In [1]:
```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import numpy.polynomial.polynomial as poly
%matplotlib inline
```

In [2]:
```
import scipy.io
mat_dict = scipy.io.loadmat('StevensonV2.mat')
mat_dict.keys()
```

Out[2]:
```
dict_keys(['__header__', '__version__', '__globals__', 'Publication'
, 'timeBase', 'spikes', 'time', 'handVel', 'handPos', 'target', 'sta
rtBins', 'targets', 'startBinned'])
```

In [3]:
```
X0 = mat_dict['spikes'].T
y0 = mat_dict['handVel'][0,:]
```

In [4]:
```
nt = X0.shape[0]
nneuron = X0.shape[1]
print("nt = {0:d}, nneuron = {1:d}".format(nt, nneuron))
```

```
nt = 15536, nneuron = 196
```

In [5]:
```
time = mat_dict['time'][0, :]
tsamp = time[1] - time[0]
ttotal = time[nt-1] - time[0]
print("tsamp = {0:f}, ttotal = {1:f}".format(tsamp, ttotal))
print(time)
```

```
tsamp = 0.050000, ttotal = 776.750000
[  12.591   12.641   12.691 ...,  789.241  789.291  789.341]
```

In [6]:
```
dat = range(nt)
tr_dat = dat[0: nt: 2]
ts_dat = dat[1: nt: 2]
Xtr = X0[tr_dat]
ytr = y0[tr_dat]
Xts = X0[ts_dat]
yts = y0[ts_dat]
```

In [7]:
```
from sklearn import linear_model
regr = linear_model.LinearRegression()
regr.fit(Xtr, ytr)
y_ts_pred = regr.predict(Xts)
y_tr_pred = regr.predict(Xtr)
RSS_ts = np.mean((y_ts_pred-yts)**2)/(np.std(yts)**2)
print("normalized RSS on the test data is = {0:f}".format(RSS_ts))
```

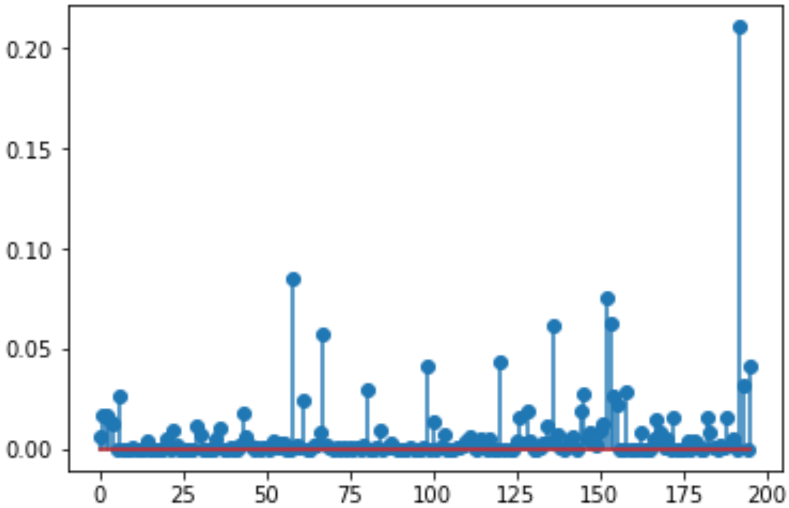
```
normalized RSS on the test data is = 1323769529480601665536.000000
```

In [8]:

```
ym = np.mean(y0)
syy = np.mean((y0-ym)**2)
Req = np.zeros(nneuron)
for k in range(nneuron):
    Xm = np.mean(X0[:, k])
    sxy = np.mean((X0[:,k]-Xm)*(y0-ym))
    sxx = np.mean((X0[:,k]-Xm)**2)
    Req[k] = (sxy)**2/sxx/syy

plt.stem(range(nneuron), Req)
```

```
/Users/jennifer/anaconda/lib/python3.6/site-packages/ipykernel_launc
her.py:8: RuntimeWarning: invalid value encountered in double_scalar
s
```

