ME C231B Assignment: Kalman Filtering (part 3)

Problem set due Thursday, April 5, 8:00PM. Submit a pdf file (a neat scan of your handwritten solution) via bCourses. The copier on 6th floor of Etcheverry is available 24/7, and can be used without a login to email scans to any `berkeley.edu` address.

---

1. (a) Verify that the equations for the signal recursion in the estimator (see *Part3* slides) can be rewritten as

$$\begin{aligned}\hat{x}_{k+1|k} &= (A_k - L_k C_k)\hat{x}_{k|k-1} + L_k y_k \\ \hat{x}_{k|k-1} &= I\hat{x}_{k|k-1} \\ \hat{x}_{k|k} &= (I - H_k C_k)\hat{x}_{k|k-1} + H_k y_k \\ \hat{w}_{k|k} &= -G_k C_k \hat{x}_{k|k-1} + G_k y_k\end{aligned}$$

where formula for L_k, H_k, G_k are given in terms of the iterated variances, all derived tjhoughout Part3 of the slides.

(b) Treating $\xi_k := \hat{x}_{k|k-1}$ as the state of the Kalman filter, the equations can be rewritten as

$$\begin{aligned}\xi_{k+1} &= (A_k - L_k C_k)\xi_k + L_k y_k \\ \hat{x}_{k|k-1} &= I\xi_k \\ \hat{x}_{k|k} &= (I - H_k C_k)\xi_k + H_k y_k \\ \hat{w}_{k|k} &= -G_k C_k \xi_k + G_k y_k\end{aligned}$$

Let K denote this system, with y_k as the input, and $\hat{x}_{k|k-1}, \hat{x}_{k|k}$ and $w_{k|k}$ as the outputs. Simply verify that the state-space model of K is

$$K: \quad \left[\begin{array}{c|c} (A_k - L_k C_k) & L_k \\ \hline I & 0 \\ I - H_k C_k & H_k \\ -G_k C_k & G_k \end{array}\right]$$

(c) For the steady-state Kalman Filter (problem data A, E, C, F, W is time-invariant, and the filter uses the constant, converged values for L_k, H_k and G_k), write `formSteadyStateKF.m` which

- implements the iteration to converge to constant values,
- Implements the state-equations above in part 1b, yielding a system K with input y_k and outputs $\hat{x}_{k|k-1}, \hat{x}_{k|k}$ and $w_{k|k}$

The function declaration line is shown below along with a significant amount of partial code to get you started. The partially completed m-file is included in the .zip file, and you can work from that as a starting point.

begin code

```
function [K,L,H,G,nIter,estErrVar] = formSteadyStateKF(A,E,C,F,W,TS)
% TS is specified sample time.  Use -1 to indicate a discrete-time
% system with unspecified sample time.
% Get dimensions
nX = size(A,1);
nY = size(C,1);
nW = size(W,1);
% Note: Your code should check for consistency of E, F, and C,
% but not necessary in this assignment.  If you have time, please
% use good programming practices.
% Initialize value for Sxkk1.  Convergence will occur for
% any positive-definite initialization.
Sxkk1 = eye(nX);
% Initialize ‘‘previous’’ values for L, H, G
prevL = zeros(nX,nY);
prevH = zeros(nX,nY);
prevG = zeros(nW,nY);
iterConverged = false;
nIter = 0;
while ~iterConverged
   nIter = nIter + 1;
   [~,Sxk1k,~,Sxkk,~,Lk,Hk,Gk,~] = ...
      KF231B([],Sxkk1,A,[],C,E,F,W,[],[]);
   if norm(Lk-prevL)<1e-8 && norm(Hk-prevH)<1e-8 && ...
         norm(Gk-prevG)<1e-8 && norm(Sxk1k-Sxkk1)<1e-8
      iterConverged = true;
      L =
      H =
      G =
      K = ss(   ,  ,[     ], [     ],TS);
      K.OutputGroup.xkk1 = 1:nX;
      K.OutputGroup.xkk = nX+1:2*nX;
      K.OutputGroup.wkk = 2*nX+1:2*nX+nW;
      % The steady-state variance of {hat x}_{k|k} - x_k is Sxkk.
      % This is a useful output to understand.  Using the
      % Chebychev inequality, you can (probabilistically)
      % bound the potential error in the state-estimate.
      estErrVar = Sxkk;
   else
      Sxkk1 =
      prevL =
      prevH =
      prevG =
   end
end
```

end code

Note: This offers similar functionality to the `kalman` command in Matlab, but adheres to the notation we used in the slides, and provides several interesting outputs of the estimator. As you move forward with your work and education, you shold be able to "translate" between various manners in which the Kalman filter is written down, and/or programmed.

