train = np.sort(x_train)
print(x_train)
```

```
[0.06008884 0.08469982 0.09224183 0.1143787  0.17328569 0.22441962
 0.28434507 0.36774859 0.37002487 0.46717926 0.49637395 0.51587294
 0.52619491 0.60897459 0.66443722 0.68845234 0.73735591 0.875593
 0.9536765  0.97859761]
```

Q2Construct the training set T = { (x 1 ,y 1),(x 2 ,y 2),.....,(x 20 ,y 20)} using the relation

Yi = sin(2 π x i) + ϵ i where ϵ i ~ N(0,0.25)

In [2]:
```
#Calculating mean
import statistics
x_mean = statistics.mean(x_train)
V = np.zeros(len(x_train))
for i in range(len(x_train)):
  V[i] = x_train[i]- x_mean

V = np.square(V)
#Standard Deviation
Std_d = np.sum(V) / len(V)
print(Std_d)
print(x_mean)
```

```
0.08138255743359009
0.4641970629027494
```

In [3]:
```
#Calculating Y function
def epsilon(std,mean,x):
  return (np.pi*std) * np.exp(-0.5*((x-mean)/std)**2)
y_train = np.zeros(len(x_train))
for i in range (len(y_train)):
  y_train[i] = np.sin(2*np.pi*x_train[i])+epsilon(0.25,0,x_train[i])
print(y_train)
```

```
[ 1.13167968  1.2490084   1.28138256  1.36573237  1.5037429   1.5120479
  1.38812713  1.00478718  0.99151721  0.34178194  0.13219027 -0.00613931
 -0.07812062 -0.59202382 -0.83596234 -0.90843703 -0.98670434 -0.7027634
 -0.28642354 -0.13370056]
```

Q3In the similar way construct a testing set of size 50 I,e. Test = { (x' 1 ,y' 1),(x' 2 ,y' 2),.....,(x' 50 ,y' 50)}

In [4]:
```
#Generating Testing Set
```

```
x_test = np.random.uniform(0.0,1.0,50)
x_test = np.sort(x_test)
y_test = np.zeros(len(x_test))
for i in range (len(x_test)):
  y_test[i] = np.sin(2 * np.pi * x_test[i]) + epsilon(0.25,0,x_test[i])
print(y_test)
```

```
[ 0.80272753  1.18585687  1.30240455  1.48621433  1.49900044  1.52127188
  1.51455558  1.51163104  1.50028895  1.49605207  1.48046053  1.46449017
  1.44713922  1.40798958  1.39910097  1.36744006  1.27283885  1.14072053
  1.097443    0.97585046  0.97576482  0.8679113   0.85272431  0.85243729
  0.45823451  0.4226598  -0.04014326 -0.1338921  -0.45499047 -0.79070221
 -0.8077641  -0.89478533 -0.89725472 -0.95110556 -0.96107641 -0.96688938
 -0.97367525 -0.99137854 -0.98665591 -0.97300448 -0.97110598 -0.96525457
 -0.89354012 -0.88452923 -0.82345078 -0.79734379 -0.52391352 -0.44758183
 -0.15059441 -0.00859404]
```

Q4Estimate the Least Square polynomial regression model of order M= 1,2, 3, 9, using the training set T. For example for M=1 , we need to estimate F(x) = β 1 x + β 0 For M = 2 F(x) = β 2 x 2 + β 1 x + β 0 .

In [9]:
```
#Polynomial Regression Coefficient
def find_coef(p,q,r):
  matrix_p =  np.zeros((r+1,r+1))
  matrix_q = np.zeros(r+1)

#Calculating A matrix
  for i in range (r+1):
    for j in range (r+1):
      if (i==0 and j==0):
        matrix_p[i][j]= len(p)
      else:
        matrix_p[i][j]=np.sum(p**(i+j))
      #Calculating y vector
  for i in range (r+1):
        matrix_q[i]=np.sum(q*(p**i))
      #do transpose of y
  matrix_q=np.matrix.transpose(matrix_q)

