```
"""
Created on Sat Jan 23 20:45:24 2016

@author: Jessica
Math 404 HW #2
"""

import numpy as np
from matplotlib import pyplot as plt
from numpy import linalg as la

"""Define global variables"""
delT = 0.1
b = 10**(-4)

F = np.eye(4)
F[2,2] -= b
F[3,3] -= b
F[0,2], F[1,3] = delT, delT

H = np.zeros((2,4))
H[0,0], H[1,1] = 1,1

Q = np.eye(4)*0.1

R = np.eye(2)*500
Rinv = np.eye(2)*0.002

g = 9.8
u = np.zeros((4,1))
u[3,0] = -g*delT


def evolveSystem(x0, nSteps):
    """
    Evolve the system (as in problem 1) starting at a given
    x0, for nSteps number of time steps.
    """
    xk = np.vstack(x0)
    X = np.zeros((nSteps,4))

    for k in xrange(nSteps):
        X[k,:] = np.squeeze(xk)
        wk = np.random.multivariate_normal(np.zeros(4), Q)
        wk = np.vstack(wk)
        xk = F.dot(xk) + u + wk
```

```
    return X

def reverseSystem(x0):
    """For problem 5 - reverse the system to determine the origin"""
    xk = np.vstack(x0)
    Finv = la.inv(F)
    while xk[1] > 0:
        xk_1 = xk.copy()
        wk = np.random.multivariate_normal(np.zeros(4), Q)
        wk = np.vstack(wk)
        xk = Finv.dot(xk_1) - Finv.dot(u) - Finv.dot(wk)
    #Interpolate to determine the intersect with 0
    intersect = np.interp([0],[xk_1[1,0],xk[1,0]], [xk_1[0,0],xk[0,0]])
    return intersect[0]

def measure(xk):
    """Add measurement noise to the state xk"""
    vk = np.random.multivariate_normal(np.zeros(2), R)
    return xk[:2] + vk

def oneStepKalman(xk_1, Pk_1, yk):
    """One step Kalman filter given an xk-1, Pk-1, and yk"""
    xk_1 = np.vstack(xk_1)
    yk = np.vstack(yk)
    #Calculate Pk
    Pk = la.inv(Q + F.dot(Pk_1).dot(F.T))
    Pk += np.dot(H.T, Rinv).dot(H)
    Pk = la.inv(Pk)
    #Calculate xk
    temp = F.dot(xk_1) + u
    xk = temp - np.dot(Pk, H.T).dot(Rinv).dot(np.dot(H,temp) - yk)
    return xk, Pk

def predictiveKalman(xk_1, Pk_1):
    """Predictive Kalman filter given xk-1 and Pk-1 (no yk)"""
    xk_1 = np.vstack(xk_1)
    Pk = F.dot(Pk_1).dot(F.T) + Q
    xk = F.dot(xk_1) + u
    return xk, Pk
```

```
"""Problems"""
def problem1_2():
    x0 = np.vstack(np.array([0., 0., 300., 600.]))
    X = evolveSystem(x0, 1200)
    Y = np.apply_along_axis(measure, 1, X[400:600,:])
    plt.plot(X[:,0], X[:,1])
    plt.scatter(Y[:,0], Y[:,1],marker=".")
    plt.ylim([0,20000])
    plt.title("Actual position with measurements")
    plt.show()

def problem3_4_5():
    """Problem 3"""
    #Initialize P
    Pk = Q*1e6
    #Calculate the actual position and measurements
    x0 = np.array([0., 0., 300., 600.])
    X = evolveSystem(x0, 1210)
    Y = np.apply_along_axis(measure, 1, X[:600,:])
    #Plot actual position and measurements
    plt.plot(X[400:600,0], X[400:600,1], label="Actual position")
    plt.scatter(Y[400:,0], Y[400:,1],marker=".",label="Measurements")
    #Set up initial state
    xk = np.zeros(4)
    xk[:2] = Y[400,:]
    xk[2:] = np.mean(X[390:400,2:], axis=0) #take average velocity from 3
    #Buffer to hold Kalman filter measurements
    kalman = np.zeros((200,4))
    for k in xrange(400, 600):
        yk = Y[k,:]
        xk, Pk = oneStepKalman(xk, Pk, yk)
        kalman[k-400,:] = np.squeeze(xk)
    plt.scatter(kalman[:,0], kalman[:,1],marker="+",c='r',s=60,label="Kal
    plt.legend(loc=0)
    plt.title('Kalman filter between 400 and 600')
    plt.show()

    """Problem 4"""
    #Buffer to hold Kalman filter predictions
    kalmanPrediction = np.zeros((1000,4))
    j = 0
    while xk[1] > 0:
        xk, Pk = predictiveKalman(xk,Pk)
        kalmanPrediction[j,:] = np.squeeze(xk)
        j += 1
```

```
#Create plot
plt.plot(X[400:,0], X[400:,1], label="Actual position")
plt.plot(kalmanPrediction[:j,0], kalmanPrediction[:j,1],c='r',label="
plt.legend(loc=0)
plt.title("Prediction of projectile arc")
plt.ylim([0,18000])
plt.show()

"""Problem 5"""
#Use the 30th estimate of the state from our Kalman filter in Problem
origin = reverseSystem(kalman[30,:])
print "The x-coordinate of the origin is",origin
```