Problem 1: Linear Regression

```
In [7]: import numpy as np
        import matplotlib.pyplot as plt
        import mltools as ml
        %matplotlib inline
        np.random.seed(0)
```

Part 1

```
In [8]: data = np.genfromtxt("data/curve80.txt",delimiter=None)

        X = data[:,0]
        X = np.atleast_2d(X).T
        Y = data[:,1]
        Xtr,Xte,Ytr,Yte = ml.splitData(X,Y,0.75)

        Xtr.shape

Out[8]: (60, 1)
```

```
In [9]: Xte.shape

Out[9]: (20, 1)
```

```
In [10]: Ytr.shape

Out[10]: (60,)
```

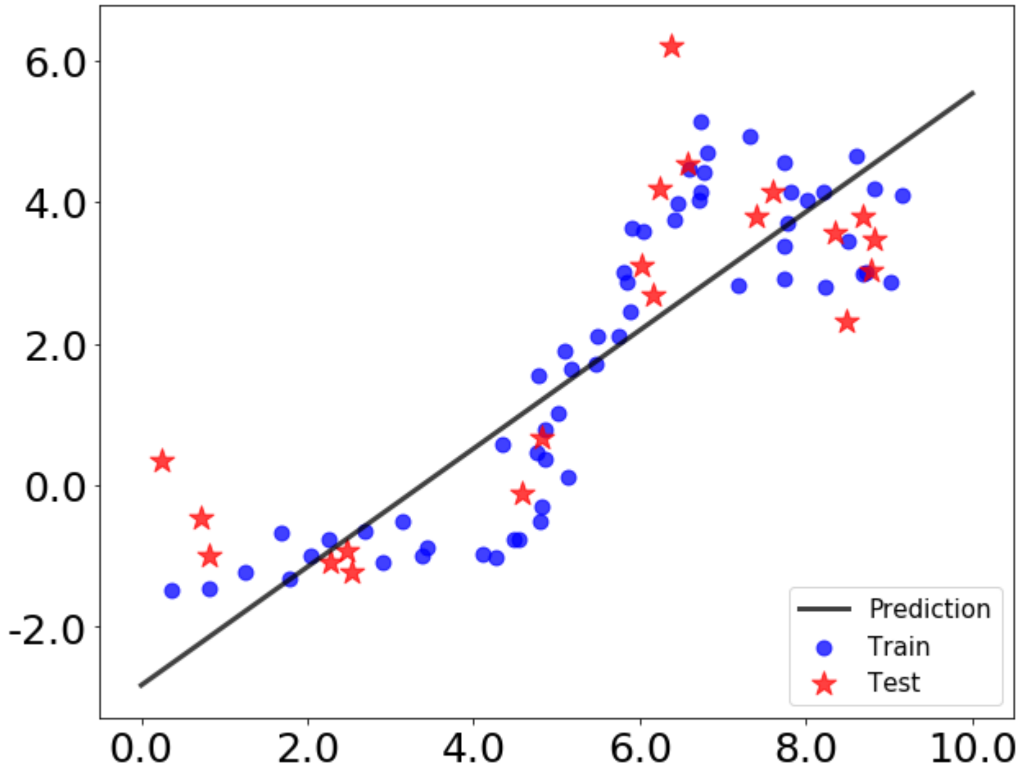
```
In [11]: Yte.shape

Out[11]: (20,)
```

Part 2

```
In [12]: lr = ml.linear.linearRegress( Xtr, Ytr )
         xs = np.linspace(0,10,200)
         xs = np.atleast_2d(xs).T
         ys = lr.predict( xs )
```

a)

```
In [13]: 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=240, marker='*', color='red', alpha=0.75, label='Test')

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

         ax.set_xticklabels(ax.get_xticks(), fontsize=25)
         ax.set_yticklabels(ax.get_yticks(), fontsize=25)

         ax.legend(fontsize=15, loc=4)

         plt.show()
```



b)

```
In [14]: lr.theta

Out[14]: array([[-2.82765049,  0.83606916]])
```

According to the results, x=0 at about -2.83, which is what the plot depicts. Also, a slope of about .84 would mean that the growth of y should be about 84% that of Y, which is also shown by the graph.

c)

```
In [145]: YteHat = lr.predict(Xte)

          def MSE(y_true, y_hat):
              e = y_true - y_hat.T
              J = np.mean(e**2)
              return J

          MSE(Yte, YteHat)
          #lr.mse(Xte, Yte)  this gives the same result

Out[145]: 2.2423492030101246
```