(d) Since we used the `OutputGroups` feature of the Matlab Control toolbox (lines 31-33), it is very easy to select off the outputs of K which are most important to you. For example, if you only want (as output) the estimate $\hat{x}_{k|k}$, simply use

begin code

```
K_xkk = K('xkk',:);
```

end code

There is nothing to turn in here, but I want you to understand how `OutputGroups` work. They (along with `InputGroups`) can be useful when dealing with large, linear systems.

2. Consider a 1-d problem involving position and velocity estimation from noisy position and acceleration measurements. The discrete-time model abstraction is

$$\begin{aligned} x_1(k+1) &= x_1(k) + \Delta \cdot x_2(k) \\ x_2(k+1) &= x_2(k) + \Delta \cdot x_3(k) \\ x_3(k+1) &= x_3(k) + w_1(k) \\ y_1(k) &= x_1(k) + w_2(k) \\ y_2(k) &= x_3(k) + w_3(k) \end{aligned}$$

where x_1 is the position, x_2 is the velocity and x_3 is the acceleration. Since w will be modeled with zero mean, the model dictates that the acceleration x_3 is "roughly" constant, being "driven" by a zero-mean, random signal w_1. Measurements y_1 and y_2 are noisy measurements of the position and acceleration, respectively. Set $\Delta = 0.1$.

(a) Assume that the variance of the zero-mean sequence $w(k)$ is constant, namely

$$\Sigma_W = \begin{bmatrix} 0.3 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Find the steady-state Kalman filter gain L, and the steady-state error-variance, $\lim_{k\to\infty} \Sigma^x_{k|k}$. Use the code in `formSteadyStateKF`, and remember that your results are approximate (we stop when a numerical convergence criteria is met)

(b) Given the steady-state error-variance computed above, show that the probability that error in the the velocity-estimate (x_2) is greater than 0.73 is less than 0.25. Use the Chebychev inequality.

(c) In the .zip file, there are 3 template files for simulation of a time-varying Kalman filter with a time-varying model. They are named

- `KFexamplesTemplateWithoutSignals.m`
- `KFexamplesTemplateWithSignalsWithoutControl.m`
- `KFexamplesTemplateWithSignals.m`

These require specification of the time-varying martrices (A, E, C, F, Σ_W) as well as other variables, depending on the particular file. **For this problem, please study these files carefully, and understand how to use them**.

(d) Do 600 simulations (using `KFexamplesTemplateWithSignalsWithoutControl.m`), each of duration 200 steps. Interpret the 600 simulations as coming from a finite sample-space with 600 outcomes, where the random variables x_0 and w_k (here I am using the subscript to mean time whereas in the state equations for this problem, the subscript meant index, and the time was expressed with a (k) notation) satisfy

$$\mathbb{E}(x_0) = 0_{3\times 1}, \qquad \mathbb{E}(w_k) = 0_{3\times 1}, \qquad \mathbb{E}(w_k w_k^T) = \Sigma_W, \qquad \mathbb{E}(w_k w_j^T) = 0_{3\times 3} \text{ for } k \neq j$$

(e) Draw a histogram of the estimation error in x_2 (here the subscript 2 means the second component of x,. ie., velocity) at the end of the simulation (time-step 200), for all 600 outcomes. Is the final error greater than 0.73 in less than 25% of the outcomes?

3. Consider a time-invariant, discrete-time, linear system

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k \\ y_k &= Cx_k + Du_k \end{aligned}$$

with $\rho(A) < 1$. Recall for any square matrix M with eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$, the *spectral radius* is defined $\rho(M) := \max_{1\leq i\leq n} |\lambda_i|$. The condition $\rho(A) < 1$ means that the system is asympotically stable (with $u \equiv 0$, all solutions x decay to 0). For $\Omega \in [0, 2\pi]$, which is a discrete-frequency, associated with the sinusoid $e^{j\Omega k}$, define the frequency-response function

$$G(\Omega) := D + C(e^{j\Omega} I - A)^{-1} B$$

You have already learned that $G(\Omega)$ give information about the steady-state response of the system to (complex) sinusoids of the form $u_k = \bar{u} e^{j\Omega k}$. **Here we make connections to the average behavior of the system to random inputs. Important Theorem:** Suppose the input u is a sequence $u_0, u_1, \ldots$ of random variables, such that for all k, j, with $k \neq j$

$$\mathbb{E}(u_k) = 0_{n_u}, \qquad \mathbb{E}(u_k u_k^T) = I_{n_u}, \qquad \mathbb{E}(u_k u_j^T) = 0_{n_u}$$

