IE406 Machine Learning - Assignment 2

202201140* | *Harsh Gajjar

Importing libraries

In [63]:
```
import numpy as np
import matplotlib.pyplot as plt
```

(a) Generate 20 real number for the variable X from the uniform distribution U [0,1]

In [64]:
```
X = np.random.uniform(0, 1, 20)
X.sort()
```

(b) Construct the training set T = {(x1,y1),(x2,y2),.......,(x20,y20)} using the relation

1. Yi = sin(2 π xi) + εi where εi ~ N(0,0.25)

In [65]:
```
epsilon = np.random.normal(0, np.sqrt(0.25), 20)
Y = np.sin(2*np.pi*X) + epsilon
training_set = list(zip(X, Y))
```

(c) In the similar way construct a testing set of size 50

a. i,e. Test = {(x′1,y′1),(x′2,y′2),.......,(x′50,y′50)}

In [66]:
```
X_test = np.random.uniform(0, 1, 50)
X_test.sort()
epsilon1 = np.random.normal(0, np.sqrt(0.25), 50)
Y_test = np.sin(2*np.pi*X_test) + epsilon1
testing_set = list(zip(X_test, Y_test))
```

(d) Estimate the regularized least squared polynomial regression model of order M = 1,2,3,9 using the training set T.

i. For example for M=1 , we need to estimate

ii. F(x) = β1x + β0

iii. For M = 2

iv. F(x) = β2x2 + β1x + β0.

Generating 'A' matrix for M=1,2,3,9

In [67]:
```
def generate_A(X, M):
  return np.array([X**i for i in range(M+1)]).T
```

In [68]:
```
A1 = generate_A(X, 1)
A2 = generate_A(X, 2)
A3 = generate_A(X, 3)
A9 = generate_A(X, 9)
```

Generating regularized 'u' matrix for M=1,2,3,9

In [69]:
```
def generate_u(A, Y, lambdaa):
    I = np.eye(A.shape[1])
    return np.linalg.inv(A.T @ A + lambdaa * I) @ A.T @ Y
```

In [70]:
```
lambdaa = 0.001
u1 = generate_u(A1, Y, lambdaa)
u2 = generate_u(A2, Y, lambdaa)
u3 = generate_u(A3, Y, lambdaa)
u9 = generate_u(A9, Y, lambdaa)

print(f"Estimated F(x) for M=1:\n{A1@u1}\n")
print(f"Estimated F(x) for M=2:\n{A2@u2}\n")
print(f"Estimated F(x) for M=3:\n{A3@u3}\n")
print(f"Estimated F(x) for M=9:\n{A9@u9}\n")
```

```
Estimated F(x) for M=1:
[ 1.00561586  0.92718954  0.78458134  0.63745388  0.61447713  0.60473407
  0.53457044  0.50941921  0.46849382  0.41697489  0.35679008  0.29740431
  0.27400891  0.24480497  0.02335873 -0.13100558 -0.23454286 -0.26356454
 -0.41807005 -0.48695577]

Estimated F(x) for M=2:
[ 0.89213997  0.84538541  0.75168772  0.64328158  0.62527556  0.61755251
  0.56039176  0.53924143  0.50408149  0.45850884  0.40341915  0.34710538
  0.32438688  0.29560487  0.06207262 -0.11669213 -0.24394907 -0.28067884
 -0.48402949 -0.57893041]

Estimated F(x) for M=3:
[ 0.39635682  0.67653877  0.93270791  0.92042521  0.8982883   0.88749331
  0.78741627  0.74304965  0.66282688  0.55004991  0.40597702  0.25590622
  0.19579154  0.12068717 -0.38718531 -0.57800844 -0.5770594  -0.55357532
 -0.22318921  0.05189209]

Estimated F(x) for M=9:
[ 0.31414204  0.58496606  0.89413794  0.96070053  0.94844918  0.94147356
  0.86102067  0.819848    0.73999775  0.6188589   0.45271493  0.26920962
  0.19329322  0.09685217 -0.56971925 -0.75638418 -0.6581498  -0.59609246
 -0.08733822  0.13848777]
```

(e) List the value of coefficients of estimated regularized least squared polynomial regression models for each case.

