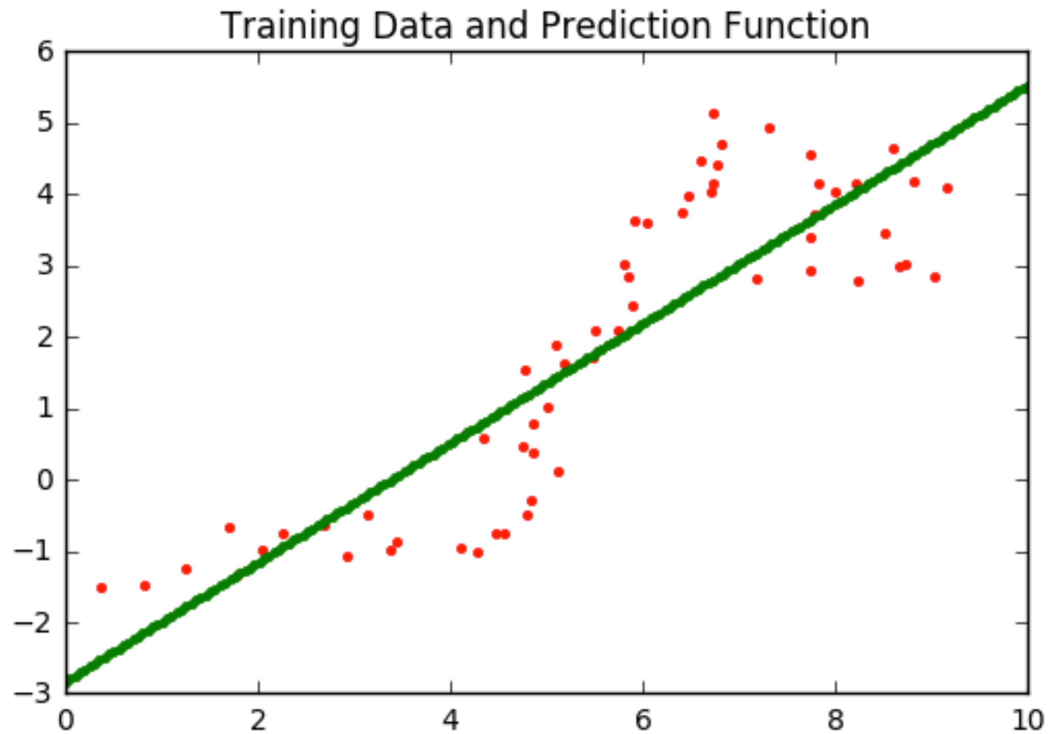
```python
#Jose Luis Vargas

import numpy as np
import matplotlib.pyplot as plt
import mltools as ml
np.random.seed(0)
```

Problem 1A

```python
data = np.genfromtxt("data/curve80.txt",delimiter=None)
X = data[:,0] #Scalar feature

X = X[:,np.newaxis] # code expects shape (M,N) so make sure it's 2-dimensional

Y = data[:,1] #Target value.

Xtr,Xte,Ytr,Yte = ml.splitData(X,Y,0.75)
#print(Xtr) #This is just one feature because there is only one column
#From my understanding X has the features and Y has the values necessary for regression.
```

Problem 1B

```
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)
#print(xs)
ys = lr.predict( xs ); # make predictions at xs
#Xtr and Ytr = Training data
#xs and ys = Prediction function
plt.title("Training Data and Prediction Function")
plt.plot(Xtr,Ytr,'r.',xs,ys,'g.')
plt.show()

#The xs and ys is used to make the prediction function which in this case is linear because the degree = 1 and from
#the graph we can see that it doesn't accuratlt match the training data points.




#Prediction Function = Green
#Training Data = Red
```




```
#The linear regression coefficient.
print (lr.theta)

#we can see that the coefficient do match because 0.836 is the slope of the line and the -2.827 is the intersection
# in the y -axis. so the linear regression function would look something like f(X) = 0.836X - 2.8
```

```
[[-2.82765049  0.83606916]]
```

```
print ("Mean Square Error for Training Data = {}".format(lr.mse(Xtr,Ytr)))
print ("Mean Square Error for Testing Data = {}".format(lr.mse(Xte,Yte)))
```

