CS178 Homework #2 Bryan Oliande 13179240

January 22, 2017

Problem 1: Linear Regression

(a) Load the "data/curve80.txt" data set, and split it into 75% / 25% training/test. The first column data[:,0] is the scalar feature (x) values; the second column data[:,1] is the target value y for each example. For consistency in our results, don't reorder (shuffle) the data (they're already in a random order), and use the first 75% of the data for training and the rest for testing:

```
In [32]: import numpy as np
         import matplotlib.pyplot as plt
         import mltools as ml
         %pylab inline

         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-dimensi
         Y = data[:,1] # doesn't matter for Y
         Xtr,Xte,Ytr,Yte = ml.splitData(X,Y,0.75) # split data set 75/25
         #Ytr = np.atleast_2d(Ytr) #Ytr was made into an array
```

```
Populating the interactive namespace from numpy and matplotlib
```

(b) Use the provided linearRegress class to create a linear regression predictor of y given x. You can plot the resulting function by simply evaluating the model at a large number of x values, xs:

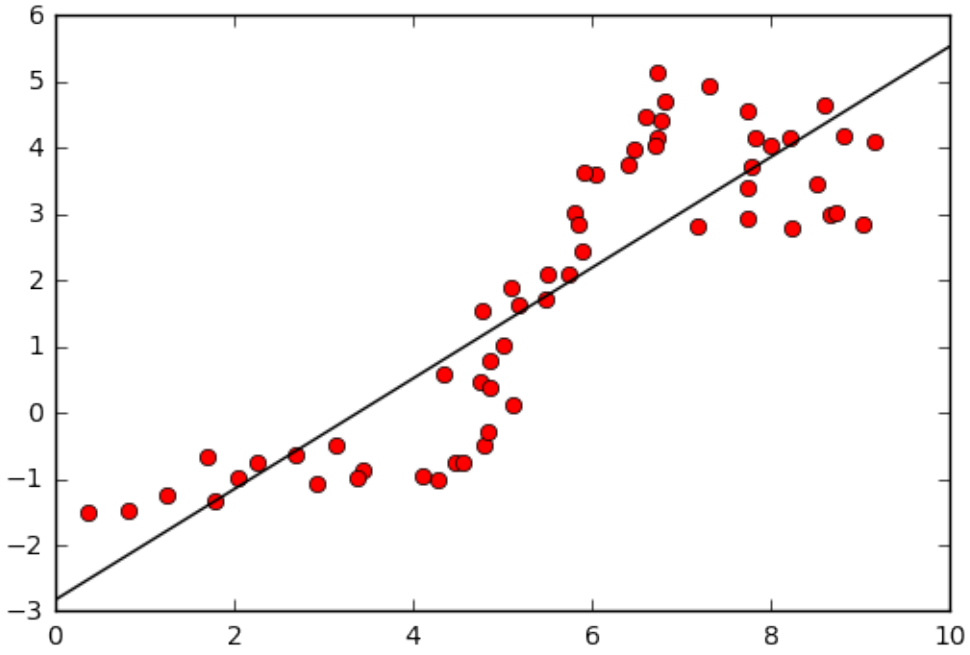
```
In [33]: lr = ml.linear.linearRegress( Xtr, Ytr ); # create and train model
         xs = np.linspace(0,10,200); # densely sample possible x-values
         xs = xs[:,np.newaxis] # force "xs" to be an Mx1 matrix (expected by our co
         ys = lr.predict( xs ); # make predictions at xs
```

Plot the training data along with your prediction function in a single plot. Print the linear regression coefficients (lr.theta) and check that they match your plot. Finally, calculate and report the mean squared error in your predictions on both the training and test data.

```
In [34]: plt.plot(Xtr, Ytr.T, 'ro', xs, ys,"black")
```

```
print("Linear Regression coefficients:")
print(lr.theta)
theta = lr.theta
m = size(Ytr)
mseTr = lr.mse(Xtr, Ytr)
mseTe = lr.mse(Xte, Yte)

print("MSE Training: ")
print(mseTr)
print("MSE Test: ")
print(mseTe)
```

