Regression Project

Ashwini Nikumbh, Ponkrshnan Thiagarajan, Vrushaketu Mali

Introduction ¶

This project aims at developing a framework to perform regression on a given dataset. In addition to conventional regression, the code can also perform cross validation to select the best model from a given database of models. The model selection procedure brings in the machine learning capability of the code which is not generally present in a conventional regression setting.

In [1]:
```
# To import the necessary python libraries and pacakages
import numpy as np
import pandas as pd
import random
from sklearn.preprocessing import PolynomialFeatures
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

1. my_regression function

Below is the my_regression function which takes the training data, the test data and the number of labels as inputs and returns the labels of the test features.

This functions calls the cross_validation function from within. The cross_validation function has a library of models (16 in this case) from which the best model for the given dataset is selected through five fold cross validation. This cross_validation function takes the same inputs as the my_regression function. It selects the best model and gives training features, training labels, test features and the regularising parameter of the selected model as the output.

The library of models used in this work has 16 models. 1 linear model, 9 polynomial models with combination of polynomial order (M = 2,3 and 4) and the regularising parameter (lamda = 0.1, 1 and 10) and 6 radial basis models with combination of dimensions (Mr = 10 and 20) and regularising parameter (lamda = 0.1, 1 and 10). A function rad_basis is defined and called from the cross_validation function inorder to the calculate the features of the radial basis.

This library can be extended in a straight forward manner to any number of models with any basis function and regularising parameter.

In [2]:
```
def my_regression(trainX, testX, noutputs):
    # Function to perform regression. Has the capability to do model
 selection by cross validation
    # Inputs:
    # trainX - training data containing both features and labels
    # testX - test data containing features
    # noutputs - number of labels in the training data
    # Output:
    # Y_pred - predictions of the test data
    X_t, Y_t, X_test, lamda = cross_validation(trainX, testX, noutput
s)
    nsiz = X_t.shape
    w = np.dot(np.dot(np.linalg.inv(lamda*np.eye(nsiz[1]) + np.dot(np
.transpose(X_t),X_t)),np.transpose(X_t)),Y_t)
    #print(w)
    Y_pred = np.dot(X_test,w)
    return Y_pred

def cross_validation(trainX, testX, noutputs):
    # Function for selecting the model by cross validation
    # Inputs:
    # trainX - training data containing both features and labels
    # testX - test data containing features
    # noutputs - number of labels in the training data
    # Outputs:
    # X_t - training feature after model selection
    # Y_t - training labels after model selection
    # X_test - testing feature after model selection

    sz = trainX.shape
    sz_t = testX.shape
    fold = 5
    ind = int(sz[0]/fold)
    avg_err = []

    # Sepearating features from labels
    Xt = trainX[:,:sz[1]-noutputs]
    Y_t = trainX[:,-noutputs:]

    # Cross validation
    for i in range(16): # Loop for the number of models
        X_t = np.transpose([np.ones(sz[0])])
        if(i==0):
            X_t = np.append(X_t,Xt,axis=1)
            lamda = 0
        elif(i==1):
            M=2
            lamda = 0.1;
            poly = PolynomialFeatures(M)
            X_t = poly.fit_transform(Xt)
        elif(i==2):
            M=2
            lamda = 1;
            poly = PolynomialFeatures(M)
            X_t = poly.fit_transform(Xt)
        elif(i==3):
```

```
            M=2
            lamda = 10;
            poly = PolynomialFeatures(M)
            X_t = poly.fit_transform(Xt)
        elif(i==4):
            M=3
            lamda = 0.1;
            poly = PolynomialFeatures(M)
            X_t = poly.fit_transform(Xt)
        elif(i==5):
            M=3
            lamda = 1;
            poly = PolynomialFeatures(M)
            X_t = poly.fit_transform(Xt)
        elif(i==6):
            M=3
            lamda = 10;
            poly = PolynomialFeatures(M)
            X_t = poly.fit_transform(Xt)
        elif(i==7):
            Mr=10
            if sz[0]<Mr:
                Mr = int(sz[0]*0.5)
            lamda =0.1
            inx = random.sample(range(sz[0]), Mr)
            mu = Xt[inx,:]
            for k in range(Mr):
                x_rad = rad_basis(Xt,mu[k,:])
                X_t = np.append(X_t,x_rad,axis=1)
        elif(i==8):
            Mr=10
            if sz[0]<Mr:
                Mr = int(sz[0]*0.5)
            lamda =1
            inx = random.sample(range(sz[0]), Mr)
            mu = Xt[inx,:]
            for k in range(Mr):
                x_rad = rad_basis(Xt,mu[k,:])
                X_t = np.append(X_t,x_rad,axis=1)
        elif(i==9):
            Mr=10
            if sz[0]<Mr:
                Mr = int(sz[0]*0.5)
            lamda =10
            inx = random.sample(range(sz[0]), Mr)
            mu = Xt[inx,:]
            for k in range(Mr):
                x_rad = rad_basis(Xt,mu[k,:])
                X_t = np.append(X_t,x_rad,axis=1)
        elif(i==10):
            Mr=20
            if sz[0]<Mr:
                Mr = int(sz[0]*0.5)
            lamda =0.1
            inx = random.sample(range(sz[0]), Mr)
            mu = Xt[inx,:]
            for k in range(Mr):
```

