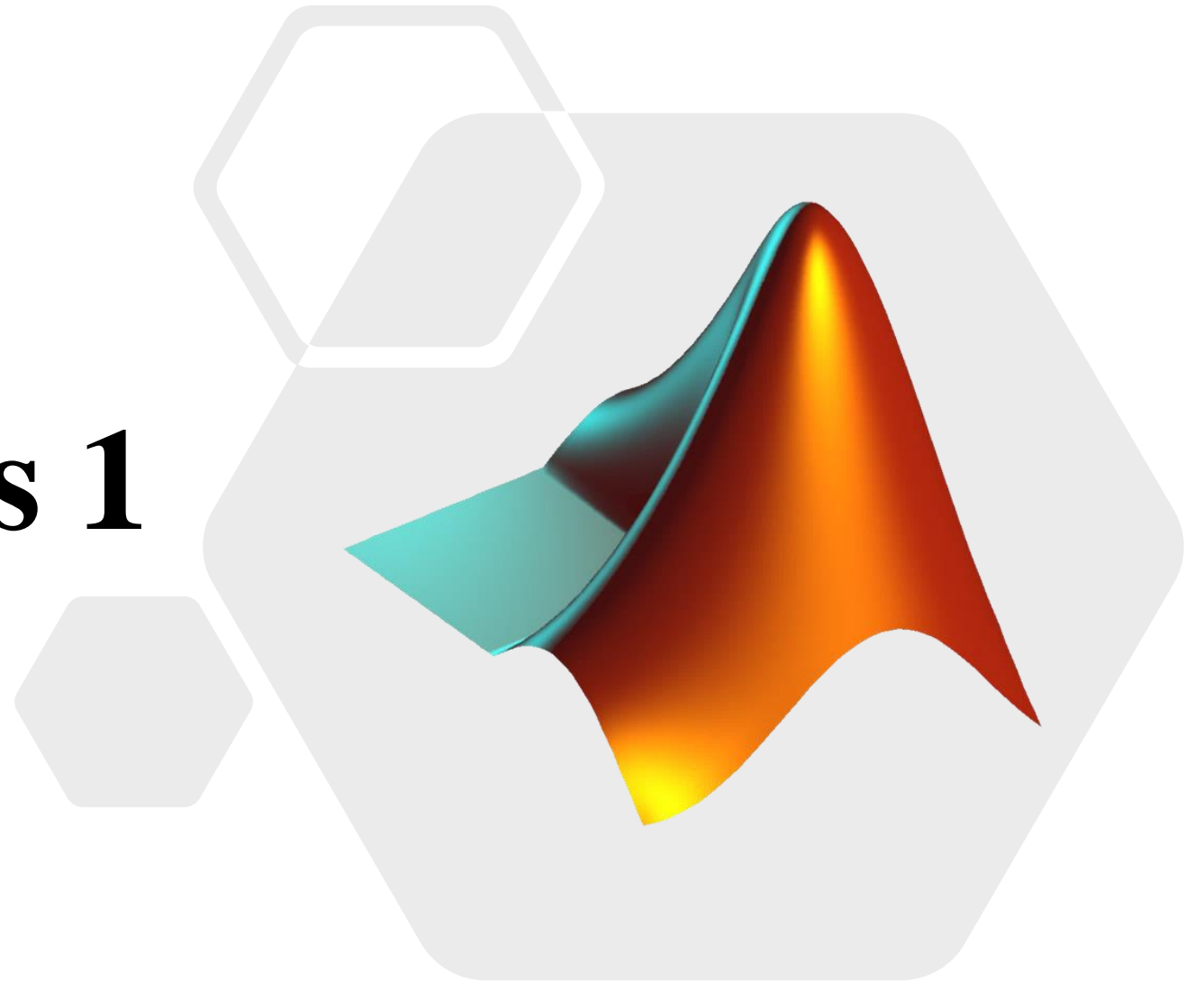


# Data Analysis 1

*Week 5 (16:00, 10/11/2021)*

**Vivatsathorn Thitasirivit**

*Student. Faculty of Engineering, Chulalongkorn University.*



# Week 5 Outlines

## **MATLAB**

- Element-wise Operations
- .M-File Scripts
- .M-File Functions
- Array-like File import

## **Assignments**

*Week 5 Quiz*

# MATLAB Competitors:

## *Mathematica*

One of the biggest MATLAB competitor is *Mathematica*, the symbolic computation program.

