# Problem 1

In [1]:
```
import numpy as np
import mltools as ml
import matplotlib.pyplot as plt
data = np.genfromtxt("data/curve80.txt",delimiter=None)
```

## Problem 1.1

In [2]:
```
X = data[:,0]
X = np.atleast_2d(X).T
Y = data[:,1]
Xtr,Xte,Ytr,Yte = ml.splitData(X,Y,0.75)
```

In [3]:
```
print("shapes of [Xtr, Xte, Ytr, Yte] are \n",Xtr.shape,Xte.shape, Ytr.shape, Yte.shape)
```

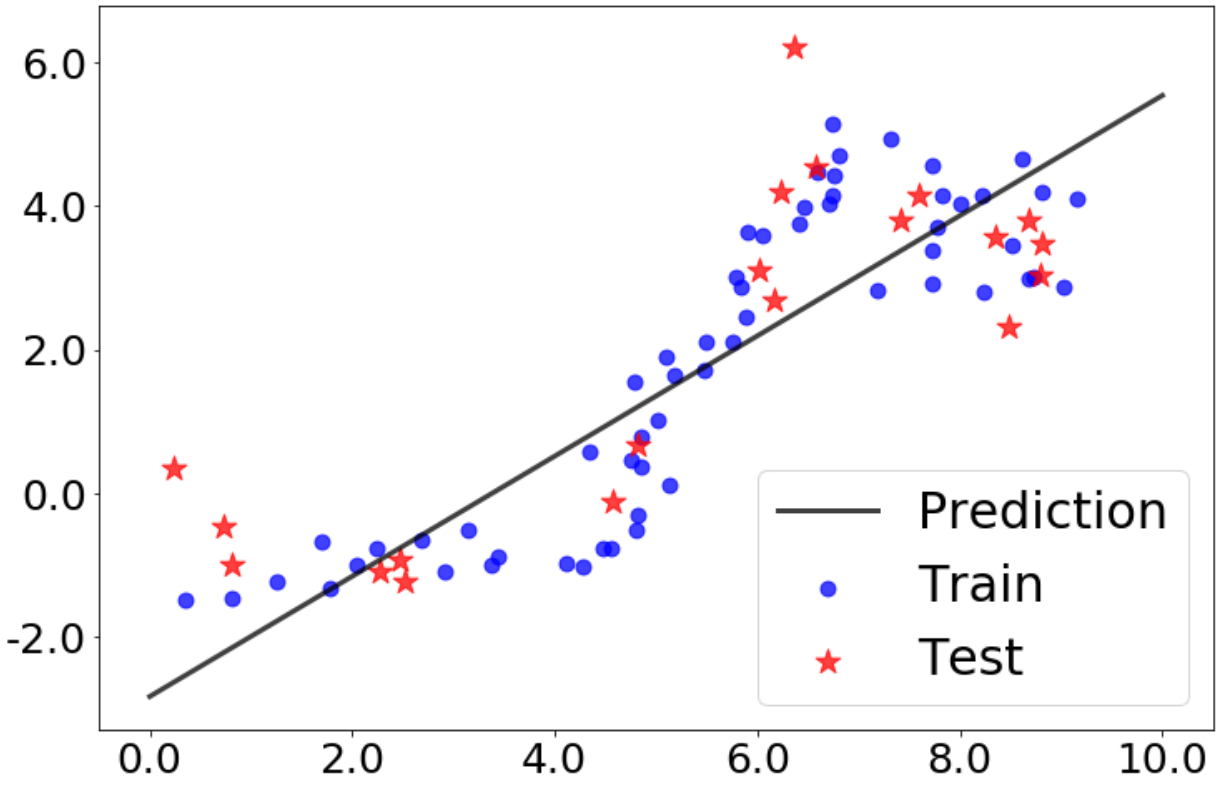
```
shapes of [Xtr, Xte, Ytr, Yte] are
 (60, 1) (20, 1) (60,) (20,)
```

## Problem 1.2

### Problem 1.2 (a)

In [4]:
```
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
```

In [5]:
```
# Plotting the data
f, ax = plt.subplots(1, 1, figsize=(12, 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')

# Also plotting the regression line
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)

# Controlling the size of the legend and the location.
ax.legend(fontsize=30, loc=4)

plt.show()
```



### Problem 1.2 (b)

In [6]:
```
print(" linear regression coefficients: ",lr.theta)
```

```
 linear regression coefficients:  [[-2.82765049  0.83606916]]
```

Y = 0.83606916 * X - 2.82765049 It intercept axis y at -2.8276504 and the cofficient is larger than 0 by just looking at the slope

