Exercise 2

August 31, 2021

1 Solution

Given a 2-dimensional coordinate frame where the x-axis points horizontally and the y-axis points vertically. A cannon ball is fired with the speed v in an angle θ realtive to the x-axis. Gravity is the only force acting on the cannon ball, i.e. air resistance can be neglected.

The dynamic equation is given as:

$$\dot{x} = Fx + Lw + Gu \tag{1}$$

, where F, L and G are matrices. The vector w contains the external forces and u is process noise.

The state vector is given as:

$$x = \begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \end{bmatrix} \tag{2}$$

1.1 Dynamic equation

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{y} \\ \ddot{y} \end{bmatrix} = \left[\begin{array}{cc|cc} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{array}\right] \begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ -g \end{bmatrix} w + \begin{bmatrix} 0 & 0 \\ \sqrt{q_v} & 0 \\ 0 & 0 \\ 0 & \sqrt{q_v} \end{bmatrix} u \tag{3}$$

[1]:
```python
from sympy import Matrix, eye, symbols, sqrt, integrate

g, qv, dt = symbols('g qv dt')

# Dynamic matrix
F = Matrix([[0, 1, 0, 0],
            [0, 0, 0, 0],
            [0, 0, 0, 1],
            [0, 0, 0, 0]])

# Control vector
L = Matrix([[0],
            [0],
            [0],
            [-g]])
```

```
# Transition matrix
phi = eye(4) + F*dt

display(phi)
```

$$\begin{bmatrix} 1 & dt & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & dt \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

[2]:
```
# Control vector
Lambda = integrate(phi@L, dt)

display(Lambda)
```

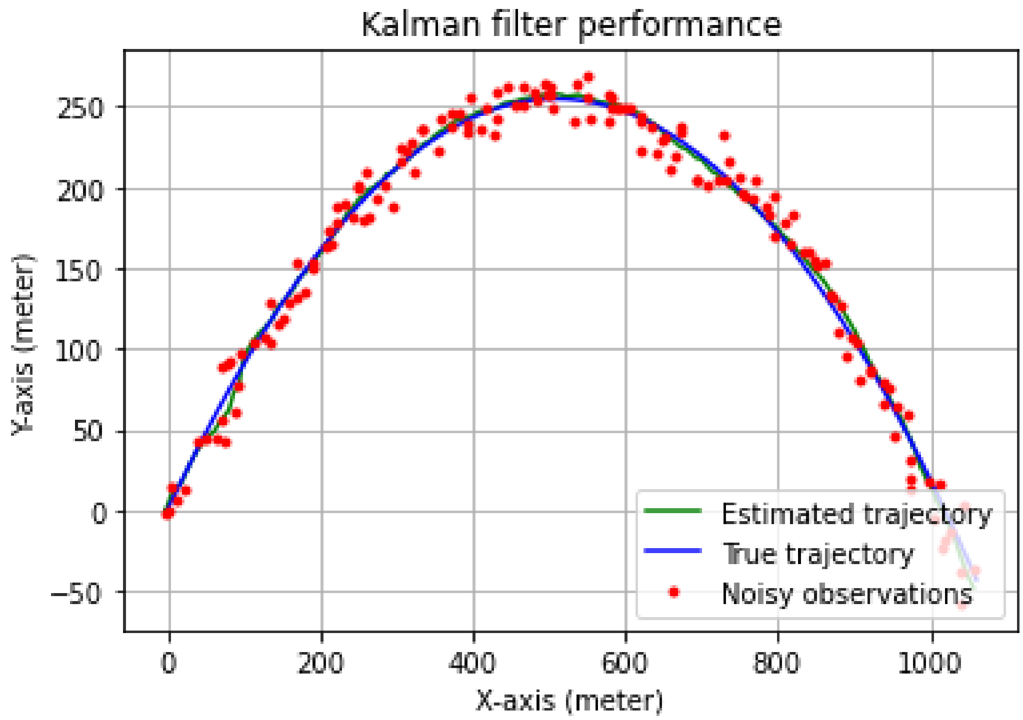
$$\begin{bmatrix} 0 \\ 0 \\ -\frac{dt^2 g}{2} \\ -dtg \end{bmatrix}$$

