Foundation of Machine Learning

Assignment-2

Name:Pranshu Parate

Id:202211063

Univariate Case

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

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)

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 [2]:
```
import numpy as np
import matplotlib.pyplot as plt
import math
import pandas as pd
```

In [3]:
```
x_train = np.random.uniform(0,1,size=20)
e_train = np.random.normal(0, 0.25, 20)

y = [math.sin(2*math.pi*x_train[i]) + e_train[i] for i in range(20)]
y_train = np.array(y)

#creating (x,y) pairs

train_df = pd.DataFrame([x_train, y_train])
train_df = train_df.T
train_df.columns = ['x', 'y']
# train_df

x_test = np.random.uniform(0,1,size=50)
e_test = np.random.normal(0, 0.25, 50)

y = []
```

```
for i in range(20):
  y.append(math.sin(2 * math.pi * x_test[i]) + e_test[i])

y_test = np.array(y)

#creating (x,y) pairs

test_df = pd.DataFrame([x_test, y_test])
test_df = test_df.T
test_df.columns = ['x', 'y']
```

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) = β2x

2 + β1x + β0 .

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

In [4]:
```
coeff = {}

# using gd

# for m = 1

lmda = -1
alpha = 0.01

aug_mat = pd.DataFrame(train_df['x'])
aug_mat['ones'] = np.ones(len(aug_mat))

scaled_y = (train_df['y'] - train_df['y'].mean()) / (train_df['y'].max() - train_df['y'].min())
theta = np.array([2,2])

for i in range(100):
  temp1 = np.dot(theta, aug_mat.T)
  temp2 = scaled_y - temp1
```

```
  temp3 = np.dot(temp2.T, aug_mat)

  temp = lmda * theta -  temp3
  theta = theta - alpha * temp

coeff[1] = list(theta)


# m = 2

aug_mat = pd.DataFrame([train_df['x']**2, train_df['x'], np.ones(20)])
aug_mat = aug_mat.T
aug_mat.columns = ['x^2', 'x', 'ones']

scaled_y = (train_df['y'] - train_df['y'].mean()) / (train_df['y'].max() - train_df['y'].min())
theta = np.array([2,2,2])

for i in range(100):
  temp1 = np.dot(theta, aug_mat.T)
  temp2 = scaled_y - temp1
  temp3 = np.dot(temp2.T, aug_mat)

  temp = lmda * theta -  temp3
  theta = theta - alpha * temp

coeff[2] = list(theta)


# for m = 3

aug_mat = pd.DataFrame([train_df['x']**3, train_df['x']**2, train_df['x'], np.ones(20)])
aug_mat = aug_mat.T
aug_mat.columns = ['x^3', 'x^2', 'x', 'ones']

scaled_y = (train_df['y'] - train_df['y'].mean()) / (train_df['y'].max() - train_df['y'].min())
theta = np.array([2,2,2,2])

for i in range(100):
  temp1 = np.dot(theta, aug_mat.T)
  temp2 = scaled_y - temp1
  temp3 = np.dot(temp2.T, aug_mat)

  temp = lmda * theta -  temp3
  theta = theta - alpha * temp
```

```
coeff[3] = list(theta)


# for m = 9

aug_mat = pd.DataFrame([train_df['x']**9, train_df['x']**8, train_df['x']**7, train_df['x']**6, train_df['x']**5, train
aug_mat = aug_mat.T
aug_mat.columns = ['x^9', 'x^8', 'x^7', 'x^6', 'x^5', 'x^4', 'x^3', 'x^2', 'x', 'ones']

scaled_y = (train_df['y'] - train_df['y'].mean()) / (train_df['y'].max() - train_df['y'].min())
theta = np.array([2,2,2,2,2,2,2,2,2,2])

for i in range(100):
  temp1 = np.dot(theta, aug_mat.T)
  temp2 = scaled_y - temp1
  temp3 = np.dot(temp2.T, aug_mat)

  temp = lmda * theta -  temp3
  theta = theta - alpha * temp

coeff[9] = list(theta)

# listing all theta

for i in coeff.keys():
  print(f'For M = {i}, coefficients are {coeff[i]}')
```

```
For M = 1, coefficients are [-0.4413685656291242, 0.27551473437165125]
For M = 2, coefficients are [-0.1269868468101464, -0.4984569977945408, 0.38883282135059427]
For M = 3, coefficients are [0.19966067444190658, -0.2827871707554427, -0.5831541408587926, 0.44049707829069573]
For M = 9, coefficients are [0.6729514850766737, 0.5220840859556666, 0.3540414778247116, 0.16595449888345226, -0.044635
05780751741, -0.2773150450802154, -0.5202763112804976, -0.7191295418470327, -0.6513839856942798, 0.6366991219384613]
```

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.

