Homework 4

Author: Boqian Wang

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

Question 1

For the dataset, please import “hw4_sample.txt”. Each line contains exactly 2 numbers representing the X and Y coordinates of points generated by adding noise to a secret function.

In [ ]:
```
data = pd.read_csv('hw4_sample.txt',delimiter='\t',names=['X','Y'])
x = np.array(data['X'])
Y = np.array(data['Y'])
sort_index = np.argsort(x)
x = x[sort_index]
Y = Y[sort_index]
```

Implement the linear least square curve fitting (No regularization terms)

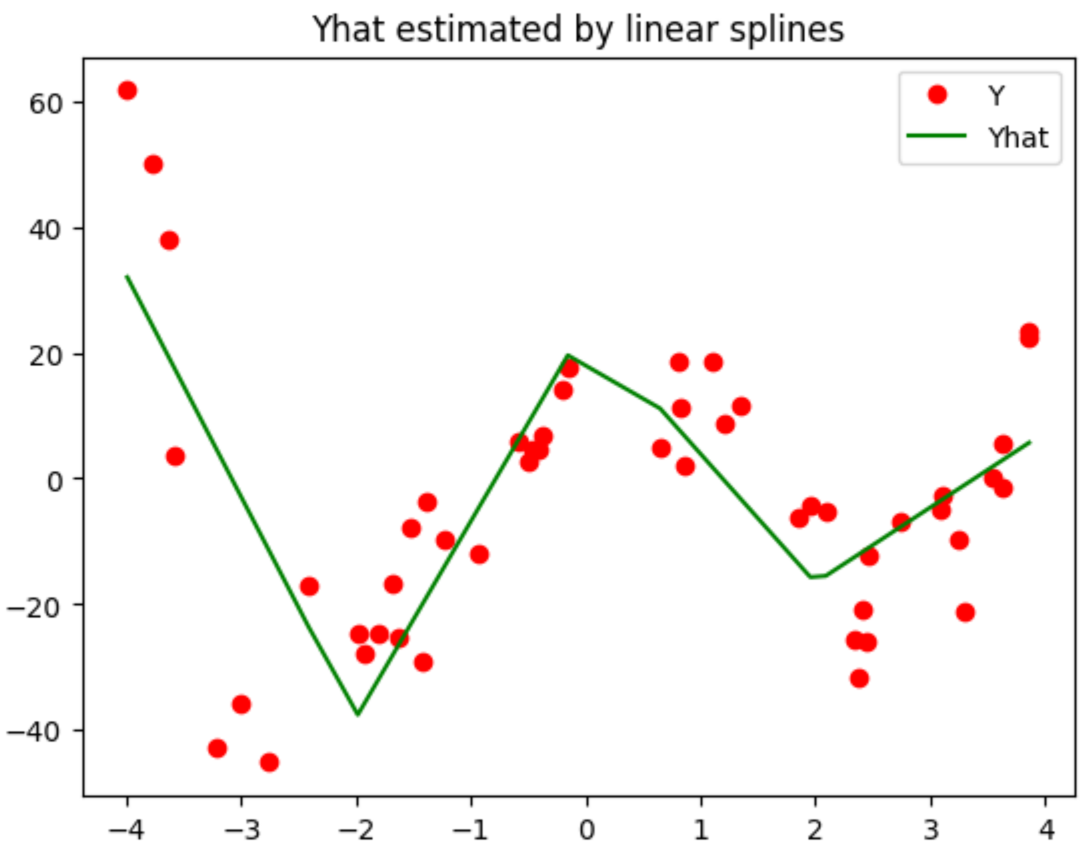
In [ ]:
```
# Step1: Implement the Sweep Operator:
def mySweep(A, m):
    n = A.shape[0]
    for k in range(m):
        for i in range(n):
            for j in range(n):
                if i != k and j != k:
                    A[i, j] = A[i, j] - A[i, k] * A[k, j] / A[k, k]
        for i in range(n):
            if i != k:
                A[i, k] = A[i, k] / A[k, k]
        for j in range(n):
            if j != k:
                A[k, j] = A[k, j] / A[k, k]
        A[k, k] = -1 / A[k, k]
    return A
```

