1)

Theta0= -2.827650

Theta1 = 0.836069

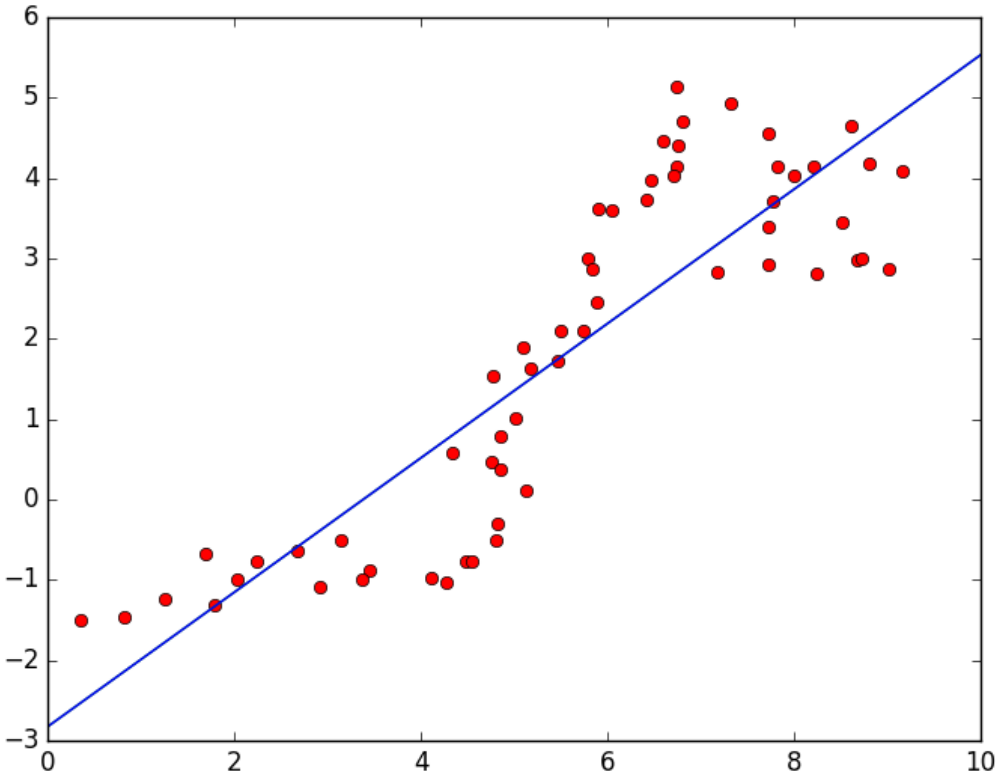
MeanSquare Training error[1.12771196]

MeanSquare Test error [2.2423492]