In [5]:
```
rmse = {}

# m = 1

aug_mat = pd.DataFrame([test_df['x'], np.ones(50)])

aug_mat = aug_mat.T

y_hat = np.dot(aug_mat, np.array(coeff[1]))
```

```
temp = math.sqrt(((test_df['y'] - y_hat)**2).mean())
rmse[1] = temp


# m = 2

aug_mat = pd.DataFrame([test_df['x']**2, test_df['x'], np.ones(50)])

aug_mat = aug_mat.T

y_hat = np.dot(aug_mat, np.array(coeff[2]))

temp = math.sqrt(((test_df['y'] - y_hat)**2).mean())
rmse[2] = temp


# m = 3

aug_mat = pd.DataFrame([test_df['x']**3, test_df['x']**2, test_df['x'], np.ones(50)])

aug_mat = aug_mat.T

y_hat = np.dot(aug_mat, np.array(coeff[3]))

temp = math.sqrt(((test_df['y'] - y_hat)**2).mean())
rmse[3] = temp


# m = 9

aug_mat = pd.DataFrame([test_df['x']**9, test_df['x']**8, test_df['x']**7, test_df['x']**6, test_df['x']**5, test_df['x

aug_mat = aug_mat.T

y_hat = np.dot(aug_mat, np.array(coeff[9]))

temp = math.sqrt(((test_df['y'] - y_hat)**2).mean())
rmse[9] = temp


for i in rmse.keys():
  print(f'For M = {i}, RMSE is {rmse[i]}')
```

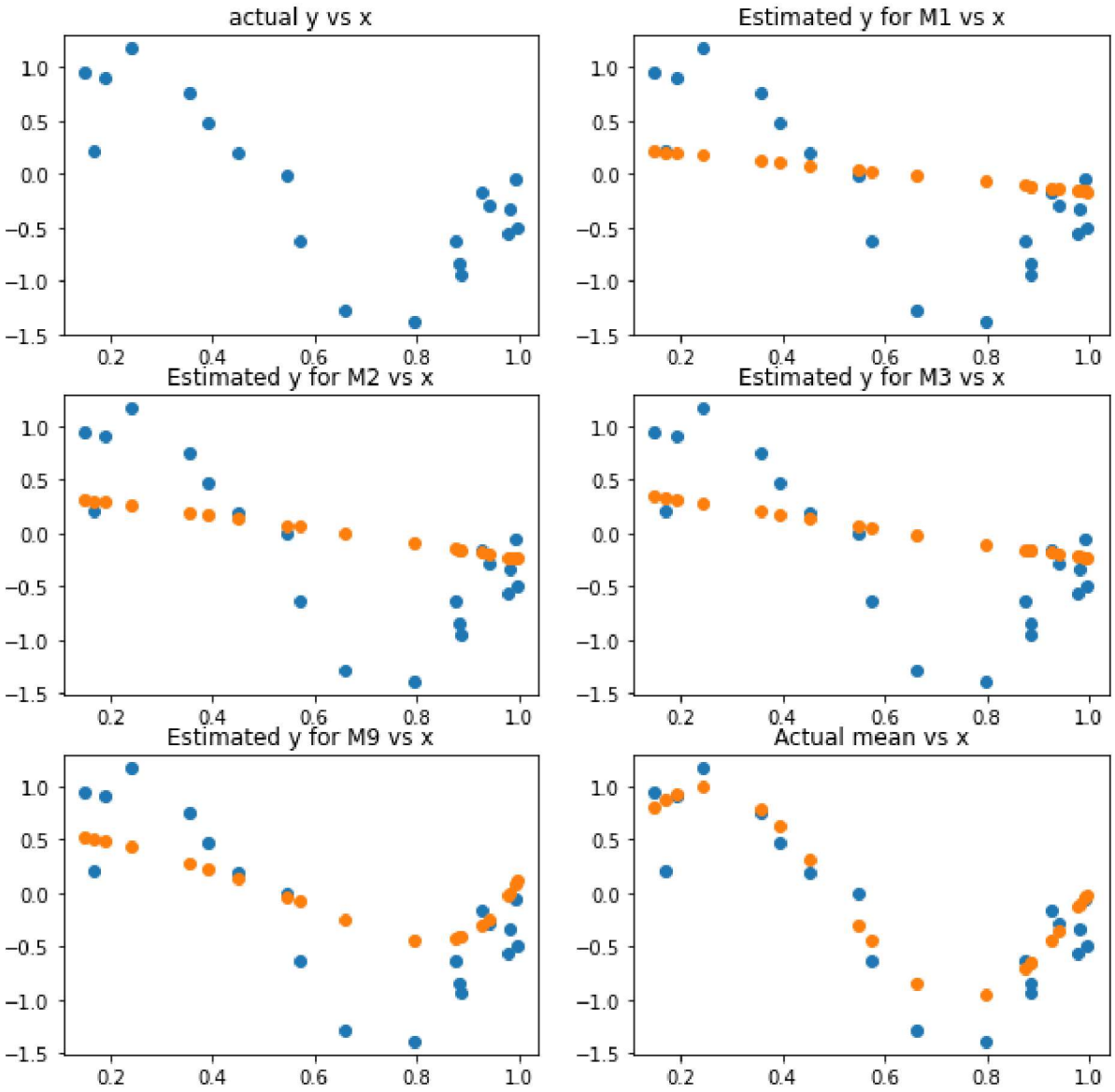
```
For M = 1, RMSE is 0.6387858408330929
For M = 2, RMSE is 0.6069184768453693
For M = 3, RMSE is 0.5960323933832512
For M = 9, RMSE is 0.5058104044217054
```

