

MATLAB Lecture 4. Programming in MATLAB

In this lecture we will see how to write scripts and functions. Scripts are sequences of MATLAB statements stored in a file. Using conditional statements (if-then-else) and loops (for and while), we can write more useful functions. At the end we illustrate functions used as parameters in other functions.

4.1 Setting the Search Path

To ensure that MATLAB will find the programs you create, you may have to adjust the search path of MATLAB. To see the current search path, simply type **path**.

To append to the current path, the drive **h:**, we can execute

```
>> path(path, 'h:');
```

You can also adjust the path from the file menu.

Alternatively, we could set the current directory to where our m-files are:

```
>> pwd      % print working directory
>> cd h:    % change directory to h:
>> ls      % list directory content
```

In the latest versions of MATLAB, the environment supports editing and debugging m-files. Select **new** from the **file** menu to begin a new MATLAB program. To modify existing m-files, we choose **open** from the **file** menu.

4.2 m-Files

The files of MATLAB programs have the **.m** extension. One basic application is to group a sequence of commands. For example, a template for the file **my_script.m** may look like

```
function my_script
% comment about what the script does
sequence of statements
```

If the search path is set right (or when **my_script.m** is in the right directory), executing

```
>> type my_script
```

will display the definition of **my_script.m**.

Our first example sums two numbers. We create a file **my_sum.m** with content:

```
function s = my_sum(a,b)
% returns the sum of a and b
s = a+b;
```

In general, the keyword **function** is followed by an equation. The right hand side of that equation starts with the name of the function, followed by the sequence of arguments, enclosed by round brackets. *The name of the function must correspond to the name of the file.* Functions may return multiply multiple results, as the file **sumavg.m** defines a function that returns sum and average of two numbers:

```
function [sum,avg] = sumavg(a,b)
% returns the sum and average of a and b
sum = a+b;
avg = (a+b)/2;
```

MATLAB does not require type declarations, which allows flexible programming.

4.3 Input and Output

To print results in a MATLAB function we use the the **fprintf** statement. The syntax is very similar to the corresponding C command. The function **input** allows the user to provide input data from the keyboard. A simple example is below.

```
function areacircle
% interactive program to compute area of circle
r = input('Give radius of circle : ');
a = pi*r^2;
fprintf('The area of circle with radius %.2f is %.6f \n', r, a);
```

Note that strings are enclosed with single quotes (not the double quotes as in C). To read a string from standard input, we need to provide the extra argument 's' to **input**. For example, the statement

```
z = input('Your name please : ','s');
```

allows the user to type in a string, which will be assigned to z.

4.4 The if Statement

The general form of the **if** statement is

```
if expression
    sequence of statements
elseif expression
    sequence of statements
else
    sequence of statements
end
```

Note that the keyword **end** is needed to terminate the **if** statement.

4.5 Loop Statements

MATLAB has a **for** and a **while** loop. Just as with the **if**, the **end** is needed to terminate the loop statement.

A generic use of the **for** loop is

```
for i = start_range:increment:end_range
    sequence of commands
end
```

The increment may be omitted if it correspond with the default value one. For example in **for** $i=1:10$, i will take the values 1 through 10. The increment can be any number, as in **for** $i=1:0.1:10$, or may be negative, like in **for** $i=10:-1:1$, where i ranges over all integer numbers from 10 down to 1.

We have seen the syntax for ranges when we created vectors to define ranges to sample functions for plotting, e.g.: $\mathbf{t} = -\pi:0.1:\pi$. While the vector \mathbf{t} can be created with a **for** loop, this construction will be very inefficient as it requires MATLAB to increase the storage for \mathbf{t} dynamically each time a new element is added to the vector.

The general structure of the **while** loop looks like

```

while condition
    sequence of commands
end

```

With **break**, we exit the innermost **for** or **while** loop.

4.6 Functions that use other functions

The argument to a function may be a function name. For example:

```

function [a,b,fail] = bisect(f,a,b,eps,N)
%
% Applies the bisection method to the function f on [a,b],
% f is assumed to be continuous and f(a)*f(b) < 0.
% Stops when |f(a)| < eps or |f(b)| < eps or |b-a| < eps.
% Failure is reported (fail = 1) when the accuracy requirement
% is not satisfied in N steps; otherwise fail = 0 on return.
%
% Example :
% >> [a,b,fail] = bisect('cos',pi/4,2*pi/3,0.0001,100)
%
fprintf('running the bisection method...\n');
for i = 1:N
    m = (a+b)/2;
    fm = feval(f,m);
    if fm*feval(f,a) < 0
        b = m;
    else
        a = m;
    end;
    fprintf(' a = %f, b = %f, m = %f, f(m) = %f\n', a,b,m,fm);
    if (abs(fm) < eps) | ((b-a) < eps)
        fail = 0;
        fprintf('succeeded after %d steps\n', i);
        return;
    end
end
fprintf('failed requirements after %d steps\n', N);
fail = 1;

```

The implementation of **bisect** uses the **feval** command to evaluate the function **f**. Suppose the function **f** is defined by **myfun.m**, then we call **bisect** like

```
>> [a,b,fail] = bisect('myfun',0,3,1.0e-4,10)
```

Typing

```
>> feval('sin',pi/2)
```

is equivalent to typing `sin(pi/2)`.

4.7 Assignments

1. Give an implementation for the function `y = staircase(x)` defined as follows :

$$y = \begin{cases} 0 & \text{if } x \leq 1 \\ 1 & \text{for } 1 < x \leq 2 \\ 2 & \text{if } 2 < x \end{cases}$$

Use `staircase` to make a plot of the function.

2. To plot a function, we first must generate a range, evaluate the function at that range, and then execute the plot command. Write an implementation for the function `plotfun(f,a,b,n)` to plot the function `f` over the interval `[a,b]` using `n+1` equally spaced points.
3. The derivative of a function `f` can be approximated by $\frac{f(x+h)-f(x)}{h}$, for $h > 0$.
Write an m-file to define the function `derive` that takes as input a function `f` and two numbers: `h` and `x`. On return is the approximation for the derivative $f'(x)$, computed using the formula above.
4. Give the m-file to implement the function `avgfun` which takes the name of a function and a vector as arguments and returns the average of the function values evaluated at the vector. For a function `f`, and for a vector `x` of size `n`, `avgfun` returns

$$\frac{1}{n} \sum_{i=1}^n f(x_i)$$

Complete the code below

```
function avgfun(f,x)
% returns average of f evaluated at x
```

5. Write a MATLAB function to implement the composition of two functions.

Complete:

```
function y = compose(f,g,x)
%
% returns f(g(x))
%
```

Give is the MATLAB command to compute `sin(cos(3))`, using `compose`.

6. As you may know, $\sqrt{\dots\sqrt{\sqrt{2}}}$ converges to one.

Complete the following m-file

```
function a = apply (f,x0,n)
%
% applies the function f n times starting at x0,
% e.g, apply(f,x0,0) returns x0
% apply(f,x0,1) returns f(x0)
% apply(f,x0,2) returns f(f(x0)), etc...
%
```

