In [3]:

```
import numpy as np
import matplotlib.pyplot as plt
import mltools as ml
data = np.genfromtxt("curve80.txt",delimiter=None) # load the text file
X = data[:,0]
X = np.atleast_2d(X).T # code expects shape (M,N) so make sure it's 2-dimensional
Y = data[:,1]
Xtr,Xte,Ytr,Yte = ml.splitData(X,Y,0.75) # split data set 75/25
```

Problem 1.1:

In [12]:

```
print(Xtr.shape)
print(Xte.shape)
print(Ytr.shape)
print(Yte.shape)
```

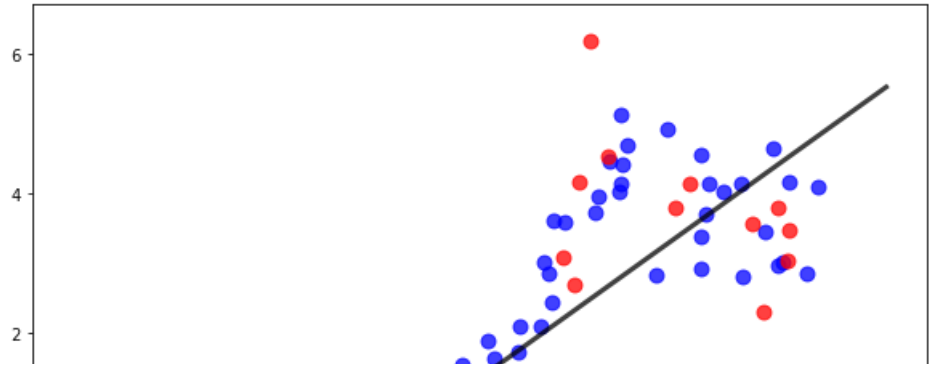
```
(60, 1)
(20, 1)
(60,)
(20,)
```

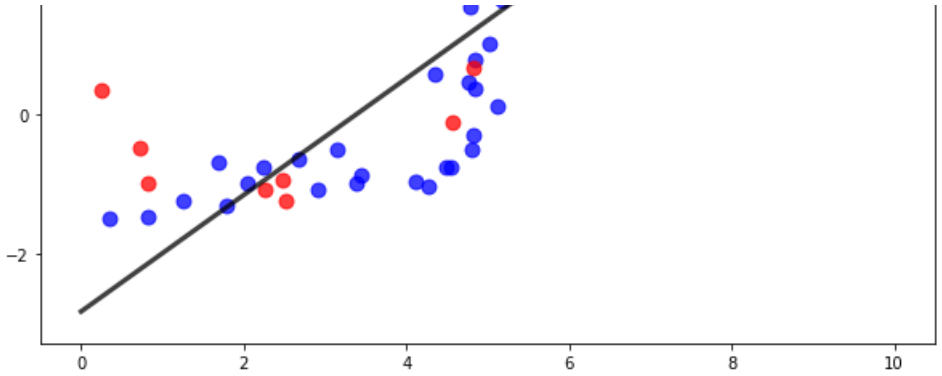
Problem 1.2:

In [24]:

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

f, ax = plt.subplots(1, 1, figsize=(10, 8))

ax.scatter(Xtr, Ytr, s=80, color='blue', alpha=0.75, label='Train')
ax.scatter(Xte, Yte, s=80, color='red', alpha=0.75, label='Test')

ax.plot(xs, ys, lw = 3, color = 'black', alpha = 0.75, label = 'Prediction')

s = str(lr.theta)
print("Linear Regression Coefficient: " + s)

print("Mean squared error of the predictions: ")
print("  On training data: " + str(lr.mse(Xtr, Ytr)))
print("  On test data: " + str(lr.mse(Xte, Yte)))
```