MATLAB is more convenient for *numerical analysis* and *linear algebra* which is widely used in *engineering* community.

Mathematica has superior symbolic manipulation, making it popular among *physicists* and *mathematician*.

One of their product is *Wolfram Alpha*. Online platform for solving symbolic mathematic expressions.



# MATLAB Matrices: Matrix Generation

A square matrix of dimension  $n \times n$  with magic square (total sum of each row and column are equal) in MATLAB can be generated using command:

```
>> magic(5)
```

```
ans =
```

```
17    24     1     8    15
23     5     7    14    16
 4     6    13    20    22
10    12    19    21     3
11    18    25     2     9
```

# MATLAB Matrices: Element-wise Operation

Normally, matrices operations are based on whole matrix algorithms approaches.

The element-wise operation is an ordinary mathematical operation, utilizing a matrix as **an array** (a table).

Matrix Multiplication

```
>> A*B
```

```
ans =
```

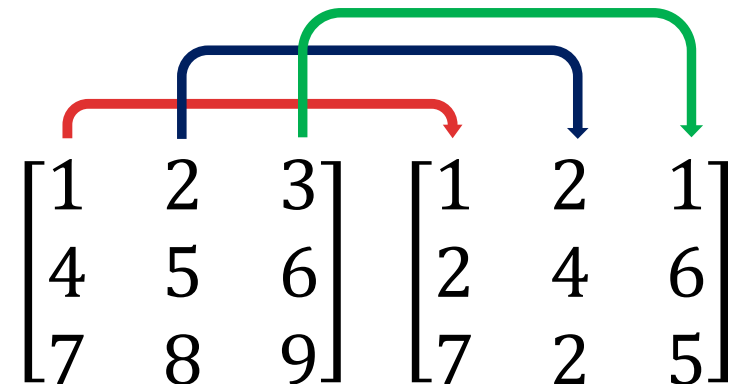
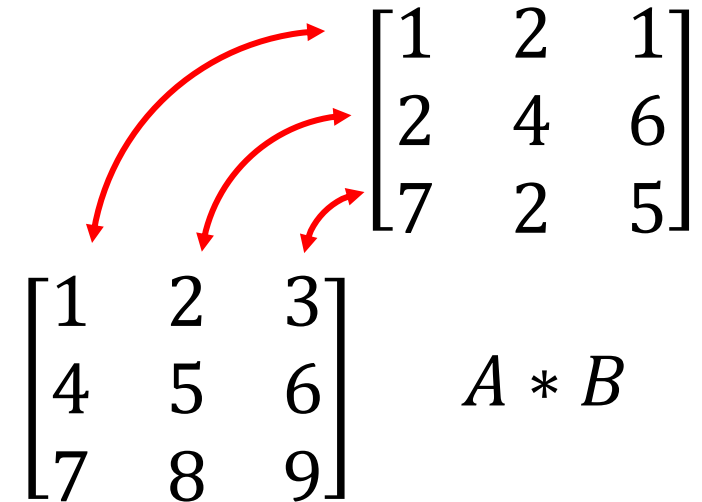
```
26    16    28
56    40    64
86    64   100
```

Element Multiplication

```
>> A.*B
```

```
ans =
```

```
1     4     3
8    20    36
49    16    45
```



# MATLAB Matrices: Element-wise (Unary) Operation

Matrix	Operation	Element (Array)
+	Addition	+
-	Subtraction	-
*	Multiplication	.*
/	<i>Right Division</i>	./
\	<i>Left Division</i>	.\
^	Exponentiation to constant	.^

Solution of  $\mathbf{xA} = \mathbf{B}$  ( $\mathbf{x} = \mathbf{B}/\mathbf{A}$ )

Solution of  $\mathbf{Ax} = \mathbf{B}$  ( $\mathbf{x} = \mathbf{A}\backslash\mathbf{B}$ )

# MATLAB Matrices: Solving Linear Equations

For solving scientific computation of simultaneous linear equations, a matrix notation is usually written instead of whole equations:

$$\mathbf{Ax} = \mathbf{b}$$

whereas  $\mathbf{A}$  is a coefficient matrix,  $\mathbf{x}$  is a variable vector, and  $\mathbf{b}$  is a constant vector.

