Armstrong State University
Engineering Studies
MATLAB Marina – While Loops Primer

Prerequisites

The While Loops Primer assumes knowledge of the MATLAB IDE, MATLAB help, arithmetic operations, built in functions, scripts, variables, arrays, logic expressions, conditional structures, and for loops. Material on these topics is covered in the MATLAB Marina Introduction to MATLAB module, MATLAB Marina Variables module, MATLAB Marina Arrays module, MATLAB Marina Logic Expressions module, MATLAB Marina Conditional Structures module, and MATLAB Marina Iteration module (for loops).

Learning Objectives

1. Be able to use while loops to iteratively execute blocks of statements.
2. Be able to determine when to use for loops versus while loops for iteration.
3. Be familiar with commonly used iterative operations.
4. Be able to write programs using while loops to solve problems.

Terms

iteration, loop, sentinel

MATLAB Functions, Keywords, and Operators

while, end, break, continue

While Loops

While loops are sentinel controlled loops. The loop body of a while loop is executed as long as the result of the loop control logic expression is true. The number of times the while loop body will be executed is not known at the start of the loop. While loops are commonly used to ensure valid data is entered, repeat operations based on a menu or selection choice, and read data from files (read in an unknown number of elements). A while loop has the general form shown in Figure 1a and a flow chart for a while loop is shown in Figure 1b.

```
initialization statements
while (logical expression)
    statements to be executed while logical expression is true
end
```

Figure 1a, General Form of while Loop

The initialization statement(s) set up any needed starting conditions (typically initializing variables used in the logical expression) before the while loop is entered. Generally the initialization sets up conditions that allow the logical expression to evaluate to true so the while loop is entered (although there are cases for which the loop is never entered). The logical expression should evaluate to a scalar Boolean or number. The statements in the while loop body are executed as long as the logical expression result is true (recall MATLAB treats numbers other than 0 as true). The while loop terminates when the logical expression evaluates to false.

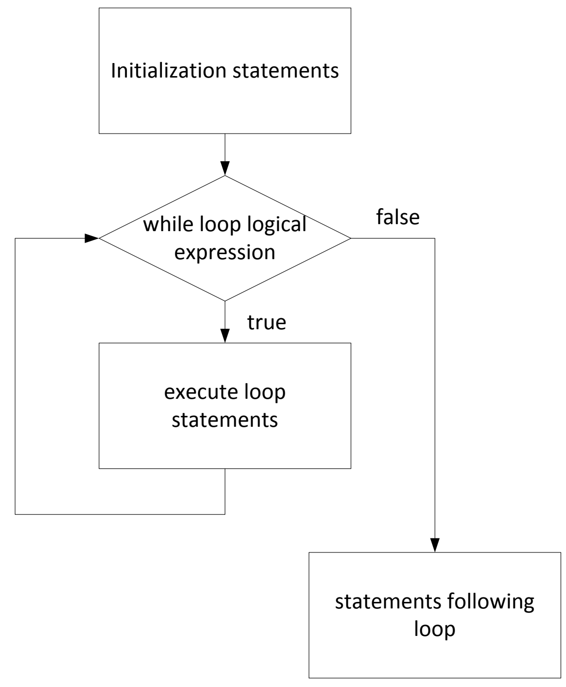
Initialization statements
while loop logical expression
false
true
execute loop statements
statements following loop

Figure 1b, Flow Chart of while Loop

The variables used in the logical expression should be updated during the while loop body otherwise an infinite loop may occur. It is possible to never execute the while loop body since the while loop logical expression test is performed first.

Counting Using while Loops

The MATLAB program of Figure 2a shows one way of displaying the numbers from one to ten using a while loop.

```
startCount = 1;
endCount = 10;

count = startCount;
while (count <= endCount)
    disp(count);
    count = count + 1;
end
```

Figure 2a, Counting Using while Loop

The program of Figure 2a operates as follows:

- Before the while loop, the initialization of loop control variables is performed. In this example, setting the initial count to the start count of one.
- The while loop logical expression (control expression) is evaluated. In this example, the variable `count` is compared to the variable `endCount` to determine if the counting has been completed.
- If the logical expression evaluates to true, the while loop body is executed. In this example, displaying the count and incrementing the count. If the logical expression evaluates to false, the while loop ends and the program goes to the statement following the while loop.
- For all subsequent iterations, the control expression is evaluated and the while loop body statements are executed until the expression is false.

