ME C231B Assignment: Kalman Filtering (part 1)

Problem set due Tuesday, February 27, 8:00PM. You must submit a pdf file (from a typed document, or a neat scan of your handwritten/typed 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. **Histograms in Matlab:** If `vecD` is a row (or column) vector, the command `hist(vecD)` draws a *histogram* of the numerical values in `vecD`. Recall what a histogram is from high-school statistics. The command determines the minimum and maximum values contained within the data; splits that interval into several (default is 10) equally spaced bins; counts the number of data points within each bin; and plots the counts (vertical axis) versus the bin-intervals (horizontal) axis. Such a figure shows (in a coarse, but informative manner) how the numerical values within the data are distributed. Obviously the order of the numbers with the vector `vecD` does not affect the plot, since the plot represents counts of values within each bin.

   The command `linspace` generates points uniformly spaced between specified limits. Histograms of such data should be flat, since there will be a uniform number of points within intervals of the same width. Using `randperm` (which creates a random permutation of the indices), the order of the data can be rearranged, but the histogram remains the same.

   begin code

```
N = 500;
X = linspace(-3,2,N);
subplot(2,1,1); hist(X);
I = randperm(N);
Y = X(I);
isequal(X,Y)
subplot(2,1,2); hist(Y);
```

   end code

   For a collection of angles uniformly chosen around (and close to) 0, the cosine of the angles are more often close to 1 than to smaller values. A histogram makes this evident

   begin code

```
N = 500;
X = linspace(-1,1,N);
subplot(2,1,1); hist(X);
xlabel('Angle, rads'); ylabel('Counts')
subplot(2,1,2); hist(cos(X));
xlabel('COS(Angle, rads)'); ylabel('Counts')
```

   end code

   The command `logspace` generates points logarithmically spaced between specified limits. Histograms of such data should show a nonuniform distribution, since the horizontal axes in a his-

togram is (by default) linearly spaced. Creating 500 points logarithmically spaced between 0.1 and 10 will give 250 points from 0.1 to 1, and 250 points from 1 to 10. Hence there will be many more points in the bin [0 1] than there are in the bin [9 10]. Also, again, using `randperm`, the order of the data can be rearranged, but the histogram remains the same.

begin code

```
N = 500;
X = logspace(-1,1,N);
subplot(2,1,1); hist(X);
I = randperm(N);
Y = X(I);
isequal(X,Y)
subplot(2,1,2); hist(Y);
```

end code

A finer (or coarser) bin selection can be done with additional arguments. The simplest syntax is to alter the number of bins.

begin code

```
nBins = 25;
subplot(2,1,1); hist(X,nBins);
```

end code

Experiment more with `histogram` and be comfortable with its behavior to examine the distribution of scalar-valued data.

2. Suppose X is a random variable defined on a probability model consisting of a sample space Ω and probability law $\mathbf{P}$. Let $a, b \in \mathbb{R}$ be real numbers. Define a new random variable $Y := aX + b$. In other words, for each $\omega \in \Omega$, define $Y(\omega) := aX(\omega) + b$. Find μ_Y, Σ_{YY} and Σ_{XY} in terms of a, b, μ_X and Σ_{XX}.

3. Suppose X is a random variable defined on a probability model consisting of a sample space Ω and probability law $\mathbf{P}$. Let $a, b \in \mathbb{R}$ be real numbers. Define a new random variable $Y := a(X - \mu_X) + b$. In other words, for each $\omega \in \Omega$, define $Y(\omega) := a\left(X(\omega) - \mu_X\right) + b$. Find μ_Y, Σ_{YY} and Σ_{XY} in terms of a, b, μ_X and Σ_{XX}.