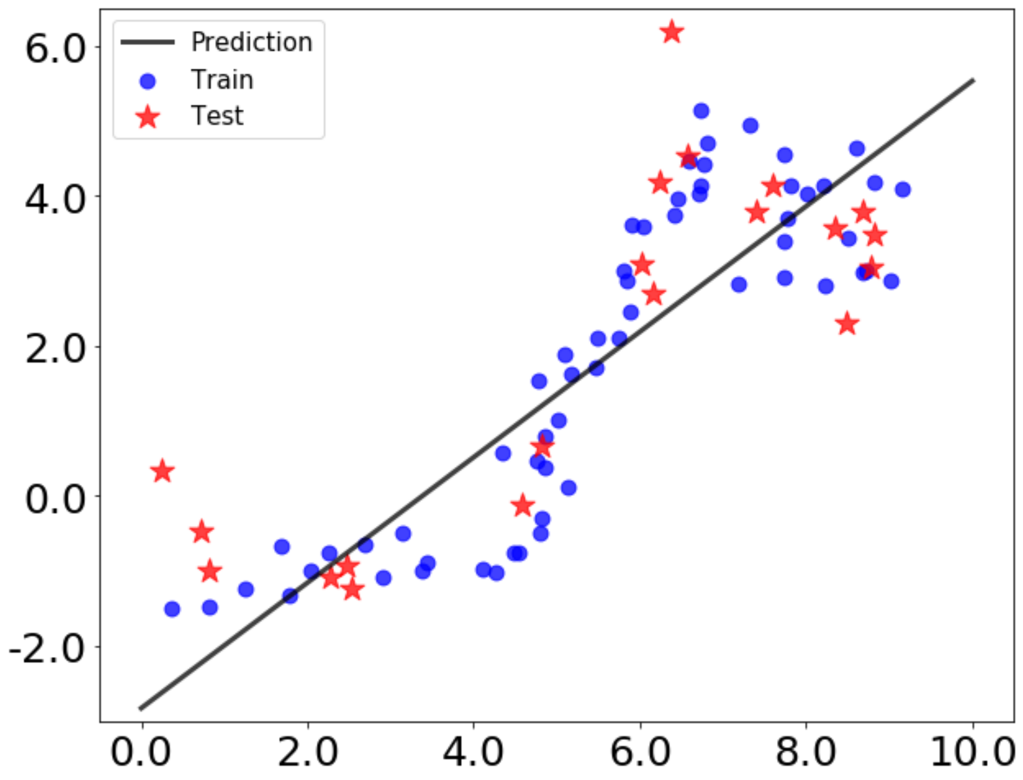
Part 3

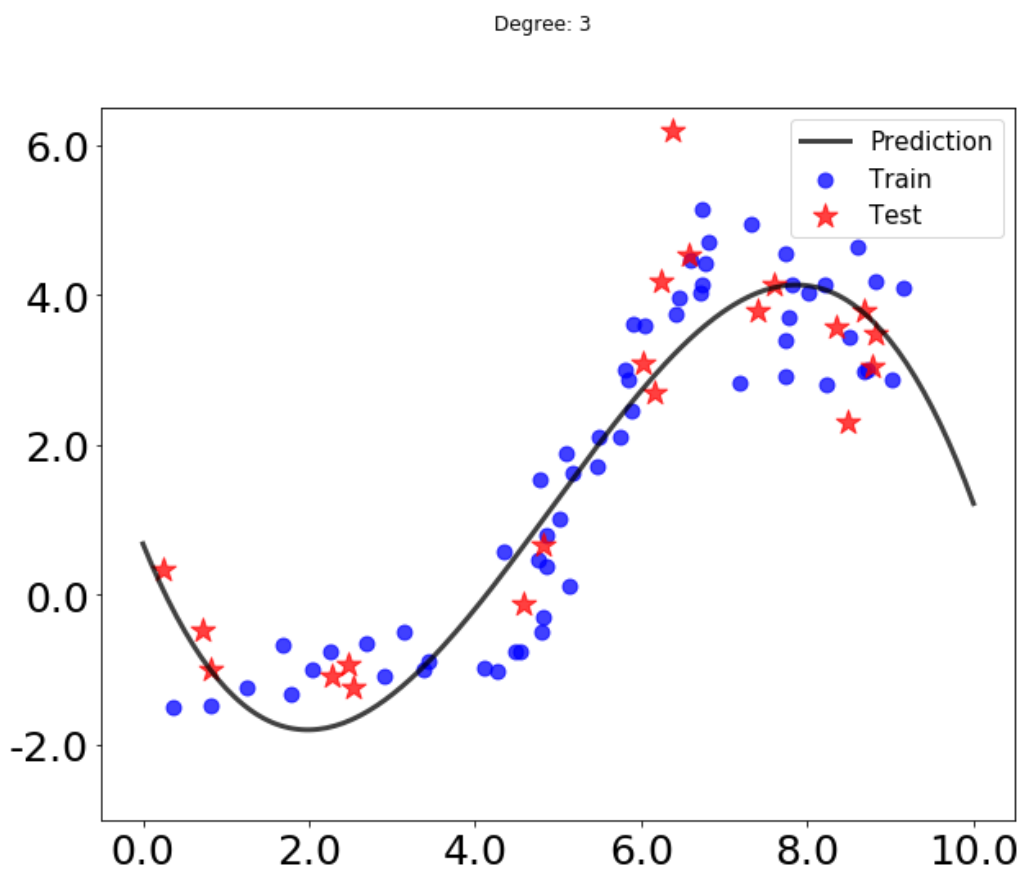
a)

```
In [151]: def phi(x, degree, params):
              return ml.transforms.rescale(ml.transforms.fpoly(x,degree,False), params)[0
          ]

          def new_degree(degree):
              XtrP = ml.transforms.fpoly(Xtr, degree, False)
              XtrP,params = ml.transforms.rescale(XtrP)
              lr = ml.linear.linearRegress( XtrP, Ytr )

              XteP,_ = ml.transforms.rescale(ml.transforms.fpoly(Xte,degree,False), param
          s)

              xs = np.linspace(0, 10, 200)
              xs = np.atleast_2d(xs).T

              ys = lr.predict(phi(xs,degree, params))

              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=240, marker='*', color='red', alpha=0.75, label='Tes