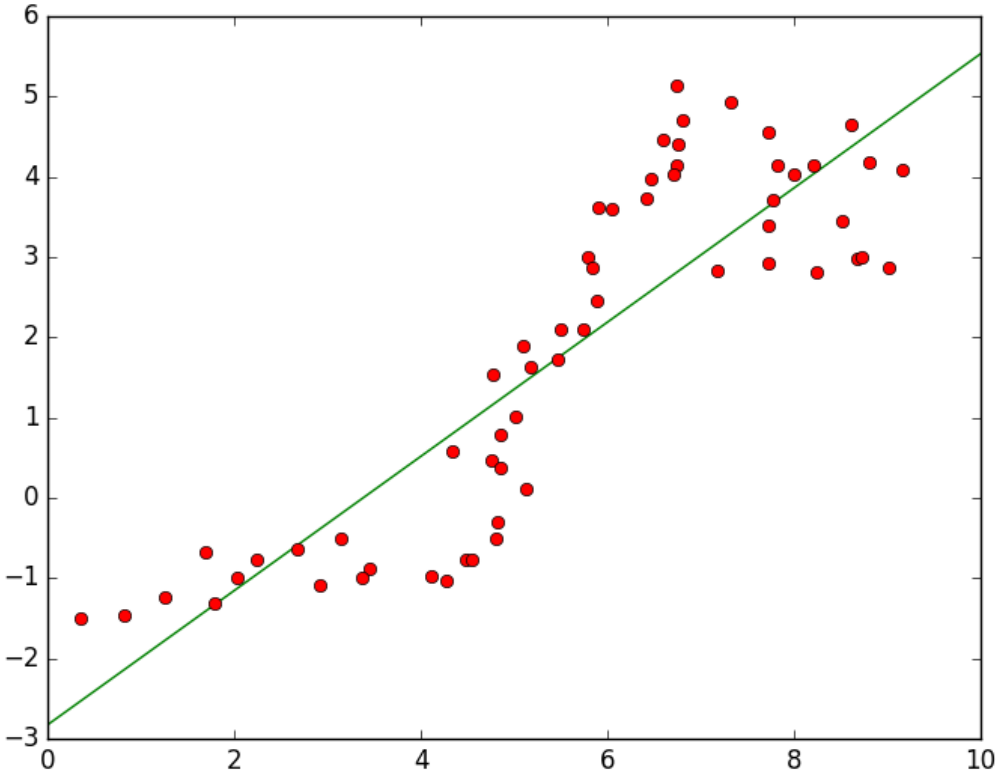
```
import numpy as np
import matplotlib.pyplot as plt
import mltools as ml



data = np.genfromtxt("../data/curve80.txt",delimiter=None)
X = data[:,0]

X = X[:,np.newaxis] # code expects shape (M,N) so make sure it's 2-dimensional
Y = data[:,1] # doesn't matter for Y
Xtr,Xte,Ytr,Yte = ml.splitData(X,Y,0.75) # split data set 75/25

lr = ml.linear.linearRegress( Xtr, Ytr ); # create and train model
xs = np.linspace(0,10,200); # densely sample possible x-values
xs = xs[:,np.newaxis]# force "xs" to be an Mx1 matrix (expected by our code)
ys = lr.predict( xs ); # make predictions at xs


th0 = lr.theta[:,0]
th1 = lr.theta[0:,1]
print( "%f %f"%(th0,th1))
plt.clf()
plt.plot(Xtr, Ytr, 'ro',color='red')
plt.plot(xs,ys,color='green')
plt.plot(xs,th0+xs*th1,color='blue')
plt.show()

Ypr = th0 + (Xtr* th1)
mse = 0

for i in range(len(Xtr)):
    mse +=((Ytr[i]-Ypr[i])*(Ytr[i]-Ypr[i]))

mse = (mse/ len(Xtr))

print(mse)

Ypr = th0 + (Xte* th1)
mse = 0

for i in range(len(Xte)):
    mse +=((Yte[i]-Ypr[i])*(Yte[i]-Ypr[i]))

mse = (mse/ len(Xte))

print(mse)

degree = [1,3,5,7,10,18]
ETr=[None]*len(degree)
ETe=[None]*len(degree)

for j in range(len(degree)):
```

```
    XtrP = ml.transforms.fpoly(Xtr, degree[j], bias=False);
    XtrP,params = ml.transforms.rescale(XtrP);
    lr = ml.linear.linearRegress( XtrP, Ytr );



    XteP = ml.transforms.fpoly(Xte, degree[j], bias=False);
    XteP, params1 = ml.transforms.rescale(XteP);

    xsP = np.linspace(0,10,200); # densely sample possible x-values
    xsP = xsP[:,np.newaxis]
    xsP = ml.transforms.fpoly(xsP, degree[j], bias=False);
    xsP,params1 = ml.transforms.rescale(xsP,params);


    ysP = lr.predict(xsP);

    Phi = lambda X: ml.transforms.rescale(ml.transforms.fpoly(X, degree, False),
params)[0]


    ETr[j] = np.mean((Ytr[:,np.newaxis] - lr.predict(XtrP))**2)

    ETe[j] = np.mean((Yte[:,np.newaxis] - lr.predict(XteP))** 2)

    plt.plot(xs,lr.predict(xsP),color='green')
    plt.plot(Xtr, Ytr, 'ro',color='red')

    plt.show()

plt.clf()

plt.xlabel('Degree')
plt.ylabel('Training & Test data error')
plt.semilogy(degree,ETr,color='red')
plt.semilogy(degree,ETe,color='green')
plt.show()
```

