Lab 1 – ECE410F Linear Control Systems

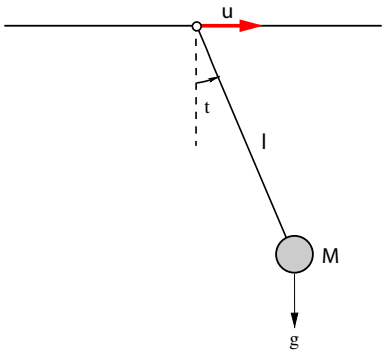
Numerical Simulation of Dynamical Systems and symbolic linearization

Main concepts of this lab

- How to numerically integrate a nonlinear (time-varying) ODEs.
- How to plot solutions of dynamical systems.
- How to symbolically linearize a nonlinear ODE.
- How to transition from a symbolic vector field to numerical simulation.
- How to define state space and transfer function representations of LTI systems in Matlab, and how to convert representations.
- How to numerically assess stability of an LTI system.
- How to stabilize an LTI system using transfer function methods.

1 Introduction

In this lab we consider the crane depicted below, modelled as a pendulum with movable massless base.



An actuator at the pivot point of the pendulum imparts a force u, the control input. The mathematical

model, using the variables depicted in the figure, is

$$\ddot{\theta} = -\frac{g}{l}\sin(\theta) - \frac{1}{Ml}\cos(\theta)u.$$

Letting the output y be the angle of the pendulum, and the state x be given by

$$x = \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix},$$

a state space model is

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= -\frac{g}{l}\sin(x_1) - \frac{1}{Ml}\cos(x_1)u \\ y &= x_1. \end{aligned} \tag{1}$$

This model is nonlinear. In Chapter 3 of the textbook, we linearized this model at the equilibrium $\bar{x} = \begin{bmatrix} 0 & 0 \end{bmatrix}^\top$ with control $\bar{u} = 0$, and obtained the LTI model

$$\begin{aligned} \dot{x} &= \begin{bmatrix} 0 & 1 \\ -g/l & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ -1/(Ml) \end{bmatrix} u \\ y &= \begin{bmatrix} 1 & 0 \end{bmatrix} x, \end{aligned} \tag{2}$$

approximating the behaviour of the pendulum near its lower position and at low speeds. In this lab, you will numerically solve the equations of motion of the pendulum in (1), and symbolically linearize them about an equilibrium to recover the linearization in (2). You will then evaluate the extent to which solutions of the linearized pendulum (2) approximate solutions of the nonlinear pendulum (1). Finally, you will design a simple controller for the linearization, and test it on the nonlinear 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
Lab report	4 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

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, that demonstrates you have understood your own work and how it connects to the course concepts. This portion of the mark will be assigned as follows:

0 out of 4	the report is either absent, or a simple printout of the Matlab outputs without commentary
1 out of 4	the report is incomplete *and* the commentary is inadequate
2 out of 4	the report is incomplete *xor* the commentary is inadequate
3 out of 4	the report is complete and the commentary is adequate, but lacks insight/clear understanding
4 out of 4	the report is complete and the commentary is adequate and demonstrates clear understanding

3 Numerical integration

In this section you will learn how to numerically integrate a (possibly time-varying) ordinary differential equation of the form

$$\dot{x} = f(t, x). \tag{3}$$

Once we choose the control input signal $u(t)$, the pendulum model in (1) fits this form, with

$$f(t, x) = \begin{bmatrix} x_2 \\ -(g/l)\sin(x_1) - 1/(ml)\cos(x_1)u(t) \end{bmatrix}.$$

In this section, you will use the following Matlab commands.

Matlab commands

`ode45`	Numerical ODE solver.
`odeset`	Used to set options for the ODE solver.
`plot`	Used to plot a function of a real variable.
`subplot`	Used to partition a figure into sub-figures.
`xlabel, ylabel`	Used to label the axes of a graph.

To solve the differential equation (3), we need to create a Matlab function that given $(t, x) \in \mathbb{R} \times \mathbb{R}^2$, returns $f(t, x) \in \mathbb{R}^2$. We will call this function `pendulum.m`, and save it in the same folder where your lab file `lab1.m` is located. Later on in this lab, we will learn a different way to create a Matlab function without saving it to a file.

The function pendulum.m.

