State Generator for Kalman Filter's Input

by Khoa Do

Abstract: One of the outcomes of the Spring 2023 ECE 751 (Detection and Estimation Theory) is the ability to develop and prototype a Kalman filter or so-called the software project. For this part I of the software project, I developed a state-space signal model (the state generator), generating both "true" state vector values and observations (i.e., outputs) as an input to the discrete time-invariant Kalman filter using MATLAB. The state generator took a set of generalized parameters from the user and produced the aforementioned outputs. To benchmark my state generator, I modeled example 9.10 in the textbook *Detection, Estimation, and Modulation Theory Part 1 – Detection, Estimation, and Filtering Theory* [1] and plotting the results to compare to those illustrated in figure 9.38 from the textbook. It was shown that my results were similar to ones of the textbook.

1. Background

1.1 The project

One of many learning goals in my ECE 751 course is to develop and prototype a Kalman filter. Kalman filter is one of the most popular estimating techniques. It has a wide range of applications and is widely used in various areas such as network communication, signal processing, and robotics. A perfect system will produce a perfect signal without any noises for example, a straight-line signal (original line or the ground truth) if plotted in MATLAB. However, signals in reality will always have noises, which cause the original straight line to be jiggling and possibly curvy (noisy ground truth). The purpose of the Kalman filter is to produce (estimate) a line that is as close to the ground truth as possible based on the noisy ground truth. Being said, the filter takes the noisy signal as its input and produce the estimate as its output. The purpose of this project part I is to generate the input to the filter using a state generator. This input includes the "true" values and the noisy values. The state generator is a state-space model that can represent any dynamic system. It generates states and observations and is generally described in the following block diagram

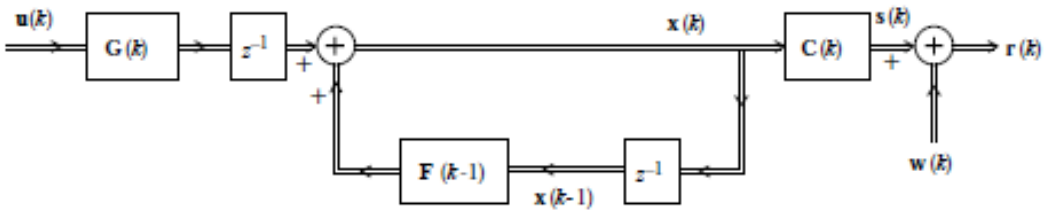

Figure 1. State generation and observation model [1].

where **F**, **G**, **C**, **x(k)**, **x(k-1)**, **u(k)**, **w(k)**, **r(k)**, and **z** are state transition matrix, input (control) transition matrix, current state, previous state, process (plant) noise, measurement noise, observation, and delay respectively.

The state generator takes F, G, C, some initial x(k), u, and w as inputs and produce observation z. These inputs vary among different systems in terms of numerical values, matrix size, and elements in the matrix. One of the requirements of the project is to generalize F, G, C, and initial x(k) in terms of matrix size, meaning the generator should work under any size of these matrices. The job of the user is to just

"enter" their matrices and their according element values especially for F, G because the explicit formulas to calculate [element] values of these matrices are complicated. Then to benchmark my generator, I was required to use values from example 9.10 in [1], which models a one-dimensional motion with roughly constant velocity, plot the results, and compare the plots to those shown in figure 9.38 of [1].

1.2 Key equations

I used equation 9.212 and 9.213 from [1] as the two key equations to implement my state generator.

$$\mathbf{x}(k) = \mathbf{F}(k-1)\mathbf{x}(k-1) + \mathbf{G}(k-1)\mathbf{u}(k-1), \qquad k = 1, \ldots, \tag{9.212}$$

$$\begin{aligned} \mathbf{r}(k) &= \mathbf{C}(k)\mathbf{x}(k) + \mathbf{w}(k) \qquad k = 1, 2, \ldots, \\ &= \mathbf{s}(k) + \mathbf{w}(k), \end{aligned} \tag{9.213}$$

X, u, and k are assumed to be normally distributed with some mean and variance for the Kalman filter. Figure I as well as equation 9.212 and 9.213 were later translated into MATLAB implementation and detailed explanations in section 2.