Generally, given simultaneous linear equation:

$$\begin{cases} A_1x_1 + A_2x_2 + A_3x_3 + \dots = A \\ B_1x_1 + B_2x_2 + B_3x_3 + \dots = B \\ C_1x_1 + C_2x_2 + C_3x_3 + \dots = C \\ \dots \\ \Omega_1x_1 + \Omega_2x_2 + \Omega_3x_3 + \dots = \Omega \end{cases}$$

We get:

$$\mathbf{A} = \begin{bmatrix} A_1 & \dots & A_N \\ \vdots & \ddots & \vdots \\ \Omega_1 & \dots & \Omega_N \end{bmatrix},$$

$$\mathbf{b} = \begin{bmatrix} A \\ \vdots \\ \Omega \end{bmatrix},$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix},$$

such that

$$\begin{bmatrix} A_1 & \dots & A_N \\ \vdots & \ddots & \vdots \\ \Omega_1 & \dots & \Omega_N \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} A \\ \vdots \\ \Omega \end{bmatrix}$$

The solution will be

$$(\mathbf{A}^{-1}\mathbf{A})\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$$

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$$

# MATLAB Matrices: Solving Linear Equations

Example:

$$\begin{cases} x + 2y + 3z = 1 \\ 4x + 5y + 6z = 1 \\ 7x + 8y + 9z = 1 \end{cases}$$

We can write this Sim. Lin. Eqn. in a matrix form:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

*There are infinitely many solutions to this eqn., so the solution may differ from PC to PC, or algorithm to algorithm.*

In MATLAB,

```
>> A = [1,2,3;4,5,6;7,8,9];
>> b = [1;1;1];
>> x = inv(A)*b
Warning: Matrix is close to
singular or
badly scaled. Results may be
inaccurate.
RCOND = 2.202823e-18.
```

```
x =
    0
   -1
    1
```

Alternatively,

```
>> x = A\b
```



# MATLAB Scripting: M-Files Script

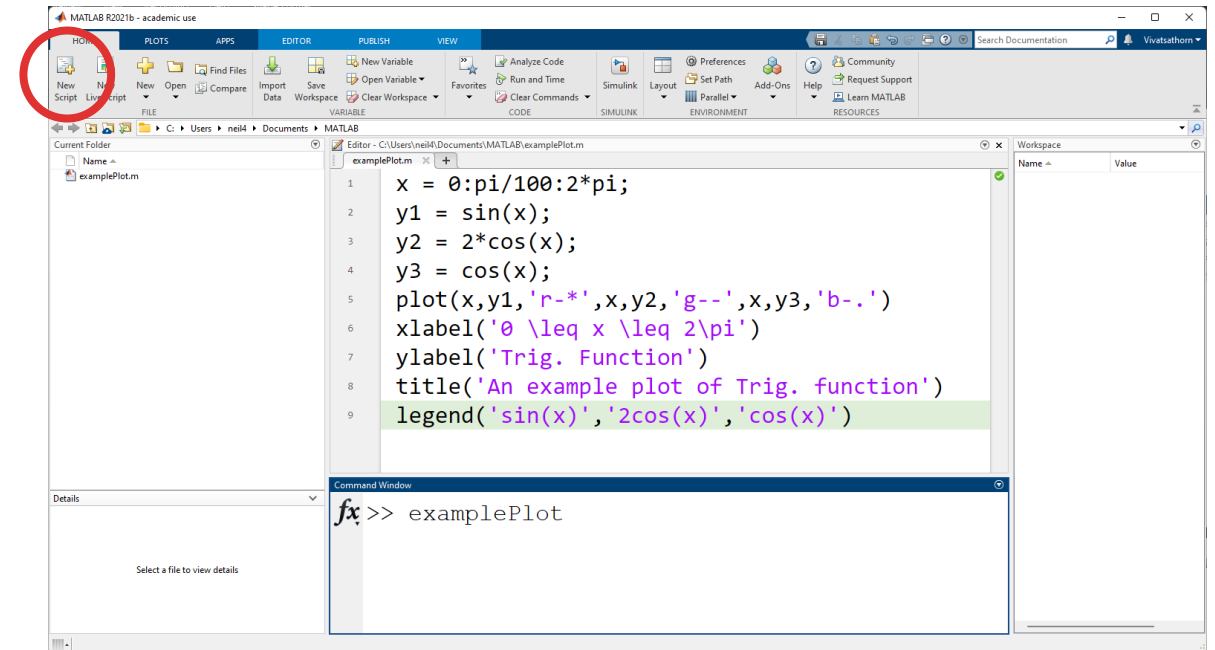
A *script file* is an external file containing a sequence of MATLAB statements. Such file must contain a file name and extension “.m”. It can execute a series of statements in a go which user does not have to type the statements one by one. A *function file* is a script file which can accept arguments and produce one or more outputs.

