Chapter 5

Control Flow Structures

MATLAB has several constructs that allow for varying ways to control the flow of program execution.

Iteration:
- `while` loops
- `for` loops

Selection:
- `if - else` statements
- `if - elseif` statements

5.1 Relational and Logical Operators

The operation of many branching constructs is controlled by an expression whose result is either `true` (1) or `false` (0). There are two types of operators that produce `true/false` results: relational operators and logic operators. Unlike C++, MATLAB does not have a boolean data type (it only has double and character). However, as in C++, MATLAB interprets zero (0) as `false` and any other value as `true`.

5.1.1 Relational Operators

Relational operators are binary operators that yield a true or a false. The general form is:

```
x op y
```

where `x` and `y` may be arithmetic expressions, variables, result of function calls, strings
`op` is one of the relational operators in the table below.

MATLAB	Operation	C++
==	Equal to Note: two == a single = is an assignment	==
~=	Not equal to	!=
>	Greater than	>
>=	Greater than or equal to	>=
<	Less than	<
<=	Less than or equal to	<=

Caution concerning the == and ~= operators. Remember that all variables in MATLAB are double and a check for equality on doubles is a bad idea. There is roundoff error during calculations;

two numbers, theoretically equal, can differ slightly causing an equality or inequality test to fail. Instead of comparing two numbers for exact equality, you should check if they are close enough, i.e., check if they are within `eps`.

```
>> a = 0
a =
             0

>> b = sin(pi)
b =
          0.00

>> a == b
ans =
             0                 %remember 0 is false

>> abs(a-b) < eps
ans =
          1.00                 %this check gives true
```

5.1.2 Logical Operators

Logical operators connect two or more relational expressions into one or reverse the logical of an expression.

MATLAB	Operation	C++
`&`	Logical AND	`and` `&&`
`|`	Logical OR	`or` `||`
`xor`	Logical exclusive OR one or the other but not both	`N/A`
`~`	Logical NOT	`!`

Inputs		and	or	xor	not
a	b	a & b	a | b	xor(a,b)	~a
0	0	0	0	0	1

Inputs		and	or	xor	not
a	b	a & b	a | b	xor(a,b)	~a
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Logical operators can be used to compare a scalar value with an array and for comparing arrays.

Examples	
`>> A = [1 2; 3 0];`	**>> B = [1 7; 0 9];**

`>> A == B`
 	
`ans =`
 	
`1 0`
`0 0`
	`>> A < B`
	
	`ans =`
	
	`0 1`
	`0 1`

`>> A & B`
 	
`ans =`
 	
`1 1`
`0 0`
	`>> A | B`
	
	`ans =`
	
	`1 1`
	`1 1`

`>> A~=B`
 	
`ans =`
 	
`0 1`
`1 1`
	`>> A<=B`
	
	`ans =`
	
	`1 1`
	`0 1`

5.1.3 Logical Functions

Function	Description	Example

`any(A)`	vector: return 1 if ANY of the elements of A are nonzero: 0 otherwise	`D = [0 1 0 1];`
		`>> any(D)`
		`ans =`
		`1.00`

Function	Description	Example

`any(A)`	matrix: operates on columns of A, returning a row vector of 1s and 0s	`C = [1 0 0 ; 0 0 0];`
		`>> any(C)`
		`ans =`
		`    1.00   0   0`
		`>>`

`all(A)`	vector: return 1 if ALL of the elements of A are nonzero: 0 otherwise	`>> all(D)`
		`ans =`
		`             0`

`all(A)`	matrix: operates on columns of A, returning a row vector of 1s and 0s	`>> all(C)`
		`ans =`
		`  0     0     0`

`find(A)`	finds the indices of the nonzero elements of A	`A = [1.00  0 0;`
		`       0  2.00  4.00]`
		`>> find(A)`
		`ans =`
		`          1.00`
		`          4.00`
		`          6.00`

`isnan(A)`	returns 1s where the elements of A are NaNs and 0s where they are not	`>> B = [1 NaN 0; 0 2 4];`
		`>> isnan(B)`
		`ans =`
		`   0   1.00   0`
		`   0   0      0`

`isempty(A)`	an empty matrix has a zero size in at least one dimension isempty(A) returns 1 is A is empty matrix and 0 otherwise	
`ischar(A)`	returns 1 if A is a character array and a 0 otherwise	

`isinf(A)`	returns a 1 if the value of A is infinite (Inf) and a 0 otherwise	`C = [1.00  0  Inf;`
		`     NaN   1.00  Inf]`
		`>> isinf(C)`
		`ans =`
		`   0   0   1.00`
		`    0  0   1.00`

`ifnumeric(A)`	returns a 1 if the A is a numeric array and a 0 otherwise	

Function	Description	Example

`finite(A)`	returns 1 where the elements of A are finite and 0s where they are not	`>> C = [1   0 inf;`
		`        NaN 1 inf];`
		`>> finite(C)`
		`ans =`
		`  1.00    1.00    0`
		`  0       1.00    0`

5.2 Iteration (Loops)

Loops allow programmers to execute a sequence of statements more than once. There are two basic forms of loop constructs: `while` loops and `for` loops. The major difference between the two types of loops is how the repetition is controlled. `While` loops are usually used when you do not know how many times you want the loop to execute. It is controlled by a condition. The `for` loop is a count controlled loop, i.e., the number of repetitions is known before the loop starts.

