EGR 111
Loops

This lab is an introduction to loops, which allow MATLAB to repeat commands a certain number of times.

New MATLAB commands: for, while, end, length

1. The `For` Loop

Suppose we want print a statement four times. One way would be to repeat the `disp` function four times as follows:

```
disp('Are we there yet?')
disp('Are we there yet?')
disp('Are we there yet?')
disp('Are we there yet?')
```

If we wanted to repeat a command a large number of times, it would be very inconvenient to copy the command over and over. Fortunately, the `for` command allows MATLAB to repeat a command an arbitrary number of times. Consider the following command:

```
>> for n = 1:4, disp('Are we there yet?'), end
```

The command above repeats the `disp` function four times, and keeps track of how many times it has repeated the command using the variable `n`, resulting in the following:

```
Are we there yet?
Are we there yet?
Are we there yet?
Are we there yet?
```

Now let’s change the command inside the loop to print the value of n instead of the string ‘Are we there yet’.

```
>> for n = 1:4, n, end
n =
     1
n =
     2
n =
     3
n =
     4
```

In the command above, the vector "1:4" is a **list** of values that are placed into the variable n one at a time. MATLAB begins by taking the first element from the vector 1:4 and placing it in the variable n. Since 1:4 = [1 2 3 4], the first value is 1, which is placed into n. Then the command "n" is executed, which prints the value of the variable n to the screen.

Then the second element from the vector 1:4, which is 2, is placed into n, and the command n is executed, which prints the value 2. And so on.

The `for` loop can be written on a single line if the commands are separated by commas or semicolons, as we have seen so far, but it is often easier to read if each command is on a separate line as shown below.

```
for variable = list
     commands
end
```

The `for` loop always has the same structure. The first line of a `for` loop always begins with `for,` followed by a variable name, followed by the equals sign, and followed by a **list** of values, which is a **row** vector. Next are one or more MATLAB commands, which are executed once for each value in the **list**. And finally there is an `end` statement to mark the end of the commands.

Note that it is good practice to indent the commands within a loop as shown above. Although indenting is not required by MATLAB, it is highly recommended, because it makes the code much easier for humans to read. Also do NOT put any command inside the loop that changes the loop variable because MATLAB places values from the **list** into that variable.

The **list** can be any vector. For example, consider the following command:

```
>> for n = [2 7 5 3], n, end
n =

    2

n =

    7

n =

    5

n =

    3
```

The `for` command above begins by taking the first element from the **list**, which is 2, and places it into the variable n. Then it executes the command inside the loop, which prints out the value of n. Then, it places the second element from the list into the variable n, and repeats the command. And so on for each of the elements in the **list**.

Examine the commands below and determine what will be printed:

```
>> Q = [3 9 8 2];
>> for m = 4:-1:1,  Q(m), end
```

Which value will be printed out first? Which will be printed second? Write down your answers and then check them by running the commands in MATLAB.

In the command above, the **list** is 4:-1:1 = [4 3 2 1]. So MATLAB begins by taking the first value from the **list**, which is 4, and places it into the variable m. Then it executes the command inside the loop, which is “Q(m)”. Since m = 4, Q(m) = Q(4), which is the 4th element in the vector Q, which is 2, so MATLAB will print 2. Then MATLAB takes the second element form the **list**, which is 3, and places it into m. Then it prints the value of Q(m) = Q(3) = 8. And so on until the list is exhausted.

Notice that every `for` command requires a corresponding `end`. Also notice that commas (or semi-colons) can be used to separate commands on the same line. Semi-colons will suppress the printing of output to the screen.

Next, let's look at the problem of computing the sum of a list of numbers, say 10, 20, 30 and 40. One way to compute this sum would be to have MATLAB add the numbers directly as follows:

```
s = 10 + 20 + 30 + 40
```

The method above would be tedious if we wanted to add a lot of numbers, so let’s see how we could accomplish this task using a loop. (There is an easy way to do this task without using a loop, but let’s use a loop anyway.) Consider the following script file:

```
clear                   % clear variables
v = [10 20 30 40]       % store the numbers in a vector
s = 0;                  % initialize s to 0
for n = 1:length(v)     % loop using n = 1, 2, 3, 4
    s = s + v(n);       % add v(n) to s and store result in s
end                     % end for statement
s                       % print the value of s
```

Note that the `length` command returns the number of elements in a vector, so in the above script `length(v)` returns 4. Also recall that `v(n)` accesses the nth element of the vector `v`. So `v(1)` is 10, `v(2)` is 20, and so on.

Before the loop, the value of s needs to be initialized to zero. If s were not initialized, the command `s + v(n)` would cause an error because the variable `s` would be undefined the first time through the loop.

The first time the loop is executed, the value of `n` is set to 1. The command `s + v(n)` is computed, which results in 0 + 10 = 10, and this value is stored in the variable s.

The second time the loop is executed, the value of `n` is set to 2. The command `s + v(n)` results in 10+20=30, and again this value is stored in the variable `s`. And so on. In this way, the numbers in vector `v` are added.

