Armstrong State University
Engineering Studies
MATLAB Marina – For Loops Primer

Prerequisites
The For Loops Primer assumes knowledge of the MATLAB IDE, MATLAB help, arithmetic operations, built in functions, scripts, variables, arrays, logic expressions, and conditional structures. 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, and MATLAB Marina Conditional Structures module.

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

Terms
iteration, loop, nested loop, running sum, running product, search

MATLAB Functions, Keywords, and Operators
for, end

For Loops
Iterative structures (for loops, while loops) allow programs to repetitively execute blocks of statements. For loops are used to execute a block of statements a fixed number of times. For loops are counter controlled loops. The number of times the loop body will be executed is known at the start of the loop. Common uses of loops include running totals and processing a vector or array of data. A `for` loop has the general form shown in Figure 1a and a flow chart for a for loop is shown in Figure 1b.

```
for loop control variable = expression
    statements to be executed
end
```

Figure 1a, General Form of for Loop

The expression must evaluate to an array and can either be an existing array or a MATLAB expression that results in an array. The loop body (statements between `for` and `end`) is executed a number of times equal to the number of columns in the array generated by the expression. Each loop iteration, the loop control variable is assigned the next set of column values in the array resulting from the expression and the loop body is executed for that set of values of the loop control variable. The for loop terminates after the last column of the array

resulting from the expression has been used. If the expression results in a scalar, the loop body is executed once.

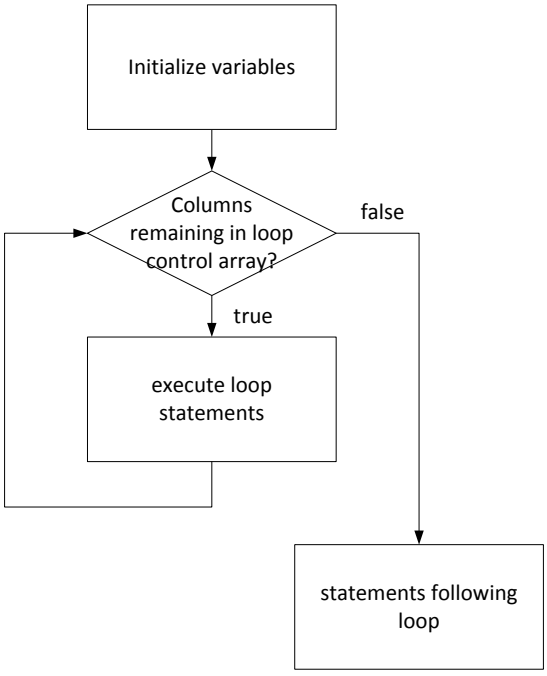

Figure 1b, Flow Chart of for Loop

The MATLAB code of Figures 2a, 2b, and 2c show three ways of displaying the numbers from one to ten using a for loop.

```
for count = 1 : 1 : 10
    disp(count);
end
```

Figure 2a, Counting Using for Loop

```
count = 1 : 1 : 10;
for k = count
    disp(k);
end
```

Figure 2b, Counting Using for Loop

```
count = 1 : 1 : 10;
for k = 1:1:length(count)
    disp(count(k));
end
```

Figure 2c, Counting Using for Loop

The loop of Figure 2a operates as follows:

- When the loop is started, the vector [1 2 3 4 5 6 7 8 9 10] is created by the 1:1:10 expression. This vector will not exist once the loop has finished executing.
- The loop control variable `count` is set to 1 (first column of the vector) for the first iteration of the loop.

- The loop body statements, in this case a display operation, are executed for `count = 1`
- For all subsequent iterations, the loop control variable `count` is set to the value in the next column of the vector and the loop body statements are executed. This is continued until the last column of the vector is used.
- The loop control variable `count` will contain the last column of the loop control expression, in this case 10.

The loop of Figure 2b operates as follows:

- A vector named `count` is created before executing the loop.
- The loop control variable `k` is set to first value in the vector resulting from the loop control expression. In this case the loop control expression is the variable `count` that contains the values from one to ten.
- The loop body statements are executed for the first value of `k`, in this case 1.
- For all subsequent iterations, the loop control variable `k` is set to the value in the next column of the vector and the loop body statements are executed. This is continued until the last column of the vector is used.
- The loop control variable `k` will contain the last column of the loop control expression, in this case 10. Since the `count` vector was created before the loop, it will still exist after the loop is executed.

