APPM 2460
Functions in Matlab

1 Introduction

We have seen in the previous worksheet that there are many situations in which we would want to repeat a chunk of code over and over again for different inputs. For example, you wrote a script that calculates n choose k. To test the code for different n and k you used a lot of copy and paste. Wouldn't it be nice if, instead of copy-and-pasting all that code, you could write something so that when you type `choose(n,k)` Matlab would return the value n choose k?

Such a construct exists, of course. It is called a **function**. In calculus you often think of a function as a plot, or a curve on some axes. We should now begin thinking of a function as a machine that takes in some stuff (numbers, strings, whatever) and returns some other stuff. For example, the function $f(x) = \exp(x)$ takes in a number between $-\infty$ and ∞ and returns a number between 0 and ∞. Matlab has many built-in functions, such as the `plot` function.

To write functions in Matlab, we create `.m` files that tell the function what values to take in, what commands to execute, and what output to produce. Up to now, we have only seen `.m` files used as scripts. A script executes a series of commands (i.e. lines of code) exactly as it would if these commands were typed directly into the command line. We will now see how we can also use `.m` files to create functions. The important difference between a script and a function is that **the function has no knowledge of any variables not passed in to it.**

2 Creating a function

Let's see how to write a function called `my_binom` that computes the value of $\binom{n}{k}$.

- First, open the `.m` file that you used to complete homework 3 (this script should compute the binomial coefficient $\binom{n}{k}$). Save a copy of this file and name it `bin_coeff`.
- For the file to be treated as a function, the top line must follow the below form:

  ```
  function [output_1,output_2, ...] = function_name(input_1,input_2, ...)
  ```

  We can have multiple inputs or outputs, but they must be separated by commas. In the first line of `bin_coeff`, write

  ```
  function [nCk] = bin_coeff(n,k)
  ```

  This tells Matlab that the function has the name `bin_coeff`, the inputs n and k, and the output n choose k.
- Make sure that only one copy of the code written for computing $\binom{n}{k}$ is written in the `.m` file.
- In the last line of `bin_coeff`, make sure to write `end`.

Test the new function that you wrote. In the command line, write something like

```
>>   bin_coeff(6,2)
```

Now we can use this new function `bin_coeff` as part of a larger script or function. Recall that if we want to expand binomial expressions of the form $(a+b)^n$, we can use the Binomial Theorem:

$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^{n-k} b^k, \quad n \in \mathbb{N}, \quad a, b \in \mathbb{R} \tag{1}$$

Let's create a function called `BinThm` that computes $(a+b)^n$.

- Open a new `.m` file. Using the structure

  ```
  function [output_1,output_2, ...] = function_name(input_1,input_2, ...)
  ```

  give the function the appropriate name (`BinThm`), the appropriate inputs (`a`, `b`, `n`), and some name for the output (e.g. `S`)

- Now use what we learned last week about loops to compute the right-hand side of (1).

  ```
  S = 0;                  % Start the sum at 0
  for k = 0:1:n           % These are indices of the sum
      S = S + bin_coeff(n,k)*a^(n-k)*b^k;
                          % Sub in a new 'k' value to define the next term
                          % in the sum, then add the previous terms
  end
  ```

- Be sure to save the changes made to `BinThm`, then type `BinThm(a,b,n)` for some `a`, `b`, `n` values:

  ```
  >>  BinThm(3,2,2)
  ```

  (Using the values in the example, you should have gotten the answer 25).

  Note: it is important to make sure that all of the `.m` functions/scripts that you will be working with are in the "current folder." Otherwise, Matlab will not be able to find the functions you call and will give an error message.

3 Naming Functions

It is important to realize that we **cannot** give our functions name like `plot` or `factorial`, which are already used as default functions in Matlab. We also should not name our scripts this, or use these terms as variable names. If you want to make a function that calculates a factorial, call it (e.g.) `my_factorial`, so that it does not conflict with the built-in Matlab function.

4 Anonymous Functions

Consider the function $f(x) = (x+2)^2$. If we have a one-line function like this, it may seem like overkill to create an entire `.m` file just for that one line. There is a nice way to create single-line functions within a script, and functions created in this way are referred to as *anonymous functions*.

The syntax for defining anonymous functions is

```
function_name = @(variables) ...function_definition... ;
```

Let's take a look at an example. We'll make a new **script** (not a function). In this script, we will make an anonymous function for $f(x) = 6e^x \sin(x)$ and then plot $f(x)$:

```
close all
clear all

% Here we define our anonymous function:
my_fun = @(x) 6*exp(x).*sin(x);

% Specify a range of x-values
x = 0:0.1:2*pi;
% plot my_fun against x

plot(x,my_fun(x))
```

Next week you will see anonymous functions used in the process of numerically solving differential equations.

APPM 2460 Homework 4

Submit a published pdf of your script solving the following problem to Canvas by Monday, September 24 at 11:59 p.m. See the 2460 webpage for formatting guidelines.

(a) Write a function that takes in a list of numbers and returns the maximum and minimum numbers in that list and the average of the list. (Note: You must write your own code for finding the max, min, and average. *DO NOT* use the built-in functions to find these.) To get started, your function should have the opening line

```
function [ list_max, list_min, list_av ] = my_fun( input_list )
```

(b) Once you have built this function, write a script with examples of the function running on two separate lists (each list should have at least seven elements). This script should assign the outputs of the function to variables and then display them.

An example of assigning the output of this function would look like

```
>>  [ l_max, l_min, l_av ] = my_fun([0 -1 10 2 7 -11 4])
```

After running this line, the variable `l_max` should have the value `10`, `l_min` should have the value of -11, and `l_av` should have the value 1.5714.

Make sure each variable and it corresponding value is displayed so that I know your function works.