5.2.1 while loop

The **while** loop repeats a group of statements an indefinite number of times under control of a logic condition. A matching **end** delineates the statements.

General syntax of a while loop is:

```
while expression
   % some loop that is executed when expression is true
end
```

Examples for both MATLAB and C++:

MATLAB	C++

`function s= sumTill(N)`
`%sums numbers from 1 - N`
`   s = 0;`
`   counter = 1;`
`   while (counter <= N)`
`      s = s + counter;`
`      counter = counter + 1;`
`  end`
 	
`Note:  the function name "sum" could not be used because there is an intrinsic fn by the name of "sum"`
`ck:  lookfor sum`
	`int sumTill(int N)`
	`//sums numbers from 1 - N`
	`{`
	`   int sum = 0;`
	`   int counter = 1;`
	`   while (count <= N)`
	`   {`
	`      sum = sum + counter;`
	`      counter = counter + 1;`
	`   }`
	`   return sum;`
	`}`

MATLAB	C++

`x = 1;`
`y = 10;`
`while (x < y)       %the ( ) are`
`   x = x + 1;       % optional`
`   y = y - 1;`
`end`
	`int x = 1;`
	`int y = 10;`
	`while (x < y)`
	`{`
	`   x = x + 1;`
	`   y = y - 1;`
	`}`

`change = 1;`
`while change >= 0.001`
`   change = change/2;`
`end`
`disp(change);`
	`double change = 1;`
	`while (change >= 0.001)`
	`   change = change/2;`
	`cout << change << endl;`

5.2.2 for

A `for` loop is used to repeat a statement or a group of statements for a fixed number of times. General form:

```
for index = expr
    statement 1
    statement 2
    ...
    statement n
end
```

where `index` is the loop control variable and `expr` is the loop control expression--usually a vector. The loop body is executed once for each column in `expr`. Shortcut notation for `expr` is:
`start: increment: last`
But unlike C++, a MATLAB `for` loop can loop through a vector or array rather than just a simple count. See examples below.

MATLAB	C++

`s = 0;`
`for i = 1:10   %increment of 1`
`   s = s + i;`
`end`
`disp(s);`
	`int sum = 0;`
	`for (int i = 1; i <= 10; i++)`
	`   sum = sum + i;`
	`cout << sum << endl;`

Examples	Output

`for i = [2,3,4,5,10]`
`   c = 2 * i`
`end`
	`c =`
	`         4.00`
	`c =`
	`         6.00`
	`c =`
	`         8.00`
	`c =`
	`        10.00`
	`c =`
	`        20.00`

The control condition is a vector containing 5 elements. The `for` loop will be run 5 times. The value of `i` will be 2 the first time; 3 the second time; 4 the 3rd time; 5 the 4th time; and 10 the last time	

`>> s=0;`
`>> for i = 1:2:10`
`       s=s+i;`
`   end`
`>> s`
	`>> s =`
	`    25              %1+3+5+7+9`

The initial value of i will be: 1; The end value will be maximum of 10; the increment will be 2; Therefore the value of index i will be 1 3 5 7 9	

`>> for i = 1:2`
`      for j = 1:2`
`         A(i,j) = i/j`
`      end`
`   end`
	`A =`
	`        1.00         0.50            0`
	`        2.00         1.00         4.00`

The inner most loop will execute first. Therefore with i= 1; j will take on the values of 1 and then 2; i will then become 2; j will take on the values of 1 and then 2	

5.3 Selection: if

The **if** statement evaluates a logical expression and executes a group of statements when the expression is **true**. The optional **elseif** and **else** keywords provide for the execution of alternate groups of statement. An **end** keyword, which matches the **if**, terminates the last group of statements. The groups of statement are delineated by the four keywords--no braces or brackets are involved.

The general syntax of an **if/elseif/else** construct is:

```
if expression1
   % Executed when expression1 is true
elseif expression2
   % Executed when expression2 is true
```

```
elseif expression3
   % Executed when expression3 is true
 ...
elseif  expressionN
   % Executed when expressionN is true
else
   % Executed when expression 1 .. N are false
end
```

Valid syntax requires the **if** and the **end**, zero or more **elseif’s** and zero or one **else**. The design is very much like C++. Examples in both MATLAB and C++ follow:

MATLAB	C++

`if (i <= 10)`
`   j = 0;`
`else`
`   k = 0;`
`end`
	`if (i <= 10)`
	`{`
	`   j = 0;`
	`}`
	`else`
	`{`
	`   k = 0;`
	`}`

`x = rand;`
`if x < 1/3`
`   disp(‘x < 1/3’);`
`elseif x < 2/3`
`   disp(‘1/3 <= x < 2/3’);`
`else`
`   disp(‘2/3 <= x ’);`
`end`
	`#include <cstdlib>`
	`x = rand();`
	`if (x < 1/3)`
	`   cout << “x < 1/3” << endl;`
	`elseif (x < 2/3)`
	`   cout << “1/3 <= x < 2/3” << endl;`
	`else`
	`   cout << “2/3 <= x “ << endl;`