```
                x_rad = rad_basis(Xt,mu[k,:])
                X_t = np.append(X_t,x_rad,axis=1)
        elif(i==11):
            Mr=20
            if sz[0]<Mr:
                Mr = int(sz[0]*0.5)
            lamda =1
            inx =random.sample(range(sz[0]), Mr)
            mu = Xt[inx,:]
            for k in range(Mr):
                x_rad = rad_basis(Xt,mu[k,:])
                X_t = np.append(X_t,x_rad,axis=1)
        elif(i==12):
            Mr=20
            if sz[0]<Mr:
                Mr = int(sz[0]*0.5)
            lamda =10
            inx = random.sample(range(sz[0]), Mr)
            mu = Xt[inx,:]
            for k in range(Mr):
                x_rad = rad_basis(Xt,mu[k,:])
                X_t = np.append(X_t,x_rad,axis=1)
        elif(i==13):
            M=4
            lamda = 0.1;
            poly = PolynomialFeatures(M)
            X_t = poly.fit_transform(Xt)
        elif(i==14):
            M=4
            lamda = 1;
            poly = PolynomialFeatures(M)
            X_t = poly.fit_transform(Xt)
        elif(i==15):
            M=4
            lamda = 10;
            poly = PolynomialFeatures(M)
            X_t = poly.fit_transform(Xt)

        err = []
        for j in range(1,fold+1): # Loop for n fold cross validation
            strt = ind*(j-1)
            end = ind*j
            if(end>sz[0]):
                end = sz[0]
            X_tr = np.delete(X_t,slice(strt,end),0)
            X_vld = X_t[strt:end,:]
            Y_tr = np.delete(Y_t,slice(strt,end),0)
            Y_vld = Y_t[strt:end,:]
            nsize = X_tr.shape
            w_trail = np.dot(np.dot(np.linalg.inv(lamda*np.eye(nsize[
1]) +np.dot(np.transpose(X_tr),X_tr)),np.transpose(X_tr)),Y_tr)
            y_trail = np.dot(X_vld,w_trail)
            err = np.append(err,np.square(Y_vld-y_trail).mean(axis=0
),axis=0)
        avg_err.append(np.mean(err,axis=0))

    # Selecting the best model
```

```
    mod_ind = np.argmin(avg_err)
    #print(min(avg_err))
    X_t = np.transpose([np.ones(sz[0])])
    X_test = np.transpose([np.ones(sz_t[0])])
    if(mod_ind==0):
        lamda = 0
        X_t = np.append(X_t,Xt,axis=1)
        X_test = np.append(X_test,testX,axis=1)
        #print('Model 1 selected')  #uncomment this and the following
print lines to see which model gets selected
    elif(mod_ind==1):
        lamda = 0.1
        M=2
        poly = PolynomialFeatures(M)
        X_t = poly.fit_transform(Xt)
        X_test = poly.fit_transform(testX)
        #print('Model 2 selected')
    elif(mod_ind==2):
        lamda = 1
        M=2
        poly = PolynomialFeatures(M)
        X_t = poly.fit_transform(Xt)
        X_test = poly.fit_transform(testX)
        #print('Model 3 selected')
    elif(mod_ind==3):
        lamda = 10
        M=2
        poly = PolynomialFeatures(M)
        X_t = poly.fit_transform(Xt)
        X_test = poly.fit_transform(testX)
        #print('Model 4 selected')
    elif(mod_ind==4):
        lamda = 0.1
        M=3
        poly = PolynomialFeatures(M)
        X_t = poly.fit_transform(Xt)
        X_test = poly.fit_transform(testX)
        #print('Model 5 selected')
    elif(mod_ind==5):
        lamda = 1
        M=3
        poly = PolynomialFeatures(M)
        X_t = poly.fit_transform(Xt)
        X_test = poly.fit_transform(testX)
        #print('Model 6 selected')
    elif(mod_ind==6):
        lamda = 10
        M=3
        poly = PolynomialFeatures(M)
        X_t = poly.fit_transform(Xt)
        X_test = poly.fit_transform(testX)
        #print('Model 7 selected')
    elif(mod_ind==7):
        lamda =0.1
        Mr=10
        if sz[0]<Mr:
                Mr = int(sz[0]*0.5)
```

```
        for i in range(Mr):
            X_t = np.append(X_t,rad_basis(Xt,mu[i,:]),axis=1)
            X_test = np.append(X_test,rad_basis(testX,mu[i,:]),axis=1
)
        #print('Model 8 selected')
    elif(mod_ind==8):
        lamda =1
        Mr=10
        if sz[0]<Mr:
            Mr = int(sz[0]*0.5)
        for i in range(Mr):
            X_t = np.append(X_t,rad_basis(Xt,mu[i,:]),axis=1)
            X_test = np.append(X_test,rad_basis(testX,mu[i,:]),axis=1
)
        #print('Model 9 selected')
    elif(mod_ind==9):
        lamda =10
        Mr=10
        if sz[0]<Mr:
            Mr = int(sz[0]*0.5)
        for i in range(Mr):
            X_t = np.append(X_t,rad_basis(Xt,mu[i,:]),axis=1)
            X_test = np.append(X_test,rad_basis(testX,mu[i,:]),axis=1
)
        #print('Model 10 selected')
    elif(mod_ind==10):
        lamda =0.1
        Mr=20
        if sz[0]<Mr:
            Mr = int(sz[0]*0.5)
        for i in range(Mr):
            X_t = np.append(X_t,rad_basis(Xt,mu[i,:]),axis=1)
            X_test = np.append(X_test,rad_basis(testX,mu[i,:]),axis=1
)
        #print('Model 11 selected')
    elif(mod_ind==11):
        lamda =1
        Mr=20
        if sz[0]<Mr:
            Mr = int(sz[0]*0.5)
        for i in range(Mr):
            X_t = np.append(X_t,rad_basis(Xt,mu[i,:]),axis=1)
            X_test = np.append(X_test,rad_basis(testX,mu[i,:]),axis=1
)
        #print('Model 12 selected')
    elif(mod_ind==12):
        lamda =10
        Mr=20
        if sz[0]<Mr:
            Mr = int(sz[0]*0.5)
        for i in range(Mr):
            X_t = np.append(X_t,rad_basis(Xt,mu[i,:]),axis=1)
            X_test = np.append(X_test,rad_basis(testX,mu[i,:]),axis=1
)
        #print('Model 13 selected')
    elif(mod_ind==13):
        lamda = 0.1
```

