



JOINT INSTITUTE
交大密西根学院

VG101: Introduction to Computers and Programming

Lecture 04

Xiaodong Wei



Class/Lab Rescheduling for Oct 12 & 14

- Oct 12 (Sat): moved to **Oct 19**, 16:00-17:40, DXY305
- Oct 14 (Mon): **online**, 16:00-17:40
 - Feishu meeting link: <https://vc.feishu.cn/j/553163584>
 - Lecture will be recorded.
 - OH on Oct 16 (Wed): 12:30 – 14:30.
- Lab1 (Oct 12): Longbin 324 for both sessions
- Lab2 (Oct 14)
 - Session 1: Longbin 310B
 - Session 2: Longbin 310E
- Contact TAs if you have time conflicts.

Outline

- Subfunctions
- Characters and strings
- Control statements

Local Functions (Subfunctions)

- M-files can contain more than one function.
- In a function file, the first function is called the *main function*.
 - Visible to other files.
 - Can be called from the Command Window.
- Additional functions within the file are called *local functions*.
 - Can appear in any order after the main function.
 - Only visible to the same file.
 - Cannot be called from the Command Window.
- A script file can contain commands and local functions.
 - Local functions must be defined after all the commands/statements.
 - Such functions only visible to this script file.
- Variables cannot be shared from within a function to outside, or vice versa.
- Example...

Characters and Strings

Characters and Strings

- In MATLAB, the term *string* traditionally (before R2016b) refers to an array of Unicode characters.
- Specify character data by placing characters inside a pair of single quotes (character vector).
- In MATLAB version after R2017a, you can create a string using double quotes (string scalar).
- We will focus on the traditional string.

Comparison, Searching, and Replacing

■ `tf = strcmp(s1, s2)`

- Compares two strings `s1` and `s2`, case-sensitive.
- Returns 1 (true) if `s1` and `s2` are identical, and 0 (false) otherwise.
- Text is considered identical if the size and content of each are the same.
- The return result `tf` is of data type logical.

■ `k = strfind(str, pat)`

- Searches `str` for occurrences of `pat`, case-sensitive.
- The output vector `k` collects the starting index of each occurrence of `pat` in `str`.
- Returns an empty array, `[]`, if `pat` is not found in `str`.

■ `newStr = strrep(str, old, new)`

- Replaces all occurrences of `old` in `str` with `new`.

Practice

- Take a look at the `strep` function and replace all the “good” with “great” in the following:

'This is a good problem with a good solution.'

Number <-> String

■ num2str

■ str2double

- `str2double('hello')`
- `str2double('2/3')`
- `str2double('2.3e3')`
- `str2double('1,200.34')`
- Text that represents a number can contain digits, a comma (thousands separator), a decimal point, a leading + or - sign, an e preceding a power of 10 scale factor, and an i or a j for a complex unit.

■ str2num (not recommended unless you know the subtleties)

- `str2num('2/3')`
- `str2num('1,200.34')`
- Evaluates the expression in the text.

Example

■ Parse the input command:

keyword parameter1 parameter2 ... add 1 2

```
% parse
command = input('Please type a command: ','s');
space = strfind(command, ' ');
if strcmp(command(1:space(1)-1), 'add')
    parameter1 = str2double(command(space(1)+1:space(2)-1));
    parameter2 = str2double(command(space(2)+1:end));
    result = parameter1+parameter2
end
```

Control Statements

If Statement

- Conditionally execute statements

- `if expression` The statements will be executed
 statements if expression is true or nonzero
 end

- `if expression1`
 statements1
 elseif expression2 Note that it's elseif, not else if
 statements2
 else
 statements3
 end

Relation Operators

Operator	Description	Example
<	Less than	<code>a < 1</code>
<=	Less than or equal to	<code>a <= 1</code>
>	Greater than	<code>a > 1</code>
>=	Greater than or equal to	<code>a >= 1</code>
==	Equal to	<code>a == 1</code>
~=	Not equal to	<code>a ~= 1</code>

Will return logical value `true` or `false`

Logical Operators and Functions

- MATLAB offers three types of logical operators and functions:
 - **Element-wise**: operate on corresponding elements of logical arrays.
 - **Bit-wise**: operate on corresponding bits of integer values or arrays.
 - **Short-circuit**: operate on scalar, logical expressions.