In [ ]:
```
# Step2: Initialize X1:
X1 = np.matrix(np.ones(50)).T
X1 = np.hstack((X1, np.matrix(x).T))
X1 = np.hstack((X1, np.matrix((x > -2) * (x + 2)).T))
X1 = np.hstack((X1, np.matrix((x > 0) * x).T))
X1 = np.hstack((X1, np.matrix((x > 2) * (x - 2)).T))
X1 = np.array(X1)
```

In [ ]:
```
# Step3: Calculate beta1:
X1T = np.transpose(X1)
beta1 = np.dot(-mySweep(np.dot(X1T,X1),5),np.dot(X1T,Y))
print(beta1)
```

```
[-108.52756836  -35.21559922   66.46906825  -51.74936275   32.45507519]
```

In [ ]:
```
# Step4: Visualization
Yhat1 = np.dot(X1,beta1)
plt.plot(x, Y, 'ro', label='Y')
plt.plot(x, Yhat1, 'g-', label='Yhat')
plt.legend()
plt.title("Yhat estimated by linear splines")
plt.show()
```



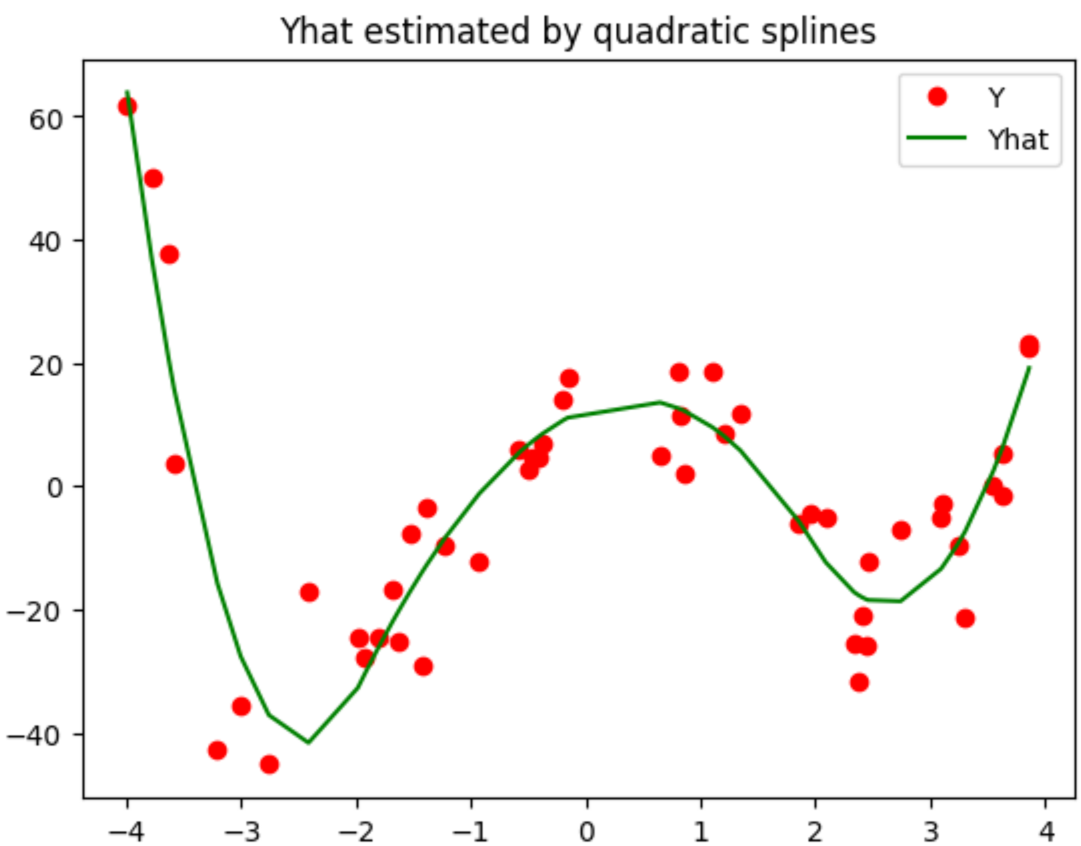
Implement the quadratic least square curve fitting (No regularization terms)

