Repetition Structures
Chapter 9

Value of the Alternating Harmonic Series

MATLAB for Engineers 3E, by Holly Moore. © 2011 Pearson Education, Inc., Upper Saddle River, NJ. All rights reserved.
This material is protected by Copyright and written permission should be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permission(s), write to: Rights and Permissions Department, Pearson Education, Inc., Upper Saddle River, NJ 07458.
Objectives

After studying this chapter you should be able to:

- Write and use for loops
- Write and use while loops
- Create midpoint break structures
- Measure the time required to execute program components
- Understand how to improve program execution times
Structures

- Sequence
- Selection
- Repetition (Loop)
Types of Loops

- Loops are used when you need to repeat a set of instructions multiple times
- MATLAB supports two types of loops
  - for
  - while
When to use loops

- In general loops are best used with scalars, or with the values stored in a matrix used one at a time.
- Many of the problems you may want to attempt with loops can be better solved by vectorizing your code or with MATLAB's logical functions such as `find`.
9.1 For Loops

```
for index = [matrix]
    commands to be executed
end
```

The loop is executed once for each element of the index matrix identified in the first line.
Flow chart for a for loop

True; You’ve run out of values in the index matrix

Check to see if the index has been exceeded

Calculations

Flow chart for a for loop

Check to see if the index has been exceeded

Calculations

Flow chart for a for loop

True; You’ve run out of values in the index matrix

Check to see if the index has been exceeded

Calculations
The FOR loop

Example: testloop.m

% program to test a for loop
for i=1:10
    disp(i)
end

>> testloop
1
2
3
...
10

>> disp(i)
10

% program to test a for loop
for i=1:10
    disp(i*0.2)
end

>> testloop
0.2
0.4
0.6
...
2.0

>> disp(i)
10

% program to test a for loop
for i=1:0.1:2
    disp(i*5)
end

>> testloop
5
5.5
6
...
9.5
10

>> disp(i)
2
Here’s a simple example

```matlab
for k=[1, 3, 7]
    k
end
```

the index can be defined using any of the techniques we’ve learned
Here’s a simple example

```matlab
for k=1:3
    a=5^k
end
```

the index can be defined using any of the techniques we’ve learned

```matlab
a =
5
a =
25
a =
125
>>
```
One of the most common ways to use a loop is to define a matrix

```matlab
for k = 1:5
    a(k) = k^2
end
```

```
a = 1
a = 1 4
a = 1 4 9
a = 1 4 9 16 25
>>
```
Hint

Most computer programs do not have MATLAB’s ability to handle matrices so easily, and therefore rely on loops similar to the one on the previous slide to define arrays. It would be easier to create the vector $a$ in MATLAB with the following code:

$$k=1:5$$
$$a = k.^2$$

which returns:

$k =
\begin{array}{c}
1 \\
2 \\
3 \\
4 \\
5 \\
\end{array}$

$a =
\begin{array}{c}
1 \\
4 \\
9 \\
16 \\
25 \\
\end{array}$

This is an example of vectorizing the code.
Each time through the loop we evaluate a single element of the scores matrix.
Example:
Assume we have series of natural numbers up to 100 and want to find the summation of these numbers.

- Use a for loop, using the loop variable as the index.
- At every step of the loop, we want to add the value corresponding to the index.

Result variable: sum

Summation is done by addition of 1st elements, obtain partial result, add 2nd element, obtain partial result, etc....

% program to test a for loop
sum=0;
for i=1:100
    sum=sum+i;
end

disp(sum)
disp(['result=',num2str(sum)])

sum=sum + next value
Summary of the for loop structure

- The loop starts with a `for` statement, and ends with the word `end`.
- The first line in the loop defines the number of times the loops will repeat, using an index number.
- The index of a `for` loop must be a variable. (The index is the number that changes each time through the loop.) Although `k` is often used as the symbol for the index, any variable name can be used. The use of `k` is a matter of style.
• Any of the techniques learned to define a matrix can be used to define the index matrix. One common approach is to use the colon operator.

• **for index = start:inc:final**

• If the expression is a row vector, the elements are used one at a time – once for each time through the loop.
• If the expression is a matrix (not common), each time through the loop the index will contain the next column in the matrix. This means that the index will be a column vector!!

• Once you’ve completed a for loop, the index is the last value used.