      #do x^-1 *Y^T
  p = np.linalg.inv(matrix_p).dot(matrix_q)
  return p
```

Q5List the value of coefficients of estimated polynomial regression models for each case.

In [10]:
```
#save the coefficient
#c=1
c1_coef=find_coef(x_train,y_train,1)

#c=2
c2_coef=find_coef(x_train,y_train,2)

#c=3
c3_coef=find_coef(x_train,y_train,3)

#m=9
c9_coef=find_coef(x_train,y_train,9)
print(c1_coef)
print(c2_coef)
print(c3_coef)
print(c9_coef)
```

```
[ 1.60917427 -2.6725463 ]
[ 1.9907895  -4.98164836  2.32520295]
[  0.58030691  10.89726529 -35.92364333  24.67347697]
[   0.78757589    6.19203435   -4.82627892  -53.12896347   78.68997192
  -58.45770264  130.09179688 -179.86126709   99.47320557  -18.96020508]
```

Q6Obtain the prediction on testing set and compute the RMSE for polynomial regression models for order M =1,2,3 and 9 .

In [12]:
```python
#RMSE Polynomial Regression
def rms(m,n,o):
  n_predict = np.zeros(len(m))
  c = 0
  for i in range(len(m)):
    n_predict[i] +=o[0]
    for j in range(len(o)-1):
      n_predict[i]+=o[j+1]*m[i]**j
    c+=(n_predict[i]-n[i])**2
    c=c**0.5/len(m)
  return [c,n_predict]
```

In [13]:
```python
#Testing of RMSE Polynomial
testing_1 = rms(x_test,y_test,c1_coef)
testing_2 = rms(x_test,y_test,c2_coef)
testing_3 = rms(x_test,y_test,c3_coef)
testing_9 = rms(x_test,y_test,c9_coef)
training_1 = rms(x_train,y_train,c1_coef)
training_2 = rms(x_train,y_train,c2_coef)
training_3 = rms(x_train,y_train,c3_coef)
training_9 = rms(x_train,y_train,c9_coef)
print(testing_1)
print(testing_2)
print(testing_3)
print(testing_9)
print(training_1)
print(training_2)
print(training_3)
print(training_9)
```

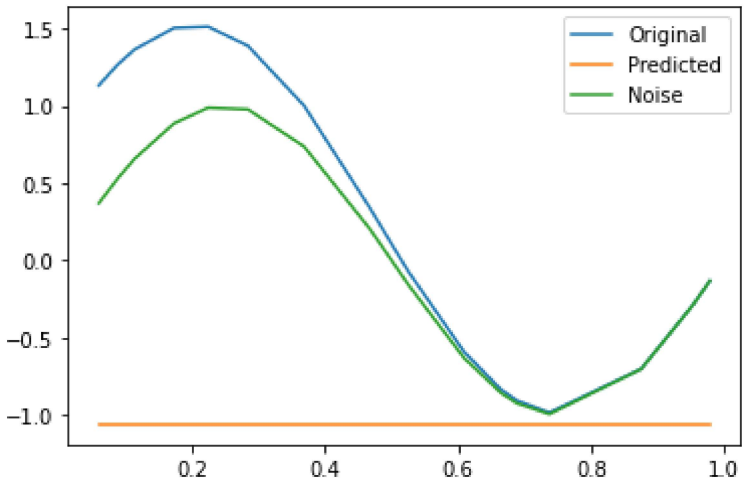
```
[0.021269212585097234, array([-1.06337203, -1.06337203, -1.06337203, -1.06337203,
-1.06337203,
       -1.06337203, -1.06337203, -1.06337203, -1.06337203, -1.06337203,
       -1.06337203, -1.06337203, -1.06337203, -1.06337203, -1.06337203,
       -1.06337203, -1.06337203, -1.06337203, -1.06337203, -1.06337203,
       -1.06337203, -1.06337203, -1.06337203, -1.06337203, -1.06337203,
       -1.06337203, -1.06337203, -1.06337203, -1.06337203, -1.06337203,
