Lab 3 – ECE410F Linear Control Systems
State Feedback Stabilization of a Cart-Pendulum Robot

Main concepts of this lab

- Design of state feedback stabilizers for LTI systems using eigenvalue assignment
- Formulation of the linear quadratic optimal control problem.
- Tuning LQR controllers.
- Effects of plant nonlinearity.

1 Introduction

In the first lab of this course, you worked on several techniques to model and analyse a nonlinear control system, and then to find a suitable linearization. In the second lab, we covered several basic techniques of numerical linear algebra, building up to the fundamental tools necessary to work with linear systems in state space form. In this lab we will build on this foundation, analysing a more complicated nonlinear system, the classic *cart-pendulum* model, and then use some new tools to develop several controllers for this system.

Throughout the lab, you will be guided through a number of steps which will require you to write Matlab code. You will write your code in a Matlab script called `labx.m`, where `x` is the lab number. You will submit this code as a group. Your code should provide certain outputs (figures, numbers, and the like). A request for output is highlighted with a shaded area of text, such as the following.

Output 1. Print the eigenvalues of the matrix.

Parts of the text containing directions for the writing of your Matlab code will be highlighted in a different colour, such as this:

Create a function `pendulum.m`. The first line of the function will be

```
function xdot = pendulum(t,x,parameters)
```

2 Submission guidelines and mark breakdown

Marks are assigned to groups. Members of each group get identical marks, unless special circumstances occur. The group lab mark will be made up of three components.

Matlab code	6 pts
Presentation video	2 pts
Lab report	2 pts
Total	**10 pts**

Matlab code. Your code should be clean and readable, and it should contain abundant commentary, so that the instructors may follow your development and check its correctness. This component of the mark will be based on the correctness of the code and its readability, and it will be assigned as follows:

0 out of 6	the code is absent
1 out of 6	the code is largely incomplete
2 out of 6	the code is largely complete, but there are parts missing and the outputs are incorrect
3 out of 6	the code is complete, but it does not produce correct outputs
4 out of 6	the code is complete, it produces correct outputs, but there is no commentary in the code, and/or the code is poorly organized
5 out of 6	the code is complete, correct, and contains some but insufficient commentary, and/or its organization is below par
6 out of 6	the code is complete, correct, and the commentary and organzation are adequate so that it is easy to read

Presentation video. Once you've completed your lab code, you will submit a 5-6 minute video presentation of the code as a group. Each group member will present a different portion of the code. The objective of the presentation is to show that you understand what you did and why you did it. It's important that each group member contributes equally to the video. In the video, include any observations about the outputs you produced, any insight you derived from the lab steps.

0 out of 2	group does not submit a presentation, or the presentation is unintelligible
1 out of 2	the presentation is somewhat intelligible but does not display adequate understanding of the code and/or the lab document, or the group members give somewhat disconnected presentations
2 out of 2	the presentation is intelligible and displays an adequate understanding of the code and the lab document. The group members contribute equally to the presentation and their contributions are well-connected

Lab report. Write a concise report containing the requested outputs, but don't just print out a bunch of matrices and figures. Add some flesh to the outpus that are requested within the lab document so that one can follow the logical development of the lab. Aim for a style like this:

> **Output 1.** The following are the bases of the subspaces $\mathcal{V}$ and $\mathcal{W}$ that were obtained:
> (...)
> We observe that the columns of V were already linearly independent, while those of $\mathcal{W}$ weren't. We further note that (...).
>
> **Output 2.** Below the solution of the differential equation. There are three figures. The first two figures depict $x_1(t)$ and $x_2(t)$, while the third figure depicts the orbit.

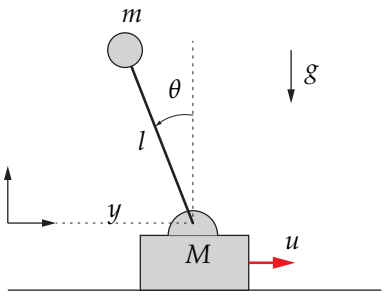
Do not screenshot Matlab figures. Rather, save them as jpeg files (or other formats) and include then in the report in the appropriate order.

When the lab document asks you to comment on something, strive to provide meaningful, insightful commentary. This portion of the mark will be assigned as follows:

0 out of 2	the report is either absent, or a simple printout of the Matlab outputs without commentary
1 out of 2	the report is incomplete and/or the commentary is inadequate
2 out of 2	the report is complete and the commentary is adequate

3 Analysing the cart-pendulum model

The cart-pendulum is a classic testbed for control techniques. Our goal in this lab will be to design linear controllers stabilizing the upright equilibrium of the system using two different methods, and briefly explore some of the problems that can arise using linear control design for a fundamentally nonlinear system.



