Chapter 4: Programming in MATLAB

Chapter 4: Programming in MATLAB

4.1– Introduction

We will see in this chapter, the simplest type of MATLAB program is called a *script*. A script is a file that contains multiple sequential lines of MATLAB commands and function calls. You can run a script by typing its name at the command line.

4.2– Functions

We have seen a number of predefined functions. It is possible to define your own functions. The first method allows you to define simple functions on a command line. The second, much more general, allows you to define very advanced functions by defining it in a file.

4.2.1– Inline functions

Let's say I want to define a new function that I call sincos defined mathematically by:

sincos(x) = sin(x) - x cos(x)

We will write:

```
>> sincos = inline('sin(x)-x*cos(x)', 'x')

sincos =

Inline function:

sincos(x) = sin(x)-x*cos(x)
```

It's very simple. Don't forget primes, which define character strings in MATLAB. We can now use this new function:

```
>> sincos(pi/12)
years =
0.0059
```

Now let's try applying this function to an array of values:

```
>> sincos(0:pi/3:pi)

??? Error using ==> inline/subsref
```

*Error in inline expression ==> sin(x)-x*cos(x)*

*??? Error using ==> **

Inner matrix dimensions must agree.

It doesn't work. Did you understand why? (this is the same problem as for the plot(x,x*sin(x)) instruction seen previously) MATLAB tries to multiply x row vector by cos(x) also row vector in the sense of matrix multiplication! Therefore it is necessary to use a term by term multiplication .* in the definition of the function:

>> **sincos = inline('sin(x)-x.*cos(x)')**

sincos =

Inline function:

*sincos(x) = sin(x)-x.*cos(x)*

>> **sincos(0:pi/3:pi)**

ans =

0 0.3424 1.9132 3.1416

We can therefore state the following rule:

When defining a function, it is preferable to systematically use the term-by-term operators: .*, ./ and .^ instead of * / and ^ if we want this function to be able to apply to tables .

We can similarly define functions of several variables:

>> **sincos = inline('sin(x)-y.*cos(x)', 'x', 'y')**

sincos =

Inline function:

*sincos(x,y) = sin(x)-y.*cos(x)*

The order of the variables (displayed on the screen) is that in which they appear in the function definition. To try :

>> sincos(1,2)

4.2.2- Functions defined in a file

In mathematics we often write functions in the form:

y = f(x) or z = g(x, y) or ....

We will see that it is the same thing under MATLAB. Function names should be more explicit than f or g. For example, a function returning the square of an argument x could be named square (we avoid accents):

y = square(x)

In the file defining the function, this syntax is also preceded by the function keyword:

General form of a function in a functionname.m file:

function Result = functionname(Parameters_Passes_to_the_Function)

% comment lines (descriptive help appearing in the % command window if you type: help functionname)

command lines

...

Result=...

Let's start by taking the previous example (sincos). The order of operations is as follows:

1. Edit a new file called sincos.m

2. Type the following lines:

function s = sincos(x)

% sincos calculates sin(x)-x.*cos(x)

s = sin(x)-x.*cos(x);

3. Save

The result is the same as before:

>> sincos(pi/12)

years =

0.0059

We will notice several things:

- the use of .* so that the function is applicable to tables.

- the variable s is only there to specify the output of the function.

- the use of; at the end of the calculation line, otherwise MATLAB would display the function result twice.

So one more rule:

When defining a function in a file, it is preferable to put '**;**' at the end of each command constituting the function. However, be careful not to put any on the first line.

Another important point: the file name must have the extension .m and the file name without suffix must be exactly the name of the function (appearing after the function keyword)

As for where in the tree the file should be placed, the rule is the same as for scripts.

