

UNIT – I

History of MATLAB

1. MATLAB was not a programming language but was a simple interactive calculator.
2. 1970 there was EISPACK (Matrix Eigen System PACKAGE) and LINPACK (LINEar Equation PACKAGE) which was invented and was invented in FORTRAN.
3. After invention of these 2 packages the MATLAB was invented in late of 1970's by **CLEVE MOLER**, he was working in computer science department at university of New Mexico.
4. After this he tried to develop MATix LABoratory (Software Libraries for numerical Computing using FORTRAN).
5. Cleve Moler with Jack Little and Steve Bangert worked in MATLAB using C and founded MathWorks.
6. In 1984 rewrote to MATLAB using C and the software libraries were known as JACKPACK and LINPACK.
7. In every 6 months they launch new version and updates.

Features of MATLAB.

1. MATLAB is a high-level language.
 - a. Study Data Structures
 - b. Control Flow Statements
 - c. Object Oriented Programming
 - d. Create and Solve Complex and large application.
2. MATLAB provides interactive environment
 - a. MATLAB allows interactive exploration, design and problem solving.
 - b. It consists of bunch of toolboxes.
 - c. It also consists of tools for development, handling, debugging, and profiling files.
3. Handling Graphics
 - a. It offers built in graphics
 - b. Tools for generating customized plots
 - c. Data visualization
 - d. 2D and 3D animations
 - e. Image Processing
 - f. Graphical Representation
4. Accessing Data
 - a. Supports sensor, video, image, telemetry, binary, and various real time data.
 - i. JDBC/ODBC Databases
 - b. Can read data from csv files
5. Application Program Interface (API)
6. Toolboxes
 - a. There are many toolboxes in MATLAB depending what kind of work we do.
 - b. Image Processing Toolbox
 - c. Aerospace Toolbox
 - d. Deep Learning
 - e. Simulink
7. Large libraries of mathematical functions
 - a. Integration
 - b. Trigonometric Functions

- c. Logical Functions
- 8. Interaction with other languages
- 9. Simulink
 - a. Designing Based Library Package
 - b. Design Control System power system etc
- 10. Interface with other languages.

Advantages of MATLAB

- 1. Has Easy User Interface
- 2. Various types of inbuilt functions / libraries
- 3. Predefined Algorithms
- 4. Data Visualization
- 5. Debugging of codes
- 6. Huge Committee of MATLAB
- 7. Platform Independent

Disadvantages of MATLAB

- 1. Very Expensive
- 2. All the errors are not much informative
- 3. Cross-Compilation of Languages is difficult
- 4. It needs fast computers

MATLAB as a good programming language

- 1. Use various types of variables / codes / tables / inbuilt functions / inbuilt libraries etc.
- 2. Use of Descriptive variable names
- 3. Write own functions and can-do things over again
- 4. Write our own scripts
- 5. Indenting (if else loop spacing)
- 6. Combine 2 or more codes simultaneously

MATLAB

Lab1

BTech (CSBS) -Semester VII

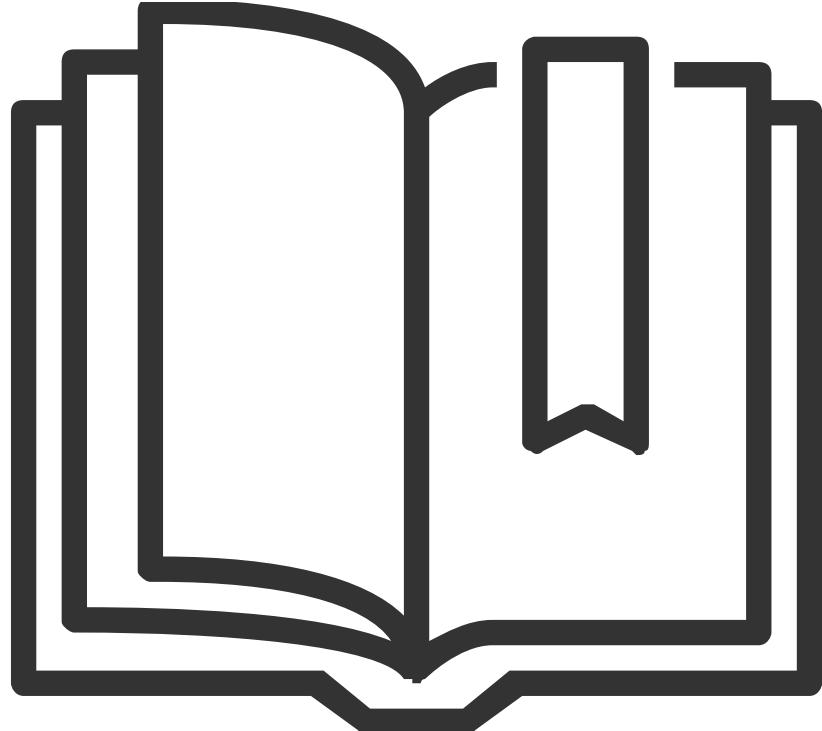
12 July 2022, 10:45AM



EXPERIMENT 1:Basic Mathematics using MATLAB

Compute the following quantities at command prompt:

- 1) $2^5/(2^5-1)$ and compare with $[1-(1/2^5)]-1$
- 2) $[3(\sqrt{5}-1)/(\sqrt{5}+1)*2] - 1$
- 3) Area = πr^2 with $r=\pi^{1/3}-1$
- 4) $e^3, \ln(e^3), \log_{10}(e^3), \log_{10}(10^5)$
- 5) $e^{\pi\sqrt{163}}, \sin(2\pi/6)+\cos(2\pi/6)$
- 6) $y= \cosh 2x - \sinh 2x$; where $x=32\pi$
- 7) Solve $3^x=17$ for x
- 8) $1+(3i/(1-3i)), e^{i\pi/4}$
- 9) Execute the commands $\exp(pi/2*i)$ and $\exp(pi/2i)$.
- 10) $\cot(0), \tan^{-1}(\infty)$



**Thank you for
listening**

09:30AM

Image and Video Processing



SVKM'S
NMIMS
Deemed to be UNIVERSITY

NAVI MUMBAI

MATLAB

Unit 1-Lecture 9

BTech (CSBS) -Semester VII

9 August 2022, 09:35AM



Introduction to MATLAB

- History,
- basic features,
- strengths and weaknesses,
- good programming practices
- plan your code



History

- The first MATLAB was not a programming language; it was a simple interactive matrix calculator.
- In 1970, a group of researchers developed EISPACK (Matrix Eigensystem Package) and LINPACK (Linear Equation Package) in FORTRAN
- The development of the MATLAB started in the late 1970s by Cleve Moler, the chairman of the Computer Science department at the University of New Mexico. Cleve wanted to make his students able to use LINPACK & EISPACK (software libraries for numerical computing, written in FORTRAN), and without learning FORTRAN.
- In 1984, Cleve Moler with Jack Little & Steve Bangert rewrote MATLAB in C and founded MathWorks. These libraries were known as JACKPAC at that time, later these were revised in 2000 for matrix manipulation and named as LAPACK (Linear Algebra Package)



Features of MATLAB

1. MATLAB is high-level language

This is a high-level programming language with data structures, control flow statements, functions, output/input, and object-oriented programming. It permits both, rapidly creating speedy throw-away programs, and creating complete, complex and large application programs.

2. Interactive Environment

MATLAB provides an interactive environment that allows iterative exploration, design, and problem-solving. It is a bunch of tools that a programmer can use. It includes abilities for handling the variables in the workspace & importing/exporting data. It also contains tools for development, handling, debugging, and profiling MATLAB files.



Features of MATLAB

3. Handling Graphics

It offers built-in graphics useful for data visualization, and tools for generating custom plots. MATLAB holds high-level instructions specially for creating two and three-dimensional data visualizations, animations, image processing, and graphical presentation. It allows users to modify graphics through GUI

4. Accessing data

MATLAB can natively support the sensor, video, image, telemetry, binary, and various real-time data from JDBC/ODBC databases. Reading data from different databases, CSV, audio, images, and video is super simple from an integrated environment.



Features of MATLAB

3. Application Program Interface (API)

MATLAB APIs allow users to write C / C++ and Fortran programs that directly interact with MATLAB. These include options for calling programs from MATLAB (dynamic linking), reading and writing MAT-files and using MATLAB as an interface to run applications.

6. Toolboxes

A "Toolbox" is a set of functions designed for a specific purpose and compiled as a package. These Toolboxes include MATLAB code, apps, data, examples and the documentation which helps users to utilize each Toolbox. There are separate Toolboxes available from Mathworks, to be used for specific purposes, for example, text analytics, image processing, signal processing, deep learning, statistic & machine learning, and many more.



Features of MATLAB

7. A large library of Mathematical functions

MATLAB has a huge inbuilt library of functions required for mathematical analysis of any data. It has common math functions like sqrt, factorial etc. It has functions required for statistical analysis like median, mode and std (to find standard deviation), and much more. MATLAB also has functions for signal processing like filter, butter(Butterworth filter design) audio read, Conv, xcorr, fft, fftshift etc. It also supports image processing and some common functions required for image processing in MATLAB are rgb2gray, rgb2hsv, adaptthresh etc.



