SM-2302: Software for Mathematicians

Matlab2: Programming in MATLAB

Dr. Huda Ramli
Mathematical Sciences, Faculty of Science, UBD
huda.ramli@ubd.edu.bn

Semester I, 2023/24

Language basics

1. **Variables**: No declaration

```
n = 25 % integer
a = 6.3 % real number
t = 'hello' % text
```

2. **Data type available**:

`double(n)`	double precision floating-point number, 64 bits in length
`single(n)`	single precision floating-point number, 32 bits in length
`int8(a), int16(a)`	8-bit, 16-bit signed integers
`nt32(a), int64(a)`	32-bit, 64-bit signed integers

By default, MATLAB performs operations on numbers of type `double` using double-precision arithmetic. For most numerical purposes, `double` is the recommended data type.

3. **Complex numbers**:

```
i, j % imaginary unit
sqrt(-1) % imaginary unit
x = 3+4i % complex number
x = complex(a,b) % real part is a, imaginary part is b
real(x) % real part of x
imag(x) % imaginary part of x
angle(x) % argument of x
abs(x) % amplitude of x
conj(x) % conjugate of x
isreal(x) % determine whether x is real or not
```

4. **Math expressions**:

```
a^3 + b^2 -3*c + d/6 + 9
abs(x) % absolute value
sin(x), cos(x), tan(x) % trigonometric functions
exp(x) % exponential

% and many more...

s = 1 - 1/2 +1/3 - 1/4 + 1/5 - 1/6 + 1/7 ...
-1/8 + 1/9 - 1/10 + 1/11 -1/12;
% use 3 dots (ellipsis) to combine long statements
```

Relational and Logical operators

The operation of many branching constructs is controlled by an expression whose result is either true(1) or false(0).

Relational operators

MATLAB	operation	MATLAB	operation
==	equal to	~=	not equal to
>	greater than	>=	greater than or equal to
<	less than	<=	less than or equal to

Logical operators

MATLAB	operation
&	logical AND
|	logical OR
xor	logical exclusive OR (not both)
~	logical NOT (not both)

Example 1

Determine what is produced by the logical statements:

a	b	a & b	a | b	xor(a,b)	~a
0	0				
0	1				
1	0				
1	1				

Inputs		and	or	xor	not
a	b	a & b	a | b	xor(a,b)	~a
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Logical functions

	Description	Example

`any(x)`	**Vector:** returns 1 if ANY elements are nonzero, 0 otherwise	`>> x = [0 1 0 1];`
		`>> any(x)`
		`ans = 1`

`any(A)`	**Matrix:** operates on columns, returning a row vector of 1s and 0s	`>> A = [1 0 0; 0 0 0];`
		`>> any(A)`
		`ans = 1 0 0`

`all(x)`	**Vector:** returns 1 if ALL elements are nonzero, 0 otherwise	`>> all(x)` `ans = 0`
`all(A)`	**Matrix:** operates on columns, returning a row vector of 1s and 0s	`>> all(A)` `ans = 0 0 0`

`find(B)`	finds the indices of the nonzero elements of matrix **B**	`>> B = [1 0 0; 0 2 4];`
		`>> find(B)`
		`ans = 1 4 6`

Other useful functions are `isnan` and `isempty`.

Control flow

MATLAB has several constructs that allow for varying ways to control the flow of program execution.

- Iteration:
  - `while` loop
  - `for` loop
- Selection:
  - `if-else` statement
  - `if-elseif` statement
  - `switch-case-otherwise` statement

Iteration: `while` loop

The `while` loop repeats a group of statements and indefinite number of times under control of a logic condition. A matching `end` delineates the statements.

```
while expression
    % some loop that is executed when expression is true
end
```

```
%% Example 1:
% sums numbers from 1 to 100
s = 0; N = 100;
count = 1;
while (count <= N)
    s = s + count;
    count = count + 1;
end
```

```
%% Example 2:
x = 1;
y = 10;
while x < y   % the ( ) are ...
    optional
    x = x + 1;
    y = y - 1;
end
```