          t')

              # Also plotting the regression line. in the plotting we plot the xs and not
          the xsP
              ax.plot(xs, ys, lw=3, color='black', alpha=0.75, label='Prediction')

              ax.set_xlim(-0.5, 10.5)
              ax.set_ylim(-3, 6.5)
              ax.set_xticklabels(ax.get_xticks(), fontsize=25)
              ax.set_yticklabels(ax.get_yticks(), fontsize=25)

              # Controlling the size of the legend and the location.
              ax.legend(fontsize=15, loc=0)
              plt.suptitle('Degree: '+ str(degree))

              plt.show()
```

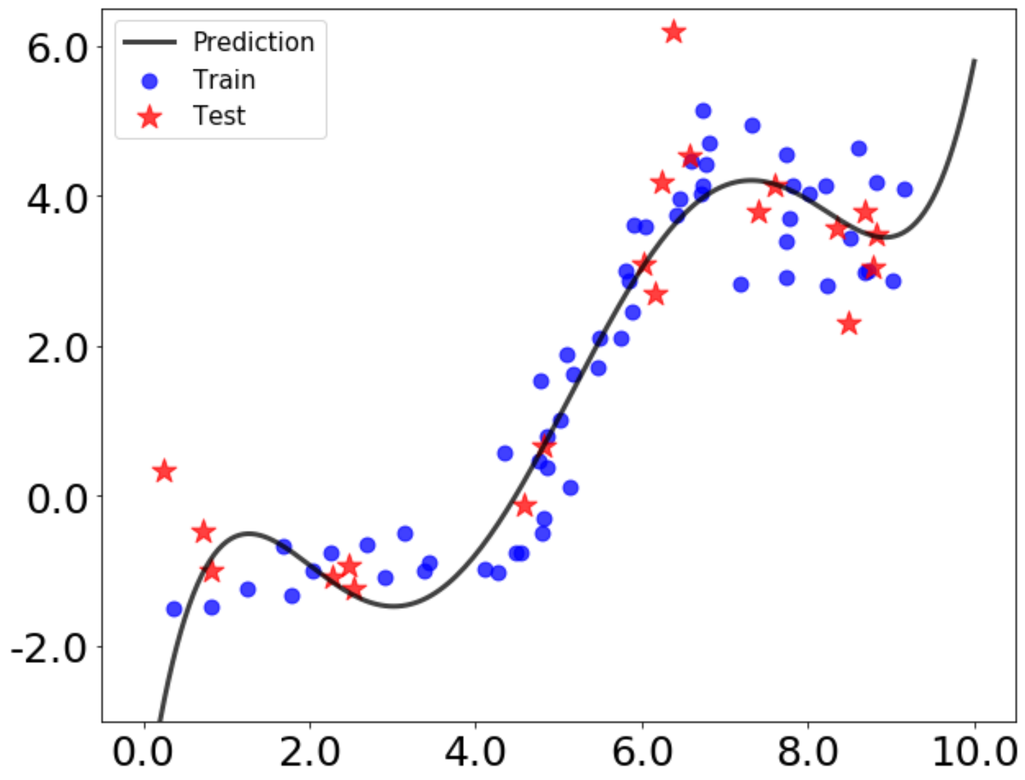
In [152]:
```
degrees = np.array([1,3,5,7,10,18])
for i, d in enumerate(degrees):
    new_degree(d)
```