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) .

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 [6]:
```
est_y_m1 = np.dot(pd.DataFrame([train_df['x'], np.ones(20)]).T, coeff[1])
est_y_m2 = np.dot(pd.DataFrame([train_df['x']**2, train_df['x'], np.ones(20)]).T, coeff[2])
est_y_m3 = np.dot(pd.DataFrame([train_df['x']**3, train_df['x']**2, train_df['x'], np.ones(20)]).T, coeff[3])
est_y_m9 = np.dot(pd.DataFrame([train_df['x']**9, train_df['x']**8, train_df['x']**7, train_df['x']**6, train_df['x']**

actual_mean = 2*math.pi*train_df['x']
actual_mean = np.sin(actual_mean)

actual_y = train_df['y']

plt.figure(figsize = (10,10))

sub1 = plt.subplot(3,2,1)
sub2 = plt.subplot(3,2,2)
sub3 = plt.subplot(3,2,3)
sub4 = plt.subplot(3,2,4)
sub5 = plt.subplot(3,2,5)
sub6 = plt.subplot(3,2,6)

sub1.scatter(train_df['x'], actual_y)
sub1.set_title("actual y vs x")

sub2.scatter(train_df['x'], actual_y)
sub2.scatter(train_df['x'], est_y_m1)
sub2.set_title("Estimated y for M1 vs x")

sub3.scatter(train_df['x'], actual_y)
sub3.scatter(train_df['x'], est_y_m2)
sub3.set_title("Estimated y for M2 vs x")

sub4.scatter(train_df['x'], actual_y)
sub4.scatter(train_df['x'], est_y_m3)
sub4.set_title("Estimated y for M3 vs x")
```

```
sub5.scatter(train_df['x'], actual_y)
sub5.scatter(train_df['x'], est_y_m9)
sub5.set_title("Estimated y for M9 vs x")

sub6.scatter(train_df['x'], actual_y)
sub6.scatter(train_df['x'], actual_mean)
sub6.set_title("Actual mean vs x")

plt.show()
```



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

From different values of lamda it has been observed that overfitting will be reduce as we increase the value of lamda because we giving penalty to high value of weights.

Bivariate Case

a) Construct the training set T = { (x1,y1),(x2,y2),.....,(x3,y20)} using the relation

Yi = sin(2 π (||xi||) + εi where εi ~ N(0,0.25) and xi = (xi 1 ,xi 2) where xi 1 ,xi 2 are from

U[0,1]. 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 [7]:
```
# train set

x1_train = np.random.uniform(0, 1, 20)
x2_train = np.random.uniform(0, 1, 20)
e_train = np.random.normal(0, 0.25, 20)

y = [math.sin(2 * math.pi * math.sqrt(x1_train[i]**2 + x2_train[i]**2)) + e_train[i] for i in range(20)]
y_train = np.array(y)

train_df = pd.DataFrame([x1_train, x2_train, y_train])
train_df = train_df.T

train_df.columns = ['x1', 'x2', 'y']
```

In [8]:
```
# test set

x1_test = np.random.uniform(0, 1, 50)
x2_test = np.random.uniform(0, 1, 50)
e_test = np.random.normal(0, 0.25, 50)

y = [math.sin(2 * math.pi * math.sqrt(x1_test[i]**2 + x2_test[i]**2)) + e_test[i] for i in range(50)]
y_test = np.array(y)

test_df = pd.DataFrame([x1_test, x2_test, y_test])
test_df = test_df.T

test_df.columns = ['x1', 'x2', 'y']
```

b) Obtain the prediction on testing set and compute the RMSE for regularized least squared polynomial regression models for order M =1,2 and 5 . Also plot the estimated function and target function for the training set and testing set.