4. Suppose X and Y are vector-valued random variables, and matrices A, B, C, D are matrices of appropriate dimension. Define

$$W := AX + BY, \qquad Z := CX + DY$$

Verify the expressions below for μ_W and Σ_W in terms of $\mu_X, \mu_Y, \Sigma_X, \Sigma_{X,Y}, \Sigma_Y$,

$$\mu_W = A\mu_X + B\mu_Y, \qquad \Sigma_W = A\Sigma_X A^T + A\Sigma_{X,Y}B^T + B\Sigma_{Y,X}A^T + B\Sigma_Y B^T$$

Similarly, find expressions for $\mu_Z, \Sigma_{W,Z}, \Sigma_Z$ in terms of $\Sigma_X, \Sigma_{X,Y}, \Sigma_Y$.

5. Let $\mathbb{1}_M \in \mathbb{R}^M$ denote the $M \times 1$ column vector, whose elements are all the number 1. Suppose $A \in \mathbb{R}^{n \times M}$. Let a_k denote the k'th column of A. Show that

$$\frac{1}{M}\sum_{k=1}^{M} a_k = \frac{1}{M} A \cdot \mathbb{1}_M, \qquad \frac{1}{M}\sum_{k=1}^{M} a_k a_k^T = \frac{1}{M} AA^T$$

(a) For an array `A`, how are the two assignments below related to the above expressions?

begin code
```
B = (1/size(A,2))*A*ones(size(A,2),1)
C = (1/size(A,2))*A*A'
```
end code

6. If $A \in \mathbb{R}^{n \times M}$, then `mean(A,1)` computes the **average** of each column (ie., "down" the first dimension), returning a $1 \times M$ array, while `mean(A,2)` computes the **average** of each row (ie., "across" the 2nd dimension), returning a $n \times 1$.

(a) For an array `A`, what do the assignments below accomplish?

begin code
```
B = A - repmat(mean(A,2),[1 size(A,2)])
C = A - repmat(mean(A,1),[size(A,1) 1])
```
end code

7. There are several commands in Matlab that produce random numbers. Without needing to know anything about the properties of the numbers that are produced, these are useful commands to quickly generate matrices populated with arbitrary numbers, and can be very useful for experimentation of calculations and learning behavior of basic Matlab commands.

(a) For example, to test the claim that real, symmetric matrices have real eigenvalues, but real nonsymmetric matrices generally have complex eigenvalues, you can try this experiment using `rand`

begin code
```
A = rand(7,7);
isequal(A, A')
eig(A)
Asym = A + A';  % make symmetric by adding transpose
isequal(Asym, Asym')
eig(Asym)
```
end code

To test the imprecise claim that "square, randomly generated matrices are almost always invertible", try

begin code
```
A = rand(5,5);
shouldBeIdentity = A*inv(A)
norm(shouldBeIdentity-eye(5))
```
end code

To test the claim that "if $W \in \mathbb{R}^{n \times n}$ is invertible, then $W^T W \succ 0$", try

begin code
```
W = rand(5,5);
eig(W'*W)
```
end code

To test the claim that "sqrtm computes the unique positive-semidefinite symmetric matrix square root of a positive-semidefinite matrix," try

begin code

```
W = rand(5,5);
M = W'*W;
isequal(M, M')
S = sqrtm(M);
isequal(S, S')
eig(S)
S*S-M
```

end code

8. Two important commands are `rand` and `randn`.

   (a) Use `hist` to visualize the distribution of the numbers generated by `A = rand(1,500)`.

   (b) The command `randn` generates random numbers with a different distribrution than that provided by `rand`. Use `hist` to visualize the distribution of the numbers generated by `A = randn(1,500)`

   The properties of these random number generators are described by the distribution of the random variables that they produce, namely "uniform" for `rand` and "normal" (or ''gaussian'') for `randn`. In this class, we did not discuss the *distribution of a random variable (scalar or vector-valued)*. We only covered three properties: *mean*, *variance* and *covariance*. We talked about probability models with a countable number of outcomes, so these 3 properties were defined in terms of infinite sums over the sample space. In that limited context, we can interpret the outputs of `Xmat = rand(n, M)` and `Xmat = randn(n, M)` as follows:

   - Each column is associated with an outcome of the probability model, (Ω, P). There are M outcomes in this probability model, and the probabilities of each outcome are equal, each being $\frac{1}{M}$.
   - The value of a random variable X at the k'th outcome is the k'th column of `Xmat`. Since the columns are n-dimensional, the random variable X is vector-valued, namely $X : \Omega \to \mathbb{R}^n$.
   - Using `Xmat = rand(n, M);`, the mean of X will be approximately $0.5 \cdot \mathbb{1}_n$, and the variance will be approximately $\frac{1}{12} I_n$.
   - Using `Xmat = randn(n, M);`, the mean of X will be approximately $0 \cdot \mathbb{1}_n$, and the variance will be approximately I_n.

