Introduction

In this assignment, you will convert your batch least squares solution to a recursive one! Recall that you have the following data:

Current (A)	Voltage (V)
0.2	1.23
0.3	1.38
0.4	2.06
0.5	2.47
0.6	3.17

This time, you will be fitting a linear model which includes an offset, $y = Rx + b$. If Ohm's law ($V = RI$) holds, we expect this offset to be near zero.

You will set the initial parameters with the assumption that your prior estimate of the resistance $R = 4$ is not very good. Also, since you are fairly certain that Ohm's law ($V = RI$) does, in fact, hold, it is safe to assume with high confidence that the offset term b will be close to zero. Initialize the estimator as follows:

$$\hat{R} \sim \mathcal{N}(4, 10.0), \hat{b} \sim \mathcal{N}(0, 0.2)$$

You can assume that we know the current perfectly, and that the voltage measurements are corrupted by additive, independent and identitically distributed Gaussian noise with variance $0.0225\ V^2$.

Getting Started

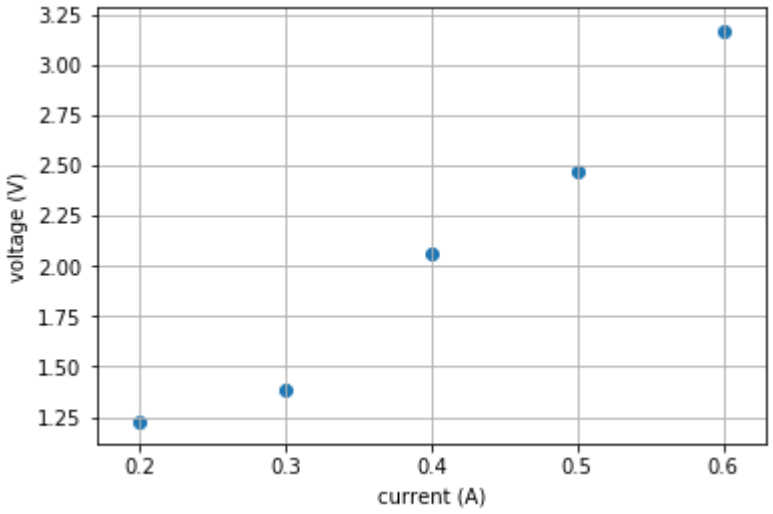
As before, load the current and voltge measurements into numpy arrays and plot the values:

In [1]:
```
import numpy as np
from numpy.linalg import inv
import matplotlib.pyplot as plt

I = np.array([0.2, 0.3, 0.4, 0.5, 0.6])
V = np.array([1.23, 1.38, 2.06, 2.47, 3.17])
```

In [2]:
```
plt.scatter(I, V)

plt.xlabel('current (A)')
plt.ylabel('voltage (V)')
plt.grid(True)
plt.show()
```



Estimating the Parameters

Batch Estimator

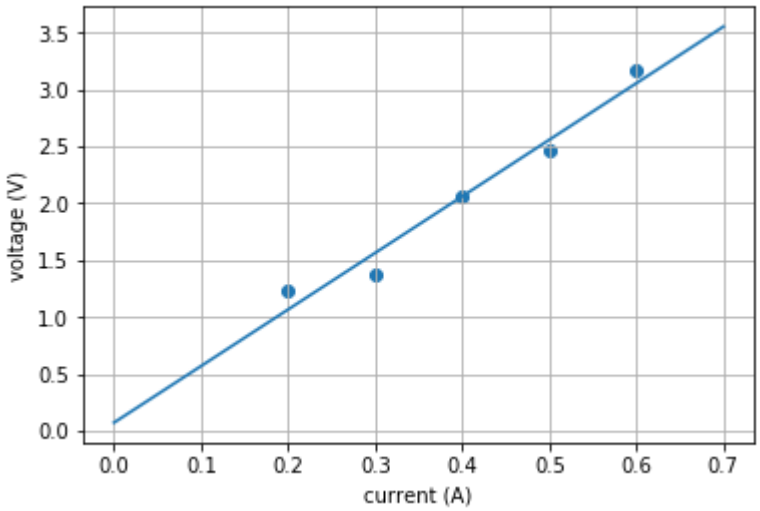
Before implementing recursive least squares, let's review the parameter estimate given by the batch least squares method used in the previous assignment. This time, you will be fitting a model which contains an offset $y = Rx + b$. We can use this result later for comparison.

In [3]:
```
## Batch Solution
H = np.ones((5,2))
H[:, 0] = I
x_ls = inv(H.T.dot(H)).dot(H.T.dot(V))
print('The parameters of the line fit are ([R, b]):')
print(x_ls)

#Plot
I_line = np.arange(0, 0.8, 0.1)
V_line = x_ls[0]*I_line + x_ls[1]

plt.scatter(I, V)
plt.plot(I_line, V_line)
plt.xlabel('current (A)')
plt.ylabel('voltage (V)')
plt.grid(True)
plt.show()
```

```
The parameters of the line fit are ([R, b]):
[4.97  0.074]
```



As expected, the offset parameter $\hat{b}$ is near zero, while $\hat{R}$ closely approximates the true resistance value of $R = 5\ \Omega$.

