# ME C231B Assignment: Kalman Filtering (part 2)

Problem set due Thursday, March 15, 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. Generalizing (and programing) the batch-estimation: In class (and the notes) we derived the batch-formulation of Kalman filtering, generating the correct matrices to compute $\mathcal{L}(x_k|y_0,\ldots,y_{k-1})$. It was mentioned that we could also improve our estimates of previous states as new information (the $y$ values) are presented, for example

$$\mathcal{L}(x_0|y_0,\ldots,y_{k-1}), \quad \text{or} \quad \mathcal{L}(x_k|y_0,\ldots,y_{k-1},y_k)$$

In this problem, we introduce the appropriate matrices (and some recursions to make building the batch matrices easier) to compute

$$\mathcal{L}\left(\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{k-1} \\ x_k \end{bmatrix} \middle| \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{k-1} \end{bmatrix}\right)$$

**Remark:** You can easily modify the ideas below to get batch solutions to

$$\mathcal{L}\left(\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_j \end{bmatrix} \middle| \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_k \end{bmatrix}\right)$$

for any $j$ and $k$. Although we did not do it in class, there are recursive formulations (fixed amount of computation each step) to compute $\hat{x}_{j|k}$ for any $j$ and $k$. Recall that in lecture, we derived recursive formula only for $\hat{x}_{k-1|k}$ and $\hat{x}_{k|k}$. Now onto the batch generalization...

**Dimensions:** $x_k \in \mathbb{R}^{n_x}, y_k \in \mathbb{R}^{n_y}, w_k \in \mathbb{R}^{n_y}$

**Relevant Matrices:** Note that I am using blue to distinguish the state-evolution matrix $A_k$ at time $k$ from the caligraphic $\mathcal{A}_{k,j}$.

$$\mathcal{A}_{k,j} := A_{k-1}\cdots A_j \in \mathbb{R}^{n_x\times n_x} \quad k > j \geq 0$$

$$N_k := \begin{bmatrix} \mathcal{A}_{k,1}E_0 & \mathcal{A}_{k,2}E_1 & \mathcal{A}_{k,3}E_2 & \cdots & \mathcal{A}_{k,k-1}E_{k-2} & E_{k-1} \end{bmatrix} \in \mathbb{R}^{n_x\times kn_w}$$

$$\Gamma_k := \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 0 \\ E_0 & 0 & 0 & \cdots & 0 & 0 \\ \mathcal{A}_{2,1}E_0 & E_1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathcal{A}_{k-2,1}E_0 & \mathcal{A}_{k-2,2}E_1 & \mathcal{A}_{k-2,3}E_2 & \cdots & 0 & 0 \\ \mathcal{A}_{k-1,1}E_0 & \mathcal{A}_{k-1,2}E_1 & \mathcal{A}_{k-1,3}E_2 & \cdots & E_{k-2} & 0 \\ \mathcal{A}_{k,1}E_0 & \mathcal{A}_{k,2}E_1 & \mathcal{A}_{k,3}E_2 & \cdots & \mathcal{A}_{k,k-1}E_{k-2} & E_{k-1} \end{bmatrix} \in \mathbb{R}^{(k+1)n_x \times kn_w}$$

$$\Psi_k := \begin{bmatrix} I \\ \mathcal{A}_{1,0} \\ \mathcal{A}_{2,0} \\ \vdots \\ \mathcal{A}_{k-2,0} \\ \mathcal{A}_{k-1,0} \\ \mathcal{A}_{k,0} \end{bmatrix} \in \mathbb{R}^{(k+1)n_x \times n_x}, \quad R_k := \begin{bmatrix} C_0 \\ C_1\mathcal{A}_{1,0} \\ C_2\mathcal{A}_{2,0} \\ \vdots \\ C_{k-2}\mathcal{A}_{k-2,0} \\ C_{k-1}\mathcal{A}_{k-1,0} \end{bmatrix} \in \mathbb{R}^{kn_y \times n_x}$$

$$\mathcal{X}_k := \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{k-2} \\ x_{k-1} \\ x_k \end{bmatrix} \in \mathbb{R}^{(k+1)n_x} \qquad \mathcal{W}_{k-1} := \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_{k-2} \\ w_{k-1} \end{bmatrix} \in \mathbb{R}^{kn_w} \qquad \mathcal{Y}_{k-1} := \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{k-2} \\ y_{k-1} \end{bmatrix} \in \mathbb{R}^{kn_y}$$

$$S_k := \begin{bmatrix} F_0 & 0 & 0 & \cdots & 0 & 0 \\ C_1E_0 & F_1 & 0 & \cdots & 0 & 0 \\ C_2A_1E_0 & C_2E_1 & F_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ C_{k-2}\mathcal{A}_{k-2,1}E_0 & C_{k-2}\mathcal{A}_{k-2,2}E_1 & C_{k-2}\mathcal{A}_{k-2,3}E_2 & \cdots & F_{k-2} & 0 \\ C_{k-1}\mathcal{A}_{k-1,1}E_0 & C_{k-1}\mathcal{A}_{k-1,2}E_1 & C_{k-1}\mathcal{A}_{k-1,3}E_2 & \cdots & C_{k-1}E_{k-2} & F_{k-1} \end{bmatrix} \in \mathbb{R}^{kn_y \times kn_w}$$