Features of MATLAB

8. MATLAB and Simulink:

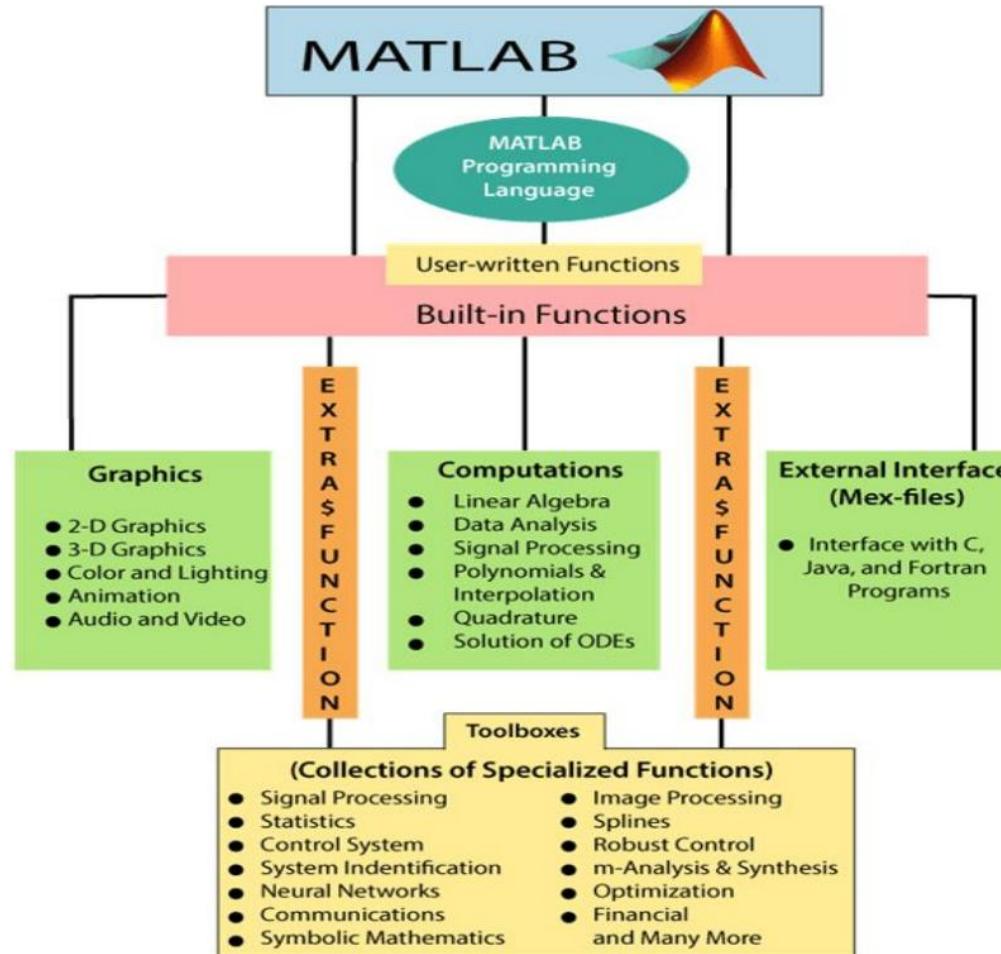
MATLAB has an inbuilt feature of Simulink wherein we can model the control systems and see their real-time behavior. We can design any system either using code or building blocks and see their real-time working through various inbuilt tools. It has lucid examples of basic control systems and their working.

9. Interface with other languages:

We can write a set of codes (libraries) in languages like PERL and JAVA, and we can call those libraries from within the MATLAB itself. MATLAB also supports ActiveX and .NET libraries.



Schematic of MATLAB



MATLAB

Unit 1-Lecture 10

BTech (CSBS) -Semester VII

12 August 2022, 09:35AM



Introduction to MATLAB

- History,
- basic features,
- strengths and weaknesses,
- good programming practices
- plan your code



Advantages of MATLAB

- **Easy to use interface:** A user-friendly interface with features you want to use is one click away.
- **A large inbuilt database of algorithms:** MATLAB has numerous important algorithms you want to use already built-in, and you just have to call them in your code.
- **Extensive data visualization and processing:** We can process a large amount of data in MATLAB and visualize them using plots and figures.



Advantages of MATLAB

- **Debugging of codes easy:** There are many inbuilt tools like analyzer and debugger for analysis and debugging of codes written in MATLAB.
- **Huge community:** It has huge community support where many of the questions will be answered
- **Platform-independent:** MATLAB is platform independent and hence it can be installed on different Operating Systems such as Windows, Vista, Linux and Macintosh.



Disadvantages of MATLAB

- Sometimes, the error messages are not much informative, so you have to figure out the error yourself.
- Matlab is more expensive. The license is very costly, and users need to buy each and every module and need to pay for the same.
- Cross-compiling of Matlab code to other languages is very difficult and requires deep Matlab knowledge to deal with errors produced.



Disadvantages of MATLAB

- Matlab is used mainly for scientific research and is not suitable for development activities that are user-specific.
- Matlab is an interpreted language; thus, it can be very slow. Poor programming practices can contribute to making Matlab unacceptably slow.
- It requires fast computer with sufficient amount of memory. This adds to the cost for individuals willing to use it for programming.



MATLAB good programming practices

- Use variables instead of hard coded numbers. Put these numbers at the top of your scripts and functions
- Write functions for things you do over and over again
- Use descriptive variable names
- Put in comments to describe tricky parts of your code
- Document your functions

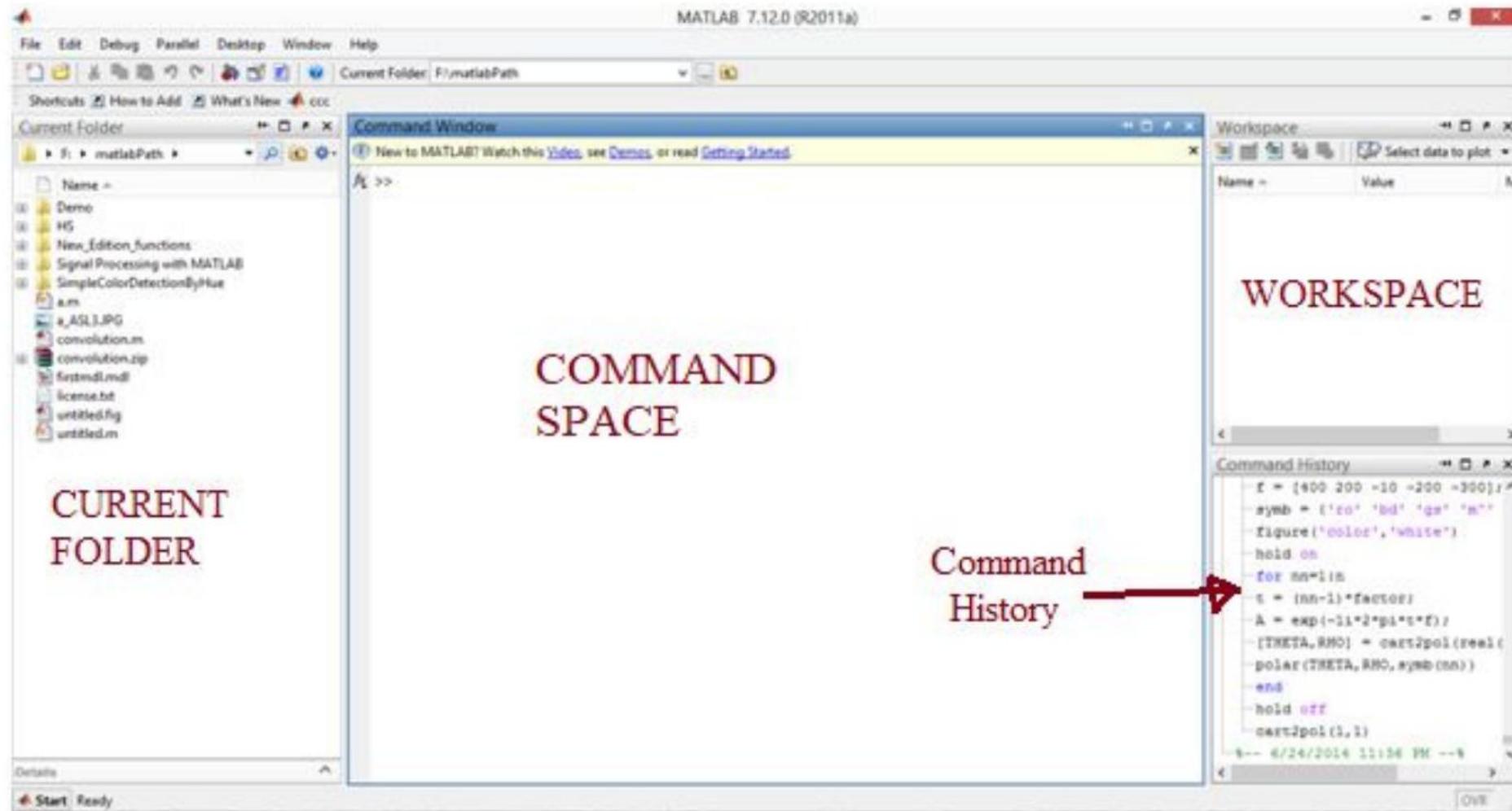


MATLAB good programming practices

- Use MATLAB built-in functions when they are available.
- Learn how to use structures and cell arrays well
- Check your code
- Learn how to use the MATLAB debugger functions.
- Write scripts for each figure you need to make.
- Always indent the body of an if construct by two or more spaces to improve the readability of the code