```
        M=4
        poly = PolynomialFeatures(M)
        X_t = poly.fit_transform(Xt)
        X_test = poly.fit_transform(testX)
        #print('Model 14 selected')
    elif(mod_ind==14):
        lamda = 1
        M=4
        poly = PolynomialFeatures(M)
        X_t = poly.fit_transform(Xt)
        X_test = poly.fit_transform(testX)
        #print('Model 15 selected')
    elif(mod_ind==15):
        lamda = 10
        M=4
        poly = PolynomialFeatures(M)
        X_t = poly.fit_transform(Xt)
        X_test = poly.fit_transform(testX)
        #print('Model 16 selected')
    return X_t,Y_t,X_test,lamda


def rad_basis(X,mu):
    # Function to calculate the radial basis functions
    # Inputs:
    # X - Feature vector
    # mu - Mean vectors of the gaussian space
    s = 0.5
    tmp = X-mu
    num = np.linalg.norm(X-mu,axis=1)
    res = np.array([np.exp(np.square(num)/s)])
    res = np.transpose(res)
    return(res)
```

2. Testing my_regression function

A simple known function of the form $y = cos(x^2) + 0.1x^3 + \epsilon$, where $x \in [-5, 5]$ and ϵ is a gaussian noise, is used to test the regression function.

2.a) A plot of the function with zero noise $\epsilon = 0$ is shown below in Figure 1. This is the actual truth that we wish to predict from the regression model.

2.b) Also a plot of this function with different noise levels (variance) is shown below in Figure 2.

In [3]:
```
# Code to generate the plots

plt.rcParams.update({'font.size': 15})
x1=np.linspace(-5,5,100)
y=np.cos(x1**2)+0.1*(x1**3)
ef1=np.random.normal(0,0.1,100)
ef2=np.random.normal(0,0.2,100)
ef3=np.random.normal(0,0.5,100)
ef4=np.random.normal(0,1,100)

y1=np.cos(x1**2)+0.1*(x1**3)+ef1
y2=np.cos(x1**2)+0.1*(x1**3)+ef2
y3=np.cos(x1**2)+0.1*(x1**3)+ef3
y4=np.cos(x1**2)+0.1*(x1**3)+ef4


txt="Fig 1: Plot of the given function with zero noise"
fig = plt.figure(figsize=(9,9))
plt.plot(x1, y, 'k-')
plt.xlabel('X')
plt.ylabel('y')
plt.xlim(-5.0, 5.0,0.1)
plt.title('Given function')
fig.text(.5, .05, txt, ha='center')