The loop of Figure 2c operates as follows:

- A vector named `count` is created before executing the loop.
- When the loop is started, a vector of the indices (1 to 10) of the vector `count` is created the is created by the `1:1:length(count)` expression. This vector will not exist once the loop has finished executing.
- The loop control variable `k` is set to 1 (first column of the vector resulting from expression) for the first iteration of the loop.
- The loop body statements, in this case a vector indexing and a display operation, are executed for `k = 1`. This is the only difference between the code of Figure 1c and that of Figure 1b. The loop control variable `k` is used as an index to extract the appropriate value from the count vector for display.
- For all subsequent iterations, the loop control variable `k` is set to the value in the next column of the vector and the loop body statements are executed. This is continued until the last column of the vector is used.
- The loop control variable `k` will contain the last column of the loop control expression, in this case 10. Since the `count` vector was created before the loop, it will still exist after the loop is executed.

There are many operations, such as processing an array of data, that require loops if implemented in programming languages such as Java or C/C++ that can be implemented using array operations in MATLAB. The MATLAB program of Figure 3a evaluates the function

$f(t) = 4t^2 - 7t + 6$ over the range $-5.0 \le t \le 5.0$ (this example was covered in the 1D arrays primer). An alternative implementation using a for loop is shown in Figure 3b.

```
% vector of independent variable values
t = -5.0 : 0.1 : 5.0;
% evaluate f(t) = 4t^2 – 7t + 6
f = 4*t.^2 - 7*t + 6;
```

Figure 3a, MATLAB Program to Evaluate Function f(t) = 4t^2 – 7t + 6

```
% allocate space for t and f arrays
numberPoints = 101;
t = zeros(1, numberPoints);
f = zeros(1, numberPoints);

for k = 1:1:numberPoints
    % generate independent variable values
    t(k) = -5.0 + (k-1)*0.1;
    % evaluate f(t) = 4t^2 – 7t + 6 for current value
    f(k) = 4*t(k)^2 - 7*t(k) + 6;
end
```

Figure 3b, MATLAB Program to Evaluate Function f(t) = 4t^2 – 7t + 6

Commonly used Iterative Code Segments using For Loops

For loops are commonly used to operate on data in arrays (processing a collection of data). Some common operations are: populating (filing) an array, summing the elements of arrays, finding the minimum or maximum value of an array, finding the mean value of an array, scaling the elements of an array, and searching for a value in the array.

Considering writing a MATLAB program that sums the elements of vectors, i.e. reproduces result of the MATLAB `sum` function for a 1D array. To sum the elements of a vector:

- Set the initial result to zero.
- Extract the first element of the vector and add it to the initial result.
- Continue extracting the next element of the vector and adding it to the previous result until all elements in the vector are used.

A skeleton for the sum vector program is given in Figure 4a. The indentation indicates something to be done inside a control structure in this case in the loop body. Figures 4b and 4c show two solutions. The solution of Figure 4b generates a vector of indices that are assigned to the loop control variable. In the loop body, the index in the control variable is used to extract the element from the vector and add it to the previous result. The solution of Figure 4c uses the vector to be summed as the vector assigned to the loop control variable. In the loop body, the vector value in the control variable is added to the previous result.

```
% set initial result to zero

% iterate through the elements of the vector

    % add each element to the previous result

% display result
```

Figure 4a, Skeleton for Summing Elements of a Vector

```
v = [17, 32, 13, 42, 63, 57];
% set initial result to zero
result = 0;

% iterate through the elements of the vector
for k = 1:1:length(v)
    % add each element to the previous result
    result = result + v(k);
end

% display result
disp(result);
```

Figure 4b, Program to Sum Elements of a Vector

```
v = [17, 32, 13, 42, 63, 57];
% set initial result to zero
result = 0;

% iterate through the elements of the vector
for k = v
    % add each element to the previous result
    result = result + k;
end

% display result
disp(result);
```

Figure 4c, Program to Sum Elements of a Vector

The MATLAB programs of Figure 4b and 4c are examples of running sums. For a running sum, the initial sum is set to zero and each subsequent number is added to the current sum. The numbers can come from a preexisting vector or be read in one at a time. There are many engineering operations that require operations similar to running sums. Some examples of

problems that use running sums in the solution are: evaluating series such as Taylor series or Fourier series, computing averages and standard deviations, and numerical integration.

Iterating Over Indices Versus Iterating Over Data

The implementation of the running sum in Figure 4b uses a loop that iterates over the indices of the array to sum.

- The loop control array is a vector of indices of the array to sum.
- The loop control variable k is assigned the next array index for each iteration of the loop
- The loop control variable k is used to index next value of the array v and this value is added to the sum

The implementation of the running sum in Figure 4c uses a loop that iterates over the array data.

- The loop control array is the array to sum.
- The loop control variable k is assigned the next piece of data (value) of the array.
- The loop control variable k which holds the next value of the array is added to the sum. Array indexing is not needed since loop control variable holds array data.

