

## Script Files

A script file is a user-created file with a sequence of MATLAB commands in it. The file must be saved with a '.m' extension, and it is referred to as an M-file. A script file is executed by typing its name (without the .m extension) at the MATLAB command prompt. Each line in the M-file is executed as if it were typed in at the command prompt. A script file may contain any number of commands, including those that call built-in functions or user-defined functions. Script files are useful when you have to repeat a set of commands several times.

### Example Script file

```
x=0:0.1:1;  
y=exp(-0.5*x);  
plot(x,y)  
xlabel('Time (s)')  
ylabel('Concentration mol/l')
```

Save this file as myfile.m To run it at the MATLAB prompt type

```
>> myfile
```

Do not use a variable in a script file with the same name as the script file. The name of a script file must begin with a letter.

## Function files

Function files are like functions or subroutines in Fortran, functions in C, and procedures in Pascal. Certain built-in functions require user-defined functions to be written; e.g. the ordinary differential equation solvers require require user-defined functions to be written to define the ODE's to be solved.

### Example function file

```
function r=mag(x)
% returns value in r of the magnitude of the input vector x
% Called as r=mag(x)
r=sqrt(sum(x.*x));
```

Save this file as mag.m (the name of the function with the extension .m). Now initialize a vector y and run the function mag. (make sure that mag.m is in the MATLAB path).

```
>> y=[1 2 1]
>> mag(y)
```

The function mag can also be called in a script file.

The first line is the function definition line. In it the name of the function, the input variables (in this case x), and the output variables (in this case r) are declared. The syntax of the function definition line is:

```
function [output variables]=function_name(input variables)
```

```
e.g. function [c,d,E]=rate(a,b,c);    function [c]=rate(a,b,c)
      function c=rate(a,b,c);          function rate(a,b,c)
```

All comment lines after the function definition line are displayed when 'help mag' is typed at the MATLAB prompt. The first comment line is searched for keywords by the lookfor command.

The variables in a function file are local and are erased after execution of the function. The variables in a script file are left in the MATLAB workspace after execution of the script. To make a set of variables to be available to a function without passing the variables in the input list, use the **global** command. e.g., if you want the variables k1 and k2 to be available to a function, put the statement: global k1 k2 in the M-file that calls the function and in the function file.

## **Programming Constructions**

The basic programming constructions are described below. While there are other constructions available, these are the most commonly used.

### **Matlab Program Control**

Matlab uses decision and loop structures to control program execution. The following structures are available in Matlab

- if statements
- switch statements
- for loops
- while loops
- break statements

## If Statements

If statements can be used to implement branching in a program. There are three types :

```
(1)         if <condition>
              <program>
            end
```

<condition> is a MATLAB logical expression, usually with values 0 or 1. In MATLAB logical expressions such as  $a == b$  or  $a <= b$  have a value 0 if false or 1 if true. This structure allows the execution of the program only if the condition is true (value=1).

```
Example:     if (i > 0)
              j=i;
            end
```

```
(2)         if <condition>
              <program1>
            else
              <program2>
            end
```

if <condition> is false, then program2 is executed.

Example:           if (i > 0 )  
                    j=i;  
                    else  
                    j=-i;  
                    end

(3)                if <condition1>  
                    <program1>  
                    elseif <condition2>  
                    <program2>  
                    elseif <condition3>  
                    <program3>  
                    else  
                    <program>  
                    end

if <condition1> is not 0, then program1 is executed,  
if condition1 is 0 and if condition2 is not 0,  
then program2 is executed, and so on.

```
Example:      if (i > 0 )
               j=i;
               elseif(i < 0)
                 j=-i;
               else
                 j=0;
               end
```

## Switch Case statements

If you need to execute different commands for different results based on a single condition then use the switch case structure

```
switch expression [ scalar or string ]
case test1
    command set 1
case test 2
    command set 2
.....
otherwise
    command set last
end
```

If expression is scalar then case is executed if the scalar value matches the expression in the test. If expression is a string then case is executed if the strings match

```
Example:      switch units          % converts to centimeters
              case {'in','inch'}
                y = 2.54*x;
              case {'m','meter'}
                y = x*100;
              case { 'millimeter','mm'}
                y = x/10;
              case {'cm','centimeter'}
                y = x;
              otherwise
                units='unknown units'
                y = NaN;
              end
              disp ([num2str(x) ' ' units...
                    ' is ' num2str(y) ' cm'])
```

The test expression in the case statements can have more than one match.

