Lab: Model Selection for Neural Data

Machine learning is a key tool for neuroscientists to understand how sensory and motor signals are encoded in the brain. In addition to improving our scientific understanding of neural phenomena, understanding neural encoding is critical for brain machine interfaces. In this lab, you will use model selection for performing some simple analysis on real neural signals.

Before doing this lab, you should review the ideas in the polynomial model selection demo (./polyfit.ipynb). In addition to the concepts in that demo, you will learn to:

- Load MATLAB data
- Formulate models of different complexities using heuristic model selection
- Fit a linear model for the different model orders
- Select the optimal model via cross-validation

The last stage of the lab uses LASSO estimation for model selection. If you are doing this part of the lab, you should review the concepts in LASSO demonstration (./prostate.ipynb) on the prostate cancer dataset.

Loading the data

The data in this lab comes from neural recordings described in:

Stevenson, Ian H., et al. "Statistical assessment of the stability of neural movement representations." Journal of neurophysiology 106.2 (2011): 764-774 (http://jn.physiology.org/content/106/2/764.short)

Neurons are the basic information processing units in the brain. Neurons communicate with one another via *spikes* or *action potentials* which are brief events where voltage in the neuron rapidly rises then falls. These spikes trigger the electro-chemical signals between one neuron and another. In this experiment, the spikes were recorded from 196 neurons in the primary motor cortex (M1) of a monkey using an electrode array implanted onto the surface of a monkey's brain. During the recording, the monkey performed several reaching tasks and the position and velocity of the hand was recorded as well.

The goal of the experiment is to try to *read the monkey's brain*: That is, predict the hand motion from the neural signals from the motor cortex.

We first load the basic packages.

In [2]:
```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
```

The full data is available on the CRCNS website http://crcns.org/data-sets/movements/dream (http://crcns.org/data-sets/movements/dream). This website has a large number of great datasets and can be used for projects as well. To make this lab easier, I have pre-processed the data slightly and placed it in the file `StevensonV2.mat`, which is a MATLAB file. You will need to have this file downloaded in the directory you are working on.

Since MATLAB is widely-used, `python` provides method for loading MATLAB `mat` files. We can use these commands to load the data as follows.

In [3]:
```
import scipy.io
mat_dict = scipy.io.loadmat('StevensonV2.mat')
```

The returned structure, `mat_dict`, is a dictionary with each of the MATLAB variables that were saved in the `.mat` file. Use the `.keys()` method to list all the variables.

In [4]:
```
mat_dict.keys()
```

Out[4]:
```
dict_keys(['__header__', '__version__', '__globals__', 'Publication'
, 'timeBase', 'spikes', 'time', 'handVel', 'handPos', 'target', 'sta
rtBins', 'targets', 'startBinned'])
```

We extract two variables, `spikes` and `handVel`, from the dictionary `mat_dict`, which represent the recorded spikes per neuron and the hand velocity. We take the transpose of the spikes data so that it is in the form time bins $\times$ number of neurons. For the `handVel` data, we take the first component which is the motion in the x-direction.

In [5]:
```
X0 = mat_dict['spikes'].T
#print(X0)

#print(X1)

y0 = mat_dict['handVel'][0,:].T# this is ydat
#y0 = mat_dict['handVel'][0,:]
#y0 = np.transpose(y0)
print(y0.shape[0])

print(y0)
X0.shape
```

```
15536
[-0.0112006  -0.01074321  0.01767953 ...,  0.05812657  0.05378452
  0.04268675]
```

Out[5]:
```
(15536, 196)
```

The `spikes` matrix will be a `nt x neuron` matrix where `nt` is the number of time bins and `neuron` is the number of neurons. Each entry `spikes[k,j]` is the number of spikes in time bin `k` from neuron `j`. Use the `shape` method to find `nt` and `nneuron` and print the values.

In [6]:
```
nt, neuron = X0.shape
print("num nt={0:d}  num neuron={1:d}".format(nt,neuron))
```

```
num nt=15536  num neuron=196
```

Now extract the `time` variable from the `mat_dict` dictionary. Reshape this to a 1D array with `nt` components. Each entry `time[k]` is the starting time of the time bin `k`. Find the sampling time `tsamp` which is the time between measurements, and `ttotal` which is the total duration of the recording.