• for loops can often be avoided by vectorizing the code.
**While Loops**

- While loops are very similar to for loops.
- The big difference is the way MATLAB decides how many times to repeat the loop.

```matlab
while criterion
    commands to be executed
end
```

- While loops continue until some criterion is met.
Check to see if the criterion is still true

Calculations

The criterion is no longer true and the program exits the loop

Flow Chart for a while loop
We have to increment the counter (in this case k) every time through the loop – or the loop will never stop!!
This loop creates the matrix $a$, one element at a time.
The **WHILE** loop

The **while** loop is used when the looping process terminates because a specified condition is satisfied, and thus the number of passes is not known in advance. A simple example of a while loop is

```matlab
x = 5;
while x < 25
    disp(x)
    x = 2*x - 1;
end
```

The results displayed by the `disp` statement are 5, 9, and 17.
This program counts how many scores in the array are greater than 90, and displays the result in the command window.
Hint

If you accidentally create a loop that just keeps running you should

1. Confirm that the computer is actually still calculating something by checking the lower left hand corner of the MATLAB window for the “busy indicator”

2. Make sure the active window is the command window and exit the calculation manually with $\text{ctrl} \; \text{c}$
The WHILE loop

Example: exloop3.m

% BAD while loop

x=1;

while x>=0
    x=x+0.5;
    y=sin(x);
end

disp(‘END of program’)

>> exloop3

This is an Infinite loop !!!!

Need to stop the script manually !!!

CTRL C

>>
Use **while loop** when the number of operation is unknown

**Example : exloop4.m**

Calculate the sum of manually entered numbers. Entering 0 or a negative number stops the loop and display the average.

- Need input statement: 
  - variable: A

- Use the length of A in the *logical_expression*
  
  \[ A > 0 \]

- Inside the loop:
  - \[ i = i + 1 \]

- Add the value A to the sum S
  
  \[ S = S + A \]

% program to test a while loop

```matlab
S=0;
i=0;
A=input('Value for A:');
while A>0
    i=i+1;
    S=S+A;
    A=input('Value for A:');
end
disp(S/i)
```

```matlab
>> testloop
Value for A: 2
Value for A: 2
Value for A: 2
Value for A: 0
4.6667
```

```matlab
>> testloop
Value for A: 1
Value for A: 5
Value for A: 8
Value for A: 0
Value for A: 2
Value for A: 2
Value for A: 0
2
```
The WHILE loop

Example 5:
Series convergence:

\[ \sum_{i=1}^{N} \frac{1}{i^2} \rightarrow \frac{\pi^2}{6} \]

Want to see how many terms you need to obtain an error of 3x10^{-6}.

- Need to define and use an error variable.
- Need to be part of the logical expression
- Need to be updated during loop

- At every step of the loop, we want to
  - Verify if test is true  \((err>3e-6)\)
  - Increment counter  \(i=i+1\)
  - Add the new term to the sum  \(S=S+1/i^2\)

% Convergence script
S=0;
i=0;
err=10;
while err>3e-6
  i=i+1;
  S=S+1/i^2;
  err=abs(S-pi^2/6);
end
disp(['N=',num2str(i)])

>> Testloop
N=333333
break and continue

- **break** causes the loop to terminate prematurely
- **continue** causes MATLAB to skip a pass through the loop, but continue on until the criteria for ending is met
- both are used in conjunction with an if statement
The “BREAK” statement

**BREAK**

- **Break** terminates the execution of a for or while loop. Statements in the loop that appear after the **break** statement are not executed.

- In nested loops, **break** exits only from the loop in which it occurs. Control passes to the statement that follows the **end** of that loop.

```matlab
% BAD while loop
x=1;
while x>=0
    x=x+0.5;
    y=sin(x);
    if x>10000
        break
    end
end
```
This program prompts the user 10 times to enter a value, and computes the natural log.

If a negative value is entered, the break command causes MATLAB to exit the loop.
Notice that `n` had a value of 3 when the program terminated – if it had run to completion it would have had a value of 10.
The **continue** command is similar to **break**, however instead of terminating the loop, the program just skips to the next pass.
The “CONTINUE” statement

% Problem of division by 0
x=1;
for i= -10:10
    if x==0
        continue
    end
    y=1/x;
end
Notice that n had a value of 10 when the program terminated.
9.4 Midpoint Break Loops

- Implemented with either a for or a while loop.
- The loop is:
  - Entered
  - Calculations processed
  - Decision is made at an arbitrary point whether or not to continue.
while *criterion*
  do some calculations
  *make a decision* on whether to continue
  do some more calculations
end
while (1)

num_c_bars = input('Enter the number of items: ');
if num_c_bars < 0
    disp('Must be a positive number')
else
    total = num_c_bars .* (0.75);
    disp('Your bill is: ')
    fprintf('$ %4.2f \
    end

break

Results in a continuous loop

Causes the loop to terminate
while(1)
    num_c_bars=input('Enter the number of items: ');
    if num_c_bars < 0
        disp('Must be a positive number')
    else
        total = num_c_bars.*(0.75);
        disp('Your bill is: ')
        fprintf('$ %4.2f
',total)
        break
    end
end
for k=1:3
num_c_bars=input('Enter the number of items: ');
if num_c_bars <0
    disp('Must be a positive number')
else
    total = num_c_bars.*(0.75);
    disp('Your bill is: ')
    fprintf('$ %.2f
',total)
end
end