Out[8]: <Container object of 3 artists>



In [9]:

```
d = 100

Isel = np.argsort(-Req)[:d]
Sel = np.argsort(-Req)
print("The neurons with the ten highest R^2 values = {0}".format(I
```

```
The neurons with the ten highest R^2 values = [192  58 152 153 136
67 120 195  98 193]
```

In [10]:
```
Xtr2 = Xtr[:,Isel]
Xts2 = Xts[:,Isel]
print(Xtr2)
Xtr2.shape
```

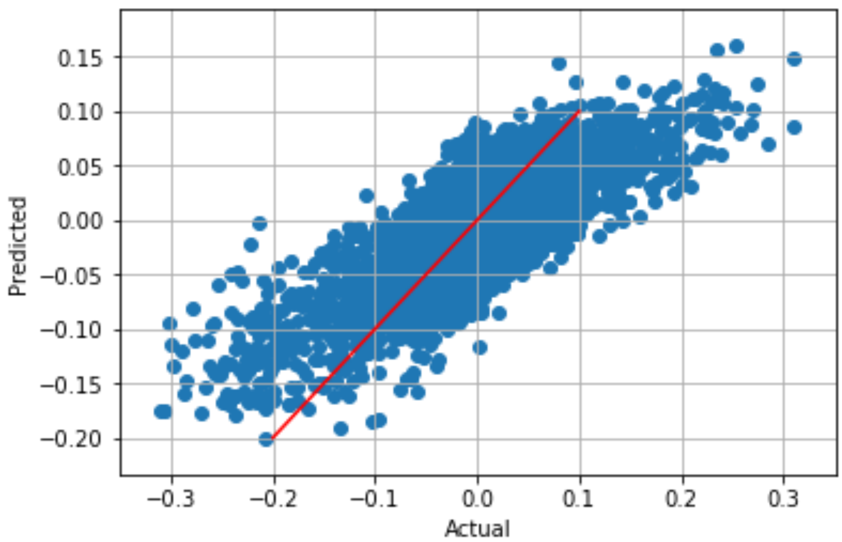
```
[[ 0  1  1 ...,  1  0  1]
 [ 0  2  0 ..., 10  0  0]
 [ 0  1  2 ...,  4  1  1]
 ...,
 [ 0  2  2 ...,  2  1  0]
 [ 0  1  2 ...,  3  0  0]
 [ 0  3  0 ...,  3  0  0]]
```

Out[10]: (7768, 100)

In [11]:
```
regr2 = linear_model.LinearRegression()
regr2.fit(Xtr2, ytr)
y_ts_pred2 = regr2.predict(Xts2)
RSS_ts2 = np.mean((y_ts_pred2-yts)**2)
RSS_ts2_nor = np.mean((y_ts_pred2-yts)**2)/(np.std(yts)**2)
print("test RSS per sample is : {0:f}".format(RSS_ts2))
print("normalized RSS on the test data is : {0:f}".format(RSS_ts2_i
```

```
test RSS per sample is : 0.001494
normalized RSS on the test data is : 0.483749
```

In [12]:
```
plt.scatter(yts, y_ts_pred2)
plt.plot([-0.2,0.1],[-0.2,0.1], 'r')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.grid()
```



In [13]:
```
yts.shape
```

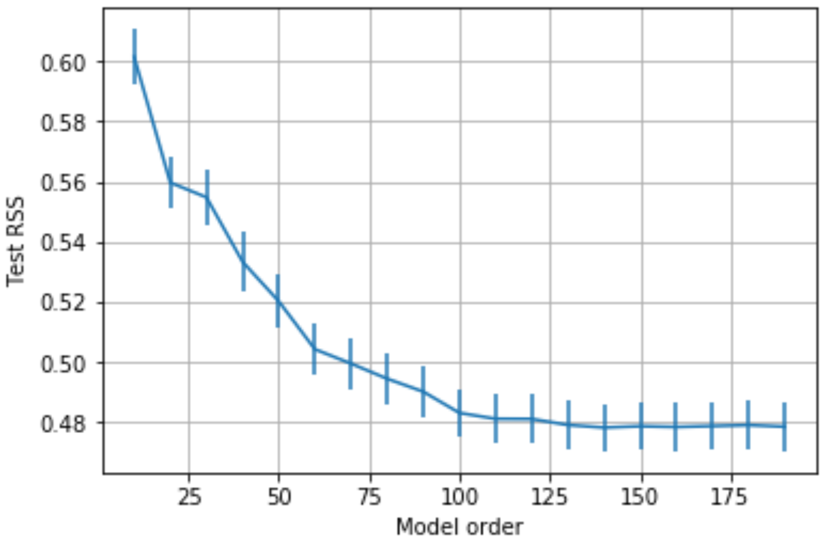
Out[13]: (7768,)