Now let's see how to define a function with multiple outputs. We want to create a function called cart2pol which converts Cartesian coordinates (x; y) (inputs of the function) into polar coordinates (r; fi) (outputs of the function). Here is the content of the cart2pol.m file:

```
function [r, theta] = cart2pol (x, y)
% cart2pol converts Cartesian coordinates (x; y) (function inputs)
% in polar coordinates (r; fi) (function outputs)
% sqrt = square root (Matlab internal function)
% atan = reciprocal tangent function (internal function of Matlab)
r = sqrt(x.^2 + y.^2);
theta = atan (y./x);
```

Note that the two output variables are enclosed in square brackets and separated by a comma.

To use this function, we will write for example:

```
>> [module,phase] = cart2pol(1,1)
module =
1.4142
phase =
0.7854
```

We therefore simultaneously assign two variables module and phase with the two outputs of the function, by putting these two variables in square brackets, and separated by a comma.

It is possible to retrieve only the first output of the function. MATLAB often uses this principle to define functions that have a main output and optional outputs.

So for our function, if only one output variable is specified, only the polar radius value is returned. If the function is called without an output variable, ans takes the value of the polar radius:

```
>> cart2pol(1,1)
```

ans =

1.4142

Exercise: Create a function: vec2col

Write the vec2col.m function which transforms any vector (row or column) passed as an argument into a column vector. Error processing (error function) will test that the variable passed to the function is indeed a vector and not a matrix (use size and the conditional if statement described later in the course).

Exercise: First and second degree equations

Complete the 2 functions whose first 2 lines are respectively:

```
function x=equ1(a,b)
% resolves a*x+b=0
```

And

```
function x=equ2(a,b,c)
% resolves a*x^2+b*x+c=0
```

which solve the first and second degree equations. Use the nargin function so that the call to equ2(a,b, c) is valid. Check that the solutions are correct.

Exercise: Nargin, nargout

Research with help the syntax and usage of these commands. Apply them to equ1 and equ2.

4.2.3-Variable scope

Inside functions like the one we just wrote, you have the right to manipulate three types of variables:

- The function's input variables. You cannot modify their values.
- The output variables of the function. You must assign a value to them.
- Local variables. These are temporary variables to split calculations for example. They only have meaning inside the function and will not be “seen” from the outside. And THAT'S ALL!

Now imagine that you write a command file, and a function (in two different files of course), the command file using the function. If you want to pass a value A from the batch file to the function, you normally have to pass through an input to the function. However, sometimes we would like the variable A of the command file to be usable directly by the function.

For this we use the global directive. Let's take an example: you have a relationship giving the Cp of a gas as a function of the temperature T:

Cp(T) = AT3 + BT2 + CT + D

and you want to make it a function. We could make a function with 5 inputs, for T, A, B, C, D, but it is more natural for this function to have only T as an input.

How to pass A,B,C,D which are constants? Answer: we declare these variables globally both in the command file and in the function, and we assign them in the command file.

Command file:

```
global A B C D
A = 9.9400e-8;
B = -4.02e-4;
C = 0.616;
D = -28.3;
T = 300:100:1000; % Temperature in Kelvins
plot(T, cp(T)) % We plot the CP as a function of T
```

The function:

```
function y = cp(T)
global A B C D
y = A*T.^3 + B*T.^2 +C*T + D;
```

Exercise: Local variable

Write a base name script containing the single line:

```
a=12345;
```

Run base, then type in the command window:

```
a ↲
```

Type the local name function containing the 2 lines into the editor:

```
function local
a=0
```

Execute the local function, then type in the command window:

```
a ↲
```

Explain the results displayed.

Exercise: Global variable

Write a globa name function containing the lines:

```
% function illustrating the concept of global variable
```

% see also globb

global A

A=101010;

Write a globb name function containing the lines:

function globb

% function illustrating the concept of global variable

% see also globa

global A

available(A)

Execute globa, then globb. Tap A↵ in the command window. Now tap global A in the command window, then A↵. Explain the results displayed. Explain the difference between the clear A and clear global A commands.

4.3-Control structures

We are going to talk here about tests and loops. Let's start with comparison operators and logical operators.

