Introduction to MATLAB
Lecture Group 1
Introduction

Instructor: Yan Wei(魏艳)

Spring, 2021

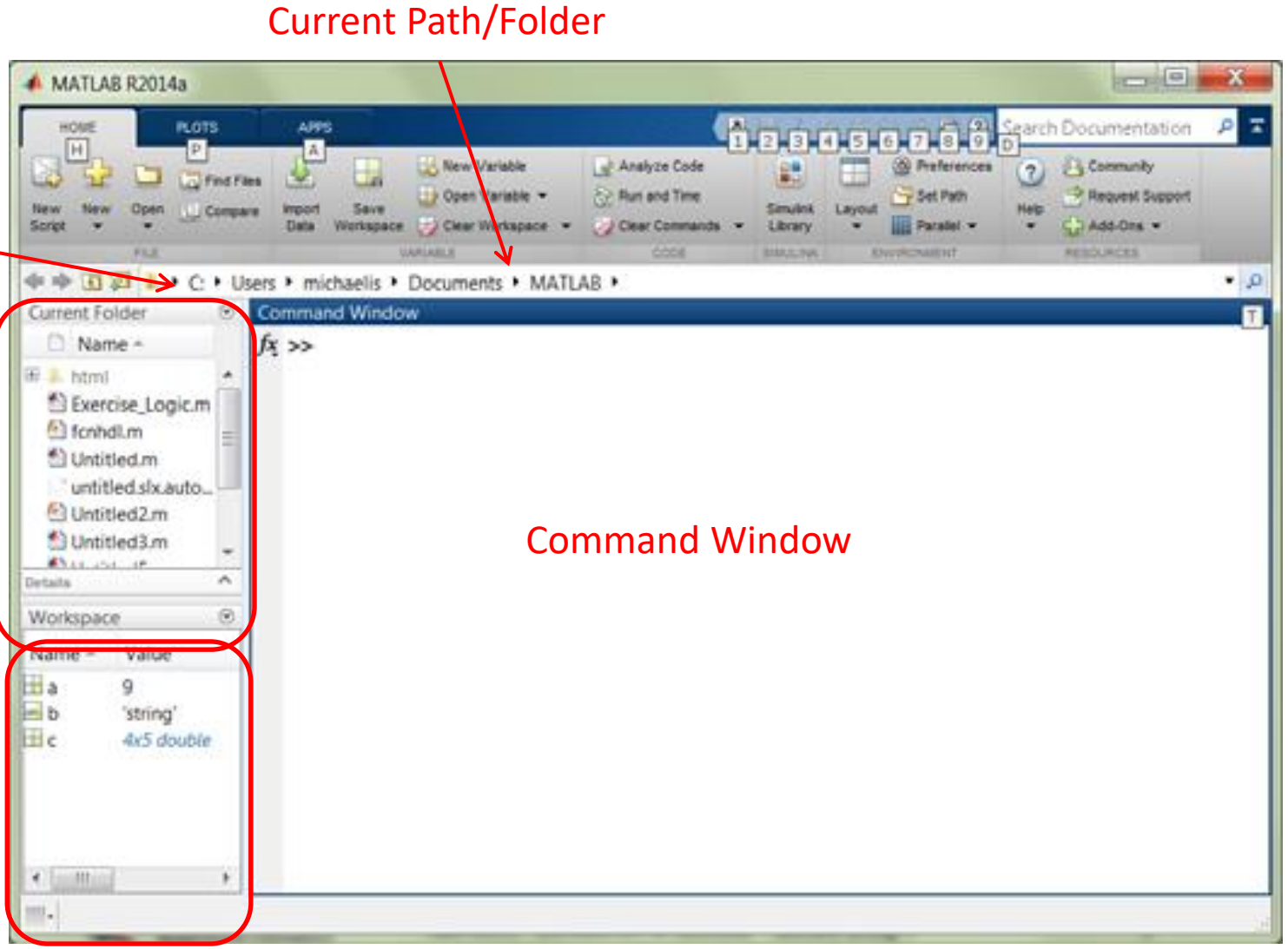
Outline

- Introduction to MATLAB
- Script m files
- Variables and Array

Introduction to MATLAB

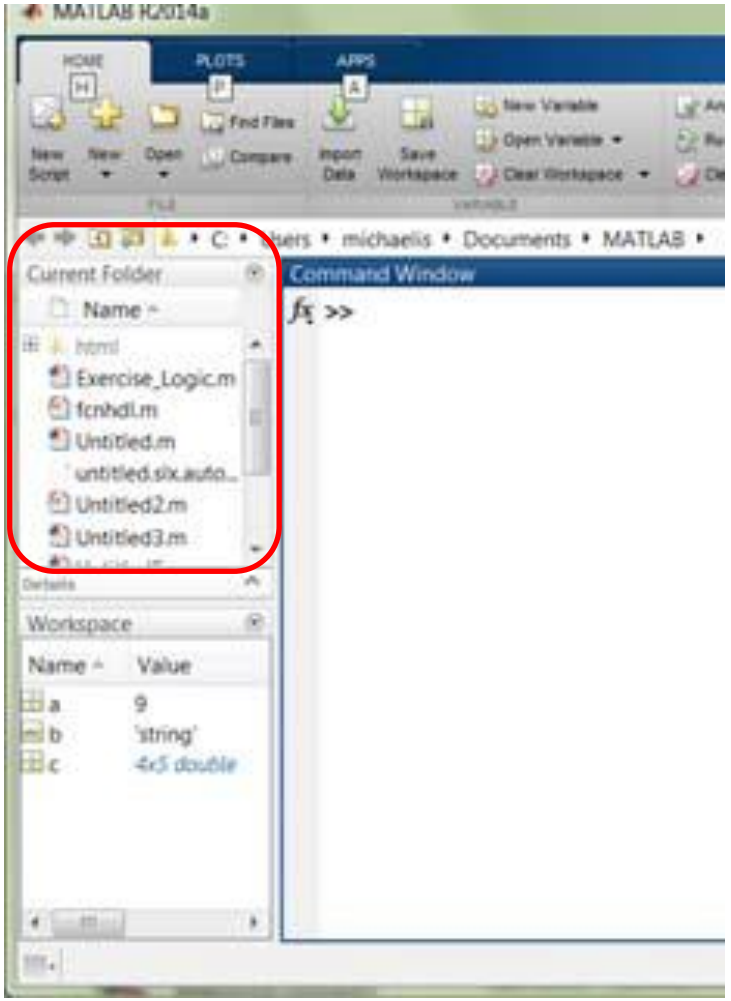
- Matrix Laboratory(MATLAB) can be used for
  - Computation/Analysis
  - Visualization
  - Programming
  - Simulation

