ELEC4631 Lab 0
Introduction to Matlab and Simulink

Term 2

1 Objective

In ELEC4631 laboratory sessions, Matlab/Simulink will be used for lab experiments, including applying real-time control to lab apparatus. The objective of this lab session is to review or learn the necessary tools required in preparation for the ELEC4631 labs. It is **essential** for everyone to get acquainted with Matlab and Simulink environments, namely:

a Matlab: Learn the basics of Matlab, knowing how to construct vectors and matrices, operate with matrices, including adding and multiplying two matrices, taking the inverse, the trace and determinant of a matrix.

b Simulink: To start the Simulink Library Browser, and how to create a new Simulink model using blocks from the toolbox libraries (sources, sinks, transfer functions, summing junctions, products, etc), to run a Simulink model, and save the results.

2 Introduction to Matlab

In this section we introduce Matlab basics with some exercises. From these basics you will be able to understand or use more complex functions in Simulink and the real-time tool Simulink Coder. The Matlab version used in this lab is 2015b. Matlab is well documented, the help documents can be accessed by typing **doc** in the Matlab command window and choosing the topic. Or use the command **help** followed by a function name. For example, **help sum** will return a help text for the function **sum**. Further more, the command **demo** will show many examples for the use of Matlab toolboxes.

Get familiar with the command window, workspace, command history, and navigation in the Matlab documentation. In command window use the cursor keys to navigate up and down to the command history and the tab key for auto-completion.

Matlab code and commands can be written and stored in so-called *m-files* (there is a built-in editor for it in Matlab and in *m-files* everything that follows the character % is treated as a comment), and it is convenient to create m files for your Matlab tasks.

Very long (or continuous)-running Matlab codes can be interrupted by **Ctrl-C**. You can leave Matlab by typing **exit** or simply closing the Matlab window.

Useful commands: doc, help, clc, clear, clear all

2.1 Pre-lab work: Matrices, Arrays and Vectors

First, you learn how to create matrices, the fundamental data structure in Matlab, using Matlab commands as an explicit list of elements, via index assignment. Learn how to suppress outputs to the command line using **;**. Understand the colon **:** operator. Understand how to concatenate matrices and how to delete rows/columns.

Commands: [ ], [1,2,3], [1;2;3], A(1,2), diag, :, linspace, rand, randn, end, zeros, ones, eye, gallery, spares, full, size, length

Pre-lab exercise 1: First, using at most three Matlab commands create the matrix

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 5 & 4 & 3 & 2 & 1 \end{bmatrix}$$

and assign random numbers to the second row of this matrix. Then using at most three Matlab commands, create the matrix

$$B = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ -1 & -2 & -3 & -4 & -5 \\ 1 & 2 & 3 & 4 & 5 \\ -1 & -2 & -3 & -4 & -5 \\ 1 & 2 & 3 & 4 & 5 \end{bmatrix}$$

Note that when accessing the elements of a matrix with a single index, the elements will be addressed column-wise starting with the first column, e.g. for the matrix in Pre-lab exercise 2, **B(6)** will return 2, the (1,2) element of the matrix.

Pre-lab exercise 2: Create a sparse zero matrix C of size $n \times n$ using the commands

```
n = 4;
C = sparse(n,n);
```

and use at most two Matlab commands to populate its entries such that the command **full(C)** will return

```
ans =
 2  -1   0   0
-1   2  -1   0
 0  -1   2  -1
 0   0  -1   2
```

Next, you should learn how to add and multiply two matrices, and take the inverse, trace and determinant of a matrix.

Commands: +, -, *, /, .*, ./, sum, prod, transpose, inv, trace, det.

Pre-lab exercise 3: Use **inv**, **trace** and **det** to obtain the inverse, trace and determinant of the matrix B from pre-lab exercise 1.