In [ ]:
```
# Step1: Initialize X2:
X2 = np.matrix(np.ones(50)).T
X2 = np.hstack((X2, np.matrix(x).T))
X2 = np.hstack((X2, np.square(np.matrix(x)).T))
X2 = np.hstack((X2, np.square(np.matrix((x > -2) * (x + 2))).T))
X2 = np.hstack((X2, np.square(np.matrix((x > 0) * x)).T))
X2 = np.hstack((X2, np.square(np.matrix((x > 2) * (x - 2))).T))
X2 = np.array(X2)
```

In [ ]:
```
# Step2: Calculate beta2:
X2T = np.transpose(X2)
beta2 = np.dot(-mySweep(np.dot(X2T,X2),6),np.dot(X2T,Y))
print(beta2)
```

```
[217.42707097 212.74113879  43.6721056  -51.23347665  -1.92459654
  34.05677044]
```

In [ ]:
```
# Step3: Visualization
Yhat2 = np.dot(X2,beta2)
plt.plot(x, Y, 'ro', label='Y')
plt.plot(x, Yhat2, 'g-', label='Yhat')
plt.legend()
plt.title("Yhat estimated by quadratic splines")
plt.show()
```



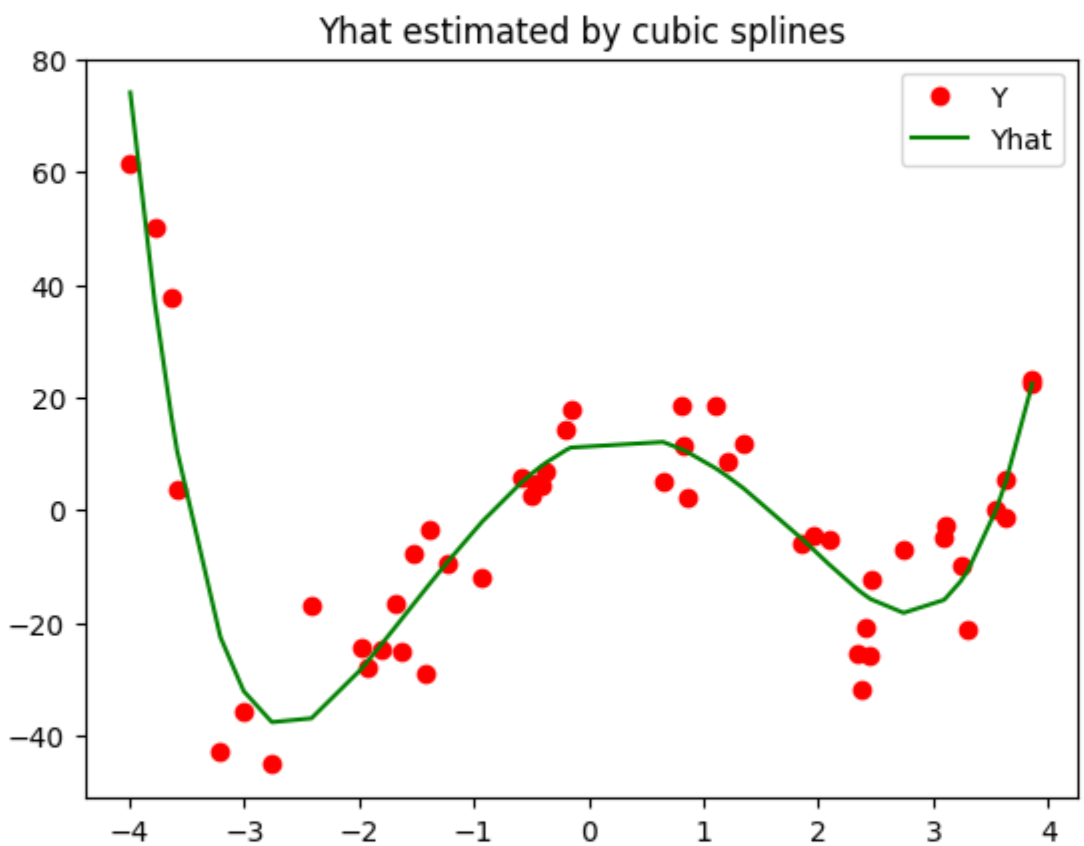
Implement the cubic least square curve fitting (No regularization terms)

