Lab 1 - Intro to Matlab

Math 340

Lab Instructor: Connor Robertson

Due: 1/28/2022

1 How to complete and submit assignments

All labs should be submitted as Matlab "Live Scripts" exported to a PDF. This is a format that allows you to easily add text to describe your code, see plots/figures, and export everything to PDF.

Most labs will be posted on Canvas as three files: the lab assignment as a PDF (e.g. `Lab1.pdf`), a set of examples illustrating Matlab or programming principles for the lab (e.g. `Lab1_Examples.mlx`), and a template for the lab assignment submission (e.g. `Lab1_Template.mlx`). The `.mlx` file extension corresponds to Matlab live scripts (compared with traditional Matlab `.m` files or scripts).

To create a new Matlab live script, open your Matlab program and click the down arrow below the `New` plus button in the top left then click `Live Script`. This live script can then be exported at any time by clicking the `Live Editor` tab at the top of the Matlab program followed by clicking the arrow next to the `Export` button in the top left of the screen. Select `Export to PDF` and follow the instructions to generate a PDF printout of your live script file which you can submit as an assignment.

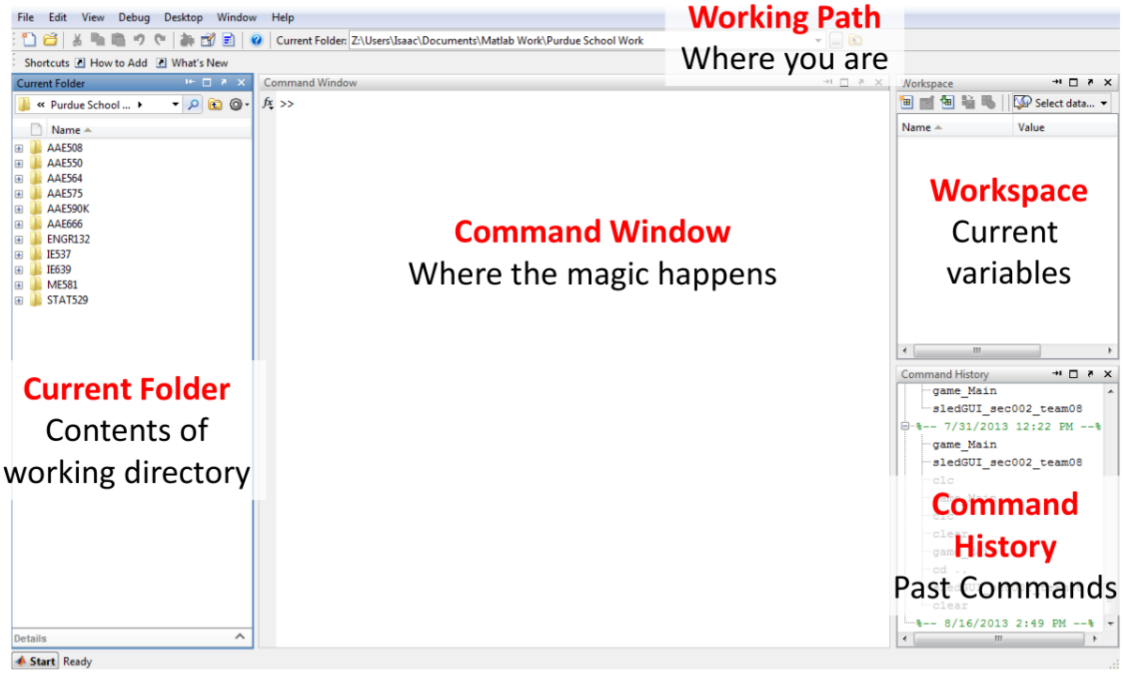
For more information on Matlab live scripts, see `https://www.mathworks.com/help/matlab/matlab_prog/create-live-scripts.html`.

2 Matlab Introduction

This assignment is meant to help introduce/review a few programming concepts in Matlab that will be used throughout the course. It is not meant to be a full introduction to Matlab, so if you need a more complete lesson or additional practice, see `https://www.mathworks.com/learn/tutorials/matlab-onramp.html`. If you feel uncomfortable with programming or Matlab, please reach out to me for help as soon as possible. Getting the fundamentals down early on will help you to complete future labs with more ease.

