Lab 1 – ECE311 Introduction to Linear Control Systems

Introduction to LTI Systems in Matlab and Simulink

Main concepts of this lab

- How to define state space and transfer function models in Matlab
- How to convert between state space and transfer function representations
- How to numerically find the time response of an LTI system in Matlab
- How to plot solutions
- Definition of state space and transfer function blocks in Simulink
- Speed control of a permanent magnet DC motor

1 Introduction

In this lab you will be introduced to the basic tools required to perform numerical simulation of control systems in Matlab and Simulink. To motivate various constructions, we will use the example of a permanent magnet DC motor, whose model is

$$\begin{aligned} &\text{Armature:} \quad L_a \frac{di_a}{dt} + R_a i_a + K_e \dot{\theta} = u, \\ &\text{Rotor:} \qquad\qquad I\ddot{\theta} + b\dot{\theta} - K_t i_a = 0. \end{aligned} \tag{1}$$

In the differential equations above, θ denotes the angle of the motor shaft, i_a the armature current, and u the voltage applied at the motor terminals, our control input. The other parameters and their numerical values are defined in the table below. We choose the state $x \in \mathbb{R}^3$ given by

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \theta \\ \dot{\theta} \\ i_a \end{bmatrix},$$

and assume we can measure the angular rate of the motor shaft, $\dot{\theta}$, via a tachometer. We take this to be our output. The control problem is speed control: make the output $\dot{\theta}$ converge to a desired constant.

We rewrite the equations above in state-space form

$$\begin{aligned} \dot{x} &= \begin{bmatrix} 0 & 1 & 0 \\ 0 & -b/I & K_t/I \\ 0 & -K_e/L_a & -R_a/L_a \end{bmatrix} x + \begin{bmatrix} 0 \\ 0 \\ 1/L_a \end{bmatrix} u \\ y &= \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} x. \end{aligned} \tag{2}$$

Motor Parameter	Description	Numerical value
L_a	Armature inductance	0.02 H
R_a	Armature resistance	3 ohms
K_e	Back emf constant	0.01 V/(rad/sec)
K_t	Motor torque constant	0.01 N m /A
I	Motor moment of inertia	$6 \cdot 10^{-4}$ N m /(rad/sec^2)
b	Viscous friction coefficient	10^{-4} N m /(rad/sec)

It is common to approximate the model (2) by assuming that the ratio R_a/L_a is very large, so that the term $L_a di_a/dt$ can be ignored, and the armature equation in (1) can be approximated by

$$R_a i_a + K_e \dot{\theta} = u,$$

in which case we can solve this identity for i_a,

$$i_a = -\frac{K_e}{R_a}\dot{\theta} + \frac{1}{R_a}u.$$

Effectively, in writing this expression we are assuming that the electrical dynamics are much faster than the rotor dynamics, and therefore the transient of the armature current can be ignored. The expression for i_a we just found is called a *quasi steady-state*. We now substitute this expression into the rotor equation, to obtain the simplified model of the DC motor:

$$I\ddot{\theta} + \left(b + \frac{K_e K_t}{R_a}\right)\dot{\theta} = \frac{K_t}{R_a}u.$$

Letting $x_1 = \theta$, $x_2 = \dot{\theta}$, and $x = \begin{bmatrix} x_1 & x_2 \end{bmatrix}^\top$, the state space model is now given by

$$\begin{aligned} \dot{x} &= \begin{bmatrix} 0 & 1 \\ 0 & -\left(b + \frac{K_e K_t}{R_a}\right)/I \end{bmatrix} x + \begin{bmatrix} 0 \\ \frac{K_t}{R_a I} \end{bmatrix} u \\ y &= \begin{bmatrix} 0 & 1 \end{bmatrix} x. \end{aligned} \tag{3}$$

To summarize, the LTI system (2) with three states is the full model of the DC motor, while the LTI system (3) has only two states and it constitutes a simplification of (3). To what extent does the model (3) approximate the full model (2)? You will explore this question in Sections 4 and 5.

Note: This lab was developed using Matlab R2020b. If you use an earlier version of Matlab there is a very small chance that some of the instructions here might not work for your version, in which case you should update your software.

