1 Introduction

This is an introduction to using MATLAB as a programming language. It is assumed that you are familiar with the basics of MATLAB, including defining variables, doing basic mathematical operations, creating scripts, loading data, and plotting results.

2 Conditional Statements

Like almost all other programming languages, MATLAB uses if-else statements as a fundamental building block of code. Conditional statements tell MATLAB to execute code if and only if a certain logical statement is true. The syntax in MATLAB to do this is:

There are six basic types of conditional statements. These statements can get you through 99% of cases where you need to compare values. They are:
x = 6;
if x > 3
   disp('x is greater than 3')
end

Figure 1: An if-statement

- x < a (x is less than a)
- x <= a (x is less than or equal to a)
- x > a (x is greater than a)
- x >= a (x is greater than or equal to a)
- x== a (x is exactly equal to a)
- x != a (x is not equal to a)

A common mistake that is made is writing x = a to attempt to check for equality; a single equals sign in MATLAB is an assignment, so x=a tells matlab to take the value in a, and store it as x. A second equals sign is needed to check for equality.

2.1 Else and Elseif

The else function in MATLAB will run if none of the previous if (or elseif) statements are true. Elseif will trigger if all previous if or elseif statements are not true and the condition on the elseif statement is true.

x = 3;
if x == 5
   disp('x is exactly equal to five')
elseif x == 7
   disp('x is exactly equal to seven')
else
   disp('x is not equal to five or seven')
end

Figure 2: If, Else, and Elseif-statements
2.2 Multiple Conditionals

Sometimes you may want multiple things to be true for a statement to execute, or to execute if one of multiple things is true. This is implemented using two ampersands (&&) for and; or is obtained by two vertical pipes (||). These symbols go in between the conditions.

```matlab
x = 5;
if x == 5 || x == 7
    disp('x is exactly equal to five or seven')
end
if x >=3 && x <=7
    disp ('x is between three and seven, inclusive')
end
```

Figure 3: Using multiple conditionals in the same statement

2.3 Example: Reynold’s Number

Reynold’s number is an important dimensionless number in fluid mechanics. Reynold’s number is given by:

\[
Re = \frac{\rho V d}{\mu}
\]

where \(\rho\) is the density of the fluid, \(V\) is the fluid velocity, \(d\) is the diameter of the pipe, and \(\mu\) is the dynamic viscosity of the fluid.

Reynold’s number determines, among other things, whether flow will be laminar or turbulent. Laminar flows behave very differently from turbulent flows, and require a different analytical approach. In pipes, the transition from laminar to turbulent flow occurs around a Reynolds number of 2300. Below is an example of a script that calculates Reynolds numbers, and then uses conditional statements to determine if the flow is laminar or turbulent.

3 Loops

Loops are used when you want MATLAB to perform an operation or series of operations many times. There are two main types of loops: for loops and while loops. For loops tend to be easier to use.
3.1 For Loops

A for loop runs a specified code for a specified number of iterations. For loops are typically set up as follows:

```matlab
rho = 1000;
v = 5;
d = 0.05;
mu = 1e-3;
Re = rho.*v.*d./mu
if Re <= 2300
    disp('The flow is Laminar')
else
    disp('The flow is Turbulent')
end
```

For loops begin by taking the first value in the array that you input (in our example, that is the numbers from 1 to 100), executing the code until it reaches the end that ends the for loop, and then returning to the array and moving to the next element in it until you have used every element in the array.

3.2 While Loops

A while loop executes as long as a statement is true. This is often useful if you do not know the exact number of iterations your loop needs to go through. However, while loops are very prone to getting stuck in infinite loops. Infinite loops rarely occur in for loops. Additionally, it is often hard to tell the difference between a functioning while loop and an infinite one. Ctrl+c is the break command in MATLAB, and allows you to terminate a program that has gone into an infinite loop.
3.3 Counters and Storing Values between Iterations

We are often interested in the value of a variable in every iteration of our loop. There are many ways to store values between iterations. One way is to store the values in a matrix. Adapting the previous for loop so that it stores all of the x values that we obtain in a matrix, we get:

```matlab
x = 0;
xstorage = [x];
for i = 1:100;
    x = x+3;
    xstorage = [xstorage x];
end
```

Figure 6: This loop will count by threes, and store all of the counts in xstorage

The line `xstorage = [xstorage x]` simply tells Matlab to concatenate x onto the end of the xstorage matrix, and store the result as xstorage again.

3.4 Example: Fibonacci Numbers

The fibonacci sequence is a sequence of numbers that begins with 1, 1, and continues with the next number as the sum of the two numbers that came before it. Try to write on your own a script that will calculate the first 50 Fibonacci numbers. My solution is on the next page.
Figure 7: A script that will calculate the first 50 Fibonacci Numbers

An example of when a while loop would be better is if we wanted all of the Fibonacci numbers that were less than 1000. It is difficult to calculate the number of iterations that will take, but we can simply place the constraint in a while loop and let MATLAB figure it out. This can be done as follows:

```matlab
TwoBefore = 1; %This is the number two before the Fibonacci number we are currently calculating OneBefore = 1; %This is the number one before the Fibonacci number we are currently calculating Fibonacci = [1 1]; %This is where I will store the Fibonacci numbers as they are calculated for i = 3:50 %Going from 3 to 50 because I need the third through 50th Fibonacci number NewNumber = TwoBefore+OneBefore; %Calculates new Fibonacci number Fibonacci = [Fibonacci NewNumber]; %Storing the newly calculated number.
TwoBefore = OneBefore; %Resetting the numbers for the next iteration OneBefore = NewNumber; end
```

Figure 8: A script that will calculate all Fibonacci Numbers below 1000

4 Adapting Existing Code

You will very often find that you do not know how to solve a problem or do a task in MATLAB. Fortunately, MATLAB is a popular programming language, and most of these issues have been encountered before. Unfortunately, these examples are often messy and poorly done. To help you wade through them, remember that the help command is your friend. If you are unsure of what a command does, type help CommandName into the command window, and MATLAB will pop up with a helpful description of the command, usually with
examples. Also, do not be afraid of trying to solve a problem in an unusual way. There are millions of ways to code any problem.

5 Other Useful Things

- **size** - This tells you how big a matrix is. If A is a 4 by 6 matrix, size(A) returns [4 6]

- **max** - returns the maximum element in a vector, or the maximum element of each column in a matrix. The min function works similarly.

- **find** - Finds the elements in a matrix that satisfy a condition.

- **sort** - sorts a vector from smallest to largest.

- Pulling elements out of a vector: If x is a vector with 5 elements, x(4) will return the fourth element. If y is a 5x5 matrix, y(2,3) will return the second column, third row of y. You can replace any of these with a colon to specify all elements.

6 Best Practices

- Comment your code! This makes it readable to you, your boss or grader, as well as to the person who looks at your code in six months wondering how you did that. Most of the problems with online code stem from a lack of commenting. Do not be a part of this problem. The CS department has a furnace that they use for people who don’t comment their code, and they said I could use it.

- Begin every script with the commandsclc, clear, and close all. clc clears the screen, clear deletes all the variables in the workspace, and close all closes all open graphs. This ensures that you are starting with a clean slate.