4.3.1-Comparison and logical operators

Let us first note the following important point, inspired by the C language:

MATLAB represents the logical constant "**FALSE**" as **0** and the constant "**TRUE**" as **1**. This is particularly useful, for example for defining functions piecewise. Then, the following two tables contain most of what you need to know:

4.3.1.1-MATLAB Syntax Operator

We may wonder what happens when we apply a comparison operator between two arrays, or between an array and a scalar. The operator is applied term by term. So:

```
>> A=[1 4 ; 3 2]

A =
1 4
3 2
>> A> 2

ans =
0 1
1 0
```

Operator	MATLAB syntax
equal to	==
different from	~=
greater than	>
greater than or equal to	>=
less than	<
less than or equal to	<=
negation	~
or	|
and	&

Terms of A greater than 2 give 1 (true), the others 0 (false).

An important application of logical operators concerns the automatic processing of large matrices. Let's create a matrix (2X5, small for the example):

```
>> A=[10, 12, 8, 11, 9; 10, 10, 128, 11, 9];
A =
10 12 8 11 9
10 10 128 11 9
```

Element 128 deviates greatly from all others. For example, it may be an aberrant point that appeared during an experimental measurement and we wish to replace values greater than 20 with the value 10. For a small matrix such as A, this is easily achievable, simply tap: A(2, 3)=10; But this method is unrealistic if we have to do with a 1000X1000 matrix! Instead, let's use:

```
>> A(A>20)=10;
```

And that's it:

```
A =
10 12 8 11 9
10 10 10 11 9
```

We can also construct functions piecewise. Let's imagine that we want to define the following function:

f(x) = sin(x) if x > 0 otherwise f(x) = sin(2x)

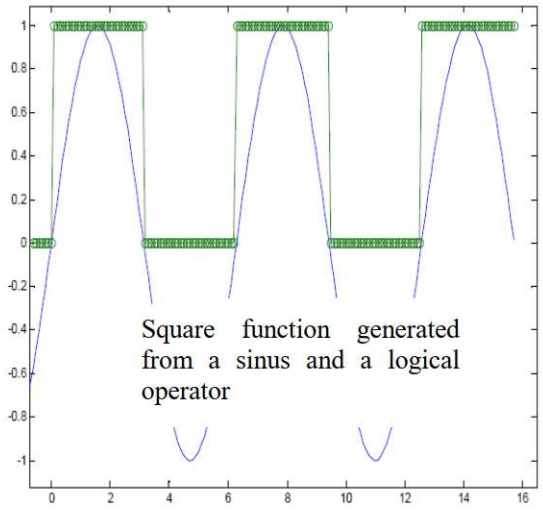
Here's how to write the function:

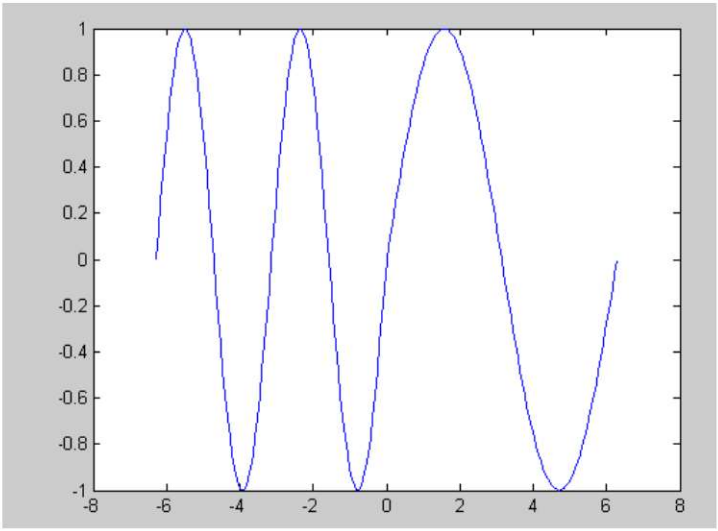
```
>> f = inline('sin(x).*(x>0) + sin(2*x).*(~(x>0))', 'x')
f =
Inline function:
f(x) = sin(x).*(x>0) + sin(2*x).*(~(x>0))
```