2.1 The Matlab Interface

We will explore the exact interface of Matlab in more detail during the lab class, but the general layout of the program can be seen in the following image:



3 Key Ideas of Programming

In this course we will approach programming from a mathematical point of view, e.g. to get some help doing math calculations. So, we first need to ask: why do we need programming for math problems? Simply put, it can help us to do long or repetitive calculations accurately and quickly. Calculations like these are common when we have many input values or when we need to perform many iterations.

A simple example of an iterative calculation like this is the factorial operation. Computing $5! = 1 \times 2 \times 3 \times 4 \times 5$ is fairly easy by hand but what if we want to compute 123! instead? Solving this would be tedious and very prone to error by hand. As a result, we focus on programming a computer to perform this computation for us. As computational procedures get more and more elaborate, computer programming becomes essential for accurate calculation.

The following subsections will explore the core ideas to most programming languages and the foundations for computational mathematics in Matlab.

3.1 Variables

In order to program a repetitive procedure, we will need a concept of "variables" that can be defined and used for calculation. These variables can have various types that your computer recognizes (you may have heard of `Ints`, `Floats`, `Strings`, etc.) but in Matlab it is sufficient to consider our variables as numbers, letters, or booleans (true/false).

For example, to define variables `x` equal to the number 10, `y` equal to the words "cool variable," and `t` to true we type in the Matlab command window:

```
x = 10
y = 'cool variable'
t = true
```

These variables are now stored in the computer and can be used for operations such as `x + 2` which would give the result 12.

3.1.1 Arrays

When programming, we often have a collection of data (numbers usually) that we want to consider. Just like variables, there are many types of data collections that our computer recognizes (usually called "data structures"). In this class, we will really only use what are called "arrays" in Matlab. They are basically ordered lists of information. For example, to define a variable `z` that is the list of the numbers $10, 27, 54, 62, 123$, I would type into the command window:

```
z = [10, 27, 54, 62, 123]
% or equivalently in Matlab
z = [10 27 54 62 123]
% lines that begin with '%' are called comments. They are just
% text in our code that doesn't get evaluated
```

Using our previous variable `x`, we can also write this as:

```
z = [x 27 54 62 123]
```

We can view, remove, change, or add elements to this array. For example:

```
% Access the 2nd element of z
z(2)
% Change the 3rd element of z to 1
z(3) = 1
% Add a 6th element equal to 12 to z
z(6) = 12
% Remove the 1st element of z
z(1) = []
```

3.2 Operations

The basic operations we will use this semester are the basic mathematical operations: add, subtract, multiply, divide, etc. These can be used on variables simply as:

```
% Set x to be x - 2 using the previous value of x
x = x - 2
% Set a new variable w that is x divided by 12
w = x / 12
% Set a new variable u that is x times x times 3
u = x*x*3
% or 3 times x squared
u = 3*(x^2)
```

These same operations work for arrays. For example:

```
% Add two lists [1 2 3] and [2 3 4] which should give [3 5 7]
[1 2 3] + [2 3 4]
```

However, if we want to multiply or divide all elements of an array `z`, we need to use special syntax (because multiply and divide two vectors or matrices can have a different meaning):

```
% Multiply all elements of [1 2 3] by 2 giving [2 4 6]
[1 2 3] .* 2
% Divide all elements of [1 2 3] by 2 giving [.5 1 1.5]
[1 2 3] ./ 2
```

It is important to enforce proper order of operations using parenthesis, just as we would on paper. For example:

```
12*2 + 1 == (12*2) + 1
12*(2 + 1) == (12*2) + 12
% double equals signs == checks if the left and right sides are equal
```

Be careful with this! I have seen many errors that students had a hard time solving because of improper use of parenthesis.

3.2.1 Functions

We can also create more complex operations called "functions" which take variables as input and usually return new variables. If the functions are simple enough, we can write these on a single line in Matlab as:

```
% Define a function f(x) = 2x+1
f = @(x) 2*x + 1
```

We can then apply the function on a variable `a` as follows:

```
a = 1
f(a)
```

If the functions are more elaborate and need several lines to write them, we can write them at the end of our Matlab file or Live Script as:

```
% Write a function g(x,y) = (x + y)^2 - 2xy - (x(y-3)^4)^2
% It takes as input variables x and y and outputs variable gxy
function [gxy] = g(x,y)
    a = (x + y)^2
    b = 2*x*y
    c = (y-3)^4
    d = (x*c)^2
    gxy = a - b - d
    % The function will return the value of gxy before the 'end' ...
       because
    % we wrote [gxy] in the function definition
end
% After the function ends, the variables a, b, c, d, gxy all disappear
```

Writing complex functions like this where variables are defined inside and combined and returned at the end will be the bread and butter of labs in this class.