   Hence, in this interpretation, both commands create a probability model and random variables whose components are (approximately) uncorrelated, with (approximately) known means and variances.

   (c) Verify the claims above for $n = 3$, using ideas from problems (5) and (6).

   We know that affine transformations (problems (2) and (3)) change the mean and variance, hence `Ymat = sqrt(12)*(rand(n, M)-0.5)` corresponds to a random variable Y with mean approximately 0, and the variance approximately I_n.

(d) Run the code below, and then verify that the means and variances are approximately equal, as claimed

begin code

```
M = 500;
n = 1;
Xmat = sqrt(12)*(rand(n, M)-0.5);
Ymat = randn(n, M);
```

end code

Recall though, from parts (a) and (b), that the actual distributions are quite different.

(e) Run the code below, and then verify that the means and variance are approximately as claimed above

begin code

```
M = 500;
n = 3;
Xmat = sqrt(12)*(rand(n, M)-0.5);
```

end code

(f) Recall the results from (4) which show that if A is a matrix, and X is a vector-valued random variable with $\mu_X = 0$ and $\Sigma_X = I$, then $Y := AX$ will have $\mu_Y = 0$ and $\Sigma_Y = AA^T$. Explain how the following code generates a random variable Y (defined on a finite Sample Space with 1000 outcomes) such that

$$\mu_Y \approx 0_{3\times 1}, \quad \Sigma_Y \approx \begin{bmatrix} 3 & -1 & 1 \\ -1 & 2 & 0.5 \\ 1 & 0.5 & 1 \end{bmatrix}$$

Also, verify that the construction of Y actually achieves the goals.

begin code

```
n = 3;
M = 1000;
X = randn(n, M);
S = [3, -1, 1; -1, 2, 0.5; 1, 0.5, 1];
A = sqrtm(S);
Y = A*X;
```

end code

(g) Somewhat surprisingly (perhaps), rearranging the numbers from `rand` and `randn` still preserves the approximate properties discussed above

begin code

```
n = 4;
M = 1000;
Xmat = randn(n, M);
muX = mean(Xmat,2)
D = Xmat - repmat(muX,[1 M]);
D*D'/M
I = randperm(n*M);
```

```
Ymat = reshape(Xmat(I),[n M]);
muY = mean(Ymat,2)
D = Ymat - repmat(muY,[1 M]);
D*D'/M
```
end code

Task: Verify the same rearrangment property for data produced by `rand`. **Remark/Connection:** In this class, we **did not** discuss the notions of *independent events* in a probability model, and *independent random variables*. These important concepts are why arbitrary rearrangements performed above have little effect on the means and variances of the randome variables produced by `rand` and `randn`.

9. **Generating random variables for initial conditions and disturbances:** In this class, we began using random variables in the Kalman filtering module to represent a large collection of "scenarios" of initial conditions and disturbances for which the estimation scheme we design must work well, at least in an average sense. For use in simulation, we want to be able to quickly create a large collection of scenarios with appropriate properties (certain means and variances), and then be able to select one specific scenario, and simulate the system under those conditions (and possibly repeat this for some of the other scenarios).

   In order to create candidate random variables representing initial conditions and disturbances with certain means and variances, we can use `rand` and `randn`, and the rearrangment properties, and the affine transformations to do so easily.