       -1.06337203, -1.06337203, -1.06337203, -1.06337203, -1.06337203,
       -1.06337203, -1.06337203, -1.06337203, -1.06337203, -1.06337203])]
[0.013380784098250293, array([-2.98442772, -2.82574721, -2.76443246, -2.61618792,
-2.5965859 ,
       -2.51076966, -2.47617444, -2.46794724, -2.44410183, -2.4369365 ,
       -2.41466854, -2.39589313, -2.37823069, -2.34460373, -2.33779225,
       -2.31525309, -2.25841303, -2.19340875, -2.17428637, -2.12428934,
       -2.1242557 , -2.08324342, -2.07765862, -2.07755347, -1.94248254,
       -1.93085755, -1.78005577, -1.74847857, -1.63188851, -1.47449713,
       -1.46411014, -1.40198753, -1.39989476, -1.34466132, -1.33088663,
       -1.32179463, -1.30969044, -1.23805842, -1.20531423, -1.17069246,
       -1.16714031, -1.15711767, -1.07754084, -1.07002832, -1.02537589,
       -1.0086257 , -0.87016528, -0.8377303 , -0.72174325, -0.66893605])]
[0.004427033763604793, array([11.37840179,  9.05106235,  8.21332794,  6.32965774,
6.09559952,
        5.11219411,  4.73476302,  4.64661273,  4.39461138,  4.31990122,
        4.09071304,  3.90098794,  3.72544632,  3.3991092 ,  3.33426343,
        3.12270777,  2.60978986,  2.05934457,  1.904761  ,  1.51636031,
        1.51610665,  1.21455002,  1.17467354,  1.17392547,  0.29632529,
        0.22857637, -0.53849386, -0.67283293, -1.09000713, -1.45638018,
       -1.47260578, -1.54909131, -1.55105463, -1.58842194, -1.59340298,
       -1.59574192, -1.59768491, -1.58181001, -1.5589562 , -1.5241481 ,
       -1.51995801, -1.50751447, -1.37617769, -1.36079294, -1.25872005,
       -1.21573638, -0.76235037, -0.63084657, -0.08202815,  0.20851879])]
[0.0006513529010557313, array([ 6.96585674,  6.39590476,  6.07422563,  5.12353404,
4.98332938,
        4.34099726,  4.07174014,  4.00705193,  3.81838047,  3.76138399,
        3.58352522,  3.43287729,  3.29076113,  3.01967303,  2.96475083,
        2.7831731 ,  2.32805234,  1.81731153,  1.66997735,  1.29282686,
        1.29257746,  0.99345193,  0.95352284,  0.95277299,  0.05713074,
       -0.01290024, -0.80198847, -0.9370387 , -1.33844056, -1.63254222,
       -1.64206535, -1.67444813, -1.67481984, -1.66830035, -1.66189773,