txt="Fig 2: Plot of the given function with varying noise"
fig = plt.figure(figsize=(9,9))
plt.plot(x1, y, 'k-', label='True Value')
plt.xlabel('X')
plt.ylabel('y')
plt.xlim(-5.0, 5.0,0.1)
plt.plot(x1, y1, 'g-', label='Sigma=0.1')
plt.plot(x1, y2, 'r-', label='Sigma=0.2')
plt.plot(x1, y3, 'b-', label='Sigma=0.5')
plt.plot(x1, y4, 'y-', label='Sigma=1')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.title('Function with varying noise')
fig.text(.5, .05, txt, ha='center');
```

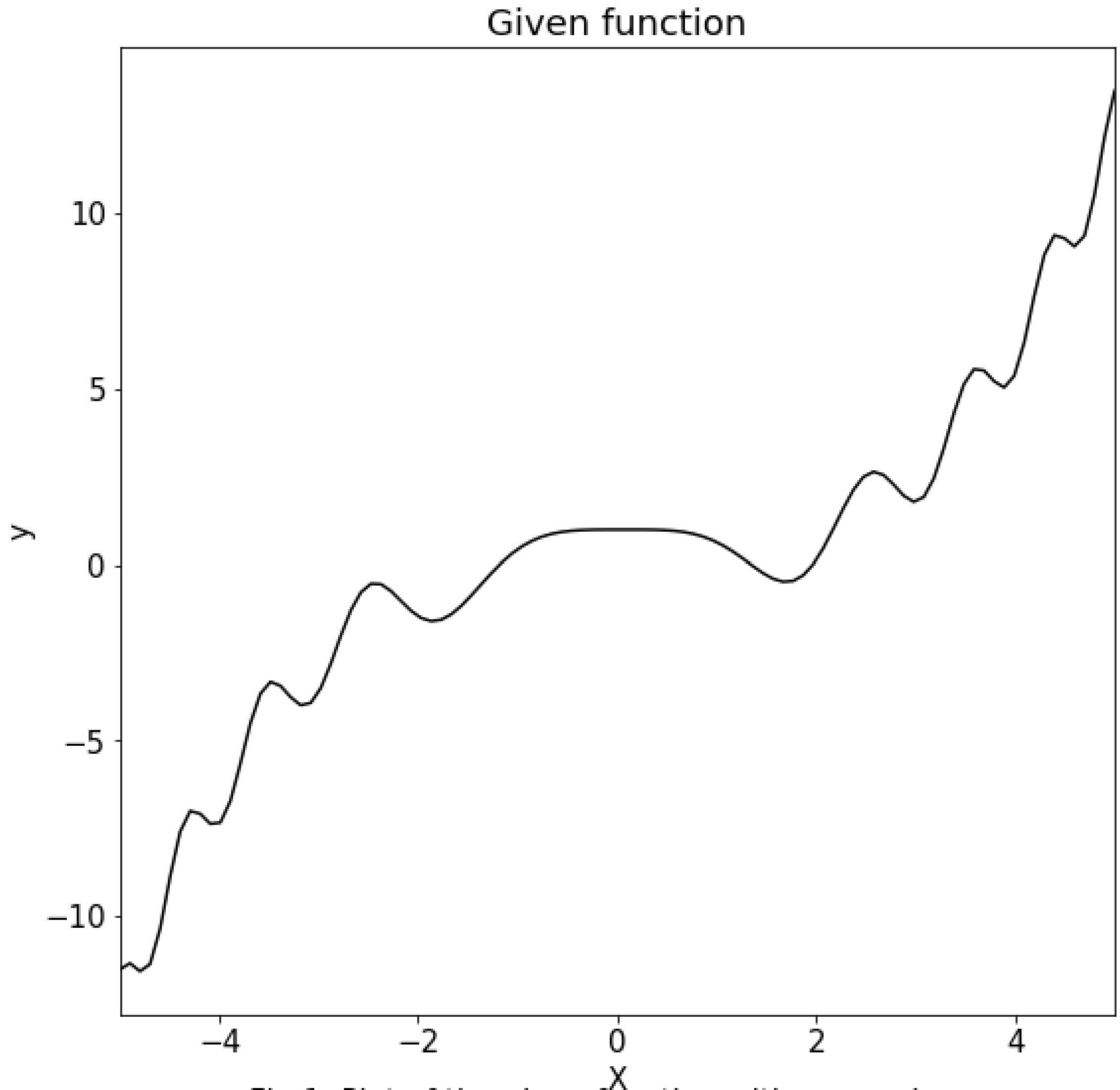

Fig 1: Plot of the given function with zero noise

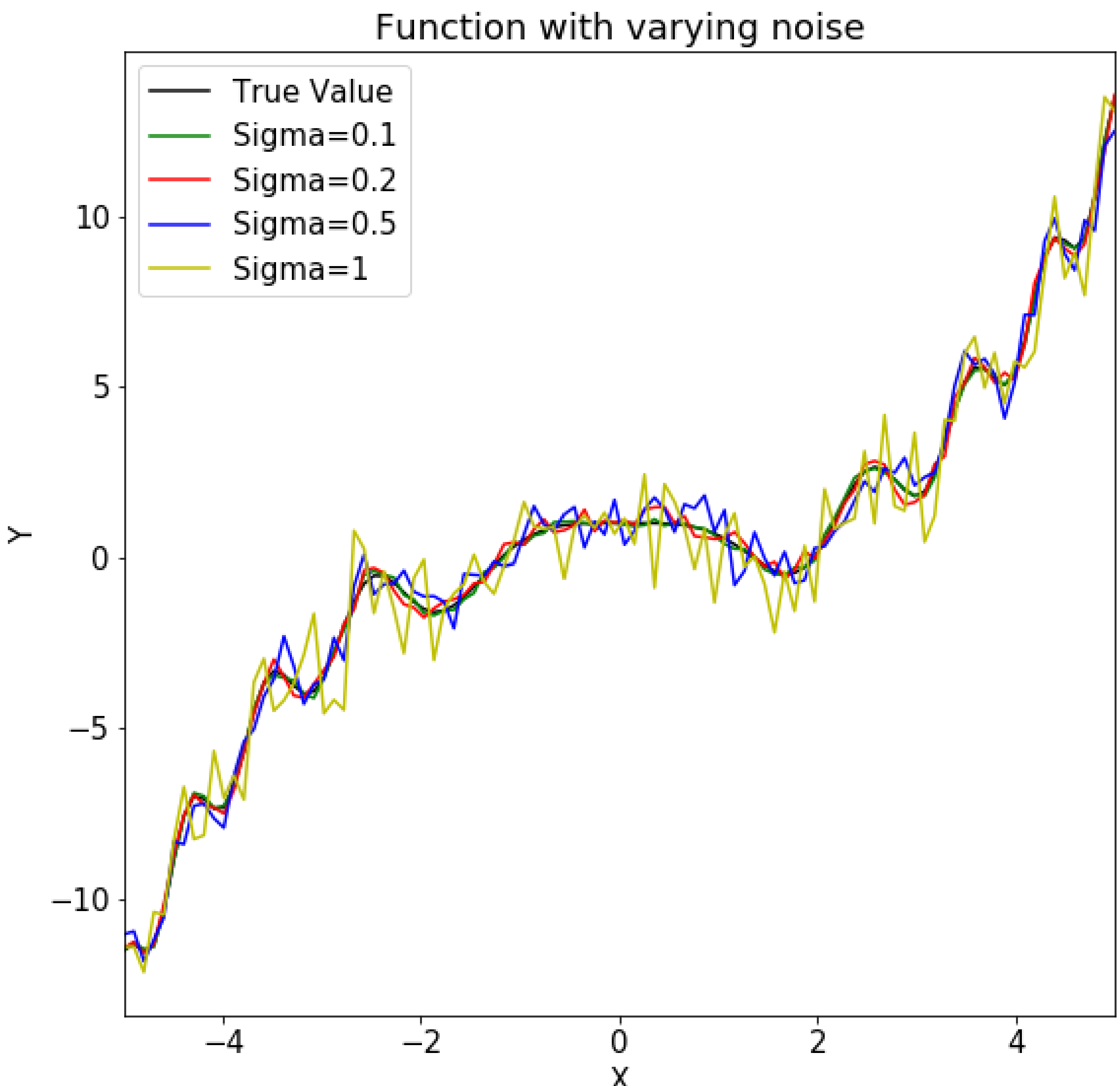

Fig 2: Plot of the given function with varying noise

2.c) In order to better understand the regression function, it is evaluated with different number of input data (t) and the varaince of noise σ^2. A table is provided below which has the mean values of the 5 fold cross validation square error for each of the combinations of t and σ

Note: Since a random function is used to generate noise and to select the means in the radial basis function, the results provided in the table and the plots for this section are not exactly reproducible.

In [4]:
```
# code to run my_regression function with different size of input dat
a and noise levels

