Introduction to Programming in MATLAB

Summary

This lab will introduce you to the basic concepts of computer programming, including conditionals, loops, and functions through examples in MATLAB.

A. Math
B. Variables
C. Vectors and Matrices
D. Scripts
E. Conditionals
F. Loops
G. Functions
H. FizzBuzz

Required equipment

1) Matlab

Part A: Math

Matlab makes an excellent calculator. Try entering some basic math expressions at the prompt in the command window:

```
2 + 2
3 * 5
4 ^ 2
sqrt(9)
sin(pi/2)
mod(7,3)
```

You may not be familiar with the last operation on that list. It is called modulus, or mod for short, and gives the remainder when the first number is divided by the second. Thus, 7 mod 3 would be 1, 4 mod 2 would be 0, etc. The modulus operator proves useful in several applications in computer programming.

Part B: Variables

Notice how when you enter an expression into MATLAB, it displays the results as something like this:

```
>> 2+2
ans =
   4
```

Whenever MATLAB performs an operation that returns a value, it automatically stores the result in a variable called “ans.” You should also see “ans” in the Workspace pane in the MATLAB window (it is usually in the top right). Try the following commands:

```
2 + 2
ans * 5
```

Notice how you can use the “ans” variable the same as if you had typed its value into the command instead. Of course, you are not limited to just the “ans” variable. You can create your own as well and give them any name you like. For example:

```
x = 4
y = 2 * x + 3
i = 10
Dog = 'Spot'
```

Notice how variables can store more than just numbers. They can also store text, as shown in the last example. Text, or a variable storing text, is referred to as a string. A string is really just a one-dimensional matrix of characters.

Part C: Vectors and Matrices

One of MATLAB’s strongest features is how easy it makes dealing with matrices. Matrices are entered into MATLAB as numbers surrounded by square brackets. Semicolons separate the rows of the matrix.

```
a = [ 1 2 ]
b = [ 1; 2 ]
C = [ 1 2; 3 4 ]
```

Individual elements of the matrix can be accessed like this:

```
a(1)
C(1,2)
```

Note that for a one-dimensional matrix (or vector), you only need one index in the parentheses. But for a two-dimensional matrix, you specify the index as row, column.

Matrix math and operations are easy as well.

```
d = C * b
inv(C)
transpose(a)
```

In addition to normal mathematical operations on matrices, you can also do element-wise operations. For example, this multiplies the first element of A by the first element of B, the second element of A by the second element of B, etc.

```
A = [ 1 2 3 ]
B = [ 4 5 6 ]
A .* B
```

You can also get the dimensions of a matrix with the size function:

```
[rows, columns] = size(A)
```

Part D: Scripts

So far, we have been entering every command in one by one at the prompt. This is fine for the simple examples so far, but when actually doing something non-trivial, you will need to create a script. From the file menu, go to New->Script. Name it whatever you want. Copy some of the code from parts A through C into the file then click the “Save

and Run" button (green arrow icon) in the toolbar. Notice how, in the command window, the output of every statement is displayed. Running commands from a script is the same as typing them in one by one, except in the script there is no delay between commands.

Part E: Conditionals

Conditional statements form the logic of a program. Nearly every program you write will have some kind of conditional in it. The simplest is the "if" statement. For example, to simulate a coin flip:

```
flip = randi(2);  % Random number either 1 or 2
if flip == 1
    disp('Heads')
end
if flip == 2
    disp('Tails');
end
```

Copy and paste this code into a script and run it. The first line assigns a random integer, either 1 or 2, to the flip variable. The next line checks to see if flip is equal to 1. If so, it displays "Heads." The "flip == 1" is the "condition" for the first if statement. If the condition is true, all code between the condition and the next "end" statement is executed. Otherwise, MATLAB skips to the end statement.

Note that "==" is used to check if two values (flip and 1 in this case) are equal. This is called the equality operator. Do not confuse this with the assignment operator (=), which is used to assign values to variables. Equality is not the only condition that can be checked in an if statement. There are also:

```
==   Equal to
~=   Not equal to
<    Less than
>    Greater than
<=   Less than or equal to
>=   Greater than or equal to
```

Notice the first line of the example above. All text following a % sign is referred to as "comments." MATLAB skips over comments when running scripts. Thus, you can add any comments you want to your scripts, which is often helpful for explaining what a segment of code does.

The example above is not the only way to write this code. The flip variable will always contain either 1 or 2. So if the first if statement is not true, then the second must be. Thus, we can rewrite the example above as:

```
flip = randi(2);  % Random number either 1 or 2
if flip == 1
    disp('Heads')
else
    disp('Tails');
end
```

Before we said that if the condition of an if statement is not true, MATLAB will skip to the next “end” statement. However, in the case of an if/else, if the condition is true, code between the if and else are executed. If the condition is false, code between the else and end is executed.

