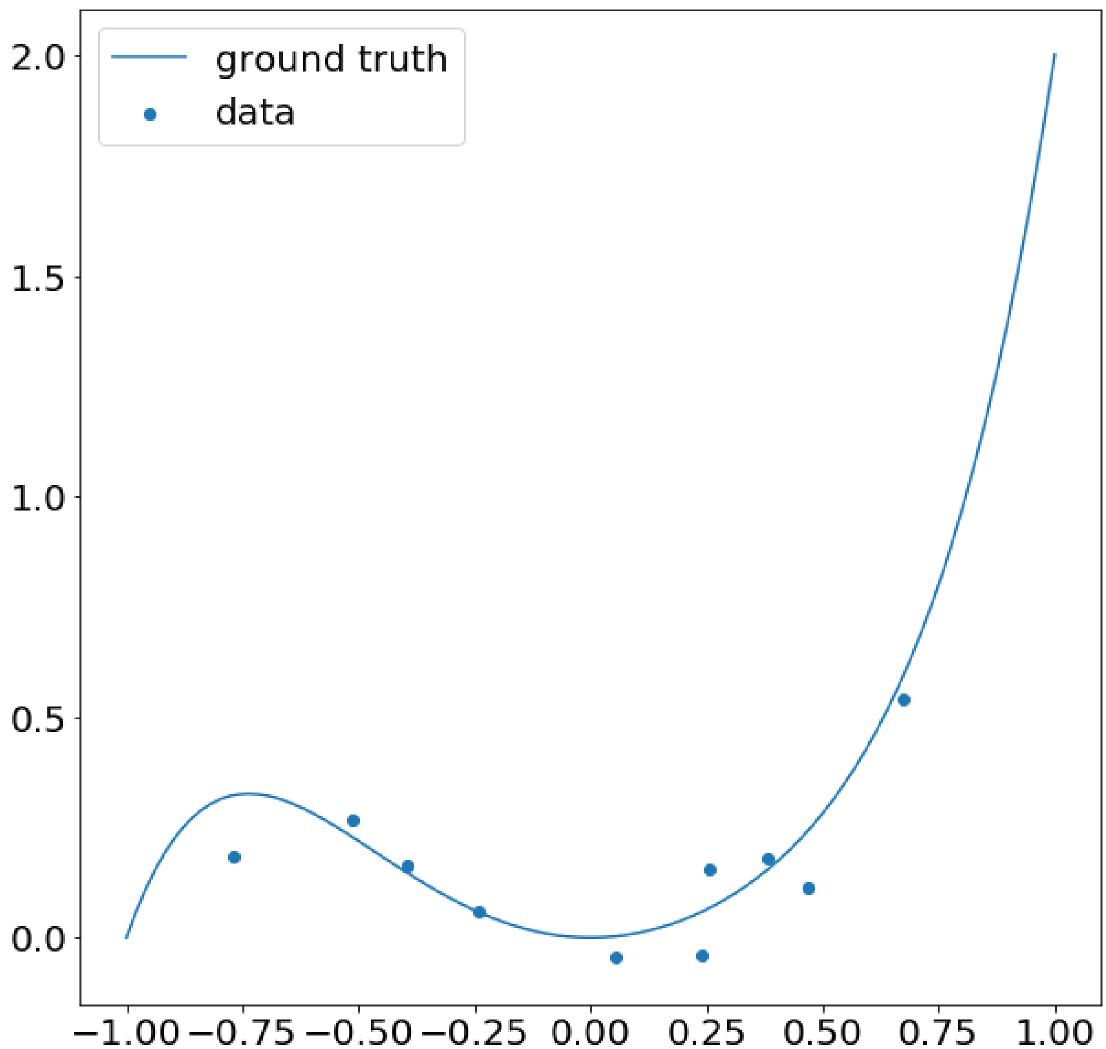
HW_2

February 7, 2018

In [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib as mpl
import numpy as np
import sklearn
import sklearn.linear_model
from sklearn import linear_model

mpl.rc('figure',figsize=(10,10))
mpl.rc('font',size=20)
```

In [2]:
```
m_training = 10
x = np.sort(np.random.uniform(-1,1,m_training))#np.linspace(-1,1,m_training)
y = x**2 + x**5 + np.random.normal(scale = .1,size=m_training)
X = np.linspace(-1,1,100)
Y = X**2 + X**5