Iteration: `for` loop

A `for` loop is used to repeat a statement or a group of statements for a fixed number of times:

```
for index = expr
    statement 1
    statement 2
    ...
    statement n
end
```

```
%% Example 1:
% sum of an array
s = 0;
b = rand(100,1);
for i = 1:100
    s = s + b(i);
end
s
```

```
%% Example 2:
% nested loop
m = 50; n = 100;
b = rand(100,1);
for i=1:m   % stripe 1
    for j = 1:2:n  % stripe 2
        H(i,j) = 1/(i+j);
    end
end
```

Selection: `if` statements

- `if` statement evaluates a logical expression and executes statements when the expression is true.
- optional `elseif` and `else` keywords provide for the execution of alternate statements.

```
if expr1
    % executed when expr1 is true
elseif expr2
    % executed when expr2 is true
elseif expr3
    % executed when expr3 is true
...
elseif exprN
    % executed when exprN is true
else
    % executed when expr1 ... N are false
end
```

Example:

```
% determine the range of a random integer
a = randi(100,1);
if a<30
    fprintf('%d is smaller than 30. \n', a);
elseif a>80
    fprintf('%d is larger than 80. \n', a);
else
   X = [num2str(a),' is between 30 and 80.'];
    disp(X)
end
```

Note:

- `fprintf` writes formatted data in the form: fprintf(string format, variables)
- `%d` denotes integer value
- `\n` introduces a new line

Selection: `switch-case` statements

The `switch-case` statement executes groups of statements based on the value of a variable or expression. The keywords `case` and `otherwise` delineate the groups. Only the first matching case is executed.

```
switch variable
  case case_value1
     statements1
  case case_value2
     statements2
     ...
   otherwise
     statements
end
```

```
% switch-case example:
n = 2
switch(n)
    case 1
       M = eye(n)
    case 2
       M = zeros(n)
    case 3
       M = ones(n)
end
```

Script and Functions

- A **script M-file** contains lines of commands that are executed when the name is entered at the command window prompt.
- A **function M-file** contains lines of commands that are executed when the function is called. The first line of the function must have the form:

  ```
  function[out1,out2, ...] = FunctionName(in1,in2,...)
  ```

- To create an M-file, enter in the command window: `>> edit ScriptName.m`

```
function PlotCircle(r)
%  This function plots a circle of radius r
%  centered at the origin.
if nargin < 2
    if nargin == 0
        r = 1;
    elseif r <= 0
       error('The input value should be > 0.')
    end
    theta = linspace(0, 2*pi, 200);
    x = r*cos(theta);
    y = r*sin(theta);
    plot(x, y)
    axis([-2*r,2*r,-2*r,2*r])
    axis square
else
  error('Too many input values.')
end
```

We can call the function to generate plots of circles.
For example, enter `>> PlotCircle(2)` to plot a circle with radius 2.

- Use percent symbol % to write comments and explain the purpose of functions and scripts.
- `nargin` counts the number of input arguments to a function. The default value is chosen as $r = 1$ if no input value for `r` is specified.
- Error is reported for too many inputs and for negative input. The `error` command terminates the computation when executed.

Anonymous Function handle

`FunctionName = @(arglist) expression`

- **One argument**

```
my_func = @(x) x.^2 + exp(x) + 5;
my_func(5)
```

- **Two arguments**

```
my_func2 = @(x,y) x.^3 + 6*sqrt(y);
my_func2(3,4)
```

Note that it is always best to use array operators (`./` and `.*`) when possible, so that functions can be evaluated at arrays of x-values.

The MATLAB function `tic` and `toc` can be used to find the time it takes to run a piece of code. For example, consider the following MATLAB statements:

```
n = 200;
f = @(x) 1 ./ (1 + x.*x);
tic % timer is started
for i = 1:n
     x =  rand(n,1);
     y = f(x);
end
toc % timer is stopped
```

Example 2

Write a script m-file containing three types of function constructs to define the Runge function:

$$f(x) = \frac{1}{1+x^2}$$

1. a separate function m-file named `runge.m`
2. an `inline` construct: `Runge = inline('1./(1+x.^2)');`
3. the annonymous function `f` handle above

Run the three codes for each of $n = 100, 200, 300, 400, 500$. What do you observe? Repeat the experiment and observe the differences in the timings.