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

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

  In the function declaration, `t` is the time variable (Matlab always requires it for ODE integration), `x` is the state vector (in our case, 2×1), and `parameters` is a structure containing the pendulum parameters M, g, l which we will pass to this function from the main script `lab1.m`.

- After the function declaration above, extract the pendulum parameters. For instance, write `M = parameters.M` to extract the mass.
- From the vector `x`, extract the pendulum states `x1 = x(1); x2=x(2);`
- Choose a control signal $u(t)$. Here, we will set `u=0;`
- Compute the 2×1 vector `xdot` according to the right-hand side of (1).
- Save the function `pendulum.m`.

Having created the function implementing the ODE to be integrated, you will now begin writing the main lab script.

Numerical integration.

- Create a script `lab1.m`. This will be your main file for this lab. Begin the script by clearing all the variables and closing all figures (see the Matlab help for commands `clear` and `close`).
- Define a structure called `parameters` containing the three pendulum parameters, $M = 0.2$ Kg, $l = 0.15$ m, and $g = 9.81$ m/sec^2. Use them to assign the structure fields `parameters.M`, `parameters.l`, `parameters.g`.
- Choose an initial condition. We will use two different initial conditions in sequence, $x^0 = \begin{bmatrix} 0 & \sqrt{g/l} \end{bmatrix}^\top$ and $x^0 = \begin{bmatrix} 0 & 1.99\sqrt{g/l} \end{bmatrix}^\top$.
- Using `odeset`, create a variable called `options` specifying the absolute and relative tolerances for numerical integration to be both equal to 10^{-7}: `options = odeset(...);`
- Using `ode45`, numerically integrate the ODE whose right hand side is the function `pendulum.m` you wrote earlier. You will pass the structure `parameters` to this ODE. The `ode45` call will look like this

  `[t,x]=ode45(@pendulum,Tspan,x0,options,parameters);`

  where:

  - `@pendulum` calls the function you wrote earlier.
  - `Tspan` is a row vector specifying the time range for the integration. Set `Tspan = linspace(0,10,1e3)` to integrate the solution between $t = 0$ and $t = 10$, forcing Matlab to return 1000 equally spaced solution samples.
  - `x0` is the 2×1 vector containing the initial condition.
  - `options` is the variable defined earlier specifying the integration tolerances.
  - `parameters` is the pendulum parameter structure which will be passed to the function `pendulum.m`.

– `ode45` outputs a pair `[t, x]` containing the solution $x(t)$ with initial condition `x0` at the time samples in `t`. The columns of the matrix `x` contain the state components. In this case, there are two columns.

Having learned how to numerically integrate solutions, take a look at different ODE solvers available in Matlab. We are using `ode45` which works well in many cases, but there are other solvers available.

Next, we turn our attention to plotting the solutions. When it comes to solutions of ODEs, there are two main types of graphs one is typically interested in: the graph over t of each component $x_i(t)$ and, for ODEs of dimension two or three, the graph of the curve in the state space traced out by the point $x(t)$ as t varies from 0 to ∞. This latter graph is called an *orbit* of the dynamical system.

Plotting of solutions. Return to your file `lab1.m`. You've just numerically solved the ODE for one of two given initial conditions. Now you will plot solutions.

- From the matrix `x`, extract the two columns and call them `x1` and `x2`, respectively. These columns represent $x_1(t_i)$ and $x_2(t_i)$, $i = 1, \ldots, 10^3$, where t_i is the i-th element of the vector `t`.
- Create a new figure. Using the commands `subplot(211)` and `subplot(212)` to divide the figure into two vertically stacked sub-figures, plot `x1` versus `t` in the first sub-figure, and `x2` versus `t` in the second sub-figure. Using `xlabel` and `ylabel`, label the axes of your plots.
- Create a new figure, and plot `x2` versus `x1`. This is the orbit mentioned earlier. You will notice that the orbit is a closed curve in the state space (the plane), a manifestation of the fact that the solution of the pendulum is periodic. Using `xlabel` and `ylabel`, label the axes of your plot.
- Repeat the integration and plotting steps for the second given initial condition.

Output 1.
- Produce *four* plots, as described above.
- Compare the results you obtained. What are the differences between the solutions with the two given initial conditions? Comment on the frequency of oscillation and the shape of the two orbits.
- Recall the physical meaning of the state x: x_1 is the angle θ and x_2 is the angular speed $\dot{\theta}$. From a physical point of view, what are the differences in behaviour of the pendulum for the two given initial conditions?