MATLAB good programming practices

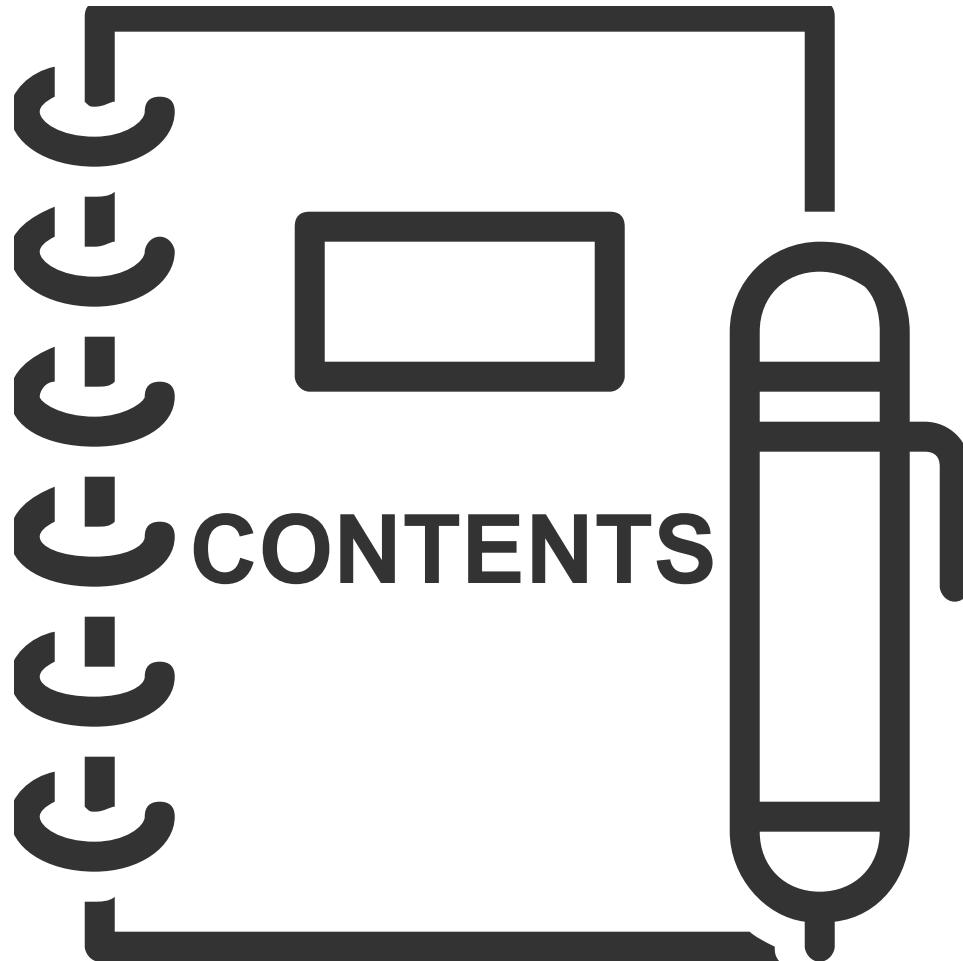


MATLAB

Unit 2-Lecture 1

BTech (CSBS) -Semester VII

12 July 2022, 09:35AM



Teaching plan



Assessment analysis



Text/Reference books



Unit 1-Lecture 1



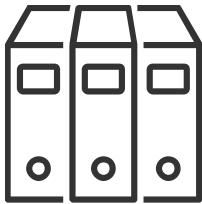
Teaching plan/Assessment analysis

Teaching plan

Teaching Scheme				Evaluation Scheme	
Lecture (Hours/week)	Practical (Hours/week)	Tutorial (Hours/week)	Credit	Internal Continuous Assessment (ICA) As per Institute Norms (50 Marks)	Theory (3 Hrs, 100 Marks)
2	2	0	3	Marks Scaled to 50	Marks Scaled to 50

Assessment analysis

Assessment Component	ICA (100 Marks) (Marks scaled to 50)					TEE (100 marks) (Marks scaled to 50)
	Lab Performance	Lab Exam and Viva	Research activity (beyond syllabus)	Class Test 1 and Class Test 2	Class Partition	
Weightage	10%	5%	10%	20%	5%	50%
Marks	20	10	20	20+20	10	100
Date/week of activity	Weekly	10 th and 11 th week	14 th week	T1: 16-23 August, 2022 T2: 10-15 October, 2022	2 nd and 13 th Week	16 th Nov.2022 to 2 nd Dec., 2022



Text/Reference books

Text Books:

- 1) Rafael C. Gonzalez, Richard E. Woods, Steven Eddins, “*Digital Image Processing using MATLAB*”, Pearson Education, Inc., Second Edition, 2004.
- 2) Stormy Attaway, Butterworth-Heinemann, “*MATLAB: A Practical Introduction to Programming and Problem Solving*”, Butterworth-Heinemann is an imprint of Elsevier, Fourth Edition, 2017.

Reference Books:

- 1) Cleve Moler, “*Experiments with Matlab*”, MathWorks, Inc., 2011.



Unit 1-Lecture 1→Agenda

- 1) Desktop Basics
- 2) Numbers & Arithmetic Operations
- 3) Workspace Variables



1: Introduction

When you start MATLAB, the desktop appears in its default layout.

- The desktop includes these panels:
- **Current Folder** — Access your files.
- **Command Window** — Enter commands at the command line, indicated by the prompt (`>>`).
- **Workspace** — Explore data that you create or import from files.
- **Command History** — View or rerun commands that you entered at the command line.

Command Window

- type commands

Current Directory

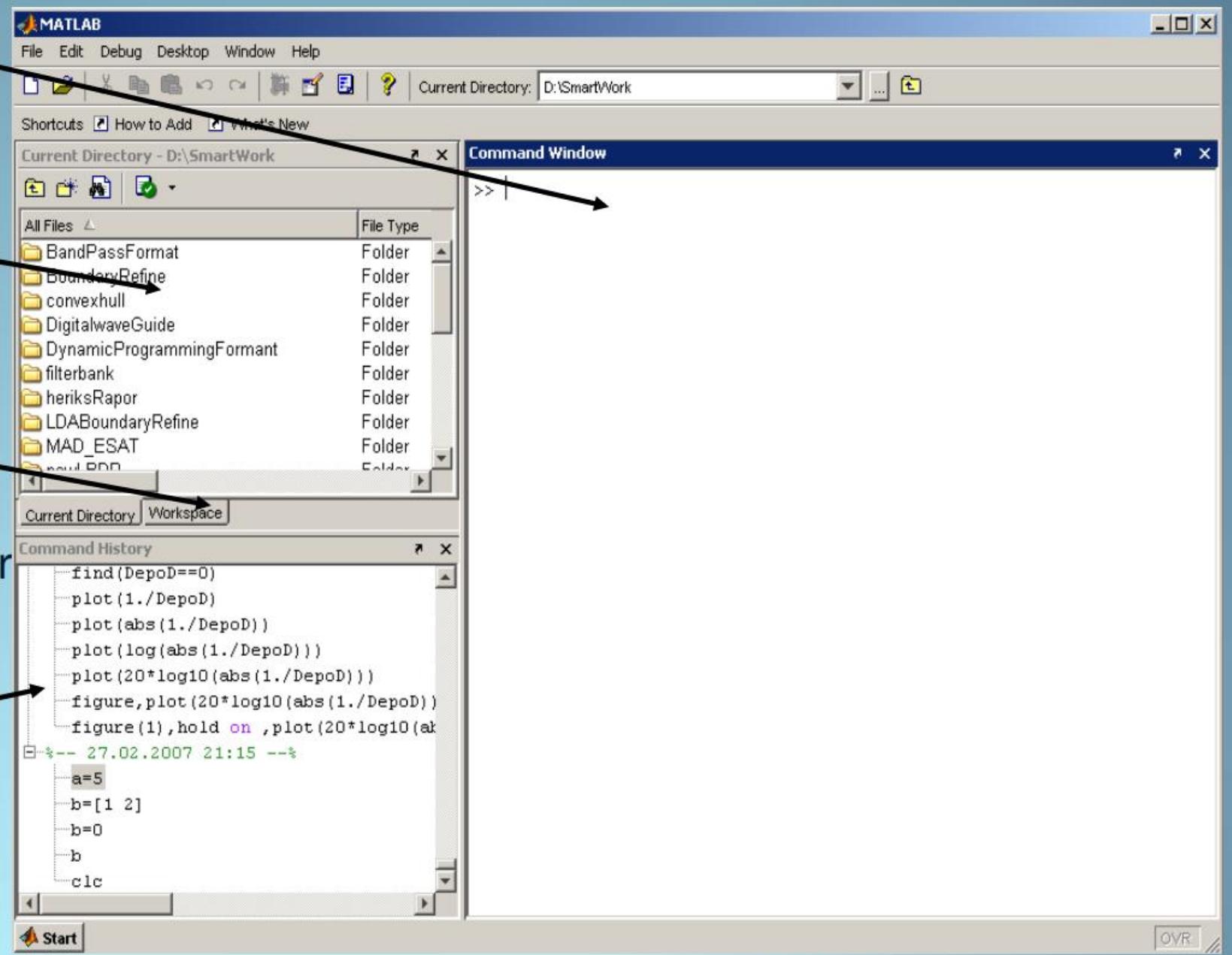
- View folders and m-files

Workspace

- View program variables
- Double click on a variable
- to see it in the Array Editor