Element-Wise Operators and Functions

■ $A = [0 \ 1 \ 1 \ 0 \ 1]; B = [1 \ 1 \ 0 \ 0 \ 1];$

Operator	Description	Example
& (and)	Returns 1 for every element location that is true (nonzero) in both arrays, and 0 for all other elements	$A \& B =$ [0 1 0 0 1]
 (or)	Returns 1 for every element location that is true (nonzero) in at least one array, and 0 for the location that is false (0) in both arrays.	$A B =$ [1 1 1 0 1]
~ (not)	Complements each element of the input array	$\sim A =$ [1 0 0 1 0]
xor	Returns 1 for every element location that is true (nonzero) in only one array, and 0 for all other elements.	$\text{xor}(A,B)=$ [1 0 1 0 0]



Short-Circuit Operators

Operator	Description
&&	Returns logical 1 (true) if both inputs evaluate to true, and logical 0 (false) if they do not.
	Returns logical 1 (true) if either input, or both, evaluate to true, and logical 0 (false) if they do not.

& vs &&

- A & B
 - both A and B are evaluated (BUT short-circuit in `if/while` statements)
 - A, B can be arrays
- A && B
 - B is **only** evaluated if A is true (short-circuit)
 - A, B can only be scalars

Operator Precedence

1. Parentheses () ← Use parentheses if you are unsure
2. Transpose (.'), power (.^), complex conjugate transpose ('), matrix power (^)
3. Unary plus (+), unary minus (-), logical negation (~)
4. Multiplication (.*), right division (./), left division (.\), matrix multiplication (*), matrix right division (/), matrix left division (\)
5. Addition (+), subtraction (-)
6. Colon operator (:)
7. Less than (<), less than or equal to (<=), greater than (>), greater than or equal to (>=), equal to (==), not equal to (~=)
8. Element-wise AND (&)
9. Element-wise OR (|)
10. Short-circuit AND (&&)
11. Short-circuit OR (||)

Example

- Implement the `abs` function

Practice

- Write a function to determine if a character is a numeric digit or not

```
function ret = myisdigit(ch)
```

Switch Statement

- Executes certain statements based on the value of a variable or expression.
- `switch` expression (scalar or string)
 - `case` value1
statements % Executes if expression is value1
 - `case` value2
statements % Executes if expression is value2
 - ...
 - `otherwise`
statements % Executes if expression does not
 % match any case
- `end`

Example

```
switch input_num
    case -1
        disp('negative one');
    case 0
        disp('zero');
    case 1
        disp('positive one');
    otherwise
        disp('other value');
end
```

Example

```
switch var
    case 1
        disp('1');
    case {2,3,4}
        disp('2, 3 or 4');
    case 5
        disp('5');
    otherwise
        disp('something else');
end
```

Multiple conditions

While Loop

- Executes a statement or group of statements repeatedly as long as the controlling expression is true

■ **while** expression
statements
end

while true – endless loop

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

For Loop

- Executes a statement or group of statements for a predetermined number of times

- `for index = initVal:endVal` **Default step = 1**
 statements
 end

- `for index = initVal:step:endVal`
 statements
 end

- `for index = valArray`
 statements
 end

- Example:

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

```
for m = 1:2:10  
    disp(m);  
end
```

```
for m = 1:-0.2:0  
    disp(m);  
end
```

```
for m = [1 2 3 5]  
    disp(m);  
end
```


For Loop

■ Example:

```
for m = 1:5
    for n = 1:5
        A(m,n) = 1 / (m + n - 1);
    end
end
```

for loop can be nested

Row-Major Order and Column-Major Order

- Methods for storing multidimensional arrays in linear memory.

- Example: $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$

- Row-major order: 1 2 3 4 5 6
- Column-major order: 1 4 2 5 3 6
- MATLAB uses column-major order: `reshape`
- C uses row-major order

Tip: Row First or Column First?

- When looping for a two-dimensional array, putting column in the outer loop will be faster.
- Example...

Tip: Vectorizing Loops

- MATLAB is a matrix language, which means it is designed for vector and matrix operations. You can often speed up your M-file code by using vectorizing algorithms that take advantage of this design. Vectorization means converting **for** and **while** loops to equivalent vector or matrix operations.

Example

```
n = 0;  
for t = 0:0.1:10  
    n = n + 1;  
    y(n) = sin(t);  
end
```

```
t = 0:0.1:10;  
y = sin(t);
```