M.E. 530.647 Lab 0
Review of Simple Linear State Feedback Control

Louis L. Whitcomb*
Department of Mechanical Engineering
G.W.C. Whiting School of Engineering
Johns Hopkins University
https://dscl.lcsr.jhu.edu/courses/530-647-adaptive-systems-fall-2017

Read the Matlab documentation on the built-in functions `ode45()` and `plot()`.

You can install the full version Matlab on your personal computer under JHU's site license. To obtain Matlab, from an on-campus internet connection log in to my.jhu.edu, and under "Technology" select "mySoftware", and follow the links to the JHU Software Catalog. Once there, serach for "Matlab for Students", and follow the instruction on how to download and install Matlab.

1. **A Simple Second Order Plant:** Write a matlab function `BALLDYN(t,x)` which accepts the scalar real argument `t` and the 2×1 vector $\texttt{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$, and returns a *result* value of

$$\texttt{result} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \texttt{x(t)} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \texttt{u(t)} \tag{1}$$

   Use this function to compute numerical solutions to the equation

$$\dot{x}(t) = Ax(t) + Bu(t); \quad A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}; \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}; \quad x(0) = \begin{bmatrix} x_{01} \\ x_{02} \end{bmatrix}. \tag{2}$$

   which you will recognize as the equation of motion for a 1 Kg mass under the external force $u(t)$. Choose a non-zero $u(t)$ such as $u(t) = 10.0\, cos(2\pi t)$.

2. **Simulating Open Loop Response of a Simple Second Order Plant:** Compute a numerical solution to the solution to the ODE defined by `BALLDYN(t,x)` using `ODE45( )`. Plot, print, annotate, and hand-in solutions for two initial conditions.

3. **State Feedback Control:** Write a matlab function `BALLFBCON(t,x)` which accepts the scalar real argument `t` and the 2×1 vector $\texttt{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$, and returns a *result* value of

$$\texttt{result} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \texttt{x(t)} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \texttt{u(t)}; \; \textit{where} \quad u(t) = k_p x(t) + k_d \dot{x}(t). \tag{3}$$

   Use this function to compute numerical solutions to the plant equation (2) which you will recognize as the equation of motion for a 1 Kg mass under the closed-loop proportional-derivative (PD) control law $u(t) = k_p x(t) + k_d \dot{x}(t)$. Choose k_p and k_d so that the closed-loop poles of the system are {-2,-2}.

Note that (2) can be rewritten in classical state-space form as a plant

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{4}$$

with a controller

$$u(t) = Fx(t) \tag{5}$$

and the closed-loop system of plant and controller can be written as

$$\dot{x}(t) = (A + BF)x(t). \tag{6}$$

In the case above, A and B are given. What are the numerical values for F in this case?

4. **Simulating State Feedback Control:** Simulate, plot, print, and annotate solutions for two nonzero initial conditions of the above system. Be sure to label and annotate your plot.

5. **State Observers:** Suppose you have a position encoder which continuously measures the position of the ball (the mass), but *does not* measure the ball velocity. You wish to know both position *and* velocity of the ball. The encoder returns the measured value

$$\begin{array}{ll} y(t) & = Cx(t) \quad C = [1 \quad 0]. \\ & = x_1(t) \end{array} \tag{7}$$

Recall the design of a Lunenberger observer for the the above system of the form

$$\dot{\hat{x}}(t) = A\hat{x}(t) + Bu(t) + L(\hat{y} - y); \quad \hat{x}(t) = \begin{bmatrix} \hat{x}_1(t) \\ \hat{x}_2(t) \end{bmatrix}. \tag{8}$$

Design the poles of your observer to be $\{-1, -1\}$.

Write a function `OBSDYN(t,x)` which accepts the scalar real argument `t` and the 4×1 vector

$$\mathtt{X} = \begin{bmatrix} x_1 \\ x_2 \\ \hat{x}_1 \\ \hat{x}_2 \end{bmatrix} \tag{9}$$

and returns a *result* value of

$$\mathtt{X} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{\hat{x}}_1 \\ \dot{\hat{x}}_2 \end{bmatrix} = \begin{bmatrix} (A + BF)x(t) \\ A\hat{x}(t) + BFx(t) + L(\hat{y} - y) \end{bmatrix}. \tag{10}$$

6. **Simulating State Observers:** Simulate, plot, print, and annotate solutions for two nonzero initial conditions of the above system. Be sure to label and annotate your plot.

7. **Observer Feedback:** Equation (6) is the equation of motion for the closed loop system when the controller $u(t)$ uses the *actual* plant state $x(t)$. When implementing a real controller, however, the control system may not have full state access, and the controller must use the observer state $\hat{x}(t)$ instead of the actual state $x(t)$.

$$\dot{x}(t) = Ax(t) + Bu(t); \quad A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}; \quad u(t) = F\hat{x}(t)x(0) = \begin{bmatrix} x_{01} \\ x_{02} \end{bmatrix}. \tag{11}$$

Note that (11) can be rewritten in classical state-space form as a plant

$$\dot{x}(t) = Ax(t) + B\vec{\hat{u}}(t) \tag{12}$$

with the controller

$$u(t) = F\hat{x}(t). \tag{13}$$

Write a function `OBSCON(t,x)` to simulate the closed-loop performance of the full controller/observer system. Your function `OBSCON(t,x)` should accept the scalar real argument `t` and the 4×1 vector

$$\texttt{X} = \begin{bmatrix} x_1 \\ x_2 \\ \hat{x}_1 \\ \hat{x}_2 \end{bmatrix} \tag{14}$$

and returns a *result* value of

$$\texttt{result} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{\hat{x}}_1 \\ \dot{\hat{x}}_2 \end{bmatrix} = \begin{bmatrix} Ax(t) + BF\hat{x}(t) \\ A\hat{x}(t) + BF\hat{x}(t) + L(\hat{y} - y) \end{bmatrix}. \tag{15}$$

Design your obeserver gain matrix L such that poles are $\{-1, -1\}$.

8. **Simulating Closed-Loop Controller-Observer Systems:** Simulate, plot, print, and annotate solutions for two nonzero initial conditions of the above system. Be sure to label and annotate your plot.

Hand in *ANNOTATED* printouts of your plots and printouts of your m-files. Hand in your m-file functions on a floppy disk or email them to me as ZIP file attached to your email. Put "530.647 LAB #0 m-files from YOUR FULL NAME" in the subject line.

Check to verify that the files you hand in run. If your matlab functions call custom matlab m-files that you have written (for this course or otherwise) be sure to include *all* necessary files.