Conditional Logic

Leigh J. Little SUNY Brockport September 20, 2021

Contents

1 Introduction

We have seen that it is possible to do elaborate computation with MATLAB using basic linear algebra and whole array operations. However, sooner or later you will encounter a need to do calculations that can't be performed using just these capabilities. You will need to be able to perform more fundamental programming operations. Over the next few lectures, we will examine two basic programming constructs (conditional logic and loops) that will allow you to perform nearly any calculation.

2 Programming Errors

It is a very rare case that a program of significant length is written correctly the first time. Programming errors are just a part of life, just like sign errors in mathematics. Errors that occur when you write a program are broken down into two main categories:

- **Syntax Errors** - A syntax error occurs when you type in something that MATLAB doesn't understand, for example

  ```
  >> plot(x,y
  ```

  In this case, the command is missing the right parenthesis. Another example would be

  ```
  >> x = (1;1;n)
  ```

  In this case the intention is to create an array index but the semicolon was used instead of the colon.

  Syntax errors are common but in most cases are easily diagnosed.

- **Logic Errors** - A logic error occurs when the program does something different than what you intend for it to do. Logic errors are also common, but are much more difficult to diagnose. Logic errors do not follow the law of local cause and effect. This means that the appearance of a logic error can occur 'far away' from where the error was made.

  A rough analogy would be a car. If you are driving down a road and smoke starts coming out of your engine, you would pull over and lift the hood to see where the smoke is coming from. If you have a program and smoke starts coming out from the engine, you lift the hood and everything seems fine. Further investigation reveals that the problem is that you don't have enough air in your tires.

3 Conditional Logic

One of the most useful things you can do in a program is to ask it a question. Depending on the nature of a calculation, you might want to take several different courses of action depending on the current value of one or more variables. Conditional logic (also called conditional branching or if-then logic) is what permits you to ask your program a question.

Type the following into a script called `social.m` and run it:

```
clear
n = input('Input a value for n ');

if (n >= 65)
   disp('You are eligible for social security')
else
   disp('You are not eligible for social security')
end
```

Run your script for several value of `n`. What do you observe when you run your script?

This is an example of an `if-then-else` block. The sequence of statements will produce one of two different outputs depending on the value of `n` that the user inputs. We have asked our program a question:

```
if (n >= 65)
```

then told it what to do if this statement is true

```
disp('You are eligible for social security')
```

and what to do if the statement is false

```
disp('You are not eligible for social security')
```

This is the essence of conditional logic. We ask our program questions, then take different actions based on the answers to the questions. The questions we ask must always have a yes or no (true or false) answer.

The general syntax of an `if-then-else` block is

```
if (conditional statement)
   |
   |  Block of statements to execute if statement is true
   |
else
   |
   |  Block of statements to execute if statement is false
   |
end
```

Some general notes about `if` statements:

- The () around the conditional statement are optional, but it's easier to read your code if you put them in.
- You can have any number of statements in each block.
- The bodies of the `if` blocks are indented. This aids in making your code more readable. This is optional in MATLAB (but not python), so I will require that you do this.
- The program will execute the statements in the appropriate block (depending on the result of the test), then jump out of the structure to the first executable statement past the `end` statement.
- Every `if` must have a corresponding `end`.

`>`	greater than
`>=`	greater than or equal to
`<`	less than
`<=`	less than or equal to
`==`	logical equality
`~=`	logical inequality
`&`	logical AND
`|`	logical OR
`~`	logical NOT (negation)

Table 1: Relational Operators in MATLAB.

Table 1 gives the available conditional (or relational) operators. Because there are many ways to ask a question, the social security script can be written in more than one way. For example, the code below is equivalent to the first one

```
clear
n = input('Input a value for n ');

if (n < 65)
   disp('You are not eligible for social security')
else
   disp('You are eligible for social security')
end
```

There are two other forms of `if` tests that can be performed. The first is the basic `if-then` statement

```
if (conditional statement)
   |
   |  Block of statements to execute if statement is true
   |
end
```

The second one is the `if-then-elseif` statement

```
if (conditional statement 1)
   |
   |  Block of statements to execute if statement 1 is true
   |
elseif (conditional statement 2)
   |
   |  Block of statements to execute if statement 2 is true
   |
elseif (conditional statement 3)
   |
   |  Block of statements to execute if statement 3 is true
   |
elseif(.....
   |
   |
   |
else
   |
   |  Block of statements to execute if none of the above statements are true
   |
end
```

Some comments on the `if-then-elseif` structure

- If a block of statements is taken (say the first two conditions are false and the third condition is true). Then the third block of statements is executed. The remainder of the tests in the main `if` block are ignored and the program will jump to the first executable statement past the `end` statement.
- Make sure that `elseif` is one word.
- You can have as many `elseif` clauses as you need
- The `else` is optional.
- There is only one `end` for the structure as a whole.

