EGR 111
Loops

This lab is an introduction to loops, which allow MATLAB to repeat commands a fixed 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
Are we there yet?
Are we there yet?
Are we there yet?
Are we there yet?
```

This statement repeats the `disp` function four times, and keeps track of how many times it has repeated the command using the variable n. Let's see how this works by printing 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 statement above, MATLAB takes the first element from the vector 1:4 and places 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. Since n is the name of a variable, MATLAB prints the value of n to the screen.

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

In MATLAB, any row vector could be used in place of 1:4, so if we wanted to count down from 4 to 1, we could do the following:

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

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. Consider the following program:

```
v = [10 20 30 40]     % store 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 when the loop starts.

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 up.

So the program above is equivalent to the commands below.

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

Exercise 1: Write a loop to sum the <u>even</u> numbers from 2 to 100 (that is find 2 + 4 + 8 + … + 100).

Exercise 2: Write a loop that prompts the user for four numbers and prints out the sum of those numbers.

<u>Checkpoint 1:</u> Show the instructor your programs and results from Exercises 1 and 2.

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 shown below.

```
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 care how or if the statements are indented.

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.

```
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.

```
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 3: Write a script that uses a nested loop to access the elements of a 5x5 matrix called A, and set the value of the elements to 1 for those elements where the row number is greater than the column number as 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
```

<u>Checkpoint 2:</u> Show the instructor your programs and results from Exercise 3.

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. MATLAB has another command called the `while` loop that is handy when the number of times the loop will repeat is not known in advance. For example, consider the following program.

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

First the variable x is set to 1. The while command checks to see if x is equal to 1, and if so it executes the input command to prompt the user for a value. Then it repeats. So the loop will keep repeating an unknown number of times until the user types something other than 1.

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.

```
x = randi(10);  % generate random integer between 1 and 10
g = 0;  % initialize guess from user to zero
disp('I picked a random number between 1 and 10')
while g ~= x,
    g = input('Guess what it is: ');
end
disp('Yep')
```

In the program above, the first statement generates a pseudorandom number 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 is not equal to the pseudorandom number x, and, if they are not equal, the input command prompts the user for another guess. This process repeats until the user guesses the correct number.

Exercise 4: Modify the program above to tell the user whether the guess is too high or too low.

Checkpoint 3: Show the instructor your program from Exercise 4.

Optional Exercise 5: Let's write an "evil" version of the program that repeatedly changes its guess based on the user's guesses to make it harder for the user to get the right number. So the program will say that it picked a number, but what it really does is keep a list of numbers that the user hasn't guessed yet. As long as there are any numbers that the user hasn't guessed yet, the program says guess again. For simplicity, this version of the program does not have to tell the user whether the guess is too high or too low.

Optional Checkpoint 4: Show the instructor your program from Exercise 5.