In [9]:
```
# coeffecients
coeff ={}

# for m = 1
```

```
aug_mat = pd.DataFrame([train_df['x1'], train_df['x2'], np.ones(20)])
aug_mat = aug_mat.T
aug_mat.columns = ['x1', 'x2', 'ones']

temp1 = np.dot(aug_mat.T, aug_mat)
temp2 = np.linalg.inv(temp1)
temp3 = np.dot(aug_mat.T, train_df['y'])
theta = np.dot(temp2, temp3)
coeff[1] = theta


# for m = 2

aug_mat = pd.DataFrame([train_df['x1']**2, train_df['x2']**2, train_df['x1'] * train_df['x2'], train_df['x1'], train_df
aug_mat = aug_mat.T
aug_mat.columns = ['x1^2', 'x2^2', 'x1*x2', 'x1', 'x2', 'ones']

temp1 = np.dot(aug_mat.T, aug_mat)
temp2 = np.linalg.inv(temp1)
temp3 = np.dot(aug_mat.T, train_df['y'])
theta = np.dot(temp2, temp3)
coeff[2] = theta



# m = 5

x1 = train_df['x1']
x2 = train_df['x2']

aug_mat = pd.DataFrame([x1**5, x2**5, x1**4 * x2, x1**3 * x2**2, x1**2 * x2 **3, x1 * x2**4, x1**4, x2**4, x1**3 * x2,
aug_mat = aug_mat.T
aug_mat.columns = ['x1**5', 'x2**5', 'x1**4 * x2', 'x1**3 * x2**2', 'x1**2 * x2 **3', 'x1 * x2**4', 'x1**4', 'x2**4', '

temp1 = np.dot(aug_mat.T, aug_mat)
temp2 = np.linalg.inv(temp1)
temp3 = np.dot(aug_mat.T, train_df['y'])
theta = np.dot(temp2, temp3)
coeff[5] = theta
```

In [10]:

```
rmse = {}

# m = 1

aug_mat = pd.DataFrame([test_df['x1'], test_df['x2'], np.ones(50)])
```

```
aug_mat = aug_mat.T

y_hat = np.dot(aug_mat, np.array(coeff[1]))

temp = math.sqrt(((test_df['y'] - y_hat)**2).mean())
rmse[1] = temp


# m = 2

aug_mat = pd.DataFrame([test_df['x1']**2, test_df['x2']**2, test_df['x1'] * test_df['x2'], test_df['x1'], test_df['x2']
aug_mat = aug_mat.T

y_hat = np.dot(aug_mat, np.array(coeff[2]))

temp = math.sqrt(((test_df['y'] - y_hat)**2).mean())
rmse[2] = temp


# m = 5

x1 = test_df['x1']
x2 = test_df['x2']

aug_mat = pd.DataFrame([x1**5, x2**5, x1**4 * x2, x1**3 * x2**2, x1**2 * x2 **3, x1 * x2**4, x1**4, x2**4, x1**3 * x2,
aug_mat = aug_mat.T

y_hat = np.dot(aug_mat, np.array(coeff[5]))

temp = math.sqrt(((test_df['y'] - y_hat)**2).mean())
rmse[5] = temp

for i in rmse.keys():
  print(f'For M = {i}, RMSE is {rmse[i]}')
```

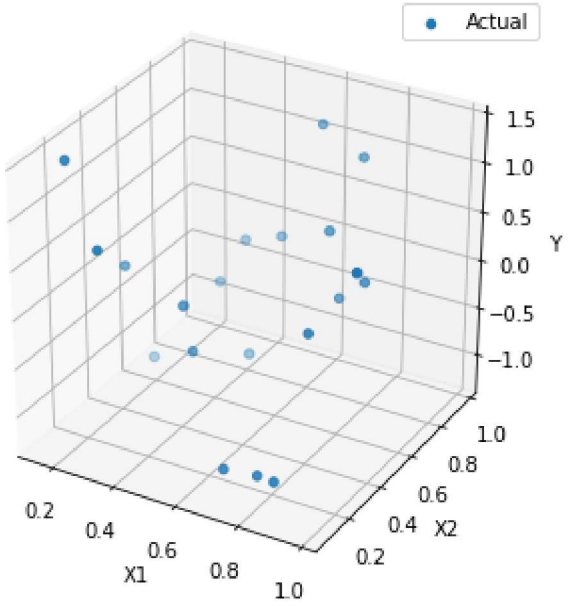
```
For M = 1, RMSE is 0.6969700269446358
For M = 2, RMSE is 0.7258442664009876
For M = 5, RMSE is 39.16337190359325
```