In [71]:
```
print(f"Coefficients for M=1:\n{u1}\n")
print(f"Coefficients for M=2:\n{u2}\n")
print(f"Coefficients for M=3:\n{u3}\n")
print(f"Coefficients for M=9:\n{u9}\n")
```

```
Coefficients for M=1:
[ 1.01342356 -1.50149513]

Coefficients for M=2:
[ 0.89660918 -0.85624452 -0.62085377]

Coefficients for M=3:
[  0.362405     6.63899689 -21.17789249  14.23356073]

Coefficients for M=9:
[  0.28338423   5.98012782 -12.5029515   -3.52157072   3.15346683
   5.5205769    4.79296671   2.2522569   -1.13208822  -4.68474382]
```

(f) Obtain the prediction on testing set and compute the RMSE for regularized least squared polynomial regression models for order M =1,2,3 and 9.

Generating A_test and Y_pred_test (prediction for testing set) and Y_pred_train (prediction for training set) for M=1,2,3,9

In [72]:
```
A1_test = generate_A(X_test, 1)
A2_test = generate_A(X_test, 2)
A3_test = generate_A(X_test, 3)
A9_test = generate_A(X_test, 9)

Y_pred_test1 = A1_test @ u1
Y_pred_test2 = A2_test @ u2
Y_pred_test3 = A3_test @ u3
Y_pred_test9 = A9_test @ u9

Y_pred_train1 = A1 @ u1
Y_pred_train2 = A2 @ u2
Y_pred_train3 = A3 @ u3
Y_pred_train9 = A9 @ u9
```

In [73]:
```
def calculate_rmse(Y_test, Y_pred):
  return np.sqrt(np.mean((Y_test-Y_pred)**2))
```

Training RMSE

In [74]:
```
train_rmse1 = calculate_rmse(Y, Y_pred_train1)
train_rmse2 = calculate_rmse(Y, Y_pred_train2)
train_rmse3 = calculate_rmse(Y, Y_pred_train3)
train_rmse9 = calculate_rmse(Y, Y_pred_train9)

print(f"Training RMSE for M=1:\n{train_rmse1}\n")
print(f"Training RMSE for M=2:\n{train_rmse2}\n")
print(f"Training RMSE for M=3:\n{train_rmse3}\n")
print(f"Training RMSE for M=9:\n{train_rmse9}\n\n")
```

```
Training RMSE for M=1:
0.5903539292645911

Training RMSE for M=2:
0.5882485663741622

Training RMSE for M=3:
0.43394897461544935

Training RMSE for M=9:
0.4203892510100091
```