Enter the number of items: -1
Must be a positive number
Enter the number of items: -2
Must be a positive number
Enter the number of items: -3
Must be a positive number
9.5 Nested Loops

- It is often useful to nest loops inside of other loops
- To illustrate, consider code to mimic the built-in max function
Nested loops and combination

Using nested loops

Example: exloop6.m
Calculate the sum of factorials up to 20

1! + 2! + 3! + 4! + .... + 20!

• Need a for loop for sums
• Need a for loop for factorials
• Calculate the factorial of element j
• Do the sum on all elements

% program to test nested loops
S = 0;
for i = 1:20
    F = 1;
    for j = 1:i
        F = F * j;
    end
    S = S + F
end
disp(S)
x = [1 2 6 3;
     4 8 2 1;
    12 18 3 5;
     6 4 2 13];

max(x)
x = [1 2 6 3; 4 8 2 1; 12 18 3 5; 6 4 2 13];
[rows, cols] = size(x);
for k = 1:cols
    maximum(k) = x(1, k);
    for j = 1:rows
        if x(j, k) > maximum(k)
            maximum(k) = x(j, k);
        end
    end
end
maximum

maximum = 
12.00 18.00 6.00 13.00
9.6 Improving the Efficiency of Loops

- In general, using a **for** loop (or a **while** loop) is less efficient in MATLAB than using array operations.
These two lines of code start a timer to measure the elapsed time required to run the lines of MATLAB code between them.

The amount of time it takes to run this code will depend on your computer, and how much else you have installed on it.
This code accomplishes the same thing with a for loop

```
clear,clc
A = ones(2000);  % Creates a 2000 by 2000 matrix of ones
t0 = clock;
for k=1:numel(A)
    B(k) = A(k) * pi;
end
time = etime(clock, t0)
```
If we predefine the B matrix and then replace each element one at a time with the new value, the run time is significantly reduced.
An alternate syntax for timing uses the tic and toc functions.
Hint

- Be sure to suppress intermediate calculations when you use a loop.
- Printing those values to the screen will greatly increase the amount of execution time.
- If you are brave, repeat the example above, but delete the semicolons inside the loop just to check out this claim.
- Don’t forget that you can stop the execution of the program with ctrl c. (But the command window needs to be active)
Summary

• Sections of computer code can be categorized as
  • sequences
  • selection structures
  • repetition structures
Summary – Sequence

• Sequences are lists of instructions that are executed in order
Summary – Selection Structure

- Selection structures allow the programmer to define criteria (conditional statements) which the program uses to choose execution paths.
Summary – Repetition Structures

- Repetition structures define loops where a sequence of instructions is repeated until some criterion is met (also defined by conditional statements).
Summary – Relational Operators

- MATLAB uses the standard mathematical relational operators
  - <
  - <=
  - >
  - >=
  - ==
  - ~=

Recall that = is the assignment operator, and can not be used for comparisons
Summary – Logical Operators

• MATLAB uses the standard logical operators
  • &&    and
  • ||     or
  • ~      not
  • xor    exclusive or
Summary - Loops

- MATLAB supports both
  - for loops
  - while loops
- **For** loops are primarily used when the programmer knows how many times a sequence of commands should be executed.
- **While** loops are used when the commands should be executed until a condition is met.
- Most problems can be structured so that either **for** or **while** loops are appropriate.
Summary
break and continue

• These commands are used to exit a loop prematurely and are widely implemented in midpoint break loops
  • **break** causes the program to jump completely out of a loop and continue execution of the remainder of the program
  • **continue** skips execution of the current pass through a loop, but allows the loop to continue until the completion criteria is met
Summary - Vectorization

- Vectorization of MATLAB code allows it to execute much more efficiently, and therefore more quickly.
- Loops in particular should be avoided in MATLAB, although this is not always possible.
- When loops are unavoidable they can be improved by defining “dummy” variables with placeholder values, such as ones or zeros.
- These placeholders can then be replaced in the loop resulting in significant improvements in execution time, which can be confirmed with timing experiments.
Summary – timing functions

- The **clock** and **etime** functions are used to poll the computer clock, and then determine the time required to execute pieces of code.
- The time calculated is the “elapsed” time. During this time the computer has not only been running MATLAB code, but has also been executing background jobs and housekeeping functions.
- The **tic** and **toc** functions perform a similar task.