Any variables used in the control expression must be initialized or set up before the while loop. Generally the initialization sets up conditions that allow the logical expression to evaluate to true so the while loop is entered (although there are cases for which the loop is never entered). The variables used in the control expression should be updated during the while loop body otherwise you may get an infinite loop.

The programs for Figures 2b and 2c show examples using a for loop and a while loop to display the numbers from a user specified start number to a user specified stop number.

```
% read in start and end count from user
startCount = input('Enter number to start count at: ');
endCount = input('Enter number to count to: ');

% display count
for count = startCount : 1 : endCount
    disp(count);
end
```

Figure 2a, Displaying a Count Using for Loop

```
% read in start and end count from user
startCount = input('Enter number to start count at: ');
endCount = input('Enter number to count to: ');

% display count
count = startCount;
while (count <= endCount)
    disp(count);
    count = count + 1;
end
```

Figure 2b, Displaying a Count Using while Loop

In the program of Figure 2b using a for loop, the vector of values to count is generated as the loop control array. In the program of Figure 2a using a while loop , a count vector is not generated. Instead an initial count set up before the while loop control expression, the while loop control expression checks that the current count has not exceeded the end count, and the while loop body displays the count and increments the count

Loop break and continue
The `break` and `continue` statements give the programmer additional flexibility with loop control statements. The `break` statement ends the innermost loop containing it and sends control to the next statement after the loop. The `continue` command ends the current iteration of the loop sending control to the end of the loop body, i.e. skips the rest of the statements in the loop body and starts the next iteration. The `break` and `continue` statements are usually used as part of a conditional statement (if) nested within the loop. The break and continue commands can be used with both for and while loops.

Programs can usually be written without using `break` and `continue` statements and one should avoid using `break` and `continue` statements unless it makes the program easier to read. Using conditionals structures, such as if statements, to avoid performing operations or ensuring the logical expression of a while loop handles all the cases of terminating the loop is generally preferable to using break and continue.

Commonly used Iterative Code Segments using While Loops
While loops iterate until a condition is false whereas for loops iterate until all of the control array columns have been used. Any operation that could be done with a for loop could also be done using a while loop (often by adding counting variables to the while loop or using the `break` statement to exit a for loop early). Generally though, if one knows before entering the loop how many times something needs to be done a for loop is the better choice than a while loop. The counting examples showed previously should use for loops rather than while loops since the number of times the loop body will be executed is known (equal to number of count values) at the start of the loop.

Common uses of while loops include: ensuring valid input, repetitively displaying menus and performing operations based on the selections, performing input of an unknown number of elements (file input), and numerical operations that terminate based on an accuracy or tolerance rather than a set number of terms or computations.

Ensuring Valid Input
One common use of while loops is to ensure that data read from the user or a file is valid. The MATLAB code of Figure 3a ensures that the user enters a number greater than zero.

The code of Figure 3a operates as follows:

- The number is read from the user. This is the initialization; the variable `number` is used in the loop logical expression.

- The loop body is executed as long as the number read in is less than or equal to zero (invalid data). The loop body will not be executed if the user enters a valid number the first time.
- In the loop body, the number is read in again. The variable in the loop logical expression is modified with the new value read in so the loop can terminate when the number read in is greater than zero.

```
number = input('Enter number greater than zero: ');
while (number <= 0)
    number = input('Enter number greater than zero: ');
end
```

Figure 3a, While Loop to Ensure user Enters Number Greater than Zero (Valid Data

The MATLAB program of Figure 3b incorporates the code from Figure 3a to ensure an end count greater than zero is read in for a counting operation.

```
% read end count from user
endCount = input('Enter number to count to ( >= 1): ');
while (endCount <= 0)
    endCount = input('Enter number greater than zero: ');
end

% display count from 1 to user entered end count
for count = 1 : 1 : endCount
    disp(count);
end
```

Figure 3b, Counting Using while Loop Ensuring End Count Greater than Zero

Repeated Menu

Another common use of while loops is to allow a user to repetitively make choices from a menu and act on them. While loops that terminate on a specific value are commonly called sentinel controlled loops. A sentinel is an indicator of some condition being met and a sentinel controlled loop usually repeatedly reads in values and tests them until the sentinel input is detected. The first input is typically read in before entering the sentinel controlled loop.