**Initializations:** Recall that Matlab can handle empty arrays with some nonzero dimensions (though at least one dimension needs to be 0 so that the array is truly empty. In the definitions below, I've included, correctly, the nonzero dimension.

$$\Gamma_0 = \text{empty}, 0_{n_x \times 0}, \qquad \Psi_0 = I_{n_x \times n_x}, \qquad N_0 = \text{empty}, 0_{n_x \times 0},$$

$$S_0 = \text{empty}, 0_{0 \times 0}, \qquad R_0 = \text{empty}, 0_{0 \times n_x}, \qquad \mathcal{A}_{0,0} = I_{n_x \times n_x}$$

**Recursions:** for $i \geq 0$

$$\mathcal{A}_{i+1,0} = A_i\mathcal{A}_{i,0}, \quad \Psi_{i+1} = \begin{bmatrix} \Psi_i \\ \mathcal{A}_{i+1,0} \end{bmatrix}, \quad N_{i+1} = \begin{bmatrix} A_iN_i & E_i \end{bmatrix},$$

$$\Gamma_{i+1} = \begin{bmatrix} \begin{bmatrix} \Gamma_i & 0_{(i+1)n_x \times n_w} \end{bmatrix} \\ N_{i+1} \end{bmatrix}, \quad R_{i+1} = \begin{bmatrix} R_i \\ C_i \mathcal{A}_{i,0} \end{bmatrix}, \quad S_{i+1} = \begin{bmatrix} S_i & 0_{in_y \times n_w} \\ C_i N_i & F_i \end{bmatrix}$$

For $i \geq 0$, the quantities

$$\{\mathcal{A}_{i+1,0}, \Psi_{i+1}, N_{i+1}, \Gamma_{i+1}, R_{i+1}, S_{i+1}\}$$

can be calculated (recursively) from

$$\{\mathcal{A}_{i,0}, \Psi_i, N_i, \Gamma_i, R_i, S_i\} \text{ and } \{A_i, E_i, C_i, F_i\}$$

This is implemented in `buildBatch.m`, shown below.

begin code

```
function [calA,Psi,N,Gam,R,S] = buildBatch(calA,Psi,N,Gam,R,S,A,E,C,F)
% Initialization call uses one input argument (state dimension)
%    [calA,Psi,N,Gam,R,S] = buildBatch(nX)
% General recursive call uses the function definition line
%    [calA,Psi,N,Gam,R,S] = buildBatch(calA,Psi,N,Gam,R,S,A,E,C,F)
if nargin==1
   nx = calA;  % first (and only) argument is NX
   calA = eye(nx); Psi = eye(nx); Gam = zeros(nx,0);
   N = zeros(nx,0); S = zeros(0,0); R = zeros(0,nx);
else
   % Determine k, ny, nw, nx from dimensions of A, E, C, F and (eg) S
   [ny,nx] = size(C);
   nw = size(E,2);
   k = size(S,1)/ny;
   % Use recursions to build new, larger matrices.  Use temporary
   % variable for the updated calA and N, since the original values are
   % needed to define some of the other updated matrices.
   newCalA = A*calA;
   newN = [A*N E];
   Psi = [Psi;newCalA];
   Gam = [[Gam zeros((k+1)*nx,nw)];newN];
   R = [R;C*calA];
   S = [S zeros(k*ny,nw);C*N F];
   calA = newCalA;
   N = newN;
end
```

end code

**Evolution equations:**

$$\mathcal{X}_k = \Psi_k x_0 + \Gamma_k \mathcal{W}_{k-1}, \qquad \mathcal{Y}_{k-1} = R_k x_0 + S_k \mathcal{W}_{k-1}$$

Define $Z := \mathcal{X}_k, P := \mathcal{Y}_{k-1}$. We are interested in $\mathcal{L}(Z|P)$ and the variance of $Z - \mathcal{L}(Z|P)$. In general, these are

$$\mathcal{L}(Z|P) = \Sigma_{Z,P}\Sigma_P^{-1}(P - \mu_P) + \mu_Z, \qquad \Sigma_{Z-\mathcal{L}(Z|P)} = \Sigma_Z - \Sigma_{Z,P}\Sigma_P^{-1}\Sigma_{Z,P}^T$$