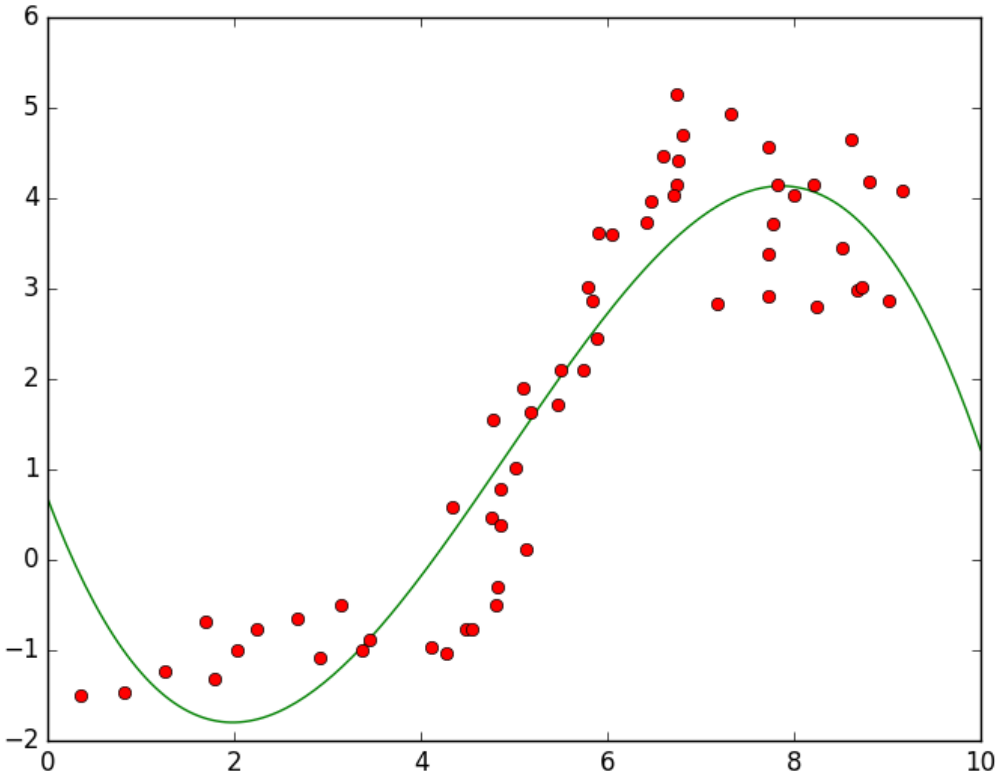
Both the blue line which is the function from the values theta0 & theta1 and the green line which is the predictor are overlapping => both are matched.