2.2 Expressions, Linear Algebra and Graphics

Expressions

In Matlab, a number is just a 1x1 matrix. There are some special numbers such as i, pi, eps, realmin, realmax, Inf and NaN (not case-sensitive), but Matlab does not stop you from reusing i, pi, eps, etc as variable names, which may cause errors. So, such variable names should be avoided.

Variables do not need to be declared as they are instantiated with their first definition by the assignment operator =.

Linear Algebra

Matlab is a powerful linear algebra tool. Matrices and vectors can be added and subtracted (commands + and -) and multiplied (command *). Be aware that matrix-matrix multiplication is different from element-wise multiplication (.*).

One of the most important commands is the backslash command (equivalently, mldivide) for solving linear systems or least squares problems.

Other commands: cond, rank, null, norm

Lab exercise 1: Create a magic square A and a vector b

```
A = magic(5)
b = ones(5,1)
```

Solve the linear system $Ax = b$, obtain the numerical solution $\tilde{x}$.

More commands: qr, chol, schur, eig, eigs, svd, svds, fft

Assessment exercise 1: Learn how to use the **eig** command. Compute the eigenvectors and eigenvalues of the matrix A from Lab exercise 1. All eigenvalues λ_j of a matrix with positive entries and equal row sums σ satisfy $|\lambda_j| \leq \sigma$, and that there is exactly one eigenvalue $\lambda_k = \sigma$ (*Perron-Frobenius Theorem*). Verify this numerically.

2.3 Graphics

There are many Matlab functions for 2D/3D plots and images as well as animations. Here we only list a few of the commonly used ones, which are useful for plotting the results in the workspace from running a Matlab command, script, as well as data saved in Matlab .mat files.

Commands: figure, close, hold, spy, plot, semilogx, loglog, bar, hist, meshgrid, contour, contourf, surf, pcolor, image, imagesc

Example 1: Plot the function $f(x) = \sin(x)$ and its zeros on $[0, 2\pi]$.

```
figure(1)
x = linspace(0,2*pi, 200); % 200 points are enough for plotting
fx = sin(x);
plot(x,fx,'b-'); % this plots points (c,sin(x)) as a (b)lue line (-)
hold on
plot([0,pi,2*pi], [0,0,0], 'ro') % this plots zeros of sin(x) on [0,2π] as (r)ed circles (o)
```

There are also commands for changing the axes, adding legends or titles, the most useful ones are listed below.

Commands: axis, xlim, ylim, xlabel, ylabel, legend, title, grid

Example 2: Using these commands the plot for the results of Example 1 can be more descriptive.

```
figure(1)
xlabel( 'x'); ylabel('f(x)');
legend('sin(x)','zeros')
title('a simple graph')
axis([0,2*pi, -1.2, 1.2])
```

2.4 Scripts and Functions

Programming Scripts

Matlab programs are typically written as *m-files* by an editor, which can be as simple as a few lines of Matlab commands, or a script with programing loops and logic. Matlab integrated editor allows interactive running of your code, which makes programming and debugging efficient.

Keywords for, while, if else, continue, break, switch, case

Logical: ==, &&, ||, -, all, any, isempty

Example 3: type **demo** in the command window and the Help window opens titled MATLAB Examples. Click **Programming** and click on the choice **Function Functions** to learn about script writing. This will help you with the function you need to create in the next example.

Functions

Functions in Matlab are essentially m-files that accept input arguments and produce outputs. They then can be called in the Matlab command line or within Matlab scripts with their defined parameters. The following example shows the creation of a function and the use of it.

Example 4: We create the content of an m-file named **quadroots.m**

```
function [z1,z2] = quadroots(p,q)
% QUADROOTS Computes the solutions of z^2+p*z+q = 0.
% Given scalar inputs p and q, this function returns
% the roots z1 and z2 of a quadratic equation.
z1 = -p/2 - sqrt(p^2-4q)/2;
z2 = -p/2 + sqrt(p^2-4q)/2;
return
end
```

then the above function can be called by its name, i.e.

```
z1 = quadroots(0,1; % z2 will be ignored
[z1,z2] = quadroots(0,1);
```

(Note that **help quadroots** will show the comments following the function keyword.)