MATLAB Components/Layout



Current Folder Window

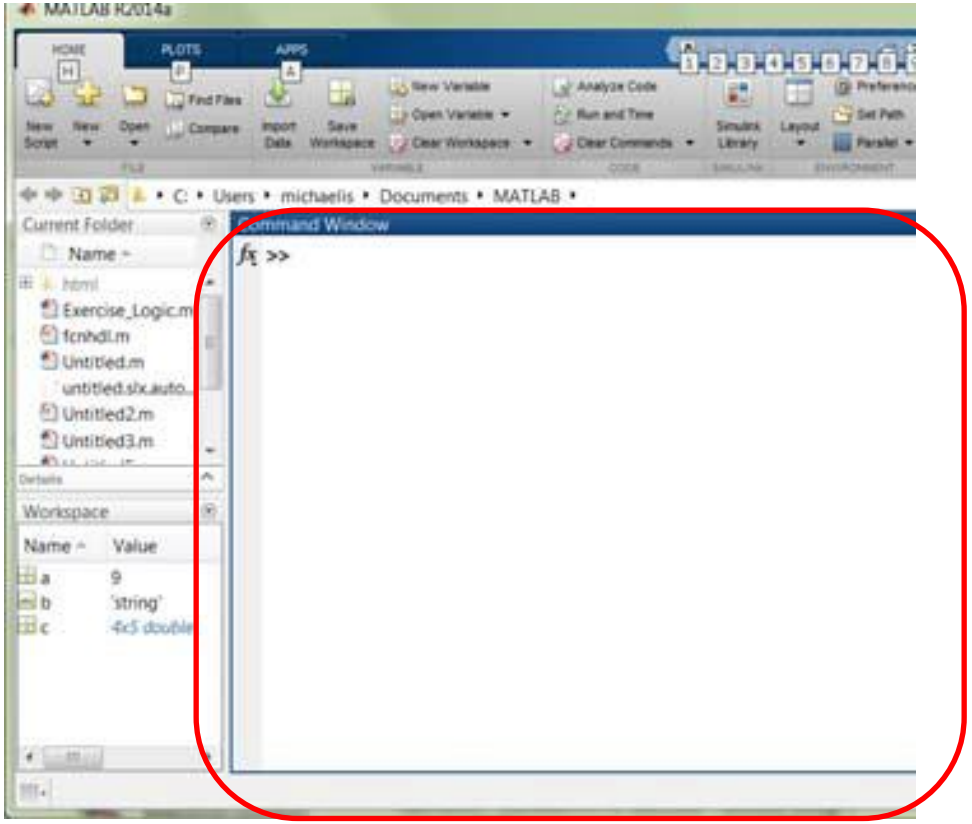
- All files in current folder are shown in this window
- You can open a file by double clicking it
- All functions called will be found in the "Current Folder" or in a path set in MATLAB



Command Window

Any function (or command) can be run by typing it in the command window

All variables created using the command window are stored in the Workspace



Command Window

The Command Window in the center is the main panel where you interact with MATLAB.

Type the commands (i.e., built-in functions or your own functions) after the prompt "**>>**" and type <Enter>. MATLAB then executes the commands and displays results (if requested).

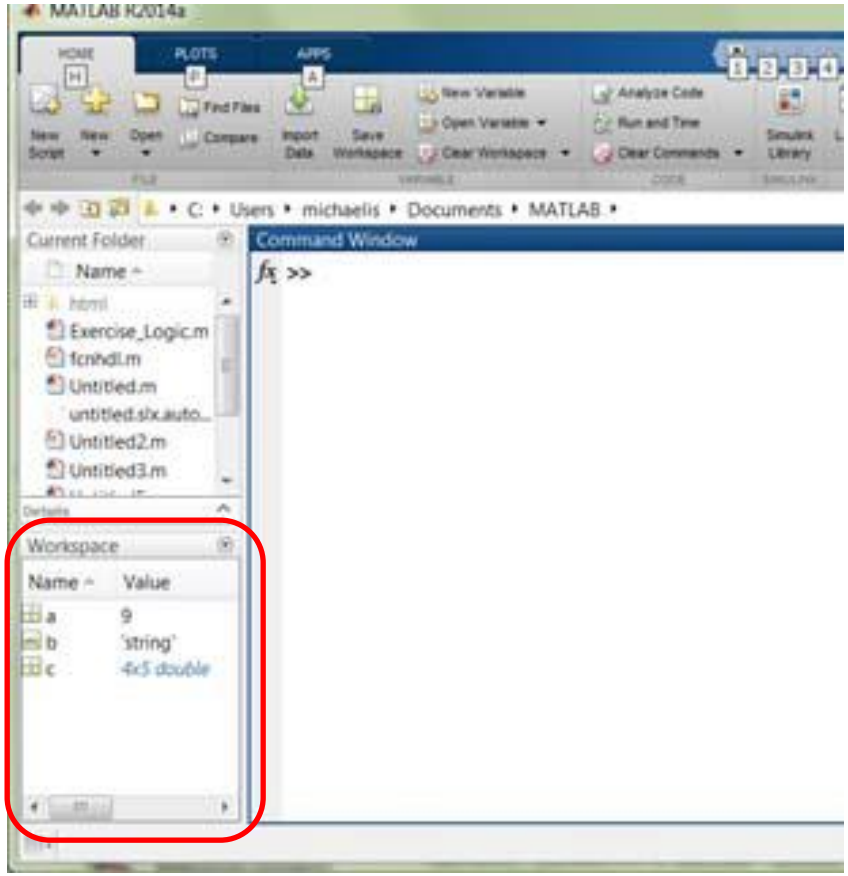
Some useful tools and commands:
– **clc** – clears the screen
– **clear all** – clears variables in Workspace
– ↑(up arrow) returns the last command input and a selectable list of all previously entered commands

Workspace Window

All variables created at command window or a script (covered later) are stored in the Workspace

Variables used in a function are NOT stored in Workspace

In the figure, variable **a**, **b**, and **c** are currently stored in the Workspace



MATLAB as a Calculator

Using various built-in functions in MATLAB, you can use MATLAB like an advanced calculator by entering command directly into the command window.

This isn't very efficient but this functionality is very helpful when doing quick investigations into the current workspace variables or debugging a program.

Entering numbers in MATLAB. All the following are equivalent (try it)

```
>> 0.00051
>> 5.1*10^-4
>> 5.1e-4 %scientific e notation

  ans = 5.1000e-4
```

Basic Operators and Functions

Basic Operators

+	**Addition**
-	**Subtraction**
*****	**Multiplication**
/	**Division**
^	**Exponentiation**
( )	**Parenthesis**

Sample Built-In Functions

sqrt( ) % $\sqrt{x}$

sin( )

cos( )

exp( ) % e^x

Search MATLAB help for “Elementary Math” for more details

Example: MATLAB as a Calculator

>> 3–2 <Enter>
>> 3*2
>> 3^2
>> 2/0
>> 0/2
>> 3*(4+7)

>> sin(3.14)
>> exp(3) **% e^3**
>> sqrt(25) **% $\sqrt{25}$**

Notice that each output is stored/assigned to the variable "**ans**" (short for answer) which is then overwritten by each new command

Variables in MATLAB

- You can store numerical values, characters, and strings into variables
- MATLAB stores all variables as matrices/arrays
  – A single value is just a 1 x 1 matrix
- Meaningful variable names will help you make a good program
  (detailed naming rules are covered later)
    – Variable names are case sensitive – be careful
    –Variable, VARIABLE, variable, and vaRIAble are all different
- The data stored in a variable is stored in the memory assigned for MATLAB in your computer and when you exit MATLAB the data is erased
- Variables created are shown in the workspace window
- **clear all** will erase all variables (and other stuff) from the workspace

Examples: Using Variables

>> a = 3 <Enter>
>> b = 2
>> A = 5
>> a – b
>> a/b
>> a^2
>> c = a*b
>> d = c^(b+1)
>> a = a + 3
>> q = 'This is a string' % notice the single quotes ' and purple color
Notice that all variables created are now shown in the Workspace.
>> clear all % clear/erase all variables from workspace

Hiding Output using “;”

• Command Window output can be suppressed using a semicolon “**;**” at the end of any command

• Examples:

>> clear; clc <Enter>

>> a = 3; % notice lack of output to command window

>> b = 2;

>> c = a*b;

>> d = c^(b+1);

>> a, b, c, d % displays the values stored in each of the variables

• Note: Multiple commands can be entered on the SAME LINE if separated by a comma “**,**” or a semicolon “**;**”

MATLAB Arrays

- By default, variables in MATLAB are arrays/matrices (i.e. they can contains **multiple objects** (usually numbers or strings)
- Typical arrays include:
  - Single element array, 1 x 1 (also called a scalar)
  - 1D array, 1 x n / n x 1 array (also called a vector)
    - Examples: $1 \times 3 \rightarrow \ （-2 \ \ 8 \ \ 9.3）$ and $3 \times 1 \rightarrow \begin{pmatrix} 3 \\ -2 \\ 7.1 \end{pmatrix}$
  - 2D array, m x n array (also called a matrix)
    - Example: $2 \times 3 \rightarrow \begin{pmatrix} 4 & -2.1 & 4 \\ 1.1 & 3 & 40 \end{pmatrix}$
  - 3D, 4D, etc. arrays are also possible