Command History

- view past commands
- save a whole session using diary





MATLAB window

- Figure Window -contains output from graphic commands
- Help Window -provides help information
- Editor Window -creates and debugs script and function files
- Current directory Window -shows files in current directory
- Launch Pad Window -provides access to tools,demos and documentation



Command window

>> type code

Press enter

Command executed and output displayed

semicolon (;

output not displayed

Ellipsis(...) if a command is too long to fit in one line

Command can continue line after line up to 4096 characters.



common commands

Matlab commands **case sensitive**

- % -comment
- clc -clear screen
- ↑ -recall previously typed commands
- ↓ -move down to previously typed



Arithmetic Operations With Scalars

<u>Operation</u>	<u>Symbol</u>	<u>Example</u>
Addition	+	$5+3$
Subtraction	-	$5-3$
Multiplication	*	$5*3$
Right division	/	$5/3$
Left division	\	$5\backslash 3=3/5$
Exponentiation	^	$5^3=125$



Order of Precedence

Parentheses

Exponentiation

Multiplication and division

Addition and subtraction



Display Formats

Command	Description
format short	Fixed-point with 4 decimal digits
format long	Fixed-point with 14 decimal digits
format bank	2 decimal digits
format compact	Eliminates empty lines
format loose	Adds empty lines



Elementary Math functions

Function	Description
<code>sqrt (x)</code>	Square root
<code>exp (x)</code>	Exponential (e^x)
<code>abs (x)</code>	Absolute value
<code>log (x)</code>	Natural logarithm Base e logarithm
<code>Log10(x)</code>	Base 10 logarithm
<code>factorial(x)</code>	Factorial function $x!$



Trigonometric/rounding Math functions

$\sin(x), \cos(x),$
 $\tan(x), \cot(x)$

Rounding function

Function	Description
round(x)	Round to the nearest integer
fix(x)	Round towards zero
ceil(x)	Round towards infinity
floor(x)	Round towards minus infinity
rem(x,y)	Returns remainder after x is divided by y



Trigonometric/rounding Math functions

Elementary functions (sin, cos, sqrt, abs, exp, log10, round)
-type **help elfun**

Advanced functions (bessel, beta, gamma, erf)
-type **help specfun**
-type **help elmat**



Defining scalar variables

variable is a name made of a letter or a combination of several letters that is assigned a numerical value

- actually name of a memory location
- assignment operator “=”

eg.

`>>x=15`

`>>x=3*x-12`

When new variable is created matlab assigns appropriate memory space where assigned value can be stored

- When variable is used stored data is used
- If assigned new value content of memory is replaced

eg. `>>ABB=72;`

`>>ABB=9;`

`>>ABB`

`ABB=`



Rules about variable names

- Variable names are case sensitive.
- Variable names can contain up to 63 characters (as of MATLAB 6.5 and newer).
- Variable names must start with a letter followed by letters, digits, and underscores.
- Must begin with a letter.
- Avoid using names of built-in functions for variable.

MATLAB

Unit 2-Lecture 2

BTech (CSBS) -Semester VII

15 July 2022, 09:35AM



Predefined variables

variable	description
ans	Value of last expression
eps	Smallest difference between 2 numbers
i	$\sqrt{-1}$
inf	Infinity
j	Same as i
NaN	Not a number
pi	The number π



Some Useful MATLAB commands

- `who` List known variables
- `whos` List known variables plus their size
- `help` `>> help sqrt` (Help on using sqrt)
- `clear` Clear all variables from work space
- `clear x y` Clear variables x and y from work space
- `clc` Clear the command window



Variable and assignment statement

variable name = expression

Command window:

```
>> mynum = 6
mynum =
    6
>>
```

Correction: The variable name must always be written on left, and expression on right.

Now write in Command window:

```
>> 6 = mynum
    6 = mynum
    |
```

Error: The expression to the left of the equals sign is not a valid target for an assignment.

```
>>
```



Initializing, incrementing & decrementing

Frequently, values of variables change, as shown previously. Putting the first or initial value in a variable is called *initializing* the variable.

Adding to a variable is called *incrementing*. For example, the statement

```
mynum = mynum + 1
```

increments the variable *mynum* by 1.

Similarly, ***mynum=mynum-1***, will be *decrementing* variable.



Floating-Point Numbers

For floating point number there are two basic types:

- Double-Precision Floating Point
- Single-Precision Floating Point

The integer type are **int8**, **int16**, **int32**, **int64**.

These integers represent the **bits** used to store the value of data type.

type **char** is used to store the **character or string** eg. ‘cat’

type **logical** is used to store true/false.



Numerical Expression

Expressions can be created using values, variables that have already been created, operators, built-in functions, and parentheses. For numbers, these can include operators such as multiplication and functions such as trigonometric functions. An example of such an expression is:

```
>> 2 * sin(1.4)  
ans =  
1.9709
```



Fromat Command

This will remain in effect until the format is changed back to **short**, as demonstrated in the following.

```
>> format long  
>> 2 * sin(1.4)  
ans =  
1.970899459976920
```

```
>> format short  
>> 2 * sin(1.4)  
ans =  
1.9709
```



Fromat Command

The **format** command can also be used to control the spacing between the MATLAB command or expression and the result; it can be either **loose** (the default) or **compact**.

```
>> format loose
```

```
>> 5*33
```

```
ans =
```

```
165
```

```
>> format compact
```

```
>> 5*33
```

```
ans =
```

```
165
```

```
>>
```



Nested Parentheses

Within a given precedence level, the expressions are evaluated from left to right (this is called *associativity*).

For the operators that have been covered thus far, the following is the precedence (from the highest to the lowest):

()	parentheses
^	exponentiation
-	negation
*, /, \	all multiplication and division
+, -	addition and subtraction



Operator precedence rule:

Operators	Precedence
Parentheses: ()	Highest
Power ^	
Unary: Negation (-), not (~)	
Multiplication, division *, /, \	
Addition, subtraction +, -	
Relational <, <=, >, >=, ==, ~=	
And &&	
Or	
Assignment =	Lowest



Practice problem:

1. Think about what the results would be for the following expressions, and then type them in to verify your answers:

$1 \backslash 2$
 $-5 ^ 2$
 $(-5) ^ 2$
 $10 - 6 / 2$
 $5 * 4 / 2 * 3$

2. What would happen if you use the name of a function , eg abs, as a variable name?
3. Use plus operator and check the results.

Also, if a function name is typed incorrectly, MATLAB will suggest a correct name.

```
>> abso(-4)
Undefined function or variable 'abso'.
Did you mean:
>> abs(-4)
```



Constant/random number

pi 3.14159...

i $\sqrt{-1}$

j $\sqrt{-1}$

inf infinity ∞

NaN stands for "not a number," such as the result of 0/0

Practice problem:

```
>> rand  
ans =  
    0.8147  
>> rand  
ans =  
    0.9058
```

Generate a random

- real number in the range [0,1]
- real number in the range [0, 100]
- real number in the range [20, 35]
- integer in the inclusive range from 1 to 100
- integer in the inclusive range from 20 to 35



Relational Expression

Expressions that are conceptually either true or false are called *relational expressions*; they are also sometimes called *Boolean expressions* or *logical expressions*. These expressions can use both *relational operators*, which relate two expressions of compatible types, and *logical operators*, which operate on logical operands.

The relational operators in MATLAB are:

Operator	Meaning
>	greater than
<	less than
\geq	greater than or equals
\leq	less than or equals
==	equality
~=	inequality

Example:

```
>> 3 < 5
ans =
    1
```

```
>> 2 > 9
ans =
    0
>> class(ans)
ans =
logical
```



Practice question:

1. Assume that there is variable x that has been initialized, what would be the value of expression $3 < x < 5$, if the value of x is 4? what if the value is 7?



Practice question:

Think about what would be produced by the following expressions, and then type them in to verify your answers.

`3 == 5 + 2`

`'b' < 'a' + 1`

`10 > 5 + 2`

`(10 > 5) + 2`

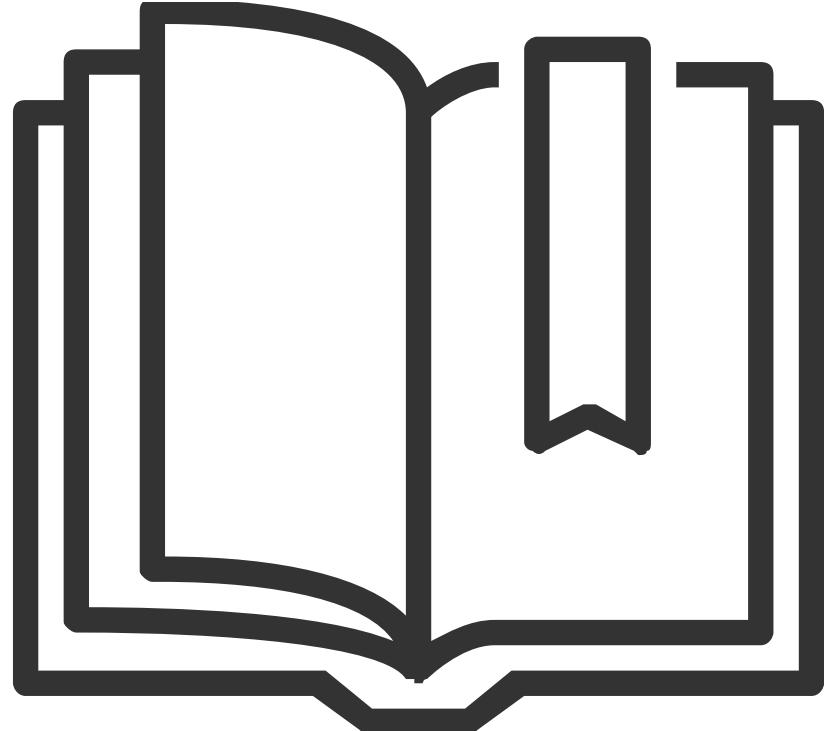
`'c' == 'd' - 1 && 2 < 4`

`'c' == 'd' - 1 || 2 > 4`

`xor('c' == 'd' - 1, 2 > 4)`

`xor('c' == 'd' - 1, 2 < 4)`

`10 > 5 > 2`



**Thank you for
listening**

10:45AM

IT Workshop/MATLAB



SVKM'S
NMIMS
Deemed to be UNIVERSITY

NAVI MUMBAI

MATLAB

Unit 2-Lecture 3

BTech (CSBS) -Semester VII

19 July 2022, 09:35AM



Creating 2D array(matrix)