Specifically, for the Kalman filter application, the various quantities are

$$\mu_Z = \Psi_k m_0, \quad \mu_P = R_k m_0$$

and

$$\Sigma_Z = \Psi_k \Sigma_0 \Psi_k^T + \Gamma_k \bar{\Sigma}_W \Gamma_k^T$$
$$\Sigma_P = R_k \Sigma_0 R_k^T + S_k \bar{\Sigma}_W S_k^T$$
$$\Sigma_{Z,P} = \Psi_k \Sigma_0 R_k^T + \Gamma_k \bar{\Sigma}_W S_k^T$$

Plugging in everything gives our optimal estimate of all states from $x_0$ to $x_k$, based on all of the measurements $y_0$ to $y_{k-1}$,

$$\mathcal{L}(\mathcal{X}_k|\mathcal{Y}_{k-1}) = \left(\Psi_k \Sigma_0 R_k^T + \Gamma_k \bar{\Sigma}_W S_k^T\right)\left(R_k \Sigma_0 R_k^T + S_k \bar{\Sigma}_W S_k^T\right)^{-1}(\mathcal{Y}_{k-1} - R_k m_0) + \Psi_k m_0$$

Simple regrouping

$$\begin{aligned}\mathcal{L}(\mathcal{X}_k|\mathcal{Y}_{k-1}) &= \underbrace{\left(\Psi_k \Sigma_0 R_k^T + \Gamma_k \bar{\Sigma}_W S_k^T\right)\left(R_k \Sigma_0 R_k^T + S_k \bar{\Sigma}_W S_k^T\right)^{-1}}_{:=L_k^{\text{batch}}} \mathcal{Y}_{k-1} \\ &\quad + \underbrace{\left[\Psi_k - \left(\Psi_k \Sigma_0 R_k^T + \Gamma_k \bar{\Sigma}_W S_k^T\right)\left(R_k \Sigma_0 R_k^T + S_k \bar{\Sigma}_W S_k^T\right)^{-1} R_k\right]}_{:=V_k^{\text{batch}}} m_0\end{aligned}$$

and error variance

$$\begin{aligned}\Sigma_{\mathcal{X}_k - \mathcal{L}(\mathcal{X}_k|\mathcal{Y}_{k-1})} &= \Psi_k \Sigma_0 \Psi_k^T + \Gamma_k \bar{\Sigma}_W \Gamma_k^T \\ &\quad -(\Psi_k \Sigma_0 R_k^T + \Gamma_k \bar{\Sigma}_W S_k^T)(R_k \Sigma_0 R_k^T + S_k \bar{\Sigma}_W S_k^T)^{-1}(\Psi_k \Sigma_0 R_k^T + \Gamma_k \bar{\Sigma}_W S_k^T)^T\end{aligned}$$

The diagonal block-entries of $\Sigma$ (each of dimension $n_x \times n_x$) are most important, as they signify the variance ("spread") in the error of the estimate of each time-instance of the state.

Summarizing: for each $k \geq 1$, there exist matrices $L_k^{\text{batch}}$ and $V_k^{\text{batch}}$ such that

$$\mathcal{L}(\mathcal{X}_k|\mathcal{Y}_{k-1}) = L_k^{\text{batch}} \mathcal{Y}_{k-1} + V_k^{\text{batch}} m_0$$

The function `batchKF` calculates these, using all of the ideas presented.

begin code

```
function [LkBatch,VkBatch,ekVar] = ...
       batchKF(Amat,Emat,Cmat,Fmat,Sigma0,SigmaW,k)
nx = size(Amat,1);
% Initialize the 6 batch matrices
[calA,Psi,N,Gam,R,S] = buildBatch(nx);

% Fill in batch matrices in order to estimate x0 to xk, as linear
% combinations of y from 0,1,...,k-1.  This needs the state-space matrices
% defined on i = [0,1,...,k-1]
for i=0:k-1
   Ai = Amat(:,:,i+1);  % i+1 is needed because Matlab indices start at 1
   Ei = Emat(:,:,i+1);
   Ci = Cmat(:,:,i+1);
```

```
    Fi = Fmat(:,:,i+1);
    [calA,Psi,N,Gam,R,S] = buildBatch(calA,Psi,N,Gam,R,S,Ai,Ei,Ci,Fi);
end
% The next line assumes that the variance of w_k is constant across k,
% given by the single matrix SigmaW.  The KRON command simply repeats this
% matrix in a block diagonal form, as in the lecture slides.
SigmaBarW = kron(eye(k),SigmaW);
SigmaZ = Psi*Sigma0*Psi' + Gam*SigmaBarW*Gam';
SigmaP = R*Sigma0*R' + S*SigmaBarW*S';
SigmaZP = Psi*Sigma0*R' + Gam*SigmaBarW*S';
LkBatch = SigmaZP/SigmaP;
VkBatch = Psi - Lk*R;
ekVar = SigmaZ - Lk*SigmaZP';
```