plt.scatter(x,y,label='data')
plt.plot(X,Y,label='ground truth')
plt.legend()
plt.show()
```

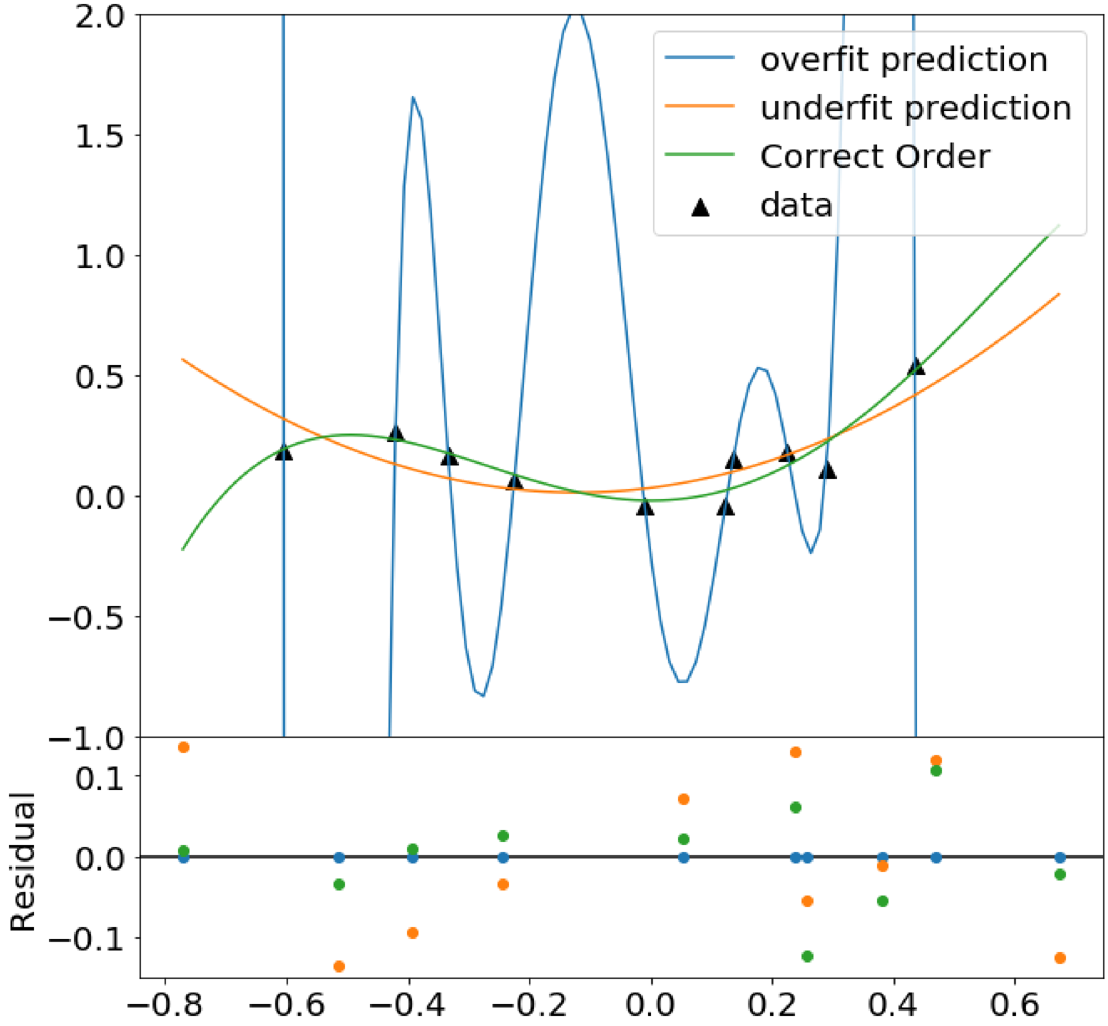



```
In [3]: def poly_basis(X, d):
            """Returns a polynomial of degree d-1.

            Args:
                X: data array, that is n
                d: degree of the polynomial
            Returns:
                coefficient matrix of the polynomials n x d """
            return np.power(np.expand_dims(X,1), np.arange(0, d))
```

```
In [4]: poly_order = 40
        A = poly_basis(x,poly_order)
        answer_40 = sklearn.linear_model.LinearRegression().fit(A, y)
        y_train_40 =answer_40.predict(poly_basis(X,poly_order))
```

