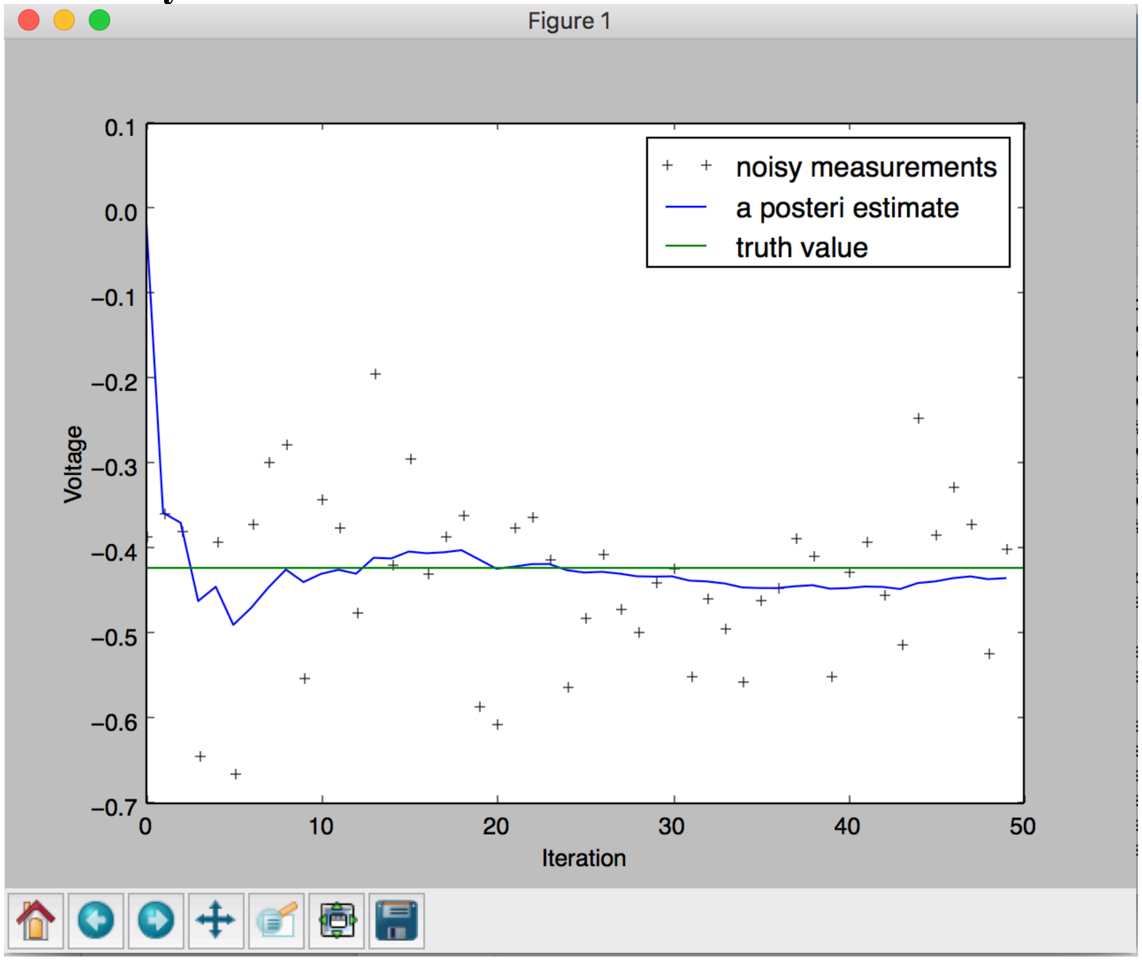
Marcus Casey
Homework 2

Summary



Using a truth value of -0.4231, I was able to implement a Kalman filter and estimate a scalar random constant, which in this case I used voltage level. Utilizing lecture resources and different implementations as well as becoming familiar with the numpy library and determining how to plot, I recognized that the Kalman required the posteriori state and posteriori error matrix for covariance. Using these two, x/p the kalman filter can then implement recursive steps to alternate and estimate the noise covariance's of Qk and Rk. I still am not 100% confident in my knowledge of Kalman filters and will continue to implement and improve my implementation and understanding to a more personal level.

Code

```
# implemented by Marcus Casey, tutorial and resources from internet to help finish this script
and implement numpy
import numpy
import pylab

# params
n_iter = 50
sz = (n_iter,) # array size
x = -0.4231 # truth value
z = numpy.random.normal(x,0.1,size=sz) # observations

Q = 1e-5 # variance

# array space
xhat=numpy.zeros(sz)      # x

P=numpy.zeros(sz)         # error

xhatminus=numpy.zeros(sz) # estimate x

Pminus=numpy.zeros(sz)    # error

K=numpy.zeros(sz)         # gain

R = 0.1**2 # measurements

# int guesses

xhat[0] = 0.0
P[0] = 1.0

for k in range(1,n_iter):

    # time update
    xhatminus[k] = xhat[k-1]
    Pminus[k] = P[k-1]+Q

    # measurement update
    K[k] = Pminus[k]/( Pminus[k]+R )
    xhat[k] = xhatminus[k]+K[k]*(z[k]-xhatminus[k])
    P[k] = (1-K[k])*Pminus[k]
```

```
pylab.figure()

pylab.plot(z,'k+',label='noisy measurements')

pylab.plot(xhat,'b-',label='a posteri estimate')

pylab.axhline(x,color='g',label='truth value')

pylab.legend()

pylab.xlabel('Iteration')

pylab.ylabel('Voltage')

pylab.figure()

valid_iter = range(1,n_iter)

pylab.plot(valid_iter,Pminus[valid_iter],label='a priori error estimate')

pylab.xlabel('Iteration')

pylab.ylabel('$(Voltage)^2$')

pylab.setp(pylab.gca(),'ylim',[0,.01])

pylab.show()
```