Testing RMSE

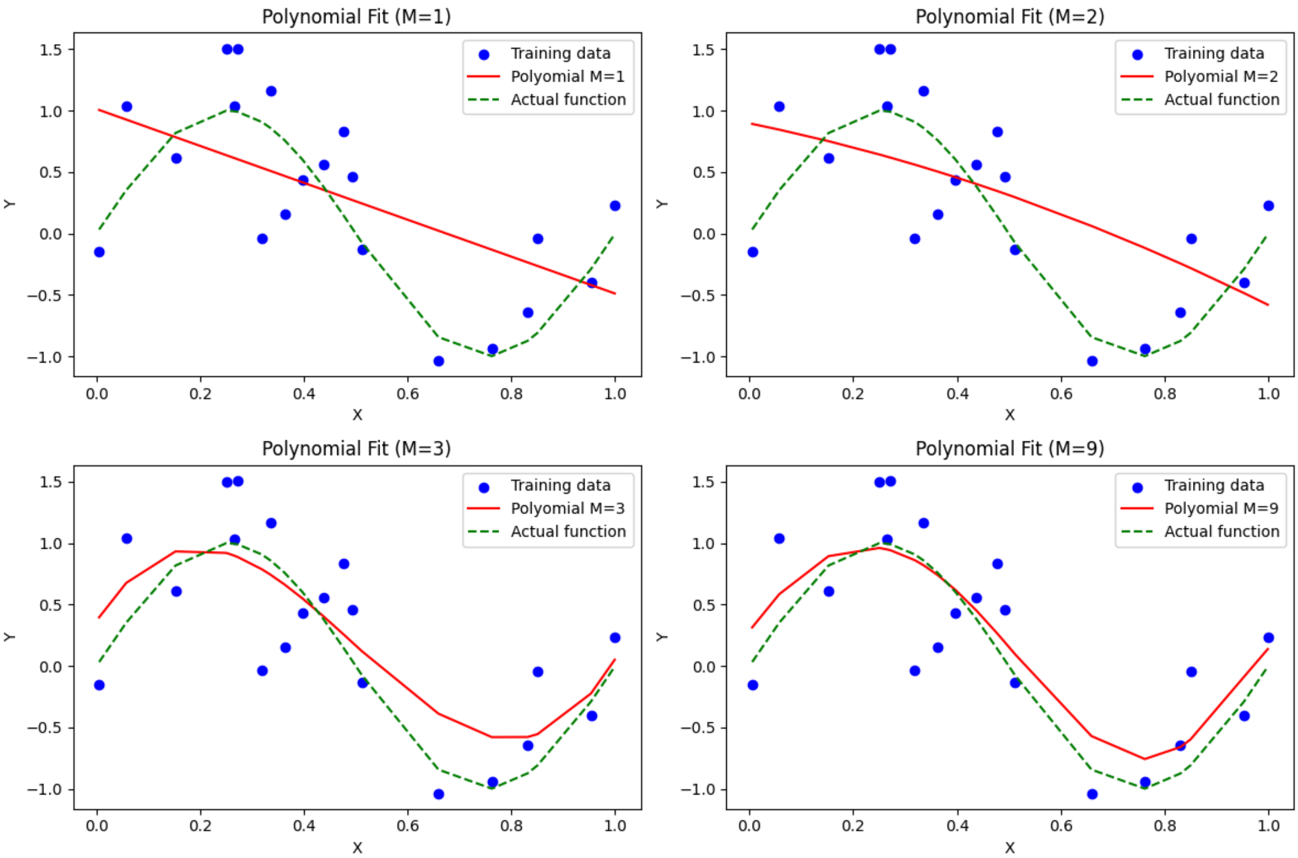
In [75]:
```
test_rmse1 = calculate_rmse(Y_test, Y_pred_test1)
test_rmse2 = calculate_rmse(Y_test, Y_pred_test2)
test_rmse3 = calculate_rmse(Y_test, Y_pred_test3)
test_rmse9 = calculate_rmse(Y_test, Y_pred_test9)

print(f"Testing RMSE for M=1:\n{test_rmse1}\n")
print(f"Testing RMSE for M=2:\n{test_rmse2}\n")
print(f"Testing RMSE for M=3:\n{test_rmse3}\n")
print(f"Testing RMSE for M=9:\n{test_rmse9}\n")
```

```
Testing RMSE for M=1:
0.8455536621624773

Testing RMSE for M=2:
0.847817514770294

Testing RMSE for M=3:
0.6289108977543594

Testing RMSE for M=9:
0.5682413979680596
```

(g) Plot the estimate obtained by regularized least squared polynomial regression models for order M =1,2,3 and 9 for training set along with y1, y2,..., y20. Also plot our actual mean estimate E(Y/X) = sin (2 π xi).