2. Design

2.1 High-level interface

The interface consists of three parts: the inputs from the user, the state generator function, and visualization. The inputs from the user were F, G, C, u, w, initialized x, and k (which is the number of samples wanted to be generated. Then, the state generator function was called with these inputs as parameters and started producing both the "true" state vector values and observations using equation 9.212 and 9.213. As mentioned in section 1.1, these inputs vary from system to system in terms of values, matrix size, and element values of the matrices, so the generator has to be generalized to them. In other words, the implementation of equations shown in section 1.2 must work at all time regardless of changes in these inputs especially the matrix size of F, G, and C. There are respectively explicit formulas to formulate elements inside these matrices depending on the dynamic system model, but this was not required for this project. Instead, it was required to generalize to different matrices' size. Once "true" state vector values and observations were obtained, they were plotted for visualization and analysis. C matrix is the simplest to be explicitly formulated because it is just an identity matrix.

2.2 MATLAB implementation

I created two files *state_generator_input.m* and *state_generator.m*. The first file was intended to hold only inputs from the user and the state generating function call, then the second file would contain the implementations of the state generator and the visualization. Simply put, the user would specify the inputs and the function call in one file and the other file would do the rest.

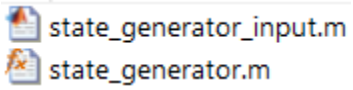

Figure 2. User's inputs file and state generator file.

However, due to the unique function call in MATLAB, I had to place the visualization implementation in the same file as the user's input file.

2.2.1 state_generator_input.m

```
clear all
close all

%----------------------------setting up variables, inputs from user----------------------------%
T = 0.1;                        % sampling interval, value 9.396 from book
N = 100;                        % no. of iterations, value from user (100 so that plots on same scale as in book fig 9.38)

sigma2_a = 40;                  % process (plant) noise's variance, value 9.395 from book
sigma2_r = 100;                 % measurement noise's variance, value 9.397 from book

F = [1 T; 0 1];                 % state transition matrix, equation 9.386 from book
G = [T^2/2; T];                 % input (control) transition matrix, equation 9.387 from book
C = [1 0];                      % output transition matrix, equation 9.390 from book

x_prev = [1000; -50];           % initialize [position(0); velocity(0)], values 9.393 & 9.394 from book
%----------------------------end----------------------------%


%----------------------------initilize variables for plotting----------------------------%
x_true = zeros(size(F,1), N);           % initialize matrix F_row-by-N to hold hold "grounth-truth" state vector values of postion and velocity with noise
x_noisy = zeros(size(G,2), N);          % initialize matrix G_column-by-N to hold position measurements (with noise)
%----------------------------end----------------------------%


%----------------------------call function to generate state values----------------------------%
[x_true, x_noisy] = state_generator(N, sigma2_a, sigma2_r, F, G, C, x_prev);     % return values for plotting
%----------------------------end----------------------------%


%----------------------------plotting----------------------------%
Nt = [1:N]'*T;                                  % steps
figure                                          % 1st figure
plot(Nt, x_true(1, :), Nt, x_noisy, ':k')       % for position - plot all values for "true" with noise and measurement with noise
legend('True', 'Noisy')
xlabel('t (s)')
ylabel('Position (m)')
axis([0, 10, 400, 1000])                        % scale to look like figure 9.38 in book
grid

figure                                          % 2nd figure
plot(Nt, x_true(2, :))                          % for velocity, plot all values for "true" with noise
legend('True')
xlabel('t (s)')
ylabel('Velocity (m/s)')
axis([0, 10, -100, 200])                        % scale to look like figure 9.38 in book
grid
%----------------------------end----------------------------%
```

Figure 3. state_generator_input.m.

Figure 3 shows my MATLAB implementation of the *state_generator_input.m* with detailed comments. The user specified the inputs from line 5 to 15. Line 8 and 9 were variances of u and k noises which I used a different variable in the second file. These noises are normally distributed with mean of 0 and known variance. Line 20-21 helped the state generating function generalize to any size of F, G, and C. In general, the inputs from the user tended to work out on their own because he/she had to known these in advance and that they were correct in size and element values before specifying in this file. Line 26 called the function to generate "true" values and observations from the *state_generator.m*. The function returned the state vector N length-N_x state vector *x_true* and N length-N_r measurements (observations) *x_noisy*. Line 31-46 implied the implementation for visualization (plotting).