```
Linear Regression Coefficient: [[-2.82765049  0.83606916]]
Mean squared error of the predictions:
  On training data: 1.127711955609391
  On test data: 2.2423492030101246
```





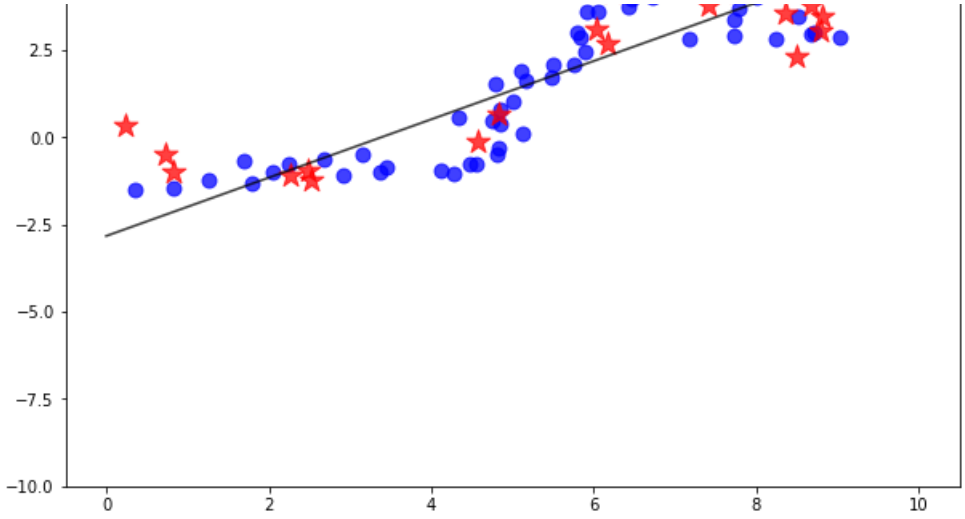
Problem 1.3:

1.3a:

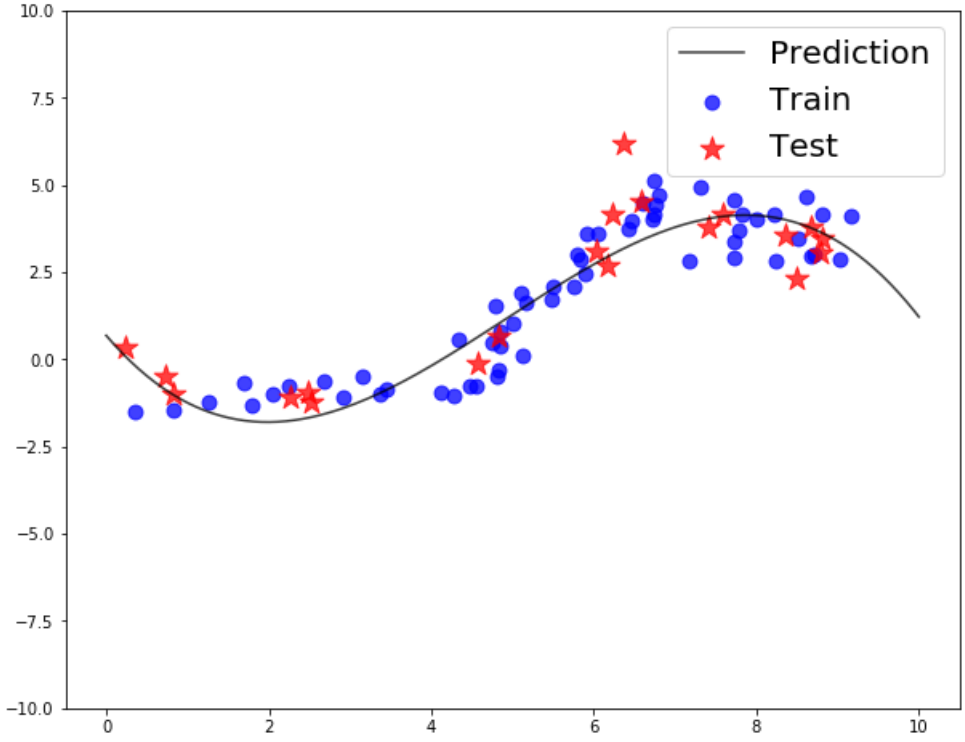
In [183]:

```
Xtr2 = np.zeros( (Xtr.shape[0],2) ) # create Mx2 array to store features
Xtr2[:,0] = Xtr[:,0] # place original "x" feature as X1
Xtr2[:,1] = Xtr[:,0]**2 # place "x^2" feature as X2
# Now, Xtr2 has two features about each data point: "x" and "x^2"
mseTraining, mseTest = [], []
deg = [1, 3, 5, 7, 10, 15, 18]
index = 0
for degree in deg:
    print('Degree: ' + str(degree))
    # Create polynomial features up to "degree"; don't create constant feature
    # (the linear regression learner will add the constant feature automatically)
    XtrP = ml.transforms.fpoly(Xtr, degree, bias=False)
    # Rescale the data matrix so that the features have similar ranges / variance
    XtrP,params = ml.transforms.rescale(XtrP)
    # "params" returns the transformation parameters (shift & scale)
    # Then we can train the model on the scaled feature matrix:
    lr = ml.linear.linearRegress( XtrP, Ytr ) # create and train model
    # Now, apply the same polynomial expansion & scaling transformation to Xtest:
    XteP,_ = ml.transforms.rescale( ml.transforms.fpoly(Xte,degree,False), params)
    # Plotting the data

    Xs, _ = ml.transforms.rescale( ml.transforms.fpoly(xs,degree,False), params)
    f, ax = plt.subplots(1, 1, figsize=(10, 8))
    XsHat = lr.predict(Xs)
    ax.scatter(Xtr, Ytr, s=80, color='blue', alpha=0.75, label='Train')
    ax.scatter(Xte, Yte, s=240, marker='*', color='red', alpha=0.75, label='Test')

    # Also plotting the regression line. in the plotting we plot the xs and not the xsP
    ax.plot(xs, XsHat, color='black', alpha=0.75, label='Prediction')
    ax.set_ylim(-10, 10)

    # Controlling the size of the legend and the location.
    ax.legend(fontsize=20, loc=0)

    plt.show()
    Xtrr, _ = ml.transforms.rescale(ml.transforms.fpoly(Xtr, degree, False), params)
    mseTraining.append(lr.mse(Xtrr, Ytr))
    mseTest.append(lr.mse(XteP, Yte))
```

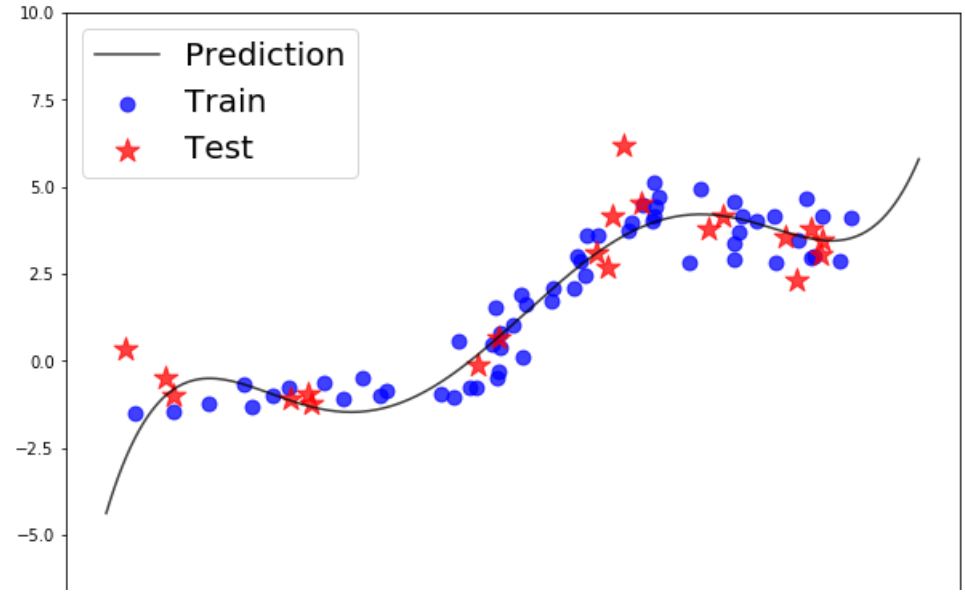
```
Degree: 1
```

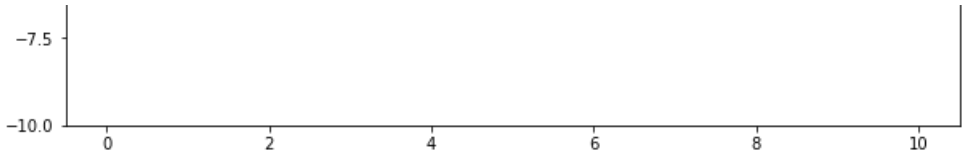