end code

Here $k \geq 1$ is an integer, and `Amat` is a 3-dimensional array, of dimension `[nX, nX, N]`, where `N` $>$ `k`, and for each nonnegative $i$, `Amat(:,:,i+1)` equals $A_i$ (**note the index issue that we being our sequences with an index of 0, but Matlab begins their arrays with an index of 1**).

Finally, if $\tilde{L}_k$ is defined as the last $2n_x$ rows of $L_k^{\text{batch}}$ and $\tilde{V}_k$ as the last $2n_x$ rows of $V_k^{\text{batch}}$, for example with the code

begin code

```
Ltildek = LkBatch(end-2*nX+1:end,:)
Vtildek = VkBatch(end-2*nX+1:end,:)
```

end code

then

$$\begin{bmatrix} \hat{x}_{k-1|k-1} \\ \hat{x}_{k|k-1} \end{bmatrix} = \mathcal{L}\left( \begin{bmatrix} x_{k-1} \\ x_k \end{bmatrix} \middle| \mathcal{Y}_{k-1} \right) = \tilde{L}_k \mathcal{Y}_{k-1} + \tilde{V}_k m_0$$

which is more related to the recursive derivation in lecture, which did not consider estimates of older states using newer measurements (eg., in lecture, we did not derive the recursion for $\hat{x}_{4|9}$ (estimate of $x_4$ based on $y_0, y_1, \ldots, y_9$).

(a) Verify the expressions in the **Recursions** section, given the definitions in the **Relevant Matrices** section.

(b) Verify the expressions in the **Evolution Equations** section.

(c) Examine the estimation problem solved below. The command `batchKF.m` is included in the assignment .zip file, and has the `buildBatch` code as a local function, so you do not have to create these files. Note that for $k = 1, 2, \ldots, 6$, we use the batch-code to get the best estimate of $\mathcal{X}_k$ from $\mathcal{Y}_{k-1}$. The code displays certain rows of $L_k^{\text{batch}}$ and $V_k^{\text{batch}}$ which are specifically relevant for the estimate $\hat{x}_{k-1|k-1}$

begin code

```
Amat = repmat(1,[1 1 20]);
nX = size(Amat,1);
```

```
Emat = repmat(0,[1 1 20]);
Cmat = repmat(1,[1 1 20]);
Fmat = repmat(1,[1 1 20]);
sX = 1000; sW = 1; m0=2;
for k=1:6
   [LkBatch,qkBatch,VkBatch,eVar] = ...
           batchKF(Amat,Emat,Cmat,Fmat,m0,sX,sW,k);
   LkBatch(end-2*nX+1:end-nX,:)
   VkBatch(end-2*nX+1:end-nX)
end
```

end code

i. Interpret the estimation problem, specifically
   - In the absence of process noise, how would the state evolve?
   - In the absence of measurement noise, how are the state and measurement related?
   - How are the process noise, measurement noise and initial condition variances related?

ii. The solution is computed for several different horizons. Study the "gain" matrix `LkBatch` and offset `qkBatch` and interpret how the optimal estimate is processing $\mathcal{Y}_{k-1}$, and using $m_0$ to obtain the estimate $x_{k-1|k-1}$

iii. If you had to intuitively design an estimator for exactly this problem, **without any Kalman filtering knowledge**, given the specific statistical properties of the initial condition, process noise, and measurement noise, would you be likely to pick a similar estimation strategy?

iv. Repeat the exercise for the following example

begin code

```
Amat = repmat(1,[1 1 20]);
nX = size(Amat,1);
Emat = repmat(0,[1 1 20]);
Cmat = repmat(1,[1 1 20]);
Fmat = repmat(1,[1 1 20]);
sX = 0.1; sW = 5; m0=4;
for k=1:6
   [LkBatch,qkBatch,VkBatch,eVar] = ...
           batchKF(Amat,Emat,Cmat,Fmat,m0,sX,sW,k);
   LkBatch(end-2*nX+1:end-nX,:)
   VkBatch(end-2*nX+1:end-nX)
end
```

end code

2. **Enhance KF code:** The code for the Kalman filter, as derived in class, is included in the assignment .zip file.