Degree: 1

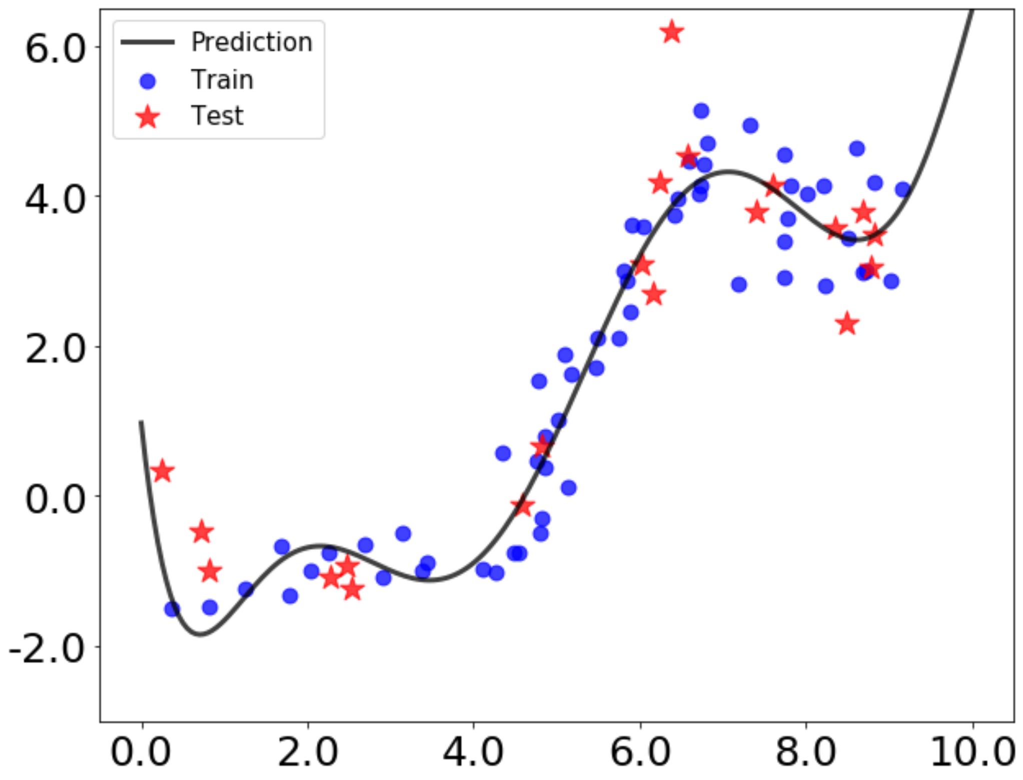



Degree: 3
Prediction
Train
Test
6.0
4.0
2.0
0.0
-2.0
0.0
2.0
4.0
6.0
8.0
10.0

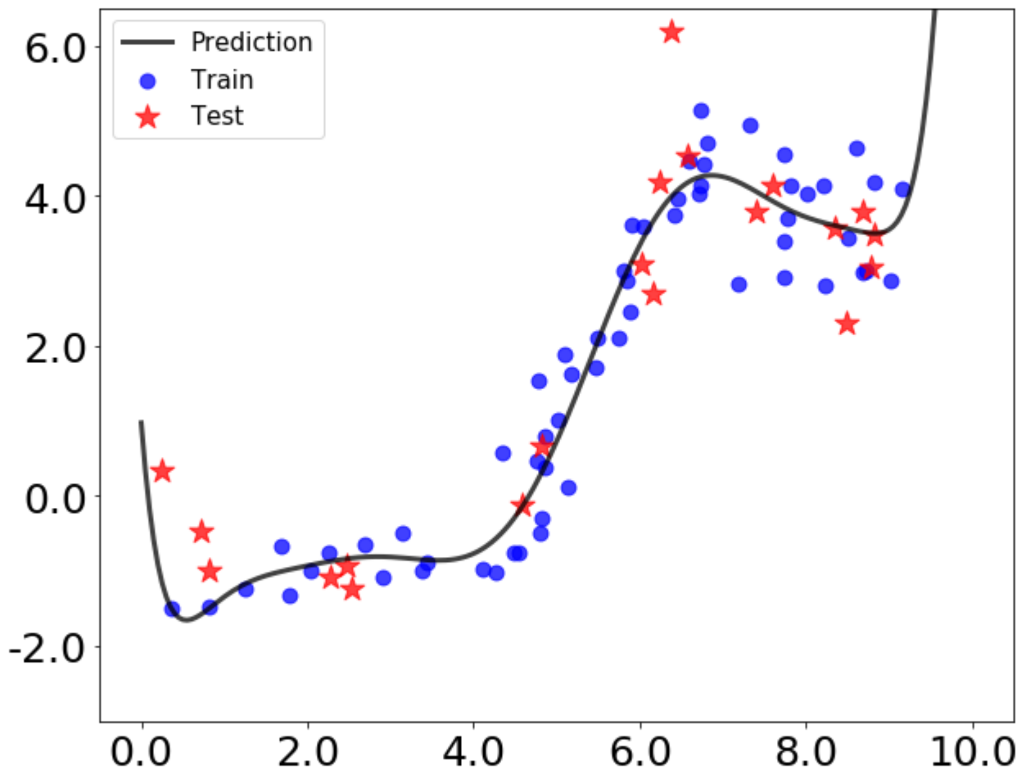
Degree: 5

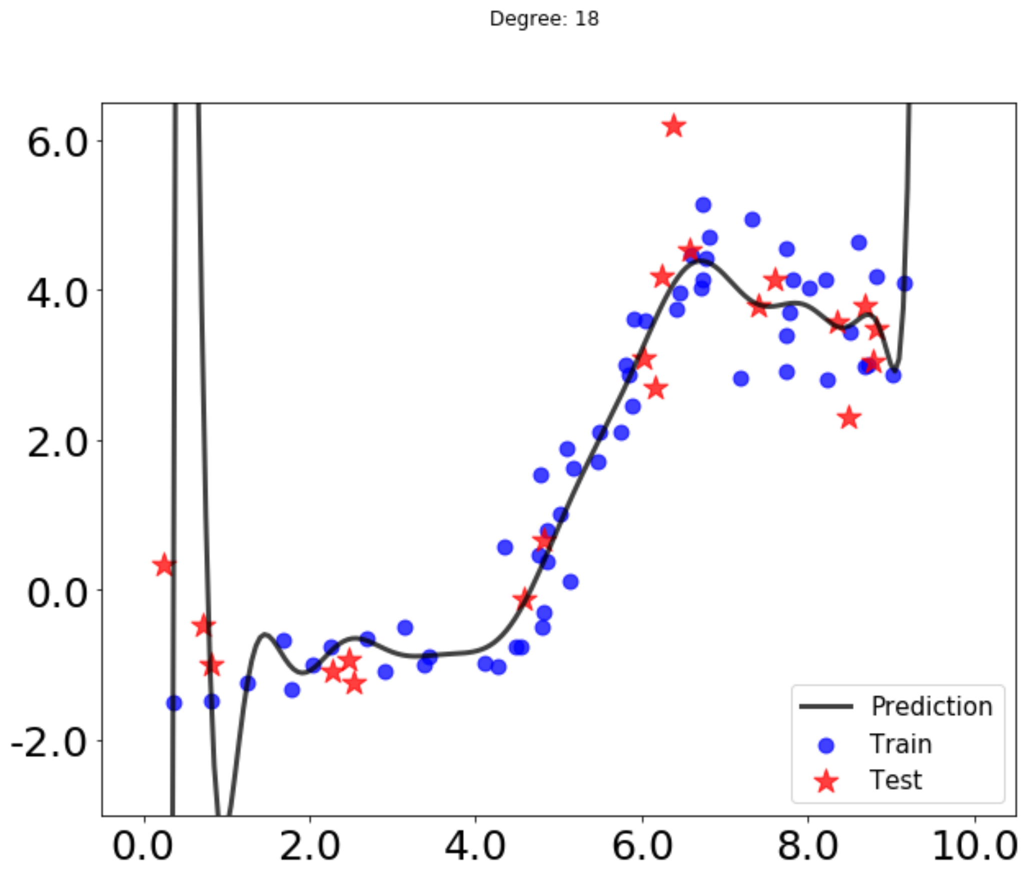


Degree: 7



Degree: 10

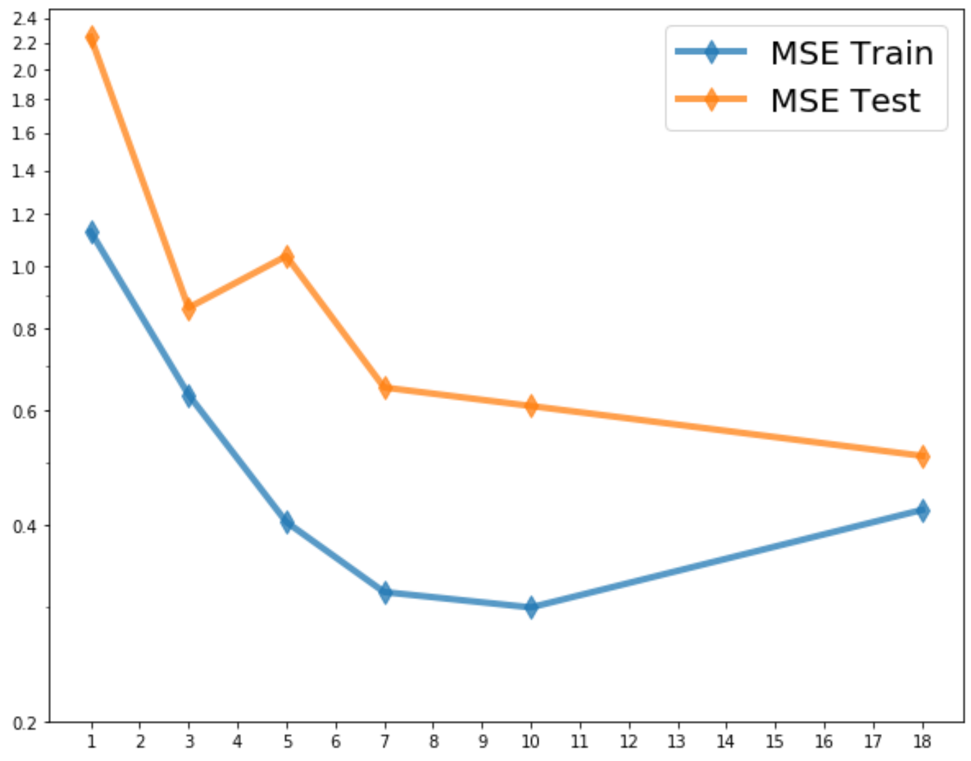




b)

In [153]:
```
mse_train = np.zeros(degrees.shape[0])
mse_test = np.zeros(degrees.shape[0])

for i, degree in enumerate(degrees):
    XtrP = ml.transforms.fpoly(Xtr, degree, False)

    lr = ml.linear.linearRegress(XtrP, Ytr)

    mse_train[i] = lr.mse(XtrP, Ytr)

    XteP = ml.transforms.fpoly(Xte, degree, False)

    mse_test[i] = lr.mse(XteP, Yte)
```

In [174]:
```
f, ax = plt.subplots(1, 1, figsize=(10, 8))

ax.semilogy(degrees, mse_train, lw=4, marker='d', markersize=10, alpha=0.75, la
bel='MSE Train')
ax.semilogy(degrees, mse_test, lw=4, marker='d', markersize=10, alpha=0.75, lab
el='MSE Test')

ax.set_xticks(np.arange(1, 19, 1))
ax.set_yticks(np.arange(.2, 2.6, .2))

ax.set_xticklabels(ax.get_xticks(), fontsize=10)
ax.set_yticklabels(ax.get_yticks(), fontsize=10)

ax.legend(fontsize=20, loc=0)

plt.show()
```



c)