In [14]:

```
import sklearn.model_selection

nfold = 10
kf = sklearn.model_selection.KFold(n_splits=nfold, shuffle=True)

dtest = np.arange(10, 200, 10)
nd = len(dtest)

RSSts = np.zeros((nd, nfold))
for isplit, Ind in enumerate(kf.split(X0)):
    Itr, Its = Ind
    xtr_d = X0[Itr]
    ytr_d = y0[Itr]
    xts_d = X0[Its]
    yts_d = y0[Its]

    for it, d in enumerate(dtest):
        Isel_tr = xtr_d[:,Sel[:d]]
        Isel_ts = xts_d[:,Sel[:d]]
        regr_d = linear_model.LinearRegression()
        regr_d.fit(Isel_tr, ytr_d)
        y_ts_hat = regr_d.predict(Isel_ts)
        RSSts[it, isplit] = np.mean((y_ts_hat-yts_d)**2)/(np.std(y
```

In [15]:

```
RSS_mean = np.mean(RSSts, axis=1)
RSS_std = np.std(RSSts,axis=1)/np.sqrt(nfold-1)
plt.errorbar(dtest, RSS_mean, yerr = RSS_std, fmt='-')
plt.xlabel('Model order')
plt.ylabel('Test RSS')
plt.grid()
```

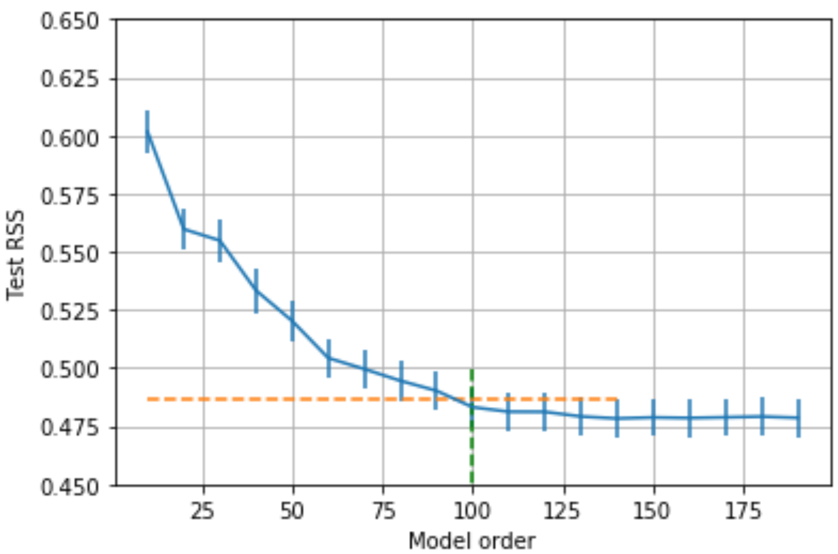


In [16]:
```
imin = np.argmin(RSS_mean)
print("Find minimize RSS, "
      "opetimal d is: {0:d} and it's RSS_mean is {1:f}"
      .format((imin+1)*nfold,RSS_mean[imin]))
```

```
Find minimize RSS, opetimal d is: 140 and it's RSS_mean is 0.478293
```

In [17]:
```
RSS_tgt = RSS_mean[imin] + RSS_std[imin]

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

plt.errorbar(dtest, RSS_mean, yerr=RSS_std, fmt='-')

# Plot the line at the RSS target
plt.plot([dtest[0],dtest[imin]], [RSS_tgt, RSS_tgt], '--')

# Plot the line at the optimal model order
plt.plot([dopt,dopt], [0,0.5], 'g--')

plt.ylim(0.45,0.65)
plt.xlabel('Model order')
plt.ylabel('Test RSS')
plt.grid()

# Print results
print("By using one standard deviation rule, ")
print("The estimated model order is %d" % dopt)
print("The opetimal RSS_mean is %f" % RSS_mean[iopt])
```

```
By using one standard deviation rule, 
The estimated model order is 100
The opetimal RSS_mean is 0.483159
```