Throughout the lab, you will be guided through a number of steps which will require you to write Matlab code or draw Simulink diagrams. You will write your code in a Matlab script called `labx.m`, where `x` is the lab number. Your Simulink diagram will be saved as `labx.slx`. If there are multiple Simulink files, save them as `labx_1.slx,labx_2.slx`, and so on. 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 poles of the transfer function.

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

Convert the state space model into a transfer function model, then find the poles.

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 numbers 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.** Below is a plot of the output signal that was obtained with this controller.
> (...)
> We observe that the output signal converges to a steady-state value of 25 rad/sec (...)
> **Output 2.** Tuning of the controller gain gave the following result, see the figure below illustrating the step response of the system. We observe that there is marked improvement in the transient performance of the control system, and indeed (...)

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

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

Late submissions. *All submissions are due at 8PM* of the day indicated on the course website. We *do not* accept late submissions under any circumstances. For more details, see the *late submission policy* on the course website.

3 LTI SYSTEM REPRESENTATIONS AND CONVERSIONS

In this section you will learn how to define LTI models in Matlab, in both state space and transfer function forms. You will also learn how to operate conversions between these two representations. You'll perform these operations using the DC motor models (2) and (3).

MATLAB commands

`sys=ss(A,B,C,D)`	defines an LTI system object in state-space form.
`sys=tf(num,den)`	creates an LTI system object in transfer function form.
`sys=zpk(Z,P,K)`	creates a transfer function with zeros `Z`, poles `P`, and gain `K`.
`ss(sys)`	converts any LTI object `sys` into state space form.
`tf(sys)`	converts any LTI object `sys` into transfer function form.
`zpk(sys)`	converts any LTI object `sys` into zpk form.
`ssdata`	extracts the (A, B, C, D) matrices from any LTI object..
`tfdata`	extracts numerator and denominator of the transfer function from an LTI object.
`zpkdata`	extracts zeroes, poles and gain from any LTI object.
`cell2mat(C)`	converts a cell array to an ordinary array.

In this course we consider *rational* transfer functions, i.e., functions given by the ratio of two polynomials,

$$G(s) = \frac{b_m s^m + \cdots + b_0}{s^n + a_{n-1}s^{n-1} + \cdots + a_0}.$$

In Matlab, such a transfer function is defined using the command `tf`, placing the coefficients a_i, b_j into two arrays, `num` and `den`. Sometimes, it is convenient to factor the numerator and denominator into products of elementary terms, so as to expose the zeroes and poles of the transfer function:

$$G(s) = K\frac{(s - z_1) \cdot (s - z_2) \cdots (s - z_m)}{(s - p_1) \cdot (s - p_2) \cdots (s - p_n)}.$$

In Matlab, the form above is called a ZPK representation, and it is defined by the two arrays $Z = [z_1 \ \ldots \ z_m]$, $P = [p_1 \ \ldots \ p_n]$, and the gain K. The command `zpk` defines an LTI object from this data.

Defining the motor model. Create a script `lab1.m` and do the following.

- Create motor variables `La,Ra,Ke,Kt,I,b` with numerical values given by the entries in the table found in Section 1.
- Create matrices `A,B,C,D` representing the motor model in (2) (the scalar `D` is zero here).
- Create matrices `A1,B1,C1,D1` representing the *simplified* motor model in (3) (the scalar `D1` is zero).
- Using the command `ss`, define the state space model of the motor (2) in an object named `motor`, and that of the simplified model (3) in an object named `motor_simplified`.
- Using the command `tf`, convert the objects `motor` and `motor_simplified` to transfer functions, and name the resulting objects `G_motor` and `G_motor_simplified`, respectively.

- Using the command `zpk`, convert the object `motor` to ZPK form, and name the resulting object `zpk_motor`. You don't need to do this for the simplified model.
- Using the commands `zpkdata` and `cell2mat`, extract the list of poles of the transfer function from the object `G_motor`. You don't need to do this for the simplified model.
- Using the commands `tfdata` and `cell2mat`, extract numerator and denominator arrays from the object `G_motor`, and name them `num` and `den`, respectively. You will need these arrays in Section 5. Repeat this procedure for the transfer function object `G_motor_simplified`, this time creating arrays `num1` and `den1`.