3.1 Example 1: Illustration of Program Flow

Consider the simple example below

```
n = 2;
if (n == 1)
   disp('n is equal to 1')
elseif(n == 2)
   disp('n is equal to 2')
elseif(n == 3)
   disp('n is equal to 3')
elseif(n == 4)
   disp('n is equal to 4')
else
   disp('n is at least 5')
end
disp('End of test')
```

Save this as `if_ex1.m`, then run the code:

```
>> if_ex1
n is equal to 2
End of test
```

Here is the same code with the sequence commented:

```
n = 2;
if (n == 1)        %% This is false, so the block below
                   %% is skipped.
   disp('n is equal to 1')
elseif(n == 2)     %% This is true, so the block below
                   %% is executed.
   disp('n is equal to 2')
                   %% All the remaing clauses are ignored.
                   %% Flow proceeds to the display command
                   %% after the end statement
elseif(n == 3)
   disp('n is equal to 3')
elseif(n == 4)
   disp('n is equal to 4')
else
   disp('n is at least 5')
end
disp('End of test')
```

3.2 Example 2: Test if a value is negative, positive or zero

If we wanted to perform a test to see if some value was negative, positive or zero, we could do the following

```
t = input('Input test value: ');
if (t > 0)
    disp('t is positive')
elseif (t < 0)
    disp('t is negative')
elseif (t == 0)
    disp('t is zero')
end
```

However, we can do this a different way. Because there are only 3 possibilities, if neither of the first 2 are true, we know the answer has to be be the third option hence we could also do

```
t = input('Input test value: ');
if (t > 0)
    disp('t is positive')
elseif (t < 0)
    disp('t is negative')
else
    disp('t is zero')
end
```

There are 10 other ways we could perform the same test (by switching the order in which the testing is done).

3.3 Example 3: Piecewise Function Evaluation

Piecewise functions are often encountered in calculations. Suppose we wanted to evaluate the function

$$y = \begin{cases} x^2 & x \geq 0 \\ x^3 - x + 5 & x < 0. \end{cases}$$

The following code would accomplish this

```
x = input('Input x: ');
if (x >= 0)
   y = x^2;
else
   y = x^3-x+5;
end
```

The code below would also work

```
x = input('Input x: ');
if (x < 0)
   y = x^3-x+5;
elseif(x >= 0)
   y = x^2;
end
```

3.4 Nesting of `if-then` Structures

We can only ask questions that have true or false answers. Some questions don't have simple yes or no answers and in such cases, the question must be broken down into a series of simple questions or must be expressed in terms of the conjunction operators (*i.e.,* `AND`, `OR` and `NOT` operators).

To allow for more complex questions, any type of `if-then` structure can be nested inside any other `if-then` structure. This means that the body of one `if` statement can itself contain an `if` statement. For example, consider evaluating a 2-dimensional piecewise function

$$z = \begin{cases} x^2 + y^2 & x \geq 0, y \geq 0 \\ x^2 - y^2 & x \geq 0, y < 0 \\ y^2 - x^2 & x < 0, y \geq 0 \\ -x^2 - y^2 & x < 0, y < 0 \end{cases}$$

This can be done using a sequence of nested `if-then-else` statements

```
x = input('Input x: ');
y = input('Input y: ');

if (x >= 0)          % If this is true, one of the first 2 equations is needed
   if (y >= 0)       % If this is true, we need the first equation
      z = x^2 + y^2
   elseif(y < 0)     % If this is true, we need the second equation
      z = x^2 - y^2
   end               % This end closes the first inner if
elseif (x < 0)       % If this is true, one of the last 2 equations is needed
   if (y >= 0)
      z = y^2 - x^2  % If this is true, we need the third equation
   elseif(y < 0)
      z = -x^2 - y^2  % If this is true, we need the fourth equation
   end               % This end closes the second inner if
end                  % This end closes the outer if
```

This could also be done using

```
x = input('Input x: ');
y = input('Input y: ');

if (x >= 0)
   if (y >= 0)
      z = x^2 + y^2
   else
      z = x^2 - y^2
   end               % This end closes the first inner if
else
   if (y >= 0)
      z = y^2 - x^2
   else
      z = -x^2 - y^2
   end               % This end closes the second inner if
end                  % This end closes the outer if
```

Some comments about nested `if-then` statements:

- Each `if` structure needs its own `end` statement.
- `if` statements that appear in a branch must logically terminate in that branch.
- There is a limit of 7 levels of nesting.

3.5 Compound Logic

The conjunction operators (`AND`, `OR` and `NOT`) can be used to build a more complex `if` statement. The tables below summarize how these are used

p	q	p AND q	p OR q
T	T	T	T
T	F	F	T
F	T	F	T
F	F	F	F

Table 2: Truth tables for AND and OR relations.

p	NOT p
T	F
F	T

Table 3: Truth table for NOT relation.

3.6 Example 4: Piecewise Function with Conjunction Operators

The previous example of a 2-dimensional piecewise function can be done using an `if-then-elseif` structure combined with the conjunction operators.

```
x = input('Input x: ');
y = input('Input y: ');

if (x >= 0 & y >= 0)
   z = x^2 + y^2;
elseif (x >= 0 & y < 0)
   z = x^2 - y^2;
elseif (x < 0 & y >= 0)
   z = y^2 - x^2;
elseif (x < 0 & y < 0)
   z = -x^2 - y^2;
end
```

Believe it or not, this is all we need to know about the mechanics and syntax of `if` statements. However, programming is like chess. There are not many rules, but it takes much practice to learn how to master them. The way to accomplish this is to write lots of programs. It is easy to write a set of `if` statements that look great, but don't do what you intended them to do. This called a *logic error*. It is important that you test a more complicated sequence of `if` statements over all the various outcomes to make sure it is producing the proper behavior.