In [18]:

```
import sklearn.preprocessing
import sklearn.model_selection
Xs = sklearn.preprocessing.scale(X0)
```

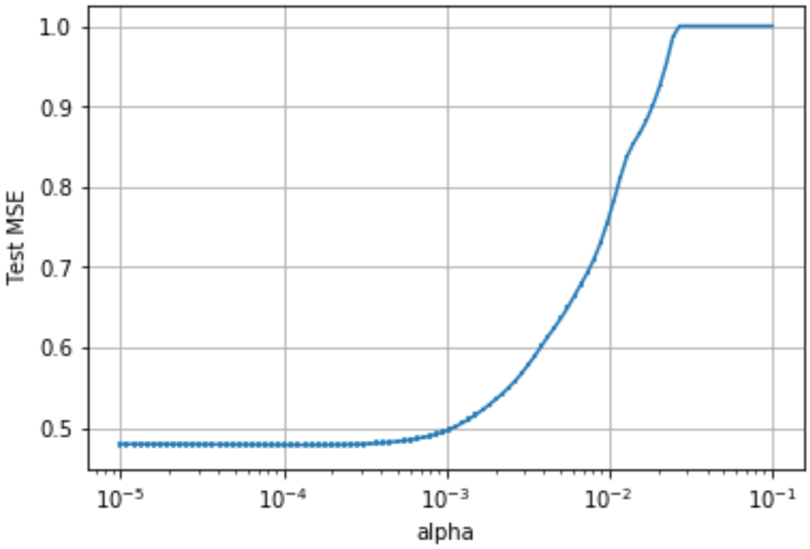
```
/Users/jennifer/anaconda/lib/python3.6/site-packages/sklearn/utils/v
alidation.py:429: DataConversionWarning: Data with input dtype uint8
was converted to float64 by the scale function.
  warnings.warn(msg, _DataConversionWarning)
```

In [19]:

```
nfold = 10
kf = sklearn.model_selection.KFold(n_splits=nfold,shuffle=True)

model = linear_model.Lasso(warm_start=True)

nalpha = 100
alphas = np.logspace(-5, -1, nalpha)

mse = np.zeros((nalpha,nfold))
RSS_LA = 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 = Xs[Itr,:]
    y_tr = y0[Itr]
    X_ts = Xs[Its,:]
    y_ts = 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,y_tr)

        # Compute the prediction error on the test data
        y_ts_pred = model.predict(X_ts)
        mse[ia,ifold] = np.mean((y_ts_pred-y_ts)**2)
        RSS_LA[ia,ifold] = np.mean((y_ts_pred-y_ts)**2)/(np.std(y_
```

In [22]:

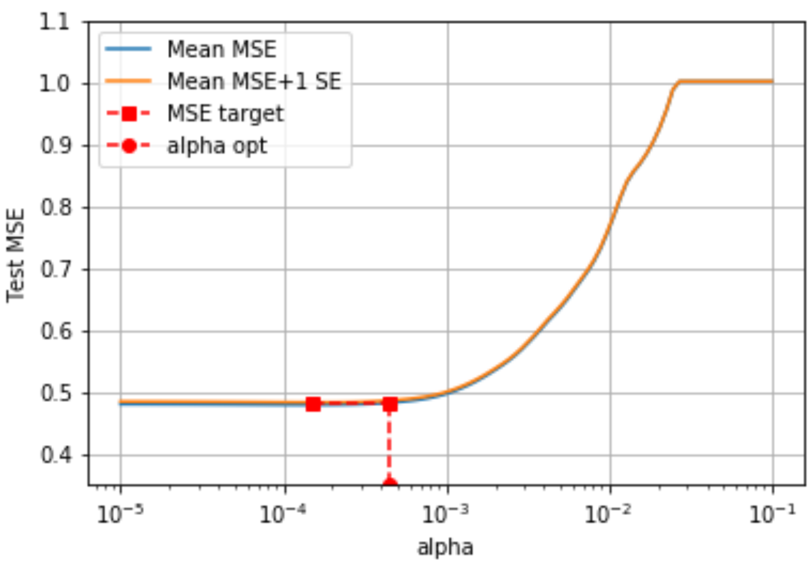
```
mse_mean = np.mean(mse, axis=1)
mse_std = np.std(mse,axis=1)/np.sqrt(nfold-1)
RSS_LA_mean = np.mean(RSS_LA, axis=1)
RSS_LA_std = np.std(RSS_LA,axis=1)/np.sqrt(nfold-1)
plt.errorbar(alphas, RSS_LA_mean, yerr = RSS_LA_std, fmt='-')
plt.xscale('log')
plt.xlabel('alpha')
plt.ylabel('Test MSE')
plt.grid()
```