In [ ]:
```
# Step1: Initialize X3:
X3 = np.matrix(np.ones(50)).T
X3 = np.hstack((X3, np.matrix(x).T))
X3 = np.hstack((X3, np.square(np.matrix(x)).T))
X3 = np.hstack((X3, np.power(np.matrix(x),3).T))
X3 = np.hstack((X3, np.power(np.matrix((x > -2) * (x + 2)),3).T))
X3 = np.hstack((X3, np.power(np.matrix((x > 0) * x),3).T))
X3 = np.hstack((X3, np.power(np.matrix((x > 2) * (x - 2)),3).T))
X3 = np.array(X3)
```

In [ ]:
```
# Step2: Calculate beta3:
X3T = np.transpose(X3)
beta3 = np.dot(-mySweep(np.dot(X3T,X3),7),np.dot(X3T,Y))
print(beta3)
```

```
[-107.66748313 -173.52313151 -102.15323964  -17.56492069   15.00881445
    4.43159612    8.86007884]
```

In [ ]:
```
# Step3: Visualization
Yhat3 = np.dot(X3,beta3)
plt.plot(x, Y, 'ro', label='Y')
plt.plot(x, Yhat3, 'g-', label='Yhat')
plt.legend()
plt.title("Yhat estimated by cubic splines")
plt.show()
```



Question 2

To find the best λ, you first randomly split your data into training set (80% of the data) and validation set (20% of the data).Then, try some different values of λs and visualize the traing error and validation error as a function of λ.

In [ ]:

```
# Step1: Implement Ridge Regression
def myRidge(X, Y, lambda_val):
    n = X.shape[0]
    p = X.shape[1]
    ones = np.ones((n, 1))
    Z = np.hstack((ones, X, Y.reshape(-1,1)))
    A = np.dot(Z.T, Z)
    D = np.diag(np.repeat(lambda_val, p+2))
    D[0, 0] = 0
    D[1, 1] = 0
    D[2, 2] = 0
    D[3, 3] = 0
    D[p+1, p+1] = 0
    A = A + D
    S = mySweep(A,p+1)
    beta = S[:p+1, p+1]
    return beta
```

In [ ]:

```
def custom_train_test_split(x, Y, test_size, random_state=None):
    if random_state is not None:
        np.random.seed(random_state)
    n = len(x)
    shuffled_indices = np.random.permutation(n)
    test_set_size = int(n * test_size)
    test_indices = shuffled_indices[:test_set_size]
    train_indices = shuffled_indices[test_set_size:]
    x_train, x_test = x[train_indices], x[test_indices]
    Y_train, Y_test = Y[train_indices], Y[test_indices]
    return x_train, x_test, Y_train, Y_test, test_indices, train_indices

x_train, x_valid, Y_train, Y_valid, test_indices, train_indices = custom_train_test
```

In [ ]:

```
x_train2 = x_train[:,1:]
```

In [ ]:

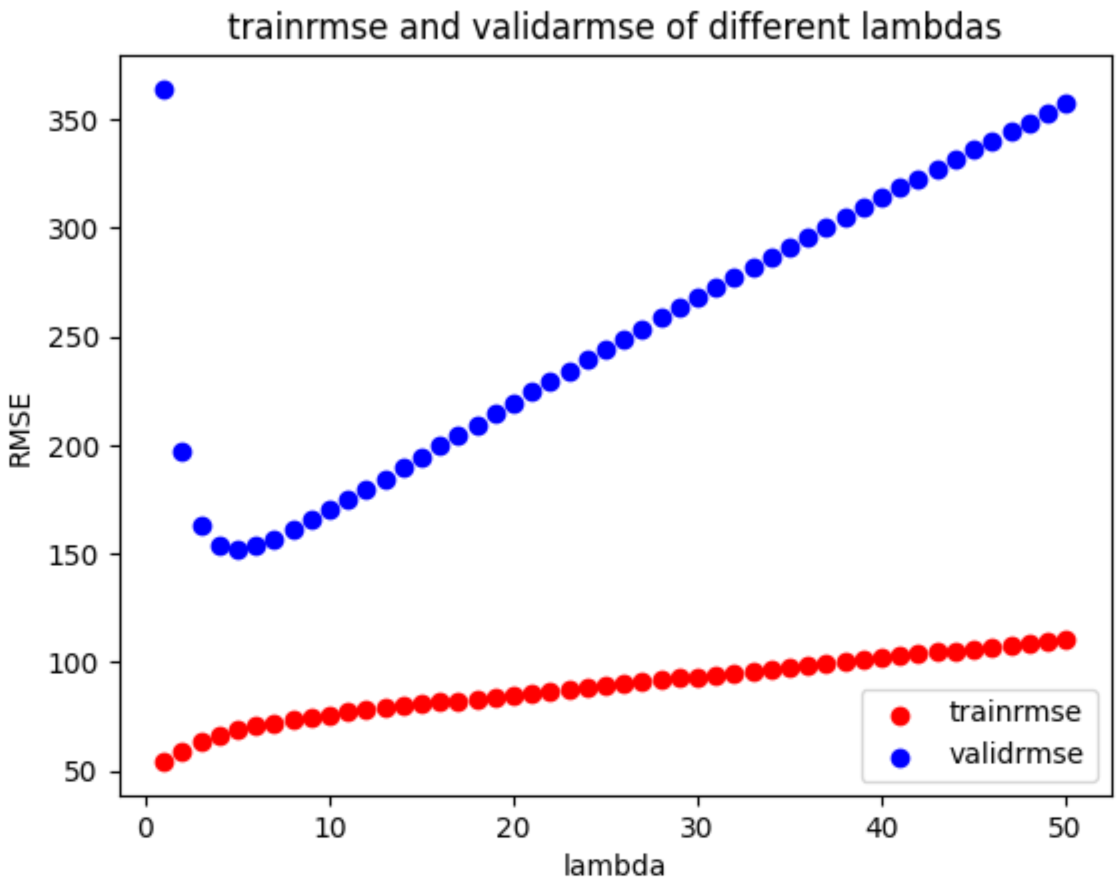
```
trainrmse = np.zeros((50,))
validrmse = np.zeros((50,))
```

In [ ]:

```
for lam in range(50):
    beta = myRidge(x_train2,Y_train,lam)
    Yhat = np.dot(x_train,beta)
    Yhat2 = np.dot(x_valid,beta)
    trainrmse[lam] = (np.sum(np.square(Yhat - Y_train))/40)
    validrmse[lam] = (np.sum(np.square(Yhat2 - Y_valid))/10)
```