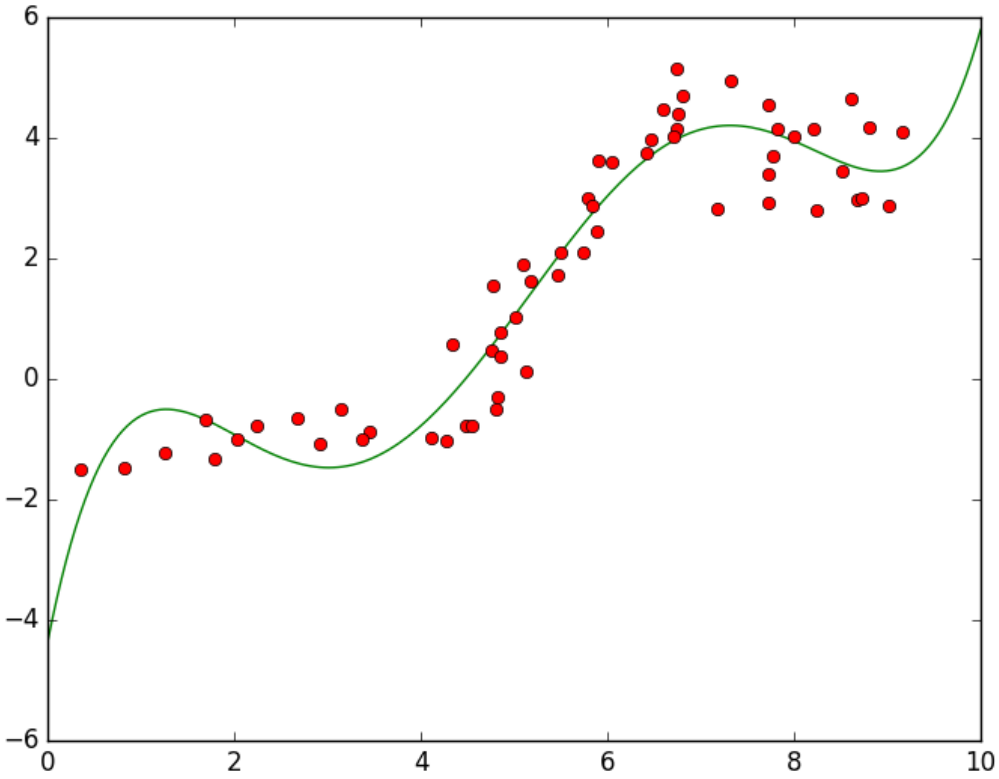
Degree= 1 for below plot



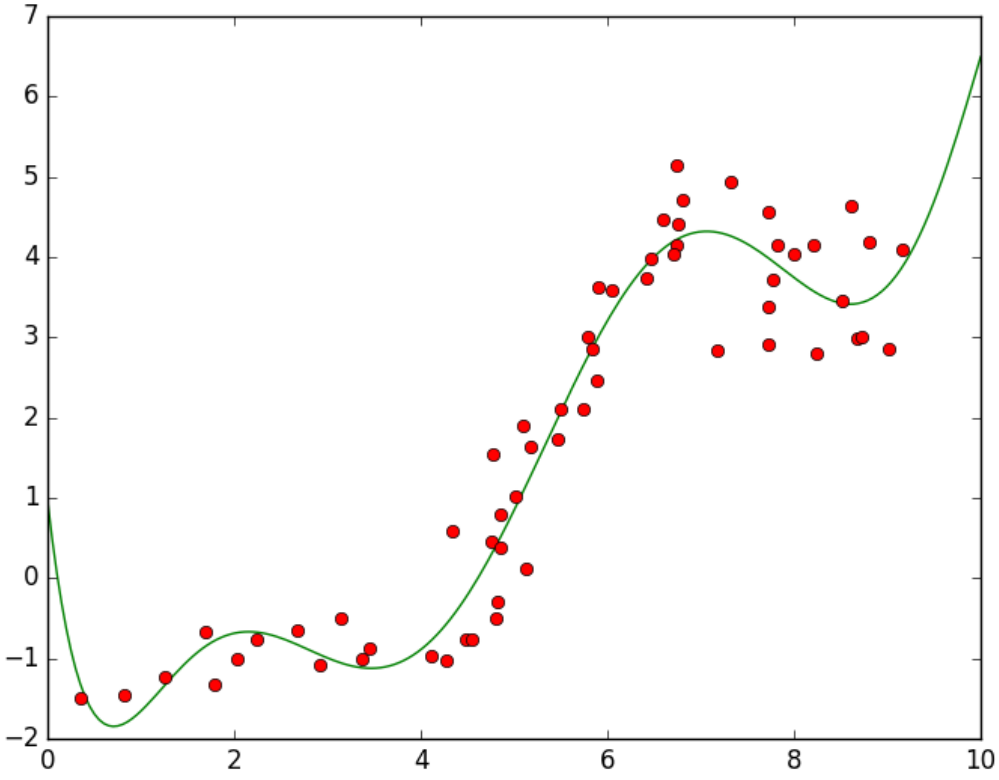
Degree= 3 for below plot



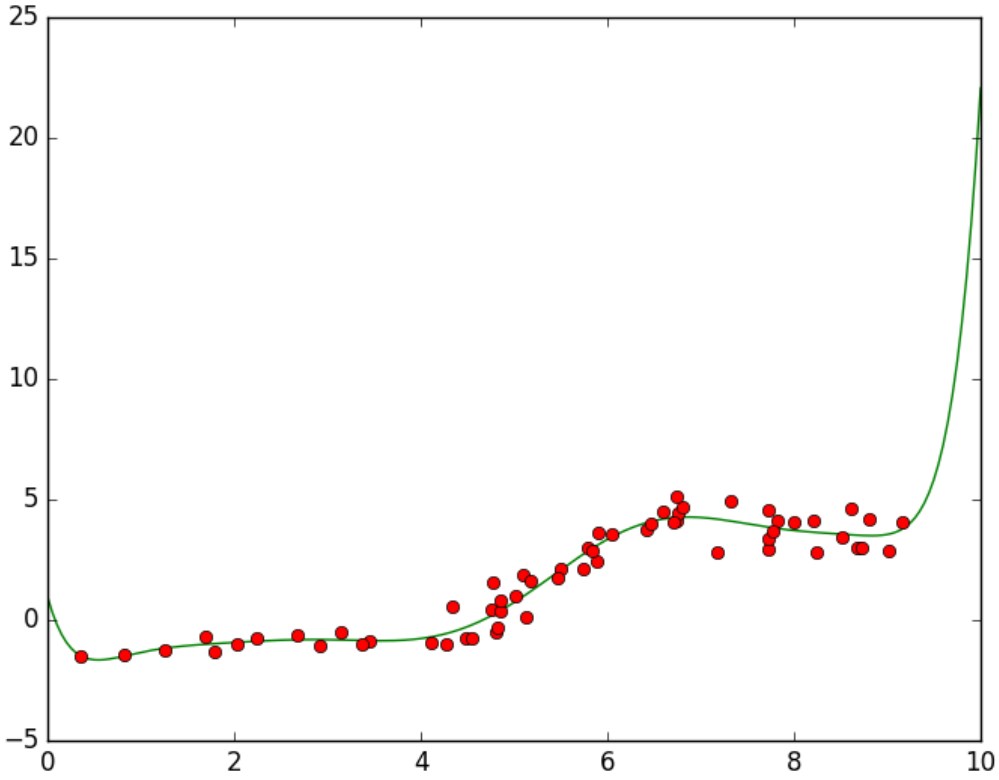
Degree= 5 for below plot



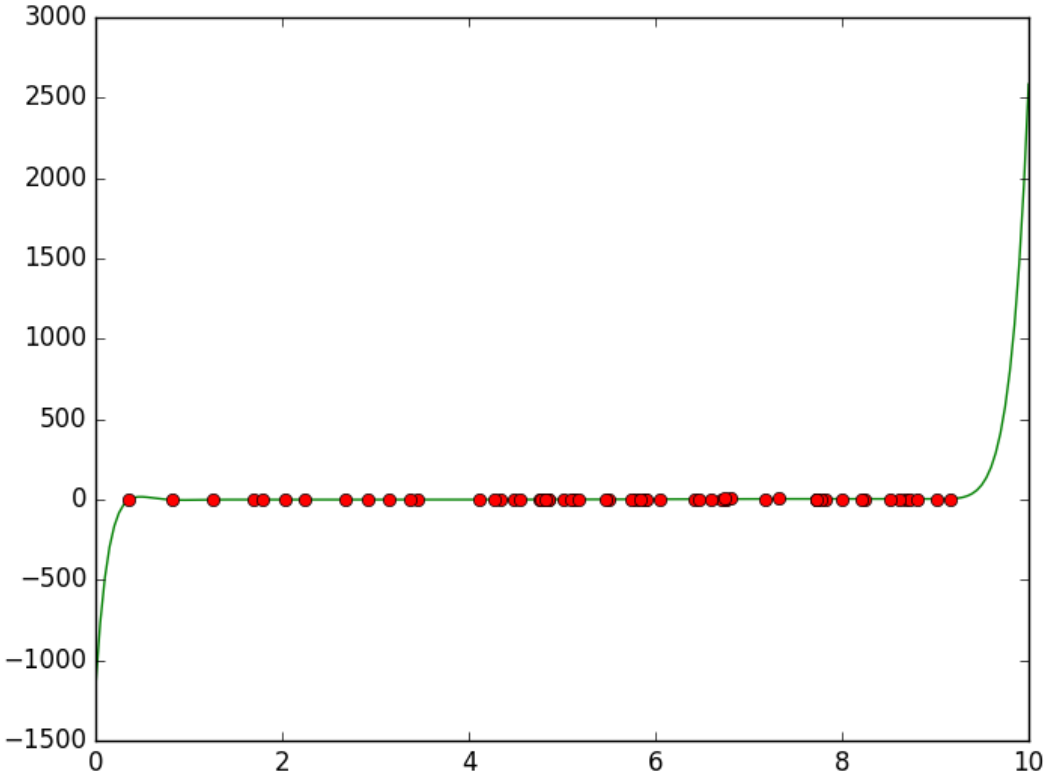
Degree= 7 for below plot

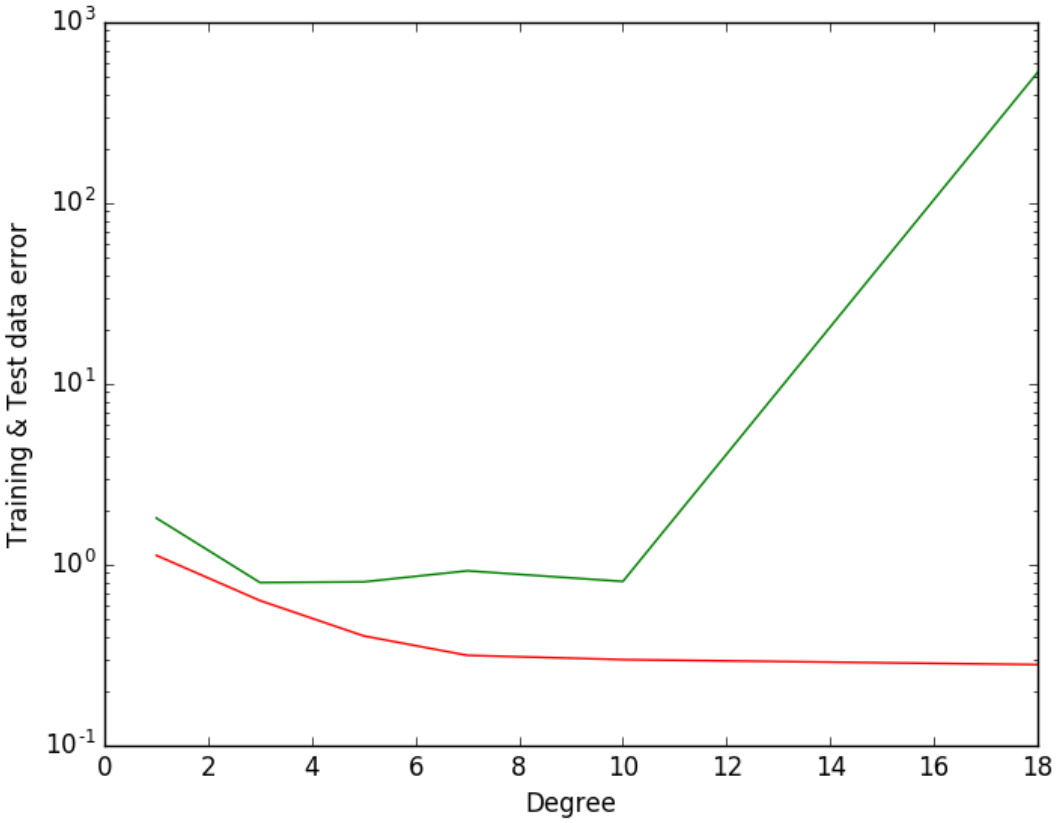


Degree= 10 for below plot



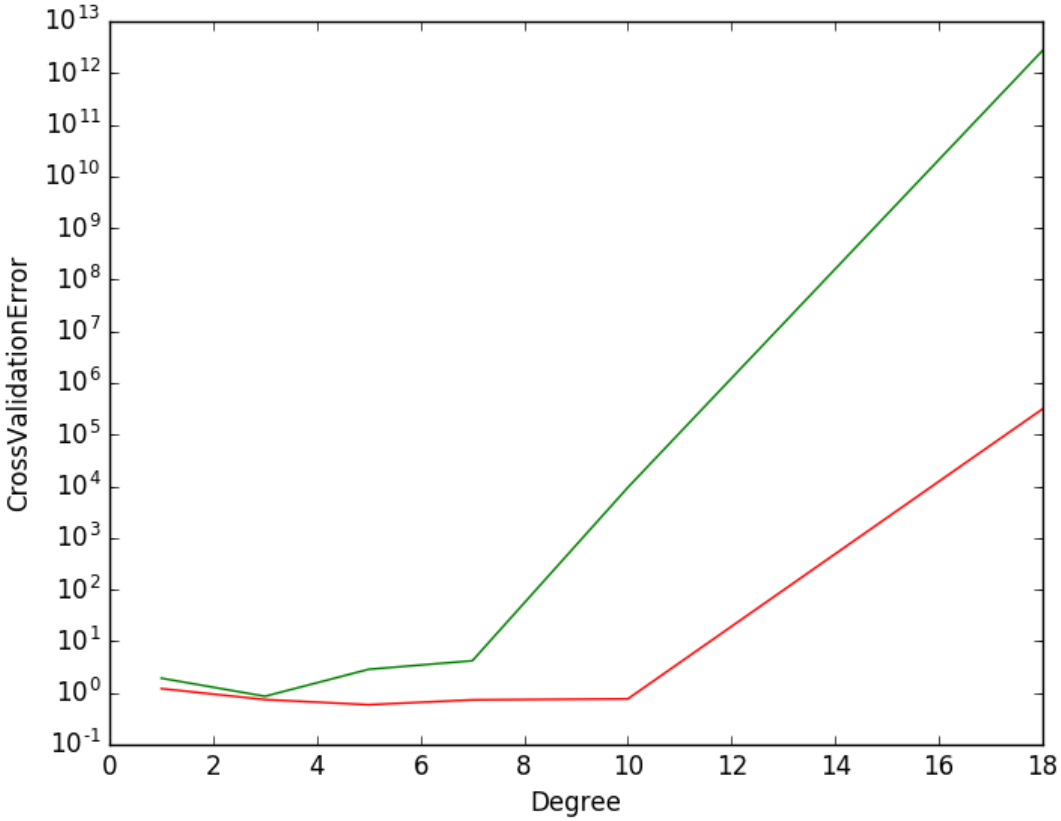
Degree= 18 for below plot


3000
2500
2000
1500
1000
500
0
−500
−1000
−1500
0
2
4
6
8