(which we refer to as "unit-intensity, discrete-time, white noise"). Then, regardless of initial condition

$$\lim_{k\to\infty} \mathbb{E}(y_k) = 0, \qquad \lim_{k\to\infty} \mathbb{E}(y_k y_k^T) = \frac{1}{2\pi} \int_0^{2\pi} G(\Omega) G^*(\Omega) d\Omega$$

The behavior of the mean of y_k is easy to understand. We will verify the variance formula (approximately) using numerical simulation and numerical integration. It is proven using manipulations of the state equations, and invoking Parseval's relation, but will not be derived here.

(a) The command `H = drss(nX,nY,nU)` creates a random discrete-time system, with specified number of states, outputs and inputs. The command works such that $\rho(A) \leq 1$ (but not necessarily $\rho(A) < 1$), so we may need to scale A by small amount to guarantee stability. For example, the code below creates a 4-state, 3-output, 2-input stable system

begin code

```
nX = 4;
nY = 3;
nU = 2;
H = drss(nX,nY,nU);
if max(abs(eig(H.A)))>0.999; H.A = 0.99*H.A; end
```

end code

Experiment with this code, perform some step-responses, and use the `isstable` command to verify that the resulting systems are stable (ie., $\rho(A) < 1$).

(b) Given such a stable system, we can perform the integration

$$\frac{1}{2\pi}\int_0^{2\pi} G(\Omega)G^*(\Omega)d\Omega$$

numerically using `integral` and the command `freqresp`.

begin code

```
% Create function_handle which returns integrand (matrix-valued)
% as a function of OMEGA. This uses the command FREQRESP
% which calculates the frequency-response at a single, given frequency.
f = @(Omega) freqresp(H,Omega)*freqresp(H,Omega)';
%
% Call INTEGRAL, specifying limits of integration as 0 to 2*PI.  Make
% sure the command INTEGRAL knows that the integrand is not
% a scalar, but is array-valued (in our case, nY-by-nY)
Int = 1/(2*pi) * integral(f,0,2*pi,'ArrayValued',true')
%
% Note that although the integrand is a complex, hermitian
% matrix, the overall integral is purely real.  This can be proven
% but we will not focus on that here.  Some small numerical
% values for imaginary parts creep in due to roundoff.  Hence,
% take the real-part of the computed answer to clean up the result.
Int = real(Int)
```

end code

Experiment with this code on a few stable systems H.

(c) The command `norm`, as applied to systems, computes the frequency-domain integral (using state-space calculations, rather than numerical integration), but reduces it to a scalar using the "trace" operation. For stable linear systems, define

$$\|H\|_2 := \left(\mathrm{Tr}\left[\frac{1}{2\pi}\int_0^{2\pi} G(\Omega)G^*(\Omega)d\Omega\right]\right)^{\frac{1}{2}}$$

Recall that Tr is the *trace* of a square matrix, which is just the sum of the diagonal entries. Since that is a linear operation, it can also be moved into the integrand without changing the value. Verify that the computed integral above (variable `Int`), is related (through these expressions) to the results from `norm`, The use of the second argument, the number 2, signifies the the `norm` command that this specific norm ($\|\cdot\|_2$) is to be computed

begin code

```
sqrt(trace(Int))
norm(H,2)
```

end code

(d) Given such a stable system, we can approximate

$$\lim_{k\to\infty} \mathbb{E}(y_k y_k^T)$$

for the white-noise input u using `randn` and `lsim` (for simulations). We saw in the first Kalman Filter homework how to interpret the output of `randn` as generating a finite (but large, if we like) sample-space, with finite-duration (but as long duration as we like) sequences that have approximately the variances and correlations we want.

