MATH 371
Lab 1: Introduction to MatLab

Mar. 6 & Mar. 8, 2018
TA: Alain Gervais

1. Variables

You can name a variable (constant, vector, matrix, string, ...) by giving it a name, such as `a`. Try typing the following commands in the "Command Window" (each input line has a >> symbol on its left):

```
a = 1
b = 2
c = 3
my_vect = [a,b,c]
my_vect = my_vect'
```

What do the square brackets `[]` allow you to input? What is the action of the `'` on `my_vect`? Clear the contents of the Command Window by typing `clc`. Clear the contents of the "Workspace" (which stores the values of each variable during the current MatLab session) by typing `clear all`, or clear any single variable (for example, `a`) by typing `clear a`. Now try typing the following commands in the Command Window (note the semicolons):

```
a = 1;
b = 2;
c = 3;
my_vect = [a,b,c];
my_vect = my_vect';
```

What is the action of the `;` at the end of each command? Try the following commands related to matrices:

```
A1 = [1,2,3;4,5,6;7,8,9]
A2 = ones(3,3)
A3 = zeros(2,4)
A1(2,3)
A1(1,:)
A1(:,1)
```

What does the `:` allow you to access/display on the screen in the two commands above? Observe that list/matrix indexing in MatLab begins at 1, *unlike* in many other languages, which begin indexing at 0. So a command like `A1(0,0)` will produce an error. Important fact: as a default setting, MatLab uses double-precision arithmetic. However, for ease of reading, MatLab displays numerical output with 4 decimal places. Clear the contents of the Command Window, then try typing the following commands:

```
pi
format long
pi
format short
pi
```

2. Operations: $+$, $-$, $*$, $/$, $\hat{}$
These are the four basic arithmetic operations and exponentiation. Try the following commands in the Command Window:

```
a = 2
b = 3
a + b
b - a
a*b
a^b
a/b
A2*b
A1*A1
A1.*A1
A1^2
```

What is the difference between the operations `*` and `.*` (compare the outputs of `A1*A1` and `A1.*A1`)?

3. Control flow: `if`/`elseif`/`else`, `for`, `while`
All of these control flow statements work in generally the same way as any language you may already be familiar with. MatLab knows how to block these statements thanks to the `end` command terminating each block.

```
if %conditional statement(s), e.g. if a < b
    %statement(s)
elseif %some other conditional statements(s), e.g. elseif a == b
    %statement(s)
else % handles all other cases, in our example this means the case a > b
    %statement(s)
end
for %statement, e.g. for i = 1:n
    %statement(s)
end
while %some conditional statement(s) is/are is true, e.g. while (a < b) && (x =/= 0)
    %statement(s)
    %make sure you don't have an infinite loop!
end
%Note: if multiple conditions must be tested, use && for "AND", and || for "OR"
```

4: Scripts, functions, `.m` files
You can write a "script" containing a MatLab program, or any sequence of input commands you want to save, by holding ctrl+n to open a blank script. Save your script with a name of your choice, such as `script1.m`.

1. Write a script that computes $\sum_{k=1}^{n} k$, where $n \in \mathbb{N}$, using:
   (a) a `for` loop
   (b) a `while` loop.
2. Your script should:
   (a) include a short comment describing what your script does (use `%` to comment out each comment line. This is a good practice for all your submissions in this course)

(b) clear the Workspace and the Command Window

(c) ask the user for input (try using `n = input('Enter a natural number: ');`)

(d) display the result as a complete sentence. For example if $n = 5$ your output should read something like "**The sum of the first 5 natural numbers is 15.**". Use `num2str(n)` to convert numbers to strings suitable for output. Note: the `disp()` command is used to display output. Enclose your desired output in square brackets `[]`, separating text/number segments of your sentence with commas (MatLab will interpret your command as outputting the string elements of a vector).