       -1.65665932, -1.64845535, -1.5724321 , -1.52301872, -1.46164015,
       -1.45483949, -1.43516539, -1.25556494, -1.23666482, -1.11841099,
       -1.07167755, -0.65051661, -0.54644995, -0.17401238, -0.00982952])]
[0.047532250723777245, array([-1.06337203, -1.06337203, -1.06337203, -1.06337203,
-1.06337203,
       -1.06337203, -1.06337203, -1.06337203, -1.06337203, -1.06337203,
       -1.06337203, -1.06337203, -1.06337203, -1.06337203, -1.06337203,
       -1.06337203, -1.06337203, -1.06337203, -1.06337203, -1.06337203])]
[0.030143897633988004, array([-2.85114011, -2.79391459, -2.77637788, -2.72490516,
-2.58793447,
       -2.46903769, -2.32969886, -2.13576876, -2.13047594, -1.90457226,
       -1.83668869, -1.79134958, -1.76734889, -1.57486936, -1.44590749,
       -1.39006745, -1.27635672, -0.95492743, -0.77336746, -0.71542082])]
[0.006089326963093526, array([ 9.40804988,  8.61185511,  8.37384514,  7.69146288,
5.99341236,
        4.6582608 ,  3.25776416,  1.60352005,  1.56318382,  0.07993682,
       -0.27476213, -0.48823694, -0.59364708, -1.24885354, -1.49866536,
       -1.55973952, -1.5961236 , -1.06067404, -0.34146295, -0.04858314])]
[0.004146820263022154, array([ 6.51417791,  6.23498536,  6.14066552,  5.84243981,
4.92058666,
        4.01563875,  2.89951368,  1.37829293,  1.33879815, -0.16666182,
       -0.53280078, -0.75104062, -0.85765152, -1.47783741, -1.65588535,
       -1.67595185, -1.61878667, -0.91461582, -0.33880935, -0.15407295])]
```

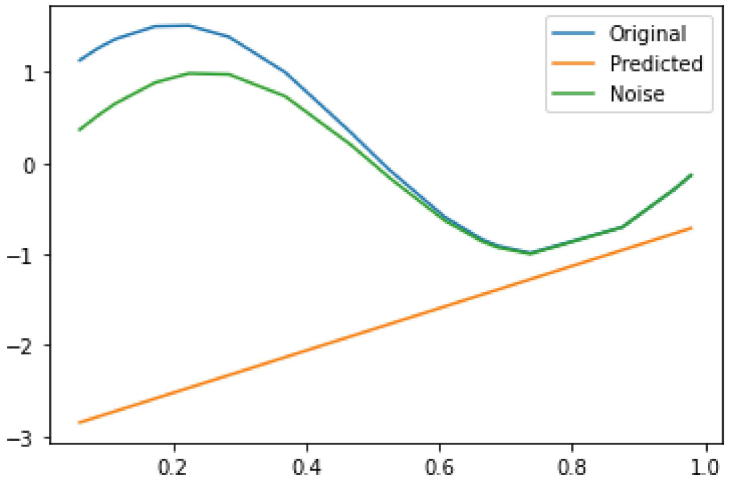
Q7Plot the estimate obtained by polynomial regression models for order M =1,2,3 and 9 for training set along with y 1, y 2, , y 50. . Also plot our actual mean estimate E(Y/X) = sin(2 π x i) .

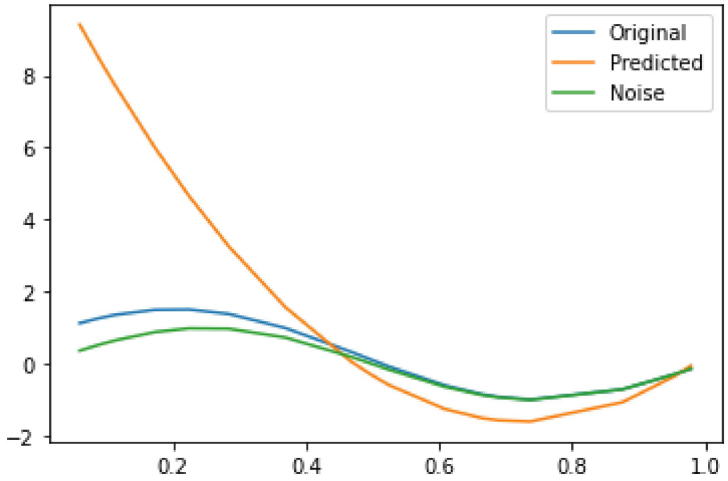
In [18]:
```
#Plotting Graph for Actual Mean Estimate Polynomial Regression
def ploting(x,y1,y2,y3):
  plt.plot(x,y1,label = "Original")
  plt.plot(x,y2,label = "Predicted")
  plt.plot(x,y3,label = "Noise")
  plt.legend()
  plt.show()
```

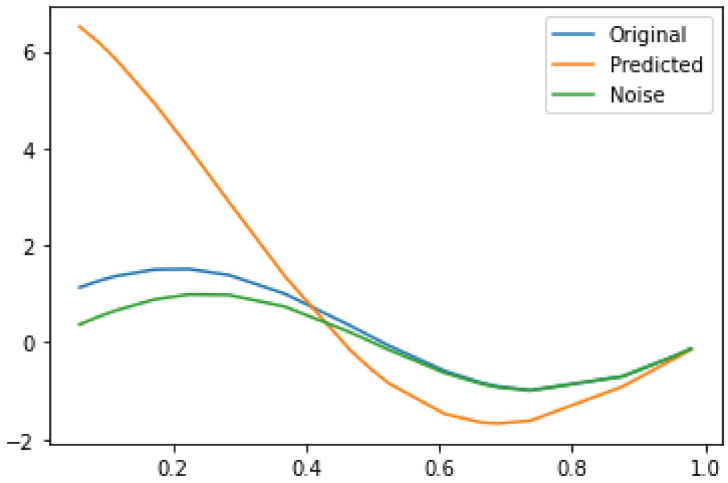
In [19]:
```
train_error = np.zeros(len(y_train))
y_test_er = np.zeros(len(y_test))
for i in range(len(y_train)):
    train_error[i] =y_train[i] - epsilon(0.25,0,x_train[i])