Output 1.
- Print out the transfer functions `G_motor` and `G_motor_simplified`.
- Print out the transfer function in ZPK form, `zpk_motor`.
- Print out the poles of the transfer function `G_motor`. Comment on the location of the two poles, and how are they related to one another: are they close to each other? Is one much smaller in magnitude than the other? Comment on how you expect the motor to behave based on the location of the poles when the input voltage is a unit step.
- Print out the numerator and denominator arrays of the transfer functions `G_motor` and `G_motor_simplified`. You will need these arrays in Sections 5 and 6.

4 Numerical simulation of LTI systems

Having defined the motor model in Matlab, you will now numerically simulate it to glean useful information about the motor behaviour when subjected to step and sinusoidal inputs. For step inputs, you will verify that the simplified model (3) does indeed approximate the full model (2) very well.

Matlab commands

`linspace(T1,T2,N)`	Generates an array of `N` equally spaced numbers between `T1` and `T2`.
`step(sys,T)`	Plots the step response of the LTI object `sys` at the times in the array `T`. If called as `[Y,T,X]=step(sys,T)`, it returns matrices `X,Y` containing the state and output samples corresponding to the times in `T`. If `sys` is a transfer function, a state space realization of it is used in producing `X`.
`lsim(sys,U,T,X0)`	Plots the time response of the LTI object `sys` at the times in the array `T` subject to the control input whose samples are in `U`, and the initial condition in `X0` (this latter is only meaningful if the model is in state space form).

`evalfr(sys,s)`	Evaluates the transfer function of the LTI object `sys` at `s`.
`plot(T,Y)`	Plots the samples in `Y` versus the samples in `T`.
`subplot`	Partitions a figure into sub-figures.
`xlabel, ylabel`	Labels the axes of a graph.
`title`	Adds a title to a figure.

Continue working on your `lab1.m` script.

- Define an array `T` of 1000 time samples, equally spaced between 0 and 30 seconds.
- Using the command `subplot(311)`, create a figure with three subplots, vertically aligned.
- Using the command `step`, find the step response of the state space model `motor` at the time samples in `T`, save it in an array `Y1`, and plot it in the first subplot versus `T`.
- Find the step response of the simplified state space model `motor_simplified` at the time samples in `T`, save it in an array `Y2`, and using `subplot(312)` plot it in the second subplot versus `T`.
- Plot `Y1-Y2` versus `T` in the third subplot using `subplot(313)`, and verify that the step response of the simplified model approximates very well that of the full model.
- The last entry of the output vector `Y1` that you've just computed is an approximation of the motor's asymptotic speed for a unit step voltage. Using the Final Value Theorem and the transfer function `G_motor`, determine the theoretical asymptotic value of the motor speed in response to a unit step.
- In a second figure, plot the armature current (third component of the state) of the full motor model (2) versus time when the input is a step. For this, you need to use the state space model `motor`.
- In a third figure, using the command `lsim` and the LTI object `motor`, find the output response with initial condition `X0=[0;-1;.5]` and input signal `sin(T)`, and plot it versus time.
- From the figure you just plotted, determine the approximate amplitude of oscillation in steady-state of the motor speed in response to the input signal `sin(T)`.
- Using the command `evalfr(G_motor,s)`, evaluate the motor transfer function at $s = i$ (the frequency of the sinusoidal input is 1 rad/sec), and verify that the amplitude you found is approximately equal to $|G(i)|$. This is the manifestation of a general property of the frequency response that we shall review later on in the course.

Output 2.

- Produce three figures as explained above. Label the axes of each (sub-) figure, and add titles explaining what the (sub-) figure represents.
- Compare the step responses of the motor model and its simplified version. How close are they to each other? What is the maximum error between the two?
- Print the approximate asymptotic value of the motor speed in response to a unit step.
- Print the theoretical asymptotic value of the motor speed in response to a unit step. Verify that the approximate value above is indeed very close to the theoretical value.
- Print the approximate amplitude of oscillation of the motor speed in response to a sinusoidal input.
- Print the theoretical amplitude of oscillation of the motor speed in response to a sinusoidal input, and compare it to the approximate value above.

