```
In [4]: import numpy as np
        from scipy.optimize import minimize
        H = np.matrix([1,0]) # H is a row
        data = np.loadtxt('../Documents/ApplePriceTargets.txt')
        y = np.matrix(data).T # column matrix, observations of Y
        n = y.size # number of observations
        def filtering(R, Q, A):
             1 1 1
            Args:
                R (float): scalar variance of y x
                Q (matrix 2x2): variance of x(k) | x(k-1)
                A (matrix 2x2): mean factor in x(k) | x(k-1)
            Returns:
                m_minus (matrix): 2xn dimension, the prior mean of x(k) i.e. E[x(k) \mid y(:k-1)]
                m (matrix): 2xn dimension, the posterior mean of x(k) i.e. E[x(k) \mid y(:k)]
                P_{minus} (list): list of matrix, each is the prior variance of x(k)
                P (list): list of matrix, each is the posterior variance of x(k)
            m_minus = np.matrix(np.empty((2,n))) # prior mean of x(k)
            P minus = [None] * n # prior variance of x(k)
            m = np.matrix(np.empty((2,n))) # posterior mean of x(k)
            P = [None] * n # posterior variance of x(k)
            for k in range(n):
                if k == 0:
                    # predict step to assign the prior of x(1)
                     m_{\min}(x,k) = np.matrix([y[0], 0]).T # set prior mean of x(1)
                    P_{minus[k]} = np.matrix(np.identity(2)) # set prior variance of x(1)
                else:
                     # the predict step for x(k) to calculate the prior of x(k)
                    m_{minus}[:,k] = A * m[:,k-1]
                    P_{minus[k]} = A * P[k-1] * A.T + Q
                # the update step to calculate the posterior mean of x(k)
                S = np.asscalar(H * P_minus[k] * H.T) + R # S is scalar
                K = P_{minus[k]} * H.T * (1 / S) # K is a column
                v = y[k] - np.asscalar(H * m_minus[:,k]) # v is scalar
                m[:,k] = m_{minus}[:,k] + K * v # posterior mean of x(k)
                P[k] = P_{minus}[k] - K * S * K.T # posterior variance of x(k)
            return m minus, m, P minus, P
        def mse(R, Q, A):
            m minus, m, P minus, P = filtering(R, Q, A)
            predicted y = (H * m minus).T
            return np.linalg.norm(y - predicted_y, ord=2)**2 / n
        def objective(theta):
            R = theta[0]
            Q = np.matrix([
                [theta[1], 0],
                [0, theta[2]]
            ])
            A = np.matrix([
                [theta[3], theta[4]],
                [0, theta[5]]
            ])
             Coturn man/D O Al
```

1 of 3 11/15/17, 12:14 AM

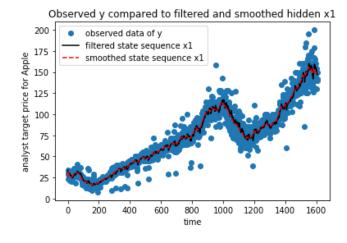
```
In [5]: np.set_printoptions(precision=5)
        bounds = [(1e-8, None)] * 6
        res = minimize(objective, np.ones(6)/100, method='L-BFGS-B', bounds=bounds)
        theta = res.x
        R = theta[0]
        Q = np.matrix([
            [theta[1], 0],
            [0, theta[2]]
        ])
        A = np.matrix([
            [theta[3], theta[4]],
             [0, theta[5]]
        ])
        print('Part A. Result of this optimization:')
        print('success: {}'.format(res.success))
        print('message: {}'.format(res.message))
        print('R: {:.4f}'.format(R))
        print('Q: ')
        print(Q)
        print('A: ')
        nrin+ /71
        Part A. Result of this optimization:
        message: b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'</pre>
        R: 0.0003
        Q:
           3.97100e-06 0.00000e+00]
        [[
            0.00000e+00 1.01627e-02]]
         [
        Α:
        [[ 1.00082 0.01032]
         [ 0.
                     0.01005]]
In [6]: # Part C
        m_minus, m, P_minus, P = filtering(R, Q, A) # m contains the filtered state sequence
        # compute the smooth sequence
        m = np.matrix(np.empty((2,n)))
        P smooth = [None] * n
        for k in range(n-1,-1,-1):
             if k == n-1:
                 m \text{ smooth}[:,k] = m[:,k]
                 P_{smooth[k]} = P[k]
                 G = P[k] * A.T * np.linalg.inv(P_minus[k-1])
                 P_{\text{smooth}[k]} = P[k] + G * (P_{\text{smooth}[k+1]} - P_{\text{minus}[k+1]}) * G.T
                 m amonths. k1 - ms. k1 + C + (m amonths. k+11 m minuas. k+11)
```

2 of 3 11/15/17, 12:14 AM

```
In [12]: # Part C plot
   import matplotlib.pyplot as plt
%matplotlib inline

   xvals = list(range(n))
   plt.figure()
   plt.plot(xvals, y, 'o', label='observed data of y')
   plt.plot(xvals, m[0,:].T, 'k-', label='filtered state sequence x1')
   plt.plot(xvals, m_smooth[0,:].T, 'r--', label='smoothed state sequence x1')
   plt.xlabel('time')
   plt.ylabel('analyst target price for Apple')
   plt.title('Observed y compared to filtered and smoothed hidden x1')
```

Out[12]: <matplotlib.legend.Legend at 0x1517e0ce10>



3 of 3