3.2.2 Loops

A key piece of functions and general programming are "loops." These constructs perform repeated operations until a certain ending condition is satisfied. There are several kinds of loops, the most common being "for" loops and "while" loops. In this introduction we will only learn for loops which perform an operation for a set number of repetitions. For example:

```
% Iterate the variable i from 1 to 5
for i=[1 2 3 4 5]
    % disp means display or print, so this should print i
    disp(i)
end
% once the loop ends, the variable i disappears
```

4 Matlab Specific Examples

The section above treats some important general concepts in programming which are fundamental to be able to complete the labs in this course. All the code snippets also use proper Matlab "syntax" (basically the grammar of the language). However, there are

many Matlab specific functions and operations that build on these concepts that we will need to know.

For this lab, go through the `Lab1_Examples.mlx` live script file and carefully read the text and comments to get examples of common Matlab procedures. Feel free to add to the examples file and try your own code to make sure you understand each section. If a computation in the examples does not make sense, the best approach is to pull out paper and pencil and work through the calculation by hand to get a handle of how MATLAB works.

After carefully reading the examples file fill out the `Lab1_Template.mlx` file with the solutions to the problems below.

5 Problems

1. Look through Example 1. Here are a few examples on how to make an array of points – all of which will come into use in this course. Practice making a few arrays and matrices of your own and write them under the text `Problem 1 Answer` in the lab template.

2. Create an array, C, of the numbers 1 through 20. Use this array, C, to make a list of the squares of the first 20 positive integers (in other words, the first 20 positive integers squared).

3. Go through Example 2 and its comments. Compute the products of the two matrices:

$$A = \begin{bmatrix} 2 & 1 & 7 \\ -2 & 4 & 0 \end{bmatrix}$$

and

$$B = \begin{bmatrix} 3 & 2 \\ 4 & 1 \\ 2 & 0 \end{bmatrix}.$$

Compute `A.*B,A*B,B*A,B.*A` in MATLAB and report which ones can be evaluated and which cannot. Comment in your own words what are the differences between these products and why some of them work and some do not.

4. Examine Example 3 where we make a plot of the function $f(x) = x^3$. Follow this example to graph the function $F(x) = x^3 \sin(x)$ on the interval $(-1, 1)$. Make sure to label the x and y-axes and to title your plot.

5. Use Example 4 as a template to create a function that computes the factorial:

$$n! = n \times (n-1) \times (n-2) \times \ldots \times 1.$$

Name this function `my_factorial` and put it at the bottom of your submission live script where noted. Use this function to calculate 10! and compare it with the true answer given by the Matlab function `factorial`.

6. Use Example 5 as a template to create a function which takes in two inputs, x and N, and outputs the truncated Taylor series for the negative exponential function,

$$e^{-x} \approx 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} - \frac{x^5}{5!} + \cdots = \sum_{n=0}^{N} \frac{(-1)^n x^n}{n!}.$$

Name this function `NegExp_Series` and put it at the bottom of your submission file where noted.

(a) Using $N = 12$ and $x = 1$, approximate e^{-1} and compare your answer with Matlab's answer `exp(-1)`.

(b) Calculate an approximate remainder term for the x and N used in part a:

$$|R_N(x)| \approx \left| \frac{f^{(N+1)}(x) x^{N+1}}{(N+1)!} \right|.$$

7. Using Example 5 and the previous problem, make a function `Geometric_Series(x,N)` that accepts as input a value `x` and a number of terms in the Taylor Series `N` to evaluate the Taylor series approximation:

$$\frac{1}{1-x} \approx 1 + x + x^2 + \ldots + x^N = \sum_{n=0}^{N} x^n.$$

Output the value of your approximation for $x = 0.8$ and $N = 3, 6, 9, 12, 15$ and the absolute error between the Taylor series approximation and the true value:

$$\left| \frac{1}{1-x} - \sum_{n=0}^{N} x^n \right|.$$

What do you notice as you increase N? Do we get a good approximation for $N = 15$? Why or why not?