MATLAB Marina: Iteration, for loops advanced

Iterating Over Indices Versus Iterating Over Values

The loop control array for a for loop can either be an array of indices as has been used previously or an array of values. Consider the program to sum an array of values shown in Figure 1a.

```
% 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 1a. Array Sum Program (loop control array contains indices)

The loop control expression creates an array of indices for the array whose elements will be summed. Each time though the loop:

- The loop control variable gets the next index for the array.
- A value is extracted from the array using indexing, `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 indices have been used.

The program of Figure 1b produces the same result as that of Figure 1a but the for loop iterates over the values of the array rather than indices. The loop control expression creates an array of values (a copy of the array values). Each time though the loop:

- The loop control variable gets the value in the array (next column of the array).
- The 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 value of the array until all of the values have been used.

```
% 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 values as loop control array
 for val = v
    % add the array value to sum, overwrite sum with new value
    res = res + val;
end

% display the array sum
fprintf('Array sum is: %d\n', res);
```
Figure 1b. Array Sum Program (loop control array contains values)

The array values do not need to be indexed as the loop control variable will be assigned the values from the array. Iterating over the array values is appropriate when the location of the value is not important. Iterating over values is generally used when producing a scalar result from an array. If producing an array result with values corresponding to the original array, iterating over indices is appropriate since one needs the array location of the original value and where to place the result. This mechanism for controlling a for loop is not typical of most programming languages.

2D Loop Control Arrays
Recall that the loop control expression must result in an array. When the loop control array is a 2D array, the loop control variable will be assigned a column array rather than a scalar for each iteration of the loop.

The program of Figure 2 processes a 2D array of exam scores and computes the average of each column of scores. Each column of the array corresponds to one student's scores and the exam average for the student is the average of the scores in the column.

The loop control array is a 2D array of scores and the loop is iterating over values. Each loop iteration, the loop control variable `vals` is assigned a three by one column array corresponding to the exam scores for a student. The average of the values in the column vector is computed and stored in the `examAvgs` array. The for loop is executed four times, once for each column in the `examScores` 2D 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.

```
% 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)
examAvgs = zeros(1,size(examScores,2));

studentNum = 1;
% for each student
for vals = examScores
    % compute average of the scores in the column
    examAvgs(studentNum) = (vals(1) + vals(2) + vals(3))/3;
    studentNum = studentNum + 1;
end
```

Figure 2. Program to Determine Exam Averages

The computation of the average score for each student in the program of Figure 2 could be done using MATLAB's `mean` function instead of adding the three scores and dividing by three,

`examAvgs(studentNum) = mean(vals);`

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 2D array

`examAvgs = mean(examScores,1);`

This would eliminate the need for a loop and is possible since many MATLAB functions are 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 for loops are commonly used when processing multi-dimensional arrays such as image processing operations. Some image processing applications require triple or quad nested loops.

The program of Figure 3 produces the same result as that of Figure 2, but the exam averages are computed using a nested for loop. The outer for loop iterates over the column indices of the array; each column corresponds to the scores for a student. The inner for loop iterates over the row indices of the array which correspond to the scores for the student.

Each time though the outer loop:

- The loop control variable k1 gets the column index corresponding to the current student.
- The inner loop is executed.
    - The loop control variable k2 gets the row index corresponding to the next score for the student.
    - The score is extracted from the 2D array, `examScores(k2,k1)`, and added to the sum of scores for the student.
- Once the inner loop sums all the scores for the student, the average scores is computed and saved.

The operations are repeated for each column of the array until all of the columns have been used. For this example, k2 is the index for the student (student scores are column data) and k1 is the index for the score.

```
% 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)

% number of exam scores and number of students
examAvgs = zeros(1,size(examScores,2));

[nrow, ncol] = size(examScores);
% for each student
for k1 = 1:1:ncol
    % each each exam score
    examSum = 0;
    for k2 = 1:1:nrow
        examSum = examSum + examScores(k2,k1);
    end
    % compute and save average
    examAvgs(k1) = examSum/nrow;
end
```

Figure 3. Program to Determine Exam Averages

Mixing Array Operations and Iteration

Operations in a loop body are generally scalar operations although there are some problems that involve both iteration and array operations for solutions using MATLAB. Computing a harmonic sum of sinusoids for a Fourier series representation of a signal is one application.

Fourier series representation of a periodic signal x(t) can be represented as a harmonic sum of sinusoids.

$$x(t) = A_0 + \sum_{k=1}^{\infty} A_k \cos(k\omega_0 t + \phi_k)$$

Harmonic frequencies are frequencies that are integer multiples of each other. Generally, the number of harmonics required is infinite, but an approximation for N terms can often be used

$$x_N(t) = A_0 + \sum_{k=1}^{N-1} A_k \cos(k\omega_0 t + \phi_k)$$

The fundamental frequency ω_0 is also the first harmonic, $2\omega_0$ is the second harmonic, $3\omega_0$ is the third harmonic and so on. The kth harmonic is $\omega_k = k\omega_0$.

The program of Figure 4 computes a Fourier series approximation of a signal using 20 harmonics.

```
% Fourier series using DC term A0 plus 10 harmonics
A0 = 1;
N = 20;

% fundamental frequency (rad/s)
w0 = 20*pi;
t = 0:0.001:0.4;
x = ones(1,length(t))*A0;

% sum the harmonic sinusoids
for k = 1:1:N
    Ak = (-1)^k*2/(pi*k);
    x = x + Ak*cos(k*w0*t);
end
```

Figure 4a. Fourier Series Approximation of Signal

A for loop is used to run through the N harmonics. In the loop body, the amplitude of the harmonic in computed and then the cosine function corresponding to that harmonic is computed using an array operation (t is an array of time values). This is done for each harmonic. The loop performs a running sum operation on arrays of values where each array corresponds to the function points for the harmonic and each has the same length as the t array. This would be difficult to program using only array operations as the lengths of the Ak and t arrays would not be the same.

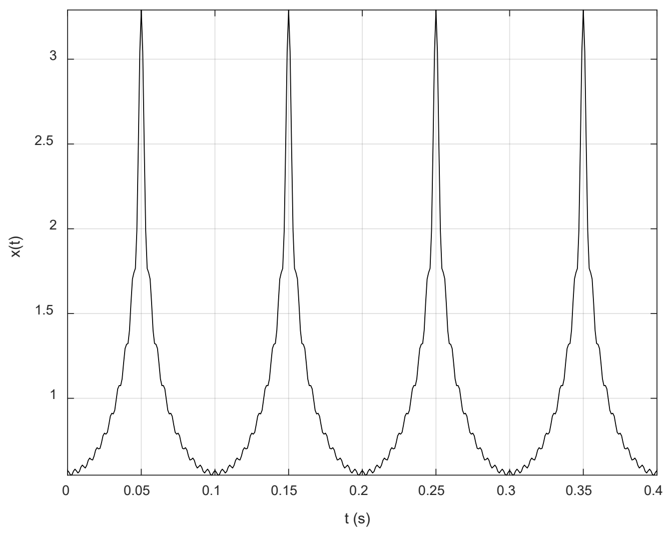

Figure 4b. Fourier Series Approximation of Signal

Last modified April 27, 2021