Week 3: If, For, and While
APPM 2460

1 Introduction

Today we will learn a little more about programming. This time we will learn how to use for-loops, while-loops and if-statements.

There may be a situation where you need to execute a block of code several time in a row. For example, suppose we want to print even numbers between 2 and 20. We could write a script as follows:

```
disp(2)
disp(4)
disp(6)
...
```

and so on. You now see how tedious this would be. An easier way would be to use something called a *loop*, which is a programming tool that allows us to easily repeat commands. We will learn about two types of loops, *for* loops and *while* loops. For-loops are the most commonly used type of loop, so we will begin with those.

2 The For-loop

A for-loop repeats an operation a set number of times, usually using a variable whose value changes size each time the loop runs. For the above problem, we would want to loop through a variable that takes values 2, 4, 6, 8, and so on, up to 20, and display the variable each time we execute the loop. Let's see how to do that.

Let's give it a try. Open a new script, enter the following, and save/run it:

```
for i = 1:10
    disp(2*i)
end
```

The for-loop works in the following manner:

- We give the loop a variable (in this case i)
- We then tell that variable (1) where to start, (2) what step-size to take between numbers, and (3) where to end.
  - In this case, we start at $i = 1$ and we take steps of size 1 (by default) until $i = 10$.
- The inside of the loop tells Matlab to display the value of $2 * i$ at each iteration of the loop.

Notice the syntax here. The `for` declaration is followed by a series of statements that Matlab executes until it reaches the `end` statement. That is to say, *the commands that get repeated by Matlab are those sandwiched between the "for" and "end" statements.* Matlab runs the commands between the `for` and `end` once for each element in the vector `i=1:10`. We say that we are "looping over" the vector `i=1:10`. We could just as well use any vector. For example, enter the following into your script:

```
        for i = [1 5 3 26 9]
            disp(i)
        end
```

($\star$) Note (regarding syntax): **there is never a semicolon after the `for` declaration statement.**

2.1 Something more complicated

Let's use a for-loop to make some plots of $y = a * \sin(x)$, where $a = 0.2 * k$, $k = 1, 2, 3, 4, 5$, and x is in the interval $[0, 1]$. Let's make it so it plots on the same figure each time. Modify your script as follows:

```
    hold on
    x = 0:.05:1
    for k = 1:1:5
        a = 0.2*k;
        plot(x,a*sin(x),'LineWidth',2)
    end
    hold off
```

Any time you need to repeat something a *known number of times*, consider using a for-loop. It will save you a great deal of time and effort.

- You can put one for-loop inside another one. Just be sure to use a different counter variable for the second loop.

3 Logic Operators

Before we move on to while-loops and if-statements, we need to learn about logic operators. We start with 'and' (&) and 'or' (|).

&: The logical operator for 'and' is &. The & operator simply *takes two statements and returns the value of "true" only if both are true, and returns "false" otherwise.*

Statement 1	Statement 2	Statement 1 & Statement 2
true	true	true
true	false	false
false	true	false
false	false	false

For example, *3 is an integer & 4 is even.* This is true because both statements are true. In fact, the only case where the & operator is true is if both statements are true statements. Otherwise, it is considered false.

|: The logical operator for 'or' is | (| is right above the enter button). This operator is much nicer. The | operator also *takes two statements and returns the value "false" only if both are false, and it returns "true" otherwise.*

Statement 1	Statement 2	Statement 1 | Statement 2
true	true	true
true	false	true
false	true	true
false	false	false

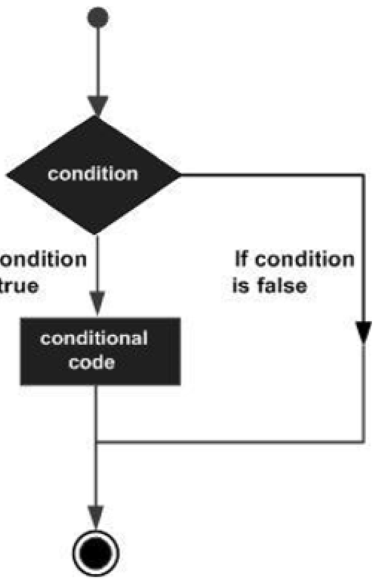
For example, *3 is larger than 4 | 4 is prime.* This is false because both statements are false. In fact, the only case where the | is false is when both statements are false statements. Otherwise, if either statement is true then the whole 'or' statement is true.

4 The If-Statement

Before, we were using a for-loop to repeat a set of instructions a given number of times. An if-statement will perform a set of instructions once the right conditions are met. The syntax is

```
if (condition goes here)
    action to be performed
end
```

A useful flow chart to visualize this is



The conditions you can put in are either equal(==), not equal (~=), less than (<), less than or equal to (<=), greater than (>), and greater than or equal to (>=).