### Problem 1.2 (c)

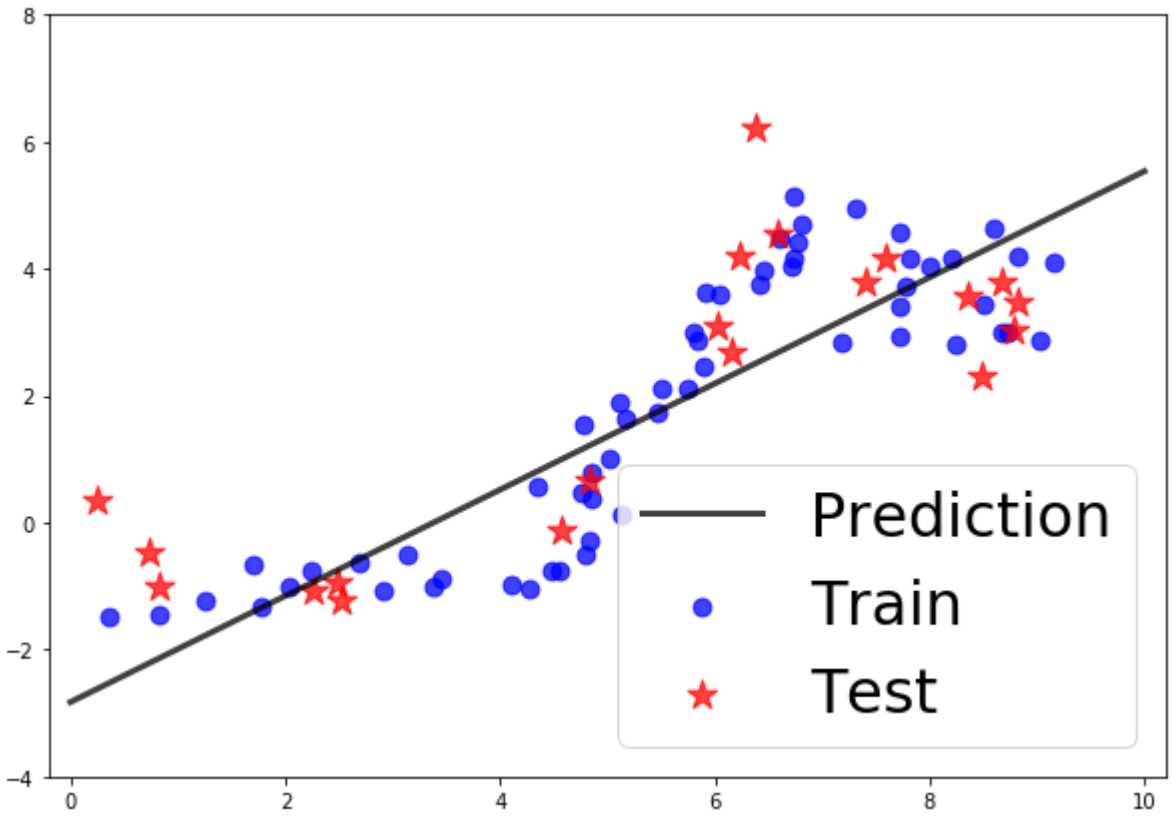
In [7]:
```
Train = lr.mse(Xtr,Ytr)
Test = lr.mse(Xte,Yte)
print("MSE of the predictions on the taining data is ",Train)
print("MSE of the predictions on the test data is ",Test)
```

```
MSE of the predictions on the taining data is  1.127711955609391
MSE of the predictions on the test data is  2.2423492030101246
```

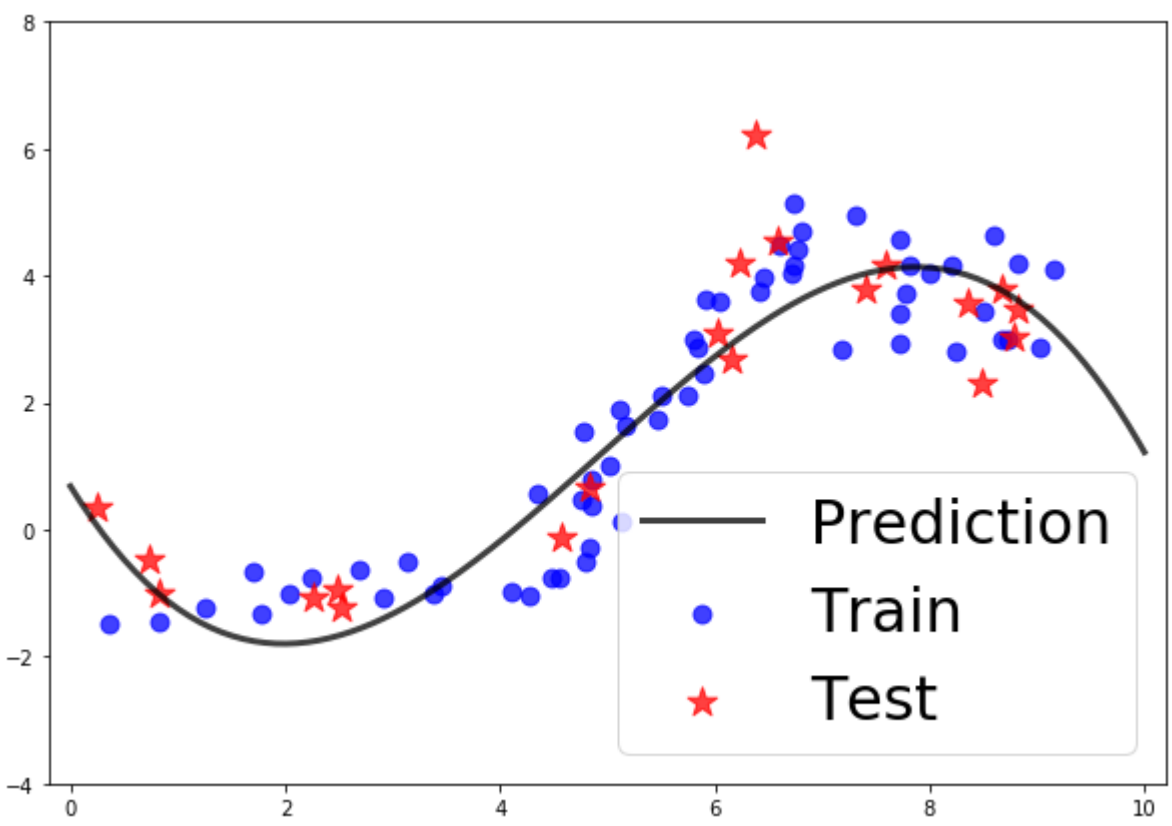
### Problem 1.3 (a)