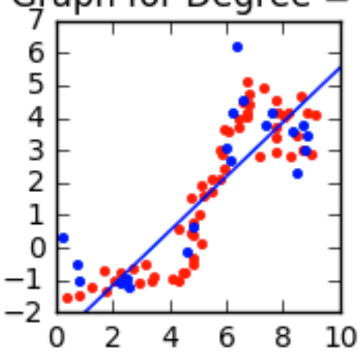
```
Mean Square Error for Training Data = 1.12771195561
Mean Square Error for Testing Data = 2.24234920301
```

Problem 1C

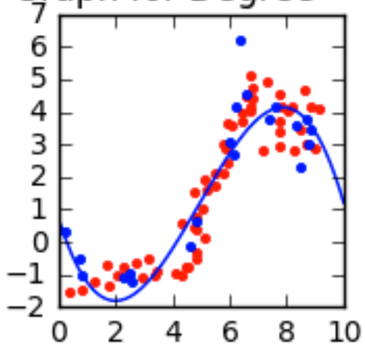
```
d = [1, 3, 5, 7, 10, 18]
errTrain = [0,0,0,0,0,0]
errVal = [0,0,0,0,0,0]

i=0
for degree in d:
    #This is for the TRAINING DATA
    XtrP = ml.transforms.fpoly(Xtr, degree, bias=False)
    #print(XtrP) #I printed this to see the increase of columns in the data, which demontrates the increase of features
                #by the value of degrees.
    XtrP,params = ml.transforms.rescale(XtrP)
    lr = ml.linear.linearRegress( XtrP, Ytr )
    YhatTrain = lr.predict(XtrP)



    #This is for the TESTING DATA
    XteP = ml.transforms.rescale( ml.transforms.fpoly(Xte,degree,bias = False),params)[0]
    lr1 = ml.linear.linearRegress(XteP, Yte)
    YhatVal = lr1.predict(XteP)
```

```python
#This is for the XS
xsP = ml.transforms.rescale( ml.transforms.fpoly(xs,degree, bias= False),params)[0]
ysP = lr.predict(xsP)



#Graphing
plt.subplot(236)
plt.title("Graph for Degree = {}".format(degree))
plt.plot(Xtr,Ytr,'r.' ,Xte,Yte,'b.')
ax = plt.axis()
plt.plot(xs,ysP)
plt.axis(ax)
plt.show()


#This is to get the Training error and test error.
errTrain[i] = lr.mse(XtrP,Ytr)
errVal[i] = lr1.mse(XteP,Yte)
i+=1
```