4 Symbolic Linearization

In this section you will write code to symbolically compute the linearization of the pendulum (1) at a given equilibrium and control pair $(\bar{x}, \bar{u})$. Recall the general theory: for a nonlinear control system $\dot{x} = f(x,u)$, $y = h(x,u)$, an equilibrium $\bar{x}$ with control $\bar{u}$ is by definition such that $f(\bar{x},\bar{u}) = 0$, and the linearization at $(\bar{x},\bar{u})$ is the LTI system

$$\dot{\tilde{x}} = A\tilde{x} + B\tilde{u}$$
$$\tilde{y} = C\tilde{x} + D\tilde{u},$$

where $\tilde{x} = x - \bar{x}$, $\tilde{u} = u - \bar{u}$, $\tilde{y} = y - h(\bar{x},\bar{u})$, and (A,B,C,D) are the Jacobian matrices

$$A = \frac{\partial f}{\partial x}(\bar{x},\bar{u}),\ B = \frac{\partial f}{\partial u}(\bar{x},\bar{u}),\ C = \frac{\partial h}{\partial x}(\bar{x},\bar{u}),\ D = \frac{\partial h}{\partial u}(\bar{x},\bar{u}). \tag{4}$$

MATLAB commands

`syms a,b,c real`	Declare symbolic variables `a,b,c`, and specify they are real.
`jacobian(f,x)`	Symbolic Jacobian of the vector function `f` with respect to the vector `x`.
`subs(f,a,b)`	In the symbolic expression `f`, substitute the vector `a` by `b`.

Symbolic linearization. Return to `lab3.m`.

- Declare real symbolic variables `x1,x2,t,u,m,l,g`.
- Define a symbolic 2×1 vector `xdot` representing the right-hand side of the state equation in (1), and a symbolic scalar `y` representing the output in (1). You have now symbolically defined the nonlinear control system.
- Place the symbolic variables `x1` and `x2` in a column vector `x`. Using the `jacobian` command, compute the four symbolic Jacobians in (4), the Jacobians of f and h with respect to x and u. Call these Jacobians `A,B,C,D` as in (4).
- Now you need to evaluate these Jacobians at $(\bar{x},\bar{u})$. We will choose the equilibrium $\bar{x} = \begin{bmatrix}0 & 0\end{bmatrix}^\top$, with $\bar{u} = 0$, corresponding to the pendulum pointing downward. Using the `subs` command, substitute $\bar{x}$ and $\bar{u}$ in place of x and u in `A,B,C,D`.

Output 2.

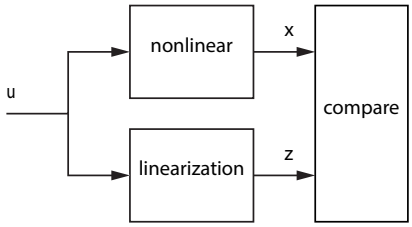
- Display matrices `A,B,C,D` you obtained and verify they coincide with those in (2).
- Adapting the steps above, compute the linearization at a generic equilibrium $\bar{x} = \begin{bmatrix}\bar{\theta} & 0\end{bmatrix}^\top$ with $\bar{u} = -mg\tan(\bar{\theta})$ and verify you get the same (A,B) matrices as in Chapter 5 of the textbook. Display these matrices.

5 From symbolic expression to numerical integration

Now we want to compare solutions of the nonlinear system (1) and those of its linearization at $\bar{x} = \begin{bmatrix} 0 & 0 \end{bmatrix}^\top$, $\bar{u} = 0$. Conceptually, this corresponds to solving the differential equations

$$\begin{aligned} \dot{x} &= f(x, u) \\ \dot{z} &= A(z - \bar{x}) + B(u - \bar{u}) \\ x(0) &= z(0), \end{aligned} \tag{5}$$

using the same initial condition and input signal for both equations.



In (5), we used z in place of x for the linearization because we need to distinguish these two states in the comparison we are about to make. We also replaced the error variables $\tilde{x}$ and $\tilde{u}$ by their definitions in terms of absolute variables in order to be able to compare solutions of the nonlinear system with those of the linearization.

In terms of implementation, we could amend the function `pendulum.m` to add the LTI dynamics of the linearized system (the second equation in (5)), but instead of doing that, we will learn a second method which starts with symbolic expressions.