We add the two expressions sin x and sin 2x by weighting them by the logical condition defining their domains of validity. Simple but effective! Let us represent the function thus defined:

```
>> x=-2*pi:2*pi/100:2*pi;
>> plot(x,f(x))
```

gives the following curve on the left:





Example: Using logical operators

From a sine function, create a square function such as the one above on the right.

4.3.2-The find command

The find command is useful for simply extracting elements from an array according to a given logical criterion.

>> k = find(x)

returns in the variable k the list of indices of the array x whose elements are non-zero.

Knowing that "FALSE" and 0 are identical in MATLAB, this allows you to find all the indices of a table respecting a given logical criterion:

>> x = [-1.2, 0, 3.1, 6.2, -3.3, -2.1]

x =

-1.2000 0 3.1000 6.2000 -3.3000 -2.1000

>> inds = find (x < 0)

inds =

1 5 6

We can then easily extract the subarray containing only the negative elements of x by simply writing:

>> y = x(inds)

y =

-1.2000 -3.3000 -2.1000

Remember this example carefully. This type of sequence proves very useful in practice because here again, it saves an (if) test. Finally, note that the find command also applies to 2D tables.

Exercise: Reshape and find

Explain precisely what the following command does:

>>a=reshape(fix((rand(1,400)-0.5)*100),20,20);

Find all elements with value equal to 7.

Transform matrix a into a column vector b, sort b in ascending order. How many elements have a value of 7?

Hints: find, sort.

WARNING: unless you add an explicit comment, this type of expression should be avoided because it makes understanding the code difficult.

4.3.3-Conditional if statements

The syntax is as follows:

if logical condition

instructions

elseif logical condition

instructions

...

else

instructions

end

Note that there is no need for parentheses around the logical condition. Note that it is often possible to avoid blocks of this type by directly exploiting the comparison operators on arrays (see the square function and the piecewise function defined in the previous section), as well as the find command.

*Exercise: Function with if and is**

A number of functions detect the state of MATLAB entities. Create a parity(x) function which returns the parity of the number x ('x is even' or 'x is odd'), specifies whether it is a prime number ('x is prime'), processes the case where x is not a number and where x is not integer (using the integer function). The function must use the isnumeric, isprime functions and the if...elseif...end control commands.

4.3.4-for loops

for variable = start value: step: end value

instructions

end

Example:

```
for i = 1:m
for j = 1:n
H(i,j) = 1/(i+j);
end
end
```

Exercise: Conditional loop, break

How many times is this loop evaluated? What are the values of i displayed on the screen?

```
for i=12:-2:1;
available(i)
if i==8
break
end
end
```

Exercise: Tick and knock

Write a script using the tic and toc functions and one (or more) loop(s) such that the elapsed time is of the order of a second. Is the result always the same?

4.3.5-While loops

while logical condition

instructions

end

Exercise: Loop while

What does the following script do?

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

```
end
eps = eps*2
```

4.3.6-return

A return to the calling function, or command window, is caused when MATLAB encounters the return command.

4.3.7-Switch, case, otherwise

Exercise: Switch

What does the script below do?

```
clc
color = input('Enter a color: ', 's');
color=lower(color);
color switch
'red' box
string='The chosen color is: red';
'green' box
string='The chosen color is: green';
box 'blue'
string='The chosen color is: blue';
otherwise
string='Unknown color';
end
disp(string)
```

Exercise: Implicit loops

Explicitly write the vector u defined by implicit loops:

```
u(1:2:10)= (1:2:10).^2;
u(2:2:10)= (2:2:10).^3;
```

Explain the results.