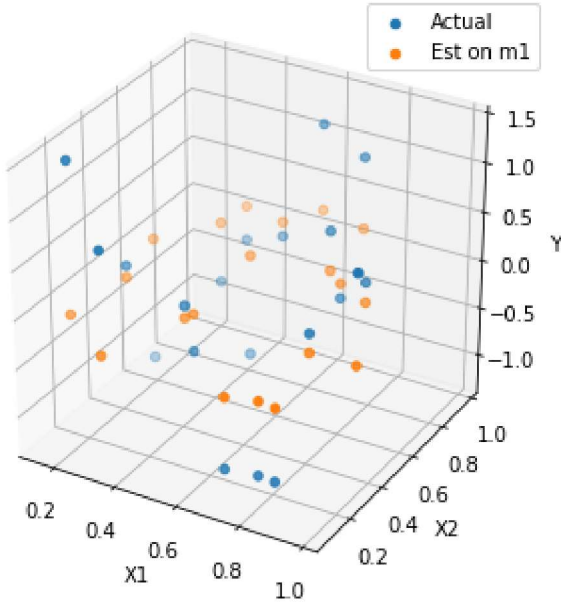
In [11]:

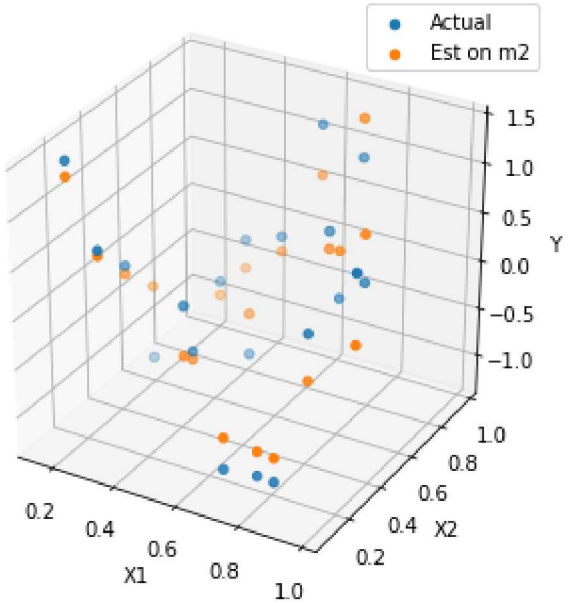
```
est_y_m1 = np.dot(pd.DataFrame([train_df['x1'], train_df['x2'], np.ones(20)]).T, coeff[1])
est_y_m2 = np.dot(pd.DataFrame([train_df['x1']**2, train_df['x2']**2, train_df['x1'] * train_df['x2'], train_df['x1'],

x1 = train_df['x1']
x2 = train_df['x2']
```

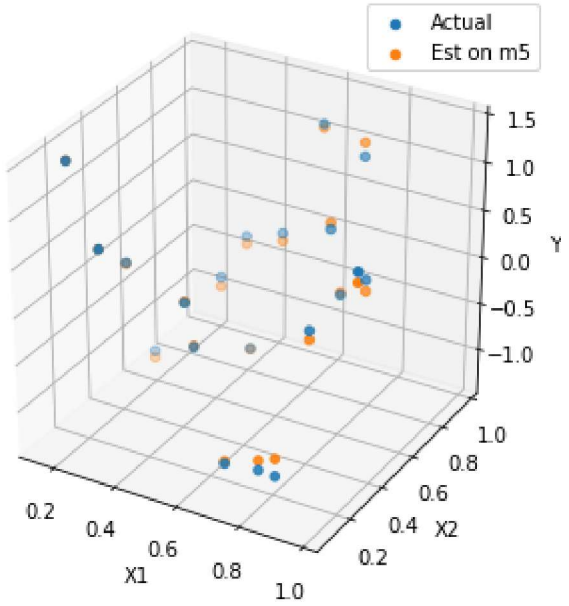
```
temp = pd.DataFrame([x1**5, x2**5, x1**4 * x2, x1**3 * x2**2, x1**2 * x2 **3, x1 * x2**4, x1**4, x2**4, x1**3 * x2, x1*
est_y_m5 = np.dot(temp.T, coeff[5])

actual_mean = np.array([2*math.pi* math.sqrt(train_df['x1'][i]**2 + train_df['x2'][i]**2) for i in range(20)])
actual_mean = np.sin(actual_mean)

actual_y = train_df['y']


fig = plt.figure(figsize=(12,12))

ax = fig.add_subplot(2, 2, 1, projection='3d')

scatter1 = ax.scatter(x1, x2, actual_y, label = "Actual")
ax.set_xlabel("X1")
ax.set_ylabel("X2")
ax.set_zlabel("Y")
ax.legend()


# m =1

ax = fig.add_subplot(2, 2, 2, projection='3d')

scatter_2_0 = ax.scatter(x1, x2, actual_y, label = "Actual")
scatter_2_1 = ax.scatter(x1, x2, est_y_m1, label = "Est on m1")

ax.set_xlabel("X1")
ax.set_ylabel("X2")
ax.set_zlabel("Y")
ax.legend()


# m = 2

ax = fig.add_subplot(2, 2, 3, projection='3d')

scatter_3_0 = ax.scatter(x1, x2, actual_y, label = "Actual")
scatter_3_1 = ax.scatter(x1, x2, est_y_m2, label = "Est on m2")

ax.set_xlabel("X1")
ax.set_ylabel("X2")
ax.set_zlabel("Y")
ax.legend()
```

```
# m = 5
ax = fig.add_subplot(2, 2, 4, projection='3d')

scatter_4_0 = ax.scatter(x1, x2, actual_y, label = "Actual")
scatter_4_1 = ax.scatter(x1, x2, est_y_m5, label = "Est on m5")

ax.set_xlabel("X1")
ax.set_ylabel("X2")
ax.set_zlabel("Y")
ax.legend()

plt.show()
```