In [23]:

```
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]
iopt = I[-1]
alpha_opt = alphas[iopt]
print("Optimal alpha = %f" % alpha_opt)
print("mean test RSS using the one standard error rule: {0:f}"
      .format(mse_mean[iopt]))

# Plot the mean MSE and the mean MSE + 1 std dev
plt.semilogx(alphas, mse_mean)
plt.semilogx(alphas, mse_mean+mse_std)

# Plot the MSE target
plt.semilogx([alpha_min,alpha_opt], [mse_tgt,mse_tgt], 'rs--')

# Plot the optimal alpha line
plt.semilogx([alpha_opt,alpha_opt], [0.35,mse_mean[iopt]], 'ro--')

plt.legend(['Mean MSE', 'Mean MSE+1 SE', 'MSE target','alpha opt']
plt.xlabel('alpha')
plt.ylabel('Test MSE')
plt.ylim([0.35,1.1])
plt.grid()
plt.show()
```

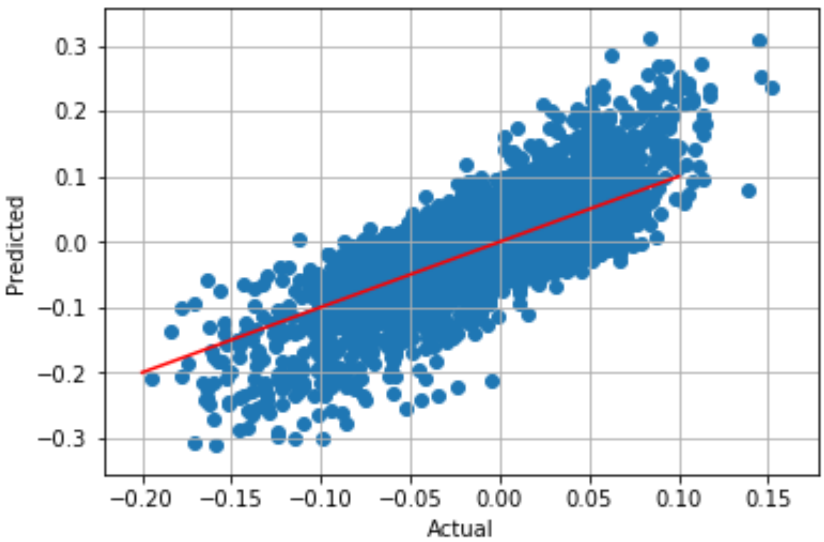
```
Optimal alpha = 0.000453
mean test RSS using the one standard error rule: 0.481872
```



In [24]:

```
Xtr_op = Xs[tr_dat]
ytr_op = y0[tr_dat]
Xts_op = Xs[ts_dat]
yts_op = y0[ts_dat]

model.alpha = alpha_opt
model.fit(Xtr_op, ytr_op)
y_op_pred = model.predict(Xts_op)
RSS_op = np.mean((y_op_pred-yts_op)**2)/(np.std(yts_op)**2)
print("Using the optimal alpha, RSS is: {0:f}".format(RSS_op))
plt.scatter(y_op_pred, yts_op)
plt.plot([-0.2,0.1],[-0.2,0.1], 'r')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.grid()
```

```
Using the optimal alpha, RSS is: 0.480135
```



In [ ]:

```

```