In [7]:
```
Time = mat_dict['time']

#time = Time.shape[0]

Time = Time.reshape(nt)
#print(Time)
#for t in Time:
#   print (t)

#need to find the tsamp
#first take the difference between the time samples
# time iteration = (0.05, 0.1, 0.15....)
tsamp= Time[2]-Time[1]
print(tsamp)

#need to find the ttotal
#subtract the initial time t0 from the last time t15536
#print(Time[nt-1])
#ttotal = Time[15535] - Time[0]
#print(ttotal +tsamp)
ttotal = Time[15535] - Time[0] + tsamp
print(ttotal)
```

```
0.05
776.8
```

Linear fitting on all the neurons

First divide the data into training and test with approximately half the samples in each. Let `Xtr` and `ytr` denote the training data and `Xts` and `yts` denote the test data.

In [8]:
```
#Xtr = np.random.choice(Time, 7768) #random training set (15536/2)half
of the training set Time
Xtr = X0[:7768,:]#spikes
print(Xtr)

#ytr is a product of np.array(something) then i use a np.random.choice
in the same range of samples as time

ytr = y0[:7768]


Xts = X0[7768:15536,:]  #random test set

yts = y0[7768:15536]


#print(ytr)
#print(Xts)

#print(yts)
#print(Xts)
# yts = ...
#yts = y[:7768]


#plt.scatter(Xtr[:,1],ytr)
#plt.xlabel('x')
#plt.ylabel('y')
#plt.grid()
#plt.legend(['True (dtrue=3)', 'Data'], loc='upper left')
#plt.show()
```

```
[[1 0 2 ..., 2 0 2]
 [3 1 1 ..., 0 0 0]
 [1 0 1 ..., 0 0 3]
 ...,
 [1 0 0 ..., 0 0 6]
 [0 0 0 ..., 0 0 5]
 [0 0 0 ..., 0 0 2]]
```

Now, we begin by trying to fit a simple linear model using *all* the neurons as predictors. To this end, use the `sklearn.linear_model` package to create a regression object, and fit the linear model to the training data.

In [9]:
```
import sklearn.linear_model
#from sklearn import linear_model
# TODO

regr = sklearn.linear_model.LinearRegression()
regr.fit(Xtr,ytr)
```

Out[9]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normaliz
e=False)
```

Measure and print the normalized RSS on the test data.

In [10]:
```
# fit isn't working
y_ts_pred = regr.predict(Xts)
RSS_rel_ts = np.mean((yts-y_ts_pred)**2)/(np.std(yts)**2)
print("Normalized test RSS = {0:f}".format(RSS_rel_ts))
```

```
Normalized test RSS = 4999054779645876502528.000000
```

You should see that the test error is enormous -- the model does not generalize to the test data at all.

Linear Fitting with Heuristic Model Selection

The above shows that we need a way to reduce the model complexity. One simple idea is to select only the neurons that individually have a high correlation with the output.

Write code which computes the coefficient of determination, R_k^2, for each neuron k. Plot the R_k^2 values.

You can use a for loop over each neuron, but if you want to make efficient code try to avoid the for loop and use python broadcasting (../Basics/numpy_axes_broadcasting.ipynb).

In [11]:
```
ym = np.mean(y0)
syy = np.mean((y0-ym)**2)
Rsq = np.zeros(neuron)
beta0 = np.zeros(neuron)
beta1 = np.zeros(neuron)
for k in range(neuron):
    xm = np.mean(X0[:,k])
    #print(xm)
    sxy = np.mean((X0[:,k]-xm)*(y0-ym))
    sxx = np.mean((X0[:,k]-xm)**2)
    beta1[k] = sxy/sxx
    beta0[k] = ym - beta1[k]*xm
    Rsq[k] = (sxy)**2/sxx/syy

    print("{0:2d}  Rsq={1:f}".format(k,Rsq[k]))

#for d: 1->
# xtemp = x1[1:10d,:]
# for i: 1-10 #crossvalidaiton
# RSS = RSS[i,d]
#RSS[19,10]