The full nonlinear equations of motions for this system were derived using the Euler-Lagrange equations in Section 5.6.5 of the textbook. For convenience, we present them here.

$$\ddot{y} = \frac{-ml\sin(\theta)\dot{\theta}^2 + mg\sin(\theta)\cos(\theta) + u}{M + m\sin^2(\theta)} \tag{1}$$

$$\ddot{\theta} = \frac{-ml\sin(\theta)\cos(\theta)\dot{\theta}^2 + (M+m)g\sin(\theta) + u\cos(\theta)}{l(M + m\sin^2(\theta))} \tag{2}$$

Note that, like in Lab 1, we have a second order differential equation, and so the state variables of this system are $y, \dot{y}, \theta, \dot{\theta}$. Define the state vector

$$x = \begin{bmatrix} y \\ \dot{y} \\ \theta \\ \dot{\theta} \end{bmatrix}.$$

We will use this ordering of states consistently through this lab.

Our first task will be to build a suitable linear model of the system.

Create a file `lab3.m`, and in it do the following

- Using the tools you learned about in Lab 1, define the above control system symbolically in Matlab. Declare symbolic variables for each of the state variables, the input u, and the parameters M, m, l, and g.
- Create a structure named `parameters` containing the following four pendulum parameters:
  - $M = 1.0731$

- $m = 0.2300$
- $l = 0.3302$
- $g = 9.8$

- Linearize the system about the upright $(0, 0, 0, 0)$ equilibrium.
- Substitute the above parameters into the A and B matrices from the linearization.

Output 1. Print the following quantities:

- The symbolic forms of the A and B matrices

With the preliminaries out of the way, we can proceed to control design.

4 Controllability and pole assignment

MATLAB commands

`place(A,B,p)`	Returns state feedback gains K for the pair (A, B) such that the closed-loop system with controller $u = -Kx$ has eigenvalues at the locations specified in p
`acker(A,B,p)`	As above, but using a different algorithm; valid only for single-input systems

In class, you learned the key result that the pole assignment problem is solvable for a linear system if and only if the system is controllable. In this section, we will design controllers by choosing the desired closed-loop poles. First, we will confirm that the system is controllable.

Continue in the same file `lab3.m`.

- Construct the controllability matrix for the linearized system.
- Confirm that the system is controllable.

Matlab provides two functions for pole placement. The first, `acker`, is only valid for single-input systems, and uses the Ackerman formula described in Section 6.2 of the textbook. This method, however, is not numerically stable. For almost all use cases, the `place` function should be preferred instead, since the algorithm used is much more robust. The `acker` function is occasionally useful for small problems when you want to place two or more poles at the same location, since the `place` function requires you to perturb the poles to be distinct in this case.

In `acker` and `place`, Matlab adopts the convention that the state feedback controller has the form $u = -Kx$. In class and in the textbook, we adopt the convention that the controller has the form $u = Kx$ so you will need

to take that into account when using Matlab. As a sanity check, each time you develop a controller using pole assignment, use the command `eig` to verify that the eigenvalues of the closed-loop system are indeed the ones you wished to assign.

Continue in the same file `lab3.m`.

- Using the Matlab `place` function, generate a gain vector K_1 placing the poles at $\{-1, -2, -3, -4\}$
- Simulate the linearized system using this feedback, with the initial condition $(-0.5, 0, -\pi/4, 0)$. To do that, you could create a file containing a Matlab function to be used in `ode45`, as we did in lab 1. Given the simplicity of an LTI system, another way to do it is to use function handles. Say you want to simulate a closed-loop equation $\dot{x} = A_{\text{cl}}x$, and that you have defined a 4×4 matrix `Acl`. The command `sys = @(t,x) Acl*x` defines a function `sys` that you can use in `ode45`, without having to save it to a file. If you use this method, in the `ode45` call, use `sys`, not `@sys`.
- Create a figure containing five subplots (see the lab 1 document to see how to do this). On these subplots, plot the four states of the system and the control signal $u(t) = -Kx(t)$ as functions of time.

  Note that you will need to compute $u(t) = Kx(t)$ *after* you have integrated the differential equation of the closed-loop system to obtain $x(t)$. Recall that Matlab returns the solution in a matrix $N \times 4$, where N is the number of samples. The vector K is 1×4. You need to produce a control signal in the form of an $N \times 1$ vector.
- Now you will investigate the effects of shifting *one* eigenvalue further to the left on the complex plane. The objective is to understand what are the repercussions of this change.