[3]:
```
from numpy import array, eye, kron, pi, sin, cos, arange, random
from scipy.linalg import inv, norm
import matplotlib.pyplot as plt
from vanloan import numeval

# System values
samples = 150
dt = 0.1  # second
theta = 45*pi/180  # radian
v = 100. # meter/second
g = 9.81  # meter/second^2

# Process values
qv = 1**2 # meter^2/second^3

# Measurement covariance matrix
r = 10**2
R = array([[r, 0],
           [0, r]])

# Initial state vector
x = array([[0],
           [v*cos(theta)],
           [0],
           [v*sin(theta)]])

# Initial state vector (wrong dynamic model)
```

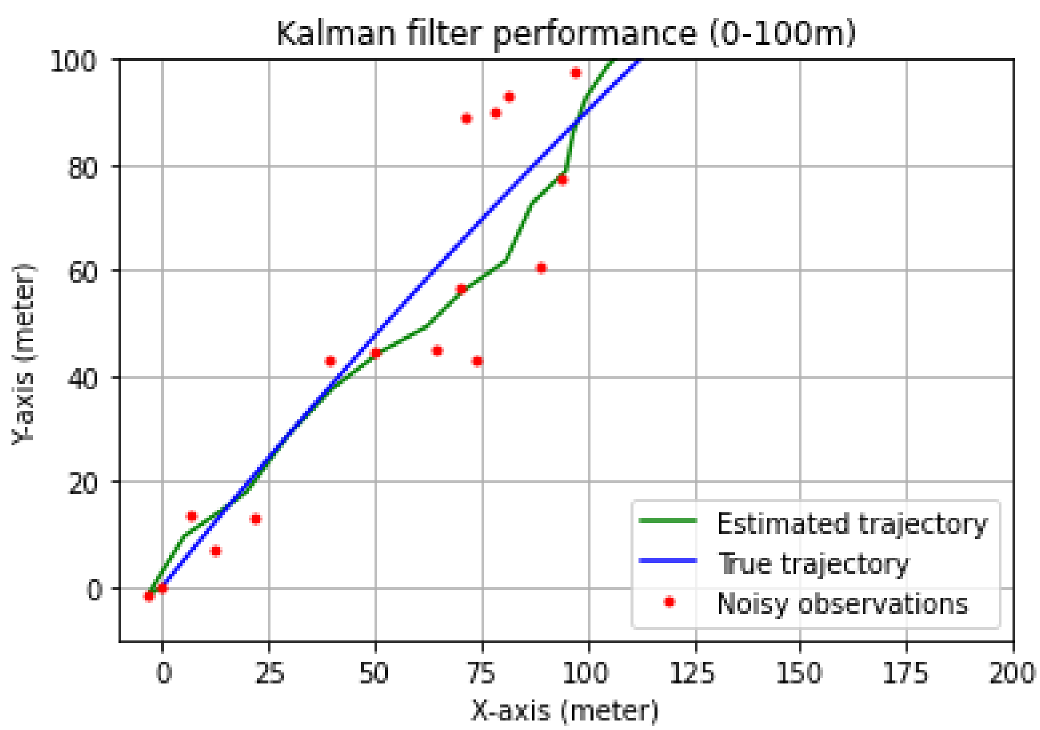
```python
x0 = x

# Initial true state vector
xt = x

# Initial state vector covariance matrix
P = kron(eye(2), [[100**2, 0],
                  [0, 10**2]])

# Design matrix
H = array([[1, 0, 0, 0],
           [0, 0, 1, 0]])

# System size
(n, e) = H.shape

# Plot vectors
state_est = [x]
state_est0 = [x0]
state_true = [xt]
obs = [array([[0],
              [0]])]
std = [0]
res = [0]
err = [0]

# Dynamic matrix
F = array([[0, 1, 0, 0],
           [0, 0, 0, 0],
           [0, 0, 0, 1],
           [0, 0, 0, 0]])

# White noise coefficients
G = array([[0, 0],
           [sqrt(qv), 0],
           [0, 0],
           [0, sqrt(qv)]])

# Control vector (Gravity)
Lambda = array([[0],
           [0],
           [-1/2*g*dt**2],
           [-g*dt]])

# Van Loan
[phi, Q] = numeval(F, G, dt)
```

```python
# KF main loop
for k in range(1, samples + 1):

    # True process
    xt = phi@xt + Lambda

    # Noisy measurements
    z = H@xt + random.normal(0, sqrt(r), (2, 1))

    # Time update
    x = phi@x + Lambda; x0 = phi@x0
    P = phi@P@phi.T + Q

    # Measurement update
    K = P@H.T@inv(H@P@H.T + R)
    x = x + K@(z - H@x); x0 = x0 + K@(z - H@x0)
    P = (eye(e) - K@H)@P

    # Accumulate plot vectors
    state_est.append(x)
    state_est0.append(x0)
    state_true.append(xt)

    obs.append(z)
    std.append(sqrt(P[0, 0] + P[2, 2]))
    res.append(norm(z - H@x))
    err.append(norm([xt[0] - x[0], xt[2] - x[2]]))

# Generate plot vectors
x_est = [x[0] for x in state_est]
y_est = [x[2] for x in state_est]

x_est0 = [x[0] for x in state_est0]
y_est0 = [x[2] for x in state_est0]

x_true = [xt[0] for xt in state_true]
y_true = [xt[2] for xt in state_true]

x_obs = [z[0] for z in obs]
y_obs = [z[1] for z in obs]

# Time
time = arange(0, samples + 1)

plt.figure(1)
plt.plot(x_est, y_est, 'g', label='Estimated trajectory')
plt.plot(x_true, y_true, 'b', label='True trajectory')
```

