ECE 8 - Lab 2: Basic MATLAB Use (Part 2)

October 18, 2022

1 Introduction

In this lab you will be introduced to some commonly used *control flow* routines and how to implement them in MATLAB. *Control flow* is the order in which a computer or automaton executes statements in an algorithm and provides additional structure and logic to our programs.

1.1 Background Reading

1.2 Necessary Files, Tools, and Equipment

- MATLAB

1.3 Lab Objectives

By the end of this lab you should be able to:

1. Create a program that properly uses if/else statements.
2. Use a for or a while loop to perform multi-step calculations.

1.4 Deliverables

1. A `.pdf` version of your `.m` file using the `publish` function, due Friday, 10/21/2022, at 6 PM via Canvas.

1.5 Grading Rubric

This assignment consists of 10 points.

- Each exercise is worth 1.3 points.
- Properly commented code is worth 0.3 points per exercise.

If/Else statements

One of the main tools used by programmers is the if/else statement. The if/else statement is a simple way to direct the behavior of your program based on the truth or *boolean* value of some condition. The basic syntax of the if/else statement is as follows:

```
if (condition)
    some action
elseif (condition)
    some action
else
    some action
end
```

You can read the syntax of the if/else statement in the same way you would tell someone of some action you might perform. For example, “If it is sunny out, then I will go to the beach.” If the condition “it is sunny out” is true, then I will perform the action of going to the beach. The elseif and else parts of the statement are optional and are only executed if the initial condition is false. Multiple elseif parts can be included in an if/else statement as needed. The condition(s) for an if/else statement can be any relationship or a series of relationships with an answer of true or false. In the context of our robots, the conditions used are typically based on comparing information from one of the sensors to a known value. For programming in general, relational operators are used to create conditions. The relational operators in MATLAB are:

Relational Operator Symbol Definition	
==	Equal
∼=	Not Equal
>	Greater Than
<	Less Than
>=	Greater Than or Equal To
<=	Less Than or Equal To

Below is an example of a working if/else statement. The command `fprintf` is used to display text on the screen. You can learn how the `fprintf` command works by using the `help` command.

```
grade = 0.8;
if (grade >= 0.6)
    fprintf(‘You passed.’  \n)
else
    fprintf(‘You failed.’  \n)
end
```

What message will this if/else statement display on the screen?

Switch statements

Another tool used by programmers, similar to the if/else statement, is the switch statement. The switch statement is typically used when we want to check a single condition but have multiple statement actions that depend on the value of the condition[1]. The basic syntax of the `switch` statement is as follows:

```
switch (condition)
  case 1
    some action
  case 2
    some action
  case 3
    some action
end
```

Below is an example of a working `switch` statement.

```
grade = ‘B’;
switch(grade)
  case ‘A’
    fprintf(‘Excellent!’  \n);
  case ‘B’
    fprintf(‘Well done’ \n);
  case ‘C’
    fprintf(‘Well done’ \n);
  case ‘D’
    fprintf(‘You passed’ \n);
  case ‘F’
    fprintf(‘Better try again’ \n);
  otherwise
    fprintf(‘Invalid grade’ \n);
end
```

Loops

Sometimes MATLAB coding becomes tedious and inefficient. If the program you are creating performs the same action over and over again, you can use loops to simplify your code, make it easier to debug and save time. For example, suppose you want to calculate 100 data points for one equation. Instead of writing 100 lines of code that all perform the same action, you can use a loop with only a few lines of code and get the same result with less effort.