## Instructions

1. In MATLAB, **HOME** => **New Script**
2. Enter a set of following commands:

```
x = 0:pi/100:2*pi;  
y1 = sin(x);  
y2 = 2*cos(x);  
y3 = cos(x);  
plot(x,y1,'r-*',x,y2,'g--',x,y3,'b-.')  
xlabel('0 \leq x \leq 2\pi')  
ylabel('Trig. Function')  
title('An example plot of Trig. function')  
legend('sin(x)', '2cos(x)', 'cos(x)')
```

3. Save the file, namely, **examplePlot.m** in default directory
4. Run the file by typing **examplePlot** in the Command Window



*For MATLAB Mobile, go to Files => New Script*

# MATLAB Scripting: M-Files Script

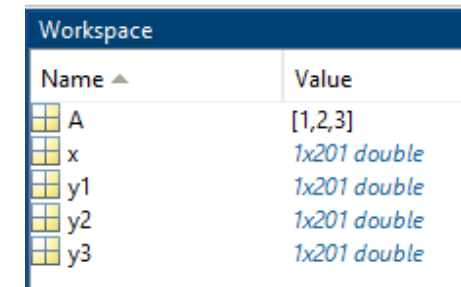
## Side effects of Scripts

After the script is run, all variables in the script will be added to the Workspace. This might cause undesirable/unwanted effects to global variables because:

1. Existing variables in the workspace will be overwritten
2. The script might be affected by existing variables in the workspace if not redeclared in the script properly. The figures might also be affected.

As *M-File Script* has quite undesirable effects, *M-File Function* is recommended to code complicated applications as the variables created in the *file* is local.

Because the script is just an automation of typing in the Command Window.



Name ▲	Value
A	[1,2,3]
x	1x201 double
y1	1x201 double
y2	1x201 double
y3	1x201 double

# MATLAB Scripting: M-Files Script

## Example 1. User prompt

*File name: examplePrompt.m*

### Editor Window

```
x1 = input('Input the first number ');  
x2 = input('Input the second number ');  
x3 = input('Input the third number ');  
average = (x1+x2+x3)/3
```

### Command Window

```
>> examplePrompt
```

# MATLAB Programming: M-Files Function

## Example 2. Anatomy of a MATLAB function

File name: `exampleFactorial.m`      Save in default dir.

### Editor Window

```
function f = exampleFactorial(n)
% FACTORIAL(N) receives N as an argument
% and returns the factorial of N
% Compute a factorial value.
f = prod(1:n);
end
```

### Command Window

```
>> exampleFactorial
```

Function declaration statement:

“function” +

“output variable” +

“=” +

“function name ” (*must be the same as file name*) +

“argument(s)”

General form:

```
function output = name(args,)
    statements
    output = calculation
end
```

Commenting in a script/function uses “%”

“f” is the final output variable. The final stage of the calculation must return at least one value. More value can be returned in array format.

# MATLAB Programming: M-Files Function

Example 3. Receiving and returning multiple values

File name: *exCalc.m*

Editor Window

```
function [a,b,c,d,e] = exCalc(x,y)
a = x+y;
b = x-y;
c = x*y;
d = x/y;
e = x^y;
end
```

Command Window

```
>> exCalc
```

# MATLAB Programming: M-Files Function

**Quiz 00. (★☆☆☆☆) Warm Up Routine**

**File name: *q00.m***

Write a function that sums up the number ranging from  $a$  to  $b$  (as inputs) and return the result.

# MATLAB Programming: M-Files Function

## Quiz 01. (★☆☆☆☆) Temperature Conversion

File name: *q01.m*

Write a function file that converts temperature in degrees Fahrenheit to degrees Celsius. The function should take  $F$  as the only input and return Celsius temperature as the only output.

*Recall:*

$$C = \frac{5}{9}(F - 32)$$

# MATLAB Programming: M-Files Function

## Quiz 02. (★★☆☆☆) Combination

File name: q02.m

From the binomial theory and the probability theory,  ${}^n C_r$  formula is widely used. Combination is a selection of items from a set that has distinct members. The formula is usually written in factorial form:

$${}^n C_r = \binom{n}{r} = \frac{n!}{r! (n - r)!}$$

Write a function file that takes  $n$  and  $r$  as inputs **respectively**, then return the value of Combination value as only output.

You must optimize the algorithm as much as possible for the computer to use less processing steps and time.