```
poly_order = 3
A3 = poly_basis(x,poly_order)
answer_3 = sklearn.linear_model.LinearRegression().fit(A3, y)
y_train_3 =answer_3.predict(poly_basis(X,poly_order))

poly_order = 5
A5 = poly_basis(x,poly_order)
answer_5 = sklearn.linear_model.LinearRegression().fit(A5, y)
y_train_5 =answer_5.predict(poly_basis(X,poly_order))
```

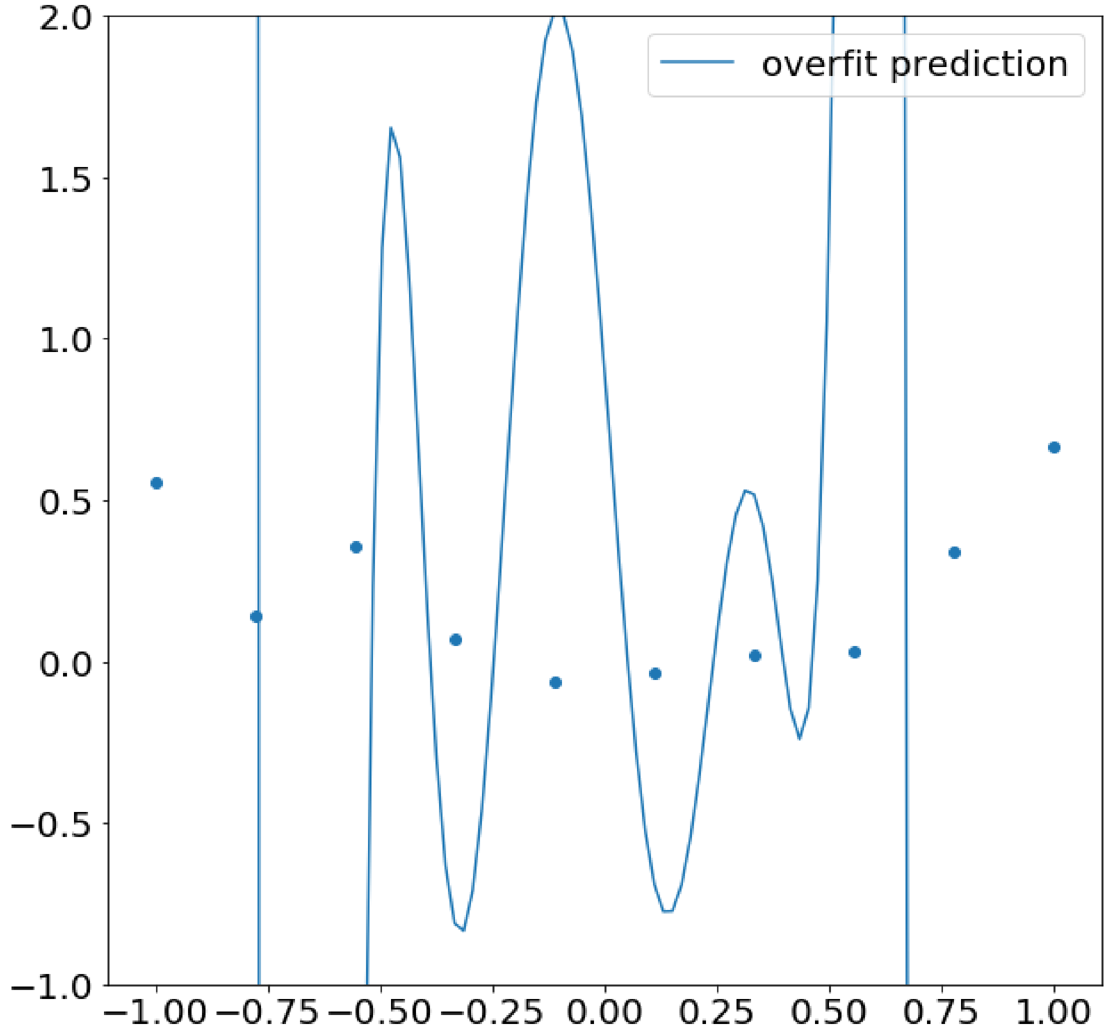
In [5]:
```
fig = plt.figure(1)
#Plot Data-model
frame1=fig.add_axes((.1,.3,.8,.6))
#xstart, ystart, xend, yend [units are fraction of the image frame, from bottom left cor
plt.scatter(x,y,marker='^',s=100,color='k',label='data')
plt.plot(X,y_train_40,label='overfit prediction')
plt.plot(X,y_train_3,label='underfit prediction')
plt.plot(X,y_train_5,label='Correct Order')
plt.ylim([-1,2])
frame1.set_xticklabels([]) #Remove x-tic labels for the first frame
plt.legend()
