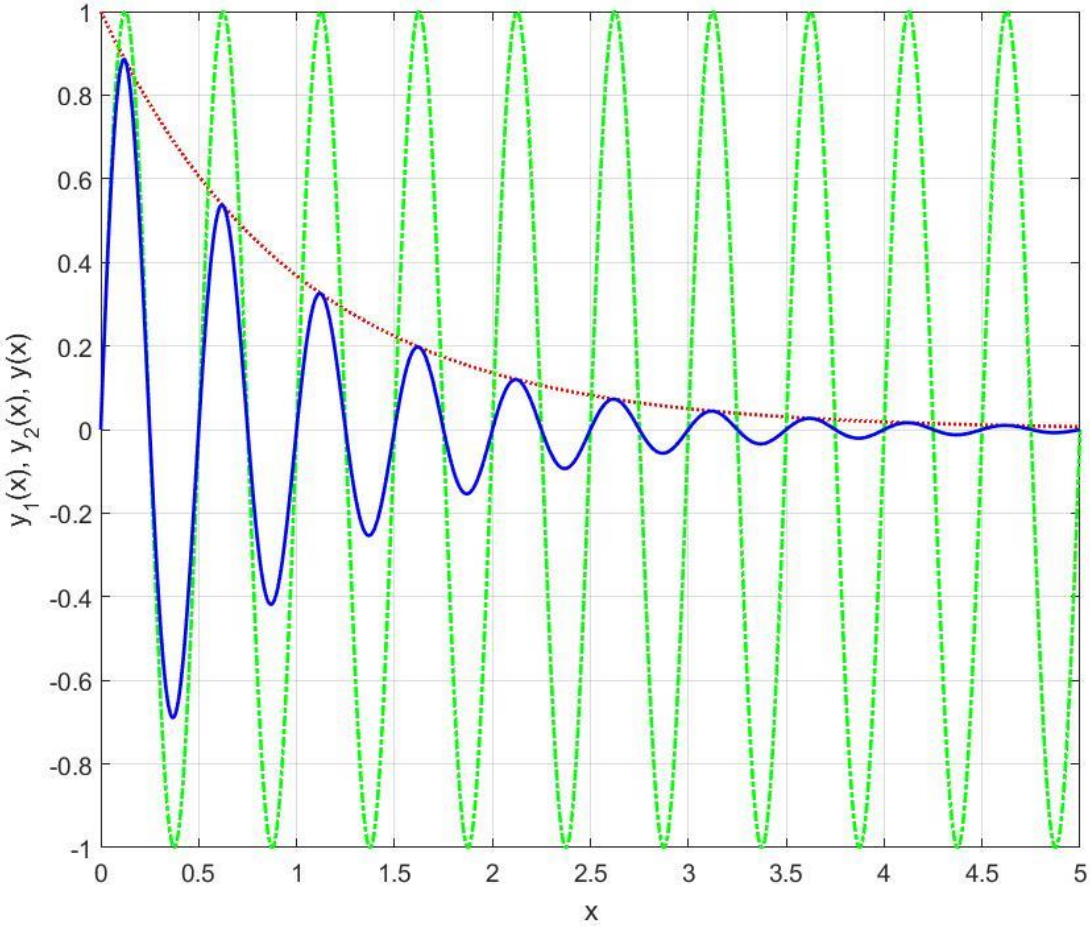
UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE & ENGINEERING
First Year Program – Core 8 and TrackOne

FIRST YEAR PROGRAM
ENGINEERING PROBLEM SOLVING LABS

MAT188: Laboratory #1
Discretization



DISCRETIZATION

This lab will introduce you to some fundamental concepts of numeric computation through the use of MATLAB. This includes basic syntax and commands, vectors, and two-dimensional plots. You will also be introduced to a key element of the **Engineers' Toolbox**: *Discretization.*

You will explore how to get help with MATLAB using various resources when you need it. There are many online resources that can help you with MATLAB, but we will focus on the resources available through Mathworks (the company that distributes MATLAB).