In [76]:
```
def plot_training_set(X_train, Y_train, u_M, M, label):

  Y_train = np.sin(2*np.pi*X_train)
  A_train = generate_A(X_train, M)
  Y_pred_train = A_train @ u_M
  Y_actual_train = np.sin(2 * np.pi * X_train)

  plt.scatter(X, Y, color='blue', label='Training data')
  plt.plot(X_train, Y_pred_train, color='red', label=f'Polyomial M={M}')
  plt.plot(X_train, Y_actual_train, color='green', linestyle='--', label='Actual
  plt.xlabel('X')
  plt.ylabel('Y')
  plt.title(f'Polynomial Fit (M={M})')
  plt.legend()
```

```
In [77]: plt.figure(figsize=(12, 8))

plt.subplot(2, 2, 1)
plot_training_set(X, Y, u1, 1, 'M=1')

plt.subplot(2, 2, 2)
plot_training_set(X, Y, u2, 2, 'M=2')

plt.subplot(2, 2, 3)
plot_training_set(X, Y, u3, 3, 'M=3')

plt.subplot(2, 2, 4)
plot_training_set(X, Y, u9, 9, 'M=9')

plt.tight_layout()
plt.show()
```

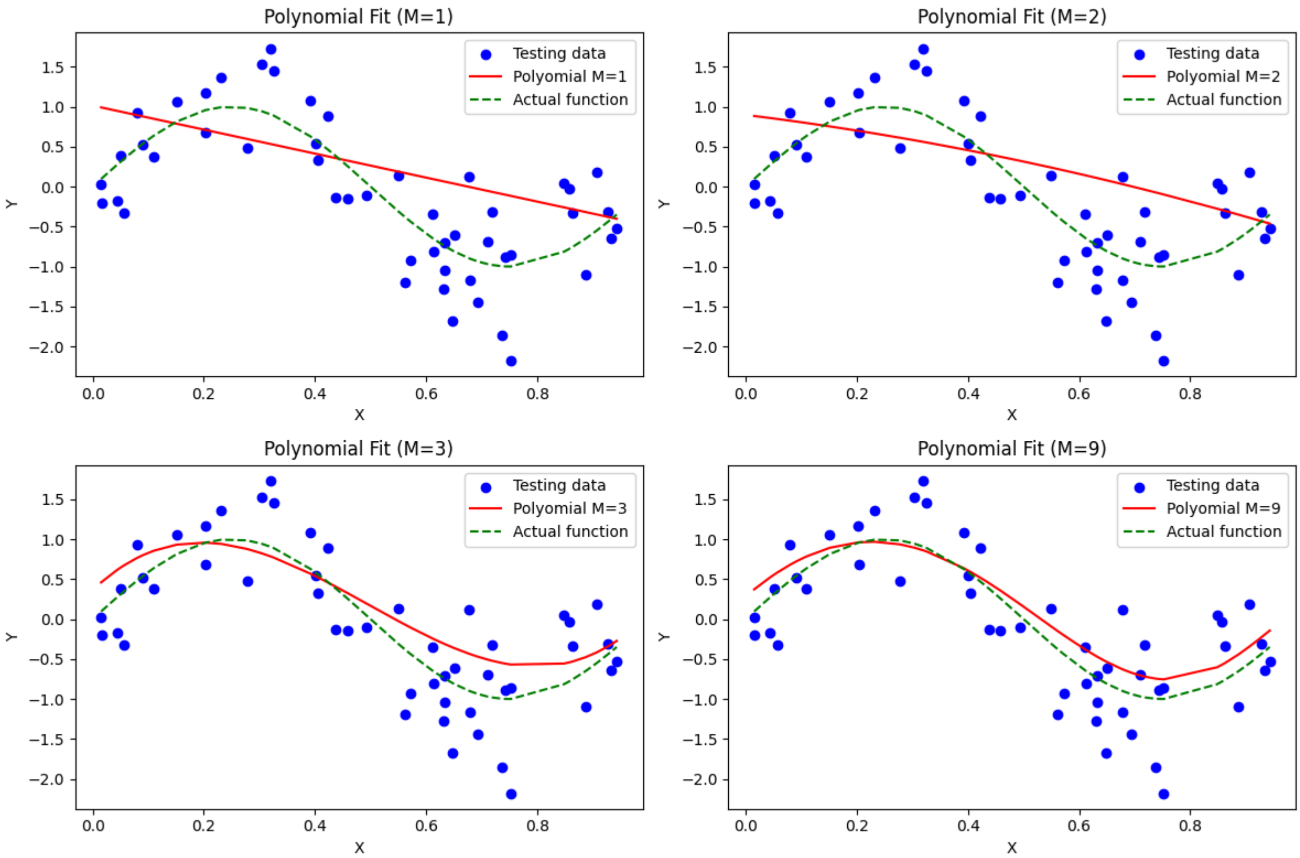


(h) Plot the estimate obtained by regularized least squared polynomial regression models for order M =1,2,3 and 9 for testing set along with y'1, y'2, , y'50. . Also plot the sin(2 π x'i).

```
In [78]: def plot_testing_set(X, Y, Y_pred_test, M, label):

  Y_actual_test = np.sin(2*np.pi*X)

  plt.scatter(X_test, Y_test, color='blue', label='Testing data')
  plt.plot(X_test, Y_pred_test, color='red', label=f'Polyomial M={M}')
  plt.plot(X_test, Y_actual_test, color='green', linestyle='--', label='Actual fu
  plt.xlabel('X')
  plt.ylabel('Y')
  plt.title(f'Polynomial Fit (M={M})')
  plt.legend()
```

In [79]:

```
plt.figure(figsize=(12, 8))

plt.subplot(2, 2, 1)
plot_testing_set(X_test, Y_test, Y_pred_test1, 1, 'M=1')

plt.subplot(2, 2, 2)