#Residual plot
difference40 = answer_40.predict(poly_basis(x,40)) - y
difference3 = answer_3.predict(poly_basis(x,3)) - y
difference5 = answer_5.predict(poly_basis(x,5)) - y
frame2=fig.add_axes((.1,.1,.8,.2))
plt.axhline(0,color='k')
plt.plot(x,difference40,'o')
plt.plot(x,difference3,'o')
plt.plot(x,difference5,'o')
plt.ylabel('Residual')
plt.legend()
```



0.0.1 From the residuals we can clearly see that the overfit prediction hits every point exactly, while the under fit prediction does worse than the correct order fit.

0.1 Generate a second set of data, a test set.

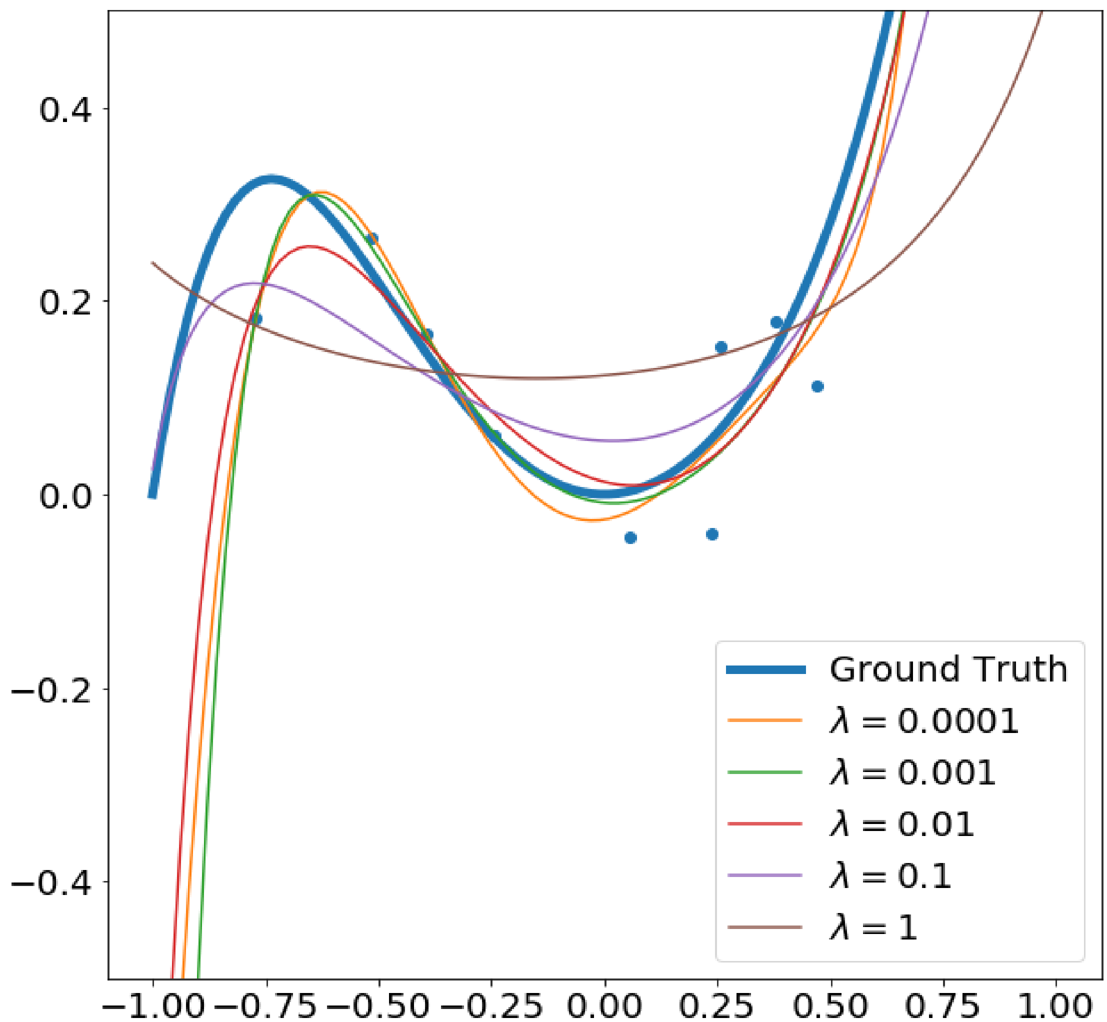
In [6]:
```
x_test = np.linspace(-1,1,m_training)
y_test = x**2 + x**5 + np.random.normal(scale = .1,size=m_training)
plt.scatter(x_test,y_test)
plt.plot(X,y_train_40,label='overfit prediction')
plt.legend()
plt.ylim([-1,2])