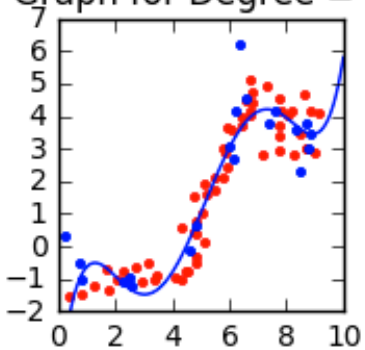
Graph for Degree = 1



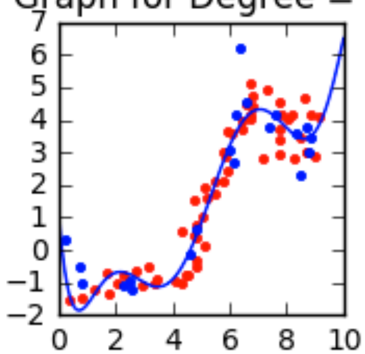
Graph for Degree = 3

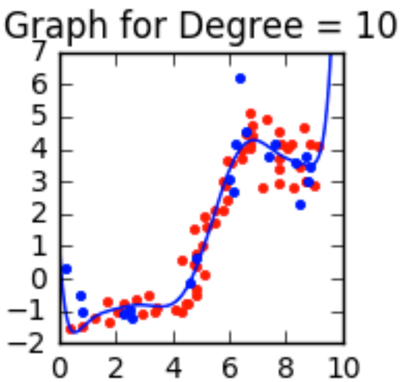


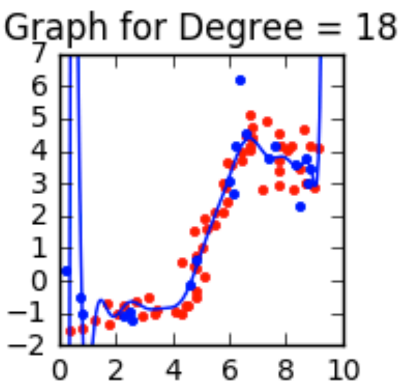
Graph for Degree = 5



Graph for Degree = 7



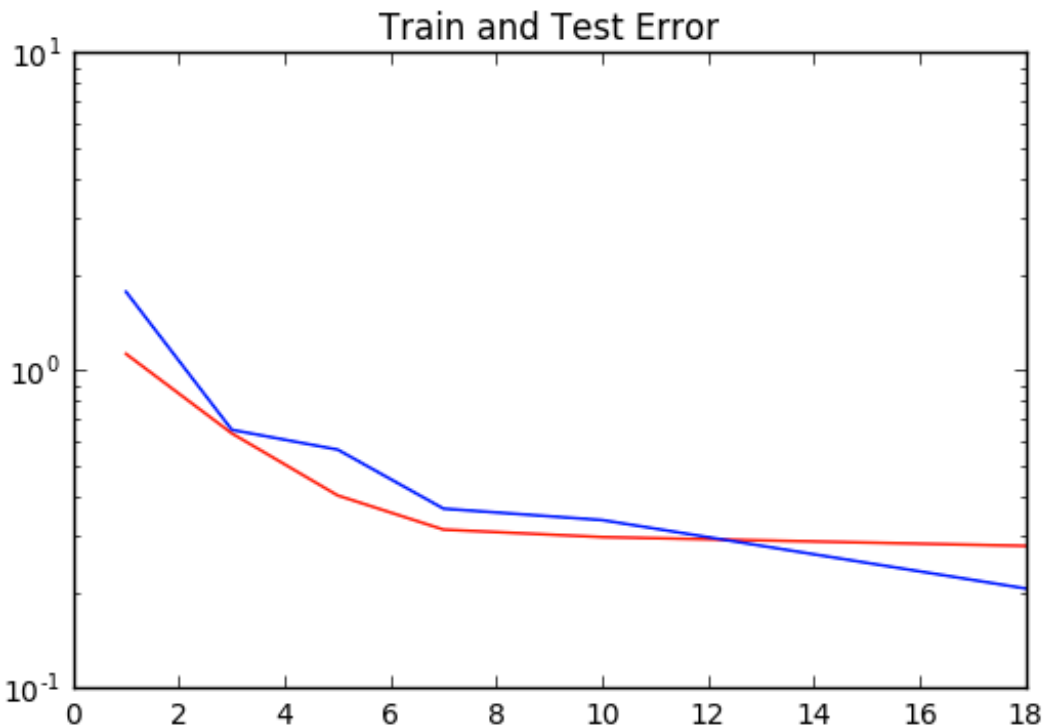





```
#for the degree graphs. We can say that degree 10 and 7 make a good prediction function to fit the data

print errTrain
print errVal    #Checking the dimmension.
```

```
[1.1277119556093909, 0.63396520631196451, 0.40424894644591752, 0.31563467398935735, 0.29894797966804609, 0.2804773728
9161084]
[1.7689831037367569, 0.65033260336933119, 0.56354562825259591, 0.36735238350481603, 0.3381778035453179, 0.20596639842
599643]
```

```
plt.title("Train and Test Error")
plt.semilogy(d, errTrain,'r', d,errVal,'b')
plt.show()

#Train Error = RED
#Test Error = BLUE
```



```python
#Jose Lusi Vargas
import numpy as np
import matplotlib.pyplot as plt
import mltools as ml
np.random.seed(0)
```

