EGR 111
Conditional Execution

This lab shows how to write a program that performs differently depending on a given condition or user input.

New MATLAB Commands: if, elseif, else, end, input, disp

1. User Input

Sometimes a program needs to prompt the user for information. Consider the following command.

```
>> x = input('Enter a number: ')
Enter a number: 5
x =
     5
```

The input to the `input` command is a string, which in the command above is `'Enter a number: '`. Recall that in MATLAB strings are enclosed in single quotes. When MATLAB executes the `input` command, it prints the string to the Command Window and waits for the user to type a value and hit the Enter key. In the above example, the user typed "5" and then hit Enter. MATLAB then placed the supplied value into the variable x. Since there is no semicolon after the `input` command to suppress printing, MATLAB also printed the value of x to the command window.

Exercise 1: Write a script file that prompts the user for two values, adds them, and prints the result to the Command Window.

Checkpoint 1: Show the instructor your program from Exercise 1.

2. Printing the Value of Variables

If we want to print the value of a variable, we simply type it into the Command Window:

```
>> x = 5;
>> x
x =
     5
```

The `disp` function is similar to typing a variable name in that it displays the value of the variable, but it does not print the name of the variable. For example, if we wanted to be more descriptive than the previous example, we could use the `disp` function to print a string as follows.

```
>> x = 5;
>> disp('The value of x is:'); x
The value of x is:
x =
     5
```

If you don't want MATLAB to print the name of the variable, you can use the `disp` function with x as the input as well:

```
>> x = 5;
>> disp('The value of x is:'); disp(x)
The value of x is:
     5
```

MATLAB always starts a new line after the `disp` function, so you can't get the value of x to print on the same line as the string using the `disp` function. However, MATLAB also has a function called `fprintf` that allows you to determine exactly how data is printed. We won't need the `fprintf` function for this course, but if you want to see how it works, read the help file for `fprintf`.

3. Conditional Statements

Thus far in this course, we have only seen MATLAB scripts that execute the same every time. Each time you have run a script, MATLAB executed the same commands in the same order. However, many programs need to execute a set of commands only if a given condition exists, or need to execute one set of commands if a given condition exists and another set of commands if the condition does not exist. There are three structures to handle this situation: if-end, if-else-end, and if-elseif-else-end. These structures are discussed in turn.

3.1 The if-end structure

Let's write a script that selects a number between 1 and 10 and prompts the user to guess the number. If the user gets the number correct, the program should print a congratulatory message. Copy or type the following commands into a new script file.

```
a = 4;
x = input('Guess a number between 1 and 10: ');
if x == a
    disp('You guessed it!')
end
```

In the script above, the first line selects a number (any number could have been selected). The second line prompts the user for a value and places the value into the variable x. The third line is an `if` statement which evaluates the relational operator `x == a`. If the comparison `x == a` is

true, then the `disp` command that comes after the `if` statement is executed. If the user guesses the wrong number (that is if `x == a` is false), nothing is printed.

The `if` command is not limited to just one conditional command; there could be any number of commands as shown below.

```
a = 4;
x = input('Guess a number between 1 and 10: ');
if x == a
    disp('Congratulations!')
    disp('You guessed it!')
    disp('Yea!')
end
```

The `end` statement marks the end of the commands that are executed if the comparison `x == a` is true. Also, note that it is good programming practice to indent the commands that are inside the `if-end` statement to make the program easier for humans to read, but MATLAB does not care if the commands are indented or not.

Run the above program a few times to verify the operation of the script.

3.2 The if-else-end structure

In the previous example, if the user guessed incorrectly, the program printed nothing which might be confusing to the user. So let's change the program so that it tells the user if they guessed wrong.

```
a = 4;
x = input('Guess a number between 1 and 10: ');
if x == a
    disp('You guessed it!')
else
    disp('Nope.')
end
```

In the script above, the command `disp('You guessed it!')` is executed if `x == a` is true, and the other command `disp('Nope.')` is executed if `x == a` is false. The if-else-end structure is used when one set of commands needs to be executed when a condition is true, and a different set of commands needs to be executed when the condition is false. Run the above script a few time to verify how it works.

3.3 The if-elseif-else-end structure

Next, let's tell the user if he or she is close to the correct number as follows.

```
a = 4;
x = input('Guess a number between 1 and 10: ');
if x == a
    disp('You guessed it!')
elseif abs(x-a) <= 1
    disp('You are very close.')
else
    disp('Nope.')
end
```

In the script above, if `x == a` is true, the string `'You guessed it!'` will be printed. Otherwise, if `x == a` is false, then the comparison `abs(x-a) <= 1` is computed, and if it is true, then MATAB will print the string `'You are very close.'`. Note that if the user guessed 3 or 5, then `abs(x-a)` will be 1, so the string `'You are very close.'` will be printed. If `x == a` is false, and `abs(x-a) <= 1` is false, then `'Nope.'` will be printed. Run the script above a few times to verify that it works as expected.

In the if-elseif-else-end structure, there can be multiple elseif statements, and the else statement is optional.

Exercise 2: Modify the script above so that it prints 'You are pretty close.' if the user guesses either 2 or 6.

Exercise 3: Modify the script above so that it prints 'Too high', 'Too low', or 'Just right' if the users guess is too high, too low, or correct respectively.

Checkpoint 2: Show the instructor your scripts for Exercises 2 and 3.