Real-world Datasets

a. Consider the motorcycle dataset. Estimate the Regularized Least Square regression models using the n sigmoidal basis functions. A variant of sigmoidal basis function can be obtained using σ(a, b, x) = a

Tx + b , a ∈ R n , b ∈ R for x ∈ R n .

I. Plot the estimated function and obtain the training RMSE error for n = 2, 5 , 10. What happens when you increase the number of basis functions.

II. For n =10, find the minimum mean and standard deviations of RMSE, NMSE and R2 using leave-one out method by tunning the parameter λ.

In [12]:
```
import numpy as np
import matplotlib.pyplot as plt
import math
import pandas as pd
```

In [13]:
```
from google.colab import drive
drive.mount('/content/drive')
motorcycle_data = pd.read_excel('/content/drive/MyDrive/ML/Share motorcycle.xlsx')
```

```
Mounted at /content/drive
```

In [14]:
```
coeff = {}


alpha = 0.01
lmda = -2**-8

scaled_x = (motorcycle_data['x'] - motorcycle_data['x'].mean()) / (motorcycle_data['x'].max() - motorcycle_data['x'].mi
# for n = 2
aug_mat = pd.DataFrame([ scaled_x**2,  scaled_x, np.ones(len( scaled_x))])
aug_mat = aug_mat.T
aug_mat.columns = ['x^2', 'x', 'ones']

# using gd
theta = np.array([2 for i in range(3)])
for i in range(100):
    temp1 = np.dot(theta, aug_mat.T)
    temp2 = 1 / (1 + math.e**( (-1) * temp1))
    temp3 = motorcycle_data['y'] - temp2
    temp4 = np.dot(temp3.T, aug_mat)
```

```
    temp = lmda * theta -  temp4
    theta = theta - alpha * temp

coeff[2] = theta


# n = 5
aug_mat = pd.DataFrame([ scaled_x**5,  scaled_x**4,  scaled_x**3,  scaled_x**2,  scaled_x, np.ones(len( scaled_x))])
aug_mat = aug_mat.T
aug_mat.columns = ['x^5', 'x^4', 'x^3', 'x^2', 'x', 'ones']

# using gd
theta = np.array([2 for i in range(6)])
for i in range(100):
    temp1 = np.dot(theta, aug_mat.T)
    temp2 = 1 / (1 + math.e**( (-1) * temp1))
    temp3 = motorcycle_data['y'] - temp2
    temp4 = np.dot(temp3.T, aug_mat)

    temp = lmda * theta -  temp4
    theta = theta - alpha * temp

coeff[5] = theta


# n = 10
aug_mat = pd.DataFrame([ scaled_x**10,  scaled_x**9,  scaled_x**8,  scaled_x**7,  scaled_x**6,  scaled_x**5,  scaled_x*
aug_mat = aug_mat.T
aug_mat.columns = ['x^10', 'x^9', 'x^8', 'x^7', 'x^6', 'x^5', 'x^4', 'x^3', 'x^2', 'x', 'ones']

# using gd
theta = np.array([2 for i in range(11)])
for i in range(200):
    temp1 = np.dot(theta, aug_mat.T)
    temp2 = 1 / (1 + math.e**( (-1) * temp1))
    temp3 = motorcycle_data['y'] - temp2
    temp4 = np.dot(temp3.T, aug_mat)

    temp = lmda * theta -  temp4
    theta = theta - alpha * temp


coeff[10] = theta
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:17: RuntimeWarning: overflow encountered in power
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:36: RuntimeWarning: overflow encountered in power
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:55: RuntimeWarning: overflow encountered in power
```