plt.show()
```



Now it misses the points :(

```
In [11]: # Write a function to make this all go quicker
         def ridge_regress_compare(x,y,n_order,lambda_reg,X):
             A = poly_basis(x,n_order)
             ridge = linear_model.Ridge(alpha=lambda_reg)
             fit = ridge.fit(A,y)
             pred = fit.predict(poly_basis(X,n_order))
             plt.plot(X,pred,label=r'$\lambda=${:}'.format(lambda_reg))
         plt.scatter(x,y)
         plt.plot(X,Y,label='Ground Truth',lw=5)
         ridge_regress_compare(x,y,40,0.0001,X)
         ridge_regress_compare(x,y,40,.001,X)
         ridge_regress_compare(x,y,40,.01,X)
         ridge_regress_compare(x,y,40,.1,X)
```

```
ridge_regress_compare(x,y,40,1,X)
plt.legend()
plt.ylim([-.5,.5])
plt.show()
```

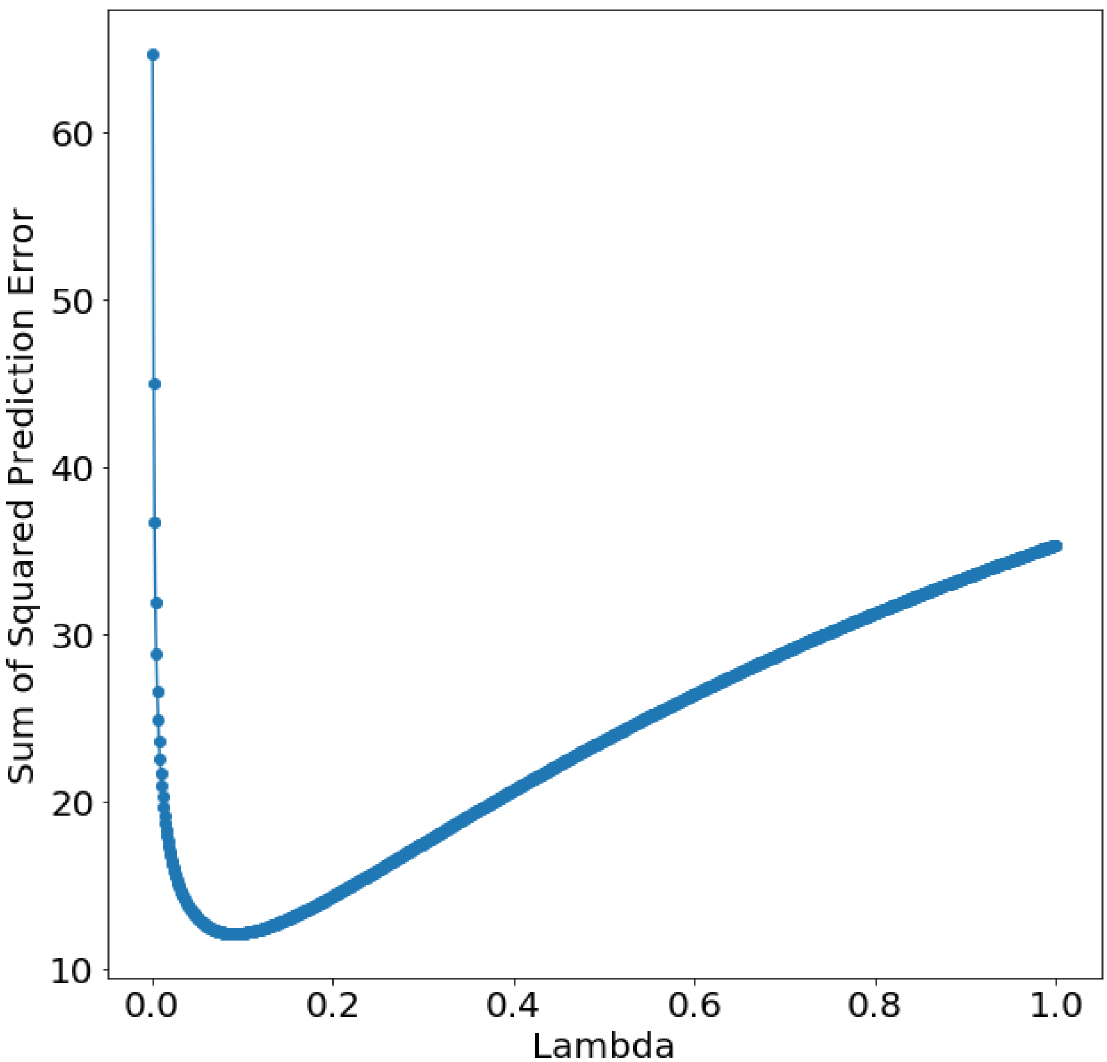


0.1.1 2.d assessing which value of lambda is best.

To do this I'll compare the performance of each fit on a test set (really a validation set)