Matlab commands	
`fun=matlabFunction(f,'Vars',t,x)`	Convert a symbolic expression `f` to a Matlab function `fun` of variables `(t,x)`.
`symvar(f)`	List of symbolic variables in the symbolic expression `f`.

From symbolic expression to Matlab function. Return once more to the script `lab1.m`.

- Declare real symbolic variables `z1,z2`, and place them in a column vector `z`.
- Define a symbolic 2×1 vector `zdot` representing the right-hand side of the $\dot{z}$ equation in (5). The matrices A and B are the ones you obtained earlier in the linearization procedure. The quantities $\bar{x}$ and $\bar{u}$ were also defined earlier in your code (and we've chosen them to be zero).
- Concatenate the vectors `xdot` and `zdot` in a 4×1 vector called `Xdot`. You have now symbolically defined the right-hand side of the augmented ODE (5).
- Using the command `subs` and the parameter structure `parameters`, substitute in `Xdot` the numerical values of m, g, l, as well as $u = 0$. The only symbolic variables left in `Xdot` should be `x1,x2,z1,z2`. Double-check that this is the case by issuing the command `symvar(Xdot)`.

- The command `matlabFunction(Xdot,'Vars',t,[x;z])` creates a Matlab function whose arguments are the pair (t, X), with X being the concatenation of x and z, and whose output is $\dot{X}$, the right-hand side of the ODE in (5). Call this function `augmented_pend`.
- Using what you learned in Section 3, integrate `augmented_pend` over the time interval $[0, 10]$ with initial conditions $x^0 = \begin{bmatrix} 0 & \sqrt{g/l} \end{bmatrix}^\top$ and $x^0 = \begin{bmatrix} 0 & 1.99\sqrt{g/l} \end{bmatrix}^\top$, in sequence. Note that $x(0) = z(0)$ in your integration, and there are four states now.
- For each initial condition, produce two figures (four figures in total) comparing the solution $x(t)$ of the nonlinear pendulum with $z(t)$, the solution of the linearization. The first figure should be subdivided into two sub-figures. Sub-figure 1 will overlap $x_1(t)$ and $z_1(t)$, while sub-figure 2 will overlap $x_2(t)$ and $z_2(t)$. The second figure will overlap the orbit $(x_1(t), x_2(t))$ of the nonlinear system and that of the linearization, $(z_1(t), z_2(t))$.

Output 3.
- Produce *four* plots, as described above.
- Compare the solutions of the nonlinear and linearized pendulum for the two given initial conditions. You will notice that, for one of the initial conditions, the linearization well approximates the nonlinear system, both in terms of frequency of oscillation and in terms of shape of the orbits. For the other initial condition, the approximation is instead very poor. Highlight all the similarities and differences you can come up with, and explain what causes them.
- Draw conclusions about the utility of the linearization. When is the LTI system a useful approximation of the nonlinear dynamics, and when is it not?

6 LTI REPRESENTATIONS

Now you will learn how to define an LTI structure and convert LTI system representations.

MATLAB COMMANDS

`double(A)`	Converts symbolic matrix `A` to double.
`eig(A)`	Eigenvalues of matrix A. The call `[V,D]=eig(A)` returns eigenvalues and eigenvectors, see Matlab's help.
`ss(A,B,C,D)`	Defines an LTI system object from the matrices (A, B, C, D).
`tf(sys)`	Converts an LTI system object `sys` to its transfer function representation.
`zpk(sys)`	Converts an LTI system object `sys` to its transfer function representation, factoring numerator and denominator into products of monomials corresponding to the real roots.
`[A,B,C,D]=ssdata(sys)`	Extracts the matrices of a state space realization of the object `sys`.

Continue working on the script `lab1.m`.

- Using `double` and the numerical values of m, g, l contained in the structure `parameters`, convert the symbolic matrices `A,B,C,D` to matrices of doubles.
- Using `ss`, define an LTI object `sys`.
- Find the transfer function of the linearized pendulum, and check it coincides with the transfer function developed in the textbook.
- Compare the poles of the transfer function and the eigenvalues of A.

Output 4.