Learning Outcomes

By the end of this lab you will be able to…

1) Complete basic calculations using MATLAB,
2) Define and work with vectors in MATLAB,
3) Create simple two-dimensional plots in MATLAB,
4) Know how to seek help for MATLAB commands, and
5) Understand how engineers use discretization in order to approximate mathematical functions.

Preparation *(Required to do* ***before*** *you complete this lab)*

1. If you have not already done this in a previous term, complete the first 3 sections of **MATLAB Onramp** online exercise at https://matlabacademy.mathworks.com (i.e., *Section 1: Course Overview* to *Section 3: Vectors and Matrices*). This should take you less than 1 hour. Remember your login information for the Mathworks account you create to complete the Onramp, since **you will be asked to log in during the lab to show your Lab TA your Progress Report at the start of the lab session. Make sure you are completing the MATLAB Onramp using version R2017a.**
2. Watch the posted *MAT188 Lab #1 Introduction Video* and follow along with the posted slides.
3. Read through the entire Lab #1 handout (i.e., these notes), so you know what will be expected of you during the lab.

Related Reference Materials *(Not required, but may be a helpful resource)*

Getting Started with MATLAB (video)
http://www.mathworks.com/videos/getting-started-with-matlab-68985.html
or through the MATLAB application by selecting Help > *Examples* in the Toolbar at the top of the window.

MATLAB Skills and Knowledge: *Basic Calculations, Functions, Working with Vectors, and Navigating the Command Window*

Command-Line Calculations

Log in to one of the ECF computers and run MATLAB. As you read through this lab, type all of the example commands into MATLAB to see the results.

MATLAB will open with the Command Window, where you can type commands one at a time to execute them. Commands are executed by pressing **Enter** or **Return**, with the output being displayed on the screen immediately after each command. For example, try

```
>> 3+7.5
```

This will give a result:

```
ans =

10.5000
```

A ***variable** is* a value that can be changed and has a name to refer to it. After a variable is defined, it can be used in subsequent commands. ***Type*** in the following commands to create the variables `a` and `b`:

```
>> a=14/4

a = 3.5000


>> b=a+1

b = 4.5000
```

If you do not assign the result to a variable, the result of the last performed computation is automatically stored in the built-in variable `ans`.

```
>> 14/4

ans = 3.5000


>> ans*2

ans = 7.0000
```

You can use scientific notation to enter very large and very small numbers. For example, ***define the scientific constants*** $\varepsilon_0 = 0.000000000008854 = 8.854 \times 10^{-12}$, $\mu_0 = 4\pi \times 10^{-7}$, and $N_A = 6.022 \times 10^{23}$ as variables. Note that `pi` is a built-in constant in MATLAB.

```
>> epsilon0=8.854e-12
>> mu0=4*pi*1e-7
>> avogadros_constant=6.022e23
```

Do the following exercises:

1. ***Ask for Help***.
   - *Command line help*: You can access help from the command line (in the Command Window) by typing `help`. To get information about a specific command, include the command name. **For example, type `help clear` to learn what `clear` does**. It will give a lot of text, but you should get a sense of the basic purpose of this command.
   - *Online documentation*: Mathworks support provides searchable documentation, examples, demos, and videos. This is accessible at www.mathworks.com/support/index.html, or by clicking Help in the toolbar at the top of the MATLAB window. ***Use this online documentation to read about the `sin` function.***
2. ***Evaluate the following expressions in MATLAB***. Observe how MATLAB follows the proper order of operations for math expressions, so parentheses `()` can be used to change the order of operations.
   a) `8 * 5 + 3`
   b) `8 * (5 + 3)`
   c) `10 / 2 / 5 - 3 + 2 * 4`
   d) First, define `a = 1.1` then evaluate `c = a^2021`
   e) First, make sure `epsilon0` and `mu0` are defined as above. Then evaluate `((epsilon0*mu0)^0.5)^-1`
   Do you notice anything special about the resulting number?