MSE=[]
for n in [5,10,20,50,100, 200, 500]:
    for sig in [0.1,0.2,0.5,1]:
        m = int(np.ceil(n*1.25))
        x = np.linspace(-5,5,m)
        x = np.array([x])
        y_true = np.cos(x**2)+0.1*x**3
        l_y = len(y_true)
        eps = np.random.normal(0,sig,l_y)
        y = y_true+eps
        xfea = np.append(np.transpose(x),np.transpose(y),axis=1)
        np.random.shuffle(xfea)
        xtrain = xfea[:n,:]
        xt = x[:,n:]
        xtest = np.transpose(xt)
        ytest = y[:,n:]
        testY = my_regression(xtrain,xtest, 1)
        MSE = np.append(MSE,np.square(testY-ytest).mean())
        del xfea
```

In [5]:
```
# Code to generate the table

r=[39.4030,0.63097,3.42889,5.54931,1.54514357,  0.77467838, 15.002243
61, 21.97299226,  0.93581388,
        0.59820986,  1.05234081,  0.92714011,  0.54053388,  0.7839495
,
        0.75825333,  0.67054765,  0.5480848 ,  0.586359   ,  0.5514055
4,
        0.53879165,  0.48436078,  0.57505597,  0.54836955,  0.5083623
4,
        0.53116724,  0.51673972,  0.49293156,  0.50043674,  0.4831283
5,
        0.4927945 ,  0.47909638,  0.48795869]
r1=np.array(r)
r2=r1.reshape(8,4)
table=pd.DataFrame(r2,index=['t=2','t=5','t=10','t=20','t=50','t=100'
, 't=200', 't=500'],columns=['sigma=0.1','sigma=0.2','sigma=0.5','sig
ma=1'])
td_props = [
  ('font-size', '14px'),
  ('text-align', 'center')
  ]
styles = [
  dict(selector="td", props=td_props)
  ]
table.style.set_table_styles(styles).set_caption('Table 1: Table to s
how the effect of number of training data and noise. Mean of the MSE
 for 5 fold cross validation is shown.').set_properties(subset=['sigm
a=0.1','sigma=0.2','sigma=0.5','sigma=1'], **{'width': '100px'})
```

Out[5]:

Table 1: Table to show the effect of number of training data and noise. Mean of the MSE for 5 fold cross validation is shown.

	sigma=0.1	sigma=0.2	sigma=0.5	sigma=1
t=2	39.403	0.63097	3.42889	5.54931
t=5	1.54514	0.774678	15.0022	21.973
t=10	0.935814	0.59821	1.05234	0.92714
t=20	0.540534	0.783949	0.758253	0.670548
t=50	0.548085	0.586359	0.551406	0.538792
t=100	0.484361	0.575056	0.54837	0.508362
t=200	0.531167	0.51674	0.492932	0.500437
t=500	0.483128	0.492795	0.479096	0.487959

2.d) The plots below (Fig 3 to 6) show the actual function with zero noise and the learned model for the two best and two worst cases from the table above.

It is seen from the plots that the maximum errors occur when a linear model is selected as the best model from the cross validation. This is an expected result because it is clearly seen from the true plot of the function that a linear curve cannot fit the data. It is also understood that the bias is high for this linear model with low variance.

Also, the minimum errors are obtained for polynomial features of order 4 with regularisation. From the plots it is seen that the models with the minimum error does not overfit the data though the order of polynomial is high. This implies that there is a good tradeoff between bias and variance for these models.

Note: The first row of the table was ignored to plot these figures as they were generated using 2 fold cross validation. That is while considering the minimum and maximum of the error values, t=2 row was ignored

In [6]:
```
# Values of the average error of the cross validation for the best mo
del selected
t=[1.54514357,  0.77467838, 15.00224361, 21.97299226,  0.93581388,
        0.59820986,  1.05234081,  0.92714011,  0.54053388,  0.7839495
,
        0.75825333,  0.67054765,  0.5480848 ,  0.586359  ,  0.5514055
4,
        0.53879165,  0.48436078,  0.57505597,  0.54836955,  0.5083623
4,
        0.53116724,  0.51673972,  0.49293156,  0.50043674,  0.4831283
5,
        0.4927945 ,  0.47909638,  0.48795869]

#Selecting two minimum values from list
a=np.array(t)
i=np.argmin(a)
M=np.delete(a, i)
#1st minimum value
w1=np.array([0.3963612 , 0.0205494 ,-0.09759651, 0.09869194, 0.003562
77])
#2st minimum value
w2=np.array([ 0.48594705,-0.00818897,-0.09818533,0.101072 , 0.0036735
6])
np.argmin(a)
#Selecting two maximum values from list
a=np.array(t)
i=np.argmax(a)
M=np.delete(a, i)
#1st maximum value
w3=np.array([0.16231754,2.14446695])
#2st maximum value
w4=np.array([0.24100787,2.27777778])