Variable_name = [1st row elements; 2nd row elements;.....;last row elements]

- Matrix are used in science & eng to describe many physical quantities.
- All rows must have same number of elements



Zeros , ones and eye commands

- **zeros(m,n)**: mxn matrix of 0's.
- **ones(m,n)**: mxn matrix of 1's.
- **eye(n)**: nxn identity matrix



Array

List of numbers arranged in row and/or columns.

Simplest array

-1D array

-usually to represent vectors.

Complex array

-2D array

-represent matrixes



Creating vector from a known list of numbers

```
Variable_name = [type vector elements]
```

Row vector :- type elements with space or comma

Column vector :- type elements with semicolon (;) or press Enter key after each element

```
Variable_name = [m:q:n] or Variable_name = m:q:n
```

First term spacing last term

```
Variable_name = linspace(xi,xf,n)
```

First element last element no of elements (when omitted default value 100)



Vector and Matrix

5

3
7
4

5	88	3	11
---	----	---	----

9	6	3
5	7	2

The scalar is 1×1 , the column vector is 3×1 (three rows by one column), the row vector is 1×4 (one row by four columns), and the matrix is 2×3 (two rows by three columns). All of the values stored in these matrices are stored in what are called *elements*.



Creating row vector

There are several ways to create row vector variables. The most direct way is to put the values that you want in the vector in square brackets, separated by either spaces or commas. For example, both of these assignment statements create the same vector v :

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

```
v =
```

```
1 2 3 4
```

```
>> v = [1, 2, 3, 4]
```

```
v =
```

```
1 2 3 4
```



Colon Operator

If, as in the preceding examples, the values in the vector are regularly spaced, the *colon operator* can be used to *iterate* through these values. For example, `2:6` results in all of the integers from 2 to 6 inclusive:

```
>> vec = 2:6
vec =
2 3 4 5 6
```

In this vector, there are five elements; the vector is a 1×5 row vector. Note that in this case, the brackets `[]` are not necessary to define the vector.

With the colon operator, a *step value* can also be specified by using another colon, in the form `(first:step:last)`. For example, to create a vector with all integers from 1 to 9 in steps of 2:

```
>> nv = 1:2:9
nv =
1 3 5 7 9
```



Line space function

The **linspace** function creates a linearly spaced vector; **linspace(x,y,n)** creates a vector with n values in the inclusive range of x to y . If n is omitted, the default is 100 points. For example, the following creates a vector with five values linearly spaced between 3 and 15, including the 3 and 15:

```
>> ls = linspace(3,15,5)  
ls =  
    3     6     9    12    15
```



Concatenating Vector

Vector variables can also be created using existing variables. For example, a new vector is created here consisting first of all of the values from *nv* followed by all values from *ls*:

```
>> newvec = [nv ls]
newvec =
    1  3  5  7  9  3  6  9  12  15
```

Putting two vectors together like this to create a new one is called *concatenating* the vectors.



Logspace function

Similarly, the **logspace** function creates a logarithmically spaced vector; **logspace(x,y,n)** creates a vector with n values in the inclusive range from 10^x to 10^y . If n is omitted, the default is 50 points. For example, **logspace(1,4,4)** creates a vector with four elements, logarithmically spaced between 10^1 and 10^4 , or in other words 10^1 , 10^2 , 10^3 , and 10^4 .

```
>> logspace(1, 4, 4)
ans =
    10      100     1000    10000
```



SVKM'S
NMIMS
Deemed to be UNIVERSITY

NAVI MUMBAI

MATLAB

Unit 2-Lecture 4

BTech (CSBS) -Semester VII

22 July 2022, 09:35AM



Workspace

- Managing the workspace
- Keeping track of your work session,
- Entering multiple statements per line



Workspace Browser

- The Workspace browser enables you to view and interactively manage the contents of the workspace in MATLAB®.
- For each variable or object in the workspace, the Workspace browser also can display statistics, when relevant, such as the minimum, maximum, and mean.
- You can edit the contents of scalar (1-by-1) variables directly in the Workspace browser. **Right-click the variable and select Edit Value.**
- To edit other variables, double-click the variable name in the Workspace browser to open it in the Variables editor.



Workspace Browser

Name	Value	Class	Min	Max	Mean
A	4x4 double	double	1	16	8.5000
B	[1;2;3;4]	double	1	4	2.5000
filename	'myfile.txt'	char			
patient	1x1 struct	struct			
t	'Hello'	char			
val1	2x3 cell	cell			
val2	[17,21,42]	double	17	42	26.6667
x	325	double	325	325	325
y	[9900,26025,39600]	uint32	9900	39600	
z	-Inf	double	-Inf	-Inf	-Inf



Open the Workspace Browser

To open the Workspace browser if it is not currently visible, do one of the following:

- MATLAB Toolstrip: On the **Home** tab, in the **Environment** section, click **Layout**. Then, in the **Show** section, select **Workspace**.
- MATLAB command prompt: Enter **workspace**.

You also can minimize the Workspace browser by collapsing the panel in which it resides. eg. if the Workspace browser is in the left side panel, click the button at the bottom left corner of the panel to collapse the panel. To restore the panel, click the button. If the Workspace browser is in the left or right side panel and the panel contains multiple tools, you also can minimize it by clicking the button to the left of the Workspace browser title bar.



Workspace Variables

Functions

<code>load</code>	Load variables from file into workspace
<code>save</code>	Save workspace variables to file
<code>matfile</code>	Access and change variables in MAT-file without loading file into memory
<code>disp</code>	Display value of variable
<code>formattedDisplayText</code>	Capture display output as string
<code>who</code>	List variables in workspace
<code>whos</code>	List variables in workspace, with sizes and types
<code>clear</code>	Remove items from workspace, freeing up system memory
<code>clearvars</code>	Clear variables from memory
<code>openvar</code>	Open workspace variable in Variables editor or other graphical editing tool
Workspace Browser	Open Workspace browser to manage workspace



Save Workspace Variables

There are several ways to save workspace variables interactively:

- To save all workspace variables to a MAT-file, on the **Home** tab, in the **Variable** section, click **Save Workspace**.
- To save a subset of your workspace variables to a MAT-file, select the variables in the Workspace browser, right-click, and then select **Save As**. You also can drag the selected variables from the Workspace browser to the Current Folder browser.
- To save variables to a MATLAB script, click the **Save Workspace** button or select the **Save As** option, and in the **Save As** window, set the **Save as type** option to **MATLAB Script**. Variables that cannot be saved to a script are saved to a MAT-file with the same name as that of the script.



Save Workspace Variables

You also can save workspace variables programmatically using the **save** function.

```
save('june10')
```

To save only variables A and B to the file june10.mat, use the command

```
save('june10', 'A' , 'B')
```



Load Workspace Variables

- To load saved variables from a MAT-file into your workspace, double-click the MAT-file in the Current Folder browser.
- To load a subset of variables from a MAT-file on the **Home** tab, in the **Variable** section, click **Import Data**. Select the MAT-file you want to load and click **Open**. You also can drag the desired variables from the Current Folder browser Details panel of the selected MAT-file to the Workspace browser. The Details panel is not available in MATLAB Online.
- To load variables saved to a MATLAB script into the workspace, simply run the script.