5 Definition of LTI system blocks in Simulink and numerical simulation

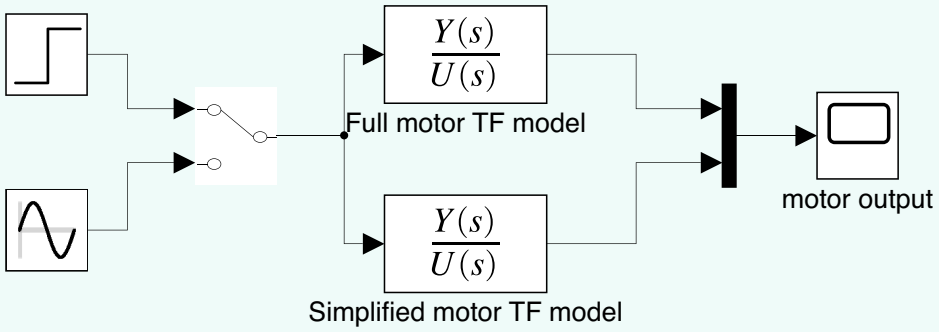
In this section you will define the motor model and its simplified version in Simulink. You will then compare their outputs and verify once again that the simplified model offers an accurate approximation.

Simulink blocks (inside the Library Browser)

`Continuous` → `State-space`	define a state space LTI object
`Continuous` → `Transfer Fcn`	define a transfer function LTI object
`Sources` → `Step`	generate unit step signal
`Sources` → `Sine wave`	generate sinusoidal signal
`Sinks` → `Scope`	plot signal vs time
`Signal routing` → `Mux`	create a vector signal from two scalar signals
`Signal routing` → `Manual switch`	manually switch between two signals

Open Simulink and the Library Browser. Create a blank diagram called `lab1_1.slx`.

- Using the blocks listed above, draw a Simulink diagram like the one depicted below. In it, the same input signal (a step or a sine wave) is fed simultaneously to two different transfer function models. The outputs of these models are then plotted versus time.



- Edit the first transfer function block. Inside the boxes labelled num, den, write `num` and `den`. This will tell Simulink to use the numerator and denominator arrays of the motor transfer function that you've previously defined in the workspace. If you have erased them, run your `lab1.m` script first.

  Note. You could replace this transfer function block by a state space block using the matrices `A,B,C,D` that you've previously defined in the workspace. The results of the following comparison would be identical, except that with the state space block you can set a nonzero initial condition.

- Edit the second transfer function block, and this time enter `num1` and `den1` in the numerator and denominator boxes. Thus the second transfer function represents the simplified motor model.

- Edit the Step block and make sure of the following: the initial value of the step is 0, the final value is 1, the step time is zero.

- Edit the Sine Wave block and make sure that the frequency is 1 rad/sec, the amplitude is 1, the phase is 0 and the bias is zero.

- Open the `Model settings` and under the `Solver` menu, set both relative and absolute tolerances to 10^{-10}. Make sure that the solver is `variable-step`, and set the `stop time` to 30 seconds.

- Run the Simulink diagram. Plot the output of the motor models when the input is a unit step first, then a sine wave, and verify that in both cases the outputs of the two transfer functions give almost identical results. Save the results into two figures, one for the step input, one for the sinusoidal input. To save the Simulink scope, select `File` → `Print to Figure`, then save the figure in the format of your choice. **Do not take screenshots.**

Output 3.

- Produce the two figures described above.
- Comment on how well the simplified transfer function model approximates the full transfer function model for the two given input signals.

6 Proportional control of the permanent magnet DC motor

Having verified that the simplified model `G_motor_simplified` well approximates the full motor model `G_motor`, we now work with `G_motor_simplified`, and try our hand at the speed control problem. The problem in question is to make the output y (the speed of the motor shaft) converge to a desired constant value.