Creating Arrays in MATLAB

- Square brackets **[ ]** are used to create MATLAB arrays/vectors/matrices

- Examples:

  >> a=[1 3 5 8]%1D,1x4 array/vector →a = 1 3 5 8

  >> b=[5,2,-1] %1D,1x3 array/vector→b = 5 2 -1

  >> c=[-2;4;3] %1D,3x1 array/vector→c = $\begin{matrix} -2 \\ 4 \\ 3 \end{matrix}$

  >> d=[1 2 4;5 6 9] %2D, 2x3 matrix → d = $\begin{matrix} 1 & 2 & 4 \\ 5 & 6 & 9 \end{matrix}$

- Elements in rows can be separated by either a 'space' OR a comma ','
- New rows are denoted with a semicolon ';'

Creating Evenly Spaced Arrays

- Colon "**:**" notation can be used to define evenly spaced (potentially large) arrays
- Format is **first:increment:last**
- Examples:

```
>> H = 1:1:6 %→ H = 1 2 3 4 5 6
>> I = -1:2:9 %→ I = -1 1 3 5 7 9
>> J = 18:-3:3%→ J = 18 15 12 9 6 3
>> K = [1:1:10 ; 3:3:30]
    %K = 1 2 3  4  5  6  7  8  9 10
         3 6 9 12 15 18 21 24 27 30
```

- Note: Arrays MUST have the same number of elements in each row/column.

Element-by-Element vs Matrix Math

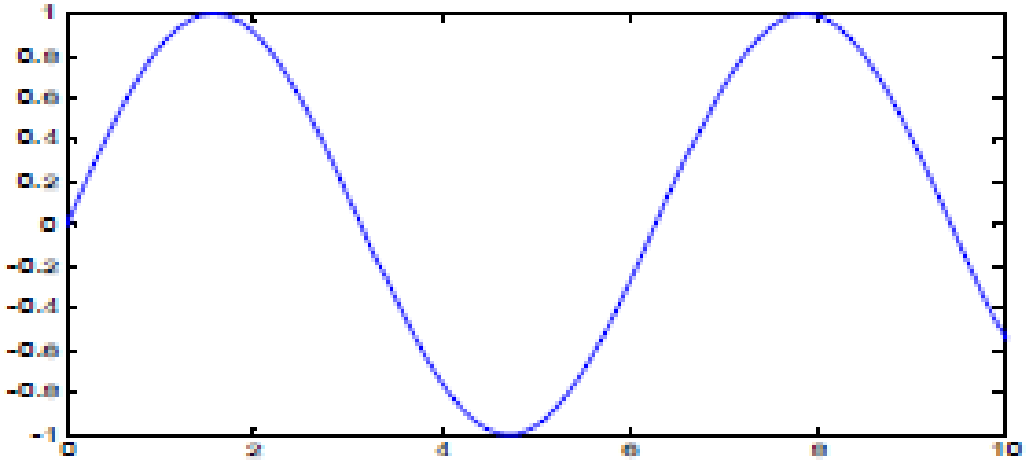
- By default, MATLAB performs matrix math when dealing with arrays/vectors/matrices.
- Often element-by-element operations are needed and MATLAB has a shorthand dot '**.**' method for doing computations on every element of an array
- For example:

```
>> a = [1 2 3];
>> b = a.^2 %note the dot '.' before the ^
        b = [1 4 9]
>> b = a^2 %gives an ERROR
```

- The dot '**.**' method is required for **.***, **./**, and **.^** operators. It is not needed, and doesn't work, for **+** or **-** operators
- The dot method is not needed (but still works) when dealing with scalars. E.g. **>> a = 3, b = 4, a*b^2**

Introduction to Plotting

```
>> x = [0:0.1:10]; %Build x array
>> y = sin(x);     %Build y array
>> plot(x,y)       %Open figure and create   simple x y plot
>> close all       %Close all open figures
```



Help Utilities

- Use “help” command in the command window to output info to command window

  >> help subplot

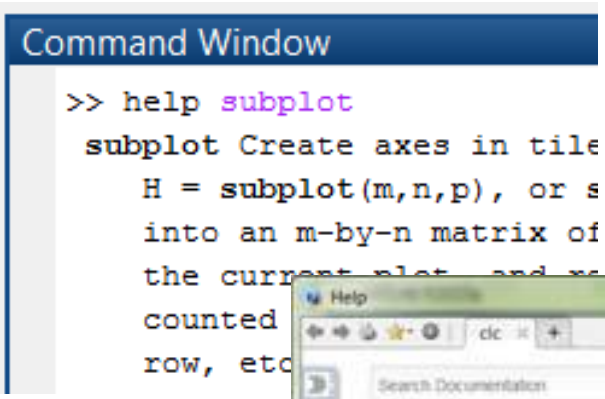
- Use “doc” command in the command window to open help window

  >> doc clc

- Use “lookfor” to find a keyword in MATLAB or any of your programs in the current path.

  >> lookfor cone

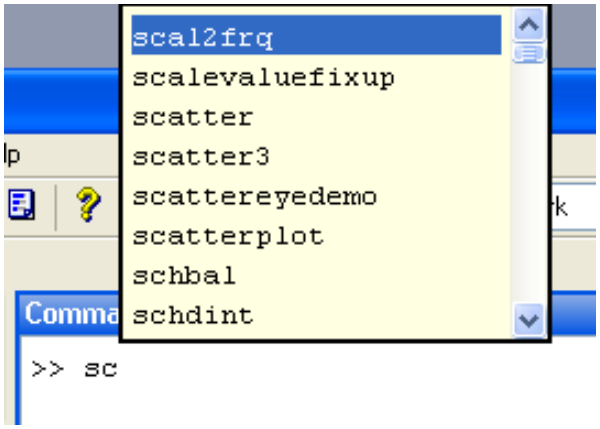
- Standard help search box located at the upper right of window (careful since same function can be in multiple MATLAB modules)

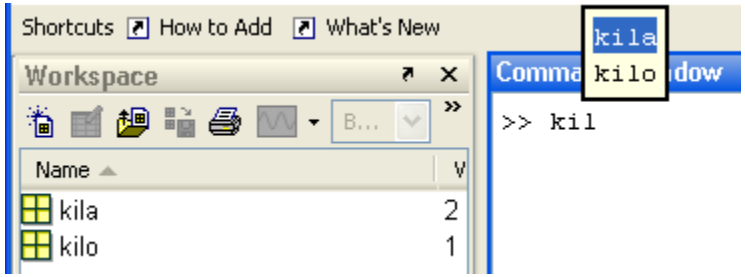


Tab completion

- The “Tab” button can be used to find functions if you only know the first few letters of a function or variable.

- In the Command Window, type the first few letters then press the “Tab” button to get a list of all available functions.





Doing a Simple Calculation

- Consider computing the volume of a cone:

volume = (pi*r^2*h)/3
radius = 6 inches
height = 12 inches

- In the command window key in:

>> clear; clc; <Enter>
>> r = 6 <Enter>
>> h = 12 <Enter>
>> v = (pi*r^2*h)/3 <Enter>
v = 452.3893

Simple Calculation Continued

- What if we want to calculate the volume for multiple values?
- How about finding the volume for 4 radii at a time instead of only 1?

>> r = [1 2 3 4], h = 4

>> v = (pi*r.^2*h)/3 %note the added "."

so that element-by-element calculations

are done so all 4 radii are squared

- Or how about volume for 5 heights but only 1 radius?