Load Workspace Variables

You also can load saved variables programmatically, use the load function.

```
load('durer')
```

To load variables X and map from the file durer.mat

```
load('durer', 'X', 'map')
```



Keep a track of work

Write to a Diary File

To keep an activity log of your MATLAB® session, use the **diary** function. **diary** creates a verbatim copy of your MATLAB session in a disk file (excluding graphics).



entering multiple statements per line

- **Enter Multiple Lines Without Running Them**
 - To enter multiple lines before running any of them, use **Shift+Enter** or **Shift+Return** after typing a line. This is useful, for example, when entering a set of statements containing keywords, such as `if ... end`. The cursor moves down to the next line, which does not show a prompt, where you can type the next line. Continue for more lines. Then press **Enter** or **Return** to run all of the lines.
 - This allows you to edit any of the lines you entered before you pressing **Enter** or **Return**.
- **Entering Multiple Functions in a Line**
 - To enter multiple functions on a single line, separate the functions with a **comma** (,) or **semicolon** (;). Using the semicolon instead of the comma will suppress the output for the command preceding it. For example, put three functions on one line to build a table of logarithms by typing

```
format short; x = (1:10)'; logs = [x log10(x)]
```

and then press **Enter** or **Return**. The functions run in **left-to-right order**.



entering multiple statements per line

- **Entering Long Statements**
- For items in single quotation marks, such as strings, you must complete the string in the line on which it was started. For example, completing a string as shown here

```
headers = ['Author Last Name, Author First Name, ' ...
'Author Middle Initial']  
results in
```

```
headers =  
Author Last Name, Author First Name, Author Middle Initial
```

B.Tech (VII)- IT Workshop/MATLAB

Unit 2

Practise Question set

NOTE:

- Explore MATLAB Desktop environment before starting these experiments.
 - All the variables provided are case sensitive.
1. Create a variable *myage* and store your age in it. Subtract 2 from the value of the variable. Add 1 to the value of the variable. Observe the Workspace Window and Command History Window as you do this.
 2. Explain the difference between these two statements:

```
result = 9*2
result = 9*2;
```
 3. Use the built-in function **namelengthmax** to find out the maximum number of characters that you can have in an identifier name under your version of MATLAB.
 4. Create two variables to store a weight in pounds and ounces. Use **who** and **whos** to see the variables. Use **class** to see the types of the variables. Clear one of them and then use **who** and **whos** again.
 5. Explore the **format** command in more detail. Use **help format** to find options. Experiment with **format bank** to display dollar values.
 6. Find a **format** option that would result in the following output format:

```
>> 5/16 + 2/7
ans =
67/112
```

7. Think about what the results would be for the following expressions, and then type them in to verify your answers.

```
25 / 5 * 5
4 + 3 ^ 2
(4 + 3) ^ 2
3 \ 12 + 5
4 - 2 * 3
```

As the world becomes more “flat,” it is increasingly important for engineers and scientists to be able to work with colleagues in other parts of the world. Correct conversion of data from one system of units to another (e.g., from the metric system to the American system or vice versa) is critically important.

8. Create a variable *pounds* to store a weight in pounds. Convert this to kilograms and assign the result to a variable *kilos*. The conversion factor is 1 kg = 2.2 lb.

9. Create a variable *ftemp* to store a temperature in degrees Fahrenheit (F). Convert this to degrees Celsius (C) and store the result in a variable *ctemp*. The conversion factor is $C = (F - 32) \times 5/9$.
10. The following assignment statements either contain at least one error, or could be improved in some way. Assume that *radius* is a variable that has been initialized. First, identify the problem, and then fix and/or improve them:
- ```

33 = number
my variable = 11.11;
area = 3.14 * radius ^2;
x = 2 * 3.14 * radius;

```
11. Experiment with the functional form of some operators such as **plus**, **minus**, and **times**.
12. Generate a random
- real number in the range [0, 20]
  - real number in the range (20, 50)
  - integer in the inclusive range from 1 to 10
  - integer in the inclusive range from 0 to 10
  - integer in the inclusive range from 50 to 100
13. Get into a new Command Window, and type **rand** to get a random real number. Make a note of the number. Then exit MATLAB and repeat this, again making a note of the random number; it should be the same as before. Finally, exit MATLAB and again get into a new Command Window. This time, change the seed before generating a random number; it should be different.
14. What is the difference between *x* and 'x'?
17. Explain the difference between constants and variables.
18. What would be the result of the following expressions?
- ```

'b' >= 'c' - 1
3 == 2 + 1
(3 == 2) + 1
xor(5 < 6, 8 > 4)
10 > 5 > 2
result = 3 ^2 - 20;
0 <= result <= 10

```
19. Create two variables *x* and *y* and store numbers in them. Write an expression that would be **true** if the value of *x* is greater than 5 or if the value of *y* is less than 10, but not if both of those are **true**.
20. Use the equality operator to verify that $3*10^5$ is equal to 3e5.
21. In the ASCII character encoding, the letters of the alphabet are in order: 'a' comes before 'b' and also 'A' comes before 'B'. However, which comes first—lower or uppercase letters?
22. Are there equivalents to **intmin** and **intmax** for real number types? Use **help** to find out.

23. Use **intmin** and **intmax** to determine the range of values that can be stored in the types **uint32** and **uint64**.
24. Use the **cast** function to cast a variable to be the same type as another variable.
25. Use **help elfun** or experiment to answer the following questions:

Is **fix(3.5)** the same as **floor(3.5)**?
 Is **fix(3.4)** the same as **fix(-3.4)**?
 Is **fix(3.2)** the same as **floor(3.2)**?
 Is **fix(-3.2)** the same as **floor(-3.2)**?
 Is **fix(-3.2)** the same as **ceil(-3.2)**?

26. For what range of values is the function **round** equivalent to the function **floor**?
 For what range of values is the function **round** equivalent to the function **ceil**?
27. Use **help** to determine the difference between the **rem** and **mod** functions.
28. Find MATLAB expressions for the following

$$\begin{aligned} & \sqrt{19} \\ & 3^{1.2} \\ & \tan(\pi) \end{aligned}$$

29. Using only the integers 2 and 3, write as many expressions as you can that result in 9. Try to come up with at least 10 different expressions (Note: don't just change the order). Be creative! Make sure that you write them as MATLAB expressions. Use operators and/or built-in functions.
30. A vector can be represented by its rectangular coordinates x and y or by its polar coordinates r and θ . Theta is measured in radians. The relationship between them is given by the equations:

$$\begin{aligned} x &= r * \cos(\theta) \\ y &= r * \sin(\theta) \end{aligned}$$

Assign values for the polar coordinates to variables r and θ . Then, using these values, assign the corresponding rectangular coordinates to variables x and y .

31. In special relativity, the Lorentz factor is a number that describes the effect of speed on various physical properties when the speed is significantly relative to the speed of light. Mathematically, the Lorentz factor is given as:

$$\gamma = \frac{1}{\sqrt{1 - \frac{v^2}{c^2}}}$$

Use 3×10^8 m/s for the speed of light, c . Create variables for c and the speed v and from them a variable $lorentz$ for the Lorentz factor.

32. A company manufactures a part for which there is a desired weight. There is a tolerance of $N\%$, meaning that the range between minus and plus $N\%$ of the desired weight is acceptable. Create a variable that stores a weight and another variable for N (e.g., set it to 2). Create variables that store the minimum and maximum values in the acceptable range of weights for this part.

33. An environmental engineer has determined that the cost C of a containment tank will be based on the radius r of the tank:

$$C = \frac{32,430}{r} + 428\pi r$$

Create a variable for the radius, and then for the cost.

34. A chemical plant releases an amount A of pollutant into a stream. The maximum concentration C of the pollutant at a point which is at a distance x from the plant is:

$$C = \frac{A}{x} \sqrt{\frac{2}{\pi e}}$$

Create variables for the values of A and x , and then for C . Assume that the distance x is in meters. Experiment with different values for x .

35. The geometric mean g of n numbers x_i is defined as the n th root of the product of x_i :

$$g = \sqrt[n]{x_1 x_2 x_3 \dots x_n}$$

(This is useful, for example, in finding the average rate of return for an investment which is something you'd do in engineering economics.) If an investment returns 15% the first year, 50% the second, and 30% the third year, the average rate of return would be $(1.15 * 1.50 * 1.30)^{1/3}$. Compute this.

36. Use the **deg2rad** function to convert 180 degrees to radians.

MATLAB

Unit 3-Lecture 5

BTech (CSBS) -Semester VII

26 July 2022, 09:35AM



Unit 3

- Matrix
- Array
- Basic mathematical functions



An array is MATLAB's basic data structure

Can have any number of dimensions. Most common are

- vector - one dimension (a single row or column)
- matrix - two or more dimensions
- Scalar - matrices with only one row and one column.

Arrays can have numbers or letters



Creating Matrices

In MATLAB, a vector is created by assigning the elements of the vector to a variable. This can be done in several ways depending on the source of the information.

- Enter an explicit list of elements
- Load matrices from external data files
- Using built-in functions
- Using own functions in M-files

A matrix can be created in MATLAB by typing the elements (numbers) inside square brackets []

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



Creating Matrices

```
>> A = [2 -3 5; -1 4 5] % Note MATLAB displays column vector vertically
```

```
A=
```

```
2 -3 5
```

```
-1 4 5
```

```
>> x = [1 4 7] % Note MATLAB displays row vector horizontally
```

```
x=
```

```
4
```

```
1 4 7
```

```
>> x = [1; 4; 7] %Optional commas may be used between the elements.Type the semicolon (or  
press Enter) to move to the next row
```

```
x=
```

```
1
```

```
4
```

```
7
```



Creating Matrices

```
>> cd=6; e=3; h=4;  
  
>> Mat=[e cd*h cos(pi/3);h^2 sqrt(h*h/cd) 14]  
  
Mat =  
    3.0000    24.0000    0.5000  
  16.0000    1.6330   14.0000
```



Concatenation of Matrices

Command Window

```
>> a=[1 2;3 4];  
b=[4 6 ;8 9];  
A=[a,b]
```

Row wise concate

```
A =
```

1	2	4	6
3	4	8	9

```
>> B=[a;b]
```

```
B =
```

1	I	2
3		4
4		6
8		9

Column wise concate

```
fxt >> |
```



Colon operator

The colon operator can be used to create a vector with constant spacing

$$x = m:q:n$$

- m is first number
- n is last number
- q is difference between consecutive numbers

```
>>x=[1:2:10]
```

$x =$

1 3 5 7 9

If omit q , spacing is one

$$y = m:n$$

```
>>y=1:5
```

$y =$

1 2 3 4 5



Colon operator

```
>> x=1:5:50  
  
x =  
  
1 6 11 16 21 26 31 36 41 46  
  
>> 1:5:50  
  
ans =  
  
1 6 11 16 21 26 31 36 41 46
```



Question 1

How can you use the colon operator to generate the vector shown below?