# MATLAB Programming: M-Files Function

## Quiz 03. (★★★★☆) Matrices Operations

File name: *q03.m*

Create a function that prompt user to input two matrices with the same dimension. The function then returns the following value:

- I. Sum of two matrices
- II. Product of two matrices
- III. Element-wise product of two arrays
- IV. Sum of determinant of two matrices

Use the multiple value return technique.

# MATLAB Programming: M-Files Function

## Quiz 04. (★★★★☆) Fake Benford

File name: q04.m

A logarithmic decaying function is a function used to describe Benford's Law first digit diminishing pattern:

$$f(x) = A \ln \left( 1 + \frac{1}{x} \right)$$

whereas  $A$  is an amplitude of decaying.

Write a function that takes  $A$  as the only input and create a figure that shows  $f(x) - x$  relation in domain  $[0,10]$  with appropriate increments/resolution.

# MATLAB Programming: M-Files Function

## Quiz 05. (★★★★☆☆) Hyperbola

File name: q05.m

Write a function that plots the following:

- I. A hyperbola in solid line, red color
- II. Both asymptotes in dashed line, black color

The function should take  $a$  and  $b$  as inputs for following equation plot of domain  $[-10,10]$  with appropriate resolution:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

Solving the equation, we get the asymptotes of

$$y = \pm \frac{b}{a}x$$

# MATLAB Programming: M-Files Function

## Quiz 06. (★★★★☆☆) Integral of Decay

File name: *q06.m*

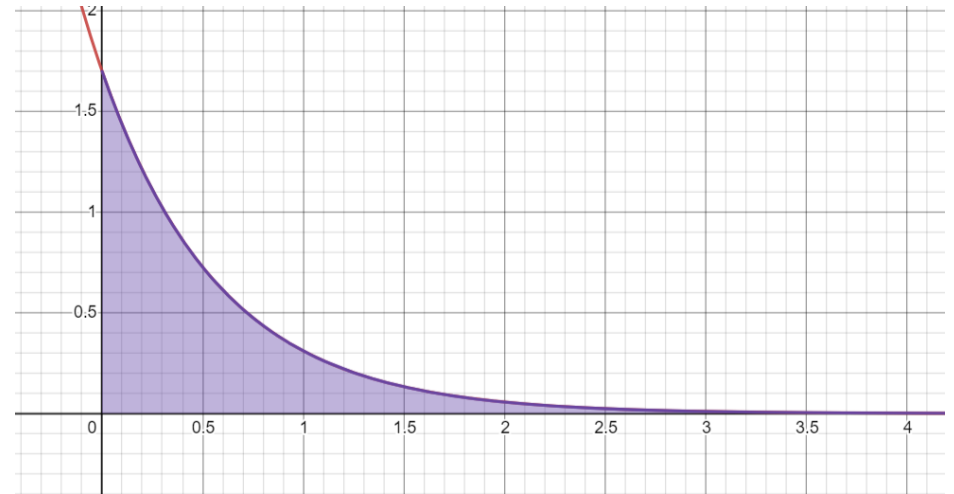
Consider an exponential decay probability density function:

$$f(x) = \lambda e^{-\lambda x}$$

Prove numerically that the total of area under curve  $f(x)$  from 0 to infinity approaches 1.

$$\text{Area} = \int_0^{\infty} f(x) dx$$

Take  $\lambda$  (the rate parameter) as the only input



For how to use integral in MATLAB:

[www.mathworks.com/help/matlab/ref/integral.html](http://www.mathworks.com/help/matlab/ref/integral.html)

