

MATLAB Marina: Iteration, for loops applications

Student Learning Objectives

After completing this module, one should:

1. Be able to process arrays of data using for loops.
2. Be able to solve problems using for loops.

Terms

NA

MATLAB Functions, Keywords, and Operators

NA

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 (filling) 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.

Evaluating a Formula using a For Statement

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 1a evaluates the function $f(t) = 4t^2 - 7t + 6$ over the range $-5.0 \leq t \leq 5.0$ using element by element operations on an array. An alternative implementation using a for loop is shown in Figure 1b.

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

Figure 1a. MATLAB Program to Evaluate Function $f(t) = 4t^2 - 7t + 6$

In the program of Figure 1b, the `zeros` function is used to create the initial space for the arrays that will hold the independent variable values and the function values.

The for loop uses the array indices as the loop control array and the loop body will be executed once for each t and $f(t)$ value, the index k corresponds to which t and $f(t)$ value is being determined.

```

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

% compute f(t) for each value of t in the array
for k = 1:1:numberPoints
    % time value for point k
    t(k) = -5.0 + (k-1)*0.1;
    % evaluate function for t value for point k
    % save result in corresponding location in f array
    f(k) = 4*t(k)^2 - 7*t(k) + 6;
end

```

Figure 1b. MATLAB Program to Evaluate Function $f(t) = 4t^2 - 7t + 6$

For each k (1, 2, 3, ..., 101), the time value is determined and saved in the t array and then the function $f(t)$ is evaluated for that time value and the result saved at the corresponding location in the f array. Notice that element by element operations are not necessary as each $t(k)$ value is a scalar.

Processing an 1D Array using a For Statement, Array Sum

Summing the elements of an array reproduces the operation of the MATLAB `sum` function for a 1D array. The initial sum is set to zero. The loop uses the indices of the array v as the loop control array. Each time through the loop:

- A value is extracted from the array, $v(k)$
- The extracted value is added to the current sum and the sum is overwritten with the new value for the sum.

The operations are repeated for each index of the array until all of the array elements have been added to the sum.

Figures 2a and 2b illustrate the process of developing a program to sum an array. Figure 2a shows an initial set of comments corresponding to the algorithm and Figure 2b shows the completed program. The indentation in the initial comments indicates something to be done inside a control structure in this case in the loop body.

```

% set initial sum to zero

% sum the elements of the array
% iterate using the array indices as loop control array

    % extract array element, add to sum, overwrite sum

% display the array sum

```

Figure 2a. Array Sum Algorithm Converted to Comments

```

%% sum the elements of an array
clear; clc;
v = [17, 32, 13, 42, 63, 57];

% set initial sum to zero
res = 0;

% sum the elements of the array
% iterate using the array indices as loop control array
for k = 1:1:length(v)
    % extract array element and add to sum, overwrite sum with
new value
    res = res + v(k);
end

% display the array sum
fprintf('Array sum is: %d\n', res);

```

Figure 2b. Array Sum Program

The MATLAB program of Figure 2b is a running sum operation.

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 array, 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.

Processing an 1D Array using a For Statement, Display Positive Array Elements

The MATLAB program of Figure 3 displays only the positive values in an array.

```

% array to process
values = [2, -1, 0, 4, 7, -3, 9];

% display positive elements of the array
fprintf('Positive elements of the array are: \n');
for k = 1:1:length(values)
    val = values(k);
    if val > 0
        fprintf('%d ', val);
    end
end
end
fprintf('\n');

```

Figure 3. MATLAB Program to Display Positive Elements of an Array

The loop uses the indices of the array `values` as the loop control array. Each time though the loop:

- A value is extracted from the array, `val = values(k);`
- The value is checked to see if it is positive, `if val > 0`
- The value is displayed if it is positive, otherwise it is not used.

The operations are repeated for each index of the array.

Creating an 1D Array using a For Statement

Building or creating an array can be done by generating the values one at a time or reading the values from the user or file. When creating an array using a for loop, the number of values that will be in the array should be known.

Figures 4a and 4b shows programs that create an array by generating or reading in the values and saving them in the array. The loop control array in each program is the indices of the array to create. Each loop iteration, a number is either generated using MATLAB's `rand` function or read and then saved in the appropriate array location indicated by the index for the current loop iteration.

```
% allocate space for the new array
arrLength = 15;
newArr = zeros(1, arrLength);

% create the array
for k = 1:1:length(newArr)
    % create the next array value
    val = 0.0 + (100.0 - 0.0)*rand(1,1);
    % save the value in the array
    newArr(k) = val;
end
```


Figure 4a. Create Array by Generating Values

```
% allocate space for the new array
arrLength = 15;
newArr = zeros(1, arrLength);

% create the array
for k = 1:1:length(newArr)
    % read in the next array value
    val = input('Enter next array value: ');
    % save the value in the array
    newArr(k) = val;
end
```

Figure 4b. Create Array by Reading Values

Last modified April 26, 2021

 [MATLAB Marina](#) is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.