9 7 5 3 1



linspace function

$v = \text{linspace}(x_i, x_f, n)$

- x_i is first number
- x_f is last number
- n is number of terms
(= 100 if omitted)

```
>> linspace (4,8,50)
ans =
Columns 1 through 11
4.0000    4.0816    4.1633    4.2449    4.3265    4.4082    4.4898    4.5714    4.6531    4.7347    4.8163
Columns 12 through 22
4.8980    4.9796    5.0612    5.1429    5.2245    5.3061    5.3878    5.4694    5.5510    5.6327    5.7143
Columns 23 through 33
5.7959    5.8776    5.9592    6.0408    6.1224    6.2041    6.2857    6.3673    6.4490    6.5306    6.6122
Columns 34 through 44
6.6939    6.7755    6.8571    6.9388    7.0204    7.1020    7.1837    7.2653    7.3469    7.4286    7.5102
Columns 45 through 50
7.5918    7.6735    7.7551    7.8367    7.9184    8.0000
```



Misc Matrix

- `zeros(r,c)` - makes matrix of r rows and c columns, all with zeros
- `ones(r,c)` - makes matrix of r rows and c columns, all with ones
- `rand(r,c)` - makes matrix of r rows and c columns, with random numbers
- `eye(n)` - makes square matrix of n rows and columns. Main diagonal (upper left to lower right) has ones, all other elements are zero
- `magic(n)` - makes a special square matrix of n rows and c columns, called Durer's matrix



Misc Matrix

```
>> a=zeros(4,3)
```

```
a =
```

0	0	0
0	0	0
0	0	0
0	0	0

```
>> c=rand(4,3)
```

```
c =
```

0.8147	0.6324	0.9575
0.9058	0.0975	0.9649
0.1270	0.2785	0.1576
0.9134	0.5469	0.9706

```
>> e=magic(4)
```

```
e =
```

16	2	3	13
5	11	10	8
9	7	6	12
4	14	15	1

```
>> b=ones(4,3)
```

```
b =
```

1	1	1
1	1	1
1	1	1
1	1	1

```
>> d=eye(4)
```

```
d =
```

1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1



Creating Matrix variable

```
>> mat = [4 3 1; 2 5 6]
mat =
    4   3   1
    2   5   6
```

```
>> mat = [3 5 7; 1 2]
Error using vertcat
Dimensions of matrices being concatenated are not consistent.
```

```
>> mat = [2:4; 3:5]
mat =
    2   3   4
    3   4   5
```



Linear indexing

```
>> intmat = [100 77; 28 14]
intmat =
    100    77
    28    14
>> intmat(1)
ans =
    100
>> intmat(2)
ans =
    28
>> intmat(3)
ans =
    77
>> intmat(4)
ans =
    14
```



Dimension

```
>> vec = -2:1
vec =
-2  -1   0   1
>> length(vec)
ans =
4
>> size(vec)
ans =
1   4
```

```
>> mat = [1:3; 5:7] '
mat =
1   5
2   6
3   7
>> [r, c] = size(mat)
r =
3
c =
2
```

```
>> size(mat)
ans =
3   2
```



Question

How could you create a matrix of zeros with the same size as another matrix?



numel function

```
>> v=9:-2:1  
  
v =  
  
9      7      5      3      1  
  
>> numel(v)  
  
ans =  
  
5
```

For vectors, **numel** is equivalent to the **length** of the vector. For matrices, it is the product of the number of rows and columns.



Question

```
mat = [1:3; 44 9  2; 5:-1:3]
mat(3,2)
mat(2,:)
size(mat)
mat(:,4) = [8;11;33]
numel(mat)
v = mat(3,:)
v(v(2))
v(1) = []
reshape(mat,2,6)
```

MATLAB

Unit 3-Lecture 6

BTech (CSBS) -Semester VII

29 July 2022, 09:35AM



Question

- 1) As the functions operate , how can we get an overall result for the matrix? Determine the overall maximum in the matrix?
- 2) Find the cummulative matrix from (1).
- 3) For vector v with a length n , $diff(v)$ will be $n-1$. Create a random integer matrix and find difference on each coloumn.
- 4) Create a matrix of all 10's.
- 5) Create a vector variable and substract 3 from every element.



Question

- 6) Create a matrix variable and divide every element by 3.
- 7) Create a matrix variable and square every element.
- 8) You are provided with following vector

```
>>vec=[5 9 3 4 6 11]
```

```
>>v=[0 1 0 0 1 1]
```

```
>>vec(v)
```

Error: Array indices must be positive integers or logical values.

Define this error and give correction for it



Question

- 9) Find logical true or false of vec from (8), for value greater than 9
- 10) Find same for value less than 9
- 11) Assume a vector vec that erroneously stores negative values, how can we eliminate those negative values?
- 12) With same vec from (11), using ‘find’ command and logical operators, instead of deleting negative values, retain only the positive values only.



Question

13) When two matrix have same dimension and are square, both array and matrix multiplication can be performed on them. For the following two matrices, perform $A.*B$, $A*B$ and $B*A$.

$$A = \begin{bmatrix} 1 & 4 \\ 3 & 3 \end{bmatrix} \text{ and } B = \begin{bmatrix} 1 & 2 \\ -1 & 0 \end{bmatrix}$$



SVKM'S
NMIMS
Deemed to be UNIVERSITY

NAVI MUMBAI

MATLAB

Unit 3-Lecture 7

BTech (CSBS) -Semester VII

2 August 2022, 09:35AM



Matrices

1. Create a matrix, *mat*. Is there a **rot180** function? Is there a **rot90** function (to rotate clockwise)?
2. Use above *mat*, and try to flip the matrix from left to right.
3. Use same *mat*, and flip up to down.
4. Create an empty vector and find the length of this vector.
5. Create a vector in 1 to 10 range and delete 4 element from same vector.



Matrices

6. Create a vector, `vec`, ranging 3 to 15. Delete a subset vector range from 9 to 12
7. From above given vector, check if individual elements could be deleted.
8. From the matrix as obtained in (1), remove the second coloumn.
9. Find the absolute vaule of vector in range -5 to 1
10. Find the sign of below given matrix

$$\begin{matrix} -4 & 2 & 8 \\ 0 & -10 & -42 \\ -9 & 15 & 0 \end{matrix}$$



Matrices

11. Find if below strings are equal:

str1= “hello”;

str2= “howdy”;



SVKM'S
NMIMS
Deemed to be UNIVERSITY

NAVI MUMBAI

MATLAB

Unit 3-Lecture 8

BTech (CSBS) -Semester VII

5 August 2022, 09:35AM



Solving Linear Equation

Solve System of Linear Equations Using linsolve:

A system of linear equations

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1$$

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2$$

...

$$a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = b_m$$

can be represented as the matrix equation $A \cdot \vec{x} = \vec{b}$ where A is the coefficient matrix,



Solving Linear Equation...contd.

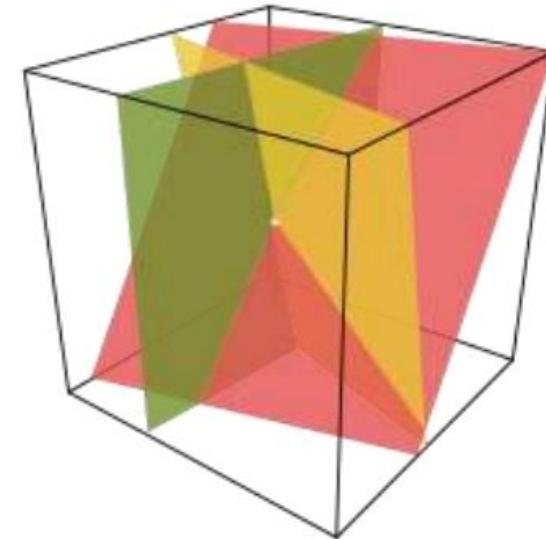
- Ex: Find values $x_1, x_2, x_3 \in \mathbb{R}$ that satisfy
 - $3x_1 + 2x_2 - x_3 - 1 = 0$
 - $2x_1 - 2x_2 + x_3 + 2 = 0$
 - $-x_1 - \frac{1}{2}x_2 - x_3 = 0$

Solution:

- Step 1: write the system of linear equations as a matrix equation

$$A = \begin{bmatrix} 3 & 2 & -1 \\ 2 & -2 & 1 \\ -1 & -\frac{1}{2} & -1 \end{bmatrix}, x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, b = \begin{bmatrix} 1 \\ -2 \\ 0 \end{bmatrix}.$$

- Step 2: Solve for $Ax = b$





Example 1

```
>> A = [5 -3 2; -3 8 4; 2 4 -9]; % Enter matrix A
>> b = [10; 20; 9]; % Enter column vector b
>> x = A\b % Solve for x
x =
    3.4442
    3.1982
    1.1868
>> c = A*x % check the solution
c =
    10.0000
    20.0000
    9.0000
```

The backslash (\) or the left division is used to solve a linear system of equations $\{A\}\{x\} = \{b\}$. For more information, type: help slash.