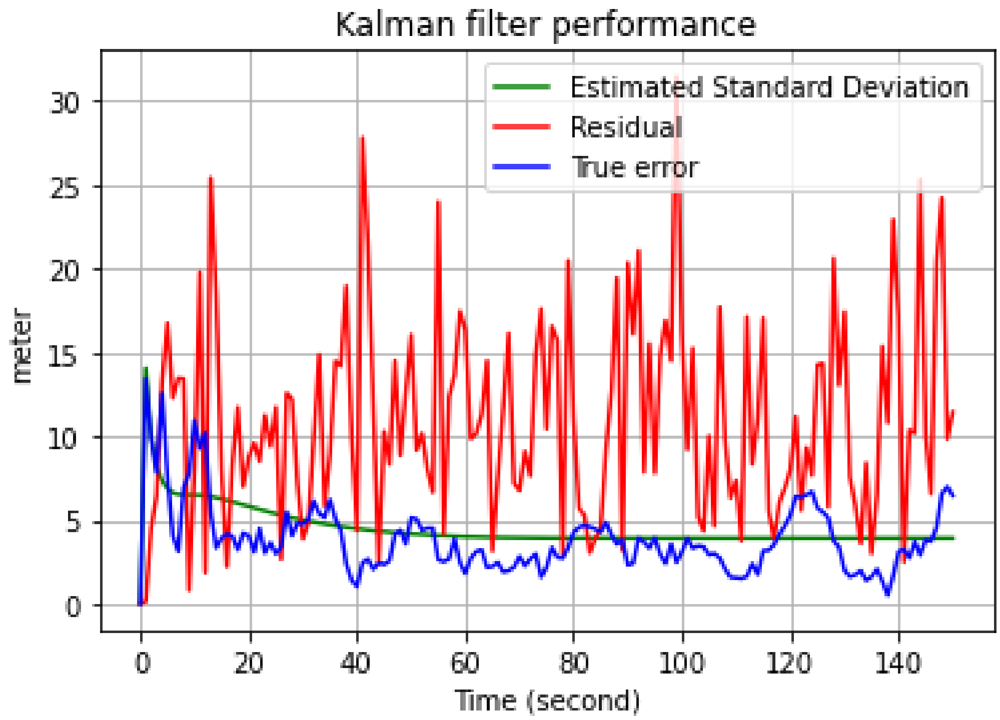
```python
plt.plot(x_obs, y_obs, 'r.', label='Noisy observations')
plt.title('Kalman filter performance')
plt.xlabel('X-axis (meter)')
plt.ylabel('Y-axis (meter)')
plt.legend(loc='lower right')
plt.grid(True, which='both')
plt.show()

plt.figure(2)
plt.plot(x_est, y_est, 'g', label='Estimated trajectory')
plt.plot(x_true, y_true, 'b', label='True trajectory')
plt.plot(x_obs, y_obs, 'r.', label='Noisy observations')
plt.title('Kalman filter performance (0-100m)')
plt.xlabel('X-axis (meter)')
plt.ylabel('Y-axis (meter)')
plt.xlim(-10, 200)
plt.ylim(-10, 100)
plt.legend(loc='lower right')
plt.grid(True, which='both')
plt.show()

plt.figure(3)
plt.plot(time, std, 'g', label='Estimated Standard Deviation')
plt.plot(time, res, 'r', label='Residual')
plt.plot(time, err, 'b', label='True error')
plt.title('Kalman filter performance')
plt.xlabel('Time (second)')
plt.ylabel('meter')
plt.legend(loc='upper right')
plt.grid(True, which='both')
plt.show()

plt.figure(4)
plt.plot(x_est0, y_est0, 'g', label='Estimated trajectory')
plt.plot(x_true, y_true, 'b', label='True trajectory')
plt.plot(x_obs, y_obs, 'r.', label='Noisy observations')
plt.title('Kalman filter performance (wrong dynamic model)')
plt.xlabel('X-axis (meter)')
plt.ylabel('Y-axis (meter)')
plt.legend(loc='lower right')
plt.grid(True, which='both')
plt.show()
```