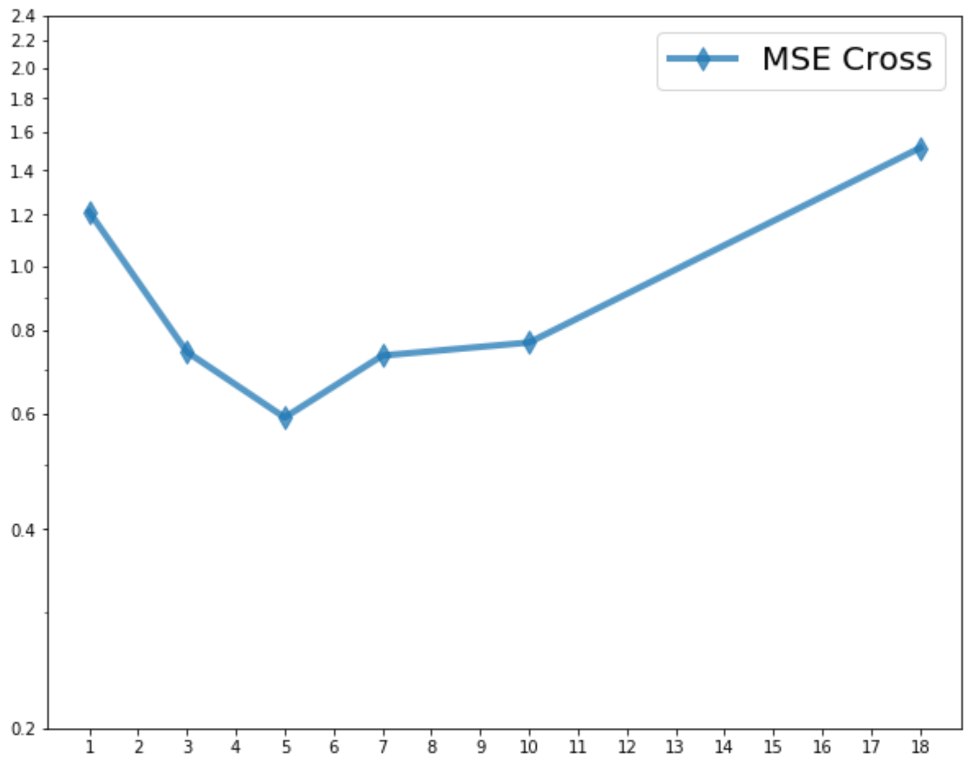
I would recommend a polynomial degree of 10 for this data. While it is true that technically 18 does better than 10 on the test data, the difference is only by approximately .05 for a significantly more complex function that also does worse on training data. In fact, since the degree being 7, the benefit of going to a higher degree isn't very significant. One could argue that 7 would be the best since it would be picking the simplest classifier from the time the benefit starts to even out. I think with more data it would be easier to tell whether degree 7, or 10 would end up being better, however, for what we do have here, degree 10 seems to minimize both Training, and Test error the best, and that is why I pick degree 10.

Problem 2: Cross-validation

Part 1

```
In [179]: mse_cross = np.zeros(degrees.shape[0])

          def x_val(folds, degree):
              for i, degree in enumerate(degrees):
                  J = []
                  for f in range(folds):
                      Xti,Xvi,Yti,Yvi = ml.crossValidate(Xtr,Ytr,folds,f)

                      XtiP = ml.transforms.fpoly(Xti, degree, False)

                      lr = ml.linear.linearRegress(XtiP, Yti)

                      XviP = ml.transforms.fpoly(Xvi, degree, False)

                      J.append(lr.mse(XviP, Yvi))
                  mse_cross[i] = np.mean(J)

          for i, d in enumerate(degrees):
              x_val(5, d)
```

```
In [180]: f, ax = plt.subplots(1, 1, figsize=(10, 8))

          ax.semilogy(degrees, mse_cross, lw=4, marker='d', markersize=10, alpha=0.75, la
          bel='MSE Cross')

          ax.set_xticks(np.arange(1, 19, 1))
          ax.set_yticks(np.arange(.2, 2.6, .2))

          ax.set_xticklabels(ax.get_xticks(), fontsize=10)
          ax.set_yticklabels(ax.get_yticks(), fontsize=10)

          ax.legend(fontsize=20, loc=0)

          plt.show()
```



Part 2

The MSE estimates from a five-fold cross-validation are actually quite different from the ones computed in Problem 1. In this one, the best degree would appear to be 5, whereas, in Problem 1, degree 5 was the second worst only to a degree of 1! In addition to that, the higher order polynomials do not taper off downwards in MSE like in Problem 1. Now, they actually go up instead. Overall, however, the distribution shown here actually makes a lot more sense. Normally anything approaching higher level degrees will increase error drastically, and in previous examples from lecture, we had also seen that something around degree 5 seemed to be optimal in a lot of cases. Because of this, it is not surprising that a plot that has relatively good grouping like this performs better with lower degree polynomials, rather than higher. The likely reason that Problem 1 did well with higher order polynomials was probably just luck (or by design by the Professor). The test points chosen from the bottom half of the data just so happened to fit the curves that were generated quite well, so it made it seem as if a higher order polynomial would be the best fit. However, upon using other methods to test the data, it was found that the higher order ones weren't actually all that good after all.