2.2.2 state_generator.m

```
1  function [x_true, x_noisy] = state_generator(N, sigma2_a, sigma2_r, F, G, C, x_prev);
2
3      %----------------------------setting up variables-------------------------%
4      seed = 1040;                    % 512(o)*2 + 16
5      rng(seed, 'twister');
6
7      Q = sigma2_a;                   % process noise u(k-1) ~ N(0,Q) from book page 881
8      W_m = sigma2_r;                 % measurement noise, consider position only, it's a matrix 9.392 fro both position and velocity
9
10     %-----------------------------end--------------------------%
11
12
13     %----------------------------for loop to generate "true" and noisy measurements---------------------------%
14     for n = 1:N
15         Q_noise = normrnd(0, sqrt(Q));              % generate normal distributed process noise
16         Wm_noise = normrnd(0, sqrt(W_m));           % generate normal distributed meaurement noise
17
18         x_current = F * x_prev + G * Q_noise;       % current "true" state vector values + noise, equation 9.212 from book
19         r = C * x_current + Wm_noise;               % current measured value of postion + noise, equation 9.213
20
21         x_true(:, n) = x_current;                   % assign to x_input for plotting
22         x_noisy(n) = r;                             % assign to x_noisy for plotting
23         x_prev = x_current;                         % update state value
24     end
25     %-----------------------------end--------------------------%
```

Figure 4. state_generator.m.

Once the state generator function was called, the *state_genrator.m*. was executed. Line 7-8 justified my use of different names for u and w shown figure 1. The process (plant) noise and measurement noise were randomly generated following normal distribution with mean of 0 and known variance from the user each time the for loop iterated (line 4-16). Line 18-19 were direct translations of equation 9.212 and 9.213 to MATLAB. All "true" state vector values and observations were store in *x_true* and and *x_noisy* respectively to be returned to *state_genrator_input.m* for plotting.

3. Extra credit (optional)

I did not attempt for the extra credit this time.

4. Benchmark

To benchmark my MATLAB implementations, I ran them using example 9.10 form [1], which was a one-dimensional motion model with roughly constant velocity. The example provided a set of inputs which was shown from line 5 to 15 in figure 3. For this example, *x_true* was a 2xN-length state vector containing 100 "true" values for position in the first-row dimension and 100 "true" values for velocity in the second-row dimension while *x_noisy* was a 1xN-length observation vector containing 100 values for noisy position. Upon completion, I obtained two plots shown in figure 5 and 6.

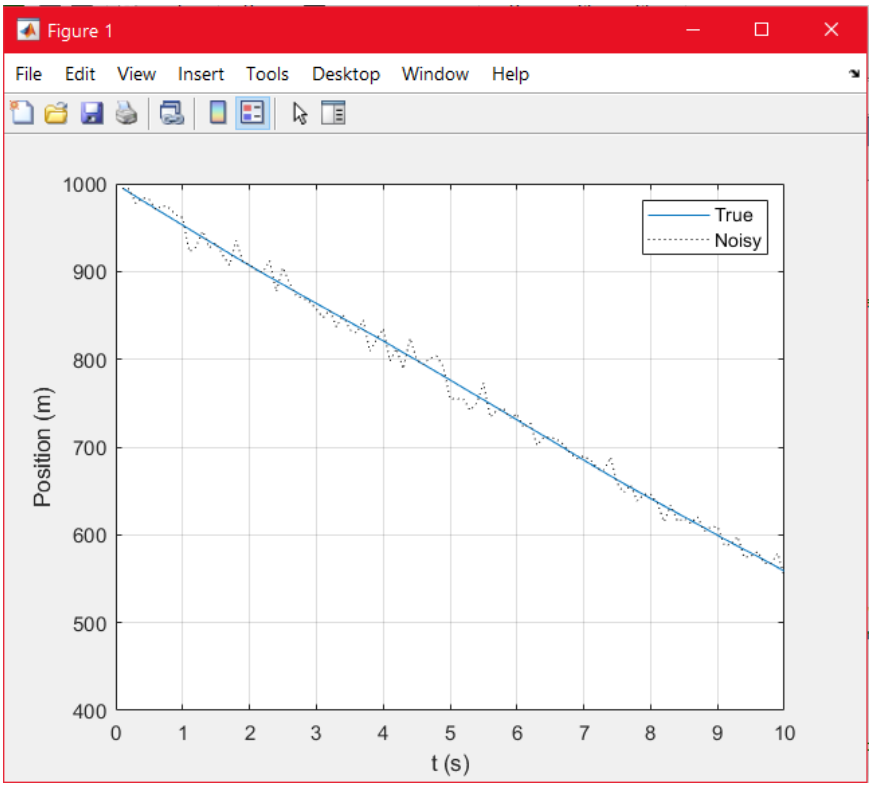

Figure 5. "True" and noisy position vs. time plot.