The code below carries out this numerical experiment.

begin code

```
% Decide on the duration (in time) of each input/output sequences.
% The formula has k -> INF, but we can only simulate for a finite
% duration.  In general, choose a duration "long" compared to the
% time-constants of the system.  Here, we simply pick a duration
% of 200 steps.
Nsteps = 200;

% So each instance (associated with a point in the sample space)
% of the u-sequence is an nU-by-200 array.  The convention in
% Matlab's simulation codes (lsim, Simulink, etc) is to
% represent this as a 200-by-nU array.

% Decide on the dimension of the sample-space. Take 2000 outcomes,
NsampleSpace = 2000;

% Hence, we need 2000, 200-by-nU arrays, as generated by RANDN.
% Create a 200-by-nU-by-2000 array with randn.
U = randn(Nsteps,nU,NsampleSpace);

% Initialize a nY-by-nY matrix to hold the expectation
% of the final value (of each simulation) of y*y^T.
E_y_yT = zeros(nY,nY);

% Our construction (from RANDN) had all outcomes equally likely,
% with probability equal to 1/NsampleSpace.  So, create a
% constant value for each individual outcome's probability
p = 1/NsampleSpace;

% Use a FOR loop to compute the response for all the
% sample-space outcomes
for i=1:NsampleSpace
   % Use LSIM to compute each response.
```

```
33
34      % Recall that U(:,:,i) is the 200-by-nU array (for input
35      % sequence) associated with the i'th sample point.
36      Y = lsim(H,U(:,:,i));
37
38      % By convention (LSIM), the matrix Y will be Nsteps-by-nY.
39      % The last value (our approximation to k->INF) is simply the
40      % last row of Y.
41      y_end  = Y(end,:)';
42      E_y_yT = E_y_yT + p*(y_end*y_end');
43  end
44
45  % Compare to limiting expected value to the integral
46  E_y_yT
47  Int
```

end code

They are not exactly the same, but if we take longer simulations and a larger sample space, our calculation will be closer to the limiting answer. Since we are already taking a relatively long simulation, the convergence is mostly improved by taking a larger sample space. Recall that RANDN generates input sequences that only approximate the white-noise process, but larger sample-spaces will make this approximation better.

(e) The Matlab command `covar` makes this computation of $\lim_{k\to\infty} \mathbb{E}(y_k y_k^T)$ directly (with state-space calculations, not esimated with a finite number of finite-duration simulations) under a slightly more general assumption, namely that for all k, j, with $k \neq j$

$$\mathbb{E}(u_k) = 0_{n_u}, \qquad \mathbb{E}(u_k u_k^T) = \Sigma_U, \qquad \mathbb{E}(u_k u_j^T) = 0_{n_u}$$

where Σ_U is a given, positive semidefinite matrix. We previously computed for $\Sigma_U = I_{n_u}$, so use that for comparison in this numerical example

begin code

```
1  covar(H,eye(nU))
2  Int      % exactly matches output of COVAR
3  E_y_yT   % approximately matches output of COVAR
```

end code

4. The model below is 1st-order Euler approximation for a mass falling through the atmosphere. The drag term accounts for variation in density (as a function of altitude) and damping forces that scale with the square of the velocity. There is also an unknown constant scale factor (modeled by x_3) called the "ballistic coefficient", which depend on the exact shape and aerodynamic characteristics of the mass, and is generally not known.

   The equations are

$$x_1(k+1) = x_1(k) + \Delta \cdot x_2(k) \qquad x_2(k+1) = x_2(k) + \Delta \cdot \left[\frac{\rho_0 e^{-\frac{x_1(k)}{\gamma}} x_2(k)^2}{x_3(k)} - g \right] + \Delta \cdot w_1(k)$$

   and $x_3(k+1) = x_3(k)$.

The states are as follows: x_1 is altitude; x_2 is velocity; x_3 is the (constant, but unknown) ballistic coefficient. The equation for x_1 represents that velocity is the derivative of position. The equation for x_2 represents Newton's law, and includes the effect of random force, represented by w_1. The equation for x_3 indicates that x_3 does not change (as it represents an unknown parameter). There is one measurement, a noisy measurement of the altitude, modeled as $y_1(k) = x_1(k) + w_2(k)$. We want to implement an Extended Kalman Filter (EKF) to estimate all 3 states.

We need the equations in the form

$$x_{k+1} = f(x_k) + Ew_k, \qquad y_k = h(x_k) + Fw_k$$

as well as the derivative matrices $\frac{df}{dx}$ and $\frac{dh}{dx}$ which are used in the EKF. These have function declaration lines

begin code

```
function xp1 = FallingObjectDynEq(x,delta,rho0,gamma,g)  % implements f(x)
function y = FallingObjectOutEq(x,delta,rho0,gamma,g)     % implements h(x)
function A = FallingObjectDynEqJac(x,delta,rho0,gamma,g)  % implements df/dx
function C = FallingObjectOutEqJac(x,delta,rho0,gamma,g)  % implements dh/dx
```

end code

These are supplied, and correct, and match exactly with equations given above. The Extended Kalman Filter is implemented, with function declaration line

begin code

```
function [xk1k,Sxk1k,xkk,Sxkk,Sykk1] = ...
   EKF231BToFinish(xkk1,Sxkk1,dfdx,dhdx,E,F,fhandle,hhandle,Swk,yk)
```

end code

(a) **Four (4) lines of code in that file are partially started, but you need to complete them to make it work.** Complete these lines, and verify that the other lines are correct, by matching the code with the slides in *Part4*. Save the completed file to a new name, `EKF231B.m`

