MATLAB Function Files

- A MATLAB function file (called an M-file) is a text (plain ASCII) file that contains a MATLAB function and, optionally, comments.
- The file is saved with the *function name* and the usual MATLAB script file extension, ".m".
- A MATLAB function may be called from the command line or from any other M-file.
- When the function is called in MATLAB, the file is accessed, the function is executed, and control is returned to the MATLAB workspace.
- Since the function is not part of the MATLAB workspace, its variables and their values are not known after control is returned.
- Any values to be returned must be specified in the function syntax.

MATLAB Function Files

- The syntax for a MATLAB function definition is:
  function [val1, … , valn] = myfunc (arg1, … , argk)
  where *val1* through *valn* are the specified returned values from the function and *arg1* through *argk* are the values sent to the function.
- Since variables are local in MATLAB, the function has its own memory locations for all of the variables and only the values (not their addresses) are passed between the MATLAB workspace and the function.
- It is OK to use the same variable names in the returned value list as in the argument. The effect is to assign new values to those variables. As an example, the following slide shows a function that swaps two values.

MATLAB Function files

Here are some example function definitions:

- function y=square(x)
- function av=average(x1,x2,x3,x4,x5)
- function printvalue(A)
- function B=readvalue()
- function [mean,sttdev]=analyse(tab)

Functions are the building blocks of your own programs. You should always try and divide your programming task into separate functions, then design, code and test each one independently. It is common to design from the top down, but build from the bottom up.

Example of a MATLAB Function File

```
 function [ a , b ] = swap ( a , b )
% The function swap receives two values, swaps them,
% and returns the result.  The syntax for the call is
%  [a, b] = swap (a, b)
temp=a;
a=b;
b=temp;
```

We then call the function from the matlab command line:

```
>> x = 5 ; y = 6 ;    [ x , y ] = swap ( x , y )
x =
    6
y =
    5
```

MATLAB Function Files

- Referring to the function, the comments immediately following the function definition statement are the "help" for the function. The MATLAB command:

```
>>help swap    %displays:
```

The function swap receives two values, swaps them,

and returns the result. The syntax for the call is

[a, b] = swap (a, b)

Common built-in functions

- Mathematical functions:
  - sum(x): sum of elements of a vector or columns of a matrix
  - mean(x): mean, similarly columnwise for matrix
  - rand, rand(m), rand(m,n): random numbers (uniform, interval [0,1])
  - abs(x)
  - sin(x), cos(x), tan(x)
  - exp(x), log(x), log10(x)
  - [value,index] = max(x), [value,index] = min(x)
  - sqrt(x)
- Rounding
  - ceil(x), floor(x): to $+\infty$ and $-\infty$ respectively
  - fix(x): nearest integer towards zero
  - round(x): to nearest integer
- String conversion
  - int2str, num2str, str2num

Exercises

1. Write a function called FtoC (ftoc.m) to convert Fahrenheit temperatures into Celsius. Formula is °C=(°F - 32) x 5/9 . Make sure the program has a help page. Test from the command window with:

```
FtoC(96)
help FtoC
```

2. Write a function (roll2dice.m) to roll 2 dice, returning two individual variables: d1 and d2. For example:

```
[d1 d2] = roll2dice
d1 =
     4
d2=
     3
```

Flow control

- **MATLAB has five flow control constructs:**
- • if statements
- • switch statements
- • for loops
- • while loops
- • break statements

The *if* construct

Has the the form:

```
if control_expr_1
   statement 1;
   statement 2; ...
elseif
    statement 1;
   statement 2; ...
else
    statement 1;
   statement 2; ...
end
```

For example,

```
if A > B
   'greater'
elseif A < B
   'less'
elseif A == B
   'equal'
else
   error('Unexpected situation')
end
```

Note that 'end' here is completely different from the index function 'end'

If statement

```
if (x < 10)
   disp(x);        % only displays x when x < 10
end
if ((0 < x) & (x < 100))
   disp('OK');    % displays OK if x between 0 and 100
end
```

You can build a chain of tests with 'elseif', as in:

```
if (n <= 0)
   disp('n is negative or zero');
elseif (rem(n,2)==0)
   disp('n is even');
else
   disp('n is odd');
end
```

If statements

You can embed, or 'nest' statements, as in:

```
if (a < b)
   if (a < c)
      disp(a);
   else
      disp(c);
   end
else
   if (b < c)
      disp(b);
   else
      disp(c);
   end
end
```

The indenting of nested statements is not necessary but is recommended.

While statements

- The 'if' statement allows us to execute a statement conditional on some logical expression. The 'while' statement adds repetitive execution to the 'if' statement.
- The format of the while statement is 'while *condition statement* end' and the statement is repeatedly executed while it is the case that the condition evaluates as true.

```
while x <= 10
   % execute these commands
end
```

- In other words the statement is executed over and over again until the condition becomes false.
- Note that an error in such a statement might lead to your program looping continuously. If your program is looping, press [Ctrl/c] in the command window to cancel it.

While statements

The while statement is useful when you do not know in advance how many times an operation needs to be performed. For example, this code finds the smallest power of two which is larger than the number n:

```
n=50;
p=1;
while (p < n)
   p = 2 * p;
end
disp(p);      % displays 64
```

While loops

- The break statement is sometimes useful to cancel a while loop in the middle of a number of statements:

```
while (1)
   req = input('Enter sum or "q" to quit : ','s');
   if (req=='q')
      break;
   end
   disp(eval(req));
end
```

For loops

- The 'for' statement is useful when you *do* know how many times you want to repeat a statement. It is just a special form of the while loop. To execute a statement 10 times with a while loop would require statements such as;

```
i=1;
while (i<=10)
   disp(i);
   i = i + 1;
end
```

- This can be written more succinctly with the *for* statement; its basic syntax is 'for *var*=*sequence statement* end'. For example, the loop above could be written:

```
for i=1:10
   disp(i);
end
```

For loops

- Note that in a for loop, the variable is set to each value in the sequence in turn and retains that value when it is referred to inside the loop.
- You can create sequences with increments different to one in the usual way with '*start*:*increment*:*stop*'. You can even loop through the values in an array. For example:

```
for even=2:2:100 …
for primes=[ 2 3 5 7 11 13 17 19 23 ] …
```

- For loops are executed efficiently without actually building an array from the sequence. This means that a loop of 1 million repetitions doesn't require an array of size one million.
- You can use the *break* statement in for loops as well as in while loops.

Example

```
function [n] = factorial (k)
% The function [n] = factorial(k) calculates and
% returns the value of k factorial. If k is negative,
% an error message is returned.
if (k < 0)
    n = 'Error, negative argument';
elseif k<2
    n=1;
else
   n = 1;
   for j = [2:k]
            n = n * j;
   end
end
```

Note that a MATLAB variable does not have to be declared before being used, and its data type can be changed by assigning a new value to it.

For example, the function factorial( ) below returns an integer when a positive value is sent to it as an argument, but returns a character string if the argument is negative.

Exercises

- 1. Print a Fahrenheit to Celsius Conversion table for each 5 degrees between 0 and 100 Fahrenheit. Use a *for* loop to invoke your FtoC() function.
- 2. Write an M-file to evaluate the equation $y(x)=x^2-3x+2$ for values of x between 0.1 and 3, in steps of 0.1.