  Again using the Matlab command `place`, generate a gain vector K_2 that places the poles at $\{-1, -2, -3, -20\}$.
- Again simulate the system, with the same initial condition as before. In the figure you generated earlier, overlay the new plots of the four states and of the control signal versus time. This can be achieved using the command `hold on` in each subplot.

Output 2.
- Print the gain vectors K_1, K_2
- Produce a single figure containing *five* subplots as described above. Make sure to include a legend to distinguish signals pertaining to the two different control gains.
- Describe the differences between the transient response for the two sets of gains. How is the change in one pole reflected across the different state variables? Is a single state affected, or are all states affected?
- Comment on this: suppose you are satisfied with the convergence to zero of the pendulum angle, but the cart position converges to zero too slowly. Using pole assignment, do you think it would be possible to only speed up the cart convergence to zero? Explain why yes, or why not.

5 Linear quadratic optimal control

Matlab commands

`lqr(sys,Q,R)`	Returns the optimal state feedback gains K of a controller $u = -Kx$ minimizing a quadratic cost with matrices Q and R

In the previous section, we designed our gains by choosing the locations of the closed-loop eigenvalues. Simply placing the eigenvalues, however, does not let us easily tune the responses of particular state variables, or to take into consideration actuator limitations. In this section, we will explore a different design methodology, the *linear quadratic regulator* (LQR). An LQR controller is a state feedback[1] $u = Kx$ minimizing the cost function

$$J = \int_0^\infty x^\top Q x + u^\top R u \, dt$$

for a positive semidefinite matrix Q and positive definite matrix R.

Since our system has a single input, R is simply a scalar. We will use a Q matrix with the following structure:

$$Q = \begin{bmatrix} q_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & q_2 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

so that our cost function reduces to

$$J = \int_0^\infty q_1 x_1^2 + q_2 x_3^2 + R u^2 dt.$$

In other words, the cost is a weighted sum of the energies of the signals $x_1(t)$ (the cart position), $x_3(t)$ (the pendulum angle), and $u(t)$ (the control). By increasing one of the parameters (q_1, q_2, R) relatively to the other two, the associated signal is given greater weight in the cost function, and the optimal controller makes the energy of that signal relatively smaller. This basic observation allows us to prioritize certain variables over others, in a way that is not easy to do with pole assignment. There are limits of course, and in particular our simple choice of Q eliminates degrees of freedom that might be helpful in tuning the transient performance.

It is important to note that, although we have three parameters to adjust in our experiments, we do not truly have three degrees of freedom. For if we scale all parameters by a constant factor, the minimum of the cost J will be achieved with the same controller. Keep this in mind as you make adjustments to your controller.

Continue in the same file `lab3.m`.

- Conduct an experiment to show the impact of changing q_1. Keeping $q_2 = 5$, $R = 0.5$ fixed, design two sets of gains with $q_1 = 0.1$ and $q_1 = 0.005$. Simulate the linearized system with these parameters, and compare the transient response in the same way as the previous section, by preparing five subplots in one figure, one for each of the state variables and the control input.
- Conduct an identical experiment, now for q_2. Keep $q_1 = 0.05$, $R = 0.5$, and compare the results with $q_2 = 1$ and $q_2 = 2000$.
- Conduct a final experiment for R. Keep $q_1 = 0.05$, $q_2 = 5$, and compare between $R = 0.005$ and $R = 10$.

[1] Once again, Matlab adopts the convention that the state feedback has the form $u = Kx$.

Output 3.
- Produce the three figures requested above, each consisting of five plots.
- Comment on the results. How does changing R compare to changing q_1 or q_2? What difference is there between changing q_1 and q_2?

6 Nonlinear comparison

We will now briefly explore the performance of these controllers on the original nonlinear system.

Continue in the same file `lab3.m`.

- Using the gains corresponding to the $R = 0.005$ set of parameters from the previous section, define a function expressing the closed-loop dynamics of the nonlinear system. Use the Matlab `subs` function to set $u = Kx$, and then the `matlabFunction` command to get a function to use with `ode45` (see Lab 1 for details).
- Simulate the nonlinear system using the feedback you selected, using the initial condition $x_0 = (-1, 0, \pi/4, 0)$. Plot the response of the linear system versus the nonlinear system, using the same kind of plots as in the previous sections.
- Decrease the initial y value by 1. This time simulate just the nonlinear system using this feedback.
- Continue to decrease the initial y until the nonlinear system fails to converge to the equilibrium.

Output 4.
- Produce the comparison plots requested above.
- Comment on how the behaviour of the nonlinear system differs from the linear system.
- Produce a single figure showing the results of the nonlinear simulations as the initial condition changes. What significant differences appear for initial conditions farther from equilibrium?