## For loops

For loops are used to repeat a set of statements a specific number of times. The general form of a FOR statement is:

```
for i = <array>
    <program>
end
```

The program is executed for each value of the loop index  $i$  in  $\langle \text{array} \rangle$ . The loop index  $i$  often occurs in some essential way inside the program.  $\langle \text{array} \rangle$  may be expressed in any of its usual forms; e.g. 'for  $i=1:2:5$ ,  $\langle \text{program} \rangle$ ,end' executes  $\langle \text{program} \rangle$  three times, with  $i$  having the values 1,3, and 5 successively.

The loop index does not have to be an integer; e.g for  $s= 1.0: -0.1: 0.0$ , steps  $s$  with increments of  $-0.1$  from  $1.0$  to  $0.0$ . If the stepsize is omitted the default value of  $1$  is assumed. You can also write 'for  $i=[1\ 3\ 7\ 10]$ ,  $\langle \text{program} \rangle$ ,end' to have the loop execute 4 times with the specified values for  $i$ . You can even use a matrix (see MATLAB documentation).

## Example

```
function c=add(a,b)      % Matrix addition
% c=add(a,b). This is the function which adds
% the matrices a and b. It duplicates the MATLAB
% function a+b.
[m,n]=size(a);
[k,l]=size(b);
if m~=k | n~=l
    r='ERROR using add: matrices are not the same size';
    return
end
c=zeros(m,n);
for i=1:m                % default stepsize is 1
    for j=1:n
        c(i,j)=a(i,j)+b(i,j);
    end
end
end
```

```
function c=mult(a,b) % matrix multiplication
% c=mult(a,b). This is the matrix product of
% the matrices a and b. It duplicates the MATLAB
% function c=a*b.
[m,n]=size(a);
[k,l]=size(b);
if n~=k
    c='ERROR using mult: matrices are not compatible
    for multiplication'
    return
end
c=zeros(m,l);
for i=1:m
    for j=1:l
        for p=1:n
            c(i,j)=c(i,j)+a(i,p)*b(p,j);
        end
    end
end
end
```

## While loops

The general form of the while loops are

```
while <condition>
    <program>
end
```

where <condition> is a MATLAB function, as with the branching construction. The program program will execute successively as long as the value of condition is not 0 (true). While loops carry an implicit danger in that there is no guarantee in general that you will exit a while loop.

```
Example:      % find all powers of 2 below 1000
              v=0; i=1; num=1;
              while num < 1000
                  num = 2^i;
                  v = [v num];
                  i=i+1;
              end
              n=size(v,2);
              disp(v(1:n-1))
              disp(v)
```

## Break statements

Break statements allow you to exit for and while loops prematurely

### Example code

```
% find root of  $x^2-x-2=0$  by  
% fixed-point iteration:  $x=x^2-2$   
% roots are  $x=2$  and  $x=-1$   
x=input('Input guess for root of  $x^2-x-2=0$  ');  
i=0;  
while (abs(x^2-x-2)>1.e-3)  
    x=x^2-2;  
    i=i+1;  
    if(i>10), break, end  
end  
disp(x)
```

## Suggestions

Pointers about programming and programming in MATLAB in particular.

1. The indented style used in the programs above makes programs easier to read and facilitates checking program syntax and debugging.
2. The liberal use of comments in your program serves to document it, making it easier for others or yourself at a later date to understand the functioning of the program.
3. Put error checks for improper input with informative error messages in your programs, as in the examples above. Complicated programs build on simpler ones and these error messages will help you debugging.
4. In MATLAB, try to avoid loops in your programs. MATLAB is optimized to run the built-in functions. For a comparison, see how much faster  $A*B$  is than using the function defined above for matrix multiplication, particularly for large array sizes.
5. If you are having trouble debugging a program, get a small part of it running and try to build on that.