#plot stem
x = range (neuron)
plt.stem(x,Rsq,'-.')
plt.grid()
```

```
 0  Rsq=0.006049
 1  Rsq=0.017449
 2  Rsq=0.017310
 3  Rsq=0.014403
 4  Rsq=0.013202
 5  Rsq=0.000047
 6  Rsq=0.026373
 7  Rsq=0.000008
 8  Rsq=0.000226
 9  Rsq=0.000082
10  Rsq=0.000698
11  Rsq=0.000221
12  Rsq=0.000438
13  Rsq=0.000002
14  Rsq=0.003749
15  Rsq=0.000232
16  Rsq=0.000438
17  Rsq=0.000081
18  Rsq=0.000427
19  Rsq=0.000002
20  Rsq=0.005763
```

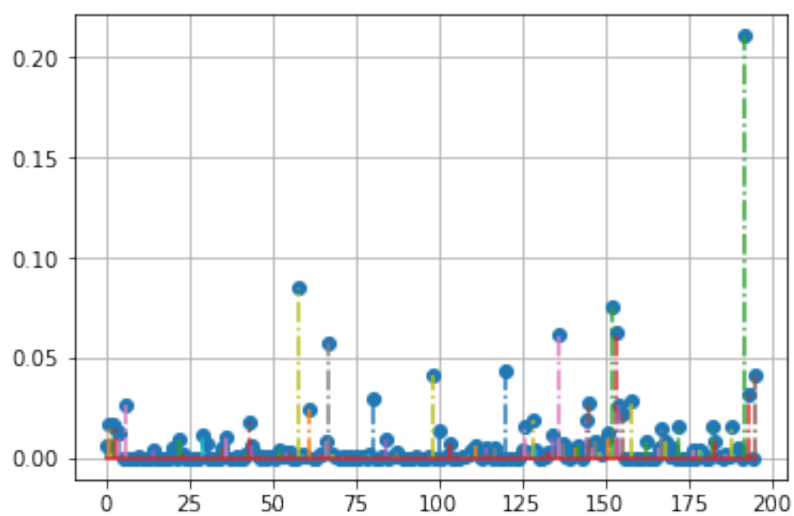
```
21  Rsq=0.000008
22  Rsq=0.009351
23  Rsq=0.001622
24  Rsq=0.000000
25  Rsq=0.000202
26  Rsq=0.000149
27  Rsq=0.000133
28  Rsq=0.000011
29  Rsq=0.011838
30  Rsq=0.006994
31  Rsq=0.000054
32  Rsq=0.000048
33  Rsq=0.000013
34  Rsq=0.000000
35  Rsq=0.005588
36  Rsq=0.011034
37  Rsq=0.000008
38  Rsq=0.000161
39  Rsq=0.001175
40  Rsq=0.000047
41  Rsq=0.000000
42  Rsq=0.001871
43  Rsq=0.018205
44  Rsq=0.006378
45  Rsq=0.001970
46  Rsq=0.000140
47  Rsq=0.000926
48  Rsq=0.000001
49  Rsq=0.000731
50  Rsq=0.001064
51  Rsq=0.000006
52  Rsq=0.004139
53  Rsq=0.000512
54  Rsq=0.003184
55  Rsq=0.002781
56  Rsq=0.000032
57  Rsq=0.000030
58  Rsq=0.084733
59  Rsq=0.001594
60  Rsq=0.001124
61  Rsq=0.024546
62  Rsq=0.000178
63  Rsq=0.000049
64  Rsq=0.001727
65  Rsq=0.002048
66  Rsq=0.008399
67  Rsq=0.057849
68  Rsq=0.002292
69  Rsq=0.001084
70  Rsq=0.000058
```

```
71  Rsq=0.001408
72  Rsq=0.000004
73  Rsq=0.000852
74  Rsq=0.000006
75  Rsq=0.000751
76  Rsq=0.000086
77  Rsq=0.000532
78  Rsq=0.000016
79  Rsq=0.001977
80  Rsq=0.029364
81  Rsq=0.000013
82  Rsq=0.000000
83  Rsq=0.001138
84  Rsq=0.009751
85  Rsq=0.000007
86  Rsq=0.001297
87  Rsq=0.003312
88  Rsq=0.000615
89  Rsq=0.000010
90  Rsq=0.000198
91  Rsq=0.000141
92  Rsq=0.000008
93  Rsq=0.000454
94  Rsq=0.000021
95  Rsq=0.000203
96  Rsq=0.000113
97  Rsq=0.001216
98  Rsq=0.041412
99  Rsq=0.000171
100  Rsq=0.013749
101  Rsq=0.000050
102  Rsq=0.000506
103  Rsq=0.007248
104  Rsq=0.000006
105  Rsq=0.000091
106  Rsq=0.000092
107  Rsq=0.000495
108  Rsq=0.000760
109  Rsq=0.001632
110  Rsq=0.004300
111  Rsq=0.006488
112  Rsq=0.000781
113  Rsq=0.000065
114  Rsq=0.005248
115  Rsq=0.001521
116  Rsq=0.000404
117  Rsq=0.005194
118  Rsq=0.000107
119  Rsq=0.000274
120  Rsq=0.043481
```