   (a) Included in the assignment .zip file is a function `KF231BtoBuildOn.m` with declaration line

begin code

```
function [xk1k,Sxk1k,xkk,Sxkk,Sykk1] = ...
                  KF231BtoBuildOn(xkk1,Sxkk1,A,B,C,E,F,Swk,uk,yk)
```

end code

Modify the code, renaming it `KF231B`, to have four additional outputs, namely as

begin code

```
function [xk1k,Sxk1k,xkk,Sxkk,Sykk1,Lk,Hk,Gk,wkk] = ...
                    KF231B(xkk1,Sxkk1,A,B,C,E,F,Swk,uk,yk)
```

end code

for the matrices $L_k, H_k, G_k$ and the estimate $\hat{w}_{k|k}$ as defined in the slides. This should only require adding one line of code to properly define $L_k$, and one or two lines for $\hat{w}_{k|k}$. If you look carefully at the existing code, you will see that $H_k$ and $G_k$ are already defined. If you look at the variables which already exist, you should be able to do this task without any additional "inverses" (or the use of the backslash operator). Make sure to modify the `help` for `KF231B.m` as follows.

begin code

```
% Implements one step of the Kalman filter, using the notation in
% slides at the end of Estimation, Part 3.  The input arguments are:
%    xkk1 = xhat_{k|k-1}
%    Sxkk1 = Sigma^x_{k|k-1}
%    A, B, C, E, F: state matrices at time k
%    Swk = variance in zero-mean disturbance w_k
%    uk = deterministic control input u_k
%    yk = measurement y_k
% The output arguments are:
%    xk1k = xhat_{k+1|k}
%    Sxk1k = Sigma^x_{k+1|k}
%    xkk = xhat_{k|k}
%    Sxkk = Sigma^x_{k|k}
%    Sykk1 = Sigma^y_{k|k-1}
%    Lk (from last slide, Part 3) for:
%           xhat_{k+1|k} = Ak x_{k|k-1} + Lk*(yk - Ck*x_{k|k-1})
%    Hk (from fact #16), for:
%           xhat_{k|k} = x_{k|k-1} + Hk*(yk - Ck*x_{k|k-1})
%    Gk (from fact #17), for: what_{k|k} = Gk*ek
%    wkk = what_{k|k}
% The input signals, xkk1, uk, yk may all be empty matrices, implying
% that the function will only update the error variances, and will not
% provide any state estimates (so xk1k, xkk and wkk will be returned
% empty as well).
```

end code

3. **Steady-State Behavior:** As mentioned in class, if the process state-space matrices $(A_k, E_k, C_k, F_k)$ and the variance of $w_k$ (which we now notate as $W_k := \mathbb{E}(w_k w_k^T)$) are constant (ie., not time-varying), then (under very mild technical conditions, which we will not worry about) the gain and variance matrices in the Kalman filter converge to steady-state values, namely

$$\lim_{k\to\infty} L_k = \bar{L}, \qquad \lim_{k\to\infty} H_k = \bar{H}, \qquad \lim_{k\to\infty} \Sigma^x_{k|k} = \bar{\Sigma}^x, \qquad \lim_{k\to\infty} \Sigma^x_{k|k-1} = \bar{\Sigma}^x_{-1}$$

Create some random data $(A, E, C, F, W)$, and confirm the convergence by calling `KF231B` in a loop, monitoring the differences between $L_{i+1}$ and $L_i$ (and so on for the other matrices), and exiting the loop when suitable convergence is attained.

**Hint:** Recall that `KF231B` can be called in such a way that signals are not needed, and only the variances and gains are updated. Carefully read the `help` again if needed.

4. **Separating signal from noise:** Suppose $P_1$ and $P_2$ are linear systems (possibly time-varying), with known state-space models. The input to system $P_1$ is $v$ (producing output $y_1$), and the input to system $P_2$ is $d$ (producing output $y_2$). Both $v$ and $d$ are random sequences, with the following properties (for all $k$ and $j \neq k$)

$$\mathbb{E}(v_k) = 0, \quad \mathbb{E}(d_k) = 0,$$

and

$$\mathbb{E}(v_k v_k^T) = V_k, \quad \mathbb{E}(d_k d_k^T) = D_k, \quad \mathbb{E}(v_k d_k^T) = 0, \quad \mathbb{E}(v_k v_j^T) = 0 \quad \mathbb{E}(d_k d_j^T) = 0, \quad \mathbb{E}(v_k d_j^T) = 0$$

Let $y := y_1 + y_2$. Draw a simple block diagram, and label $y_1$ as "noise" and $y_2$ as "signal."