>> r = 2, h = [2:2:10]

>> v = (pi*r^2*h)/3 %note that the period

is not needed for scalar multiplication

- Volume for 4 radii by 5 heights is more advanced

In-Class Exercise – Command Line

Use the command line to calculate the following (some of the expected results are shown in parentheses):

1. Create your name as a string and assign it to the variable “Name”.

2. Calculate $\frac{15}{8\times10}$ (ans = 0.1875)

3. Assign variable “a” to have a value of 6

4. Create a variable “c” and set it equal to variable “a” times 2 (c = 12).

5. Create a variable “w” that is equal to:

$$w = \begin{matrix} 18 & 12 & 1 \\ 4 & 8 & 1 \end{matrix}$$

6. Find the cube of each element in w. That is, w^3

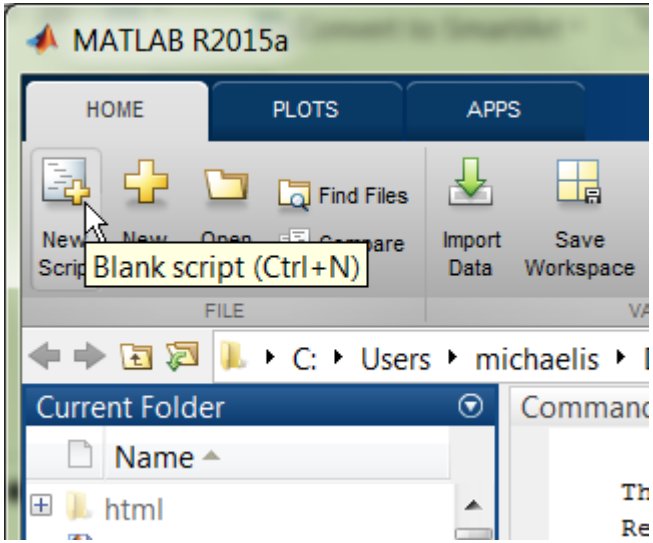
$$\left(ans = \begin{matrix} 5832 & 1728 & 1 \\ 64 & 512 & 1 \end{matrix}\right)$$

MATLAB Programs/Scripts

- If you put a collection of commands into a file, it becomes a MATLAB program
- MATLAB is built around these files (scripts and functions)
- When a MATLAB program is run, it is the same as sequentially entering the commands in the Command Window.
- Controlling the execution sequence of built-in functions or your own functions is called “programming” in MATLAB.
- The execution flow can be controlled by some keywords such as **if**, **else**, **for**, **end**, etc. (covered later).
- MATLAB programs all have a ‘.m’ extension (e.g. MyFirstProgram.m)

Creating a MATLAB Program/Script

- Click on “New Script” at the upper left corner of the “Home” tab to create a new script (*.m) file.
- The MATLAB editor window is opened whenever a new script is created or an existing one is opened.

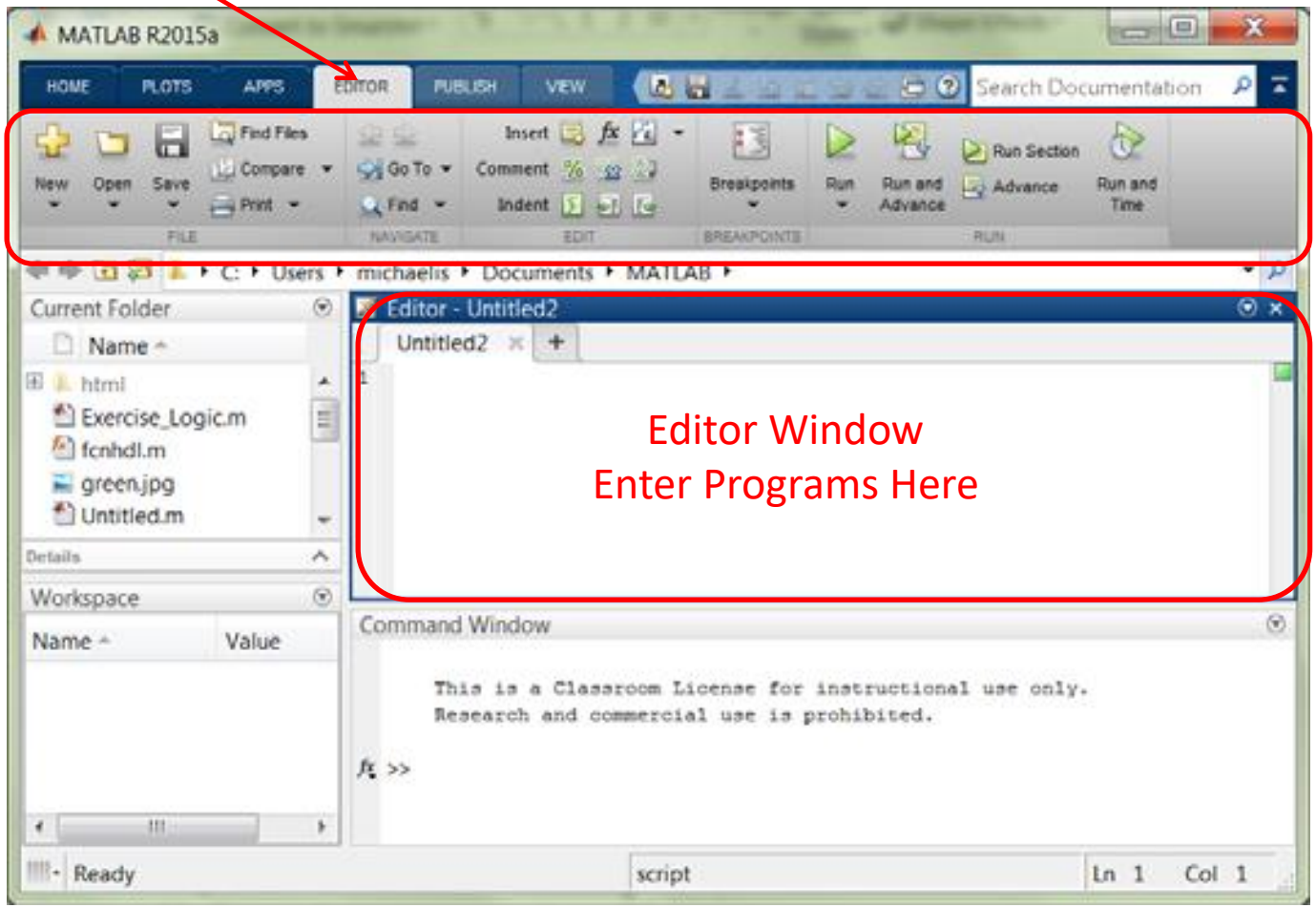


MATLAB Program Editor

- MATLAB has a built-in program/script editor that can be used to write your programs (i.e. creating and editing your mfile files).

- A MATLAB m-file (*.m) is simply a text file (*.txt) containing a collection of MATLAB commands. Thus, it can actually be edited in in any text editor also (Notepad, Word, etc.)

MATLAB Editor Window



Creating Your First MATLAB Program

- Create a new script as described previously
- Type in text in the Editor so that it looks similar this:

Comments (green)

Executed Code (black)

Standard Script Beginning

Editor - C:\Users\michaelis\Dropbox\Teaching\MENG 201 - Matlab\Lectures\cone.m

cone.m

```
%-------------------------------------------------
% MATLAB script/program to calculate volume of a cone
% YYYY-MM-DD
% M. Michaelis
%-------------------------------------------------
clear all;close all;clc %Clear variables, figures, screen

%parameters
rr = 4;              %Radius of cone (mm)
h = 15;              %Height of cone (mm)

%calculate volume
v = (pi*(r^2)*h)/3   %Volume of a cone (mm^3)
```

- Save the file as "**cone.m**"

Execute/Run a MATLAB Program

- There are several ways to run your program

1. Pushing the green “run” button
2. Pushing F5
3. Typing on the Command Line

>> cone <Enter>

- If you entered the code as written on the previous slide you get the following error in the Command Window!