```
Degree: 3
```

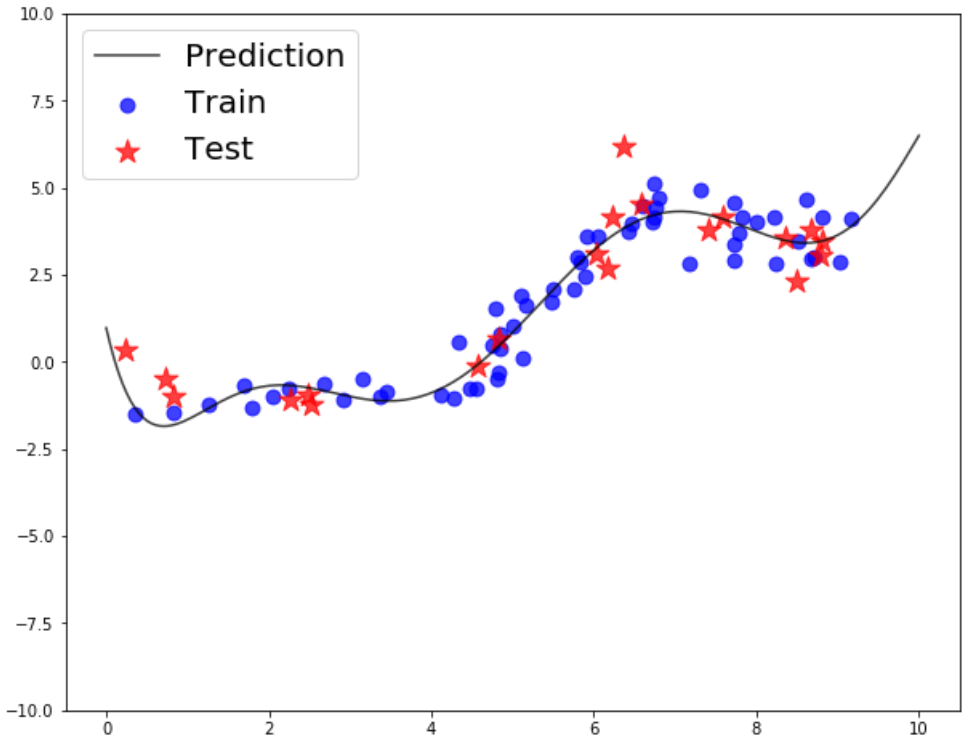



```
Degree: 5
```