#Plot of first minimum value
txt="Fig 3: Figure to show the learned curve vs the actual function f
or the least cross validation error"
x1=np.linspace(-5,5,100)
y=np.cos(x1**2)+0.1*(x1**3)
X=[x1/x1,x1,x1**2,x1**3,x1**4]
y_pred=w1.dot(X)

fig = plt.figure(figsize=(9,9))
plt.plot(x1, y, 'k-',label='truth')
plt.plot(x1, y_pred, 'g-',label='2nd Minimun MSE plot')
fig.text(.5, .05, txt, ha='center')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.xlim(-5.0, 5.0,0.1)
plt.title('t=500, sigma=0.5,model15')

#Plot of Second minimum value
txt="Fig 4: Figure to show the learned curve vs the actual function f
or the second least cross validation error"
x1=np.linspace(-5,5,100)
y=np.cos(x1**2)+0.1*(x1**3)
```

```
X=[x1/x1,x1,x1**2,x1**3,x1**4]
y_pred=w2.dot(X)

fig = plt.figure(figsize=(9,9))
plt.plot(x1, y, 'k-',label='truth')
plt.plot(x1, y_pred, 'm-',label='2nd Minimun MSE plot')
fig.text(.5, .05, txt, ha='center')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.xlim(-5.0, 5.0,0.1)
plt.title('t=500, sigma=0.1,model15')

# Plot of First max value
txt="Fig 5: Figure to show the learned curve vs the actual function f
or the maximum cross validation error"
x1=np.linspace(-5,5,100)
y=np.cos(x1**2)+0.1*(x1**3)
X=[x1/x1,x1]
y_pred=w3.dot(X)

fig = plt.figure(figsize=(9,9))
plt.plot(x1, y, 'k-',label='truth')
plt.plot(x1, y_pred, 'r-',label='1st Max MSE value plot')
fig.text(.5, .05, txt, ha='center')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.xlim(-5.0, 5.0,0.1)
plt.title('t=5, sigma=1, model1')

#Plot of 2nd max value
txt="Fig 6: Figure to show the learned curve vs the actual function f
or the second maximum cross validation error"
x1=np.linspace(-5,5,100)
y=np.cos(x1**2)+0.1*(x1**3)
X=[x1/x1,x1]
y_pred=w4.dot(X)