Error in cone (line 13)

v = (pi*(r^2)*h)/3 %Volume of a cone (mm^3)

- Can you spot the error?
- Repair your program (Change rr = 4 to r = 4.), save it, and run it again.
- With programs, changes are easy so change the height to 24 and save and run your program again.

Comments in programs

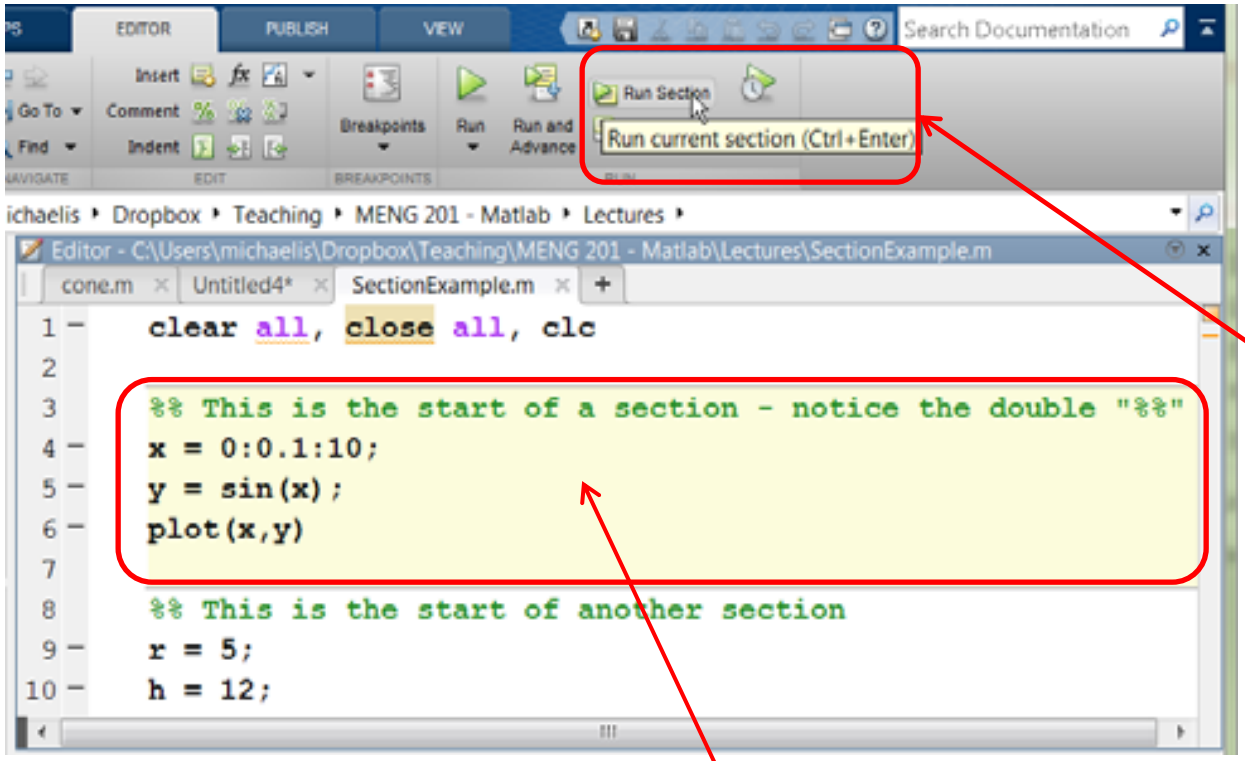
- Every time you write a program to be saved, it is helpful for you to comment (i.e. describe) it well.
- To insert a comment on a line in the Editor or in the Command Window, use the comment operator "**%**", then type your comment
- MATLAB:
  - Will not run lines that begin with the comment operator (in the editor comments appear in green). See **cone.m** on previous slide for examples
- Notes:
  - Comments allow you (and others) to more easily understand your program
  - When your lines of code are easy to understand, your code will be easier to use later
  - A comment can be an entire, or only part of a, line

Use lots of comments!!

They are 10% of your homework score!!

What is a section ?

A way to break programs into smaller parts



Active section is highlighted a yellowish color (click in section to make it active)

Notes:

- A section, and only that section, can be executed/run by choosing “Run Section” or pressing Ctrl+Enter

- The %% MUST have a space after them or a section isn’t created. E.g. %% New Section NOT %%New Section.

Editor Features

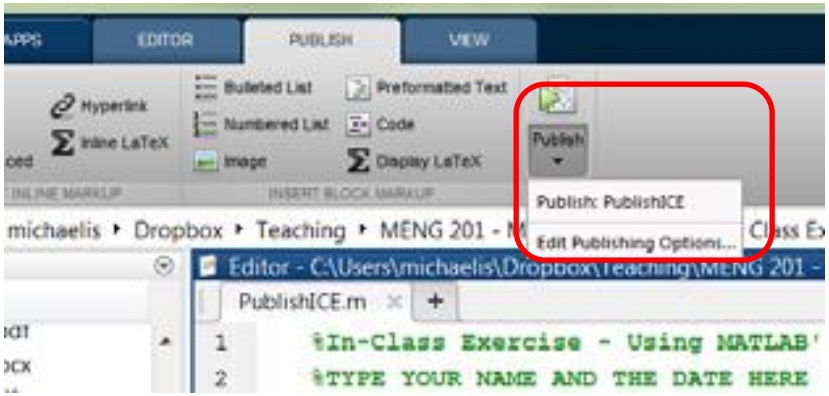
- Editor test color indicates command type
  - black: executed code, e.g. **a = (3+c)*b**
  - green: comments (not executed),
    e.g. **%This is a comment**
  - blue: reserved words,
    e.g. **if**, **else**, **for**, **while**, etc.
  - purple: strings (i.e. text),
    e.g. **q = 'This is a string'**
- Matching parenthesis pairs (i.e. **( )**) are underlined to aid in formula debugging, e.g. **a = (1+2))**
- Unmatched parentheses are strikethrough,
  e.g. a = (2+b)}

Art of well-written code

- A well-written program is like literature; it contains comments explaining:
  - What your program requires as input
  - What the variables in the program represent including units
  - What your program computes and displays
- It is useful for you to add a set of header comments containing:
  - The name of the program
  - Your name (as the programmer)
  - Date the program was created or modified
  - What the program is designed to accomplish

Publishing a MATLAB script

- A MATLAB script with all of its command window and figure outputs can be published into any of the following formats:
  - Word Document
  - Pdf
  - Power Point
  - HTML (default)
  - Others
- Search help for "Publishing" for lots of cool formatting options



Publishing Example

MATLAB Script

```
%% Homework 1
% Simple Plotting and Arrays,
% M. Michaelis, 2016-01-05

%% Problem 1
% Generate and Plot Data for
%
% $y = e^{-x/4}sin(x)$
clear all;close all;clc;
x = [0:0.1:4*pi]; %Suppress output of large arrays
y = exp(-x/4).*sin(x);  %suppressed output is NOT published
plot(x,y)

%% Problem 2
% Create a 2D array
clear all;close all;clc;
A = [3 3.2 6 4;6 8 9.7 10] %unsuppressed output is published
```

TITLE
SUBSTITLE

HEADING 1

HEADING 2

Resulting Word Document

Homework 1

Simple Plotting and Arrays, M. Michaelis, 2016-01-05

Problem 1 1

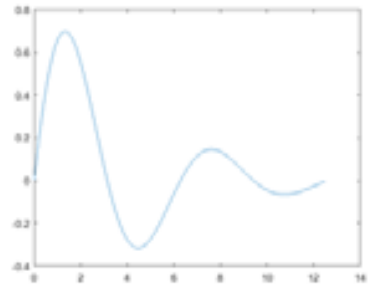
Problem 2 1

Problem 1

Generate and Plot Data for

$$y = e^{-x/4}sin(x)$$

```
clear all;close all;clc;
x = [0:0.1:4*pi]; %Suppress output of large arrays
y = exp(-x/4).*sin(x);  %suppressed output is NOT published
plot(x,y)
```