In [8]:
```
degree = [1,3,5,7,10,18]

for i in degree:

    XtrP = ml.transforms.fpoly(Xtr,i, bias=False)
    XtrP,params = ml.transforms.rescale(XtrP)
    lr = ml.linear.linearRegress( XtrP, Ytr )# create and train model

    XS,p = ml.transforms.rescale( ml.transforms.fpoly(xs,i,False), params)
    YS = lr.predict(XS)

    print("Degree = ",i)
    f, ax = plt.subplots(1, 1, figsize=(10, 7))

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

    ax.set_xlim(-0.2, 10.2)
    ax.set_ylim(-4, 8)
    # Controlling the size of the legend and the location.
    ax.legend(fontsize=30, loc=4)

    plt.show()
```

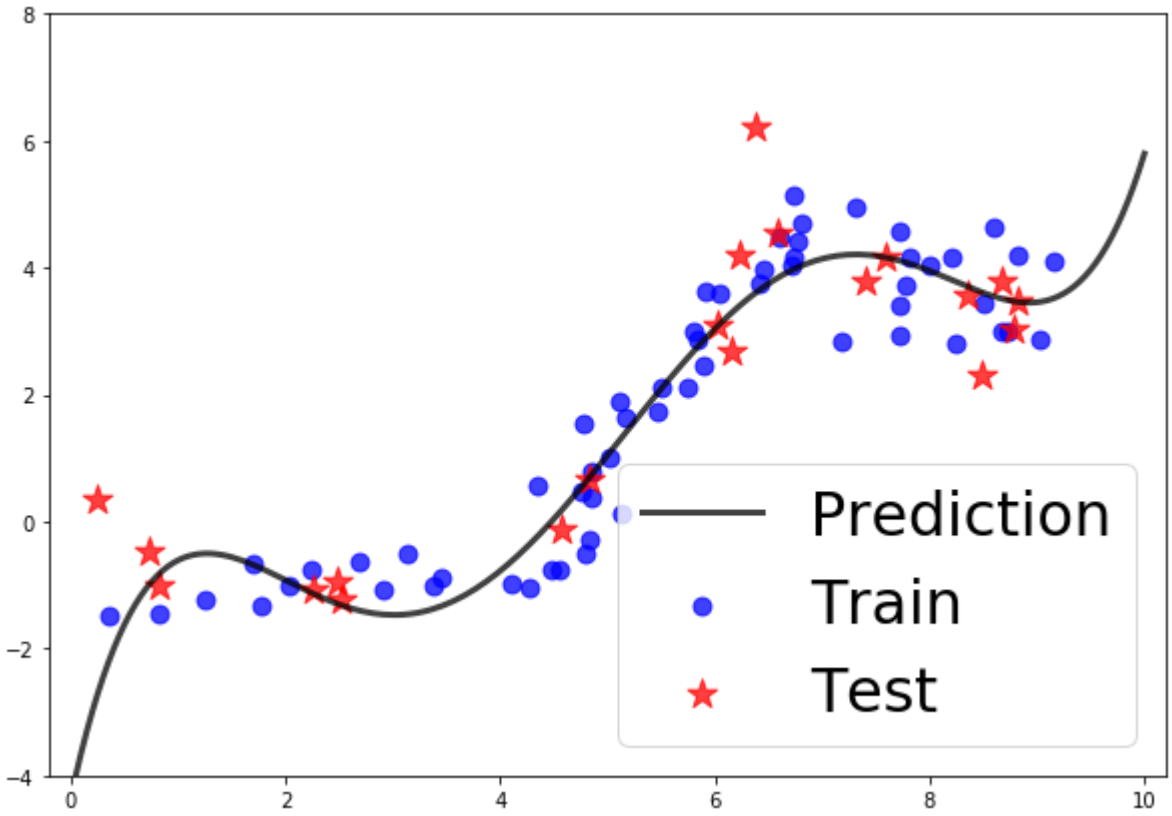
Degree = 1



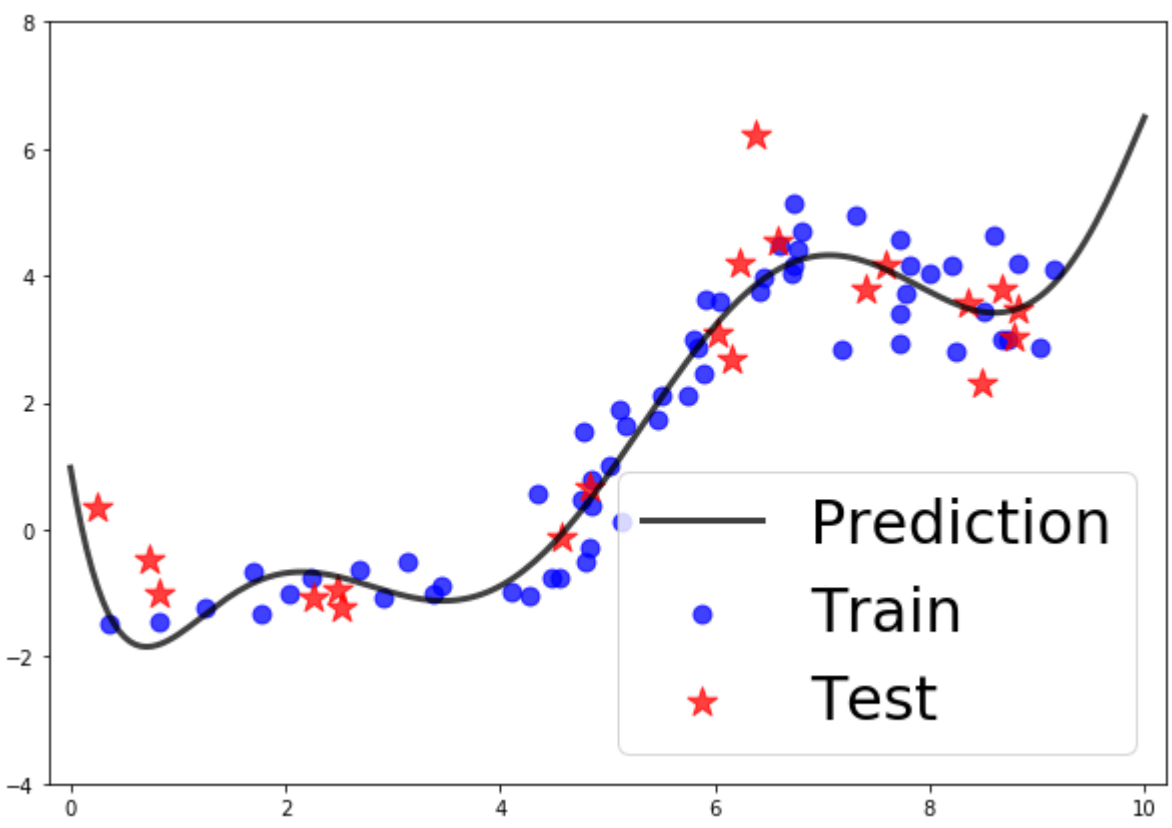

Degree = 3



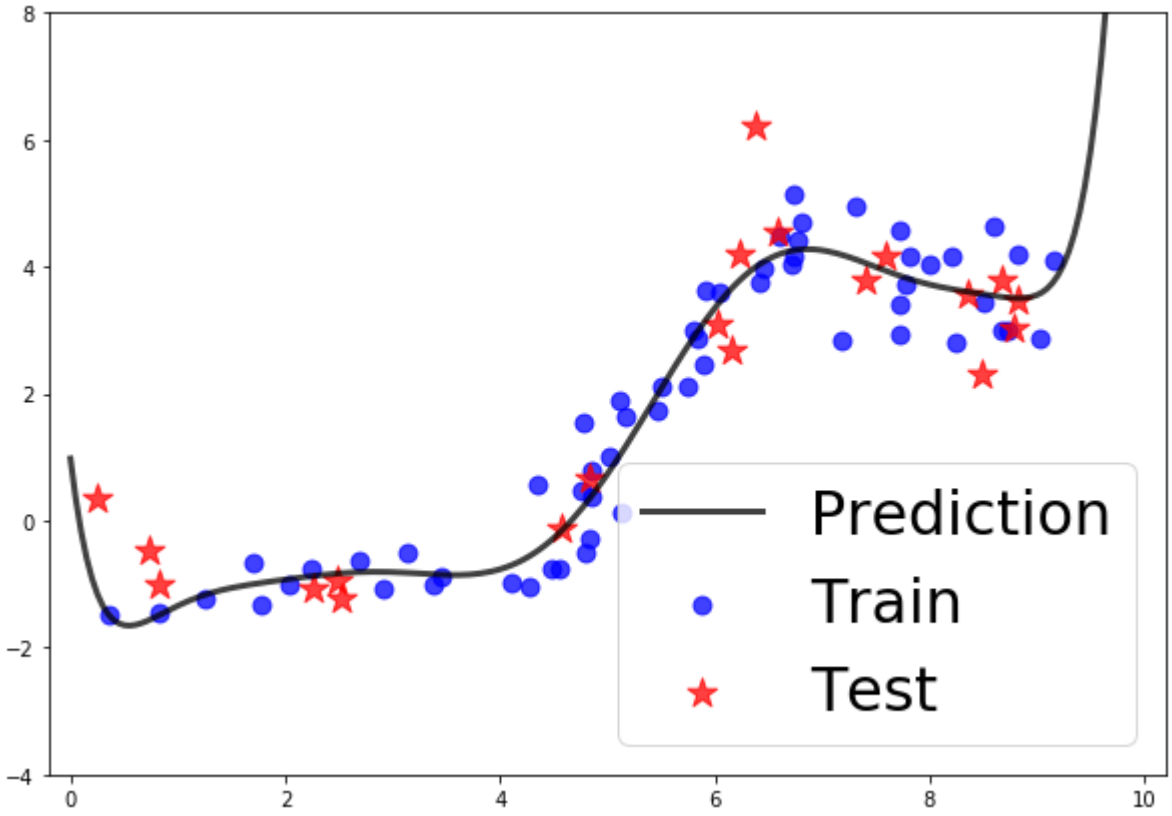

Degree = 5



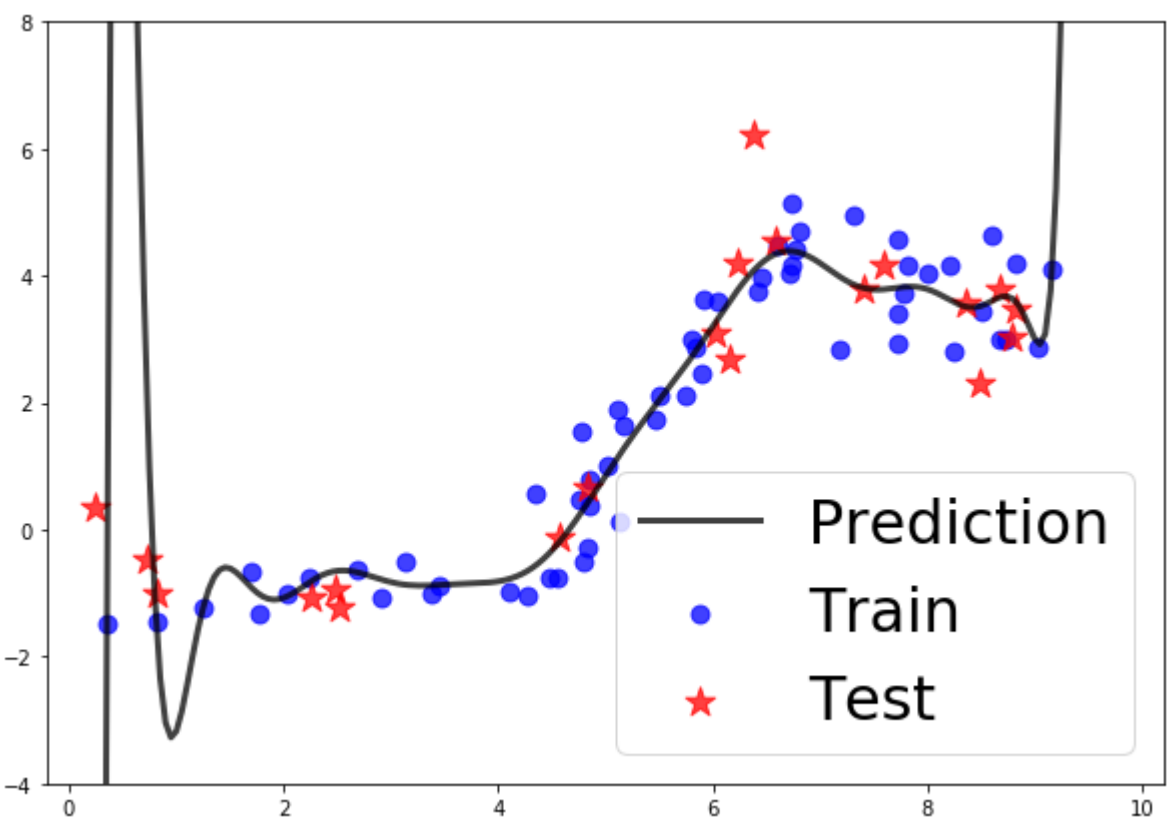

Degree = 7




Degree = 10




Degree = 18

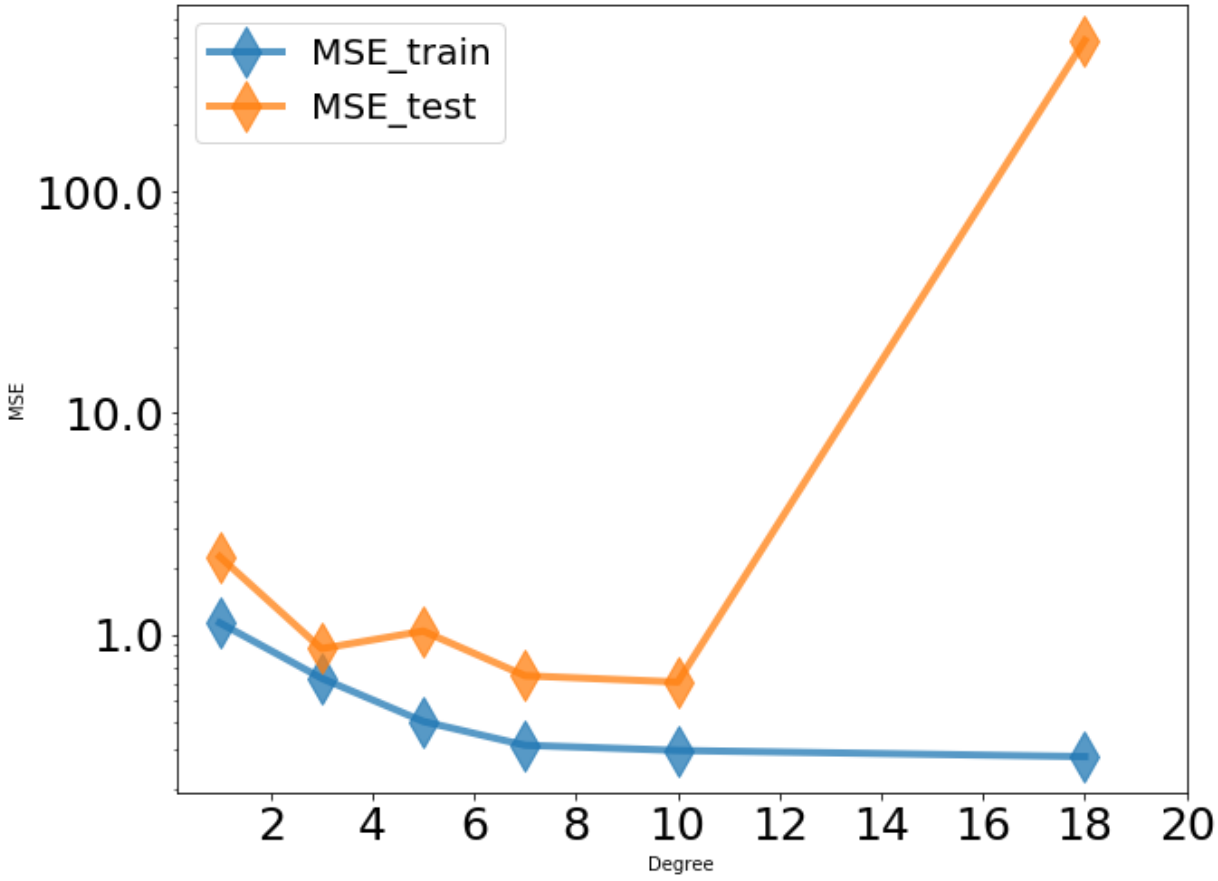



**Problem 1.3 (b)**

In [9]:
```
degrees = np.array([1,3,5,7,10,18])
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, bias=False)
    XtrP,params = ml.transforms.rescale(XtrP)
    lr = ml.linear.linearRegress( XtrP, Ytr )

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