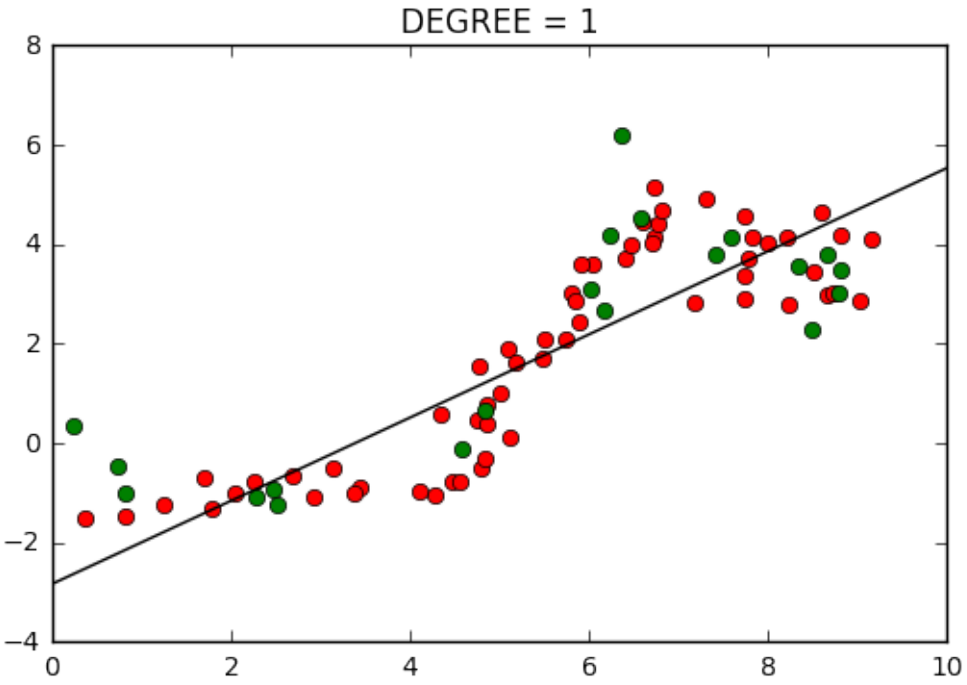
```
Linear Regression coefficients:
[[-2.82765049  0.83606916]]
MSE Training:
1.12771195561
MSE Test:
2.24234920301
```