Reference: Review the **MATLAB Summary** document, Table 1 for a summary of basic operators and commands.

Functions

Functions in MATLAB are analogous to functions in mathematics. They are processes which take one or more inputs and produce one or more outputs, according to some rule. In MATLAB, inputs and outputs can be all kinds of data types: numbers, vectors, matrices, etc.

For instance, try calling:

```
>> sin(1.2)
```

There is a function in MATLAB called `sin` which takes any number and produces its sine value. Any function in MATLAB is activated the same way – by calling its name, followed by the arguments (inputs) that such a function requires. You can assign the return value (output) to a variable:

```
>> y=sin(1.2)
```

In later labs, you will use functions that return multiple outputs.

If you call `help` on a function, it will tell you the possible inputs and outputs. For example, type `help meshgrid`, whose output will include the following line:

```
[X,Y] = meshgrid(xgv,ygv)
```

We will use this function in a future lab, so don't worry about understanding what it does right now. This line gives the top-level information about the function: output arguments, function name, and input arguments. This indicates that the function `meshgrid` will have two ***input*** arguments (`xgv` and `ygv`) and produce two ***output*** results (`X` and `Y`).

Built-in Functions

MATLAB contains many built-in commands for common functions and constants. See the **MATLAB Summary** document, Tables 2-3 for these functions and constants.

Do the following exercise: Evaluate these expressions using MATLAB's built-in functions.

a) `cos(pi)`
b) `sin(pi/2)`
c) `sin(45*pi/180)`
   Note: The standard trigonometric functions in MATLAB assume the angle is in radians. To use an angle in degrees, you must convert it to radians as shown here. Alternatively, in this case you could use the `sind` function.
d) `log(sqrt(exp(9)*15))`

Defining and Working with Vectors – The Dot Operator

An ***array*** is a group of values, which can have many dimensions and is essential for organizing data. A ***vector*** is an array with one column or one row (a sequence of numbers).

Start by defining a simple vector. We can use square brackets `[]` and spaces between numbers to do this:

```
>> x=[2 3 4 5 6]
```

Alternatively, you can use commas instead of spaces to separate numbers:

```
>> x=[2,3,4,5,6]
```

You can also define a vector using the colon operator (`:`), which creates numbers in an interval separated by a "step" value. For example, to create a vector of numbers from 1 to 9, increasing by 2 each time:

```
>> x2=1:2:9
```

The `linspace` (*linearly spaced vector*) command creates a vector of evenly spaced numbers (points) in a given interval. The difference between consecutive numbers depends on the number of points. For example, to create a vector of 7 numbers evenly spaced in the interval [2, 5]:

```
>> x3=linspace(2,5,7)
```

***Use the* `help` *command to read about* `linspace`.**
With vectors you can do simple mathematical operations:

```
>> y=3*x

>> y2=x+x2

>> y3=x-x2
```

Addition and subtraction operations add or subtract the individual ***components*** within the vectors. For example, the 3rd component of `y2` is the sum of the 3rd components of `x` and `x2`. Performing an operation between a constant and a vector applies the constant to each component of the vector.

Standard multiplication and division do not operate on the individual components. For example (ignore the error that MATLAB will give):

```
>> y4=x*x2
```

does not multiply the individual components of `x` and `x2`. It performs a different operation called *matrix multiplication*, which you will learn about later this term in linear algebra.

To multiply or divide the *individual components* of two vectors, you must use the ***dot operator*** (`.*` or `./`):

```
>> y4=x.*x2
>> y5=x./x2
```

Applying an exponent to the individual components of a vector also requires the dot operator (`.^`), since it multiplies a vector by another vector (itself).

```
>> y6=x.^2
```

You can use MATLAB's built-in functions to operate on vectors. Applying a function to a vector applies it to each individual component of the vector, creating a new vector of the same size:

```
>> z=sin(x)
>> z2=exp(-x)
```

Navigating the Command Window and the Importance of the Semi-Colon

The *Page Up* and *Page Down* keys allow you to quickly scroll through the command window.