Many Matlab built-in functions are implemented as m-files and can be viewed in the editor, e.g. **edit magic** will open the implementation of the function that generates magic squares (you already created such a square in Lab exercise 1). Alternatively, to display the contents of the m-file in the main Matlab window you can type **type magic** on the command line.

Example 5: Similarly type **demo**, click **Programming** and choose **Nested Functions** to see more about the use of functions.

2.5 Pay Attention to Integration and Differential Equations

Matlab has many routines for computing, such as elementary math operations, linear algebra, statistics and random numbers, interpolation, optimization, fourier analysis and filtering, sparse matrix computation and computational geometry, just to name a few.

However please have a good look at the functions for numerical integration and solving differential equations, as they are most useful for the analysis of dynamic systems. It will make you understand the underlying methods used for Simulink simulations.

Commands: integral, integra12, integra12, quad, ode45, ode15s, ode23, odeset, odeexamples

There are a number of commands in **Control Toolbox** commonly used in control system analysis and design, please refer to *Control Systems Engineering* by Nise.

3 Simulink

3.1 Introduction

Simulink is an environment in Matlab for simulation and model-based design of dynamic and embedded systems. It provides an interactive graphical environment and a customizable set of block libraries that allows you to design, simulate, implement, and test a variety of time-varying systems, including for communications, controls, signal and image processing.

Using Simulink, in many cases, you can quickly develop your model, which looks visually like a block diagram, using the blocks in its libraries, instead of writing a text based programming language such as C. Moreover, Simulink has integrated solvers for the simulation of dynamic equations.

3.2 Start with Simulink Library Browser

Type **simulink** in the Command window, the **Simulink Library Browser** opens. The **Simulink Library Browser** shows the libraries with the blocks you may use for your Simulink model. It is an extensive library of functions commonly used in modelling a system, including:

- Continuous and discrete dynamics blocks, such as Integration, Transfer Functions, Transport Delay, etc.
- Math blocks, such as Sum, Product, Add, etc.
- Sources, such as Ramp, Random Generator, Step, etc.

3.3 Create a new Simulink model

Click on the **New** icon on the Toolbar to create a new Simulink model. A new window opens with the model name **untitled**. Now you may drag the blocks you want to use from the **Simulink Library Browser** into the window of your model, wiring them using linking lines to form a functional model with inputs and outputs, then saving it as an *.slx file with a name of your choice.

Lab exercise 2: Drag a Sine Wave and a Scope block into the model window, and then click and hold the mouse to wire the output of Sine Wave to the input of Scope. Now you can run the model by clicking the Run icon in the Toolbar and double clicking the Scope to see the results.

3.4 Configuration

There are parameters you can configure that control how the simulation runs. Click on the Simulation icon on the Toolbar to open the Simulation menu, and select Configuration Parameters to browse the options.

Among the many the important parameters are:

- Start/stop time for the simulation
- What solver to be used (ode45 or ode23 etc.)
- Fixed-step/variable-step

Solvers are numerical integration algorithms that compute the system dynamics over time using information in the model. Simulink's solvers support the simulation of a broad range of systems, including continuous-time, discrete-time, hybrid and multi-rate systems of any size (of course, limited by the amount of memory and computational resources available).

Assessment exercise 2: Continue with the model created in Lab exercise 2 and insert an **Integrator** block between the **Sine Wave** and **Scope**. Click the scope open and run the simulation to show the results, as well as store the results to the Matlab workspace as a data array with time and plot it from workspace.

3.5 Help window

For more detailed information about individual Simulink blocks, use the built-in Help. All standard blocks in Simulink have detailed help. Click the Help button in the Block Parameter window of a specific block for its details.

(End of Lab 0 note.)