```
In [27]: def get_mod(x_train,y_train, n_order, lambda_reg):
             A = poly_basis(x_train,n_order)
             ridge = linear_model.Ridge(alpha=lambda_reg)
             fit = ridge.fit(A,y)
             return fit
         x_test = np.linspace(-1,1,1000)
```

```
y_test = x_test**2 + x_test**5 + np.random.normal(scale = .1,size=1000)
x = np.linspace(-1,1,m_training)
y = x**2 + x**5 + np.random.normal(scale = .1,size=m_training)
models = {}
lambdas = np.linspace(0.001,1,1000)
pred_errs =np.zeros_like(lambdas)
for i, lam in enumerate(lambdas):
    fit = get_mod(x,y,40,lambda_reg=lam)
    models['{:}'.format(lam)] = fit
    y_pred = fit.predict(poly_basis(x_test,40))
    pred_errs[i] = np.sum((y_test-y_pred)**2)
#     print('{:0.2}\t{:0.3}'.format(lam,pred_errs[i]))
plt.plot(lambdas,pred_errs,'-o')
plt.ylabel('Sum of Squared Prediction Error')
plt.xlabel('Lambda')
plt.show()
```



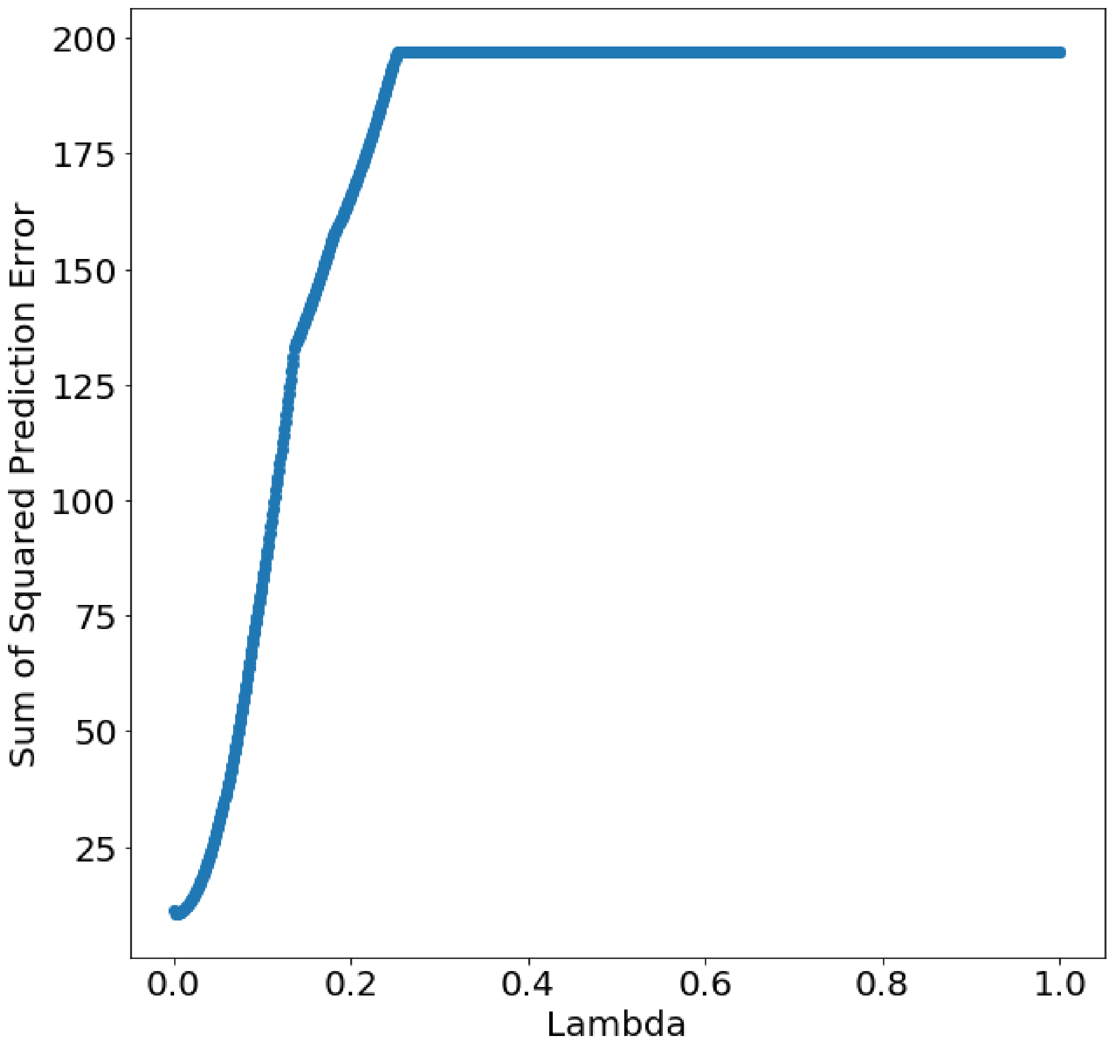
```
In [29]: #Find the best lambda
         print(lambdas[np.argmin(pred_errs)])
```