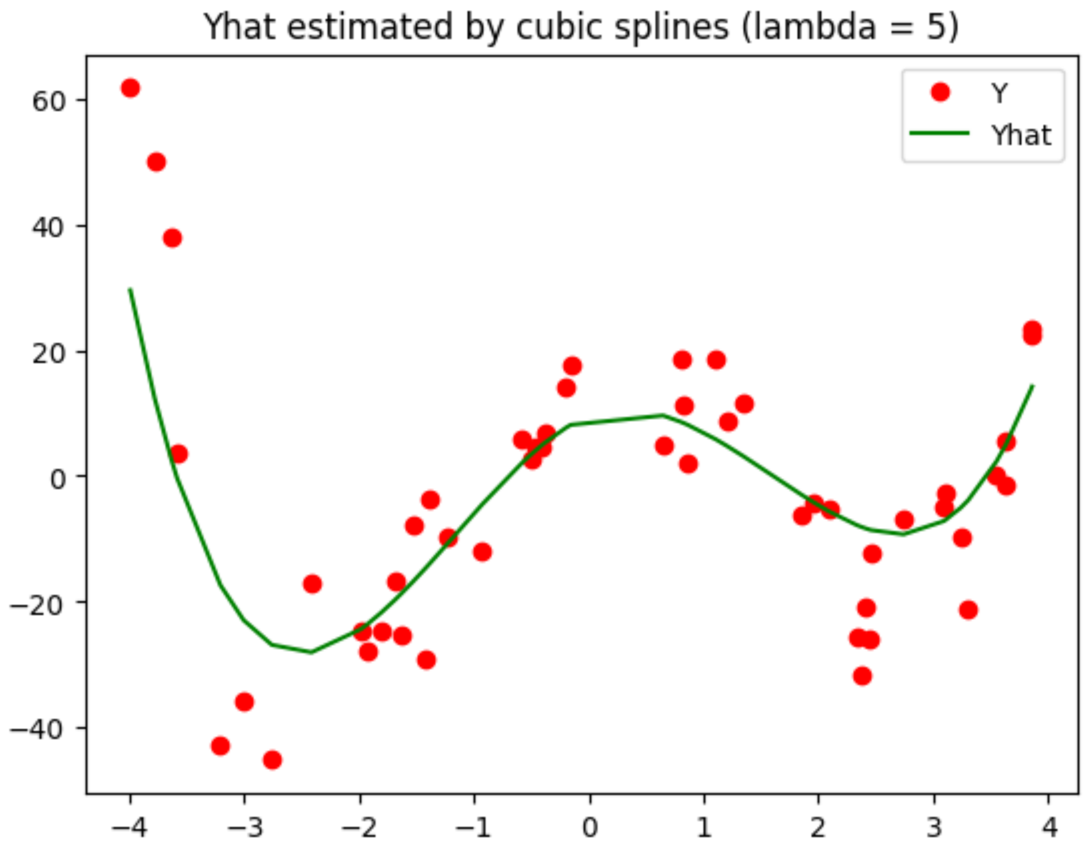
In [ ]:

```
x2 = np.arange(1, 51)
plt.scatter(x2, trainrmse, label='trainrmse', color='red')
plt.scatter(x2, validrmse, label='validrmse', color='blue')
plt.xlabel('lambda')
plt.ylabel('RMSE')
plt.legend()
plt.title("trainrmse and validarmse of different lambdas")
plt.show()
```



In [ ]:

```
Yhat12 = np.dot(X3,beta)
plt.plot(x, Y, 'ro', label='Y')
plt.plot(x, Yhat12, 'g-', label='Yhat')
plt.legend()
plt.title("Yhat estimated by cubic splines (lambda = 5)")
plt.show()
```



When $\lambda = 5$, validrmse reaches its minimum, which is the λ I like most.

Question 3

Determine the best k for the dataset.

When k = 1:

In [ ]:
```
train_indices = np.sort(train_indices)
train_indices
```

Out[ ]:
```
array([ 2,  3,  4,  5,  6,  8,  9, 10, 11, 13, 14, 15, 16, 18, 19, 20, 22,
       23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 37, 38, 39, 40,
       41, 42, 46, 47, 48, 49])
```

In [ ]:
```
test_indices = np.sort(test_indices)
test_indices
```

Out[ ]:
```
array([ 0,  1,  7, 12, 17, 21, 36, 43, 44, 45])
```

In [ ]:
```
x_train = x[train_indices]
Y_train = Y[train_indices]
x_test = x[test_indices]
Y_test = Y[test_indices]
```