   (a) **Explain how we can use the Kalman filtering theory to estimate $y_2$ from $y$.** Define the appropriate matrices, so that appropriate calls to `KF231B` along with some simple arithmetic would produce the desired estimate.

   **Note:** In this problem we can interpret $y_2$ as a meaningful signal, which is additively corrupted by noise, $y_1$, to produce a measurement $y$. The goal of filtering is to recover the meaningful signal ($y_2$) from the measured, "noisy" signal $y$. **Remark:** Be careful with subscripts. In the notation above, the subscript on $d$ and $v$ was interpreted as a time-index, but the subscript on $y_1$ and $y_2$ is just referring to two different signals (each of which depends on time).

   (b) In the commented example below, $P_1$ is a first-order high-pass filter, and $P_2$ is a first-order low-pass filter, and the separation is done using the steady-state values of the Kalman filter. Study and run the code. Carefully examine the plots.

begin code

```
%% Separating Signal from Noise
% ME C231B, UC Berkeley, Spring 2018

%% Sample Time
% For this discrete-time example, set TS=-1.  In Matlab, this
% just means an unspecified sampling time, totally within the
% context of pure discrete-time systems.
TS = -1;

%% Create high-pass filter
P1 = 0.4*tf([.5 -.5],[1 0],TS);
[A1,E1,C1,F1] = ssdata(P1);
nX1 = size(A1,1); % will be 1
nW1 = size(E1,2); % will be 1

%% Create low-pass filter
P2 = tf(.04,[1 -.96],TS);
[A2,E2,C2,F2] = ssdata(P2);
nX2 = size(A2,1); % will be 1
nW2 = size(E2,2); % will be 1
```

```
21
22  %% Bode plot of both
23  bOpt = bodeoptions;
24  bOpt.PhaseVisible = 'off';
25  bOpt.MagUnits = 'abs';
26  bOpt.MagScale = 'log';
27  bOpt.FreqScale = 'linear';
28  bOpt.Xlim = [0 pi];
29  bOpt.Ylim = [1e-4 2];
30  bodeplot(P2,'r',P1,'k',bOpt)
31
32  %% Form overall system which adds the outputs
33  A = blkdiag(A1,A2);
34  E = blkdiag(E1,E2);
35  C = [C1 C2];
36  F = [F1 F2];
37  nX = size(A,1);
38  nY = size(C,1);
39  nW = size(E,2);
40
41  %% Noise variance and initial condition variance
42  % Keep it simple, and make everything Identity (appropriate
43  % dimension)
44  SigW = eye(nW);
45  Sxkk1 = eye(nX);
46
47  %% Run several iterations to get the steady-state Kalman Gains
48  nIter = 40;
49  for i=1:nIter
50     Swk = SigW;
51     [~,Sxk1k,~,Sxkk,Sykk1,Lk,Hk,Gk,~] = ...
52        KF231B([],Sxkk1,A,[],C,E,F,Swk,[],[]);
53     Sxkk1 = Sxk1k;
54  end
55
56  %% Form Kalman filter with 3 outputs
57  AKF = A-Lk*C;
58  BKF = Lk;
59  CKF = [eye(nX);eye(nX)-Hk*C;-Gk*C];
60  DKF = [zeros(nX,nY);Hk;Gk];
61  SSKF = ss(AKF,BKF,CKF,DKF,TS);
62
63  %% Form matrix to extract estimate of y2_{k|k}
64  % We need [0 C2]*xhat_{k|k} + [0 F2]*what_{k|k}.  Everything
65  % is scalar dimension, but we can form this matrix properly
66  % so that the example would work on other systems too.
67  M = [zeros(nY,nX) zeros(nY,nX1) C2 zeros(nY,nW1) F2];
68
```

```
%% Bode plot of filter
% It makes sense that the filter will try to "pass" some
% low frequencies, to preserve y2, but will cutoff
% high-frequencies to reject y1.  The "pass" region should
% extend over the region where P2 has modest gain.  The Bode
% magnitude plot confirms this
bodeplot(P2,'r',P1,'k',M*SSKF,bOpt)
legend('P2','P1','Filter');

%% Single Simulation
% Create a w sequence consistent with variance assumption
wSeq = randn(100,2);
%% Get y1 and y2 (separate simulations) for later comparison
y1 = lsim(P1,wSeq(:,1));
y2 = lsim(P2,wSeq(:,2));
y = y1 + y2;
%%
% Form the cascade (system output goes directly to Kalman
% Filter), and simulate, obtaining the outputs of Kalman
% Filter
Est = lsim(SSKF*ss(A,E,C,F,TS),wSeq);

%% Form Estimate of y2
% Est matrix is 100-by-6, so use transpose correctly to do
% reconstruction as a matrix multiply
y2Est = (M*Est')';

%% Plot
subplot(1,2,1);
plot(0:99,y2,'b+',0:99,y2Est,'ko',0:99,y,'r*');
legend('y2 (actual)','y2 (Est)','y (Measured)');
subplot(1,2,2);
plot(0:99,y2,'b+',0:99,y2Est,'ko');
legend('y2 (actual)','y2 (Est)');
```

end code

(c) In the code cell above, we iterated 40 steps to get the "steady-state" gains, but did not verify convergence. Using simple methods, estimate approximately how many steps are actually needed for "convergence."