The MATLAB code of Figure 4 shows an example of a repeated menu used to let a user choose to create sine or cosine plots. The program of Figure 4 operates as follows:

- The menu is displayed and the first choice read in from the user. This is the initialization; the variable `choice` is used in the loop logical expression.
- The loop body is executed as long as the choice is not quit. The loop body will not be executed if the user chooses the quit choice initially.
- In the loop body, the selected function is evaluated and plotted. The menu is then displayed and the next choice read in from the user. The variable in the loop logical expression is modified with the new value read in so the loop can terminate when the choice is quit.

```
% read in plot type from user
% display menu and read in initial choice
choice = menu('','sine', 'cosine', 'quit');

% repeat menu and operation chosen until user chooses quit
figureNumber = 0;
while(choice ~= 3)
    % generate trig function according to choice
    t = (0.0 : 0.05 : 1.0);
    if (choice == 1)  % sine
        f = sin(2*pi*t);
    else              % cosine
        f = cos(2*pi*t);
    end

    % plot selection in new figure window
    figureNumber = figureNumber + 1;
    figure(figureNumber);
    plot(t, f);

    % display menu and read in next choice
    choice = menu('','sine', 'cosine', 'quit');
end

disp('You chose to quit');
```

Figure 4, Sentinel Controlled Loop for Repeated Menu

Running Concatenation

The MATLAB code of Figure 5 populates (builds) a row vector of unknown size. Since the size of the resulting vector is unknown a while-loop is used to repetitively read in the values and save them. Operations similar to this are commonly used for low level file input.

```
% start with empty vector
values = [];
% read in and save values until negative number entered
newValue = input('Enter value (negative number to terminate):');
while (newValue >= 0)
    % concatenate new value to existing vector
    values = [values, newValue];
    % read in next value
    newValue = input('Enter value (negative number to terminate):');
end
```

Figure 5, Populating Vector of Unknown Size using Running Concatenation

The code of Figure 5 operates as follows:

- The vector is initialized as an empty vector (no elements) and the first value is read in.
- The loop body is executed as long as the number read is not the terminating delimiter, in this example any negative number terminates the building of the vector. For file input, the loop would be terminated based on an end of line or end of file delimiter. The loop body will not be executed if the first number is negative and the `values` array will be empty.
- In the loop body, each new value is concatenated (appended) onto the end of the existing vector increasing the vector size by one each loop iteration. The next value is then read in. The variable in the loop logical expression is modified with the new value read in so the loop can terminate when the value is negative.

The code of Figure 5 has a major inefficiency. Each loop iteration, the array size is increasing by one. To increase the array size, MATLAB is allocating memory for a new array, copying the old array and new value into the new array, and deallocating the memory for old array. This is much less efficient than if the array size was preallocated. If the array size is not known beforehand the following is often done to make the operation more efficient:

- Preallocate the amount of memory you think you will need for the operation.
- Perform the operation.
- If while performing the operation, the array fills up, allocate memory for a new larger array (typically 20% to 50% larger than the current array). Copy the old array into the new array and deallocate the memory for the old array. Perform this array resizing as needed until the operation is complete.
- Once the operation is completed, resize the array down to deallocate unused memory. Allocate memory for a new smaller array, copy the portion of the array being used into the new array and deallocate the memory for the old array.

Approximating a Taylor Series

The Taylor series of a real valued (or complex valued) function that is infinitely differentiable at a value a is

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(a)}{k!}(x-a)^k$$

Where $k!$ is the factorial of k and $f^{(k)}(a)$ is the kth derivative of $f(x)$ with respect to x evaluated at x = a. The Taylor series for the function $f(x) = e^{-x}$ at $a = 0$ is

$$e^{-x} = 1 - \frac{x}{1!} + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots = \sum_{k=0}^{\infty} \frac{(-1)^k x^k}{k!}$$