- Display the transfer function of the linearized pendulum and comment on the verification mentioned above.
- Find the poles of the transfer function and the eigenvalues of A, and compare. Are they the same or different, and why?
- Is the linearized system stable, asymptotically stable, or unstable? Is it BIBO stable? Explain.
- The eigenvalues of A give the frequency of oscillation of solutions of the linearized system. Based on what you have learned about modal decomposition, explain what is the relationship between eigenvalues and frequency of oscillation. Then, verify that your predicted frequency of oscillation coincides with the one found numerically when looking at solutions of the linearized system.

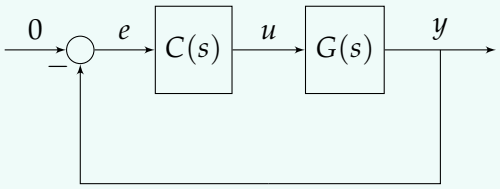
7 Pendulum stabilization

Having found the transfer function of the linearized pendulum, the objective now is to use basic feedback control tools that you learned in a third year control course (e.g., ECE311, ECE356, AER372) to stabilize the linearization. In particular, we want to check that a controller stabilizing the linearization will also stabilize the original nonlinear model, provided the initial conditions are sufficiently close to the equilibrium $\bar{x}$.

Take out pen and paper and note down the transfer function you found at the last step of your development. Call this transfer function $G(s)$.

- Using any technique you've learned in a previous control course, design a simple controller $C(s)$ such that the unity feedback closed-loop system depicted below is BIBO stable (note that the reference signal is 0 in this case). In this case, a lead controller with appropriate break frequencies will work. You may try, for instance, having the controller zero at 10, the pole at 1000, and its DC gain at -30.

  Note: the plant transfer function, $G(s)$ has a negative DC gain in this problem, so your controller $C(s)$ will also need to have a negative DC gain. To avoid confusions in your control design, remove the minus sign in front of $G(s)$, design $C(s)$, and when done add a minus sign in front of $C(s)$.



- Now you will test that your controller does indeed render the closed-loop unity feedback system BIBO stable. Back to `lab1.m`, use the command `C=tf(num,den)` to define an LTI system object in transfer function form representing your controller.

- The poles of the closed-loop system transfer function from reference to output are the zeros of $1 + C(s)G(s)$. Using the Matlab command `zpk(1+C*G)`, find the zeros of $1 + C(s)G(s)$, and verify that they lie in $\mathbb{C}^-$. If not, your control design is incorrect and should be revised.

- Now you need to implement your controller on the nonlinear pendulum model, a model expressed in state space form. To begin with, using `[F,G,H,L]=ssdata(C)` convert your controller to state space form and extract the matrices of the state space realization. We will call these matrices (F, G, H, L). Your controller is then given by

$$\begin{aligned} \dot{z} &= Fz - Gy \\ u &= Hz - Ly. \end{aligned} \tag{6}$$

  Pause and reflect on the above. Note that the input to the controller is $-y$, the negative output of the plant, and that's because the reference signal is zero. The output of the controller is the control input of the pendulum.

We remark that if we had linearized the pendulum at a nonzero equilibrium $\bar{x} = \begin{bmatrix} \bar{\theta} & 0 \end{bmatrix}^\top$, with $\bar{u} \neq 0$, the controller in (6) would have a different form:

$$\dot{z} = Fz - G(y - \bar{\theta})$$
$$u = \bar{u} + Hz - L(y - \bar{\theta}).$$

- Create a copy of the function `pendulum.m` and call it `controlled_pendulum.m`. In this function, the state `x` will be the concatenation of the state of the pendulum with the state of the controller in (6), and `xdot` will be the concatenation of the right-hand side of the pendulum model (1) and the right-hand side of the controller (6). This is the closed-loop system formed by the pendulum model and the linear controller you designed.
- Using `ode45`, integrate `controlled_pendulum.m` selecting an initial condition of the pendulum, and initializing the controller state at zero. Plot the pendulum states as functions of time, and verify they converge to the zero equilibrium.

Output 5.

- Display the transfer function of your controller, $C(s)$, and comment on how you designed your controller. Include Bode plots, or any other information that justifies and supports your design.
- Print the zeros of $1 + C(s)G(s)$ and discuss their location in the complex plane.
- Display your choice of initial condition, and display the pendulum states as functions of time, verifying that they converge to zero.
- Trying picking initial pendulum angles further away from zero. Does the solution still converge to the equilibrium? How far can you push the initial pendulum angle? Comment on your findings.