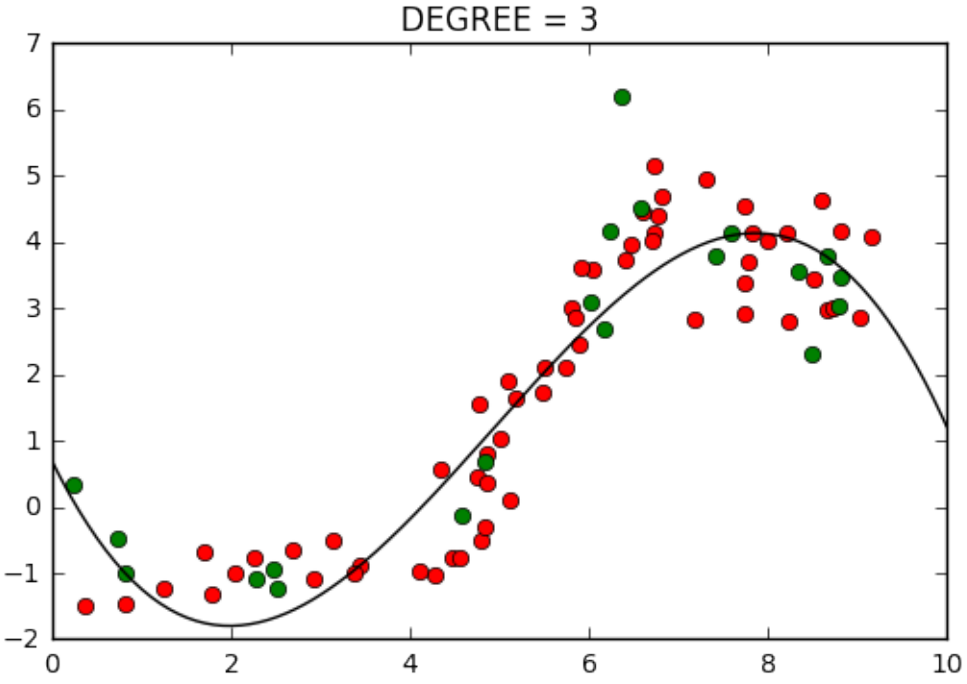


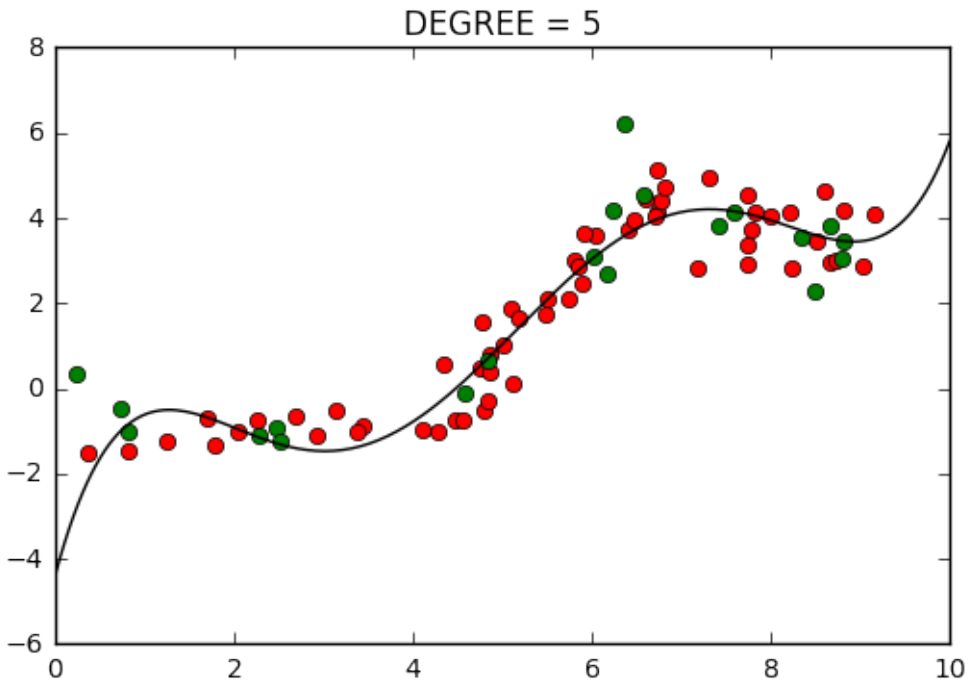
(c) Try fitting y = f(x) using a polynomial function f(x) of increasing order. Do this by the trick of adding additional polynomial features before constructing and training the linear regression object.