In [ ]:
```
X9 = np.matrix(np.ones(40)).T
X10 = np.matrix(np.ones(10)).T
X9 = np.hstack((X9, np.matrix(x_train).T))
X10 = np.hstack((X10, np.matrix(x_test).T))
X9T = X9.T
beta9 = np.dot(np.linalg.inv(np.dot(X9T,X9)),np.dot(X9T,Y_train).T)
Y_train9_hat = np.dot(X9,beta9)
Y_test9_hat = np.dot(X10,beta9)
trainrmsetemp9 = (np.sum(np.square(Y_train9_hat - Y_train.reshape(40,1)))/40)
testrmsetemp9 = (np.sum(np.square(Y_test9_hat - Y_test.reshape(10,1)))/10)
```

When k is from 2 to 8:

In [ ]:
```
X_train_list = []
X_test_list = []
beta_list = []
for k in np.arange(2,9):
    X6 = np.matrix(np.ones(40)).T
    X7 = np.matrix(np.ones(10)).T
    X6 = np.hstack((X6, np.matrix(x_train).T))
    X7 = np.hstack((X7, np.matrix(x_test).T))
    k_array = np.linspace(-4, 4, k+2)

    for i in np.arange(1,k):
        temp = np.zeros((40,))
        temp2 = np.zeros((10,))
        temp += ((x_train > k_array[i])*np.power((x_train - k_array[i]),3)-np.power
        temp2 += ((x_test > k_array[i])*np.power((x_test - k_array[i]),3)-np.power(
        X6 = np.hstack((X6, np.matrix(temp).T))
        X6T = np.transpose(X6)
        X7 = np.hstack((X7, np.matrix(temp2).T))
        beta_temp = np.dot(np.linalg.inv(np.dot(X6T,X6)),np.dot(X6T,Y_train).T)
    X_train_list.append(X6)
    X_test_list.append(X7)
    beta_list.append(beta_temp)
```

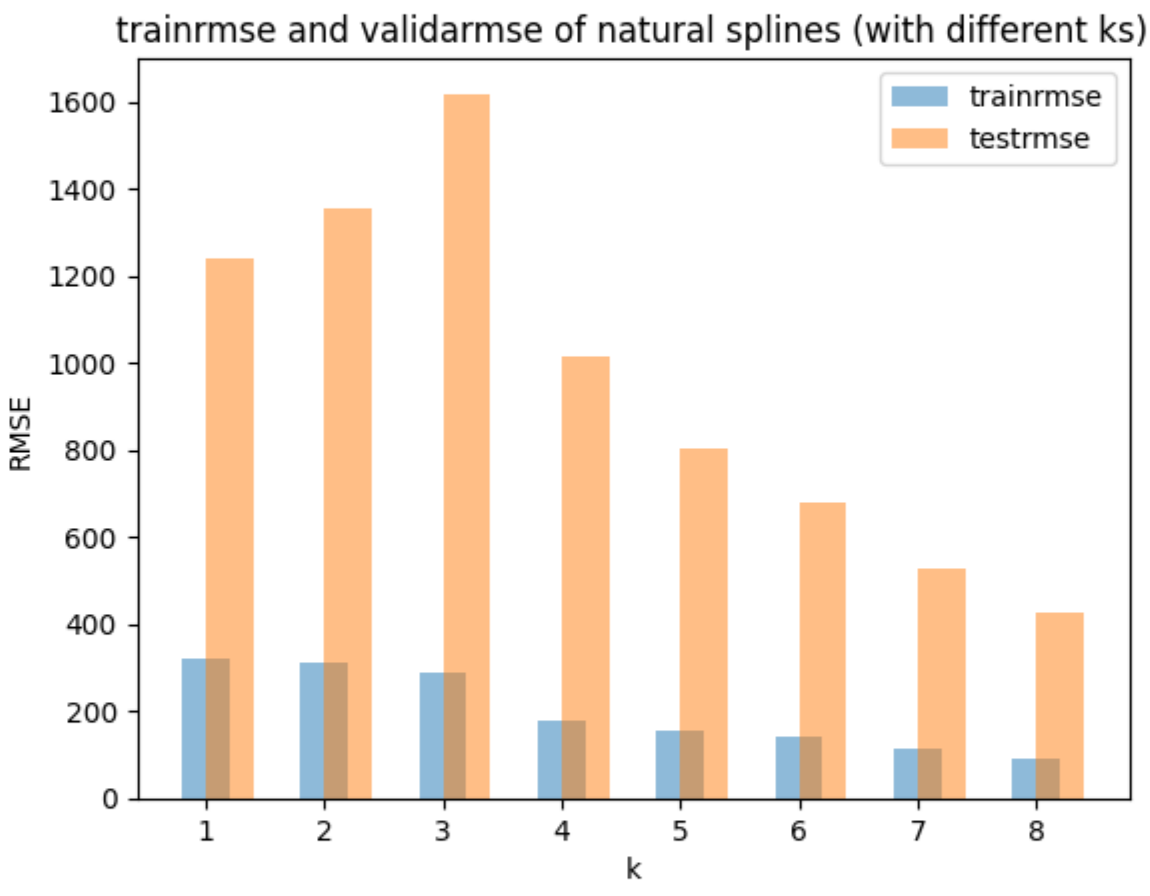
In [ ]:
```
train_rmse_list = []
test_rmse_list = []
for i in np.arange(0,7):
    Y_train_hat = np.dot(X_train_list[i],beta_list[i])
    Y_test_hat = np.dot(X_test_list[i],beta_list[i])
    trainrmsetemp = (np.sum(np.square(Y_train_hat - Y_train.reshape(40,1)))/40)
    testrmsetemp = (np.sum(np.square(Y_test_hat - Y_test.reshape(10,1)))/10)
    train_rmse_list.append(trainrmsetemp)
    test_rmse_list.append(testrmsetemp)
```