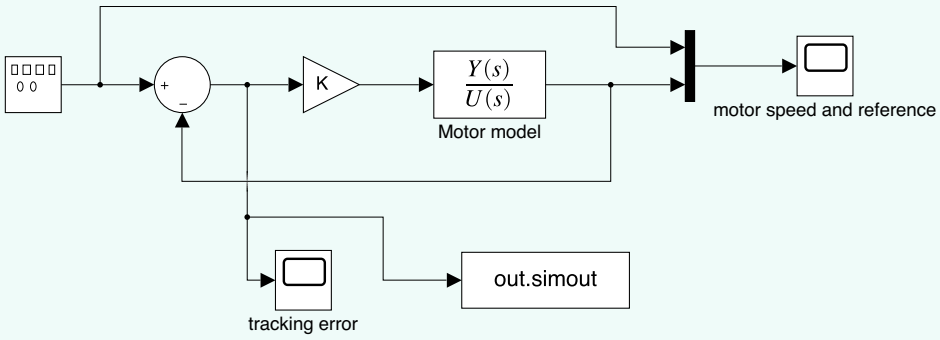
To achieve this control objective, we use the simplest kind of controller, a proportional one. A proportional controller is one where $u(t) = Ke(t)$, where $e(t)$ denotes the tracking error, and $K > 0$ is the control gain. We don't yet have the theoretical tools to check whether and to what extent this type of controller is appropriate for our speed control problem, but we'll nonetheless go ahead and try it out. Later in our lectures we'll justify the viability of this controller and understand its limitations, limitations that you'll have an opportunity to observe in a moment.

Simulink blocks (inside the Library Browser)

`Sources` → `Signal Generator`	generates various input signal classes
`Math Operations` → `Add`	summing block
`Math Operations` → `Gain`	constant gain block
`Sinks` → `To Workspace`	saves signal to the Matlab workspace

Create a blank Simulink diagram named `lab1_2.slx`.

- Using the blocks described above, draw in Simulink the block diagram depicted below. There are two scopes in the diagram. One scope monitors the tracking error, while the second scope monitors the output signal superimposed to the reference signal.



- Open the summing block, select `round shape` and write this list of signs: `|+-` in place of the existing signs.
- In the `Transfer function` block, you will specify the numerator and denominator arrays, `num1,den1`, of the object `G_motor_simplified`.

- The `Signal Generator` block generates the reference signal representing the desired speed of the motor shaft. We'll edit this block and select a square wave with amplitude 1 and frequency 0.02 Hertz. In other words, we are asking the motor speed to converge to 1 rad/sec or -1 rad/sec, with periodic switching between these two values.
- The `To Workspace` block saves the tracking error to the workspace in the form of a structure, from which you can extract the time samples (in `out.simout.Time`) and the tracking error samples (in `out.simout.Data`).
- Open the `Model settings` and under the `Solver` menu, set both relative and absolute tolerances to 10^{-10}. Make sure that the solver is variable-step, and set the `stop time` to 100 seconds.
- Edit the `Gain` block and set the control gain value to 0.1. Run the feedback control loop and save the figure corresponding to the output and reference signal scope. Once again, do not take a screenshot of the figure. Rather, use the technique explained in Section 5 to save them.
- From the tracking error scope, deduce the approximate steady-state tracking error. You can do that using the `Tools` $\rightarrow$ `Measurements` $\rightarrow$ `Cursor Measurements` tool within the scope.
- Now set the control gain value to 1 and repeat the previous two steps.

Output 4.
- Produce two figures displaying the motor speed and the reference signal for the two cases of $K = 0.1$ and $K = 1$.
- Compare the results in the two figures, and comment on your findings. You should notice that the motor speed does *not* converge to the reference signal, but it gets close to it. Determine the approximate asymptotic value of the tracking error in the two cases $K = 0.1$ and $K = 1$. How is this value affected by increasing the control gain K? What about the rate of convergence, do you note differences in the rate of convergence of the tracking error to its asymptotic value as K is increased?
- Make concluding remarks. Do you think that proportional control is an adequate means to regulate the speed of a DC motor? Do you have any ideas for improvement?