fig = plt.figure(figsize=(9,9))
plt.plot(x1, y, 'k-',label='truth')
plt.plot(x1, y_pred, 'b-',label='2nd Max MSE value plot')
fig.text(.5, .05, txt, ha='center')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.xlim(-5.0, 5.0,0.1)
plt.title('t=5, sigma=0.5,model1');
```

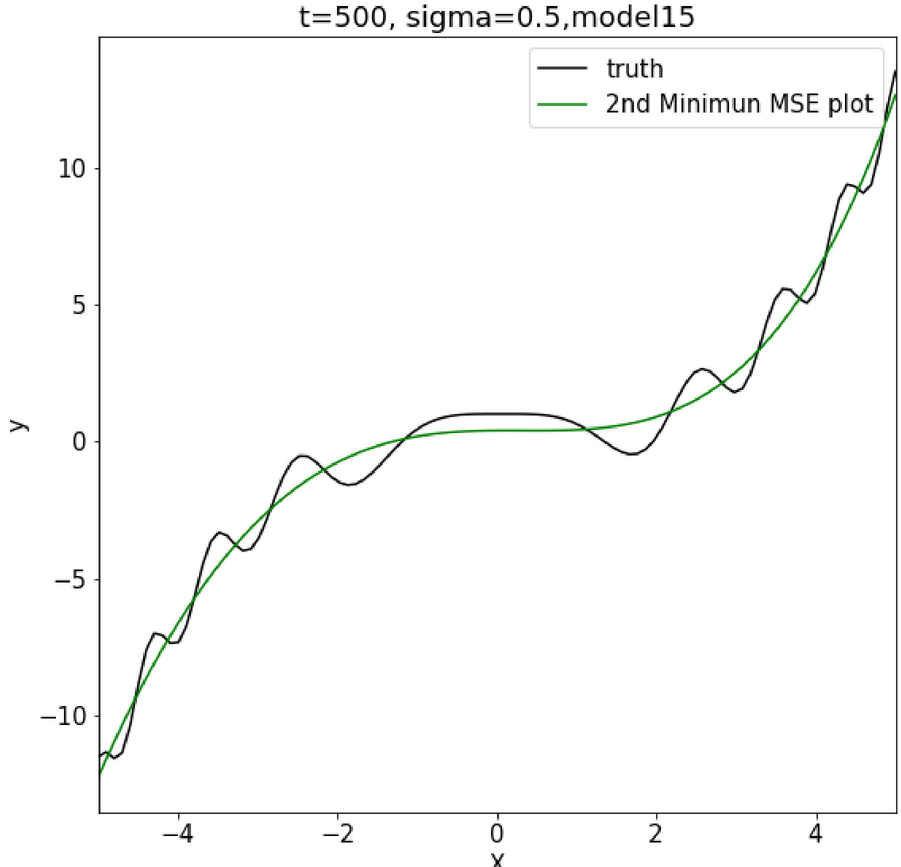

Fig 3: Figure to show the learned curve vs the actual function for the least cross validation error

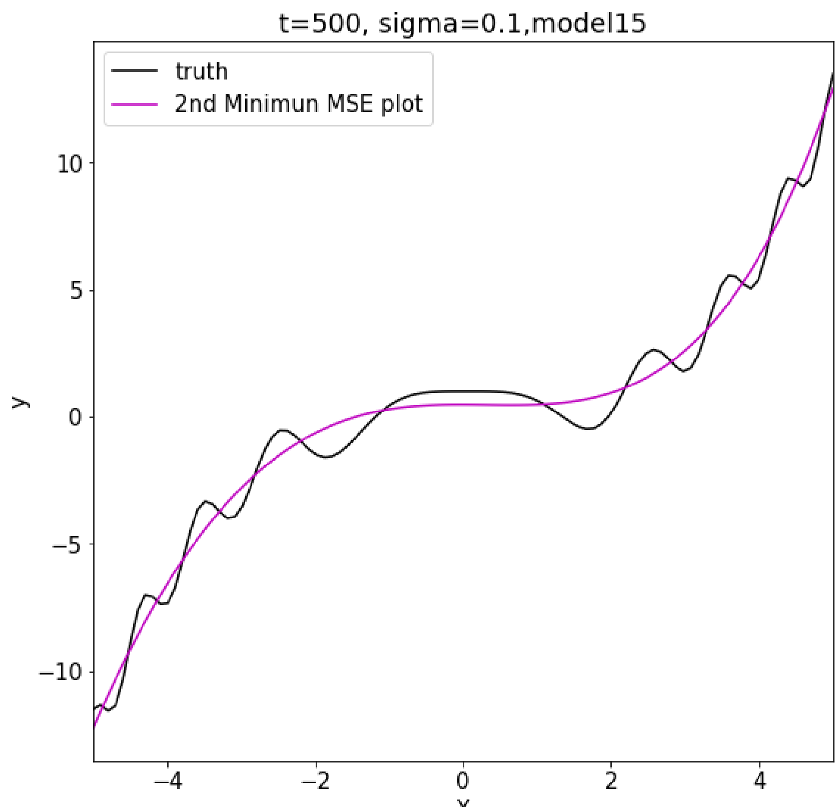

Fig 4: Figure to show the learned curve vs the actual function for the second least cross validation error

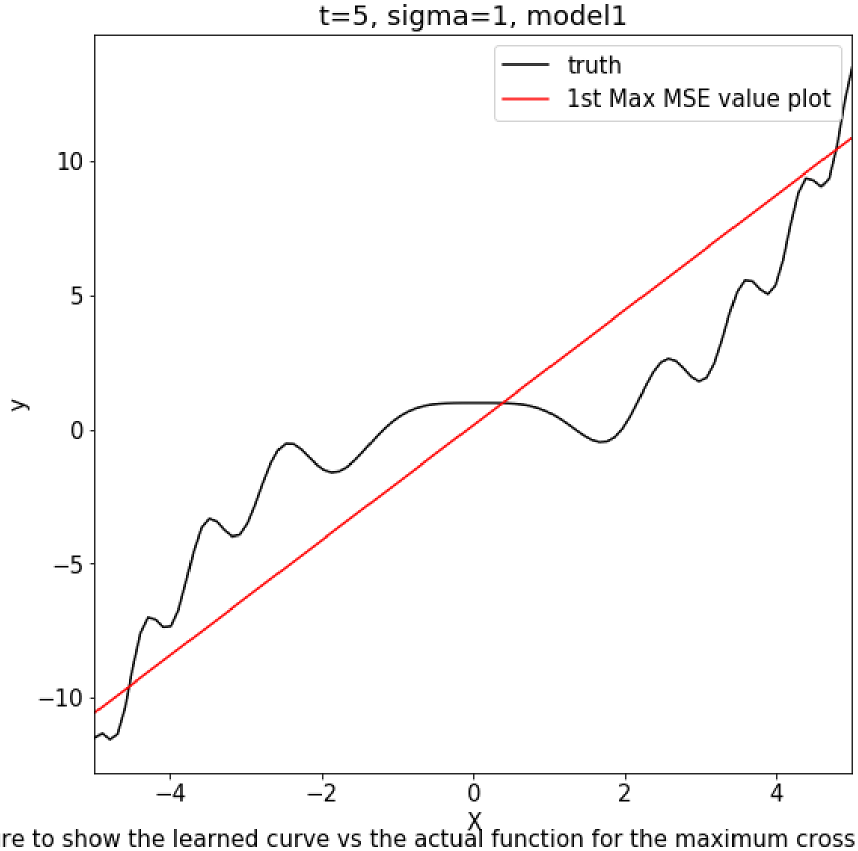

Fig 5: Figure to show the learned curve vs the actual function for the maximum cross validation error

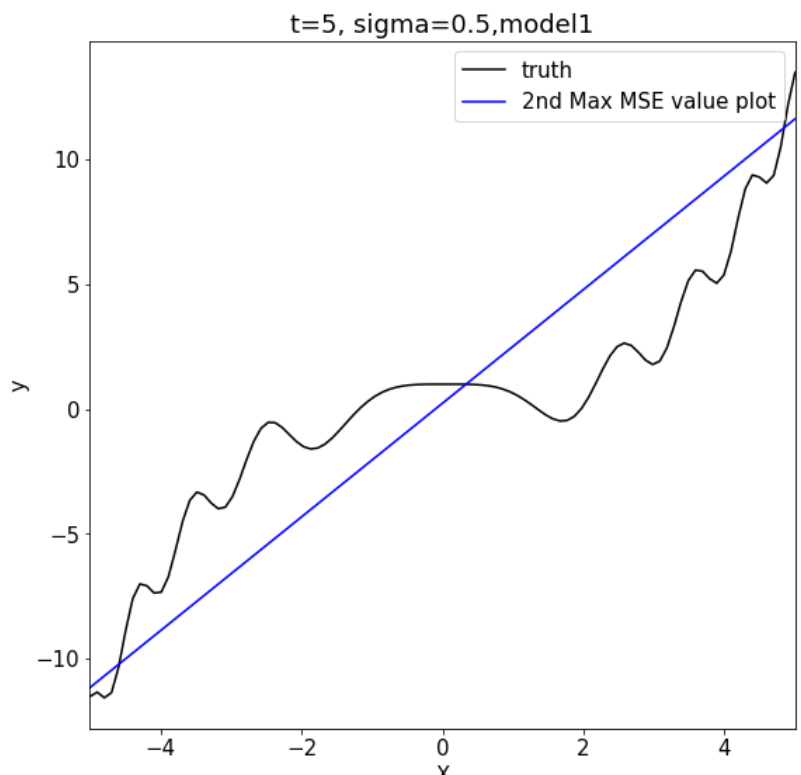

Fig 6: Figure to show the learned curve vs the actual function for the second maximum cross validation error

UCI datasets

The my_regression function is used to learn the UCI datasets and the results are presented in the following sections. It was observed that different models were selected depending on the type of dataset. Cross validation proved benefitial in model selection so as to best fit the given data. The mean squared error of the three dataset after training on the best model is presented below.

It was also noted that, the error in the test predections reduced after the data was randomly shuffled before passing it to the my_regression function. Feature scaling also improved the errors in the test predictions.

In [7]:
```
# Code to run my_regression on airfoil data