Building a Vector Using a for Loop

Building a vector by generating the values one at a time or reading the values from the user or file is one example of iterating over the indices of an array. Figures 5a and 5b show MATLAB code to build a row vector of real numbers by reading values from the user and generating the values respectively.

```
% Allocate space for the row vector
vectorLength = 15;
newVector = zeros(1, vectorLength);
% Build the vector
for k = 1:1:length(newVector)
    newVector(k) = input(''‘Enter next value: ');
end
```

Figure 5a, Building a Vector by Reading Values

```
% Allocate space for the row vector
vectorLength = 15;
newVector = zeros(1, vectorLength);
% Build the vector
for k = 1:1:length(newVector)
    newVector(k) = 0.0 + (100.0 – 0.0)*rand(1,1);
end
```

Figure 5b, Building a Vector by Generating Values

The loop control array in each example is a vector of the indices of the array to build. Each loop iteration, a real number is either read in or generated (code of Figure 5b uses MATLAB's `rand` function to generate random numbers) and assigned to the appropriate array location indicated by the index for the current loop iteration.

2D Loop Control Arrays

Recall that the loop control expression must result in an array. When the array is a 2D array, the loop control variable will be assigned a column vector rather than a scalar for each iteration of the loop. The program of Figure 6 computes the average exam score of a class of students. The class's exam scores are stored in 2D array with each column of the array holding the scores for a student.

```
% Class exam scores for four students
examScores = [[85;78;80], [75;77;78], [65;71;52], [95;86;90]];
% Allocate space for the averages (row vector)
examAverages = zeros(1,size(examScores,2));

studentNumber = 1;
for k = examScores
    examAverages(studentNumber) = (k(1) + k(2) + k(3))/3;
    studentNumber = studentNumber + 1;
end

disp(examAverages);
```

Figure 6, Program with 2D Array as Loop Control Array

The loop is iterating over the exam score data. Each loop iteration, the loop control variable `k` is assigned a three by one column vector corresponding to the exam scores for a student. The average of the values in the column vector is computed in the loop body and the result stored in the `examAverages` array. The loop is executed four times, one for each column in the `examScores` array. Iteration using a 2D array as the loop control array is useful for iterating over complex data structures since column indexing is done as part of the for loop operation.

The computation of the average exam score for each student in the program of Figure 5 could be done using MATLAB's `mean` function instead of adding the three scores and dividing by three, `examAverages(studentNumber) = mean(k);`.
The averages for all students could be done directly using MATLAB's `mean` function by taking the mean along the columns of the exam scores,
`examAverages = mean(examScores,1);`. This would eliminate the need for a for loop since MATLAB's `mean` function is defined for 2D arrays.

Nested For Loops

Nested loops have one or more inner loops contained in the loop body of an outer loop. Nested loops are commonly used when operating on multi-dimensional arrays. To process a 2D array,

one typically needs a nested loop. The operation performed on the elements of the array may also require loop constructs yielding triple or higher nested loops.

The program of Figure 7 sums the elements of a 2D array. The outer loop with k1 as loop control variable keeps track of which row and the inner loop with k2 as loop control variable runs through all the columns for the current row. Thus each row is summed and this is done for all rows.

```
% 5 by 30 array of random integers between 0 and 100
V = round(0.0 + 100*rand(5,30));

% sum all elements of 2D array V
result = 0;
numberRows = size(V,1);
numberColumns = size(V,2);
% iterate over row indices of array V
for k1 = 1:1:numberRows
   % iterate over column indices of array V
    for k2 = 1:1:numberColumns
        % index next value of V and add to the running sum
        result = result + V(k1,k2);
    end
end

disp('The sum of elements in V is: '),disp(result);
```

Figure 7, Sum of Values in a Two-Dimensional Array

Searching an Array

Common searching operations on arrays are: finding the minimum value, finding the maximum value, and finding a particular value. Consider writing a program that determines the minimum value in a vector, i.e. reproduces the result of the MATLAB `min` function for a row vector.

To find the minimum value of the elements of a vector:

- Set the initial minimum to be the first element (or a number much larger than is expected in the vector)
- Compare the second (or first if not already used) element to the current minimum. If it is smaller replace the current minimum with the newly found minimum.
- Continue comparing the next element of the vector and updating the minimum if appropriate.

The MATLAB program of Figure 8 implements a vector search for the minimum value.

```
% test array
v = [17, 32, 13, 42, 63, 57];

% set initial minimum to first value of array
result = v(1);
% search rest of array for a smaller value
for k = 2 : 1 : length(v)
    % compare current element to minimum, update result if smaller
    if (v(k) < result)
        result = v(k);
    end
end

disp(result);
```

Figure 8, Minimum Value of 1D Array Program

The program of Figure 8 assumes that only the value of the minimum is desired not the location in the array of where the minimum value occurs. If the location was desired we would also need to worry if multiple elements had the same value.

Last modified Tuesday, September 09, 2014