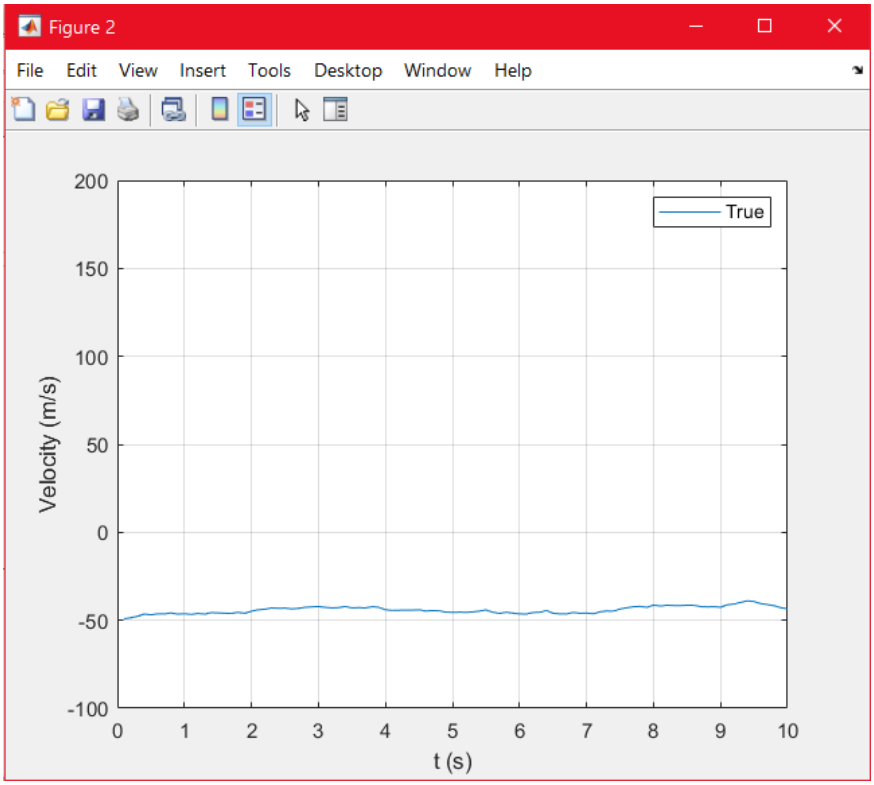

Figure 6. "True" velocity vs. time plot.

I scaled the plots to the same scales as those shown in figure 9.38 of [1]. The result showed that my plots looked similar to those in figure 9.38 of [1] (or figure 7 in this report).

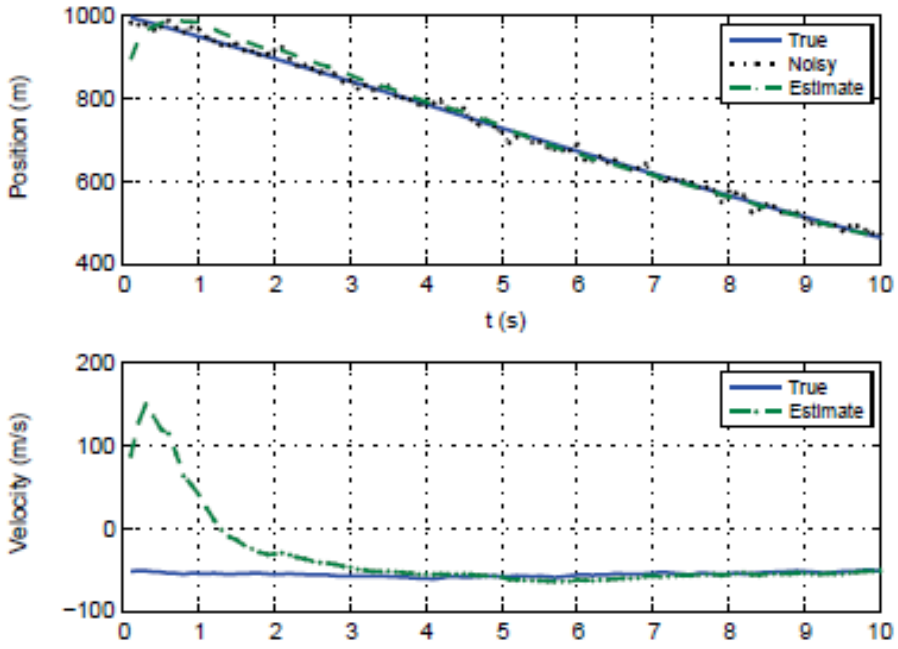

Figure 7. Figure 9.38 in [1].