The infinite Taylor series can be approximated by using the nth Taylor polynomial $P_n(x)$ where $P_n(x) = \sum_{k=0}^{n} \frac{f^{(k)}(a)}{k!}(x-a)^k$. The program of Figure 6a shows one way to approximate the function $f(x) = e^{-x}$ at $a = 0$ using n = 5, $e^{-x} \approx P_5(x) = 1 - \frac{x}{1!} + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} - \frac{x^5}{5!}$.

```
% Order of Taylor polynomial
n = 5;

x = input('Enter value to approximate f(x) for: ');
% Evaluate Taylor polynomial for e^-x about a = 0
Pn = 0.0;
for k = 0:1:n
    % compute term in series and add to result
    term = ((-1)^k*(x^k))/factorial(k);
    Pn = Pn + term;
end
```

Figure 6a, MATLAB Code to Approximate of Taylor Series of $f(x) = e^{-x}$

The program of Figure 6a is a running sum of the terms of the Taylor polynomial and since the program is approximating the Taylor series using a set n, a for loop is used for the iteration.

The approximation of the infinite Taylor series using the nth Taylor polynomial has a positive error bound of $|E_n(x)| = |f(x) - P_n(x)| \le \frac{M}{(n+1)!}|x-a|^{n+1}$ where M is a value satisfying $|f^{(n+1)}(x)| \le M$ on the interval $[a, x]$. The value of M is chosen so the error bound is as large as possible (worst case). Approximating the function $f(x) = e^{-x}$ by the 2nd order Taylor polynomial at a = 0, $e^{-x} \approx P_2(x) = 1 - \frac{x}{1!} + \frac{x^2}{2!}$, has an error bound of $|E_3(x)| \le \frac{M}{3!}|x|^3$ for $|f^{(n+1)}(x)| = e^{-x} \le M$.

The program of Figure 6b approximates the function $f(x) = e^{-x}$ at $a = 0$ to an accuracy of $\varepsilon = 0.00001$, .i.e. the error bound is $|E_n(x)| \le 0.00001$. To approximate a function to a set accuracy using a Taylor polynomial, the value of n is not known beforehand so the program of Figure 6b uses a while loop to perform a running sum of the terms of the Taylor polynomial until the error bound is met. To determine the error bound, the (n+1)st derivative of the function is needed. In practice, if we are approximating a function using a Taylor polynomial, the (n+1)st derivative of the function is not known and so the error bound must be estimated.

In the case where the series converges and the terms have alternating signs, the error bound $\left|E_n(x)\right|$ can be estimated as the magnitude of the (n+1)st term.

```
% Error bound of Taylor series approximation
epsilon = 0.00001;

x = input('Enter value to approximate f(x) for: ');

% Evaluate Taylor polynomial for e^-x about a = 0
Pn = 0.0;
done = false;
k = 0;
term = ((-1)^k*(x^k))/factorial(k);
% add term to result and compute terms until desired accuracy
while(abs(term) >= epsilon)
    % add term to result
    Pn = Pn + term;
    % compute next term in series
    k = k + 1;
    term = ((-1)^k*(x^k))/factorial(k);
end
```

Figure 6b, MATLAB Code to Approximate of Taylor Series of $f(x) = e^{-x}$

The program of Figure 6b is a running sum of the terms of the Taylor polynomial and since the program is approximating the Taylor series with n not known beforehand, a while loop is used for the iteration. The while loop logic expression will evaluate to false and the loop will end when the desired accuracy is achieved (magnitude of next term is less than error bound).

Loop Use Guidelines

When using for loops:

- Decide whether to iterate over the data or the indices of the data.
- Ensure the loop control array has correct number of columns.
- The loop control variable will hold a column of the loop control array (either next index or next data value) each iteration.
- Ensure the loop body operation is coded correctly depending on whether the loop iterates over indices or data. If iterating over indices make sure to index the array to extract the data and if iterating over the data the loop control variable will hold the data.
- It is considered bad form to change the loop control array of a for loop inside the body of the for loop.

When using while loops:

- Ensure the variables in the loop logical expression have been initialized before the loop.

- The loop logical expression is checked before the first iteration (generally the loop body should be executed at least once).
- Ensure the variable(s) used in the loop logic expression are modified in the loop body so that there is a way to exit the loop.
- If the loop requires indexing of an array, the variable used for the index will need to be incremented in the loop body.