(b) Work out the necessary partial derivatives by hand, and verify that the four `FallingObject` files are all correct

(c) Examine the script `ekfIterationScript.m`, which simulates the real system and implements the EKF on each iteration. This file is similar in layout the `Template` files provided earlier. Make sure you can connect the lines of code to the evolution of the real system and the EKF implementation.

(d) Run the script `ekfIterationScript.m`. It will make six plots of the state-estimation errors, and the actual state trajectories, and some "bounds" on the errors, using the variance matrix of the state estimation error, namely $\Sigma^x_{k|k}$. Run the script many times to see the general trends.

(e) Examine the script file to see how the initial conditions $x(0)$ are set, and how "large" the process disturbance and measurement noise are. Answer the following questions

i. Are initial conditions for the simulation being set randomly, and "consistent" with the probabalistic model for x_0?

ii. Is the disturbance sequence $w(k)$ being set randomly, and "consistent" with the probabalistic model for w?

iii. Look at the parameters (ρ_0, γ, g) and the typical altitudes and velocities at the beginning of the simulation. Why does the estimator not "'learn" much about x_3 during the early portion of the simulation?

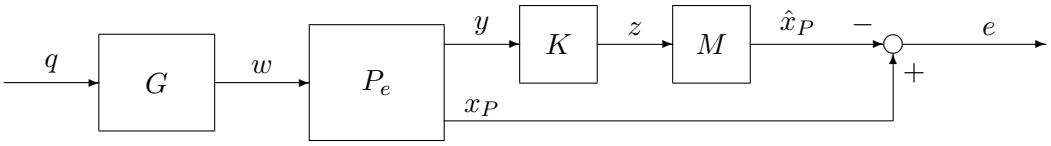
5. (The code you write for this problem will be used on the midterm) Suppose G, P, K are discrete-time, time-invariant linear systems, and M is a real matrix. Let G, P, and K have realizations

$$G = \left[\begin{array}{c|c} A_G & B_G \\ \hline C_G & D_G \end{array}\right], \qquad P = \left[\begin{array}{c|c} A_P & B_P \\ \hline C_P & D_P \end{array}\right], \qquad K = \left[\begin{array}{c|c} A_K & B_K \\ \hline C_K & D_K \end{array}\right]$$

Define P_e, with state-space model as

$$P_e = \left[\begin{array}{c|c} A_P & B_P \\ \hline C_P & D_P \\ I & 0 \end{array}\right]$$

Consider the interconnection



(a) Verify that a realization of the interconnection, with input q, output e and states $[x_G; x_P; x_K]$ is

$$\left[\begin{array}{ccc|c} A_G & 0 & 0 & B_G \\ B_P C_G & A_P & 0 & B_P D_G \\ B_K D_P C_G & B_K C_P & A_K & B_K D_P D_G \\ \hline -M D_K D_P C_G & I - M D_K C_P & -M C_K & -M D_K D_P D_G \end{array}\right]$$

(b) Write a Matlab function, `kfBuildIC.m`, with function declaration line

begin code

```
function IC = kfBuildIC(G,P,K,M)
```

end code

The function should make sure dimensions are compatible, and form the interconnection. The input arguments `G`, `P` and `K`, along with the output argument `IC` are all Matlab `ss` objects. The agument `M` should be a numerical matrix. The sample times of G, P and K should all be the same, and the sample time of `IC` should match that common value.

(c) I will post some test data that you can use to test your implementation. Look for an **Announcement** at bCourses.

(d) One application of this is as follows: P_e is the linear, time-invariant plant for which we want to do optimal state estimation. The system G is a signal model for the noise w, where

the signal q satisfies the "whiteness" assumptions in the KF derivation. The steady-state Kalman Filter K generates an estimate of both x_P and x_G, but the matrix M selects off only the estimate of x_P, which can be compared to the true value (at least in a simulation model, where the true value x_P is known). **Nothing to turn in here - make sure that explanation is clear**.