    mse_train[i] = lr.mse(XtrP, Ytr)
    mse_test[i] = lr.mse(XteP, Yte)
    print("When degree = ",degree,", mse_train = ",mse_train[i]," and mse_test = ",mse_tes
```

```
When degree =  1 , mse_train =  1.1277119556093909  and mse_test =  2.242349203010125
When degree =  3 , mse_train =  0.6339652063119635  and mse_test =  0.8616114815449999
When degree =  5 , mse_train =  0.4042489464459056  and mse_test =  1.0344190205632156
When degree =  7 , mse_train =  0.3156346739892996  and mse_test =  0.6502246079670317
When degree =  10 , mse_train =  0.2989479796813433  and mse_test =  0.6090600748904027
When degree =  18 , mse_train =  0.2804804223080565  and mse_test =  481.20396934201574
```

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

# Plotting a line with markers where there's an actual x value.
ax.semilogy(degrees, mse_train, lw=4, marker='d', markersize=20, alpha=0.75, label='MSE_tr
ax.semilogy(degrees, mse_test, lw=4, marker='d', markersize=20, alpha=0.75, label='MSE_tes

#ax.set_xlim(1.2, 20.5)
#ax.set_ylim(30, 1100)

# Setting the X-ticks manually.
ax.set_xticks(np.arange(2, 21, 2))

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

ax.set_xlabel("Degree")
ax.set_ylabel("MSE")