So the program above is equivalent to the commands below (you do NOT need to type the commands below, they are just to explain how the loop above works).

```
clear
v = [10 20 30 40]
s = 0;
n = 1;
s = s + v(n);    % s = 0 + 10 = 10
n = 2;
s = s + v(n);    % s = 10 + 20 = 30
n = 3;
s = s + v(n);    % s = 30 + 30 = 60
n = 4;
s = s + v(n);    % s = 60 + 40 = 100
s
```

Exercise 1: Write a loop to sum the <u>even</u> numbers from 2 to 100 (that is find 2 + 4 + 8 + … + 100). There is an easy way to do this without a loop, but let's use a loop here because we are practicing loops.

2. Nested For Loops

Since the `for` loop allows MATLAB to repeat any MATLAB command, what would happen if we put a `for` loop inside another `for` loop? We call a structure like this a **nested loop**. Consider the script file shown below.

```
clear
disp('     i     j')
for i = 1:3
    for j = 1:2
        disp([i j])
    end % end for j
end     % end for i
```

In the script above, the first `end` statement marks the end of the commands in the inner loop "`for j = 1:2`". The second `end` statement marks the end of the commands in the outer loop "`for i = 1:3`". It is good programming practice to indent the `end` statement the same amount as the corresponding `for` command to make the program easier for humans to read, but MATLAB does not require this indentation.

First try to figure out what MATLAB will print to the Command Window when the script above is run, then run the script and see if you were correct.

In the script above, the first `for` loop causes the second `for` loop to be repeated three times, first with i = 1, then with i = 2, and then with i = 3. The second `for` loop causes the `disp` command to be repeated twice, first with j = 1, and then with j = 2.

So the script above is equivalent to the following commands (you do NOT need to type the commands below, they are just to explain how the loop above works).

```
clear
disp('     i     j')

i = 1;
j = 1;
disp([i j])
j = 2;
disp([i j])

i = 2;
j = 1;
disp([i j])
j = 2;
disp([i j])
```

```
i = 3;
j = 1;
disp([i j])
j = 2;
disp([i j])
```

In the same way that a single for loop can be used to access each element of a vector, nested for loops can be used to access the elements of a matrix. For example, the script below uses a nested loop to set all of the elements of a matrix to one.

```
A = zeros(3,2)
for i = 1:3
    for j = 1:2
        A(i,j) = 1
    end
end
```

In the script above, the `zeros` command creates a 3x2 matrix, sets all of the elements to zero, and saves the result in a matrix called A. Then the nested loop accesses each element one at a time and changes each element to one. The variable i is used to store the row number, and the variable j is used to store the column number. The script above is equivalent to the following commands (you do NOT need to type the commands below, they are just to explain how the loop above works).

```
A = zeros(3,2)
i = 1;
j = 1;
A(i,j) = 1
j = 2;
A(i,j) = 1

i = 2;
j = 1;
A(i,j) = 1
j = 2;
A(i,j) = 1

i = 3;
j = 1;
A(i,j) = 1
j = 2;
A(i,j) = 1
```

Exercise 2: Write a script that generates a 5x5 matrix of zeros and save it in variable A. Then, using a nested loop, check to see if the row number is greater than the column number, and if so, set the value of that element of A to 1. If the row number is NOT greater than the column number, leave the value of the element unchanged. Print out the final matrix. The result is shown below.

```
A =
     0     0     0     0     0
     1     0     0     0     0
     1     1     0     0     0
     1     1     1     0     0
     1     1     1     1     0
```

Checkpoint 1: Show the instructor your script file and results from Exercises 2.

3. The While Loop

`For` loops are mainly used when the number of times the loop will be repeated is known in advance, which was the case in all of the preceding examples. The `while` loop is often used when the number of times the loop will repeat is not known in advance. For example, consider the following program.

```
clear
x = 1
while x == 1
    x = input('To keep doing this, type 1: ');
end
```

First the variable x is set to 1. Then the while command checks to see if `x == 1` is true, and if so it executes the input command to prompt the user for a value and save that value in x. Then it repeats. So the loop will keep repeating an unknown number of times until the user types something other than 1. Run the program to verify that works.

A Common Error: Note that if the command inside of the loop does not modify the variable that is being tested, the loop will never terminate. We call that an infinite loop. In the script file below, the user's guess is stored in the variable g instead of x. Since x is never changed, this loop will keep repeating regardless of the value the user types. Note, however, that you can force a program to quit by holding down the Ctrl key and pressing c (Ctrl-c).

```
clear
x = 1
while x == 1   % Oops, this is an infinite loop!
    g = input('To keep doing this, type 1: ');
end
```

Let's write a program to play a guessing game where MATLAB picks a number between 1 and 10, and repeatedly prompts the user until the user guesses correctly.

```
clear
x = randi(10);  % generate an integer between 1 and 10
g = 0;  % initialize the guess to zero
while g ~= x
    g = input('Guess what it is: ');
end
disp('Yep')
```

In the program above, the `randi` function generates a pseudorandom integer from 1 to 10. (Pseudorandom numbers have the properties of random numbers, but are completely predictable if you know the algorithm used to generate them.) The variable `g` will hold the user's guess and is initialized to 0 so that it cannot be equal to the pseudorandom number. The `while` statement checks to see if `g ~= x` is true, and if so, it will prompt the user for another guess. This process repeats until the user guesses the correct number (or the user forces the program to quit using Ctrl-C).

Exercise 3: Modify the program above to tell the user whether the guess is too high or too low.

Checkpoint 2: Show the instructor your script file from Exercise 3.