```
0.091
```

0.1.2 So i find the best lambda value to be .91 and note that it makes a significant difference over a lambda of 0.

1 1. e Lasso Regularization

```
In [30]: def get_mod_L1(x_train,y_train, n_order, lambda_reg):
             A = poly_basis(x_train,n_order)
             lasso = linear_model.Lasso(alpha=lambda_reg)
             fit = lasso.fit(A,y)
             return fit

         models = {}
         lambdas = np.linspace(0.001,1,1000)
         pred_errs =np.zeros_like(lambdas)
         for i, lam in enumerate(lambdas):
             fit = get_mod_L1(x,y,40,lambda_reg=lam)
             models['{:}'.format(lam)] = fit
             y_pred = fit.predict(poly_basis(x_test,40))
             pred_errs[i] = np.sum((y_test-y_pred)**2)
         #     print('{:0.2}\t{:0.3}'.format(lam,pred_errs[i]))
         plt.plot(lambdas,pred_errs,'-o')
         plt.ylabel('Sum of Squared Prediction Error')
         plt.xlabel('Lambda')
         plt.show()
```



1.0.1 Weird, for this L1 where things are already sparse it seems that not regularizing is best.

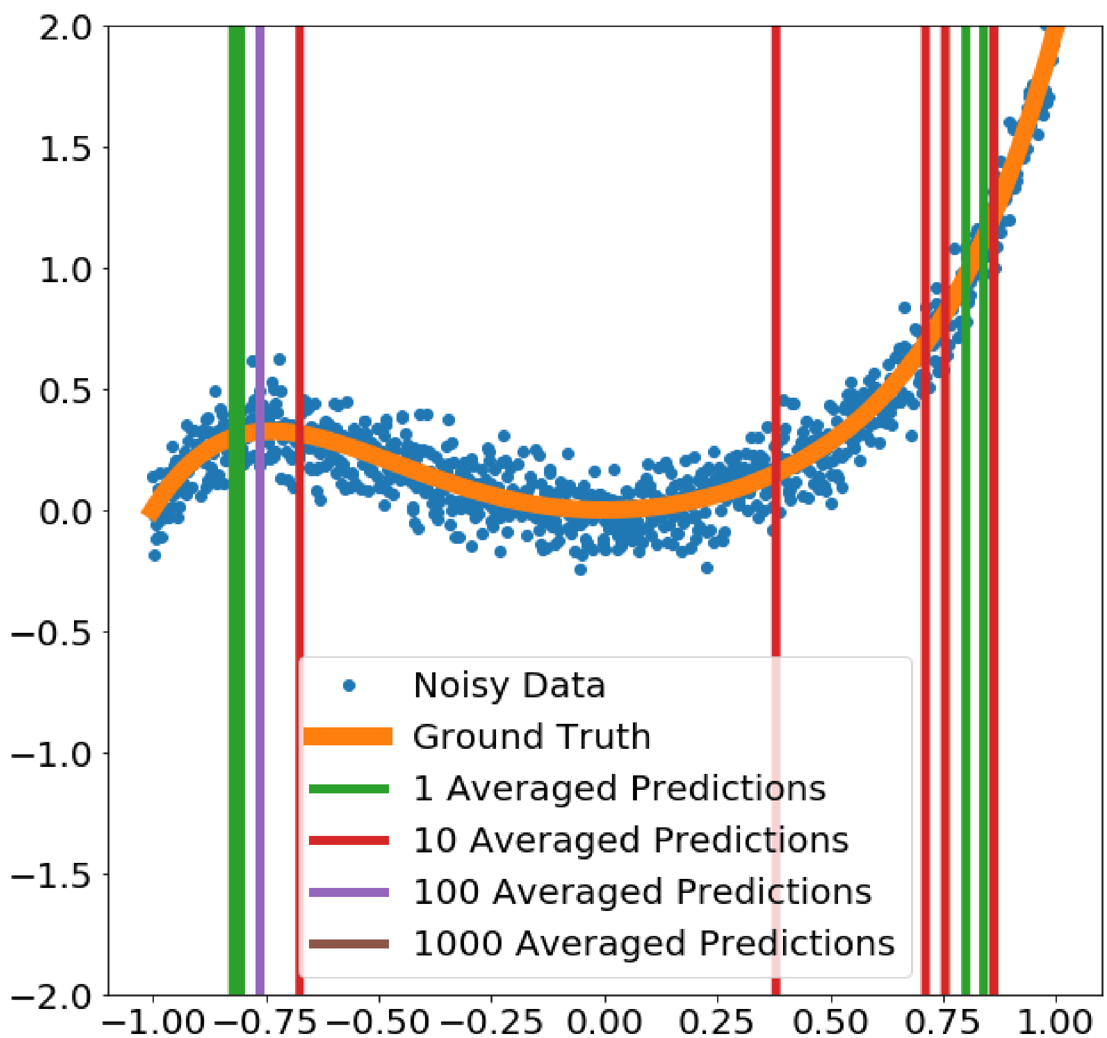
2 2.a

What follows are two different approaches to averaging. The first I randomly sample some number of points from [-1,1] for my training data. I found that with the order of my polynomial > n_training_points this lead to wildly incorrect averages as below.