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

plt.show()
```




**Problem 1.3 (c)**

I recommend polynomial degree to be 10 because it has least MSE on test data

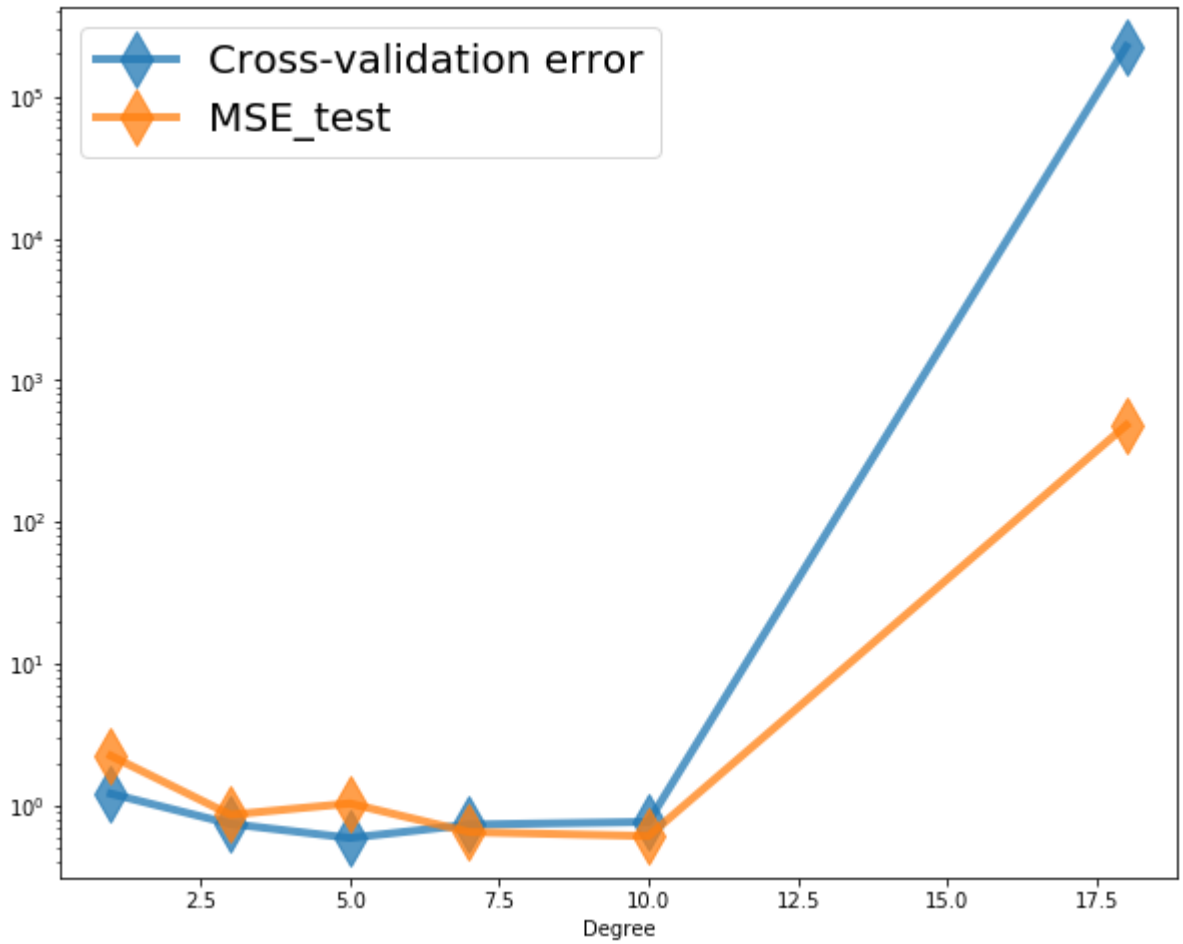
# Problem 2

## Problem 2.1

In [11]:
```
def compute_mse(degree):
    XtiP = ml.transforms.fpoly(Xti,degree, bias=False)
    XtiP,params = ml.transforms.rescale(XtiP)
    lr = ml.linear.linearRegress( XtiP, Yti )

    XviP,_ = ml.transforms.rescale(ml.transforms.fpoly(Xvi,degree,False), params)
    r = lr.mse(XviP,Yvi)
    return r
```

In [12]:
```
nFolds = 5
degrees = np.array([1,3,5,7,10,18])
each_mse = np.zeros(nFolds)
cve = []
for d in degrees:
    for iFold in range(nFolds):
        Xti, Xvi, Yti, Yvi = ml.crossValidate(Xtr, Ytr, nFolds, iFold)
        each_mse[iFold] = compute_mse(d)
    cve.append(np.mean(each_mse))
print(" Cross-validation Error of different degrees [1,3,5,7,10,18] is \n",cve)
f, ax = plt.subplots(1, 1, figsize=(10, 8))
ax.semilogy(degrees, cve, lw=4, marker='d', markersize=20, alpha=0.75, label='Cross-valida
ax.semilogy(degrees, mse_test, lw=4, marker='d', markersize=20, alpha=0.75, label='MSE_tes
ax.set_xlabel("Degree")
ax.legend(fontsize=20, loc=0)