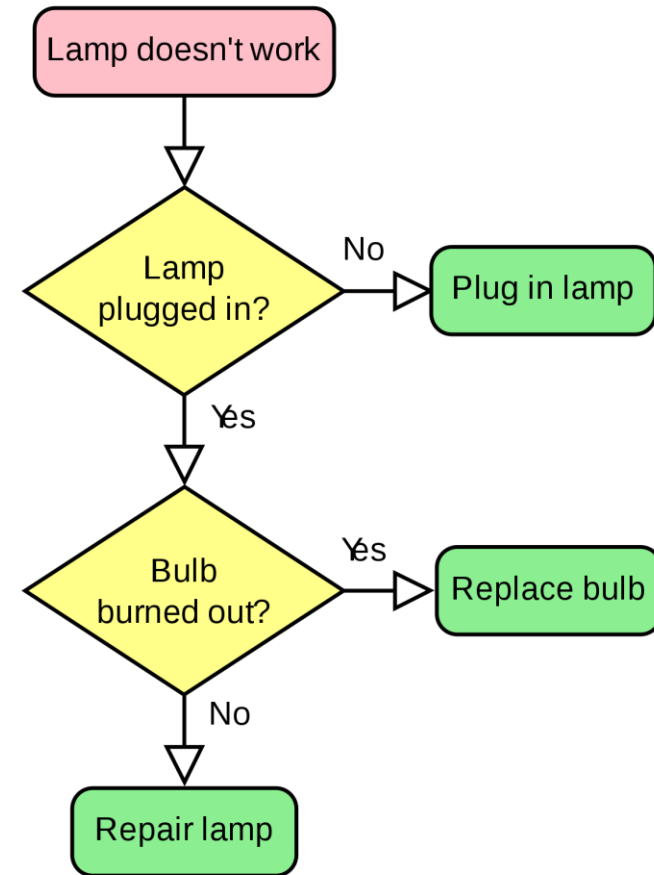
# MATLAB Programming: Control flow and operators

## Introduction

MATLAB is not only a *scripting* language but is also *programming* language. Like other com. prog. languages, MATLAB has decision-making statements/structures for command execution. A.K.A. *control flow* structures which include:

- I. *for* loop
- II. *while* loop
- III. *if-elseif-else-end* construction

So, it is possible to skip commands or to execute specific commands for specific actions.



Wikipedia—A Simple Flowchart

# MATLAB Programming:

## Control flow and operators

### The “*if ... end*” conditional statement

MATLAB supports various combination of *if*, *elseif*, *else*, and *end* constructions.

The simplest form of conditional statement is

```
if expression
    statements
end
```

The *else* is used for conditions other than stated. The *elseif* is used by specifying a condition after passing through *if* checking first. So, the combinations might be:

```
if expression
    statement_1
else
    statement_2
end
```

```
if expression_1
    statement_1
elseif expression_2
    statement_2
end
```

```
if expression_1
    statement_1
elseif expression_2
    statement_2
else
    statement_3
end
```

and more combinations of them.

# MATLAB Programming:

## Control flow and operators

### Side note

- I. The *elseif* statement has no space between *else* and *if* (one word).
- II. No semicolon (;) is needed at the end of conditional statement.
- III. Indentation in MATLAB is not required but facilitate the reading.
- IV. The *end* statement is required.

### Relational/Comparison/Logical Operator

Operator	Description
>	Strictly greater than
<	Strictly less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal to
~=	Not equal to
&	AND operator
	OR operator
~	NOT operator

# MATLAB Programming:

## Control flow and operators

### Example 4. Discriminant of quadratic formula

File name: *exDisc.m*

#### Editor Window

```
discr = b*b - 4*a*c
if discr < 0
    disp('Warning! Discriminant is NEGATIVE! No real root.')
elseif discr == 0
    disp('The quadratic equation has ONE real root.')
else
    disp('The quadratic equation has TWO real roots.')
```

#### Instruction

Apply the statement in either *script* or *function* to check whether the quadratic equation has real roots or not. If so, how many roots are there? and if possible, calculate and show only the real root of the equation.



# MATLAB Programming: Control flow and operators

## The “*for ... end*” loop statement

MATLAB *for ... end* loop (iteration) statement syntax can be written in:

```
for index = values
    statements
end
```

For *values*, we use *Colon Operator* for the iteration of index. For example,

```
for i = 1:10
    x = i*i
end
```

```
for v = 1.0:-0.2:0.0
    disp(v)
end
```

## Recall:

Colon Operator is usually written in a form of:

$a:s:b$

or

$a:b$

meaning the index will iterate from  $a$  to  $b$  with a step of 1 or with a step of  $s$ .

# MATLAB Programming: Control flow and operators

The “*for ... end*” loop statement

The *for* loop can be nested.

## Independent nested loop

```
n = 5; A(1:n,1:n) = 0;  
for i = 1:n  
    for j = 1:n  
        A(i,j) = 2  
    end  
end
```

*Generate a square matrix  
with all elements equal 2*

## Dependent nested loop

```
n = 5; A(1:n,1:n) = 0;  
for i = 1:n  
    for j = 1:i  
        A(i,j) = 2  
    end  
end
```

*Generate a triangle matrix  
with triangle elements equal  
2, and others equal 0*

# MATLAB Programming: Control flow and operators

## The “*while ... end*” loop statement

MATLAB *while ... end* loop (iteration) statement is a loop with condition(s).

```
while expression
    statements
end
```

For *expression*, you can use the same term as in *if* structure.