[1]This begs the question, when should I use a switch statement or a long if/else if/else block? A good rule of thumb is when you have over 5 cases, its better to use a switch statement because *most likely* the compiler will incorporate the cases into a hash-map for faster query time

`For` loops

`For`-loops are a common type of loop where the repeated code statements in the loop are executed a specified number of times. Below is an example of a working `for`-loop.

```
x = 0;
for i = 1:2:10
     x = x+2;
end
```

In the example above, the variable `i` is called the loop iterator. The iterator determines how many times the actions inside the `for`-loop are repeated. The syntax `i = 1:2:10` is used to set the starting point (1), increment size (2), and ending point (10) of the loop counter. In this example, `i` will take on the values `[1 3 5 7 9]` one after another and the `for`-loop will repeat the action `x = x + 2` five times. What is the value of `x` after the loop is complete? What is the value of `i`?

`While` loops

A `while`-loop is a hybrid between a `for`-loop and an `if/else` statement. A `while`-loop is like a `for`-loop in that it performs a given action multiple times. Unlike a `for`-loop, a `while`-loop does not use a loop iterator to determine the number of times to perform the action. Instead, a `while`-loop uses the boolean value of some condition, much like an `if/else` statement, to determine how long the action should be performed. The basic syntax of a `while`-loop is given below:

```
while (condition)
    some action
end
```

Just like an `if/else` statement, you can read the syntax the same way you would tell someone about an action you might perform. For example, "While it is sunny outside, I will stay at the beach." While the condition "it is sunny outside" is true, I will perform the action of staying at the beach. The while statement implies that when the condition becomes false, i.e. when it is not sunny outside, then I will leave the beach. Again, just like an `if/else` statement, a `while`-loop uses relational operators to determine if the condition is true. Unlike an `if/else` statement, the condition is not just evaluated once but it is evaluated after each iteration of the loop.

2 Exercises

1. Using a for-loop, calculate the summation of all the integers from 1 to 10.

2. Create a vector `x = 0:0.1:2*pi`. Using a for loop, generate ten different plots on the same figure where the first curve is `y1 = sin(x)`, the second plot is `y2 = sin(y1)`, the third curve is `y3 = sin(y2)` and so on.

3. Using an `if/else` statement like the one above, display one of three different messages depending on the value of the variable grade. If grade is greater than or equal to 0.9, display the message "You aced the course." If grade is between 0.8 and 0.9, display the message "You almost aced the course." If grade is less than or equal to 0.8, display the message "You didn't ace the course...nice try."

4. Using a `while` loop, divide 1000 by 2 until the result becomes less than 1. Count and display on the workspace the total number of iterations it takes.

5. Make a very simple program that does the following: For any number between -1 and 1, the program calculates and displays the arccosine and arcsine of the number in both degrees and radians. For example: if the number is 0, the arccosine is either 90° or $\frac{\pi}{2}$ radians, and 270° or $\frac{3\pi}{2}$ radians. To make things easier, only need to be given for angles between 0° and 180° (π).

6. You will be given a **data set** from a quadrotor drone experiment. The data can be downloaded from Canvas. It includes position in the x direction, y direction, and z direction. We are only interested in certain instances where the drone flies close to the objective point **(x: 0.0m, y: 0.0m, z: 0.7m)**, your objective is to filter the data given certain conditions, and plot the result. Note that during an experiment, data points are stored in a vector starting at the beginning of the experiment.

   (a) Filter the x data to only keep the data-points that are within +/- 0.2m, plot the x positions, as well as the value of the objective point for the x coordinate as a constant over time.

   (b) Filter the y data to only keep the data-points that are within +/- 0.2m, plot the y positions, as well as the value of the objective point for the y coordinate as a constant over time.

   (c) Filter the z data to only keep the data-points that are within +/- 0.2m, plot the z positions, as well as the value of the objective point for the z coordinate as a constant over time.

   (d) Filter the x, y, and z data when the data-points are all within +/- 0.2m on each axis, create a single figure and graph the subplots x,y and z and their respective objective points as a constant over time

   (e) Include legends, axes, and title for all plots

   Note: In total, your code should generate four figures, with the last figure including three plots in it.

Note: It's very IMPORTANT to comment your code. Please make sure to include a brief explanation of the code used to complete each of the exercises. You will lose points for no comments or poorly commented code.