for i in range(len(y_test)):
    y_test_er[i] = y_test[i] - epsilon(0.25,0,x_test[i])
```

In [20]:
```
ploting(x_train,y_train,training_1[1],train_error)
ploting(x_train,y_train,training_2[1],train_error)
ploting(x_train,y_train,training_3[1],train_error)
ploting(x_train,y_train,training_9[1],train_error)
```



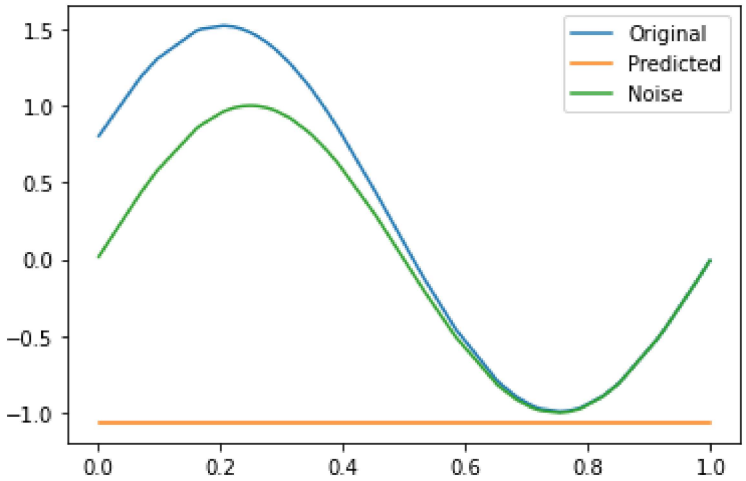


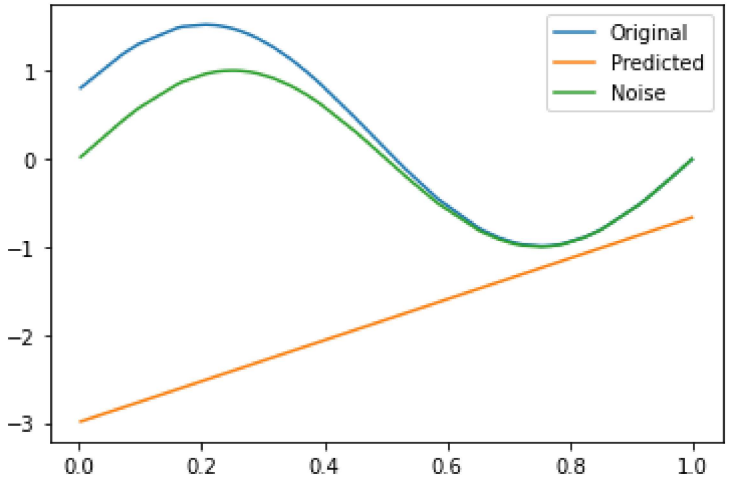


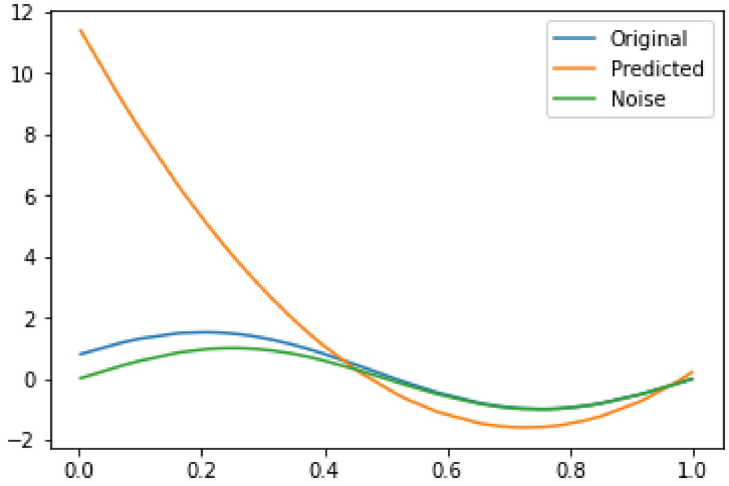


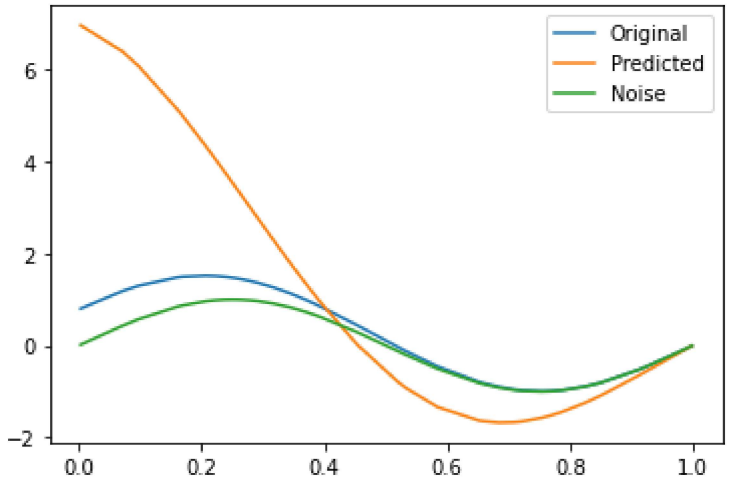
Q8Plot the estimate obtained by polynomial regression models for order M =1,2,3 and 9 for testing set along with y′ 1, y′ 2, , y′ 50. . Also plot the sin(2 π x′ i) .

In [22]:
```
#Plotting Graph for Estimated Mean of Polynomial Regression
ploting(x_test,y_test,testing_1[1],y_test_er)
ploting(x_test,y_test,testing_2[1],y_test_er)
ploting(x_test,y_test,testing_3[1],y_test_er)
ploting(x_test,y_test,testing_9[1],y_test_er)
```









Q9What happens when we increase the value of M. Note down your observations.

As M increases as it will give the best fitting to training.But it will give not proper fitting on testing set. So it is overfitting.

Q10Also ,try to find the statistical reasons behind your observation.

By increasing the degree of polynomial will improve on training set. So, it will give overfitting model, while it will be problem for testing set as it can't generate the general model.