Recursive Estimator

Now try to implement the least squares method recursively! Recall the steps described in Module 1, Lesson 2 - "Recursive Least Squares":

Initialize the parameter and covariance estimates:

$$\hat{\mathbf{x}}_0 = E[\mathbf{x}] \quad , \quad \mathbf{P}_0 = E\left[(\mathbf{x} - \hat{\mathbf{x}}_0)(\mathbf{x} - \hat{\mathbf{x}}_0)^T\right]$$

For every measurement k:

- Calculate the correction gain

$$\mathbf{K}_k = \mathbf{P}_{k-1}\mathbf{H}_k^T\left(\mathbf{H}_k\mathbf{P}_{k-1}\mathbf{H}_k^T + \mathbf{R}_k\right)^{-1}$$

- Update the parameter estimate

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_{k-1} + \mathbf{K}_k\left(\mathbf{y}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k-1}\right)$$

- Update the covariance estimate

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\,\mathbf{P}_{k-1}$$

In [9]:
```
## Recursive solution

#Initialize the 2x2 covaraince matrix
P_k = np.diag([10.0, 0.2])

#Initialize the parameter estimate x
x_k = np.array([4.0, 0.0])




#Our measurement variance
Var = 0.0225

#Pre allocate our solutions so we can save the estimate at every step
num_meas = I.shape[0]
x_hist = np.zeros((num_meas + 1,2))
P_hist = np.zeros((num_meas + 1,2,2))

x_hist[0] = x_k
P_hist[0] = P_k

#print('x_k shape is    ' + str(x_k.shape))
#print('P_k shape is    ' + str(P_k.shape))
#print('x_hist shape is ' + str(x_hist.shape))
#print('P_hist shape is ' + str(P_hist.shape))
#print('H_k shape is ' + str(H_k.shape))


#Iterate over the measurements
for k in range(num_meas):

    H_k = np.array([I[k], 1.0])
    #Construct K_k
    temp = np.matmul(H_k, np.matmul(P_k, H_k.T))  + np.array([Var])
    #print(temp.shape)
    #print(temp)

    K_k  = np.matmul(P_k, np.matmul(H_k.reshape((2,1)), 1/temp))
    #print('K_k shape is ' + str(K_k.shape))
    #print(K_k)

    #Update our estimate
    x_k = x_k + K_k * (V[k] - np.matmul(H_k.T, x_k))

    #Update our uncertainty
    P_k = np.matmul( (np.identity(2) - np.matmul(K_k, H_k)), P_k)

    #Keep track of our history
    P_hist[k+1] = P_k
    x_hist[k+1] = x_k

print('The parameters of the line fit are ([R, b]):')
print(x_k)

print(x_hist)
```

```
The parameters of the line fit are ([R, b]):
[ 5.92977448 -0.38870826]
[[ 4.          0.        ]
 [ 5.3815261   0.13815261]
 [ 5.37064163 -0.23409631]
 [ 5.7129436  -0.22716318]
 [ 5.71351571 -0.38751217]
 [ 5.92977448 -0.38870826]]
```

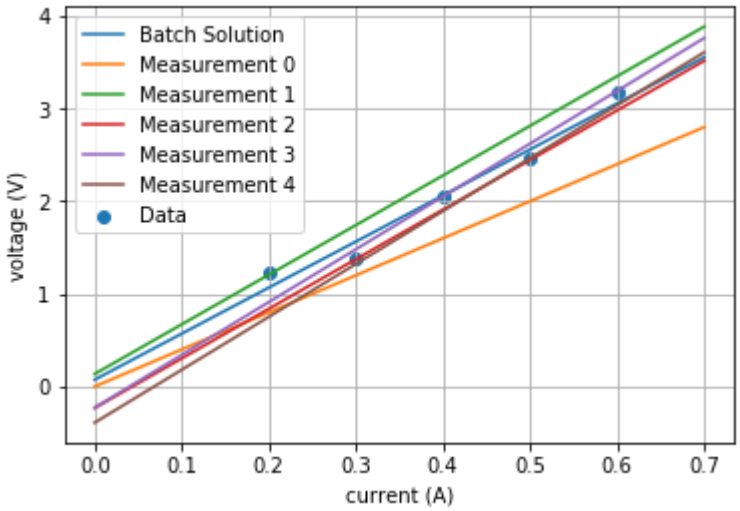
Plotting the Results

Let's plot out the solutions at every step. Does the value converge towards the batch least squares solution from the previous assignment?

In [6]:
```
#Plot
plt.scatter(I, V, label='Data')
plt.plot(I_line, V_line, label='Batch Solution')
plt.xlabel('current (A)')
plt.ylabel('voltage (V)')
plt.grid(True)

I_line = np.arange(0, 0.8, 0.1)
for k in range(num_meas):
    V_line = x_hist[k,0]*I_line + x_hist[k,1]
    plt.plot(I_line, V_line, label='Measurement {}'.format(k))

plt.legend()
plt.show()
```



The resistance estimate does approach the true resistance value of $R = 5\ \Omega$. Try modifying the initialization values (e.g., the intial uncertainty) - can you get a better final estimate?