10


10^3
10^2
10^1
10^0
10^{-1}
Training & Test data error
0
2
4
6
8
10
12
14
16
18
Degree

2)

```
import numpy as np
import matplotlib.pyplot as plt
import mltools as ml

degree = [1,3,5,7,10,18]
data = np.genfromtxt("../data/curve80.txt",delimiter=None)
X = data[:,0]

X = X[:,np.newaxis] # code expects shape (M,N) so make sure it's 2-dimensional
Y = data[:,1] # doesn't matter for Y
Xtr,Xte,Ytr,Yte = ml.splitData(X,Y,0.75)

def CrossValidation(l1,l2,col):
    nFolds = 5;

    CVE= [None] * len(degree)
    for i,j in enumerate(degree):
        XtrP = ml.transforms.fpoly(l1, degree=j, bias=False)
        XtrP,params = ml.transforms.rescale(XtrP)
        J = [None] * nFolds
        for iFold in range(nFolds):
            Xti, Xvi, Yti, Yvi = ml.crossValidate(XtrP, l2, nFolds, iFold);  # take
ith data block as validation
            learner = ml.linear.linearRegress(Xti,Yti) # TODO: train on Xti, Yti ,
the data for this fold
            J[iFold] = np.mean((Yvi[:, np.newaxis] - learner.predict(Xvi)) ** 2)
        CVE[i] = np.mean(J)
        print(CVE)
        print(i)
    plt.semilogy(degree, CVE, col)

CrossValidation(Xtr, Ytr, 'r')
CrossValidation(Xte, Yte, 'g')

plt.xlabel('Degree')
plt.ylabel('CrossValidationError')

plt.show()
```


CrossValidationError
Degree
10^{13}
10^{12}
10^{11}
10^{10}
10^{9}
10^{8}
10^{7}
10^{6}
10^{5}
10^{4}
10^{3}
10^{2}
10^{1}
10^{0}
10^{-1}
0
2
4
6
8
10
12
14
16
18