Problem 2

Create a 2D array

```
clear all;close all;clc;
A = [3 3.2 6 4;6 8 9.7 10] %unsuppressed output is published
```

```
A =

    3.0000    3.2000    6.0000    4.0000
    6.0000    8.0000    9.7000   10.0000
```

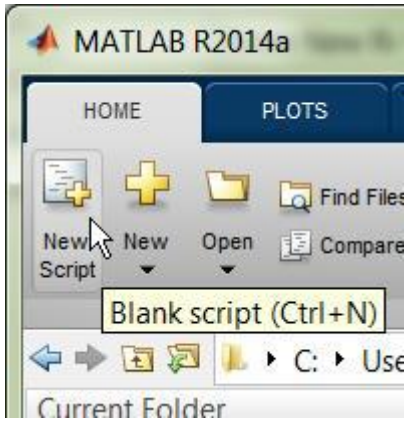
Your Script
Sections/Code

Miscellaneous Useful Commands and Some Special Characters

- "clear" to clear variables, e.g. **clear all** or **clear varA**
- "clc" to clear command window
- "close" to close figure, e.g. **close all** or **close(1)**
- ";" at the end of line hides the results so they are not to be displayed in command window. E.g. **varA = 1:1:100;**
- "%" signifies start of comments
- "," or ";" can be used to put multiple commands on a single line, e.g. **>> VarA=3, b=4; c=[2 3 4], d=12;**
- "…" can be used to break long commands into multiple lines.
  e.g. **>> a = [1 2 3 4 5 6 7 8 9 …**
  10 11 12 13 14 15 16]
- "input( )", e.g. **>> a = input('Enter a number')**
- "format long" or "format short" % how many digits displayed

In-Class Exercise

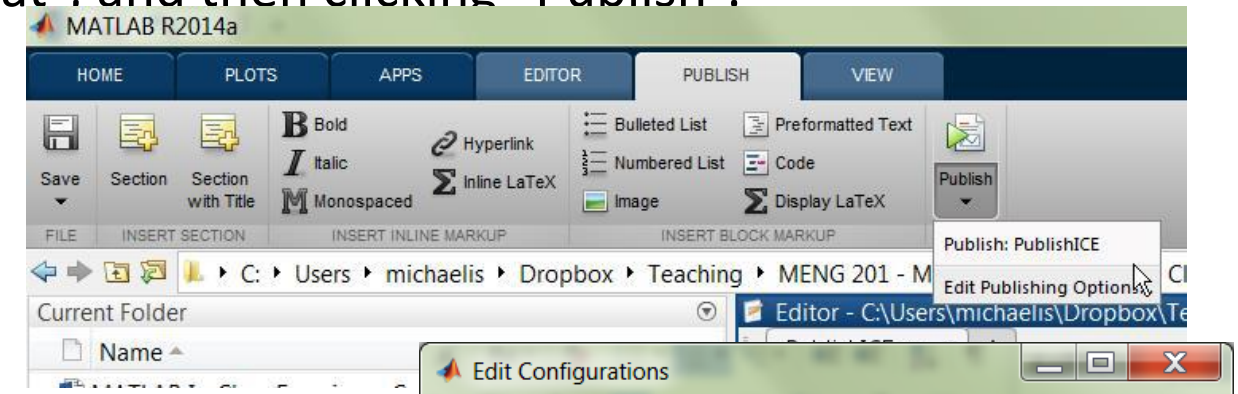
1. Create a blank MATLAB script file.

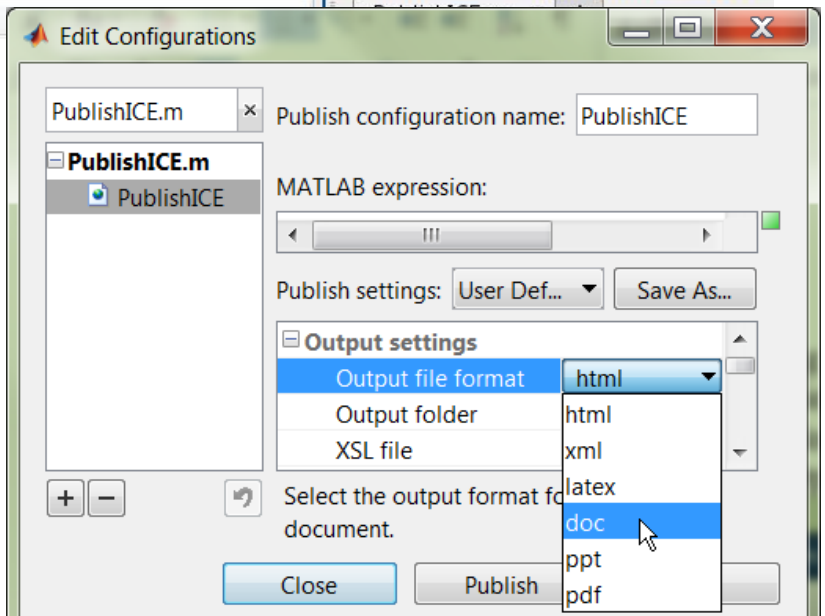


2. Copy and paste the following into the script

```
%% In-Class Exercise - Using MATLAB's Publish Feature
% TYPE YOUR NAME AND THE DATE HERE
%
% This exercise illustrates MATLAB's publish feature for a script with user
% inputs, sections, and a simple plot
%
% Note: If the first section is all comments (i.e. doesn't include any
% commands), it becomes the published document's title and header.
%% Section 1: Define Constants and Parameters
clear all;close all;clc;
a = input('input a: ') %Prompt user for value
%a = 7 %hard coded for publishing, i.e. pretend the user input “7”
%% Section 2: Generate Data
% Using the data input by the user, create a set of time values
% and calculate displacement
x = 1:a %output NOT suppressed so output is included when published
y = x.^3; %output IS suppressed so this won't output when published
%% Section 3: Plot data
% Plot data with title, axis labels with units, and other info
plot(x,y)
title('Membrane Displacement')
xlabel('Time (s)') % For full credit ALWAYS include units
ylabel('Displacement (m)')
% Add your name to your plot
text(1.5, 10,'REPLACE WITH YOUR NAME')
```

3. Run the script by pressing F5 and enter a number in the Command Window when prompted.
4. Modify the script by following the two (2) instructions in ALL CAPS embedded in the script.
5. Publish a the script file to a word document by selecting “Edit Publishing Options” from the Publish tab, selecting “doc” as the “Output file format”, and then clicking “Publish”.





6. In the word file generated, find the error produced by the input statement. In the script, the error can be fixed by commenting out (i.e. add “%” in front of it) the line with the “input” statement and making the line below it active by removing the comment character “%”. Modify but do NOT delete both lines.

7. Publish the script again after dealing with the input statement. If done correctly, there should be no errors and the command line outputs and plot should be in the document.

8. Compare the script file and the published file. Notice how the sections (i.e. parts of the script separate by lines starting with “%% ”) become section headers in the published document and everything output to the command line (i.e. all output not suppressed with a “;” at the end) or plotted is included just after each section.

Variables and Arrays

- What are variables?
  – Variables are arrays of objects.
  – You name the variables (as the programmer) and assign them numerical values, text, etc.
  – You execute the assignment command “=“ to place the variable in the workspace memory (memory is part of hardware used for storing information).
  – You are allowed to use the variable in algebraic expressions, etc. once it is assigned.