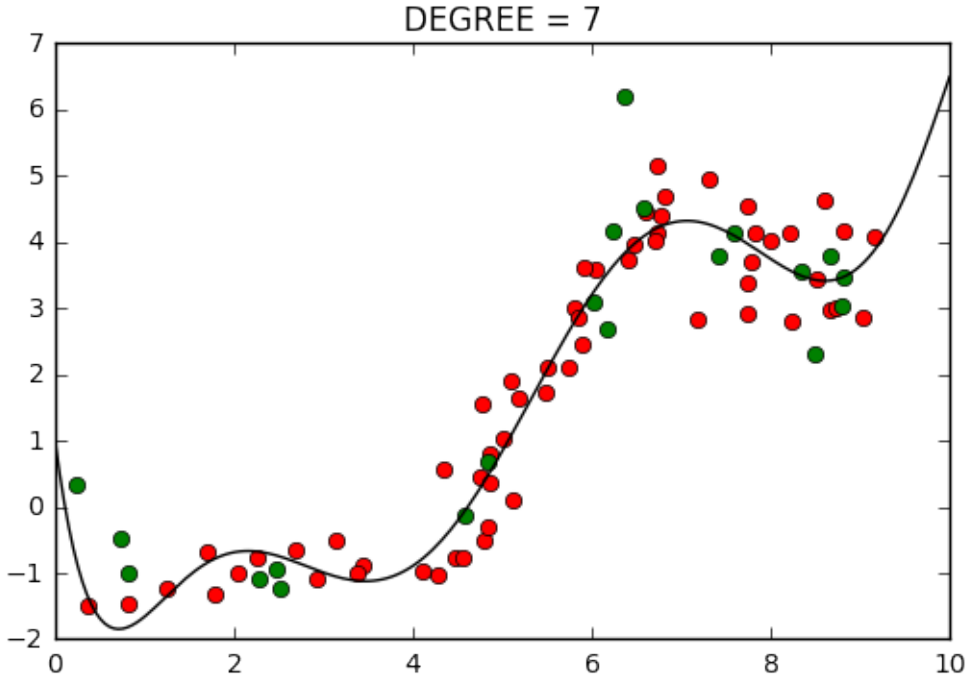
Train models of degree d = 1, 3, 5, 7, 10, 18 and (1) plot their learned prediction function f (x)

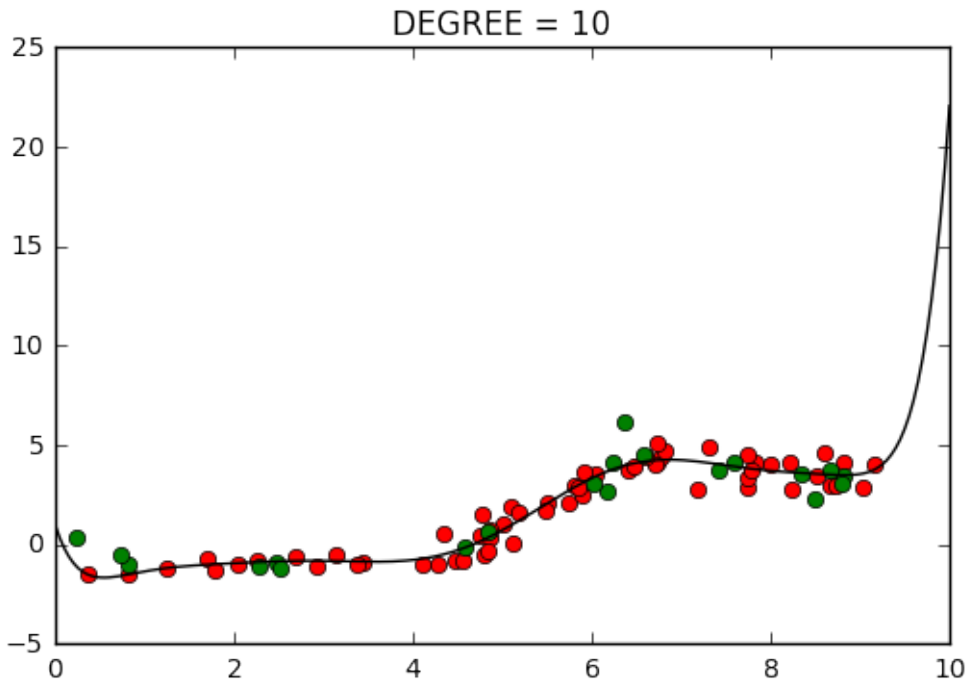
In [35]:
```
# Define a function "Phi(X)" which outputs the expanded and scaled feature
Phi = lambda X: ml.transforms.rescale( ml.transforms.fpoly(X, degree,False
                                        params)[0]
# the parameters "degree" and "params" are memorized at the function
#definition
#
Degree = [1, 3, 5, 7, 10, 18]
errTrain = [None] * len(Degree)
errTest = [None] * len(Degree)



for i,degree in enumerate(Degree):
    # Create polynomial features up to "degree"
    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);



    xs = np.linspace(0,10,200); # densely sample possible x-values
    xs = xs[:,np.newaxis] # force "xs" to be an Mx1 matrix
    ys = lr.predict( Phi(xs) ); # make predictions at xs
    errTrain[i] = lr.mse(XtrP, Ytr)
    errTest[i] = lr.mse(XteP, Yte)



    plt.title("DEGREE = " + str(degree))
    plt.plot(Xtr, Ytr.T, 'ro', Xte, Yte.T, 'go', xs, ys,"black")

    plt.show()
```