In [ ]:
```
train_rmse_list.insert(0,trainrmsetemp9)
test_rmse_list.insert(0,testrmsetemp9)
print(train_rmse_list)
print(test_rmse_list)
```

```
[318.55339778114694, 311.51096990383877, 286.7639429188533, 180.07150773443885, 153.
73071014046437, 139.28361036335588, 114.3279289384981, 92.6344741064368]
[1239.9542752604004, 1354.6755479215462, 1618.2409438611653, 1013.4318905501417, 80
2.6845452971136, 677.0838367621768, 528.7557275670586, 426.55627916642635]
```

Visualization:

In [ ]:
```
index = np.array(range(len(train_rmse_list)))+1
plt.bar(index, train_rmse_list, width=0.4, align='center', alpha=0.5, label='trainr
plt.bar(index, test_rmse_list, width=0.4, align='edge', alpha=0.5, label='testrmse'
plt.xlabel('k')
plt.ylabel('RMSE')
plt.legend()
plt.title("trainrmse and validarmse of natural splines (with different ks)")
plt.show()
```



According to the bar plot, k = 8 introduces the least RMSE both for training and validation data.

In [ ]:
```
X11 = np.matrix(np.ones(50)).T
X11 = np.hstack((X11, np.matrix(x).T))
k11 = np.linspace(-4, 4, 10)
for i in np.arange(1,8):
    temp11 = np.zeros((50,))
    temp11 += ((x > k11[i])*np.power((x - k11[i]),3)-np.power((x > 4) * (x - 4),3))
    X11 = np.hstack((X11, np.matrix(temp11).T))
    X11T = np.transpose(X11)
    beta_temp11 = np.dot(np.linalg.inv(np.dot(X11T,X11)),np.dot(X11T,Y).T)
Yhat11 = np.dot(X11,beta_temp11)
plt.plot(x, Y, 'ro', label='Y')
plt.plot(x, Yhat11, 'g-', label='Yhat')
plt.legend()
plt.title("Yhat estimated by natural splines (k=8)")
plt.show()
```