```
In [33]: def get_new_fit_coef(n_pts = 20,order = 16):
             x = np.sort(np.random.uniform(-1,1,n_pts))
         #     x = np.linspace(-1,1,n_pts)+np.random.normal(scale = .1,size=n_pts)
             y = x**2 + x**5 + np.random.normal(scale = .1,size=n_pts)
             A = poly_basis(x,order)
             fit = sklearn.linear_model.LinearRegression().fit(A, y)
             return fit.coef_
```

```
order = 50
avg_numbers = np.array([1,10,100,1000])
n_tries = avg_numbers.shape[0]
a = np.zeros([n_tries, get_new_fit_coef(order=order).shape[0]])
for i, n_avg in enumerate(avg_numbers):
    for j in range(n_avg):
        a[i] += get_new_fit_coef(order=order)
    a[i] /= n_avg
```

In [34]:
```
plt.plot(x_test,y_test,'o',label='Noisy Data')
plt.plot(X,Y,label='Ground Truth',lw=10)

for i, n_avg in enumerate(avg_numbers):
    plt.plot(x_test,np.dot(poly_basis(x_test,order),a[i].T),label='{:} Averaged Predict

plt.legend()
plt.ylim([-2,2])
plt.show()
```



2.1 Proper averaging:

```
In [38]: def get_new_fit_coef_2(n_pts = 20,order = 16):
         #     x = np.random.uniform(-1,1,n_pts)
             x = np.linspace(-1,1,n_pts)+np.random.normal(scale = .2,size=n_pts)
             y = x**2 + x**5 + np.random.normal(scale = .1,size=n_pts)
             A = poly_basis(x,order)
         #     fit = sklearn.linear_model.LinearRegression().fit(A, y)
             coef=np.matmul(np.linalg.pinv(A),y)[::-1]

             return x,y, coef
         order = 7
         n_pts = 6
         N = 1000

         data_x = np.zeros([N,n_pts])
         data_y = np.zeros([N,n_pts])
         a = np.zeros([N,get_new_fit_coef(order=order,n_pts = n_pts).shape[0]])
         for j in range(N):
             out = get_new_fit_coef_2(order=order,n_pts=n_pts)
             data_x[j] = out[0]
             data_y[j] = out[1]
             a[j] += out[2]
         coef = np.average(a,axis=0)
         for A in a:
             plt.plot(X,np.polyval(A,X),alpha=.1,color='black')
         pred = np.polyval(coef,X)#fit.predict(poly_basis(X,order))
         plt.scatter(data_x.flatten(),data_y.flatten(),s=15,color = 'orange',label='Noisy Data')
         plt.plot(X,pred,lw=10,label='Averaged Coefficients')
         plt.plot(X,X**2+X**5,label='Ground Truth')

         plt.ylim([-1,1])
         plt.xlim([-1,1])
         plt.legend()
         plt.show()
```