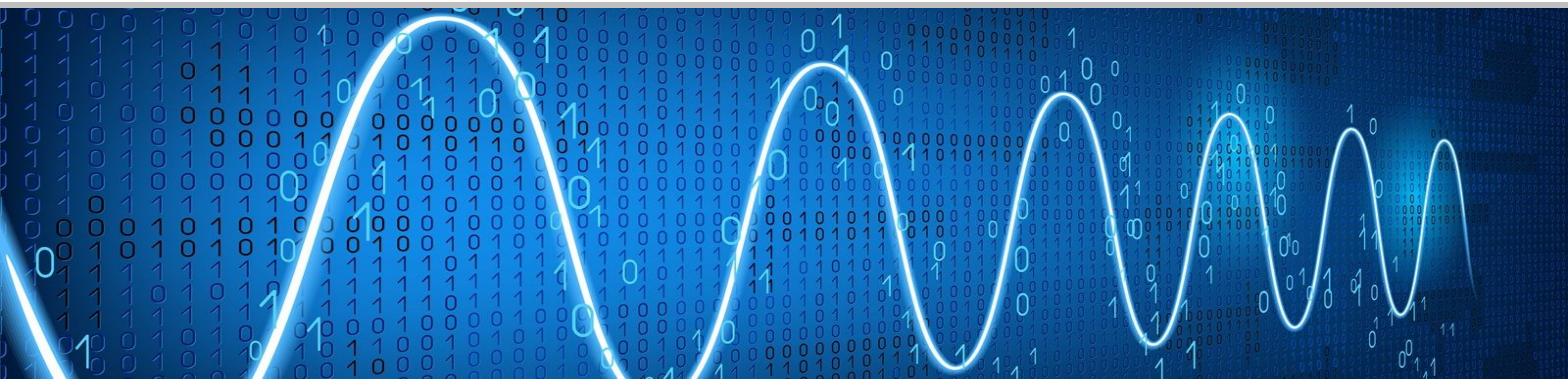
Digital Signal Processing

Lab 01: MATLAB Basics

Abdallah El Ghamry



MATLAB

The purpose of this Lab is to learn the basics of MATLAB including:

- MATLAB Environment
- Variables and Arrays
- Creating Vectors and Matrices
- Accessing, Adding Modifying, Deleting Array/Matrix Elements
- Predefined Special Values
- Common Array and Matrix Operations
- Common MATLAB Functions
- Character Arrays and Strings
- Complex Numbers
- Input-Output Functions

MATLAB

- MATLAB is an abbreviation for "matrix laboratory".
- While other programming languages mostly work with numbers, MATLAB is designed to operate primarily on matrices and arrays.
- The fundamental unit of data in MATLAB program is the array.
- An array is a collection of data values organized into rows and columns and known by a single name.
- Even scalars are treated as arrays by MATLAB, they are arrays with only one row and one column.

Typical Uses

- **Digital Signal Processing**
- Digital Image Processing
- Math and Computation
- Data Analysis, Exploration, and Visualization.
- Modeling and Simulation
- Scientific and Engineering Graphics
- Application Development Including GUIs.
- Algorithm Development
- Etc…

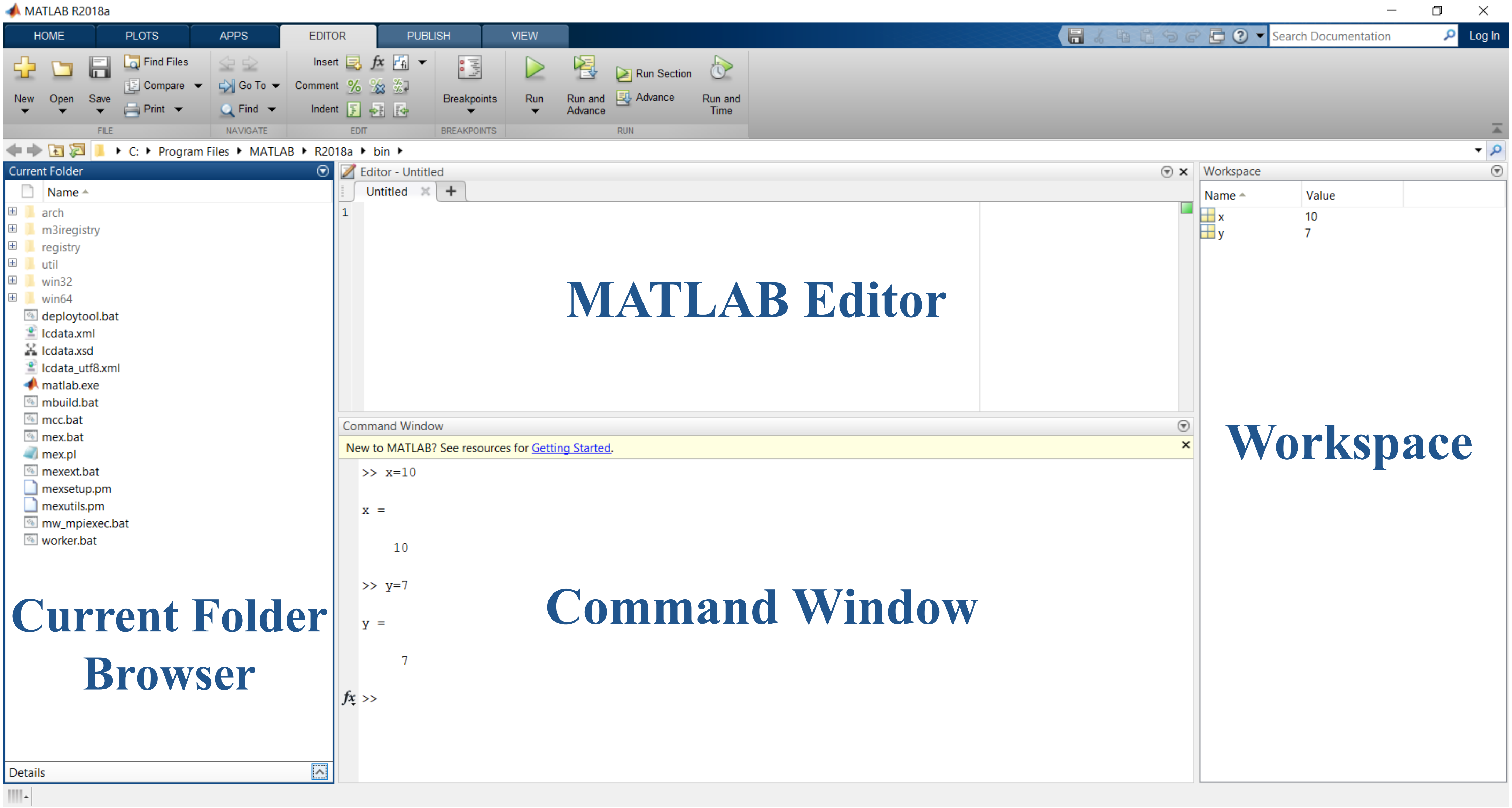
Why MATLAB?

- Ease of Use
- Platform Independence

  Windows XP/Vista/7, Linux, Unix, and the Macintosh.
- Predefined Functions

  MATLAB comes complete with an extensive library of predefined functions that provide solutions to many basic technical tasks.
- Graphical User Interface (GUI)
- EXTENSIVE Documentation.

MATLAB Desktop



MATLAB Desktop

- **Command Window**

  A window where the user can type commands and see results.

- **Workspace Browser**

  A window that displays the names and values of variables stored in the MATLAB Workspace.

- **Current Folder Browser**

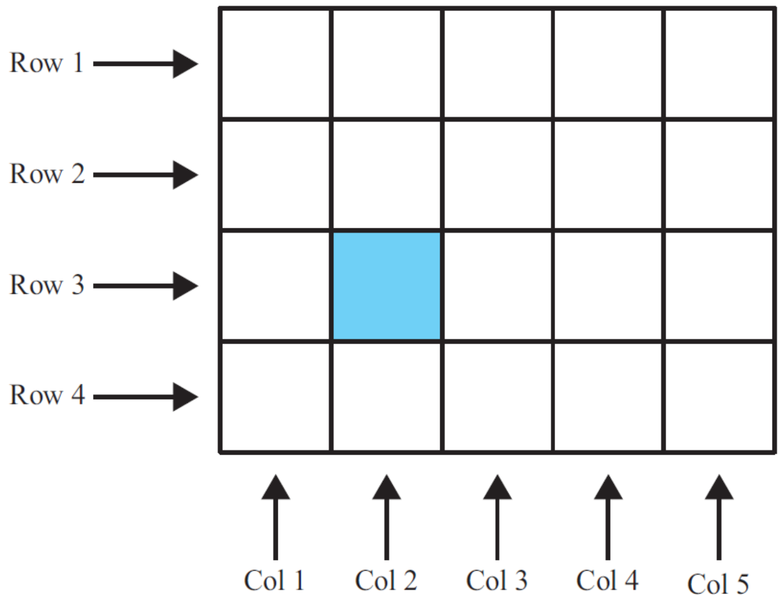
  A window that displays the names of files in the current directory.

- **MATLAB Editor**

  Where scripts are created and edited.

Variables and Arrays

- Arrays can be classified as either vectors or matrices.
- The term "vector" is usually used to describe an array with only one dimension.
- The term "matrix" is usually used to describe an array with two or more dimensions.



Variables and Arrays

- A MATLAB variable is a region of memory containing an array and is known by a user-specified name.
- MATLAB variable names must begin with a letter, followed by any combination of letters, numbers, and underscore _.
- The MATLAB language is case-sensitive, which means that uppercase and lowercase letters are not the same.
- When naming a variable, make sure you are not using a name that is already used as a function name.

Creating and Initializing Variables

- Variables are automatically created when they are initialized.

```
var = expression;
```

```
>> a = 1
a =
     1
>> b = 2
b =
     2
>> c = a + b
c =
     3
```

Workspace

Name	Value
a	1
b	2
c	3

Creating and Initializing Variables

```
>> x = 20;
>> y = 5;
>> sig = x + y
sig =
    25
>> diff = x  - y
diff =
    15
>> prod = x * y
prod =
   100
>> div = x / y
div =
     4
```

Workspace

Name	Value
diff	15
div	4
prod	100
sig	25
x	20
y	5

Arithmetic Operations between Two Scalars

Table 2.5: Arithmetic Operations between Two Scalars

Operation	Algebraic Form	MATLAB Form
Addition	$a + b$	`a + b`
Subtraction	$a - b$	`a - b`
Multiplication	$a \times b$	`a * b`
Division	$\frac{a}{b}$	`a / b`
Exponentiation	a^b	`a ^ b`

Hierarchy of Arithmetic Operations

Table 2.7: Hierarchy of Arithmetic Operations

Precedence	Operation
1	The contents of all parentheses are evaluated, starting from the innermost parentheses and working outward.
2	All exponentials are evaluated, working from left to right.
3	All multiplications and divisions are evaluated, working from left to right.
4	All additions and subtractions are evaluated, working from left to right.

$$
\begin{aligned}
2 \wedge ((8+2)/5) &= 2 \wedge (10/5) \\
&= 2 \wedge 2 \\
&= 4
\end{aligned}
$$

```
>> 2 ^ ((8 + 2)/5)
ans =
     4
```

Creating Row Vectors

```
>> r = [7 8 9 10 11]
r =
     7     8     9    10    11
>> r = [7, 8, 9, 10, 11]
r =
     7     8     9    10    11
>> r = [7, 8 9 10, 11]
r =
     7     8     9    10    11
```

Creating Column Vectors

```
>> c = [7;  8;  9;  10; 11]

c =

     7
     8
     9
    10
    11
```

Creating Matrices

```
>> m = [1 2 3; 4 5 6; 7 8 9]

m =

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

Accessing Array Elements

```
>> x = [11 55 88 77 63 45]
x =
    11    55    88    77    63    45
>> x(2)
ans =
    55
>> x(2:end)
ans =
    55    88    77    63    45
>> x(3: end-1)
ans =
    88    77    63
```

Adding and Modifying Array Elements

```
>> x
x =
     11     55     88     77     63     45
>> x(end + 1) = 99
x =
     11     55     88     77     63     45     99
>> x(2) = 22
x =
     11     22     88     77     63     45     99
>> x(end + 1: end + 3) = 7
x =
     11     22     88     77     63     45     99      7      7      7
```

Deleting Array Elements

```
>> x

x =

      11      22      88      77      63      45      99       7       7    7

>> x(2:4) = []

x =

    11      63      45      99       7       7       7

>> x(end) = []

x =

    11      63      45      99       7       7

>> x = []

x =

     []
```

Accessing Matrix Elements

```
>> a = [1 2 3; 4 5 6; 7 8 9; 10 11 12]
a =
     1     2     3
     4     5     6
     7     8     9
    10    11    12

>> r3 = a(3, :)
r3 =
     7     8     9
```

Accessing Matrix Elements

```
>> a
a =
     1     2     3
     4     5     6
     7     8     9
    10    11    12
>> c2 = a(:, 2)
c2 =
     2
     5
     8
    11
```

Accessing Matrix Elements

```
>> a(:, 2) = -1
a =
     1    -1     3
     4    -1     6
     7    -1     9
    10    -1    12

>> a(4, :) = []
a =
     1    -1     3
     4    -1     6
     7    -1     9
```

Accessing Matrix Elements

```
>> arr4 = [1 2 3 4; 5 6 7 8; 9 10 11 12]
arr4 =

     1     2     3     4
     5     6     7     8
     9    10    11    12

>> arr4(2:end,2:end)
ans =

     6     7     8
    10    11    12
```

Colon Operator

- MATLAB provides a special shortcut notation using the colon operator.
- The colon operator specifies a whole series of values by specifying the first value in the series, the stepping increment, and the last value in the series.
- The general form of a colon operator is

```
first:incr:last
```

Colon Operator: Examples

```
>> x = 1:2:10
x =
     1     3     5     7     9

>> y = 1:10
y =
     1     2     3     4     5     6     7     8     9    10

>> z = 10:-2:0
z =
    10     8     6     4     2     0
```

Creating Variables: Examples

`[3.4]`	This expression creates a 1×1 array (a scalar) containing the value 3.4. The brackets are not required in this case.
`[1.0 2.0 3.0]`	This expression creates a 1×3 array containing the row vector $[1 \quad 2 \quad 3]$.
`[1.0; 2.0; 3.0]`	This expression creates a 3×1 array containing the column vector $\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$.
`[1, 2, 3; 4, 5, 6]`	This expression creates a 2×3 array containing the matrix $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$.
`[1, 2, 3` ` 4, 5, 6]`	This expression creates a 2×3 array containing the matrix $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$. The end of the first line terminates the first row.
`[]`	This expression creates an **empty array**, which contains no rows and no columns. (Note that this is not the same as an array containing zeros.)

Predefined Special Values

Table 2.2: Predefined Special Values

Function	Purpose
`pi`	Contains π to 15 significant digits.
`i, j`	Contain the value i ($\sqrt{-1}$).
`Inf`	This symbol represents machine infinity. It is usually generated as a result of a division by 0.
`NaN`	This symbol stands for Not-a-Number. It is the result of an undefined mathematical operation, such as the division of zero by zero.
`clock`	This special variable contains the current date and time in the form of a 6-element row vector containing the year, month, day, hour, minute, and second.
`date`	Contains the current data in a character string format, such as `24-Nov-1998`.
`eps`	This variable name is short for "epsilon." It is the smallest difference between two numbers that can be represented on the computer.
`ans`	A special variable used to store the result of an expression if that result is not explicitly assigned to some other variable.

Predefined Special Values: Examples

```
>> pi
ans =
     3.1416

>> i
ans =
   0.0000 + 1.0000i

>> nan
ans =
   NaN
```

Initializing with Built-in Functions

Table 2.1: MATLAB Functions Useful for Initializing Variables

Function	Purpose
`zeros(n)`	Generates an n × n matrix of zeros.
`zeros(m,n)`	Generates an m × n matrix of zeros.
`zeros(size(arr))`	Generates a matrix of zeros of the same size as `arr`.
`ones(n)`	Generates an n × n matrix of ones.
`ones(m,n)`	Generates an m × n matrix of ones.
`ones(size(arr))`	Generates a matrix of ones of the same size as `arr`.
`eye(n)`	Generates an n × n identity matrix.
`eye(m,n)`	Generates an m × n identity matrix.
`length(arr)`	Returns the length of a vector, or the longest dimension of a two-dimensional array.
`numel(arr)`	Returns the total number of elements in an array, which is the product of the number of rows times the number of columns.
`size(arr)`	Returns two values specifying the number of rows and columns in `arr`.

Initializing with Built-in Functions

```
a = zeros(2);
b = zeros(2,3);
c = [1 2; 3 4];
d = zeros(size(c));
```

These statements generate the following arrays:

$$a = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad b = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$c = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \qquad d = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Initializing with Built-in Functions: Examples

```
>> zeros(3, 3)
ans =
     0     0     0
     0     0     0
     0     0     0

>> ones(3, 4)
ans =
     1     1     1     1
     1     1     1     1
     1     1     1     1
```

Initializing with Built-in Functions: Examples

```
>> eye(3, 4)
ans =
     1     0     0     0
     0     1     0     0
     0     0     1     0

>> size(ans)
ans =
     3     4
```

Common Array and Matrix Operations

Table 2.6: Common Array and Matrix Operations

Operation	MATLAB Form	Comments
Array Addition	`a + b`	Array addition and matrix addition are identical.
Array Subtraction	`a - b`	Array subtraction and matrix subtraction are identical.
Array Multiplication	`a .* b`	Element-by-element multiplication of `a` and `b`. Both arrays must be the same shape, or one of them must be a scalar.
Matrix Multiplication	`a * b`	Matrix multiplication of `a` and `b`. The number of columns in `a` must equal the number of rows in `b`.
Array Right Division	`a ./ b`	Element-by-element division of `a` and `b`: `a(i,j) / b(i,j)`. Both arrays must be the same shape, or one of them must be a scalar.
Array Left Division	`a .\ b`	Element-by-element division of `a` and `b`, but with `b` in the numerator: `b(i,j) / a(i,j)`. Both arrays must be the same shape, or one of them must be a scalar.
Matrix Right Division	`a / b`	Matrix division defined by `a * inv(b)`, where `inv(b)` is the inverse of matrix `b`.
Matrix Left Division	`a \ b`	Matrix division defined by `inv(a) * b`, where `inv(a)` is the inverse of matrix `a`.
Array Exponentiation	`a .^ b`	Element-by-element exponentiation of `a` and `b`: `a(i,j) ^ b(i,j)`. Both arrays must be the same shape, or one of them must be a scalar.

Common Array and Matrix Operations

$$a = \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix} \qquad b = \begin{bmatrix} -1 & 2 \\ 0 & 1 \end{bmatrix}$$

$$c = \begin{bmatrix} 3 \\ 2 \end{bmatrix} \qquad d = 5$$

What is the result of each of the following expressions?

(a) `a + b`

(b) `a .* b`

(c) `a * b`

(d) `a * c`

(e) `a + c`

(f) `a + d`

(g) `a .* d`

(h) `a * d`

Common Array and Matrix Operations

```
>> a = [1 0; 2 1]
a =
     1     0
     2     1
>> b = [-1 2; 0 1]
b =
    -1     2
     0     1
>> c = [3;2]
c =
     3
     2
>> d = 5
d = 5
```

Common Array and Matrix Operations

```
>> a + b
ans =
     0     2
     2     2
>> a + c
ans =
     4     3
     4     3
>> a .* b
ans =
    -1     0
     0     1
```

Common Array and Matrix Operations

```
>> a * b
ans =
    -1     2
    -2     5
>> a * c
ans =
     3
     8
>> a + d
ans =
     6     5
     7     6
```

Common Array and Matrix Operations

```
>> a .* d
ans =
     5     0
    10     5

>> a * d
ans =
     5     0
    10     5
```

Matrix Transpose

```
>> arr = [1 2 3 4]
arr =
     1     2     3     4

>> arr'
ans =
     1
     2
     3
     4
```

Matrix Transpose

```
>> arr = [1 2 3 4; 5 6 7 8; 9 10 11 12]
arr =
     1     2     3     4
     5     6     7     8
     9    10    11    12

>> arr'
ans =
     1     5     9
     2     6    10
     3     7    11
     4     8    12
```

Common MATLAB Functions

Table 2.8: Common MATLAB Functions

Function	Description
	Mathematical Functions
`abs(x)`	Calculates the absolute value $\|x\|$.
`acos(x)`	Calculates $\cos^{-1}x$ (results in radians).
`acosd(x)`	Calculates $\cos^{-1}x$ (results in degrees).
`angle(x)`	Returns the phase angle of the complex value x, in radians.
`asin(x)`	Calculates $\sin^{-1}x$ (results in radians).
`asind(x)`	Calculates $\sin^{-1}x$ (results in degrees).
`atan(x)`	Calculates $\tan^{-1}x$ (results in radians).
`atand(x)`	Calculates $\tan^{-1}x$ (results in degrees).
`atan2(y,x)`	Calculates $\theta = \tan^{-1}\frac{y}{x}$ over all four quadrants of the circle (results in radians in the range $-\pi \leq \theta \leq \pi$).
`atan2d(y,x)`	Calculates $\theta = \tan^{-1}\frac{y}{x}$ over all four quadrants of the circle (results in degrees in the range $-180° \leq \theta \leq 180°$).
`cos(x)`	Calculates $\cos x$, with x in radians.
`cosd(x)`	Calculates $\cos x$, with x in degrees.
`exp(x)`	Calculates e^x.
`log(x)`	Calculates the natural logarithm $\log_e x$.
`log10(x)`	Calculates the logarithm to the base 10 $\log_{10} x$.

Common MATLAB Functions

Table 2.8: Common MATLAB Functions (*Continued*)

`[value,index] = max(x)`	Returns the maximum value in vector x, and optionally the location of that value.
`[value,index] = min(x)`	Returns the minimum value in vector x, and optionally the location of that value.
`mod(x,y)`	Remainder or modulo function.
`sin(x)`	Calculates sin x, with x in radians.
`sind(x)`	Calculates sin x, with x in degrees.
`sqrt(x)`	Calculates the square root of x.
`tan(x)`	Calculates tan x, with x in radians.
`tand(x)`	Calculates tan x, with x in degrees.
Rounding Functions	
`ceil(x)`	Rounds x to the nearest integer toward positive infinity: `ceil(3.1) = 4` and `ceil(-3.1) = -3`.
`fix(x)`	Rounds x to the nearest integer toward zero: `fix(3.1) = 3` and `fix(-3.1) = -3`.
`floor(x)`	Rounds x to the nearest integer toward minus infinity: `floor(3.1) = 3` and `floor(-3.1) = -4`.
`round(x)`	Rounds x to the nearest integer.

Common MATLAB Functions

Character Array Conversion Functions

`char(x)`	Converts a matrix of numbers into a character array. For ASCII characters the matrix should contain numbers ≤ 127.
`double(x)`	Converts a character array into a matrix of numbers.
`int2str(x)`	Converts the value of x into an character array representing the nearest integer.
`num2str(x)`	Converts the value of x into a character array representing the number.
`str2num(c)`	Converts character array c into a numeric array.

Common MATLAB Functions: Examples

```
>> maxval = max ([1 -5 6 -3])
maxval =
     6
>> [maxval, index] = max ([1 -5 6 -3])
maxval =
     6
index =
     3
>> sqrt(25)
ans =
     5
>> exp(1)
ans =
    2.7183
```

Simultaneous Linear Equations

A 3 × 3 set of simultaneous linear equations takes the form

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3$$

which can be expressed as

$$Ax = B$$

where $A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$, $B = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$, and $x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$.

If A is a non-singular matrix, the result is

$$x = A^{-1}B$$

Simultaneous Linear Equations

```
>> A = [2 1 1; -1 1 -1; 1 2 3]
A =
     2     1     1
    -1     1    -1
     1     2     3
>> B = [2; 3; -10]
B =
     2
     3
   -10
>> x = inv(A) * B
x =
     3
     1
    -5
```

$$2x + y + z = 2$$
$$-x + y - z = 3$$
$$x + 2y + 3z = -10$$

Character Arrays

- Each element of a character array stores a single character.
- A MATLAB character array is an array of type char.
- Each character is stored in two bytes of memory.
- Character array constants are defined using text strings surrounded by single quotes:

```
s = 'Hello, world';
```

- By default, MATLAB uses the Unicode character set.

```
s = 'الحمد لله';
```

Character Arrays

```
>> seq = 'GCTAGAATCC';
>> whos seq
  Name      Size            Bytes  Class    Attributes
  seq       1x10               20  char
>> seq(4)
ans =
    'A'
>> length(seq)
ans =
    10
```

Character Arrays

```
>> chr = 'Hello, world'
>> chr(end)
ans =
    'd'
>> chr(end + 1) = '!'
chr =
    'Hello, world!'
>> chr(1:5)
ans =
    'Hello'
```

Strings

- Strings are defined using text strings surrounded by double quotes:

```
>> s = "Hello, world"
s =
    "Hello, world"
>> whos s
  Name      Size            Bytes  Class     Attributes
  s         1x1               150  string
>> strlength(s)
ans =
    12
```

Strings

```
>> A = ["a","bb","ccc"; "dddd","eeeeee","fffffff"]
A =
  2×3 string array
    "a"       "bb"        "ccc"
    "dddd"    "eeeeee"    "fffffff"

>> strlength(A)
ans =
     1     2     3
     4     6     7
```

Strings

```
>> f = 71;
>> c = (f-32)/1.8;
>> tempText = "Temperature is " + c + "C"
tempText = 
    "Temperature is 21.6667C"
```

Complex Numbers

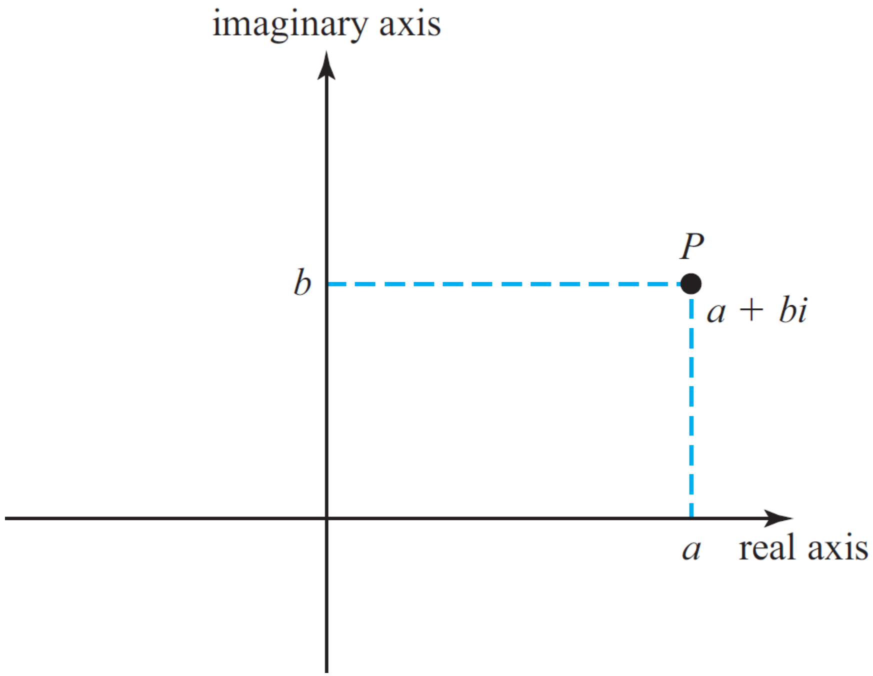
- A general complex number is in the form

$$c = a + bi$$

- The number a is called the real part and b is called the imaginary part of the complex number c.
- Where $i = \sqrt{-1}$
- In MATLAB, `i` and `j` represent the basic imaginary unit.
- Complex numbers will be used in working with signals, linear systems and various transforms.

Representing Complex Numbers in Rectangular Coordinates

- Since a complex number has two components, it can be plotted as a point on a plane using rectangular coordinates.
- The horizontal axis of the plane is the real axis, and the vertical axis of the plane is the imaginary axis.



Representing Complex Numbers in Polar Coordinates

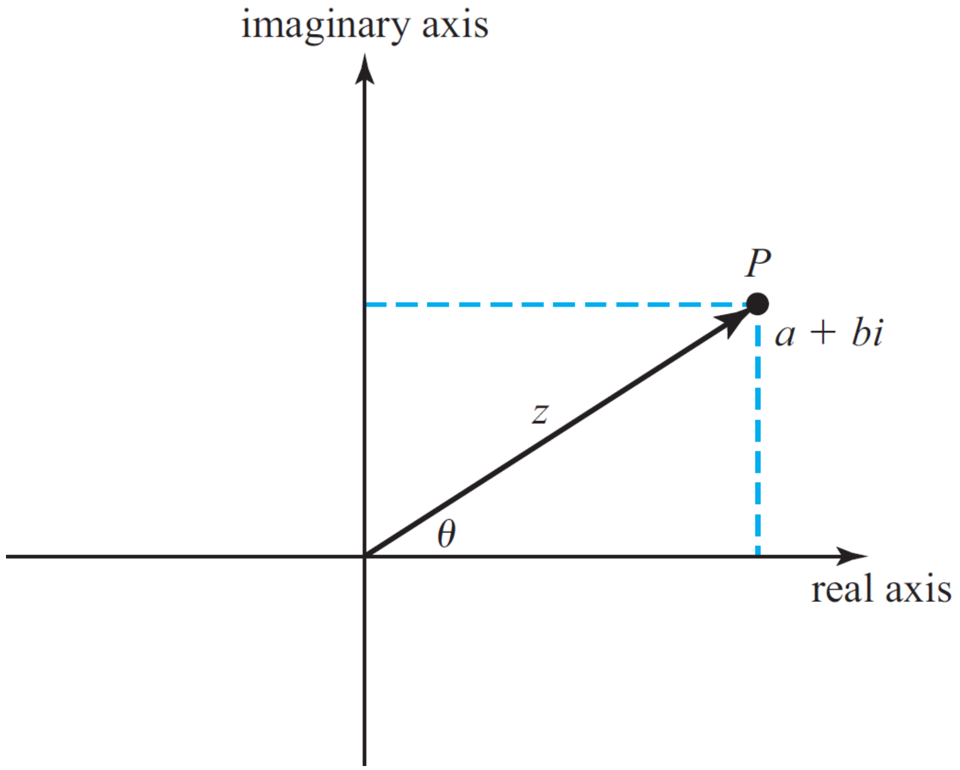
- A complex number can also be represented as a vector of length z and angle θ pointing from the origin of the plane to the point P.
- A complex number represented this way is said to be in polar coordinates.

$$z = \sqrt{a^2 + b^2}$$

$$\theta = \tan^{-1}\frac{b}{a}$$

$$a = z\cos\theta$$

$$b = z\sin\theta$$



Complex Numbers