```
121  Rsq=0.000208
122  Rsq=nan
123  Rsq=0.000251
124  Rsq=0.000239
125  Rsq=0.003851
126  Rsq=0.015655
127  Rsq=0.002613
128  Rsq=0.018871
129  Rsq=0.004513
130  Rsq=0.000000
131  Rsq=0.000161
132  Rsq=0.001233
133  Rsq=0.002636
134  Rsq=0.011346
135  Rsq=0.003845
136  Rsq=0.062266
137  Rsq=0.007324
138  Rsq=0.000570
139  Rsq=0.000015
140  Rsq=0.000891
141  Rsq=0.005568
142  Rsq=0.006446
143  Rsq=0.000384
144  Rsq=0.019351
145  Rsq=0.027149
146  Rsq=0.005472
147  Rsq=0.008138
148  Rsq=0.002579
149  Rsq=0.002082
150  Rsq=0.009285
151  Rsq=0.012662
152  Rsq=0.076110
153  Rsq=0.062699
154  Rsq=0.026915
155  Rsq=0.022048
156  Rsq=0.000025
157  Rsq=0.000176
158  Rsq=0.028348
159  Rsq=0.000003
160  Rsq=0.000199
161  Rsq=0.000240
162  Rsq=0.008190
163  Rsq=0.000027
164  Rsq=0.000006
165  Rsq=0.000050
166  Rsq=0.004791
167  Rsq=0.015360
168  Rsq=0.008925
169  Rsq=0.000566
170  Rsq=0.004922
```

```
171  Rsq=0.000000
172  Rsq=0.016245
173  Rsq=0.000024
174  Rsq=0.000001
175  Rsq=0.000253
176  Rsq=0.004500
177  Rsq=0.000005
178  Rsq=0.004516
179  Rsq=0.002839
180  Rsq=0.000009
181  Rsq=0.000033
182  Rsq=0.016409
183  Rsq=0.008125
184  Rsq=0.000704
185  Rsq=0.000267
186  Rsq=0.001975
187  Rsq=0.001295
188  Rsq=0.016119
189  Rsq=0.001894
190  Rsq=0.004888
191  Rsq=0.000049
192  Rsq=0.210846
193  Rsq=0.031629
194  Rsq=0.000219
195  Rsq=0.041982
```

```
/anaconda/lib/python3.6/site-packages/ipykernel_launcher.py:11: Runt
imeWarning: invalid value encountered in double_scalars
  # This is added back by InteractiveShellApp.init_path()
/anaconda/lib/python3.6/site-packages/ipykernel_launcher.py:13: Runt
imeWarning: invalid value encountered in double_scalars
  del sys.path[0]
```



We see that many neurons have low correlation and can probably be discarded from the model.

Use the `np.argsort()` command to find the indices of the `d=100` neurons with the highest R_k^2 value. Put the `d` indices into an array `Isel`. Print the indices of the neurons with the 10 highest correlations(meaning the highest Rsq value).

In [12]:
```
d = 100  # Number of neurons to use

Isel = np.zeros(d)
#################
# the Rsq at 122 is undefined because the sxy is 0
# therefore i am going to omit it from the data
#################

Rsq[122]= 100.0000 ###need to omit it
#print(Rsq)
Indecies = np.argsort(-Rsq)
###########returns the index of the values from low to high need to ne
gate#####

#print(Indecies.shape)
#for k in range(len(Indecies)):
#    print("{0:2d}  Highest Rsq={1:f}".format(k,Indecies[k]))
#for k in range(d):
#    Isel[k] = Rsq[Indecies[k]]


#print("The neurons with the ten highest R^2 values = ...)

