Spring 2018

Lab 5 - For/While Loops

Objectives: To learn how to use MATLAB loops, in particular:

- How a for loop is constructed
- How a while loop is constructed
- How MATLAB handles logic and flow

A MATLAB for loop allows you to pass through ALL values of a vector and calculate certain terms that are dependent upon the vector values. The structure of a for loop is as follows (you do not need to type this...it is for informational purposes only):

```
for [vector] % loop through all values in the vector in order of increasing index
    statement 1 ...
    statement 2 ...
    ...
end % you must end your for loop
```

A while loop is slightly different in that the loop continues provided the loop condition is met. The structure of a while loop is as follows (you do not need to type this...it is for informational purposes only):

```
while (condition) % loop through as long as condition is true
    statement 1 ...
    statement 2 ...
    ...
end % you must end your while loop
```

Nesting loops is necessary for a lot of problems you will likely encounter. You can have a for loop inside of a for loop, a for loop inside of a while loop, a while loop inside of a while loop, or a while loop inside of a for loop. In fact, you may have many loops inside of your loops. One example is as follows (you do not need to type this...it is for informational purposes only):

```
while (condition) % loop through as long as condition is true
    statement 1 ...
    statement 2 ...
    for [vector] % loop through all values in the vector in order of increasing index
        statement1 ...
        ...
    end % you must end your for loop
    ...
end % you must end your while loop
```

1. **Boot It Up**
   (a) Start up MATLAB
2. **Warm Up**
   (a) Try these to get the hang of what you'll need to use in the problems below.
   - Suppose you wanted to sum the numbers 1 through 100. One approach could be as follows:

```
x = 0;
for n=1:100
  x = x + n;
end
x
```

   Create a loop that sums $\frac{1}{2^n}$ for $n = 0$ to 100. You can enter your commands into the command window (to enter a new line in the command window, press 'shift' + 'enter'):

- Now suppose you wanted to calculate 13! ($13 \cdot 12 \cdot 11 \cdot 10 \cdot 9 \cdot 8 \cdot 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$). As we progress through this multiplication, we multiply the previous value by the new value and subtract 1 from the previous value.

```
\% define our starting value; we will subtract 1 from n with each pass
\% through the loop
n = 13;

\% assign the starting value to another variable that will be the
\% multiplication value
f = n;

while n > 0
  f = f * n; \% multiply previous value of f by new value of n
  n = n-1;   \% decrease the value of n for the next pass
end
f \% display result
```

  Using a similar thought process as the factorial problem above, write a while loop that sums the numbers 1 through 100.

3. **Loops**

Compound interest can be calculated using the following formula:

$$A = P\left(1 + \frac{r}{n}\right)^{nt}$$

where

A	=	amount of money accumulated after n years, including interest
P	=	principal amount (the initial amount you borrow or deposit)
r	=	annual rate of interest (as a decimal)
t	=	number of years the amount is deposited or borrowed for
n	=	number of times the interest is compounded per year

VERY IMPORTANT: This will give you the final balance only so you'll need to let $t = 1$ in your loops to iterate one year at a time. Ask your TA for clarity if you are confused on this point.

(a) Write a function that accepts the starting principal (P), the annual interest rate (r), the number of years your money is in the bank (t), and the number of times the interest is compounded per year (n) then displays a table that has the accumulated amount (A) for each year. You will need to use fprintf to format your table as follows:

year	balance
1	$amount1
2	$amount2
...	...
n	$amountn

Plot the amount vs the time in years. (Note: You can build a vector in your loop then plot that over time or you can use *hold on* and plot the data with each loop iteration.) Name your function *interestOnly.m* and run it with the following values:

principal	rate	time	frequency
$10,000	9%	30 years	quarterly
$10,000	9%	30 years	annually
$1,000	12%	20 years	annually
$100,000	4%	10 years	semi-annually

(b) Write a similar function (*interestWithDeposits.m*) that accepts all values as above, but now allows the user to deposit a set amount every year (assume the deposit is one lump sum once per year). Again, display a table with annual values and plot the accumulated amount vs the time in years. Run your function with the following values:

principal	rate	time	frequency	annual deposit
\$10,000	9%	30 years	quarterly	\$5,000
\$10,000	9%	30 years	annually	\$5,000
\$1,000	12%	20 years	annually	\$500
\$100,000	4%	10 years	semi-annually	\$10,000

(c) Suppose you want to deposit a certain amount annually into an account that earns interest with a target amount in mind. Write a function *targetBalance.m* that accepts the initial deposit, amount deposited annually, the interest rate, and the number of times interest is compounded annually, and the target amount the user is seeking. Return to the user the time (in years) needed to reach the target amount, as well as, a table with all annual values. Provide the user with a graphical representation of the growth. Run your function with the following values:

annual deposit	rate	frequency	target value
\$10,000	9%	quarterly	\$1,000,000
\$10,000	9%	annually	\$1,000,000
\$5,000	8%	semi-annually	\$1,000,000
\$10,000	5%	annually	\$1,000,000

(d) Write a script ($financialProfile.m$) that accepts from the user the following values:

- current age
- age of retirement
- current salary
- percentage of salary deposited into an interest earning account annually (decimal)
- average interest rate expected (decimal)
- frequency of compounding (number of times per year)
- target account balance for retirement

Assume a constant salary the user's entire career for the sake of simplicity. You will need to use both *interestWithDeposits.m* and *targetBalance.m* in your script. The functions you have already written should provide the user with 2 tables and 2 plots: one showing the growth until retirement age and one with the growth until their target is met. Display to the user the number of years required to reach their target.

(e) Once the TA has checked your files (*interestOnly.m*, *interestWithDeposits.m*, *targetBalance.m*, and $financialProfile.m$) you are free to go. Do not forget to upload these files to Canvas! Log off of your machine before you leave the lab.