```
sqrt(-1)
ans = 0.0000 + 1.0000i
>> c = 3 + 4i
c = 3.0000 + 4.0000i
>> real(c)
ans = 3
>> imag(c)
ans = 4
>> abs(c)
ans = 5
>> angle(c)
ans = 0.9273
```

Displaying Output Data

- When data is echoed in the Command Window, values are printed using a default format.
- The default format shows four digits after the decimal point.

```
>> sqrt(5)
ans =
     2.2361
```

- Values may be displayed in scientific notation with an exponent if the number is too large or too small.

```
>> 1000000000
ans = 1.0000e+09
```

Displaying Output Data

- The `format` command changes the default format according to the values given in Table 2.3

Table 2.3: Output Display Formats

Format Command	Results	Example[1]
`format short`	4 digits after decimal (default format)	`12.3457`
`format long`	14 digits after decimal	`12.34567890123457`
`format short e`	5 digits plus exponent	`1.2346e+001`
`format short eng`	5 digits plus exponent digits plus exponent with exponent being powers of 1000	`12.347e+000`
`format short g`	5 total digits with or without exponent	`12.346`
`format long e`	15 digits plus exponent	`1.234567890123457e+001`
`format long eng`	15 digits plus exponent with exponent being powers of 1000	`12.34567890123457e+000`
`format long g`	15 total digits with or without exponent	`12.3456789012346`
`format hex`	hexadecimal display of bits	`4028b0fcd32f707a`

Displaying Output Data

- Another way to display data is with the `disp` function.

```
>> disp([1 2 3])
     1     2     3
>> disp('Hello World.')
   Hello World.
>> A = [1 2; 3 4];
>> disp(A)
     1     2
     3     4
>> format long
>> disp(sqrt(5))
   2.236067977499790
```

Formatted Output

- An even more flexible way to display data is with the `fprintf` function.
- The `fprintf` function lets the programmer control the way that the displayed value appears.

Format String	Results
`%d`	Display value as an integer.
`%e`	Display value in exponential format.
`%f`	Display value in floating-point format.
`%g`	Display value in either floating-point or exponential format, whichever is shorter.
`\n`	Skip to a new line.

Formatted Output

```
>> fprintf('The value of pi is %f \n', pi)
The value of pi is 3.141593
>> fprintf('The value of pi is %.2f \n', pi)
The value of pi is 3.14
>> fprintf('The value of pi is %e \n', pi)
The value of pi is 3.141593e+00
>> fprintf('The value of sqrt(25) is %f \n', sqrt(25))
The value of sqrt(25) is 5.000000
>> fprintf('The value of sqrt(25) is %d \n', sqrt(25))
The value of sqrt(25) is 5
```

User Input

```
>> x = input('Enter data: ');
Enter data: 5
>> disp(x)
     5

>> s = input('Enter string: ');
Enter string: 'DSP'
>> s
s =
    'DSP'
```

User Input

```
>> A = input('Enter data: ');
Enter data: [1 2; 3 4]
>> disp(A)
     1     2
     3     4

>> expr = input('Enter data: ');
Enter data: 5+6-4
>> expr
expr =
     7
```

User Input

```
>> s = input('Enter string: ');
Enter string: 2.7
>> s
s =
   2.700000000000000

>> s = input('Enter string: ', 's');
Enter string: 2.7
>> s
s =
    '2.7'
```