airfoil = np.loadtxt('airfoil_self_noise.dat')
np.random.shuffle(airfoil)
air_X = (airfoil-airfoil.mean(axis=0))/airfoil.std(axis=0)
air_sz = air_X.shape
air_ind = int(air_sz[0]*0.8)
air_trainx = air_X[:air_ind,:]
air_nout = 1
air_testx = air_X[air_ind+1:,:air_sz[1]-air_nout]
air_testy = air_X[air_ind+1:,-air_nout:]
air_predy = my_regression(air_trainx, air_testx, air_nout)
air_MSE = np.square(air_testy-air_predy).mean()
```

In [8]:
```
air_MSE
```

Out[8]:
```
0.17711707012665795
```

In [9]:
```
# Code to run my_regression on yacht hydrodynamics data

hydro = np.loadtxt('yacht_hydrodynamics.data')
np.random.shuffle(hydro)
hydro_X = (hydro-hydro.mean(axis=0))/hydro.std(axis=0)
hydro_sz = hydro_X.shape
hydro_ind = int(hydro_sz[0]*0.8)
hydro_trainx = hydro_X[:hydro_ind,:]
hydro_nout = 1
hydro_testx = hydro_X[hydro_ind+1:,:hydro_sz[1]-hydro_nout]
hydro_testy = hydro_X[hydro_ind+1:,-hydro_nout:]
hydro_predy = my_regression(hydro_trainx, hydro_testx, hydro_nout)
hydro_MSE = np.square(hydro_testy-hydro_predy).mean()
```

In [10]:
```
hydro_MSE
```

Out[10]:
```
0.001219101964052773
```

In [11]:
```
# Code to run my_regression on slump test data

slump = pd.read_csv('slump_test.data')
slump = np.array(slump)
np.random.shuffle(slump)
slump_X = (slump-slump.mean(axis=0))/slump.std(axis=0)
slump_sz = slump_X.shape
slump_ind = int(slump_sz[0]*0.8)
slump_trainx = slump_X[:slump_ind,:]
slump_nout = 3
slump_testx = slump_X[slump_ind+1:,:slump_sz[1]-slump_nout]
slump_testy = slump_X[slump_ind+1:,-slump_nout:]
slump_predy = my_regression(slump_trainx, slump_testx, slump_nout)
slump_MSE = np.square(slump_testy-slump_predy).mean(axis=0)
```

In [12]:
```
slump_MSE
```

Out[12]:
```
array([0.44758489, 0.35972035, 0.21065391])
```

Comparison of n-fold cross validation

Table 2 is presented below to show the varaiation in the average of the squared error in n fold cross validation by considering different values on n. It was observed that, changing n not only affected the value of the mean squared error but it also affected the best model that got selected from cross validation.

In [13]:

```
air=np.array([0.2626632 , 0.24698393, 0.24319547, 0.23299031, 0.23587
634,
       0.23157452, 0.22811835, 0.2281376 , 0.22855694])
hydro=np.array([0.07804408, 0.07581261, 0.07592678, 0.07284154, 0.049
49982,
       0.03244766, 0.03623286, 0.03962309, 0.03470172])
slump=np.array([0.74273763, 0.62844686, 0.57900707, 0.51918042, 0.611
70532,
       0.63184642, 0.61064543, 0.60515943, 0.58647856])
t=np.array([air,hydro,slump])
t1=np.transpose(t)
Table=pd.DataFrame(t1,index=['fold=2','fold=3','fold=4','fold=5','fol
d=6', 'fold=7', 'fold=8','fold=9','fold=10'],columns=['Airfoil','Hydr
odynamic','Concrete slump'])
td_props = [
  ('font-size', '14px'),
  ]
styles = [
  dict(selector="td", props=td_props)
  ]
Table.style.set_table_styles(styles).set_caption('Table 2: Table to s
how the effect of n fold cross validation').set_properties(subset=['A
irfoil','Hydrodynamic','Concrete slump'], **{'width': '150px'})
```

Out[13]:

Table 2: Table to show the effect of n fold cross validation

	Airfoil	Hydrodynamic	Concrete slump
fold=2	0.262663	0.0780441	0.742738
fold=3	0.246984	0.0758126	0.628447
fold=4	0.243195	0.0759268	0.579007
fold=5	0.23299	0.0728415	0.51918
fold=6	0.235876	0.0494998	0.611705
fold=7	0.231575	0.0324477	0.631846
fold=8	0.228118	0.0362329	0.610645
fold=9	0.228138	0.0396231	0.605159
fold=10	0.228557	0.0347017	0.586479