(d) Follow the main ideas of the template `KFexamplesTemplateWithSignalsButNoControl.m` to rerun this example using the time-varying (not steady-state) filter. Compare the estimates achieved by the time-varying filter and the steady-state filter over the first 20 time steps.

5. **Converting between Kalman filter formulations:** In our class notes, we formulated the disturbance/noise as one vector valued noise variable, $w_k \in \mathbb{R}^{n_w}$, with two matrices, $E \in \mathbb{R}^{n \times n_w}$ and $F \in \mathbb{R}^{n_y \times n_w}$ where $Ew$ additively affects the state evolution, and $Fw$ additively affects the

measurement. The variance $w_k$ was $\Sigma_W \in \mathbb{R}^{n_w \times n_w}$. By contrast, in the Matlab `kalman` implementation, there are two separate disturbance/noises, labeled $d \in \mathbb{R}^{n_d}$ and $v \in \mathbb{R}^{n_y}$. The state evolution is additively affected by $Gd$, and the measurement by $Hd + v$ for matrices $G \in \mathbb{R}^{n_x \times n_d}$ and $H \in \mathbb{R}^{n_y \times n_d}$. The variance of the combined vector is

$$\mathbb{E}\begin{bmatrix} d_k d_k^T & d_k v_k^T \\ v_k d_k^T & v_k v_k^T \end{bmatrix} = \begin{bmatrix} Q & N \\ N^T & R \end{bmatrix}$$

Suppose the problem is specified in the Matlab format, namely

$$\{n_d, G, H, Q, N, R\}$$

How do you convert this description into our class format, namely

$$\{n_w, E, F, \Sigma_W\}$$

so that the problems are equivalent? **Hint:** Look at the means and variances of

$$\begin{bmatrix} Gd \\ Hd + v \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} Ew \\ Fw \end{bmatrix}$$

For the problems to be equivalent, these statistical properties need to be equal.

6. This problem generalized fact #10 in the Kalman derivation. Suppose $D_k$ and $R_k$ are known matrices, and a random variable $\xi_k$ is defined $\xi_k := D_k x_k + R_k w_k$.

   (a) Define $\hat{\xi}_{k|k-1} := \mathcal{L}(\xi_k | y_0, y_1, \ldots, y_{k-1})$. Show that

   $$\hat{\xi}_{k|k-1} = D_k \hat{x}_{k|k-1}$$

   (b) Define $\hat{\xi}_{k|k} := \mathcal{L}(\xi_k | y_0, y_1, \ldots, y_{k-1}, y_k)$. Show that

   $$\hat{\xi}_{k|k} = D_k \hat{x}_{k|k} + R_k W_k F_k^T \left(\Sigma^y_{k|k-1}\right)^{-1} e_k$$

7. In the derivation in class, there was a missing variance calculation, associated with $\hat{x}_{k|k}$ in Fact 16. It is easy to derive this. Recall the setup for Fact 16:

   $$Z = x_k, \quad P = \mathcal{Y}_{k-1}, \quad Q = y_k$$

   We want to determine $\Sigma^x_{k|k}$, which is defined to be $\Sigma_{x_k - \hat{x}_{k|k}}$. In terms of $Z, P$ and $Q$, this is

   $$\Sigma_{x_k - \hat{x}_{k|k}} = \Sigma_{Z - \mathcal{L}(Z|P,Q)}$$

   The general formula for $\Sigma_{Z-\mathcal{L}(Z|P,Q)}$ is

   $$\Sigma_{Z-\mathcal{L}(Z|P)} \; - \; \Sigma_{Z-\mathcal{L}(Z|P),Q} \Sigma^{-1}_{Q-\mathcal{L}(Q|P)} \Sigma^T_{Z-\mathcal{L}(Z|P),Q}$$

   as derived in Fact #6. Fact #16 has all of these terms (for the specific choices of $Z, P, Q$). **Task:** Substitute these expressions to derive

   $$\Sigma^x_{k|k} = \Sigma^x_{k|k-1} - \Sigma^x_{k|k-1} C^T \left(\Sigma^y_{k|k-1}\right)^{-1} C \Sigma^x_{k|k-1}$$

   as claimed in the final Kalman Filter equations.

