TOPIC 11 :

Handle Loop Problems Using For-End and While-End Command

Sometimes we need to control the flow of command execution in a program to create a decision making. For example we want the program loop back to repeat a group of commands for several times, or until they meet specified condition. MATLAB has two structures (commands) to handle loop problems, which are for-end and while-end.

1. *for* loops.

For loops allow a group of a commands to be repeated a fixed, predetermined number of times. The general format is

```
for variable = expression
   statement
   ...
   statement
end
```

The columns of the expression are stored one at a time in the variable while the following statements, up to the end, are executed. In practice, the expression is almost always of the form scalar : scalar, in which case its columns are simply scalars. The scope of the for statement is always terminated with a matching end.

Syntax

```
for k=start_ind : stop_ind,
....
end
```

where '*k*' is index (loop variable) running from '*start_ind*' to '*stop_ind*'. Statements in the loop are executed (*stop_ind - stop_ind +1*) times.

Example:

```
for n=1:10
    x(n)=sin(n*pi/10);
end
```

For the example we can say:' *for n equals one to ten evaluate all statements until the next end statements'*. After the n=10 case, the For Loops ends and any commands afther the end statement are evaluated.
There are some important notes about for loop, i.e:

1. Loop execution cannot be terminated by reassigning loop variable to '*stop_ind*'.
2. Statement *break* causes stop of the loop execution.
3. For loop can be nested as desired
4. For loop should be avoided whenever there is an equivalent array approach to solving a given problem.
5. to maximize speed of processing the loops, arrays should be preallocated before a for loop is executed.

Example:

```
k=3;
a = zeros(k,k)  % Preallocate matrix
for m = 1:k
   for n = 1:k
      a(m,n) = 1/(m+n -1);
   end
end
```

2. *while* loops.

Repeat statements an indefinite number of times (until *statement* becomes logical false).

Syntax:

```
while expression
        statements
end
```

The statements are executed while the real part of expression has all nonzero elements.
Expression is usually of the form expression rel_op expression where rel_op is ==, <, >, <=, >=, or ~=.
The scope of a while statement is always terminated with a matching **end**.

expression

Expression is a MATLAB expression, usually consisting of variables or smaller expressions joined by relational operators (e.g., count < limit) or logical functions (e.g., isreal(A)). Simple expressions can be combined by logical operators (&, |, ~) into compound expressions such as the following. MATLAB evaluates compound expressions from left to right, adhering to operator precedence rules.

(count < limit) & ((height - offset) >= 0)

Statements

Statements is one or more MATLAB statements to be executed only while the expression is true or nonzero.

example:

```
n=0; m=1;
while (1/m) > realmin,
m=(m+1)^2; n=n+1; end
disp(n)
```

this return:
10

Nonscalar Expressions

If the evaluated expression yields a nonscalar value, then every element of this value must be true or nonzero for the entire expression to be considered true. For example, the statement while (A < B) is true only if each element of matrix A is less than its corresponding element in matrix B.

Partial Evaluation of the Expression Argument

Within the context of an if or while expression, MATLAB does not necessarily evaluate all parts of a logical expression. In some cases it is possible, and often advantageous, to determine whether an expression is true or false through only partial evaluation.

For example, if A equals zero in statement **while (A & B)** , then the expression evaluates to false, regardless of the value of B. In this case, there is no need to evaluate B and MATLAB does not do so.

In **while (A | B),** if A is nonzero, then the expression is true, regardless of B. Again, MATLAB does not evaluate the latter part of the expression.

You can use this property to your advantage to cause MATLAB to evaluate a part of an expression only if a preceding part evaluates to the desired state.
Here are some examples.

```
while (b ~= 0) & (a/b > 18.5)
    if exist('myfun.m') & (myfun(x) >= y)
    if iscell(A) & all(cellfun('isreal', A))
```

Example

The variable **eps** is a tolerance used to determine such things as near singularity and rank. Its initial value is the machine epsilon, the distance from 1.0 to the next largest floating-point number on your machine. Its calculation demonstrates while loops.

```
eps = 1;
while (1+eps) > 1
   eps = eps/2;
end
eps = eps*2
```

Example 2 - Nonscalar ExpressionGiven matrices A and B,

A = B =

1	0
2	3

1	1
3	4

Expression	Evaluates As	Because
`A < B`	`false`	`A(1,1)` is not less than `B(1,1)`.
`A < (B + 1)`	`true`	Every element of `A` is less than that same element of `B` with `1` added.
`A & B`	`false`	`A(1,2) & B(1,2)` is false.
`B < 5`	`true`	Every element of `B` is less than `5`.