plt.show()
```

```
 Cross-validation Error of different degrees [1,3,5,7,10,18] is
 [1.2118626629641984, 0.7429005752051661, 0.5910703726406558, 0.7335637831345124, 0.767
7056877294718, 225451.03371884333]
```




## Problem 2.2

When degree is less then 10, MSE of them are similar, and at degree = 5, cross validation data is the lowest. When degree is larger than 10, Cross-validation error increases faster than the other.

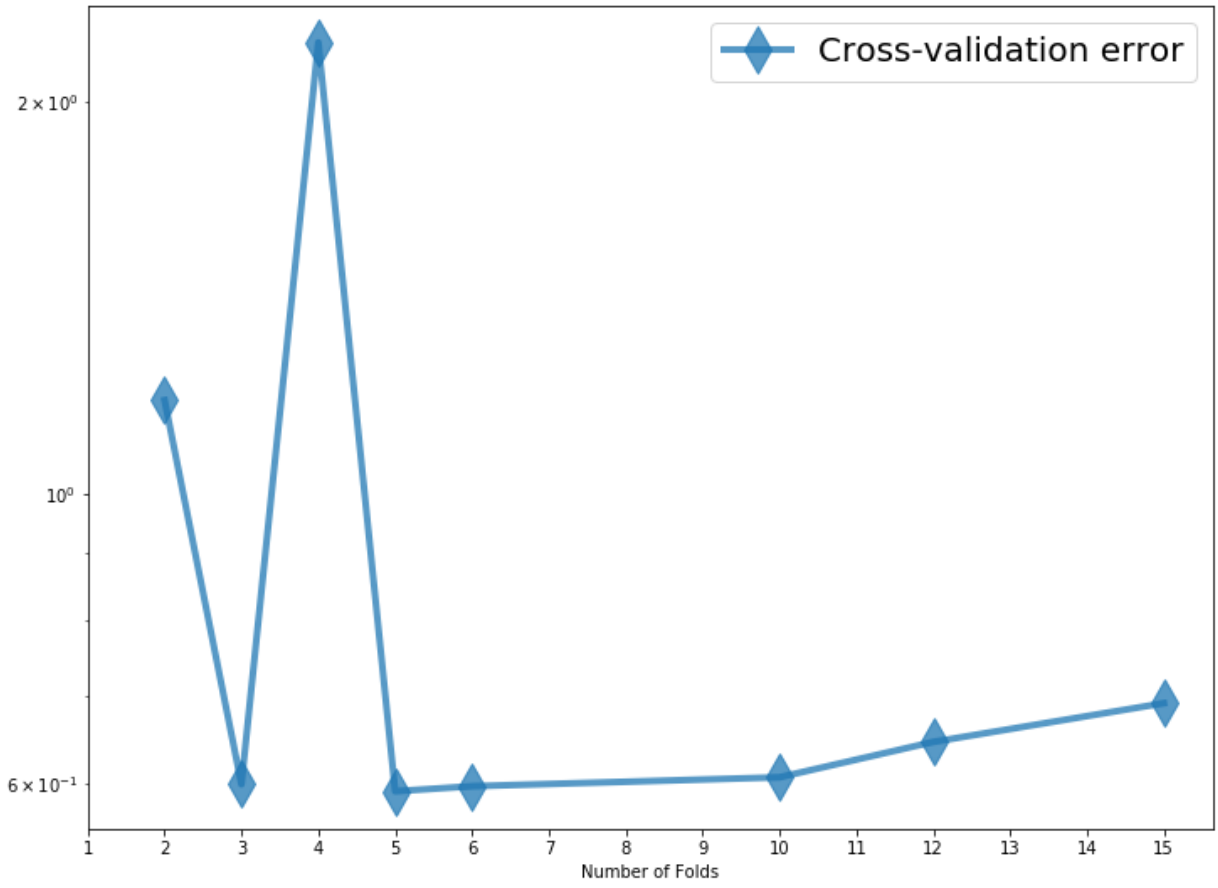
## Problem 2.3

I recommend polynomial degree to be 5 because its MSE is the lowest.

## Problem 2.4

In [13]:
```
Folds_list = np.array([2,3,4,5,6,10,12,15])
degree = 5
cve = []

for i in Folds_list:
    each_mse = np.zeros(i)
    for iFold in range(i):
        Xti, Xvi, Yti, Yvi = ml.crossValidate(Xtr, Ytr, i, iFold)
        each_mse[iFold] = compute_mse(5)
    cve.append(np.mean(each_mse))

print(" Cross-validation Error of different number of folds [2,3,4,5,6,10,12,15] is \n",cve

f, ax = plt.subplots(1, 1, figsize=(12, 9))
ax.semilogy(Folds_list, cve, lw=4, marker='d', markersize=20, alpha=0.75, label='Cross-val
ax.set_xlabel("Number of Folds")
ax.legend(fontsize=20, loc=0)
ax.set_xticks(np.arange(1, 16, 1))

plt.show()
```

```
 Cross-validation Error of different number of folds [2,3,4,5,6,10,12,15] is
 [1.1795458641313101, 0.5984555010978514, 2.219526156064185, 0.5910703726406558, 0.5963
38005001163, 0.6058256908836257, 0.6448758386950665, 0.6905669661744517]
```




MSE changes greatly at the beginning epecially when number of folds is 3,4,5

After 5, error increases gently.

# Problem 3

I have followed the academic honesty guidelines posted on the course website