Variable Naming Rules

- Must begin with a LETTER
- May only contain **letters**, **numbers** and **underscores** '_'
- No spaces or punctuation marks allowed!
- Only the first 63 characters are significant; beyond that the names are truncated.
- Case sensitive (e.g. the variables **a** and **A** are ***NOT*** the same)

Variable Naming Rules Cont.

- Reserved keywords are not allowed as a variable names – (e.g. **if**, **else**, **elseif**, **end**, **for**,

while, **switch**, **case**, **return**, **break**, **function**, or any word that is displayed in **blue** in the MATLAB Editor or Command Window)

- While valid variable names, avoid using (overwriting/overloading) built-in MATLAB variables like **pi**, **i**, **j**, **Inf**, **inf**, **NaN**, **nan**, **eps**, etc.
- Same rules apply to script and user-defined function (*.m) file names

Which variable names are valid?

12oclockRock

%NO – begins with a number

tertiarySector

%Yes

blue cows

%No – spaces not allowed

Eiffel65

%Yes – numbers OK sometimes

red_bananas

%Yes

This_Variable_Name_Is_Quite_Possibly_Too_Long_To_Be_Considered_Good_Practice_However_It_Will_Work

%Yes – BUT the green part is not part of the recognized name

pi = 3

%Yes – BUT you are overwriting a built-in MATLAB variable and screwing with a universal constant

Built-In Variables

- MATLAB has several built-in variables:
  - “ans” : Default variable used for results
  - “pi” : 3.141592653589793
  - “inf” or “Inf” : infinite
  - “NaN” : not a number (e.g., 0/0 or Inf*0)
  - “i or j” : sqrt(-1)
  - “nargin” : Number of function input arguments
  - “nargout” : Number of function output arguments
  - “true” : Logical true (equal to 1)
  - “false" : Logical false (equal to 0)

Variable Naming Conventions

- There are different ways to name variables. The following illustrate some of the conventions used:
  - **lowerCamelCase**
  - **UpperCamelCase**
  - **underscore_convention**
- If a variable is a constant, some programmers use all caps:
  - **CONSTANT**
- It does not matter which convention you choose to work with; it is up to you.

Variables as Arrays

- In MATLAB, a variable is stored as an array of objects (numbers, text, etc.). When appropriate, it is interpreted as a scalar, vector or matrix.

scalar	vector	matrix
1 × 1	n × 1 or 1 × n	n × m

- where n and m are positive integers. The size of an array is specified by the number of rows and the number of columns in the array, with the number of rows indicated first. E.g. rows × columns

Scalar Variables

- Scalars are 1 × 1 arrays
- They contain a single value, for example:

  r = 5

  height = 5.3

  width = 9.07

Vectors

- A vector is a list of numbers expressed as a 1 dimensional array.
- A vector can be a $1 \times n$ row vector

  (2.1 -1 56)

Or an $n \times 1$ column vector

$$\begin{pmatrix} 34 \\ -5.4 \\ 2 \\ 7 \end{pmatrix}$$

Matrices

- A matrix is a two dimensional array of numbers.
- For example, this 4×3 matrix has 4 rows and 3 columns

Rows \ Columns	1	2	3
1	3.0	1.8	3.6
2	4.6	-2.0	21.3
3	0.0	-6.1	12.8
4	2.3	0.3	-6.1

Use m = [3.0 1.8 3.6; 4.6 -2.0 2.13; 0.0 -6.1 12.8; 2.3 0.3 -6.1] to create this matrix

Defining (or assigning) arrays

- An array can be defined by typing in a list of numbers enclosed in square brackets:

– Commas or spaces separate numbers.

>> A = [12 18 -3] or A = [12, 18, -3]

A = 12 18 -3

– Semicolons indicate a new row.

>> B = [2 5 2; 1 1 2; 0 -2 6]

$$B = \begin{matrix} 2 & 5 & 2 \\ 1 & 1 & 2 \\ 0 & -2 & 6 \end{matrix}$$

Defining Arrays Continued

- You can define an array in terms of other arrays:

C = [A;B]

$$C = \begin{matrix} 12 & 18 & -3 \\ 2 & 5 & 2 \\ 1 & 1 & 2 \\ 0 & -2 & 6 \end{matrix}$$

D=[C,C]

```
D =

    12    18    -3    12    18    -3
     2     5     2     2     5     2
     1     1     2     1     1     2
     0    -2     6     0    -2     6
```

The equal sign “=” assigns

Consider the command lines:

>> ax = 5;

>> bx = [1 2];

>> by = [3 4];

>> b = bx + by; %yields b = [4 6]

>> b = b + 2; %yields b = [6 8] -Adds 2 to whatever the current value of b is and stores it back in b. Note that both side of this statement cannot be equal to each other

The equal sign “**=**” computes the value of the formula on the right side and stores is in the variable named on the left side; thus, it is an assignment operation. It does **not** require both sides to be equal.

Basic Operators

+ Addition

- Subtraction

*** Multiplication**

/ Right division (traditional)

\ Left division

^ Exponentiation

( ) Parenthesis

Search help for “Elementary Math” for more details

Element by Element Operations

Operation	Algebraic Syntax	MATLAB Syntax
Addition	a + b	`a + b`
Subtraction	a - b	`a - b`
Multiplication	a × b	`a .* b`
Division	a ÷ b	`a ./ b`
Exponentiation	a^b	`a .^ b`

Operation is done to every value in the array by adding the period

Array operations & dot-operators .* , ./ and .^

- Because scalars are equivalent to a 1×1 array, you can either use the standard or the dot-operators when doing multiplication, division and exponentiation of scalars (i.e., of single numbers).
- It is okay for you to always use the dot-operators, unless you intend to perform vector or matrix multiplication or division.

Element by Element Examples

```
a = 2, b = [1 0 2], c = [3 1 4]
>> f = b.*a
                f = 2   0   4
 >> g = c./a  % right division
                g = 1.5  0.5  2.0
  >> h = a.\c   % left division
                h = 1.5  0.5  2.0
>> i = b*c    % missing the "."
                    Error using *
                    Inner matrix dimensions must agree.
>> j = c.^b
                j = 3    1    16
```

Matrix Operations

- Arrays of numbers in MATLAB can be interpreted as vectors and matrices if vector or matrix algebra is to be applied.
- Matrices are mathematical objects that can be multiplied by the rules of matrices.
- To do matrix multiplication, you need to use the standard *****, **/**, and **^** operators without the preceding "." (dot)
- They are *not* for array multiplication, division and exponentiation

Array vs. Matrix Operations

Example:

```
>> x = [ 2 1; 2 3]
>> y = [5 6; 7 8]
>> z = x .* y   % array multiplication
```

```
z =  10     6
     14    24
```

```
>> zm = x * y  % matrix multiplication
```

```
zm =  17    20
      31    36
```

Try reversing the variable order

Hierarchy of Operations

- Just like in mathematics the operations are done in the following order:
- *Left to right* doing what is in
- Parentheses & Exponents first, followed by Multiplication & Division, and then Addition & Subtraction last.

An example:

c = 2+3^2+1/(1+2)

Additional Hierarchy

- Transpose of an array (') has the highest precedence
- Colon operator (:) to create a range of numbers has low precedence

  >> a = 1+1:1:5

% a = 2 3 4 5 since this is really 2:5

- Creating the vector increases precedence

  >> b = 1+[1:1:5]

% b = 2 3 4 5 6

Creating Zeros & Ones arrays

- Create an array of zeros:

E = zeros(3,5)

```
E =
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
```

Create an array of ones:

F = ones(2,3)

```
F =
     1     1     1
     1     1     1
```

Note: Placing a single number inside either function will return an $n \times n$ array.
e.g. `ones(4)` will return a 4 × 4 array filled with ones.