In [15]:
```
rmse = {}

# m = 2

aug_mat = pd.DataFrame([motorcycle_data['x']**2, motorcycle_data['x'], np.ones(len(motorcycle_data))])
aug_mat = aug_mat.T

y_hat = np.dot(aug_mat, np.array(coeff[2]))

temp = math.sqrt(((motorcycle_data['y'] - y_hat)**2).mean())
rmse[2] = temp


# m = 5
aug_mat = pd.DataFrame([motorcycle_data['x']**5, motorcycle_data['x']**4, motorcycle_data['x']**3, motorcycle_data['x']

aug_mat = aug_mat.T

y_hat = np.dot(aug_mat, np.array(coeff[5]))

temp = math.sqrt(((motorcycle_data['y'] - y_hat)**2).mean())
rmse[5] = temp

# m = 10
aug_mat = pd.DataFrame([motorcycle_data['x']**10, motorcycle_data['x']**9, motorcycle_data['x']**8, motorcycle_data['x'
aug_mat = aug_mat.T

y_hat = np.dot(aug_mat, np.array(coeff[10]))

temp = math.sqrt(((motorcycle_data['y'] - y_hat)**2).mean())
rmse[10] = temp


for i in rmse.keys():
  print(f'For M = {i}, RMSE is {rmse[i]}')
```

```
For M = 2, RMSE is 32190.424603655472
For M = 5, RMSE is 355296201.4345765
For M = 10, RMSE is 1.2519053134374347e+17
```

In [16]:
```
# leave one out method
```

```
rmse = []
nmse = []
r2 = []


lmda = 5
alpha = 0.01

length = len(motorcycle_data)

for i in range(length):
  motorcycle_data = pd.read_excel('/content/drive/MyDrive/ML/Share motorcycle.xlsx')

  test_set = motorcycle_data.iloc[i]
  train_set = motorcycle_data.drop(labels = i, axis = 0)

  motorcycle_data['x'] = (motorcycle_data['x'] - motorcycle_data['x'].mean()) / (motorcycle_data['x'].max() - motorcycl
  aug_mat = pd.DataFrame([motorcycle_data['x']**10, motorcycle_data['x']**9, motorcycle_data['x']**8, motorcycle_data['
  aug_mat = aug_mat.T
  aug_mat.columns = ['x^10', 'x^9', 'x^8', 'x^7', 'x^6', 'x^5', 'x^4', 'x^3', 'x^2', 'x', 'ones']

  # using gd
  motorcycle_data['y'] = (motorcycle_data['y'] - motorcycle_data['y'].mean()) / (motorcycle_data['y'].max() - motorcycl
  theta = np.array([2,2,2,2,2,2,2,2,2,2,2])

  for i in range(100):
    temp1 = np.dot(theta, aug_mat.T)
    temp2 = 1 / (1 + math.e**( (-1) * temp1))
    temp3 = motorcycle_data['y'] - temp2
    temp4 = np.dot(temp3.T, aug_mat)

    temp = lmda * theta -  temp4
    theta = theta - alpha * temp

  # rmse
  y_hat = [test_set['x']**10 * theta[0], test_set['x']**9 * theta[1], test_set['x']**8 * theta[2], test_set['x']**7 * t
  y_hat = sum(y_hat)

  temp = math.sqrt(((test_set['y'] - y_hat)**2).sum())
  rmse.append(temp)

  # nmse
  mse = ((test_set['y'] - y_hat)**2).sum()
  obs = (test_set['y'] ** 2).sum()
```

```
  if obs != 0:
    nmse.append(mse/obs)
  else:
    nmse.append(0)

  # r2
  ssr = ((test_set['y'] - y_hat)**2).sum()
  tss = ((test_set['y'] - test_set['y'].mean())**2).sum()

  if tss != 0:
    r2.append(1 - (ssr/tss))
  else:
    r2.append(0)
```

b. Consider the Boston Housing dataset. Use the ten-fold cross validation for obtaining the prediction of house price using the regularized least square RBF kernel regression model. By tunning the regularization parameter λ and RBF kernel parameter σ , obtain the minimum of mean of RMSE, NMSE, R 2 , MAE, training times (in seconds) along with their standard deviation across different folds.
Note :- The RBF kernel for two data points xi and xj ∈ R n is given by

K (xi , xj) = e −||xi−xj || 2 σ2 .