plot_testing_set(X_test, Y_test, Y_pred_test2, 2, 'M=2')

plt.subplot(2, 2, 3)
plot_testing_set(X_test, Y_test, Y_pred_test3, 3, 'M=3')

plt.subplot(2, 2, 4)
plot_testing_set(X_test, Y_test, Y_pred_test9, 9, 'M=9')

plt.tight_layout()
plt.show()
```



(i) Study the effect of regularization parameter λ on testing RMSE and flexibility of curve and list your observations

In [80]:

```
lambdas = [0.001, 1.0, 10.0]
test_RMSE_lambdas = {l: {} for l in lambdas}

for lambdaa in lambdas:
  for M in [1, 2, 3, 9]:
    A = generate_A(X, M)
    u = generate_u(A, Y, lambdaa)
    Y_pred = A @ u
    test_RMSE_lambdas[lambdaa][M] = calculate_rmse(Y, Y_pred)

for lambdaa in lambdas:
  print(f"Testing RMSE for lambda = {lambdaa}:\n{test_RMSE_lambdas[lambdaa]}\n")
```

```
Testing RMSE for lambda = 0.001:
{1: np.float64(0.5903539292645911), 2: np.float64(0.5882485663741622), 3: np.float
64(0.43394897461544935), 9: np.float64(0.4203892510100091)}

Testing RMSE for lambda = 1.0:
{1: np.float64(0.6232163438652369), 2: np.float64(0.6036434601685304), 3: np.float
64(0.6046295535668766), 9: np.float64(0.5793724029009463)}

Testing RMSE for lambda = 10.0:
{1: np.float64(0.7190988290816616), 2: np.float64(0.7025941686544869), 3: np.float
64(0.6933460226872893), 9: np.float64(0.6873319982502403)}
```

Summary of Observations

Lower λ: The model is less regularized, which might lead to overfitting and a "lower training RMSE but higher testing RMSE".

Higher λ: The model is more regularized, which might result in "higher training RMSE but lower testing RMSE".

Adjusting λ helps balance the trade-off between fitting the training data and generalizing to testing data.