Chapter 3

Functions and Scripts

3.1 M-Files: Functions and Scripts

A **script file** is a file that includes Matlab commands. When Matlab executes a script file, it simply executes the commands that are written there as if you were typing them in.

Here's an example. Open the editor, type these lines in and save as `myscript.m`. It looks very similar to the function-

```
clear
clc
A=rand(3,2);
c=5;
H1=c+A;
H2=c*A;
H3=A-c;
temp=H1+H2+H3
```

To run the script, in the command window type `myscript`, then you can type `whos` to see what the new variables are.

Making the script nice.

In more recent editions of Matlab, they've introduced the ability to break up a script into **cells** so that you can work with smaller pieces of your code at a time. To do this, use the double percentage sign.

```
%% The double percentage sign breaks up a script into cells.

(Matlab commands here)

%% So that you can work on smaller pieces of code separately.

(Matlab commands here)
```

Also in later editions of Matlab, they've added the ability to put typesetting into a script in order to publish nicer looking documents. For example, the header of a script is what comes after the first double percent sign:

```
%% Document Header Here
%
%  Comments here

(Matlab commands here)

%% The double percentage sign breaks up a script into cells.

(Matlab commands here)

%% So that you can work on smaller pieces of code separately.

(Matlab commands here)
```

Later, we'll see how to incorporate LaTeX typesetting into a document as well.

Functions

A function has a given input (or *argument*), and produces something, its output.

Computer function differ from the mathematical definition of a function in that computer functions (and Matlab functions as well) can produce more than one output for each input. We won't worry about that- Just think of a function as a small piece of code that is designed to do a local computation.

Some functions are built-in (like sine, etc.). To extend the usefulness of Matlab, we can write our own functions. Here is a simple example that takes in two things- a constant and an array, and outputs three things- The sum of the array and the constant, the product and the difference (not a very useful function).

Open Matlab's editor (type `edit` in the command window), and type the following:

```
function [A,B,C]=myfunc(x,Z)
% [A,B,C]=myfunc(x,Z)
%  Inputs:  x is a constant, Z is an array
%  Outputs:  A=x+Z, B=x*Z and C=Z-x

A=x+Z; B=x*Z; C=Z-x;
temp=A+B+C
```

Save this file as the function name with a `.m.` suffix, or, `myfunc.m`. The line with `temp` is meaningless, but is there to show you something. But first, some things to notice about a function:

- The first line should always begin with the word "function".
- You should always include remarks that tell you how to use the function.
- In particular, note what the inputs and outputs are.

Now in the command window, we can type things like:

```
help myfunc
A=rand(3,2);
c=5;
[H1, H2, H3]=myfunc(c,A)
whos
```

Notice that the array `temp`, while computed during the execution of the function, is gone once the function is finished. Therefore, inside of a function call, you can write your variables as if nothing has been defined (except your input variables). These are called **local variables**, and functions create a small workspace that is independent of the main workspace in which to do its computations.

3.1.1 Functions "on the fly"

If you're writing some code and you want to include a small function, Matlab allows you to incorporate that into a script. These types of functions are not stored as separate $m-$files, they are defined "on the fly", and are called **anonymous functions**.

```
sqr = @(x) x.^2;
a=sqr(5)
```

Use the "at" symbol to denote the anonymous function, then define what the variable name is. In this case, we end up with a being 25.

```
sqr=@(x,y) [x.^2-y.^2, 2*x.*y];
A=sqr( [ones(3,1), [1;2;3] );
```

In this case, A will be a matrix with two columns that are computed as shown in the function definition.

```
a=3;
b=4;
MyFunc=@(x) a*x^2+b*x+1;
y=MyFunc(2);
```

This might seem strange- The function can use previously defined variables when you define the function, but they stay locked at those values until the function is re-defined. In this case, the extra parameters $a = 3$ and $b = 4$ are fixed in the function. The result is that $y = 21$.

Exercise Set

1. Write an anonymous function that will take in a vector and output the index of the maximum value. We'll call it the **argmax** function, and here is an example of what it should do:

   ```
   x=[1 3 2 1 3];
   idx=argmax(x)
   ```

   where `idx` should be a vector $[2, 5]$.

2. Write a function using an $M-$file to compute the value of $y = \text{sincX}(x)$, where we'll define the function as:

$$\text{sincX}(x) = \begin{cases} \sin(x)/x & \text{if } x \neq 0 \\ 1 & \text{if } x = 0 \end{cases}$$

(NOTE: There is a sinc(x) that is defined in Matlab- We want to write this one ourselves)

The problem is that there can be a lot of numerical error if $x \approx 0$, so for your $M-$file, we'll adjust the function to be:

$$\text{sincX}(x) = \begin{cases} \sin(x)/x & \text{if } |x| \geq 0.001 \\ 1 & \text{if } |x| < 0.001 \end{cases}$$

As a hint, see what this piece of code will do:

```
x=rand(100,1);
idx01=find(abs(x)<=0.1);
y=zeros(size(x));
y(idx01)=ones(size(idx01));
```

When you're finished, write this short driver file (as another $M-$file, using the editor) and publish it:

```
x=-15.0:0.1:15;
y=sincX(x);
plot(x,y);
title('My sinc function');
xlabel('x'); ylabel('y'); grid on;
```

3. What does this function do (think about what each line does individually; use the help features in Matlab)? The input B is an $m \times n$ matrix.

```
function A=mystery(B)

[m,n]=size(B);
Temp=sqrt(sum(B.*B));
A=B./repmat(Temp,m,1);
```

(Hint: It is often very useful to have a matrix where each column has unit length).