Problem 2

```python
data = np.genfromtxt("data/curve80.txt",delimiter=None)
X = data[:,0] #Scalar feature

X = X[:,np.newaxis] # code expects shape (M,N) so make sure it's 2-dimensional

Y = data[:,1] #Target value.
```

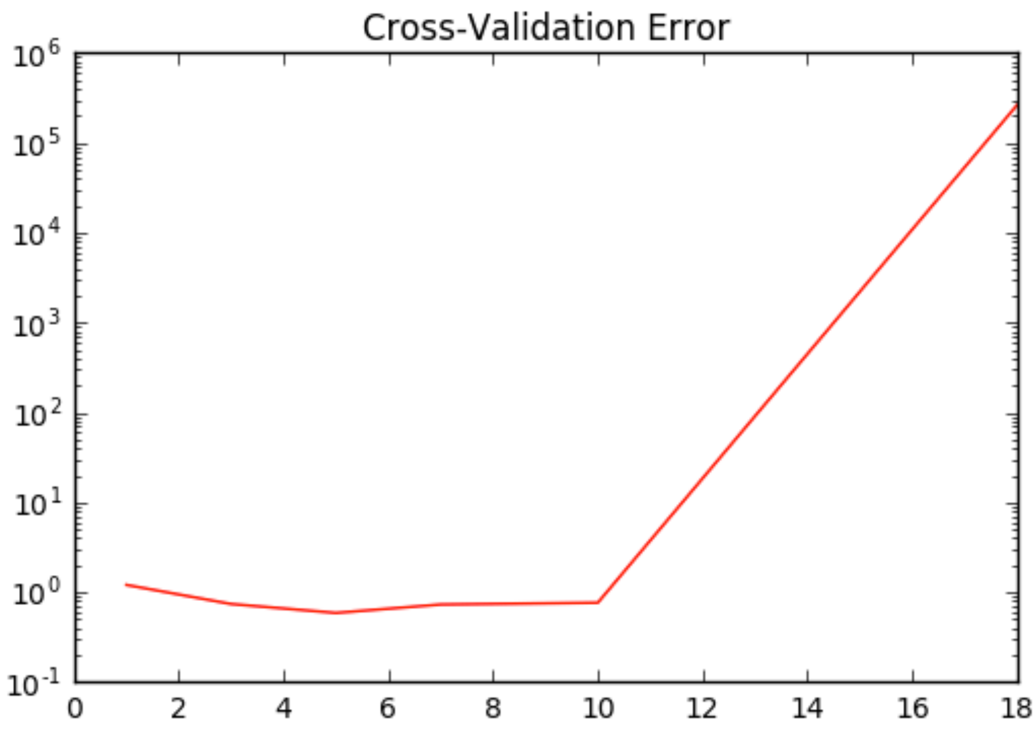
```python
Xtr,Xte,Ytr,Yte = ml.splitData(X,Y,0.75)
```

```python
d = [1, 3, 5, 7, 10, 18]
J=[0,0,0,0,0]
cv_error = [0,0,0,0,0,0]
```

```python
#For Degree = 1
nFolds = 5
XtrP = ml.transforms.fpoly(Xtr, 1, bias=False)
XtrP = ml.transforms.rescale(XtrP)[0]
lr = ml.linear.linearRegress( XtrP, Ytr )
YhatTrain = lr.predict(XtrP)

for iFold in range(nFolds):
    Xti,Xvi,Yti,Yvi = ml.crossValidate(XtrP,Ytr,nFolds,iFold); # we cross validate using the XtrP
    learner = ml.linear.linearRegress(Xti,Yti) # TODO: train on Xti, Yti , the data for this fold
    J[iFold] = learner.mse(Xvi, Yvi) # TODO: now compute the MSE on Xvi, Yvi and save it
cv_error[0] = np.mean(J)



#For Degree = 3
nFolds = 5
XtrP = ml.transforms.fpoly(Xtr, 3, bias=False)
XtrP = ml.transforms.rescale(XtrP)[0]
lr = ml.linear.linearRegress( XtrP, Ytr )
YhatTrain = lr.predict(XtrP)

for iFold in range(nFolds):
    Xti,Xvi,Yti,Yvi = ml.crossValidate(XtrP,Ytr,nFolds,iFold); # we cross validate using the XtrP
    learner = ml.linear.linearRegress(Xti,Yti) # TODO: train on Xti, Yti , the data for this fold
    J[iFold] = learner.mse(Xvi, Yvi) # TODO: now compute the MSE on Xvi, Yvi and save it
cv_error[1] = np.mean(J)
```

```python
#For Degree = 5
nFolds = 5
XtrP = ml.transforms.fpoly(Xtr, 5, bias=False)
XtrP = ml.transforms.rescale(XtrP)[0]
lr = ml.linear.linearRegress( XtrP, Ytr )
YhatTrain = lr.predict(XtrP)

for iFold in range(nFolds):
    Xti,Xvi,Yti,Yvi = ml.crossValidate(XtrP,Ytr,nFolds,iFold); # we cross validate using the XtrP
    learner = ml.linear.linearRegress(Xti,Yti) # TODO: train on Xti, Yti , the data for this fold
    J[iFold] = learner.mse(Xvi, Yvi) # TODO: now compute the MSE on Xvi, Yvi and save it
cv_error[2] = np.mean(J)




#For Degree = 7
nFolds = 5
XtrP = ml.transforms.fpoly(Xtr, 7, bias=False)
XtrP = ml.transforms.rescale(XtrP)[0]
lr = ml.linear.linearRegress( XtrP, Ytr )