#for k in range(96,neuron):
    #print("{0:2d}  Highest 100 Rsq={1:f}".format(k,Rsq[Indecies[k]]))
#Isel = Indecies[neuron-d:]

Isel = Indecies[:d]
#############indecies to 100##############

for k in range(186,neuron):
    print("{0:2d}  index with highest correlations {1:d}".format(k,Ind
ecies[k]))
print(Rsq[122])
```

```
186  index with highest correlations 13
187  index with highest correlations 19
188  index with highest correlations 174
189  index with highest correlations 48
190  index with highest correlations 82
191  index with highest correlations 41
192  index with highest correlations 34
193  index with highest correlations 24
194  index with highest correlations 171
195  index with highest correlations 130
100.0
```

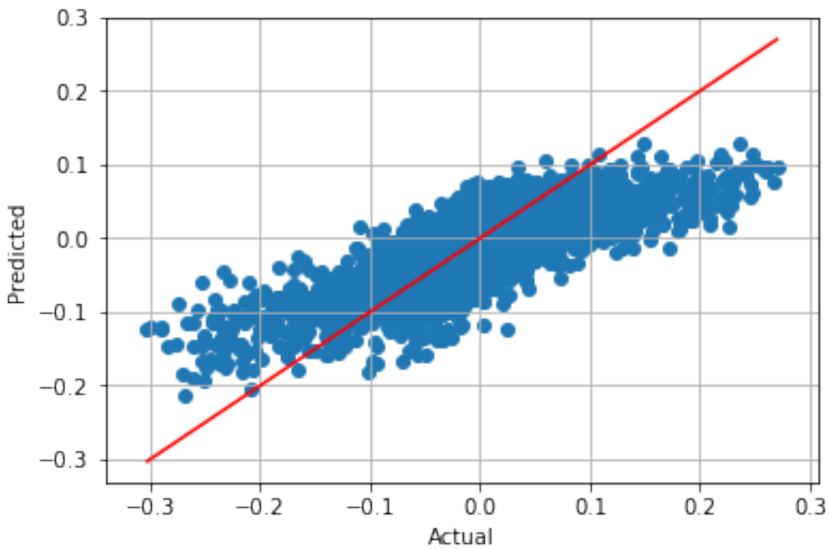
Fit a model using only the `d` neurons selected in the previous step and print both the test RSS per sample and the normalized test RSS.

In [13]:
```
XS = X0[:,Isel]

#y_tr.shape

X_tr = XS[:7768,:]#spikes


#ytr is a product of np.array(something) then i use a np.random.choice in the same range of samples as time

y_tr = y0[:7768]


X_ts = XS[7768:15536,:]  #random test set



y_ts = y0[7768:15536]

reg = sklearn.linear_model.LinearRegression()
#X_tr.shape
#y_tr.shape
reg.fit(X_tr,y_tr)

yts_pred = reg.predict(X_ts)

RSS_re_ts = np.mean((y_ts-yts_pred)**2)/(np.std(y_ts)**2)
print("Normalized test RSS = {0:f}".format(RSS_re_ts))
```

```
Normalized test RSS = 0.496102
```

Create a scatter plot of the predicted vs. actual hand motion on the test data. On the same plot, plot the line where `yts_hat = yts`.

In [14]:
```
ymin = np.min(y_ts)
ymax = np.max(y_ts)
plt.scatter(y_ts,yts_pred)
plt.plot([ymin,ymax],[ymin,ymax],'r')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.grid()
```



Using K-fold cross validation for the optimal number of neurons

In the above, we fixed `d=100`. We can use cross validation to try to determine the best number of neurons to use. Try model orders with `d=10,20,...,190`. For each value of `d`, use K-fold validation with 10 folds to estimate the test RSS. For a data set this size, each fold will take a few seconds to compute, so it may be useful to print the progress.

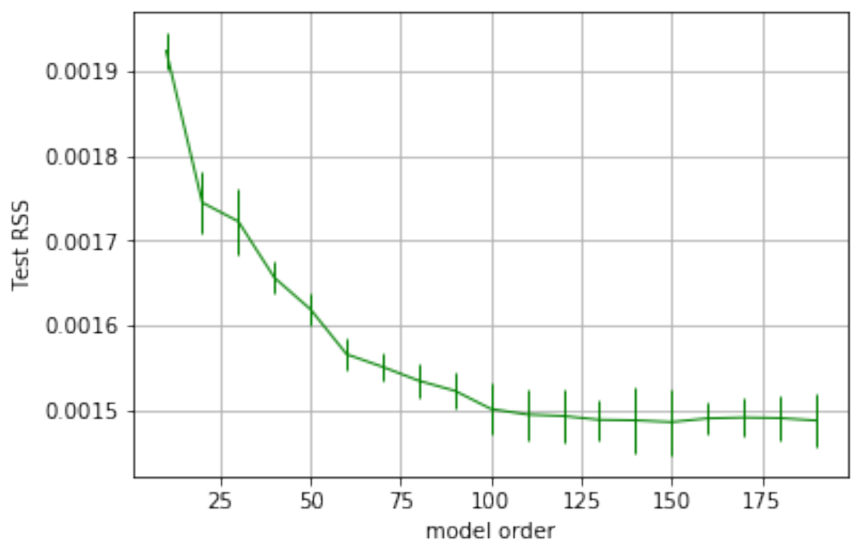
In [15]:
```
# Create a k-fold cross validation object
nfold = 10
kf = sklearn.model_selection.KFold(n_splits=nfold,shuffle=True)