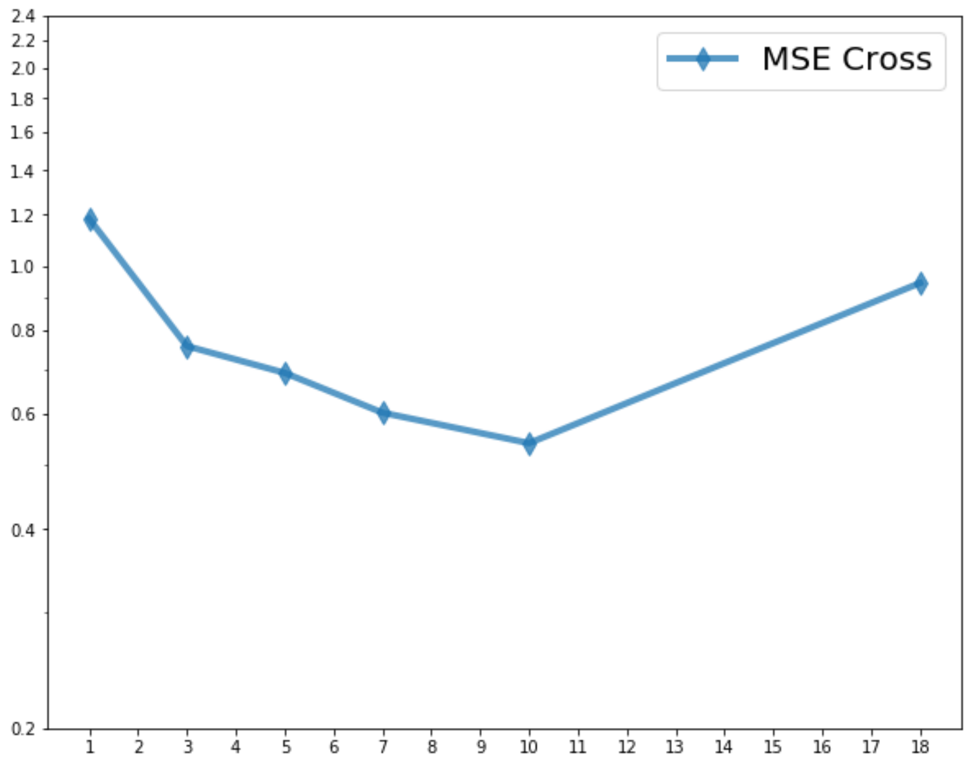
Part 3

Based on the graph above, the results are pretty clear. As I mentioned above, the best degree based on the cross-validation method seems to be degree 5, because it minimizes the MSE of all the tested degrees, and by a somewhat significant margin (by approximately .15 on both sides of it, and even more for any other point).

Part 4

In [181]:
```
folds = np.array([2,3,4,5,6,10,12,15])

for i, f in enumerate(folds):
    x_val(f, 5)
```

In [185]:
```
f, ax = plt.subplots(1, 1, figsize=(10, 8))

ax.semilogy(degrees, mse_cross, lw=4, marker='d', markersize=10, alpha=0.75, la
bel='MSE Cross')

ax.set_xticks(np.arange(1, 19, 1))
ax.set_yticks(np.arange(.2, 2.6, .2))

ax.set_xticklabels(ax.get_xticks(), fontsize=10)
ax.set_yticklabels(ax.get_yticks(), fontsize=10)

ax.legend(fontsize=20, loc=0)

plt.show()
```



Based on this plot, it would appear that, up to a point, increasing the number of folds results in a better MSE. It seems that the magic number is around 10 folds for this data set. Any number before or after does increase the MSE a little bit. This result also makes sense, because generally, the more training data that you have, the better you are able to estimate what will come ahead. However, if you go to the other extreme, and make it so that there is only one point left in validation, not only are you overfitting the training data, but the MSE would get thrown off more because one point that is far off would be the only validation data, and could result in a bad score. The goal of making these models is to make them general, so that they will apply to new data found as best as possible. However, if they are so general that they don't recognize the trends in the data, it would perform poorly. On the other hand, if they conform to the test data too much, even if it fits the test data really well, it still might not perform well in practice. It is important to find a balance between these two extremes, and the results of this plot, as well as the one above varying the degrees seem to both indicate this as well.

Statement of Collaboration

I state that I did not collaborate with anybody on this assignment. I did it entirely by myself with no help from another person.