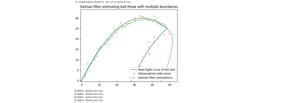
Implementing the trajectory of the ball throw with noisy and hidden observations

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
def plotset(lim_x, lim_y):
    plt.xlim(lim_x[0], lim_x[1])
    plt.ylim(lim_y[0], lim_y[1])
def kalman_throw(observations=False):
    dt = 0.1
    g = 9.81
    k_2 = 0.04
    k_1 = 0.5*rho*A
    mass = 0.145
    rho = 1.225
    sigma_w = lambda x:x*k_1*k_2
    sigma_v = lambda x:x*0.3
    F = np.array([[1, dt, 0, 0],
                  [0, 1, dt, 0],
                  [0, 0, 1, dt],
                  [0, 0, 0, 1]])
    H = np.eye(4)[[0,2],:]
    def f(x,u=0):
        return np.dot(F,x) + np.array([0,
                                       -g*dt,
                                       0,
                                       -g*dt])
    def h(x):
        return np.dot(H,x)
    def sigma_w_observations(x,u=0):
        return np.diag([sigma_w(x[1]), sigma_w(x[3])])
    def sigma_v(x):
        return np.diag([sigma_v(x[0]), sigma_v(x[2])])
    x = np.array([0, 20, 0, 20])
    P = np.diag([0.1, 0.1, 0.1, 0.1])
    Q = np.diag([0.1, 0.1, 0.1, 0.1])
    R = np.diag([0.1, 0.1])
    # We'll keep track of the state at each time step.
    # For plotting purposes only.
    x_t = []
    y_t = []
    # Simulate the robot for 50 steps.
    for i in range(50):
        # Save current state.
        x_t.append(x[0])
        y_t.append(x[2])
        # Ground truth.
        x = f(x)
        # Measurement.
        y = np.dot(sigma_v(x), np.random.normal(size=(2,)))
        y = h(x) + y
        # Update.
        K = np.dot(P, np.dot(H.T, np.linalg.inv(np.dot(H, np.dot(P, H.T)) + np.dot(sigma_v(x), sigma_v(x)))))
        x = x + np.dot(K, (y - h(x)))
        P = np.dot(np.eye(4) - np.dot(K, H), P)
        x = f(x)
        P = np.dot(F, np.dot(P, F.T)) + np.dot(sigma_w_observations(x), np.dot(Q, sigma_w_observations(x).T))
        y = np.dot(sigma_v(x), np.random.normal(size=(2,)))
        y = h(x) + y
        P = np.dot(np.eye(4) - np.dot(K, H), P)
    if observations:
        return x_t, y_t
    else:
        return x_t, y_t
x_t, y_t = kalman_throw(True)
plotset([-5, 55], [-5, 55])
plot(x_t, y_t, 'r-')
plot(y_t, x_t, 'b-')
title('Kalman filter estimating ball throw with multiple boundaries')
legend(['True trajectory', 'Estimated trajectory'])
```

Plotting the trajectory of the ball and the observations

```
# Implementation of the trajectory and observations
dt = 0.1
dt2 = dt**2 / 2
mass = 0.145
g = 9.81
k_2 = 0.04
k_1 = 0.5*rho*A
sigma_w = lambda x:x*k_1*k_2
sigma_v = lambda x:x*0.3
x = np.array([0, 20, 0, 20])
x_t = []
y_t = []
sigma_w_observations = lambda x:np.diag([sigma_w(x[1]), sigma_w(x[3])])
for i in range(50):
    x = f(x)
    y = np.dot(sigma_v(x), np.random.normal(size=(2,)))
    y = h(x) + y
    x = f(x)
    y = np.dot(sigma_v(x), np.random.normal(size=(2,)))
    y = h(x) + y
    x = f(x)
    x = f(x)
    x_t.append(x[0])
    y_t.append(x[2])
plotset([-5, 55], [-5, 55])
plot(x_t, y_t, 'r-')
plot(y_t, x_t, 'b-')
title('Kalman filter estimating ball throw with multiple boundaries')
legend(['True trajectory', 'Estimated trajectory'])
# Plot measurements.
x = np.array([0, 20, 0, 20])
x = f(x)
y = np.dot(sigma_v(x), np.random.normal(size=(2,)))
y = h(x) + y
x = f(x)
y = np.dot(sigma_v(x), np.random.normal(size=(2,)))
y = h(x) + y
x = f(x)
x = f(x)
x_t = []
y_t = []
for i in range(50):
    x = f(x)
    y = np.dot(sigma_v(x), np.random.normal(size=(2,)))
    y = h(x) + y
    x = f(x)
    y = np.dot(sigma_v(x), np.random.normal(size=(2,)))
    y = h(x) + y
    x = f(x)
    x = f(x)
    x_t.append(x[0])
    y_t.append(x[2])
plot(x_t, y_t, 'b.')
title('Kalman filter estimating ball throw with multiple boundaries')
legend(['True trajectory', 'Observations with noise'])
```




```
Kalman filter
Kalman filter
Kalman filter
```