Functions can be built-in (for example, `sin()`, `exp()`, `norm()`, `abs()`, `plot()`, `length()`). In many cases, functions can take individual number, vector, or matrix arguments. If in doubt, Google is a good way to check what type of argument a function can take. Or, consider typing a function in the Command Window to see if some particular argument will cause an error. You can also type "`help name_of_command`" in the Command Window for information on any command/function. Try the following commands:

```
sin(pi)
cos([pi,pi/2,0])
norm(3,4,2)
norm([3,4],2)
norm([3,4],inf)
```

(Where `norm()` is the ℓ^p norm, and the second argument specifies p. If only the first argument is given, MatLab uses $p = 2$. Note that the `inf` norm is equivalent to $\max\{\cdot\}$). Often, you will find it useful to define your *own* functions! These must be defined in their own MatLab script, which must be saved as the name of the function. It is easiest to save these functions in the same folder as the script calling the function (it is possible to direct MatLab to a different directory if needed). User-defined functions always take the form

```
function output_arguments = name_of_function(input_arguments)
%statement(s): all the stuff you want your function to do
%use a local variable(s) inside name_of_function, then the last statement in
%name_of_function should pass your temporary variable(s) to output_arguments
end
```

Let's add onto **`script1.m`** (or whatever name you chose for your script earlier):

1. open a new script and define a new function called **`fact`**, which will take one input argument and return one output argument.

2. The input will be some $m_1 \in \mathbb{N}$, and the output will be some $m_2 \in \mathbb{N}$. Your function will return $m_2 = m_1!$.

3. Note: MatLab provides a function called `factorial(N)`, where $N \in \mathbb{N}$, accurate for $N \leq 21$. However, one of the aims of the Math 381 labs is to give you practice at implementing methods yourself, without relying on pre-defined functions. So, for this exercise, *do not* use the built-in factorial function. With that in mind, be sure to only use $0 \leq m_1 \leq 21$ in your factorial function. You are encouraged to keep a saved copy of all functions you define, so that you may call them in any future scripts you write for this course.

4. Ensure you include a short comment describing how your function works/what it does.

5. Lab 1 Exercise

Create and save a new script with a name of your choice, perhaps `script2.m`. This script will ask a user to input some number, and you will pass this input to a function described below:

1. open a new script and define a new function called `sine`, which will take two input arguments, and return one output argument.
2. The input will be some $x \in \mathbb{R}$, and some $m \in \mathbb{N}$. Your function will return the value of the first m terms of the Taylor series for $\sin(x)$. Recall that
$$\sin(x) = \sum_{k=0}^{\infty} \frac{(-1)^k}{(2k+1)!} x^{2k+1}.$$
3. Hint: use your function `fact.m`. What is the largest m you can safely pass from your `sine.m` to your `fact.m`? If you wish, consider adding error detection, which will check that your input to `fact.m` is acceptable.
4. Ensure you include a short comment describing how your function works/what it does.
5. What choice of x should you use to test your code? Once you know your code is working, modify your `script2.m` code so that it will call `sine.m` iteratively for a fixed x, incrementing m by 1 for each iteration, up to $\max\{m\}$. Store the results in a vector of appropriate length.
6. Plot the absolute value of the difference between your `sine.m` and the built-in `sin()`, using fixed x. You should see the difference getting smaller as the number of terms included gets bigger. Use `plot(my_list)`, where `my_list` is a vector containing the differences you calculated. You can add a title and axis labels using `title('Relevant_title')`, `xlabel('x_axis_label')`, and `ylabel('y_axis_label')`, respectively.

Looking for more of a challenge? Create another function with a name such as `sine2`, which will take two input arguments and return two output arguments:

1. The input will be some $x, T \in \mathbb{R}$, where `sine2` will approximate $\sin(x)$ as before, and T is the "tolerance": your `sine2` function will compare two successive approximations of $\sin(x)$ (i.e. the results from using k terms versus $k+1$ terms of the Taylor series expansion), and continue approximating $\sin(x)$ with enough terms such that the difference between two successive iterates is less than T (use `abs()`). The outputs will consist of the result of the approximation, and the number of terms required to achieve the specified accuracy.
2. Make sure your function has some stopping criteria in case it ends up reaching $\max\{m\}$ terms of the Taylor series (you don't want `fact.m` to get inaccurately large).
3. Ensure you include a short comment describing how your function works/what it does.
4. How many terms are required to approximate $\sin(\pi/2)$ with a tolerance of:
   (a) 10^{-2}?
   (b) 10^{-4}?
   (c) 10^{-6}?
   (d) 10^{-8}?