Conditionals with Strings
Strings cannot be compared using the same operators as numbers. If you try it, you will get something like the following:

```
>> 'Dog' == 'Dog'
ans =
     1     1     1
>> 'Cat' == 'Rat'
ans =
     0     1     1
```

Each character of the first string is compared to the corresponding character in the second string. If you try to compare strings of different lengths, you will get an error.

Instead, you must use the string comparison functions built in to MATLAB. Use the strcmp function to test if two strings are the same:

```
strcmp(string1, string2)
```

This will return true (1) if the strings are the same and false (0) if they are different.

Nested Conditionals
Conditionals can also be nested within each other. For example:

```
if A
   % Code here run if A is true
else
   if B
      % Code here run if A is false and B is true
   else
      % Code here run if both A and B are false
   end
end
```

The “elsif” statement can be used as shorthand for nested conditionals. The following is equivalent to the above:

```
if A
   % Code here run if A is true
elsif B
   % Code here run if A is false and B is true
else
   % Code here run if both A and B are false
end
```

Multiple Conditions
An if statement can also check more than one condition at a time. The example below generates a random number on a range of 1 to n and checks to see if the number is in the middle one third of the range.

```
n = 100;
r = randi(100);
if r >= n/3 && r <= 2*n/3
   disp('Middle');
end
```

The “&&” in the condition is an “and” statement. In this case, the condition is only true if both “r >= n/3” and “r <= 2*n/3” are both true.

Likewise, an “or” statement (||) can be used to check if the number is outside the middle third.

```
n = 100;
r = randi(100);
if r < n/3 || r > 2*n/3
   disp('Outside');
end
```

As usual though, there is more than one way to do this. If the number is outside the middle third, it is not in the middle third. Just like that last sentence says the same thing two different ways, there are multiple ways to say it in code. In this case, using the “not” operator (~).

```
n = 100;
r = randi(100);
if ~ ( r >= n/3 && r <= 2*n/3 )
   disp('Outside');
end
```

The not operator simply inverts the truth-value of a condition. True becomes false and false becomes true.

Part F: Loops

Loops are another concept that make an appearance in 99% of programs. The basic idea is simple; a loop executes a block of code while a condition is true. This type of loop is called a “while” loop. For example, to create a 1x10 vector and fill it with values from 1 to 10:

```
v = zeros(1,10);
i = 1;
while i <= 10
   v(i) = i;
   i = i + 1;
end
v
```

Notice that the condition of the while loop depends on the variable i and also that i is updated inside the loop. This should almost always be the case. If no part of the condition is updated within the loop then once the loop starts it will never end. This is called an infinite loop. For example:

```
i = 1;
while i <= 10
   disp(i);
end
```

If you run this, you will notice the MATLAB prompt disappears and "Busy" is displayed on the status bar at the bottom of the screen. If you do accidentally (or intentionally) create an infinite loop, you can manually interrupt it by pressing Control+C. After doing this, you should see a message: "Operation terminated by user."

You can use infinite loops in programs. However, if you do, you should make sure there is some condition that "breaks" the loop. For example:

```
v = zeros(1,10);
i = 1;
while true
   v(i) = i;
   i = i + 1;
   if i > 10
      break
   end
end
v
```

This is an infinite loop because the condition is simply "true." True is always true. Thus, the loop goes on forever. The "break" statement causes MATLAB to skip to the loop's "end" statement.

For Loops

A "for" loop is a special case of a while loop. It is basically short hand for a loop that iterates over a range of values. The above example could be rewritten as:

```
v = zeros(1,10);
for i = 1:10
   v(i) = i;
end
v
```

This has the advantage of being easier to read and having fewer lines to code.

Part G: Functions

Another key concept is the function. Functions allow you to split code into smaller segments that are each easily understood. For example, to calculate distance between two points (a point here being a 1x2 matrix):

```
function d = distance( pt1, pt2 )
   d = sqrt( (pt2(1) - pt1(1))^2 + (pt2(2) - pt1(2))^2 );
end
```

The first line is the function declaration. It contains the name of the function (distance), the return value (or values) (d), and the arguments (pt1 and pt2). Now when you run the command:

```
n = distance([3 4],[0 0])
```

Matlab will run your distance function, assigning [3 4] to pt1 and [0 0] to pt2. The return value, d, will be stored in the variable n.

Note that in MATLAB, all functions must be in their own script file and the name of the file must be the same as the name of the function.

Part H: FizzBuzz

Time to put all your newfound knowledge to the test. FizzBuzz has been used as an interview question to test basic programming ability. The task is to write a function that accepts a variable n and prints numbers from 1 to n. However, if a number is divisible by 3, the function should print 'Fizz' instead of the number. If a number is divisible by 5, 'Buzz' should be printed instead. And if the number is divisible by both 3 and 5, 'FizzBuzz' should be printed.