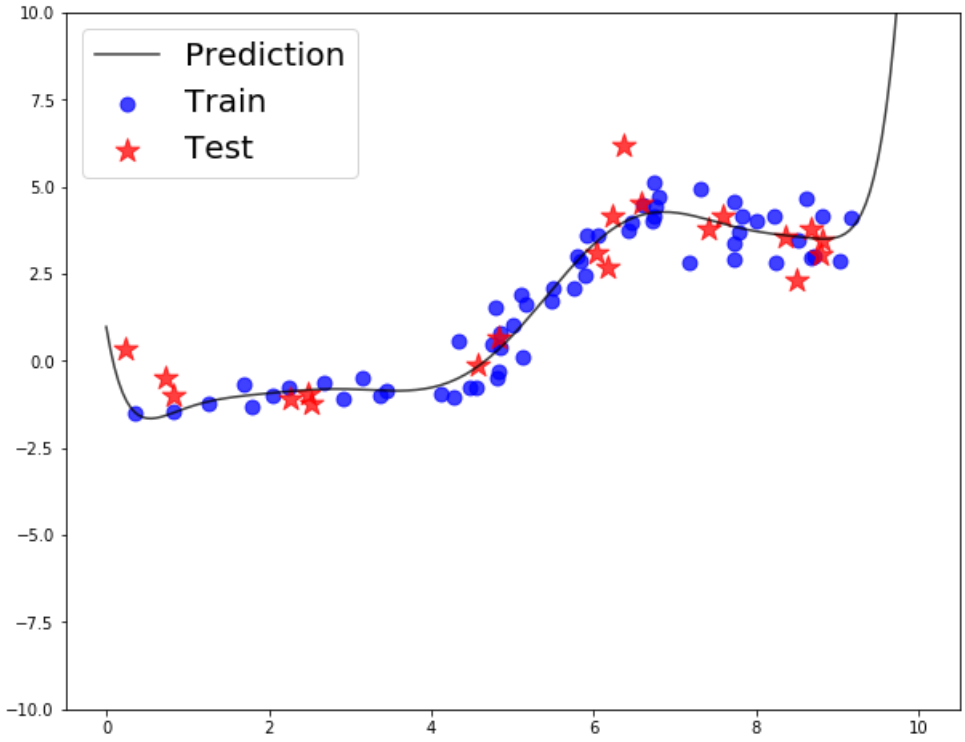




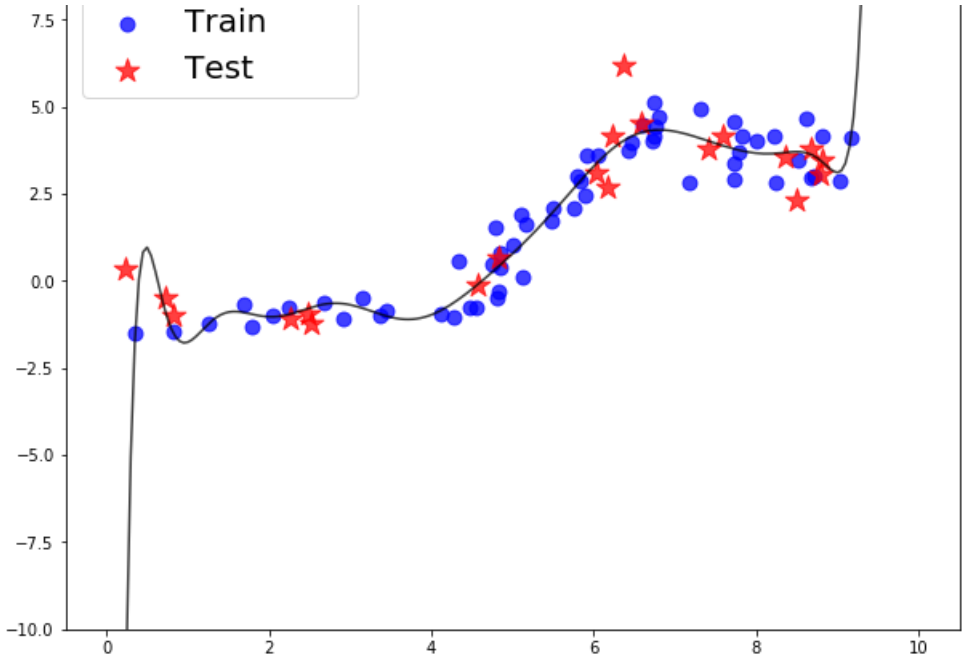

