Lab 4

Andres Coronad

Kalman Filters

```
def naive_KF(m, P_pred, s_pred, Phi, H, Q, R):
    """Naive Kalman Filter implementation.

    Inputs:
    - m (M-dim float) the new measurement
    - P_pred (NxN-dim float array) error covariance prediction
    - s_pred (N-dim float array) state prediction
    - Phi (float array) state transition matrix
    - H (MxN-dim float array) measurement matrix
    - Q (NxN-dim float array) process noise covariance
    - R (MxM-dim float) measurement noise variance

    Outputs:
    - s
    - P_pred
    - s_pred

    """

   ### complete here ###

    # Kalman gain
    K = P_pred @ (H.T * 1/np.float64((np.dot(np.dot(H,P_pred),H.T) +R)))

    #Update
    s = s_pred + ( K[:,None] * (m - H @ s_pred))
    P = (np.eye(len(s_pred)) - K[:,None] @ H[:,None].T) @ P_pred

    #prediction
    s_pred = np.dot(Phi, s)
    P_pred = (np.dot(np.dot(Phi,P),Phi.T)) + Q

    return s, P_pred, s_pred
```

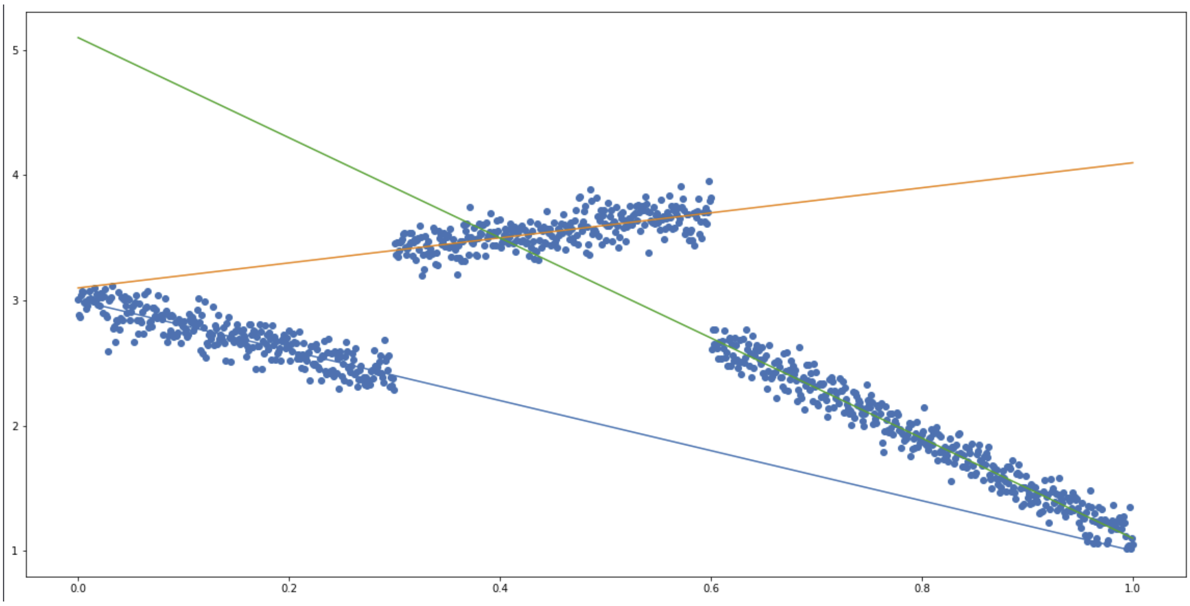
The hard problem I found is how to parametrize the Q matrix !

If i consider the Noisy Signal:

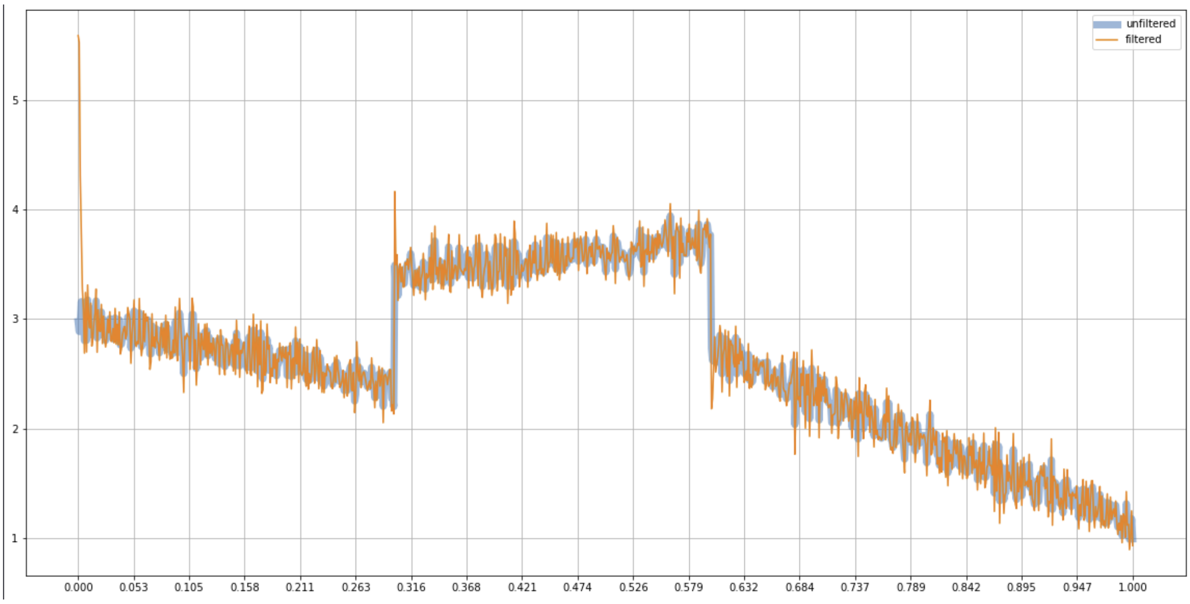
```
def piece1(k) : return -2 * k + 3
def piece2(k) : return k  + 3.1
def piece3(k) : return -4 * k + 5.1
```

```
def f(k):
    if k >= 0 and k < 0.3:      return piece1(k)
    elif k >= 0.3 and k < 0.6:  return piece2(k)
    elif k >= 0.6 and k <= 1:   return piece3(k)
    else: return 0

# params
time_0 = 0
time_max = 1
freq = 1000
time = np.linspace(time_0,  time_max, freq)
sigma_r = 0.1
r = np.random.normal(0, sigma_r, len(time))
m = np.array([f(k) for k in time] + r  )
plt.plot(time,[piece1(k) for k in time])
plt.plot(time,[piece2(k) for k in time])
plt.plot(time,[piece3(k) for k in time])
plt.scatter(time, m)
```



Kalman filtering with `Q = [[1, 0], [0, 1]]` results in



The noise is still too much, there is need of tuning. My guess is to scale down the 1 factor on the q matrix to be something like 10^{-n} with some n that optimizes the filtering process in the project/prediction part.

After playing around i found a good compromise with `Q = [[1/1000, 0], [0, 1/10000 ]] resulting in the following`

I lost some information near Time 0 (I can get a better initializzation state i guess) but the general filtered signal looks much better!

I read a paper about fuzzy logic optimization in kalman filtering, but i understood very little. I wonder if there is some form of bayesian multiplication process which allows to estimate better Q as measurements come in, given a little more memory to the system in which the kalman filter is installed.