The *up* and *down* arrow keys allow you to recall previous commands, which can be very helpful for making small changes to commands you have already entered. For example, use `linspace` to create a vector of 10 evenly spaced numbers in the interval [1, 6].

```
>> h=linspace(1,6,10)
```

Press the up arrow to recall this command, then change it so `h` ***has 100 points instead of 10.*** After running this command, MATLAB will output all 100 numbers of the vector. When working

with large sets of data, you will usually not want MATLAB to output all of the values.

Normally, MATLAB outputs the result after you enter each command. If you put a semicolon (`;`) after the command, it runs but the output is suppressed. ***Press the up arrow to recall the previous command, then add a semicolon at the end.*** Running it will not give any output.

The `whos` command displays a list of variables, including their sizes and data classes (types). Also notice how the current variables that exist in memory are listed in the *Workspace* window on the right.

```
>> whos
```

You will often want to clean up your workspace without having to restart MATLAB. The `clear` command removes all existing variables from the workspace, while `clc` removes all text from the command window.

```
>> clear
>> clc
```

The Engineer's Toolbox: *Discretization and Plotting*

Discretization is the process of approximating a continuous function with a discrete number of points so we can store and use this function in digital calculations. No matter whether we use 5 points or 5,000,000 points, this discrete version is ***an approximation*** of the continuous function. All results found through numeric computation are approximations to the solutions that might be found through analytic means (i.e., from first principles).

It is important to ***always*** consider the results from MATLAB with skepticism. Using your own understanding of the underlying concepts, you should always have a general idea of the expected results before using MATLAB. **Always ask yourself, "*Do these results from my numeric computations make sense*?"**

We will now use MATLAB to define and plot the function $\boldsymbol{y(x) = e^{-x}\,sin(4\pi x)}$**.** For comparison, we will first plot the decaying exponential function and the sinusoidal function on their own, then combine them to plot the final function, resulting in a figure with three functions.

This is an exponentially decaying sinusoidal function (also called a damped sine wave), an important type of function in engineering and science. It can be used to represent the settling of a robotic arm as it moves into a specific position, the "ringing" within a steel beam of a bridge after a sudden movement, or any other *underdamped* mechanical or electrical vibration.

To define this function, first create a discrete collection of values of the independent variable x. Use `linspace` to create a vector that starts at 0, ends at 5, and has 101 points.

```
>> x=linspace(0,5,101)
```

Note that we use 101 points here to get 100 intervals (differences between pairs of consecutive numbers). Using 100 points would have given 99 intervals, which is valid but more likely to have small rounding errors in the approximation. In most cases, 101 points is a good number to plot a

function.

We will define the exponential function and the sinusoidal function as separate vectors so they can be plotted individually, then combine them to get the vector of the complete function. Use semicolons to suppress output of the large vectors.

```
>> y1=exp(-x);

>> y2=sin(4*pi*x);

>> y=y1.*y2;
```

Observe how we used the dot operator to multiply the exponential function and the sine function. Recall that each function is a vector with each component corresponding to the value of the function for that component of `x`. This results in a reasonably close approximation to the actual function $y(x) = e^{-x}\sin(4\pi x)$ over this range of x.

Plot the approximation of the final function on its own:

```
>> plot(x,y);
```

Now, plot all three functions on the same figure to understand how the two separate functions combine to form the decaying sinusoidal function.

```
>> plot(x,y1,x,y2,x,y);
```

You will learn more about plotting and graphics in upcoming lab sessions, but you can see how simple it is to use the basic `plot` command.

Exercise:

Use this process to define and plot the vectors `y1`, `y2`, and `y` using 11 points instead of 101. Do this again for 5,000,001 points. ***What do you observe about both exercises?*** *Hint*: You should start with the `clear all` command, and the up arrow (↑) should be helpful.