As an example, let's write a script that does the following:

- If the value of some number a is greater than or equal to zero, then we'll print out the value. Type

```
for a = -3:1:3
    if (a >= 0) % Parenthesis are not necessary here but will be needed later
        disp(a)
    end
end
```

See how the values of a that were greater than or equal to zero were printed to the command window? Note that we have nested an if-statement inside the for-loop. You can nest if-statements and for-loops as many times as needed. We can have more than one condition in our if-statements. *If we want two conditions to hold, we use the 'and' operator, &. If we want either one of two conditions to hold we use the 'or' operator, |.*

- Let's take the previous example and make it work not only for a greater than or equal to zero, but also less than 2. Try

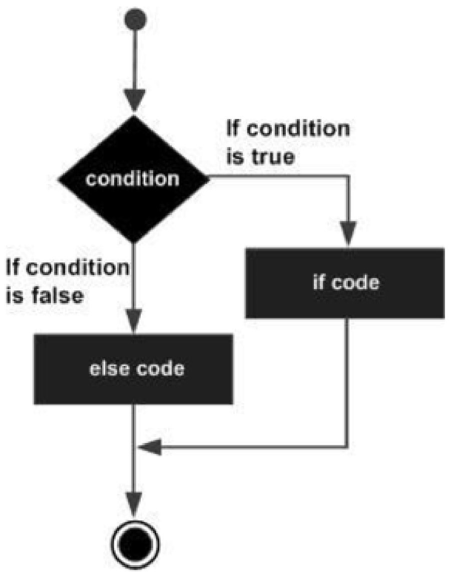
```
for a = -3:1:3
    if (a>=0)&(a<2)
        disp(a)
    end
end
```

- We can modify the if-statement so that if the condition is not met, the loop does something else. We use the else-statement for this. Let's take the previous example, and make it so that if $2 \leq a \leq 3$, then we print $4 * a$. Type

```
for a = -3:1:3
    if (a>=0)&(a<2)
        disp(a)
    else
        disp(4*a)
    end
end
```

Note: We can include a condition of the else-statement by typing `elseif` instead of `else` and then giving it a condition just like we did before.

- We can have more than one elseif-statement if needed.



So let's try a couple of examples. In both examples, we will be calculating the factorial of a number. Create two scripts, "factorial_1" and "factorial_2" which take an input value and output the factorial of that value. We must first make sure that the integer being used is positive. We must also make sure that if it is zero, we set the value equal to 1. We could do this by (Case 1) nested if-statements or (Case 2) by using just one if-statement with an `elseif`.

- Case 1: We begin with the nested if-statements.

```
a = 6;                  % or whatever number you want to calculate n! of
if (a>=0)
    f = 1;
    if (a == 0)
        return          % jump out of the loop and give the answer 0! = 1
    end                 % end if-statement
    for i = 1:a
        f = f*i;
    end                 % end for-loop
end                     %end if-statement
disp(f)                 % the factorial of a
```

  Notice, each if-statement and for-loop needs an `end` for this to work. This set-up also requires a `return` inside the nested if-statement. Otherwise, the program could still run the for-loop and may give a wrong answer.

- Case 2: We show the same process except with the `elseif`

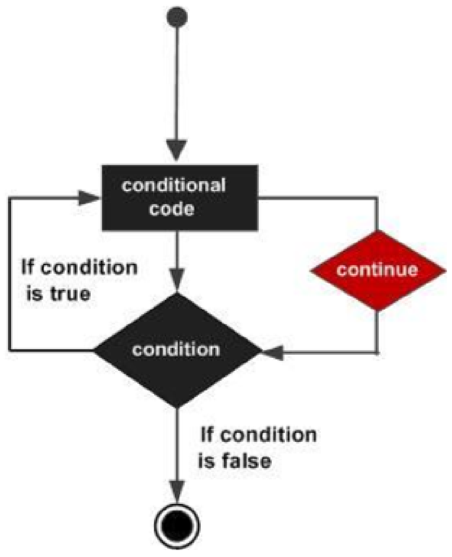
```
a = 6;          % or whatever number you want to calculate n! of
if (a<0)        % return an error message if a is negative
    disp('Negative Integer');
elseif (a == 0)
    f = 1;
else
    f = 1;
    for i = 1:a
        f = f*i;
    end         % end for-loop
end             % end if-statement
disp(f)         % the factorial of a
```

Notice, there is only a need for one end on the if-statement. There is also no need for a return call because the structure of the if-statement only allows for one outcome to occur. We do not have to worry about the for-loop being executed when $a = 0$.

5 While-loops

A while-loop is another way of repeating a statement. You use while-loops when **you want to repeat a calculation as long as a certain condition is satisfied** and/or **you are not sure how many steps the process will take**. The while-loop has a *condition statement* that it checks each time it repeats. If the condition is true, it will repeat the body of the loop. If the condition is false, it will go to the next line after the end of the loop. **Note that it is very easy to get stuck in a while-loop if your condition is always true. When this happens, your code will just run forever. Be careful of this.**

A flow diagram to help visualize this type of loop is



For example, suppose you want to double the number 2 over and over again until you get to a number over 1,000,000 and you want to see how many times you must run the loop to obtain this result. We could write a while-loop that performs that task as follows:

```
number = 2;
counter = 1;

while number <= 10^6
    number = 2*number;
    counter = counter + 1;
end
```

Submit a published pdf of your script solving the following problem to Canvas by Monday, September 14 at 11:59 p.m. See the 2460 webpage for formatting guidelines.

<u>Definition</u> The binomial coefficient $\binom{n}{k}$ (read "n choose k") is defined as

$$\binom{n}{k} = \frac{n!}{(n-k)!\;k!}$$

where the ! is the factorial symbol (e.g. $5! = 5 * 4 * 3 * 2 * 1 = 120$) and n and k are non-negative integers with $n > k$.

(a) Write a script that calculates n choose k **without using Matlab's built-in `factorial` function.** This will involve multiple loops.

- Your script should include `if`-statements that return an error (for example, `disp('Danger Will Robinson!')`) if n is less than k OR if either n or k is negative.

(b) Divide your script into three sections using the command `%%`. Copy and paste your script into each section and run the scripts (i) once in the scenario where everything works, (ii) once where $n < k$ and an error message is returned, and (iii) once where one of n or k is negative and an error message is returned.