I also compared my state values as well as observation(s) after the for loop executed for one iteration to the theoretical state value shown in [1]. The results were similar, which indicated that my MATLAB implementations were correct.

5. Conclusion

In this software project part I, I developed a state-space signal model (the state generator), generating both "true" state vector values and observations (i.e., outputs) as an input to the discrete time-invariant Kalman filter using MATLAB. The state generator took a set of generalized parameters from the user and produced the aforementioned outputs. To benchmark my state generator, I modeled example 9.10 in the textbook *Detection, Estimation, and Modulation Theory Part 1 – Detection, Estimation, and Filtering Theory* [1] and plotting the results to compare to those illustrated in figure 9.38 from the textbook. It was shown that my results were similar to ones of the textbook, which indicated that my model implementation was correct.

6. References

[1] V. T. H. L., K. L. Bell, and Z. Tian, Detection estimation and modulation theory. Oxford: Wiley-Blackwell, 2013.

7. Appendix

7.1 state_generator_input.m

```m
clear all
close all

%----------------------------setting up variables, inputs from user----------------------------%
T = 0.1;                        % sampling interval, value 9.396 from book
N = 100;                        % no. of iterations, value from user (100 so that plots on same scale as in book fig 9.38)

sigma2_a = 40;                  % process (plant) noise's variance, value 9.395 from book
sigma2_r = 100;                 % measurement noise's variance, value 9.397 from book

F = [1 T; 0 1];                 % state transition matrix, equation 9.386 from book
G = [T^2/2; T];                 % input (control) transition matrix, equation 9.387 from book
C = [1 0];                      % output transition matrix, equation 9.390 from book

x_prev = [1000; -50];           % initialize [position(0); velocity(0)], values 9.393 & 9.394 from book
%----------------------------end----------------------------%


%----------------------------initilize variables for plotting----------------------------%
x_true = zeros(size(F,1), N);           % initialize matrix F_row-by-N to hold hold "grounth-truth" state vector values of postion and velocity with noise
x_noisy = zeros(size(G,2), N);          % initialize matrix G_column-by-N to hold position measurements (with noise)
%----------------------------end----------------------------%


%----------------------------call function to generate state values----------------------------%
[x_true, x_noisy] = state_generator(N, sigma2_a, sigma2_r, F, G, C, x_prev);     % return values for plotting
%----------------------------end----------------------------%


%----------------------------plotting----------------------------%
Nt = [1:N]'*T;                                  % steps
figure                                          % 1st figure
plot(Nt, x_true(1, :), Nt, x_noisy, ':k')       % for position - plot all values for "true" with noise and measurement with noise
legend('True', 'Noisy')
xlabel('t (s)')
ylabel('Position (m)')
axis([0, 10, 400, 1000])                        % scale to look like figure 9.38 in book
grid

figure                                          % 2nd figure
plot(Nt, x_true(2, :))                          % for velocity, plot all values for "true" with noise
legend('True')
xlabel('t (s)')
ylabel('Velocity (m/s)')
axis([0, 10, -100, 200])                        % scale to look like figure 9.38 in book
grid
%----------------------------end----------------------------%
```

7.2 state_generator.m

```m
function [x_true, x_noisy] = state_generator(N, sigma2_a, sigma2_r, F, G, C, x_prev);

    %----------------------------setting up variables----------------------------%
    seed = 1040;                    % 512(o)*2 + 16
    rng(seed, 'twister');

    Q = sigma2_a;                   % process noise u(k-1) ~ N(0,Q) from book page 881
    W_m = sigma2_r;                 % measurement noise, consider position only, it's a matrix 9.392 fro both position and velocity

    %----------------------------end----------------------------%


    %----------------------------for loop to generate "true" and noisy measurements----------------------------%
    for n = 1:N
        Q_noise = normrnd(0, sqrt(Q));              % generate normal distributed process noise
        Wm_noise = normrnd(0, sqrt(W_m));           % generate normal distributed meaurement noise

        x_current = F * x_prev + G * Q_noise;       % current "true" state vector values + noise, equation 9.212 from book
        r = C * x_current + Wm_noise;               % current measured value of postion + noise, equation 9.213

        x_true(:, n) = x_current;                   % assign to x_input for plotting
        x_noisy(n) = r;                             % assign to x_noisy for plotting
        x_prev = x_current;                         % update state value
    end
    %----------------------------end----------------------------%
```