Other built-in array generating functions

- Create an identity matrix:

  G = eye(3)

```
G =
     1     0     0
     0     1     0
     0     0     1
```

- Create a linear spaced vector (similar to using colon "**:**"):

H = linspace(1,2,6)

H = 1.0 1.2 1.4 1.6 1.8 2.0

Creates vector with 6 evenly spaced elements between 1 and 2.

Can also use: H = [1:1/(6-1):2] to do the same thing.

Retrieving Values in an Array

- Index – a number used to identify elements in an array (starts at 1)

– Retrieving a value from an array:

>>G = [1 2 3: 4 5 6: 7 8 9]

G =

1	2	3
4	5	6
7	8	9

>> G (2,1) % row 2 col 1

ans = 4

>> G (3, 2) % row 3 col 2

ans = 8

Indexed-location of numbers in a matrix array

- Each item in an array is located in the (row, column)

>> m(2,3)

ans = 21.3

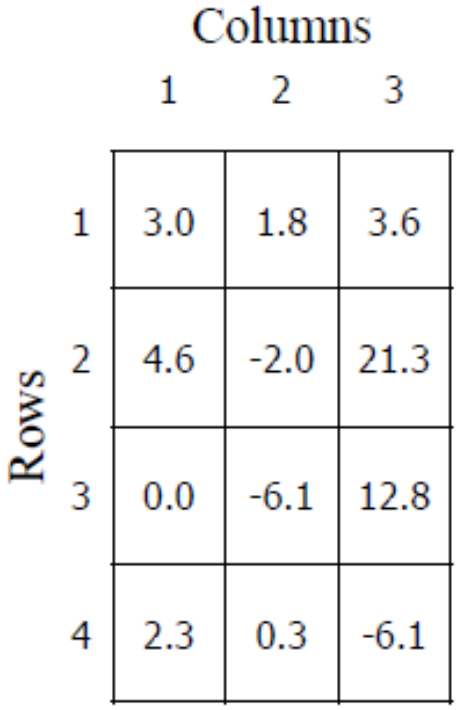
>> m(4,1)

ans = 2.3

>> m(end,end-1)

ans = 12.8

- Check some others



Changing Values in an Array

- You can change a value in an element in an array with indexing:

```
>> A (2) = 5
```

A = 12 5 -3

You can extend an array by defining a new element:

```
>> A (6) = 8
```

A = 12 5 -3 0 0 8

– Notice how undefined values of the array are filled with zeros

Creating Evenly Spaced Arrays

- Colon "**:**" notation can be used to define evenly spaced (potentially large) arrays
- Format is **first:increment:last**
- Examples:

```
>> H = 1:1:6
H = 1 2 3 4 5 6
>> I = 1:2:11
I = 1 3 5 7 9 11
>> J = 18:-3:3
J = 18 15 12 9 6 3
>> K = [1:1:10 ; 3:3:30]
K = 1  2  3  4  5  6  7  8  9  10
    3  6  9 12 15 18 21 24 27 30
```

Extracting Data with the Colon Operator

- The colon by itself represents an entire row or column when used in as an array index in place of a particular number.

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

- >> G (: , 1)
- >> G (: , 3)
- >> G (2, :)

```
     1
ans = 4
     7
     3
ans=6
     9
ans =4    5    6
```

Extracting Data with the Colon Operator Continued

- The colon operator can also be used to extract a range of rows or columns:

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

```
>> G (2:3, :)

G =
     4     5     6
     7     8     9
```

```
>> G (1, 2:3)

ans =
     2     3
```

Manipulating Arrays

- The transpose operator, an apostrophe (‘), changes all of an array’s rows to columns and columns to rows.

```
>> K = [1   3   7]

K =
     1     3     7
```

```
>> K'

ans =
     1
     3
     7
```

Hands-on exercise:

- Create the following matrix using colon notation:

```
W =
     1     2     3     4     5
    10    12    14    16    18
     6     5     4     3     2
```

- All three rows are evenly spaced

– The first row ranges from 1 to 5 in increments of 1

1:5

– The second row ranges from 10 to 18 in increments of 2

10:2:18

– The third row ranges from 6 to 2 in increments of -1 **6:-1:2**

– All together:

– **>> W = [1:5 ; 10:2:18 ; 6:-1:2]**

Manipulating Matrices Continued

- The functions fliplr() and flipud() flip a matrix left-to- right and top-to-bottom, respectively.

– Experiment with these functions to see how they work.

Results of fliplr and flipud

- >> wlr = fliplr(W)
- >> wup = flipud(W)

wlr =

5	4	3	2	1
18	16	14	12	10
2	3	4	5	6

Flipped left to right

wup =

6	5	4	3	2
10	12	14	16	18
1	2	3	4	5

Flipped top to bottom

Hands-on continued

- Create the following matrix using colon notation:

```
X =
   1.2   2.3   3.4   4.5   5.6
   1.9   3.8   5.7   7.6   9.5
     0    -3    -6    -9   -12
```

- Transpose this matrix and assign it to variable **Y**.

```
>> Y = x'
```

- Extract the 2nd row from Y and assign it to variable **Z**.

```
>> Z = Y(2,:)
```

Function to extract information about matrices/vectors

- length() returns the length of a 1D row/column vector

>> A = 1:20;

>> L = length(A) RETURNS--> L = 20

- size() returns **number** of rows and columns

$$A = \begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{matrix}$$

```
>> L = size(A) RETURNS--> L = 2   3
>> [row cols] = size(A) RETURNS--> rows = 2; cols = 3
```

- numel() returns **total number** of elements

>> q = numel(A) RETURNS--> q = 6

Review of Some Key MATLAB Bits

- ";" at the end of a command suppresses showing of the results
- "%" is used to put in a comment
- "…" is used to continue a line
- "ctrl-c" is used to stop executing commands
- "format long" is used to display numbers in higher
- precision (15 – 16 significant figures)
- "format short" is standard 4 decimal place display (stored as full 15-16 significant figures)

Array Review Examples

- Using colon

```
>> x = 1:15 % 15 x 1 row vector
>> x = (1:15)' % the transpose ‘ gives a 1 x 15 column vector
>> x = 1:0.5:15 % incremented by 0.5
```

- Using “linspace” command

```
>> x = linspace(1,15,30)
% it generates a row vector of 30 linearly equally spaced points from 1 to 15
```

- Using delimiters（分隔符）

```
>> A = [1 2 3; 4 5 6; 7 8 9]
>> B = [(1:3); (4:6); (7:9)]
>> C = [1:3; 4:6; 7:9]
% 3 routes to the same result
```

More Array Review Examples

- >> a = (3:8)

% row array – same as a = [3:8] and a = 3:8

- >> b = a’

% converted to column array using transpose operator

- >> c = [2;4;6;8]

% another column array

- >> d = [a a]

% creates a 1 x 12 array from two 1 x 6 arrays

- >> e = [a' a’]

% creates a 6 x 2 array from two 6 x 1 arrays

- >> f = [a; a]

% creates a 2 x 6 array by stacking two 1 x 6 arrays

- >> g = 3.*f-2

% element-by-element multiplication & subtraction

- >> h = e.^2

% element-by-element squaring

- >> k = a.*a

% element-by-element multiplication

- >> l = a./a

% element-by-element division

Array Indexing Review

- An index for an array starts from 1 (not 0).
- An index number must be given with parenthesis.
- For A=[1 2 3; 4 5 6; 7 8 9]
  - A(3, 2) is 8
  - A(:, 1) is 1, 4, and 7 – column array
  - A(2, 2:3) is 5 and 6 – row array
  - A(end, 2) is 8 – 2nd column or the last row
  - A(8) is 6 – 8th element of the array – counts down the columns – try it with some other values from 1 to 9
- Row or column can be deleted as follows:
  - A(2, :) = [] gives you A = [1 2 3; 7 8 9]
  - A(:, 3) = [] gives you A = [1 2; 7 8]
  - A(end, 2) gives you 8
  - A(:) stretching A into one column array