# Model orders to be tested
dtest = np.arange(10,200,10)
nd = len(dtest)

#
RSSts = np.zeros((nd,nfold))
print(RSSts.shape)

for i, d in enumerate(dtest):
    Isel_d = Indecies[:d]
################ okay then we need to go from High RSS to low RSS(more
accurate)#############
    X_d = X0[:,Isel_d]
   # print(X_d.shape)
    for ifold, ind in enumerate(kf.split(X_d)):


        # Get the training data in the split
        Itr,Its = ind
        X_tr_k = X_d[Itr,:]
        y_tr_k = y0[Itr]
        X_ts_k = X_d[Its,:]
        y_ts_k = y0[Its]

        #linear fit
        regd = sklearn.linear_model.LinearRegression()
        regd.fit(X_tr_k,y_tr_k)

        # Compute the prediction error on the test data
        y_ts_pred_k = regd.predict(X_ts_k)
        RSSts[i,ifold] = np.mean((y_ts_k-y_ts_pred_k)**2)
        #print(RSSts[i,ifold])
        #print(a)
print(RSSts)
```

```
(19, 10)
[[ 0.00200377  0.00187628  0.00190083  0.00185802  0.00183574  0.002
016
   0.00188236  0.00190841  0.0019501   0.00200372]
 [ 0.00181851  0.00150285  0.00180729  0.00174197  0.00177415  0.001
65565
   0.00194295  0.00174671  0.00174211  0.00171339]
 [ 0.00184177  0.00156146  0.00174427  0.00166121  0.00166505  0.001
```

```
81823
   0.00152341  0.00174115  0.00174781  0.00191888]
 [ 0.00170323  0.00167819  0.00160337  0.00152952  0.00164699  0.001
61978
   0.00170973  0.0017102   0.00169622  0.00166184]
 [ 0.00160432  0.00153974  0.00159406  0.00166341  0.00161284  0.001
74407
   0.00166364  0.00156219  0.00159218  0.00160957]
 [ 0.0015602   0.00151301  0.00162653  0.00165851  0.00161355  0.001
53451
   0.00147122  0.00155615  0.00159796  0.00152553]
 [ 0.00161859  0.00160553  0.00158628  0.00153694  0.0015526   0.001
47522
   0.001599    0.00149749  0.00152223  0.00151405]
 [ 0.00143568  0.001599    0.0016082   0.00159808  0.00151819  0.001
56634
   0.00154199  0.00147545  0.00144041  0.0015614 ]
 [ 0.00152143  0.00156836  0.0015163   0.00158606  0.00149052  0.001
57234
   0.00137592  0.00145804  0.00154095  0.00159581]
 [ 0.00157843  0.00152698  0.00135927  0.00140451  0.00161589  0.001
55928
   0.00141174  0.00161509  0.00141015  0.00153024]
 [ 0.00152043  0.00154753  0.00165524  0.0014226   0.00133273  0.001
5101
   0.00150537  0.00151892  0.00156187  0.00137167]
 [ 0.00163753  0.00151859  0.00143113  0.00143511  0.00160018  0.001
59079
   0.00136518  0.00140504  0.00155646  0.00138889]
 [ 0.00141685  0.00156237  0.00138891  0.00157003  0.00159697  0.001
51728
   0.00144979  0.00148237  0.00149834  0.00140135]
 [ 0.00152335  0.00159732  0.00133782  0.0015604   0.00161291  0.001
48737
   0.00141535  0.00127091  0.00163928  0.00143291]
 [ 0.00151061  0.00171378  0.00144116  0.00141336  0.00126501  0.001
46199
   0.00163524  0.00140467  0.00151084  0.00150131]
 [ 0.0014926   0.00150025  0.00138791  0.00148594  0.00143973  0.001
53854
   0.00158165  0.00148712  0.00155536  0.00143184]
 [ 0.00140293  0.00145939  0.00150656  0.00139797  0.0015645   0.001
55356
   0.00148464  0.00162211  0.00148678  0.00143259]
 [ 0.0015004   0.00165096  0.00144004  0.00143647  0.00147131  0.001
38487
   0.00151449  0.00156444  0.00140155  0.00153966]
 [ 0.00136409  0.0014156   0.00150389  0.00156824  0.00157393  0.001
47918
   0.00144906  0.00166261  0.00134329  0.00151628]]
```

Compute the RSS test mean and standard error and plot them as a function of the model order `d` using the `plt.errorbar()` method.

In [16]:
```
# Compute the mean and standard deviation over the different folds.
RSS_mean = np.mean(RSSts,axis=1)
RSS_std = np.std(RSSts,axis=1) / np.sqrt(nfold-1)