   For example, take a system 5 states, and 3 disturbances, so $n_x = 5, n_d = 3$. In order to create a probability model of initial conditions and sequences of disturbances of length 100 (ie., for $k = 0, 1, \ldots, 99$), we can use `rand` and `randn`. Suppose we want a probability model with 5000 different outcomes, all with means of 0 and variance matrices equal to I.

```
M = 5000;  % outcomes in probability model
nX = 5;
nD = 3;
duration = 100;  % length of disturbance sequence
R = randn(nX+nD*duration, M);
```
end code

   The matrix R has all of the data representing the $M(= 5000)$ outcomes, namely

$$\left\{ \left.\begin{bmatrix} x_0 \\ d_0 \\ d_1 \\ \vdots \\ d_{99} \end{bmatrix}\right|_{@\omega_1} \quad \left.\begin{bmatrix} x_0 \\ d_0 \\ d_1 \\ \vdots \\ d_{99} \end{bmatrix}\right|_{@\omega_2} \quad \ldots \quad \left.\begin{bmatrix} x_0 \\ d_0 \\ d_1 \\ \vdots \\ d_{99} \end{bmatrix}\right|_{@\omega_M} \right\}$$

   By construction, we know that over this probability model has for all k and $k \neq j$

$$\mathbb{E}(x_0) \approx 0_{5\times 1}, \quad \mathbb{E}(d_k) \approx 0_{3\times 1}, \quad \mathbb{E}(d_k d_k^T) \approx I_3, \quad \mathbb{E}(d_k d_j^T) \approx 0_{3\times 3}$$

Each column of R has dimension 305×1, and is an outcome of the random variable

$$\begin{bmatrix} x_0 \\ d_0 \\ d_1 \\ \vdots \\ d_{99} \end{bmatrix}$$

For simulation studies, we can chose **any** such column, and use that in the simulation.

Because of the properties of `randn`, we can get the same results (in an average sense) by using `randn` to generate just the one single column we would use in a simulation. And, we can generate these in different rearrangements, perhaps more conducive to our needs, for example the single 305×1 column might be easier to manipulate if it was broken out into the initial condition vector, and the sequence of disturbances stored column-wise, as below.

begin code

```
x0vec = randn(nX, 1);
dSeq = randn(nD, duration);
```

end code

In this construction the `x0vec` vector is the initial condition and the columns of `dSeq` are the values of the disturbance $d_0, d_1, \ldots, d_{99}$ associated with one outcome.

This data represents a single outcome from an underlying probability model with and by construction, we know that over this probability model

$$\mathbb{E}(x_0) \approx 0_{5\times 1},$$

and for all k and all $j \neq k$

$$\mathbb{E}(d_k) \approx 0_{3\times 1}, \quad \mathbb{E}(d_k d_k^T) \approx I_3, \quad \mathbb{E}(d_k d_j^T) \approx 0_{3\times 3}$$

(a) We wish to simulate the system

$$x_{k+1} = Ax_k + Ed_k, \quad y_k = Cx_k + Fd_k$$

under random initial conditions and random disturbances with the following properties

$$\mathbb{E}(x_0) = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \quad \Sigma_{x_0} = \begin{bmatrix} 3 & -1 & 1 \\ -1 & 2 & 0.5 \\ 1 & 0.5 & 1 \end{bmatrix},$$

and for all $k \geq 0$, and $j \neq k$

$$\mathbb{E}(d_k) = 0, \quad \Sigma_{x_0,d_k} = 0, \quad \Sigma_{d_k,d_j} = 0, \quad \Sigma_{d_k} \approx \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 1 \end{bmatrix}$$

with the following state-space data

$$A = \begin{bmatrix} 0.8 & -0.6 & 0.15 \\ 1.5 & 0.02 & 1.3 \\ -0.26 & -0.16 & -1 \end{bmatrix}, \quad E = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \end{bmatrix}, \quad F = \begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix}$$

Write code that (via `for` loop) picks 8 outcomes from a probability model (for x_0 and d) with those properties, and simulates the system from $k = 0$ to $k = 30$. The code should plot (on a single plot, all 8 outcomes) the second component of y, $y_k^{[2]}$ versus k. Please document the code so that we can see how you are getting (in an average sense) the appropriate mean and variance for x_0 and d.