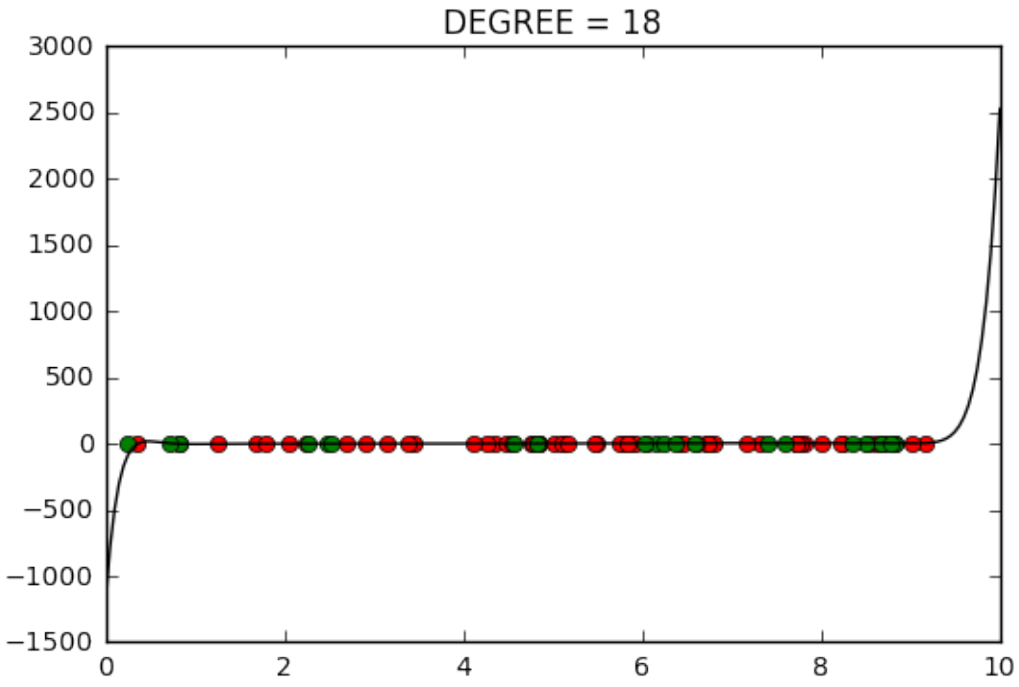

DEGREE = 1
8
6
4
2
0
−2
−4
0
2
4
6
8
10


DEGREE = 3
7
6
5
4
3
2
1
0
−1
−2
0
2
4
6
8
10


DEGREE = 5
8
6
4
2
0
−2
−4
−6
0
2
4
6
8
10

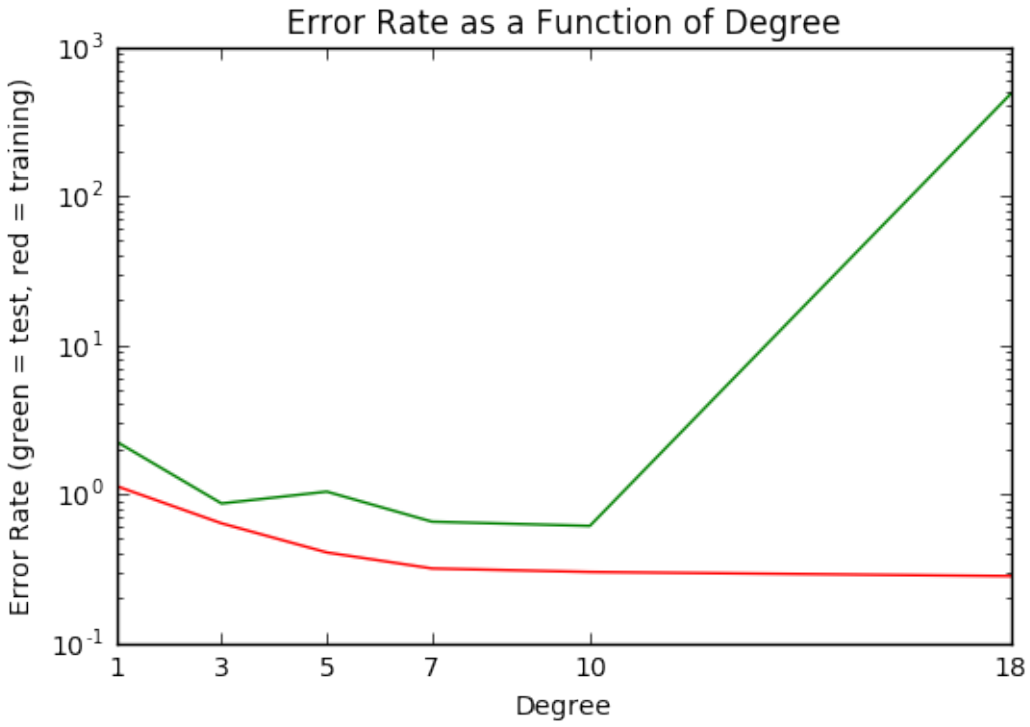

DEGREE = 7
7
6
5
4
3
2
1
0
−1
−2
0
2
4
6
8
10





and (2) their training and test errors (plot the error values on a log scale, e.g., semilogy).

```
In [36]: fig = plt.figure()
         ax = fig.add_subplot(1,1,1)
         major_ticks = [1,3,5,7,10,18]