# Plot the mean test RSS and test RSS standard error
plt.errorbar(dtest,RSS_mean,fmt='g-',yerr=RSS_std,linewidth=1)
# the x axis should be log of alphas
plt.grid()
plt.xlabel('model order')
plt.ylabel('Test RSS')
plt.show()
```



Find the optimal order using the one standard error rule. Print the optimal value of `d` and the mean test RSS per sample at the optimal `d`.

In [17]:
```
# Find the minimum RSS target
imin = np.argmin(RSS_mean)
RSS_tgt = RSS_mean[imin] + RSS_std[imin]

# Find the lowest model order below the target
I = np.where(RSS_mean <= RSS_tgt)[0]
print(I)
iopt = I[0]
dopt = dtest[iopt]

print("the optimal value of d is",dopt)
print(RSS_mean[I[0]])
```

```
[ 8  9 10 11 12 13 14 15 16 17 18]
the optimal value of d is 90
0.00152257348773
```

Using LASSO regression

Instead of using the above heuristic to select the variables, we can use LASSO regression.

First use the `preprocessing.scale` method to standardize the data matrix `X0`. Store the standardized values in `Xs`. You do not need to standardize the response. For this data, the `scale` routine may throw a warning that you are converting data types. That is fine.

In [18]:
```
from sklearn import preprocessing

Xs = sklearn.preprocessing.scale(X0)
#y = sklearn.preprocessing.scale(y0)
```

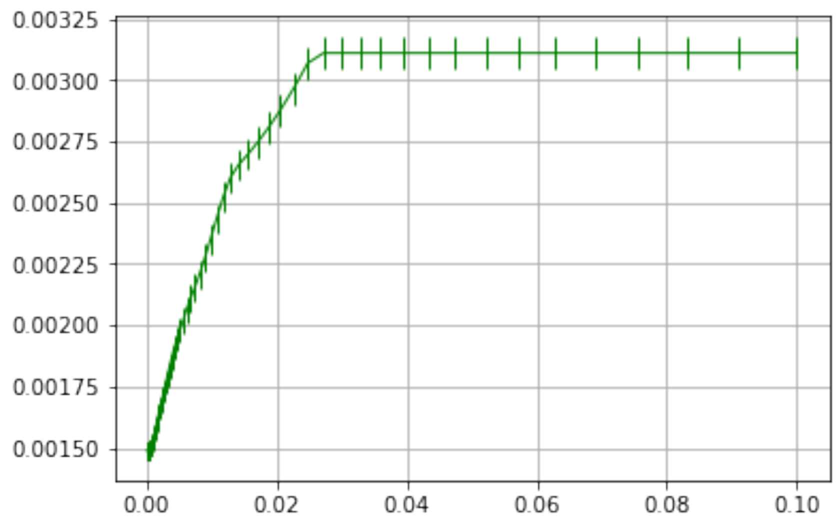
```
/anaconda/lib/python3.6/site-packages/sklearn/utils/validation.py:42
9: DataConversionWarning: Data with input dtype uint8 was converted
to float64 by the scale function.
  warnings.warn(msg, _DataConversionWarning)
```

Now, use the LASSO method to fit a model. Use cross validation to select the regularization level `alpha`. Use `alpha` values logarithmically spaced from `1e-5` to `0.1`, and use 10 fold cross validation.

In [19]:
```
# Create a k-fold cross validation object
nfold = 10
kf = sklearn.model_selection.KFold(n_splits=nfold,shuffle=True)

# Create the LASSO model.  We use the `warm start` parameter so that t
he fit will start at the previous value.
# This speeds up the fitting.
model = sklearn.linear_model.Lasso(warm_start=True)