8. (nothing to turn in - please read carefully) In the derivation in class, there was no control input in the dynamics - we derived the Kalman Filter under the dynamics

$$x_{k+1} = A_k x_k + E_k w_k$$

If the dynamics include control, then

$$x_{k+1} = A_k x_k + B_k u_k + E_k w_k$$

There are a few situations to consider:

**Case 1:** the sequence $\{u_k\}_{k=0}^{\infty}$ is deterministic and known. This occurs if the signal $u_k$ is just a prespecified forcing function.

**Case 2:** the value of $u_k$ is a linear combination of $\mathcal{Y}_k$, say $u_k = J_k \mathcal{Y}_k$ for some deterministic matrix $J_k$. Note that the dimension of $J_k$ changes with $k$. In this case, since $\mathcal{Y}_k$ is a random variable, it follows that $u_k$ is a random variable. This situation occurs if there is a feedback controller mapping the measurement $y_k$ to $u_k$ through a linear, time-varying dynamical system.

**Case 3:** the value of $u_k$ is a linear combination of $\mathcal{Y}_k$ and a deterministic, known signal. This situation is a combination of the two simpler cases, and is very common (a feedback controller with an external reference input). Once they are understood, handling this case is quite easy.

In order to modify the derivation, it is important to consider all of the slides, starting with some of the batch matrix manipulations, and especially at Fact #8. Regarding Fact #8, the two cases differ as follows:

(a) For Case 1, the expression for $x_k$ still is linear in $x_0$ and $\mathcal{W}_{k-1}$, but also has a linear term with $\mathcal{U}_{k-1}$. Since $u$ is deterministic, this only changes the mean of $x_k$, but not any correlations or variances.

(b) For Case 2, since $y_k$ is a linear combination of $x_k$ and $w_k$, the expression for $x_k$ is of the identical form (linear in $x_0$ and $\mathcal{W}_{k-1}$, but the matrices are different, and involve $B_0, B_1, \ldots, B_{k-1}$ and $J_0, J_1, \ldots, J_{k-1}$.

Because of these simple differences, the boxed conclusions from Fact 8 remain true:

$$\Sigma_{x_k, w_k} = 0, \qquad \Sigma_{\mathcal{Y}_{k-1}, w_k} = 0.$$

Since Facts 9-18 do not involve the evolution equation for $x_{k+1}$, they are unchanged.

Fact 19 does change, since it involve the evolution equation. For Case 1, the term $B_k u_k$ is simply added to the update, since it is not a random variable. For Case 2, the evolution is $x_{k+1} = A_k x_k + B_k J_k \mathcal{Y}_k + E_k w_k$. The lineariy of the affine estimator means that

$$\begin{aligned}
\mathcal{L}(x_{k+1}|\mathcal{Y}_k) &= \mathcal{L}(A_k x_k + B_k J_k \mathcal{Y}_k + E_k w_k | \mathcal{Y}_k) \\
&= A_k \mathcal{L}(x_k|\mathcal{Y}_k) + B_k J_k \mathcal{L}(\mathcal{Y}_k|\mathcal{Y}_k) + E_k \mathcal{L}(w_k|\mathcal{Y}_k) \\
&= A_k \hat{x}_{k|k} + B_k J_k \mathcal{Y}_k + E_k \hat{w}_{k|k} \\
&= A_k \hat{x}_{k|k} + B_k u_k + E_k \hat{w}_{k|k}
\end{aligned}$$

where we used the fact that $\mathcal{L}(\mathcal{Y}_k|\mathcal{Y}_k) = \mathcal{Y}_k$ and $u_k = J_k \mathcal{Y}_k$.

So, the change to the boxed expression on Fact 19 is simply the addition of the term $B_k u_k$, in both cases.

Fact 20 is unchanged. Fact 21 involves the expression $x_{k+1} - \hat{x}_{k+1|k}$, so it must be studied. However the only difference (for both Case 1 and 2) is that now both terms, $x_{k+1}$ and $\hat{x}_{k+1|k}$ include the additional additive term $B_k u_k$. By subtraction, these cancel, and the calculation of the variances is unaffected.

Hence, for both Case 1 and Case 2 (and Case 3, by a simple argument involving both ideas), the only change to the Kalman filter equations is the inclusion of the $b_k u_k$ term in the **State Prediction** equation, namely

$$\hat{x}_{k+1|k} = A_k \hat{x}_{k|k} + B_k u_k + E_k W_k F_k^T \left(\Sigma^y_{k|k-1}\right)^{-1} e_k$$