**Note** that if the condition is not well defined, the loop will continue *indefinitely* (an infinite loop). If this ever happens, you can stop the execution by pressing **CTRL + C**.

## A well-defined loop

```
x = 1
while x <= 10
    x = 3*x
end
```

## A poorly defined loop

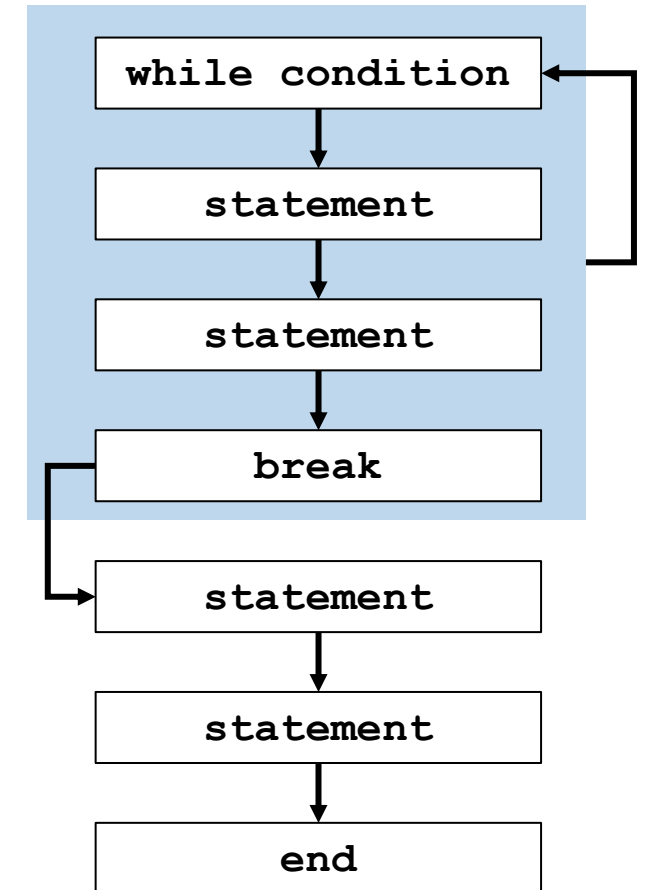
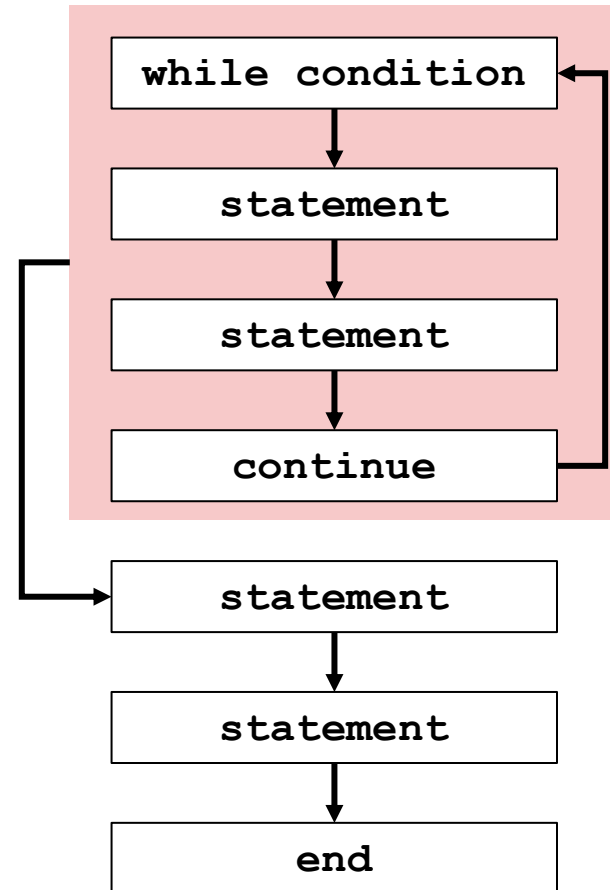
```
x = 1
while x ~= 10
    x = 3*x
end
```

# MATLAB Programming: Control flow and operators

## Other control flow structure

In MATLAB, like other programming languages, there are several flow structure statements which can be used:

- I. The *break* statement. A *while* and *for* loop can be terminated with this statement. Place a *break* before the end statement. If a *break* is reached, the loop will immediately stop and be passed.
- II. The *continue* statement in a *while* and *for* loop is like the *break*, but it does not terminate the whole loop, it terminate only an iteration of loop (skipping the remaining statements in the loop).
- III. There are *return*, *switch*, etc. For more detail, please read MATLAB documentation.