# Regularization values to test
nalpha = 100
alphas = np.logspace(-5,-1,nalpha)

# MSE for each alpha and fold value
mse = np.zeros((nalpha,nfold))
for ifold, ind in enumerate(kf.split(Xs)):


    # Get the training data in the split
    Itr,Its = ind
    X_tr_l = Xs[Itr,:]
    y_tr_l = y0[Itr]
    X_ts_l = Xs[Its,:]
    y_ts_l = y0[Its]

    # Compute the lasso path for the split
    for ia, a in enumerate(alphas):

        # Fit the model on the training data
        model.alpha = a
        model.fit(X_tr_l,y_tr_l)

        # Compute the prediction error on the test data
        y_ts_pred_l = model.predict(X_ts_l)
        mse[ia,ifold] = np.mean((y_ts_pred_l-y_ts_l)**2)
```

Plot the mean test RSS and test RSS standard error with the `plt.errorbar` plot.

In [20]:
```
# Compute the mean and standard deviation over the different folds.
mse_mean = np.mean(mse,axis=1)
mse_std = np.std(mse,axis=1) / np.sqrt(nfold-1)

# Plot the mean test RSS and test RSS standard error
plt.errorbar(alphas,mse_mean,fmt='g-',yerr=mse_std,linewidth=1)
######################## the x axis should be log of alphas##########
###
plt.grid()
plt.show()
```



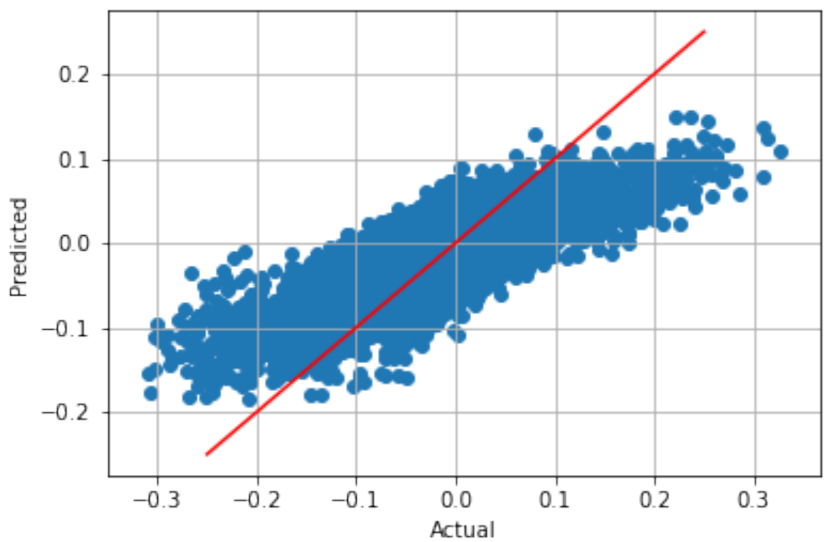
Find the optimal `alpha` and mean test RSS using the one standard error rule.

In [21]:
```
#We find the optimal alpha, by the following steps:
#Find the alpha with the minimum test MSE
#Set mse_tgt = minimum MSE + 1 std dev MSE
#Find the least complex model (highest alpha) such that MSE < mse_tgt

# Find the minimum MSE and MSE target
imin = np.argmin(mse_mean)
mse_tgt = mse_mean[imin] + mse_std[imin]
alpha_min = alphas[imin]

# Find the least complex model with mse_mean < mse_tgt
I = np.where(mse_mean < mse_tgt)[0]
#print(I)
iopt = I[-1]
alpha_opt = alphas[iopt]
print("Optimal alpha = %f" % alpha_opt)
```

```
Optimal alpha = 0.000722
```

Using the optimal alpha, recompute the predicted response variable on the whole data. Plot the predicted vs. actual values.

In [22]:
```
model.alpha = alpha_opt
model.fit(Xs,y0)


yts_pred_a = model.predict(Xs)


plt.scatter(y0,yts_pred_a)
plt.plot([-0.25,0.25],[-0.25,0.25],'r-')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.grid()
```



More Fun

You can play around with this and many other neural data sets. Two things that one can do to further improve the quality of fit are:

- Use more time lags in the data. Instead of predicting the hand motion from the spikes in the previous time, use the spikes in the last few delays.
- Add a nonlinearity. You should see that the predicted hand motion differs from the actual for high values of the actual. You can improve the fit by adding a nonlinearity on the output. A polynomial fit would work well here.

You do not need to do these, but you can try them if you like.

In [ ]:

```
```