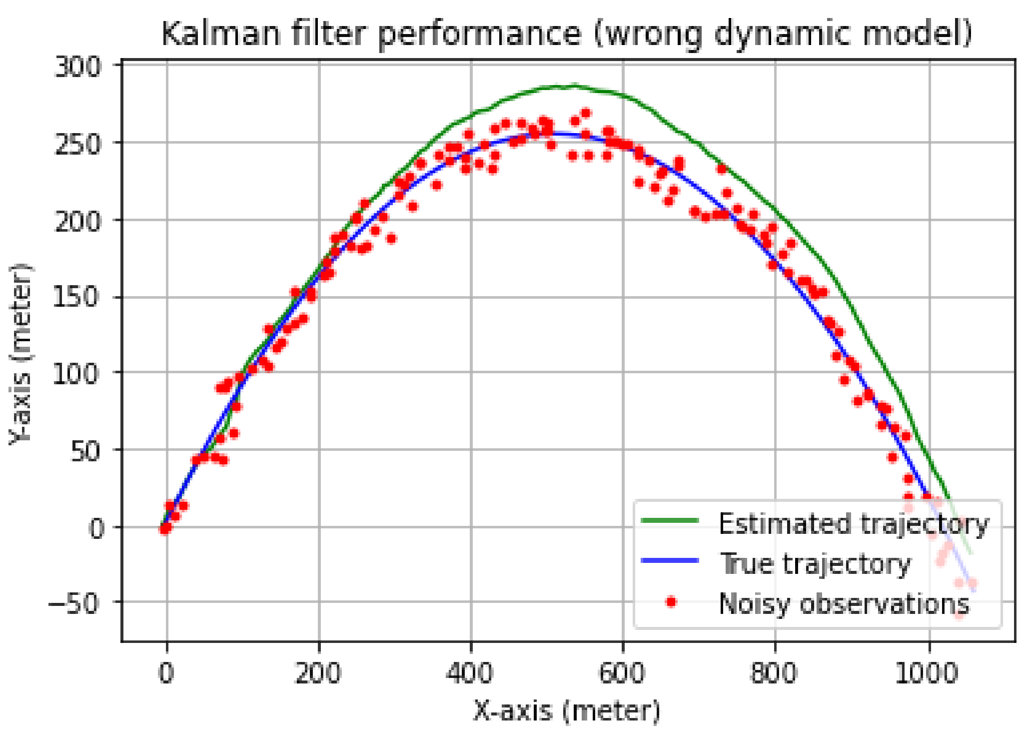


Kalman filter performance
Y-axis (meter)
X-axis (meter)
Estimated trajectory
True trajectory
Noisy observations



Kalman filter performance (0-100m)
Y-axis (meter)
X-axis (meter)
Estimated trajectory
True trajectory
Noisy observations


Kalman filter performance
Estimated Standard Deviation
Residual
True error
meter
Time (second)
0
5
10
15
20
25
30
0
20
40
60
80
100
120
140


Kalman filter performance (wrong dynamic model)
Estimated trajectory
True trajectory
Noisy observations
Y-axis (meter)
X-axis (meter)
300
250
200
150
100
50
0
−50
0
200
400
600
800
1000

1.2 Comments

Note that due to the large uncertainty represented the initial covariance of the state vector, all the trust is put on the first observation. However, as soon as the cannonball gains more velocity and the filter reach its steady state, the trust will eventually favor the dynamic model.

When we remove gravity in the dynamic model, the kalman filter will introduce a significant lag as it puts to much trust on the (wrong) dynamic model. Thus the observations (correct) are given to little weight and have a hard time trying to convince the filter to follow them. One can easily see a constant gap between the true or measured values and the estimated values in the last figure. This gap is called a systematic error / lag error.