# MATLAB Programming: M-Files Function

## Quiz 07. (★☆☆☆☆) Upside down triangular matrix

File name: *q07.m*

It normally can be generated using *tril* and *triu* functions, but for loop learning purposes, do not use such command. For real-world application, you can use any tools you want.

Create a function that takes  $n$  and  $k$  as inputs that create an upper triangular matrix size of  $n$  and  $k$  value.

Example: 4x4 value 5 upper triangular matrix:

$$\begin{bmatrix} 5 & 5 & 5 & 5 \\ 0 & 5 & 5 & 5 \\ 0 & 0 & 5 & 5 \\ 0 & 0 & 0 & 5 \end{bmatrix}$$

# MATLAB Programming: M-Files Function

## Quiz 08. (★☆☆☆☆) Diagonal matrix

File name: *q08.m*

It normally can be generated using *eye* or *diag* command.

Create a function that takes  $n$  and  $k$  as inputs that create a diagonal matrix size of  $n$  and  $k$  value.

Example: 4x4 value 5 diagonal matrix:

$$\begin{bmatrix} 5 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 5 \end{bmatrix}$$

# MATLAB Programming: M-Files Function

**Quiz 09.** (★★★★☆) A hollow square matrix

*File name:* *q09.m*

Create a function that takes  $n$  as an input for generating a hollow square matrix with 1 outside and 0 inside.

Example: 4x4 hollow square matrix:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

# MATLAB Programming: M-Files Function

## Quiz 10. (★★★★☆) Diagonal matrix

File name: *q10.m*

It normally can be generated using *eye* or *diag* command.

Create a function that takes  $v$  as inputs that create a diagonal matrix consisting of elements in  $v$  row vector.

Example:

$$v = [1 \quad 2 \quad -0.5 \quad 7]$$

4x4 diagonal matrix:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & -0.5 & 0 \\ 0 & 0 & 0 & 7 \end{bmatrix}$$



# MATLAB Programming: Control flow and operators

## Operator Precedence (Hierarchy)

We can combine *arithmetic*, *relational*, and *logical operators*. The precedence rules determine the order of execution. In week 4, we already talk about arithmetic precedence.

Precedence	Operator
1 (Highest)	Parentheses ()
2	Transpose (.'), power (.^), matrix power (^)
3	Unary plus (+), unary minus (-), negate (~)
4	Multiplication (.*), right division (./), left division (.\), matrix multiplication (*), matrix right division (/), matrix left division(\)
5	Addition (+), subtraction (-)
6	Colon operator (:)
7	Strictly less than (<), less than or equal to (≤), strictly greater than (>), greater than or equal to (≥), equal to (==), not equal to (~=)
8	Element-wise AND (&)
9 (Lowest)	Element-wise OR ( )

# MATLAB Programming: File

## Writing output to a file

In addition to displaying the output on the Command Window, *fprintf* command can be used for writing output to a file. The saved data can be used later in MATLAB or external software.

## Instruction

```
f = fopen(filename, permission);  
fprintf(f, statement);  
fclose(f);
```

## Example

```
op = fopen('weekdays.txt', 'wt');  
fprintf(op, 'Sunday\nMonday\nTuesday\nWednesday\n');  
fprintf(op, 'Thursday\nFriday\nSaturday\n');  
fclose(op)
```

# MATLAB Programming: File

## Reading a matrix/array from a file

In MATLAB, you can open a file to read, write, or append text/number to it. The format of an array in a text form can vary. One of the most used format is Comma-delimited value (CSV) format. For example,

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

can be written in text format as:

1,2,3  
4,5,6  
7,8,9

There are other formats as well, e.g., tab-delimited, character-delimited, etc.

## Reading a matrix from delimited file

```
M = readmatrix('filename.csv');
```

## Writing a matrix to a CSV-formatted file

```
writematrix(M, 'filename.csv');
```

MATLAB also supports reading/writing a matrix from/to Excel spreadsheet file (.xls, .xlsx, .xslm)

A CSV format will be autogenerated. If you wish to use other formats, consider the following command:

```
writematrix(M, 'filename.csv', 'Delimiter', 'tab');
```