```
Degree: 7
```



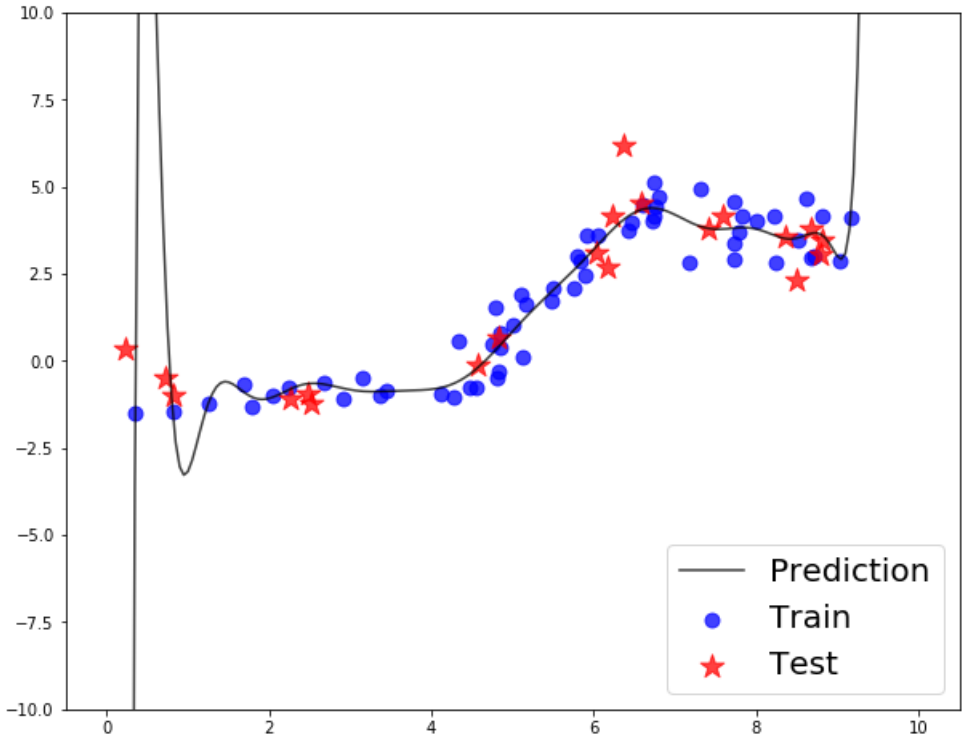

```
Degree: 10
```




```
Degree: 15
```



Degree: 18



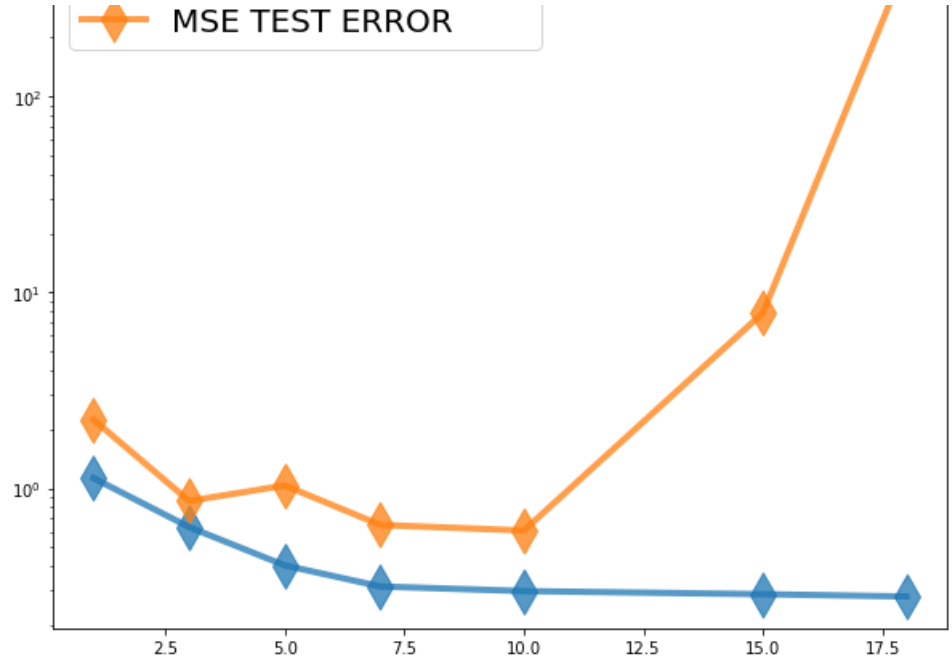
1.3b:

In [126]:

```
f, ax = plt.subplots(1, 1, figsize=(10, 8))
ax.semilogy(deg, mseTraining, lw=4, marker='d', markersize=20, alpha=0.75, label='MSE TRAINING ERRO
R')
ax.semilogy(deg, mseTest, lw=4, marker='d', markersize=20, alpha=0.75, label='MSE TEST ERROR')
ax.legend(fontsize=20, loc=0)
```

Out[126]:

<matplotlib.legend.Legend at 0x118ecf350>



1.3c:

I personally would recommend degree 7 because the prediction line does not seem to make extreme predictions at the end or beginning. For the degrees higher than 7, the beginning or end of the predictions seem to in a sense think the closest training point's direction is going to be continued. The lower degrees' prediction doesn't seem to fit the training data enough.

Problem 2:

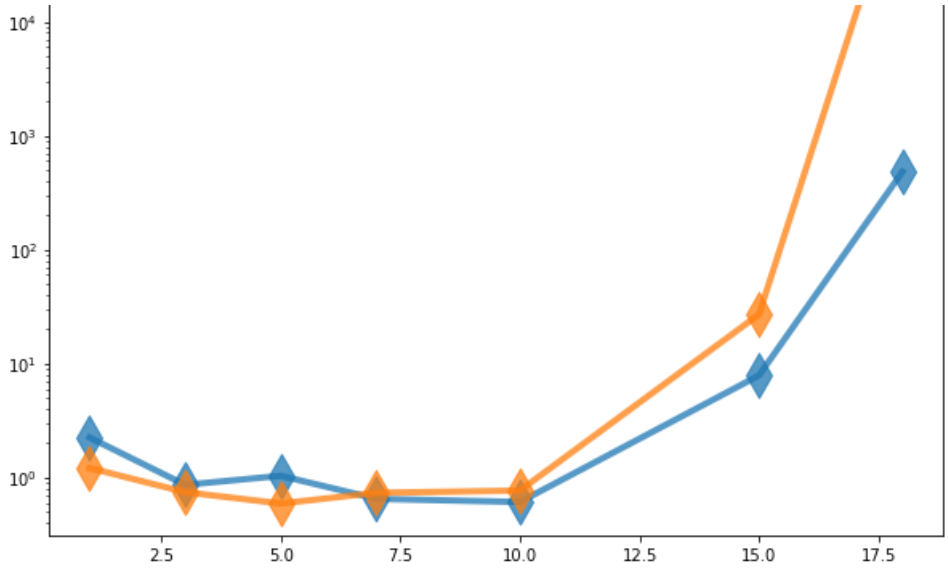
2.1:

In [177]:

```
nFolds = 5
means = []
J = []
for degree in deg:
    for iFold in range(nFolds):
        Xti,Xvi,Yti,Yvi = ml.crossValidate(Xtr,Ytr,nFolds,iFold) # use ith block as validation
        XtiP = ml.transforms.fpoly(Xti, degree, bias=False)
        XtiP,params = ml.transforms.rescale(XtiP)
        learner = ml.linear.linearRegress(XtiP, Yti) # TODO: train on Xti, Yti, the data for this f
old
        Xvii, _ = ml.transforms.rescale(ml.transforms.fpoly(Xvi, degree, False), params)
        J.append(learner.mse(Xvii, Yvi)) # TODO: now compute the MSE on Xvi, Yvi and save it

    # the overall estimated validation error is the average of the error on each fold
    mean = np.mean(J)
    J = []
    means.append(mean)
f, ax = plt.subplots(1, 1, figsize=(10, 8))

ax.semilogy(deg, mseTest, lw=4, marker='d', markersize=20, alpha=0.75, label='MSE TEST ERROR')
ax.semilogy(deg, means, lw=4, marker='d', markersize=20, alpha=0.75, label = 'CROSS-VALIDATION ERRO
R')
ax.legend(fontsize=20, loc=0)
```

Out[177]:

```
<matplotlib.legend.Legend at 0x11c837310>
```



2.2:

It seems like the actual test data is less over-fitting than the cross-validation. The cross-validation error starts off lower than the MSEs computed on the actual test data but then starts to go higher around degree 7.5.

2.3:

I would recommend the five fold cross-validation error at degree 5 because that is when the MSE is the lowest meaning there isn't a lot of errors between the data and the predictions.

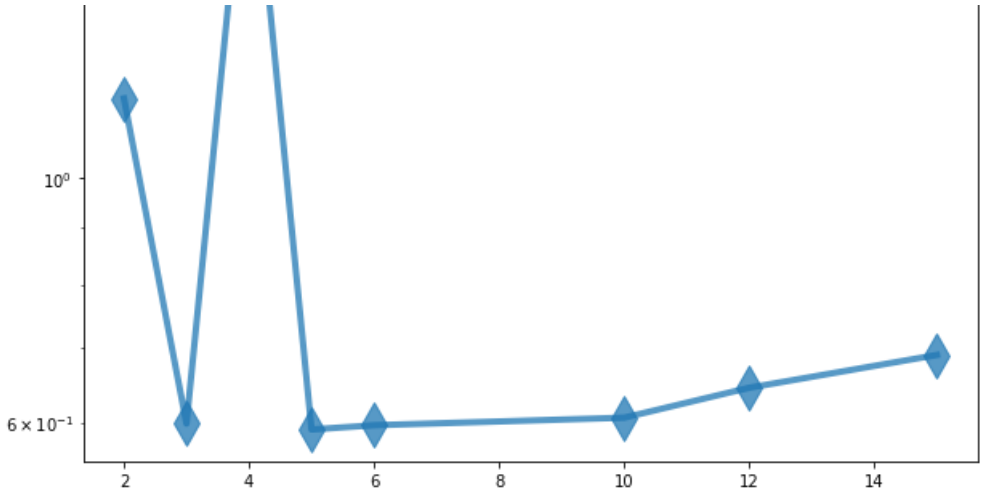
2.4:

In [178]:

```
n = [2, 3, 4, 5, 6, 10, 12, 15]
degree = 5
means = []
J = []
for i in n:
    nFolds = i
    for iFold in range(nFolds):
        Xti,Xvi,Yti,Yvi = ml.crossValidate(Xtr,Ytr,nFolds,iFold) # use ith block as validation
        XtiP = ml.transforms.fpoly(Xti, degree, bias=False)
        XtiP,params = ml.transforms.rescale(XtiP)
        learner = ml.linear.linearRegress(XtiP, Yti) # TODO: train on Xti, Yti, the data for this f
old
        Xvii, _ = ml.transforms.rescale(ml.transforms.fpoly(Xvi, degree, False), params)
        J.append(learner.mse(Xvii, Yvi)) # TODO: now compute the MSE on Xvi, Yvi and save it

# the overall estimated validation error is the average of the error on each fold
    mean = np.mean(J)
    J = []
    means.append(mean)
f, ax = plt.subplots(1, 1, figsize=(10, 8))
ax.semilogy(n, means, lw=4, marker='d', markersize=20, alpha=0.75, label='CROSS-VALIDATION ERROR')
```

Out[178]:

```
[<matplotlib.lines.Line2D at 0x11b5ad8d0>]
```



The pattern seems to be that the for folds starting off there seems to be dramatic changes and at some points, the error is extremely high. The amounts for the errors does seem to stabilize and raise slowly as more folds are done. One thing that stands out to me is the turning point for the graph is at x = 5 which is the degree I had chosen for my previous answer.

Problem 3:

I checked Piazza regularly for updates about homework assignment to see what kind of feedback the professor was providing. I did message a friend, Niralee Patel, to find out where she was in the homework but I did not get a response. I did not collaborate with anyone else for this homework assignment.