YhatTrain = lr.predict(XtrP)

for iFold in range(nFolds):
    Xti,Xvi,Yti,Yvi = ml.crossValidate(XtrP,Ytr,nFolds,iFold); # we cross validate using the XtrP
    learner = ml.linear.linearRegress(Xti,Yti) # TODO: train on Xti, Yti , the data for this fold
    J[iFold] = learner.mse(Xvi, Yvi) # TODO: now compute the MSE on Xvi, Yvi and save it
cv_error[3] = np.mean(J)
```

```python
#For Degree = 10
nFolds = 5
XtrP = ml.transforms.fpoly(Xtr, 10, bias=False)
XtrP = ml.transforms.rescale(XtrP)[0]
lr = ml.linear.linearRegress( XtrP, Ytr )
YhatTrain = lr.predict(XtrP)

for iFold in range(nFolds):
    Xti,Xvi,Yti,Yvi = ml.crossValidate(XtrP,Ytr,nFolds,iFold); # we cross validate using the XtrP
    learner = ml.linear.linearRegress(Xti,Yti) # TODO: train on Xti, Yti , the data for this fold
    J[iFold] = learner.mse(Xvi, Yvi) # TODO: now compute the MSE on Xvi, Yvi and save it
cv_error[4] = np.mean(J)



#For Degree = 18
nFolds = 5
XtrP = ml.transforms.fpoly(Xtr, 18, bias=False)
XtrP = ml.transforms.rescale(XtrP)[0]
lr = ml.linear.linearRegress( XtrP, Ytr )
YhatTrain = lr.predict(XtrP)

for iFold in range(nFolds):
    Xti,Xvi,Yti,Yvi = ml.crossValidate(XtrP,Ytr,nFolds,iFold); # we cross validate using the XtrP
    learner = ml.linear.linearRegress(Xti,Yti) # TODO: train on Xti, Yti , the data for this fold
    J[iFold] = learner.mse(Xvi, Yvi) # TODO: now compute the MSE on Xvi, Yvi and save it
cv_error[5] = np.mean(J)

# the overall estimated validation performance is the average of the performance on each fold
```



]:
```
#Which degree has the minimum cross-validation error?

# ANSWER: minimum cross-Validation occurs when degree = 5 (Lowest point in the graph) then there is also degree 10.
#after degree =10 the cross validation error increases drastically.

#How does its MSE estimated from cross-validation compare to its MSE evaluated on the actual test data?

# ANSWER: The MSE of cross_validation is minimal at degree = 5. However the MSE for the actual test data, decreases
#as it approaches degree = 18. This means that for cossvalidation the best performance and most accurate prediction
#function occurs when degree = 5.
```