         ax.set_xticks(major_ticks)


         plt.title("Error Rate as a Function of Degree")
         plt.xlabel("Degree")
         plt.ylabel("Error Rate (green = test, red = training)")
         plt.semilogy(Degree, errTrain, color = 'red')
         plt.semilogy(Degree, errTest, color = 'green')
         plt.show()

         #print(errTest) Was cutting off, where I obtained the values
```



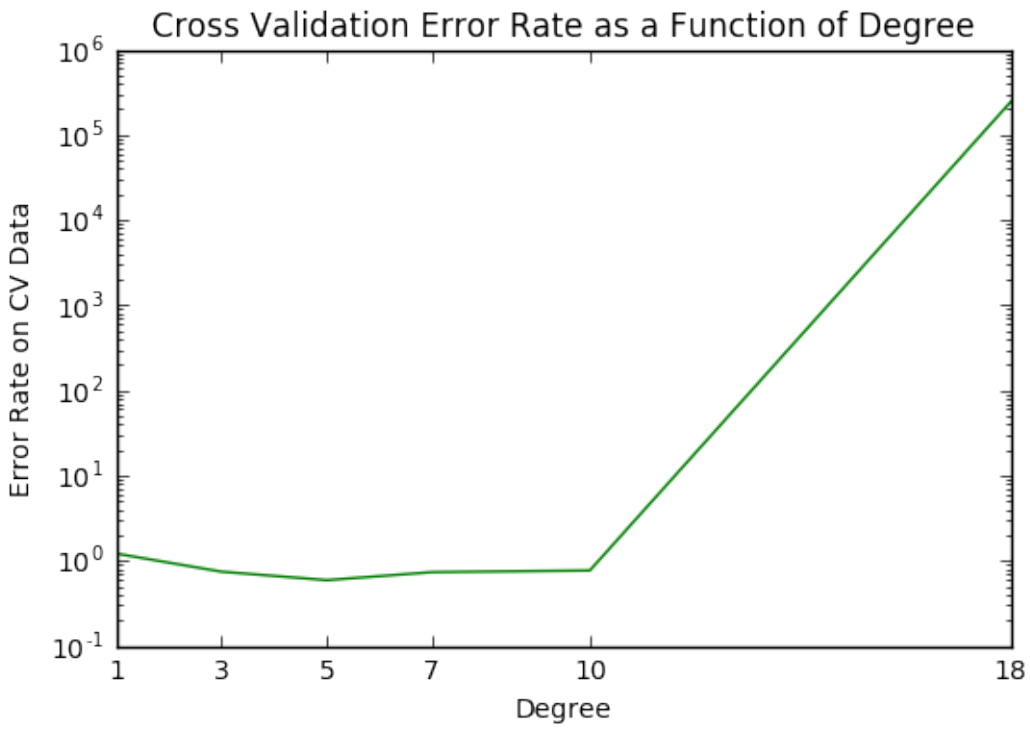
Actual errTest values for Degree = 1, 3, 5, 7, 10, 18:

[2.2423492030101251, 0.86161148154499989, 1.0344190205632156, 0.65022460796703174, 0.6090600748904027, 482.28024629144403]

Problem 2: Cross-validation

Using this technique on your training data Xtr from the previous problem, find the 5-fold cross- validation MSE of linear regression at the same degrees as before, d = 1,3,5,7,10,18 (or more densely, if you prefer). Plot the cross-validation error (with semilogy, as before) as a function of degree.

In [37]:
```
nFolds = 5;
Degree = [1, 3, 5, 7, 10, 18]
Errors = [None] * len(Degree)



for i,degree in enumerate(Degree):
    J = [None] * nFolds

    for iFold in range(nFolds):
        # take ith data block as validation
        Xti,Xvi,Yti,Yvi = ml.crossValidate(Xtr,Ytr,nFolds,iFold);

        #setting the degree
        Xti = ml.transforms.fpoly(Xti, degree, bias=False);
        Xti, params = ml.transforms.rescale(Xti)
        Xvi,_ = ml.transforms.rescale( ml.transforms.fpoly(Xvi,degree,Fals
                                       params);


        learner = ml.linear.linearRegress(Xti, Yti);

        xs = np.linspace(0,10,200); # densely sample possible x-values
        xs = xs[:,np.newaxis] # force "xs" to be an Mx1 matrix
        ys = learner.predict( Phi(xs) ); # make predictions at xs
        J[iFold] = learner.mse(Xvi, Yvi)

    Errors[i] = np.mean(J)

fig = plt.figure()
ax = fig.add_subplot(1,1,1)
major_ticks = [1,3,5,7,10,18]
ax.set_xticks(major_ticks)


plt.title("Cross Validation Error Rate as a Function of Degree")
plt.xlabel("Degree")
plt.ylabel("Error Rate on CV Data")
plt.semilogy(Degree, Errors, color = 'green')
plt.show()

#print("Cross Validation Error values for Degree = 1, 3, 5, 7, 10, 18: ")
#print(Errors) Was cutting off, where I obtained the values
```



Which degree has the minimum cross-validation error? As shown above, Degree = 5 gives the minimum cross-validation error.

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

Cross Validation Error values for Degree = 1, 3, 5, 7, 10, 18: [1.2118626629641984, 0.74290057520516606, 0.59107037264065576, 0.73356378313451243, 0.767705687615024, 244153.62141992437]

Actual test data error values for Degree = 1, 3, 5, 7, 10, 18: [2.2423492030101251, 0.86161148154499989, 1.0344190205632156, 0.65022460796703174, 0.6090600748904027, 482.28024629144403]

The MSE estimated from cross-validation performs better for degree 1, 3, and 5 polynomials. It performs slightly worse for degree 7 and 10 polynomials. For degree 18 polynomials it performs much more poorly than the actual test data.

In [ ]: