Kalman Projectile HW

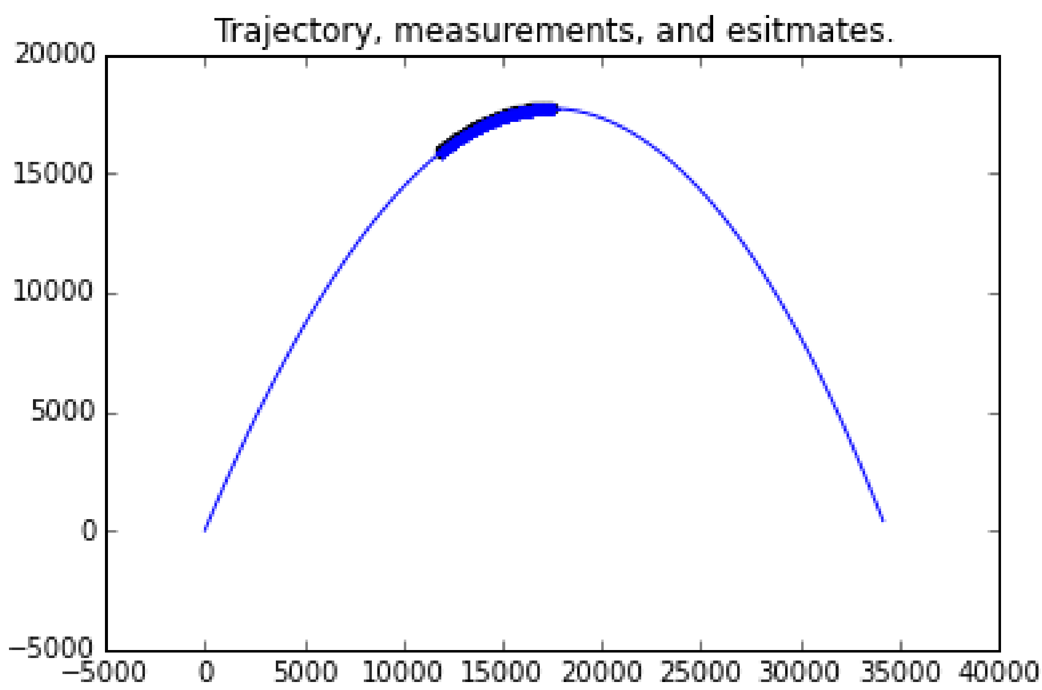
January 24, 2016

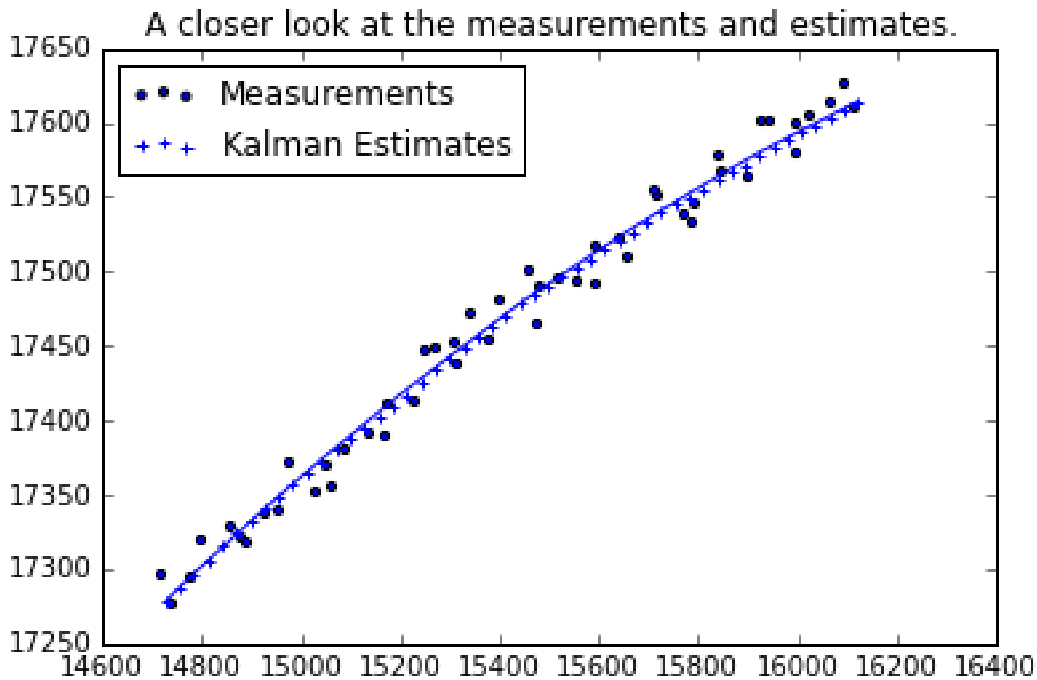
In [12]:
```
%matplotlib inline
import numpy as np
import numpy.linalg as la
import matplotlib.pyplot as plt

"""
The purpose of this program is to apply the Kalman Filter to the problem of projectile motion.
Sample projectile data with random noise is generated.
We use the Kalman filter on a subset of the data to reconstruct the entire trajectory, includi
"""

g = 9.8 #gravitational acceleration

def make_F(dt,b):
 F = np.eye(4)
 F[2,2] = 1-b
 F[3,3] = F[2,2]
 F[:2,2:] = dt*np.eye(2)
 return F

def make_u(dt):
 u = np.zeros(4)
 u[-1] = -g*dt
 return u

def make_Q():
 return 0.1*np.eye(4)

def make_R():
 return 500*np.eye(2)

def make_H():
 H = np.zeros((2,4))
 H[:,:2] = np.eye(2)
 return H


def make_noise(covariance_matrix, num_points, noise_size):
 L = la.cholesky(covariance_matrix)
 #We need to force the noise to have mean zero
 #np.random.random returns floats in the interval [0,1], and has mean 1/2
 return (2*(np.random.random((num_points,noise_size))-.5)).dot(L.T) #because right multiplicat
```

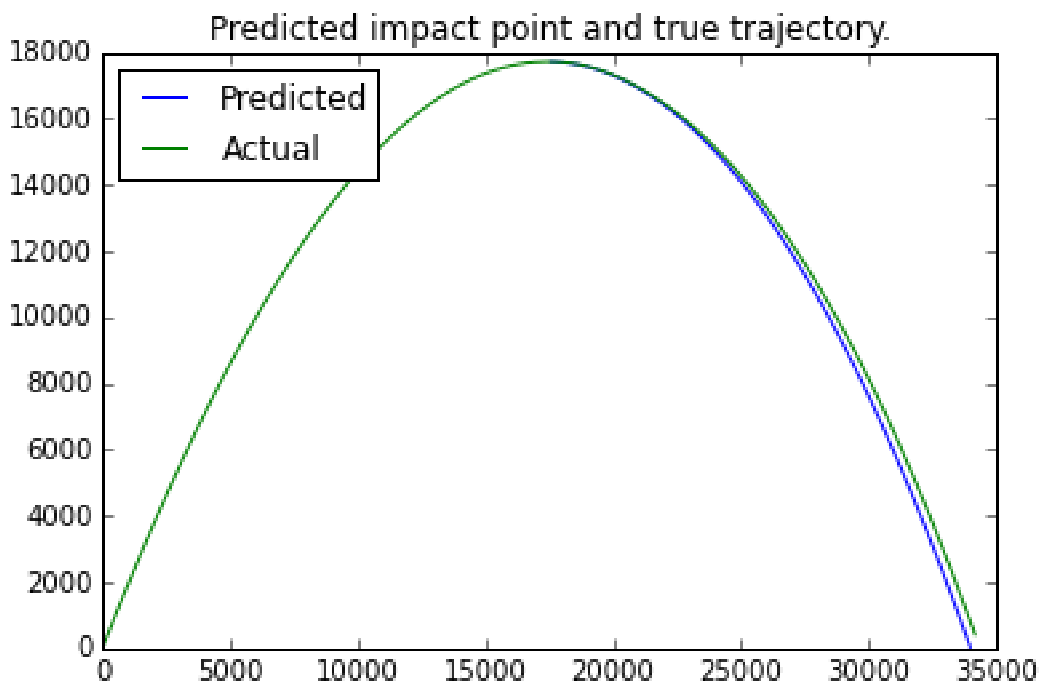
```python
def gen_projectile_data(steps, x0, dt=.1,b=1e-4):
 Q,F,u = make_Q(), make_F(dt, b), make_u(dt)
 w = make_noise(Q,steps+1,4)
 #Store the points as rows so they can be accessed in row major form
 points = np.zeros((steps+1,len(x0)))
 points[0,:] = x0
 for i in xrange(1,steps+1):
   points[i,:] = F.dot(points[i-1,:]) + u + w[i]
 return points


def constant_kalman(x0, y, u, F, G, H, Q, R, P0):
 """
 This two-step Kalman filter implementation assumes that F,H,G,P, and Q are all constant.
 x0 is the inital state
 y is the sequence of observations
 u is the sequence of inputs
 x_(k+1) = F*x_k + G*u_k + w, where w has covariance Q
 y_(k+1) = H*x_(k+1) + v, v of covariance R
 """

 N = len(y)
 if not N == len(u):
  raise ValueError("The Kalman filter needs the same number of measurements as inputs. The len

 res = np.zeros((N, x0.size))
 res[0,:] = x0
 P = np.copy(P0)
 x = np.copy(x0)
 R_inv = la.inv(R) #an optimization as R is constant in this model

 for i in xrange(1,N):
  #predictive step
  P = F.dot(P).dot(F.T) + Q
  x = F.dot(x) + G.dot(u[i])

  #update step using next measurement
  P = la.inv(la.inv(P) + H.T.dot(R_inv).dot(H))
  x = x - P.dot(H.T.dot(R_inv)).dot(H.dot(x) - y[i])

  #add state to result
  res[i,:] = x

 return res

#generate projectile data (Problem 1)
H = make_H()
delta_t,b = .1, 1e-4
data = gen_projectile_data(1200,np.array([0,0,300,600]),dt = delta_t, b= b)
#plot the true trajectory
plt.plot(data[:,0],data[:,1])


#generate noisy measurements (Problem 2)
```

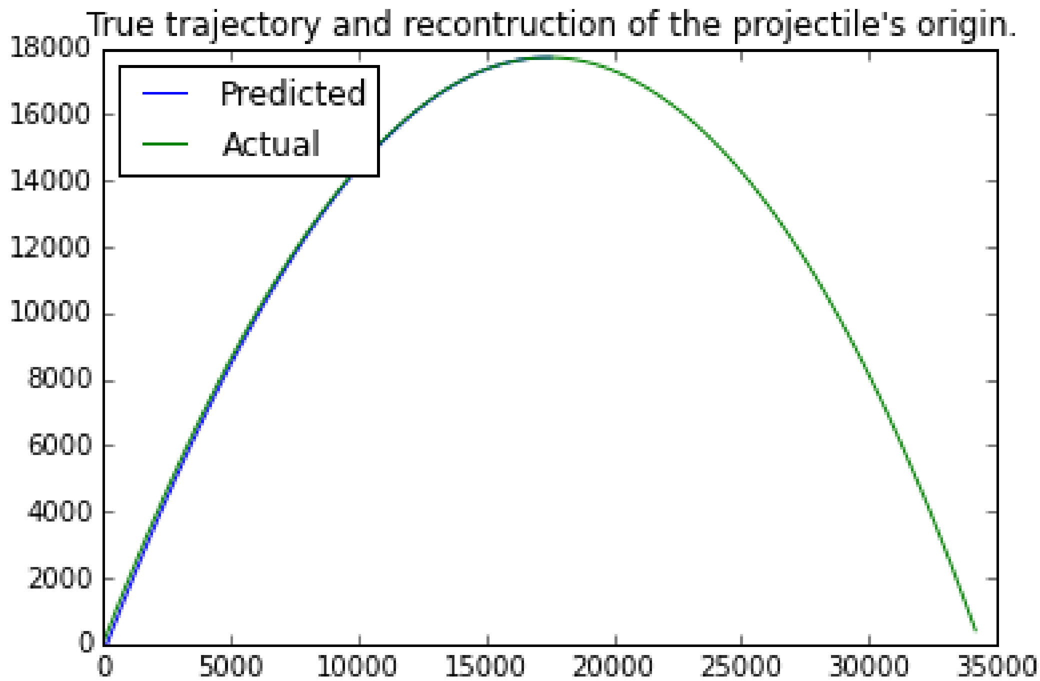
```
start, stop = 400, 600
npoints = stop-start + 1
sample = data[start:stop+1].dot(H.T)
sample = sample.T
R = 500*np.eye(2)
v = make_noise(R, npoints, 2)
measurements = sample + v.T
#plot the noisy measurements
plt.scatter(measurements[0,:], measurements[1,:], s=10)


#Apply the kalman filter to estimate the state (Problem 3)
Q = make_Q()
P0 = 10**6*Q
x0 = np.zeros(4)
x0[:2] = measurements[:2,0]
#now estimate the velocity by averaging the finite differences over 10 points
velocity_estimates = (measurements[:,1:11] - measurements[:,:10]) / delta_t
x0[2:] = np.mean(velocity_estimates, axis = 1)

u = make_u(delta_t)
u_arr = [u for i in xrange(npoints)]
F = make_F(delta_t, b)
est = constant_kalman(x0,measurements.T, u_arr, F, np.eye(4), H, make_Q(), R, P0)

#plot the Kalman estimates
plt.scatter(est[:,0], est[:,1], marker='+')

#Show the true trajectory along with the datapoints
plt.title("Trajectory, measurements, and esitmates.")
plt.show()

#Show a close up of data points
plt.scatter(measurements[0,100:150], measurements[1,100:150], s=10,label="Measurements")
plt.scatter(est[100:150,0], est[100:150,1], marker='+', label="Kalman Estimates")
plt.plot(data[start+100:stop-50,0],data[start+100:stop-50,1])
plt.title("A closer look at the measurements and estimates.")
plt.legend(loc = "upper left")
plt.show()

#Use the last predicted state to estimate future states and find the projectile's landing poin
x = est[-1]
x_coord, y_coord = [x[0]],[x[1]]
tol = 1
while True:
 x = F.dot(x) + u
 if x[1] < 0:
  break
 x_coord.append(x[0])
 y_coord.append(x[1])

#Plot the predicted trajectory to impact
plt.plot(x_coord,y_coord,label='Predicted')
#plot the true trajectory
```

```
plt.plot(data[:,0],data[:,1], label='Actual')
plt.title("Predicted impact point and true trajectory.")
plt.legend(loc='upper left')
plt.show()


#Now figure out where the projectile started (problem 5)
F_inv = la.inv(F)
x = est[-1]
x_coord, y_coord = [x[0]],[x[1]]
while True:
 x = F_inv.dot(x-u)
 if x[1] < 0:
  break
 x_coord.append(x[0])
 y_coord.append(x[1])

#Plot the predicted trajectory back to origin
plt.plot(x_coord,y_coord,label='Predicted')
#plot the true trajectory
plt.plot(data[:,0],data[:,1], label='Actual')
plt.title("True trajectory and recontruction of the projectile's origin.")
plt.legend(loc='upper left')
plt.show()
```




A closer look at the measurements and estimates.
Measurements
Kalman Estimates
17650
17600
17550
17500
17450
17400
17350
17300
17250
14600
14800
15000
15200
15400
15600
15800
16000
16200
16400


Predicted impact point and true trajectory.
Predicted
Actual
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
0
5000
10000
15000
20000
25000
30000
35000


True trajectory and recontruction of the projectile's origin.
Predicted
Actual
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
0
5000
10000
15000
20000
25000
30000
35000