In [17]:
```
# boston housing
import time
import statistics
boston_data = pd.read_excel('/content/drive/MyDrive/ML/Share bostonhousing.xlsx')

# normalisation
boston_data['x1'] = (boston_data['x1'] - boston_data['x1'].mean()) / (boston_data['x1'].max() - boston_data['x1'].min()
boston_data['x2'] = (boston_data['x2'] - boston_data['x2'].mean()) / (boston_data['x2'].max() - boston_data['x2'].min()
boston_data['x3'] = (boston_data['x3'] - boston_data['x3'].mean()) / (boston_data['x3'].max() - boston_data['x3'].min()
boston_data['x4'] = (boston_data['x4'] - boston_data['x4'].mean()) / (boston_data['x4'].max() - boston_data['x4'].min()
boston_data['x5'] = (boston_data['x5'] - boston_data['x5'].mean()) / (boston_data['x5'].max() - boston_data['x5'].min()
boston_data['x6'] = (boston_data['x6'] - boston_data['x6'].mean()) / (boston_data['x6'].max() - boston_data['x6'].min()
boston_data['x7'] = (boston_data['x7'] - boston_data['x7'].mean()) / (boston_data['x7'].max() - boston_data['x7'].min()
boston_data['x8'] = (boston_data['x8'] - boston_data['x8'].mean()) / (boston_data['x8'].max() - boston_data['x8'].min()
boston_data['x9'] = (boston_data['x9'] - boston_data['x9'].mean()) / (boston_data['x9'].max() - boston_data['x9'].min()
boston_data['x10'] = (boston_data['x10'] - boston_data['x10'].mean()) / (boston_data['x10'].max() - boston_data['x10'].
boston_data['x11'] = (boston_data['x11'] - boston_data['x11'].mean()) / (boston_data['x11'].max() - boston_data['x11'].
boston_data['x12'] = (boston_data['x12'] - boston_data['x12'].mean()) / (boston_data['x12'].max() - boston_data['x12'].
boston_data['x13'] = (boston_data['x13'] - boston_data['x13'].mean()) / (boston_data['x13'].max() - boston_data['x13'].

X = boston_data.iloc[:,:13]
X['ones'] = np.ones(len(X))
```

```
y = boston_data["MEDV Median value of owner-occupied homes in $1000's"]

lmda = 5
sigma = 11

rmse = []
nmse = []
r2 = []
mae = []
train_time = []

k_fold = int(len(boston_data)/10)

for i in range(9):

  tt = time.time()

  temp_test_x = X.iloc[i*k_fold:(i+1)*k_fold]
  temp_train_x = X.drop(X.index[i*k_fold:(i+1)*k_fold])
  temp_test_y = y.iloc[i*k_fold:(i+1)*k_fold]
  temp_train_y = y.drop(y.index[i*k_fold:(i+1)*k_fold])

  lmda = 5
  sigma = 11
  h_mat = []

  def kernel(x1, x2, sigma):
    d= (x1 - x2) **2
    d= ((-1) / (2 * sigma**2)) * d
    ans = math. e ** (d. sum() )
    return ans

  for i in range(len(temp_train_x)):
    temp = []
    for j in range(len(temp_train_x.columns)):
      temp.append(kernel(temp_train_x.iloc[i], temp_train_x.iloc[j], sigma))
    h_mat.append(temp)


  h_mat = np.array(h_mat)

  iden_mat = np.identity(len(temp_train_x.columns))

  temp1 = lmda * iden_mat
  temp2 = np.dot(h_mat.T, h_mat)
```

```
  temp3 = temp1 + temp2
  temp4 = np.linalg.inv(temp3)
  temp5 = np.dot(h_mat.T, temp_train_y)
  theta = np.dot(temp4, temp5)


  # rmse
  temp1 = np.dot(temp_test_x, theta.T)
  temp2 = (temp_test_y - temp1) ** 2
  temp = math.sqrt(temp2.sum())
  rmse.append(temp)

  # nmse
  mse = temp2.sum()
  obs = (temp_test_y ** 2).sum()
  if obs != 0:
    nmse.append(mse/obs)
  else:
    nmse.append(0)


  # r2
  ssr = temp2.sum()
  tss = ((temp_test_y - temp_test_y.mean())**2).sum()

  if tss != 0:
    r2.append(1 - (ssr/tss))
  else:
    r2.append(0)

  # mae
  temp1 = np.dot(temp_test_x, theta.T)
  err = abs(temp1 - temp_test_y)
  mae.append(err.sum() / len(temp_test_y))

  # train time
  train_time.append(time.time() - tt)



print("Mean of RMSE:", statistics.mean(rmse))
print("Standard deviation of RMSE:", statistics.stdev(rmse))

print("Mean of NMSE:", statistics.mean(nmse))
print("Standard deviation of NMSE:", statistics.stdev(nmse))
```

```
print("Mean of R2:", statistics.mean(r2))
print("Standard deviation of R2:", statistics.stdev(r2))

print("Mean of MAE:", statistics.mean(mae))
print("Standard deviation of R2:", statistics.stdev(mae))

print("Mean of Training Time: ", statistics.mean(train_time))
print("Standard deviation of Training Time:", statistics.stdev(train_time))
```

```
Mean of RMSE: 87.8231285330316
Standard deviation of RMSE: 32.962058666161404
Mean of NMSE: 0.8636404977874459
Standard deviation of NMSE: 0.05992500572430794
Mean of R2: -5.449254127665529
Standard deviation of R2: 2.993038824358746
Mean of MAE: 11.172367488060422
Standard deviation of R2: 4.395811479543578
Mean of Training Time:  2.369571261935764
Standard deviation of Training Time: 0.03320627861187629
```