Week 4: Functions
APPM 2460

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 `bin_coeff` 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 Week 4

Submit a published pdf of your script and any other supporting code needed to solve the following problem to Canvas by Monday, February 10 at 11:59 p.m.

The *Binomial Distribution* is a mathematical function that provides the probability of outcomes for experiments satisfying the following conditions:

- It is possible to perform n trials of the experiment.
- In each trial of the experiment there are only two possible outcomes. Call these outcomes S and F.
- The probability of S is constant from one trial to the next and is denoted p. (By the laws of probability, F has a constant probability $1-p$.)

The mathematical function that governs this probability distribution is

$$f(x) = \binom{n}{x} p^x (1-p)^{n-x}$$

1. The *expected value* of a random variable is the average of all possible values that the variable could take. To compute the expected value of a binomial random variable, we must evaluate

$$\mathbb{E}[X] = \sum_{x=0}^{N} x \cdot f(x) = \sum_{x=0}^{N} x \cdot \binom{n}{x} p^x (1-p)^{n-x}$$

   (a) Write a function called `bin_mean` that computes $\mathbb{E}[X]$. This function should take in variables n (the number of trials of the experiment), p (the probability of S), and N (the upper bound of the sum, since we cannot run the sum all the way to infinity).[1]

   (b) Evaluate `bin_mean` if $n = 100$, $p = 0.3$ and $N = 75$.

   (c) If X is a binomial random variable, we should get $\mathbb{E}[X] = n \cdot p$. Does this match the answer you found in (b)?

2. The *variance* of a random variable measures how different the outcomes of an experiment are.

   (a) One term needed to calculate variance is

$$\mathbb{E}[X^2] = \sum_{x=0}^{N} x^2 \cdot f(x) = \sum_{x=0}^{N} x^2 \cdot \binom{n}{x} p^x (1-p)^{n-x}$$

   Make a copy of your code `bin_mean`, name the copy `bin_xsq`, and adjust this code to compute $\mathbb{E}[X^2]$ instead of $\mathbb{E}[X]$.

   (b) By definition, $Var(X) = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$. Use `bin_mean` and `bin_xsq` together to compute variance if $n = 100$, $p = 0.3$ and $N = 75$.

   (c) If X is a binomial random variable, we should get $Var(X) = n \cdot p \cdot (1-p)$. Does this match the answer you found in (b)?

[1] Hint: To compute $\binom{n}{k}$ you *must* use the code you wrote for homework 3. *DO NOT* use a built-in Matlab function.