Question 1

- What's the A , x , and b for the following linear equations?
 - $2x_2 - 7 = 0$
 - $2x_1 - 3x_3 + 2 = 0$
 - $4x_2 + 2x_3 = 0$
 - $x_1 + 3x_3 = 0$

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 2 & 0 & -3 \\ 0 & 4 & 2 \\ 1 & 0 & 3 \end{bmatrix}, x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, b = \begin{bmatrix} 7 \\ -2 \\ 0 \\ 0 \end{bmatrix}.$$



Question

```
LinearEquation.m  ✘ +  
1  syms x y z  
2  eqn1= 5*x + y + 4*z == 12;  
3  eqn2= -x + y - 2*z == 43;  
4  eqn3= x - y + z== -10;  
5  [A,B]=equationsToMatrix([eqn1,eqn2,eqn3], [x,y,z])  
6  X=linsolve(A,B)
```

Command Window

```
>> LinearEquation
```

```
A =
```

```
[ 5, 1, 4]  
[-1, 1, -2]  
[ 1, -1, 1]
```

```
B =
```

```
12  
43  
-10
```

```
X =
```

```
167/6  
29/6  
-33
```



Solve System of Linear Equations Using `solve`

Use `solve` instead of `linsolve` if you have the equations in the form of expressions and not a matrix of coefficients.

```
8      %-----using solve function-----
9      sol=solve([eqn1, eqn2, eqn3], [x,y,z])
10     xSol = sol.x
11     ySol = sol.y
12     zSol = sol.z
```

```
sol =
struct with fields:
x: 167/6
y: 29/6
z: -33
```

```
xSol =
```

```
167/6
```

```
ySol =
```

```
29/6
```

```
zSol =
```

```
-33
```



Question

1. **Linear algebraic equations:** Find the solution of the following set of linear algebraic equations, as advised below.

$$x + 2y + 3z = 1$$

$$3x + 3y + 4z = 1$$

$$2x + 3y + 3z = 2.$$

- Write the equation in matrix form and solve for $\mathbf{x} = [x \ y \ z]^T$ using the left division \.



Gaussian elimination method

MATLAB has a built-in function, rref, that does precisely this reduction, i.e., transforms the matrix to its row reduced echelon form.

```
A=[1 2 3; 3 3 4; 2 3 3];  
B=[1; 1; 2];  
%C=B\A  
%linsolve(A,B)  
%transpose (C)  
%-----Gaussian elimination method  
C=[A B];  
Cr=rref(C)
```

```
Cr =  
1.0000 0 0 -0.5000  
0 1.0000 0 1.5000  
0 0 1.0000 -0.5000
```



Find eigenvalues and eigenvectors

Step 1: Enter matrix A and type $[V, D] = \text{eig}(A)$

```
>> A = [ 5 -3 2; -3 8 4; 2 4 -9];
```

```
>> [V, D] = eig(A)
```

```
V =
```

-0.1709	0.8729	0.4570
-0.2365	0.4139	-0.8791
0.9565	0.2583	-0.1357

```
D =
```

-10.3463	0	0
0	4.1693	0
0	0	10.1770

Step 2: Extract what you need:

'V' is an ' $n \times n$ ' matrix whose columns are eigenvectors

'D' is an ' $n \times n$ ' diagonal matrix that has the eigenvalues of 'A' on its diagonal.



Question

2. **Eigenvalues and eigenvectors:** Consider the following matrix.

$$\mathbf{A} = \begin{bmatrix} 3 & -3 & 4 \\ 2 & -3 & 4 \\ 0 & -1 & 1 \end{bmatrix}$$

- Find the eigenvalues and eigenvectors of \mathbf{A} .
- Show, by computation, that the eigenvalues of \mathbf{A}^2 are square of the eigenvalues of \mathbf{A} .
- Compute the square of the eigenvalues of \mathbf{A}^2 . You have now obtained the eigenvalues of \mathbf{A}^4 . From these eigenvalues, can you guess the structure of \mathbf{A}^4 ?
- Compute \mathbf{A}^4 . Can you compute \mathbf{A}^{-1} without using the `inv` function?

B.Tech (VII)- IT Workshop/MATLAB

Unit 3 (Part 1)

Practise Question set

NOTE:

- Explore MATLAB Desktop environment before starting these experiments.
 - All the variables provided are case sensitive.
1. If a variable has the dimensions 3×4 , could it be considered to be (check all that apply):
a matrix
a row vector
a column vector
a scalar
 2. If a variable has the dimensions 1×5 , could it be considered to be (check all that apply):
a matrix
a row vector
a column vector
a scalar
 3. If a variable has the dimensions 5×1 , could it be considered to be (check all that apply):
a matrix
a row vector
a column vector
a scalar
 4. If a variable has the dimensions 1×1 , could it be considered to be (check all that apply):
a matrix
a row vector
a column vector
a scalar
 5. Using the colon operator, create the following row vectors

2	3	4	5	6	7
1.1000	1.3000	1.5000	1.7000		
8	6	4	2		
 6. Using a built-in function, create a vector `vec` which consists of 20 equally spaced points in the range of $-\pi$ to $+\pi$.
 7. Write an expression using `linspace` that will result in the same as `2:0.2:3`

8. Using the colon operator and also the **linspace** function, create the following row vectors:

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

9. How many elements would be in the vectors created by the following expressions?

```
linspace(3,2000)
logspace(3,2000)
```

10. Create a variable *myend* which stores a random integer in the inclusive range of 5 to 9. Using the colon operator, create a vector that iterates from 1 to *myend* in steps of 3.

11. Using the colon operator and the transpose operator, create a column vector *myvec* that has the values -1 to 1 in steps of 0.5 .

12. Write an expression that refers to only the elements that have odd-numbered subscripts in a vector, regardless of the length of the vector. Test your expression on vectors that have both an odd and even number of elements.

13. Generate a 2×4 matrix variable *mat*. Replace the first row with $1:4$. Replace the third column (you decide with which values).

14. Generate a 2×4 matrix variable *mat*. Verify that the number of elements is the product of the number of rows and columns.

15. Which would you normally use for a matrix: **length** or **size**? Why?

16. When would you use **length** vs. **size** for a vector?

17. Generate a 2×3 matrix of random

real numbers, each in the range $[0, 1]$

real numbers, each in the range $[0, 10]$

integers, each in the inclusive range from 5 to 20

18. Create a variable *rows* that is a random integer in the inclusive range from 1 to 5. Create a variable *cols* that is a random integer in the inclusive range from 1 to 5. Create a matrix of all zeros with the dimensions given by the values of *rows* and *cols*.

19. Create a matrix variable *mat*. Find as many expressions as you can that would refer to the last element in the matrix, without assuming that you know how many elements or rows or columns it has (i.e., make your expressions general).

20. Create a vector variable *vec*. Find as many expressions as you can that would refer to the last element in the vector, without assuming that you know how many elements it has (i.e., make your expressions general).

21. Create a 2×3 matrix variable *mat*. Pass this matrix variable to each of the following functions and make sure you understand the result: **flip**, **fliplr**, **flipud**, and **rot90**. In how many different ways can you **reshape** it?

22. What is the difference between `fliplr(mat)` and `mat = fliplr(mat)`?
23. Use **reshape** to reshape the row vector `1:4` into a 2×2 matrix; store this in a variable named `mat`. Next, make 2×3 copies of `mat` using both **repelem** and **repmat**.

B.Tech (VII)- IT Workshop/MATLAB

Unit 3 (Part 2)

Practise Question set

NOTE:

- Explore MATLAB Desktop environment before starting these experiments.
 - All the variables provided are case sensitive.
24. Create a 3×5 matrix of random real numbers. Delete the third row.
25. Given the matrix:

```
>> mat = randi ([1 20] , 3 , 5)
mat =
    6 17 7 13 17
    17 5 4 10 12
    6 19 6 8 11
```

Why wouldn't this work:

```
mat (2:3 , 1:3) = ones (2)
```

26. Create a three-dimensional matrix with dimensions $2 \times 4 \times 3$ in which the first "layer" is all 0 s, the second is all 1 s and the third is all 5 s. Use **size** to verify the dimensions.
27. Create a vector **x** which consists of 20 equally spaced points in the range from $-\pi$ to $+\pi$. Create a **y** vector which is **sin(x)**.
28. Create a 3×5 matrix of random integers, each in the inclusive range from -5 to 5 . Get the **sign** of every element.
29. Find the sum $3+5+7+9+11$.
30. Find the sum of the first n terms of the harmonic series where n is an integer variable greater than one.

$$1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \dots$$

31. Find the following sum by first creating vectors for the numerators and denominators:

$$\frac{3}{1} + \frac{5}{2} + \frac{7}{3} + \frac{9}{4}$$

32. Create a matrix and find the product of each row and column using **prod**.
34. Write a relational expression for a vector variable that will verify whether the last value in a vector created by **cumsum** is the same as the result returned by **sum**.

35. Create a vector of five random integers, each in the inclusive range from -10 to 10 .
10. Perform each of the following:
- subtract 3 from each element
 - count how many are positive
 - get the cumulative minimum
36. Create a 3×5 matrix. Perform each of the following:
- Find the maximum value in each column.
 - Find the maximum value in each row.
 - Find the maximum value in the entire matrix.
 - Find the cumulative maxima.
37. Find two ways to create a 3×5 matrix of all 100s (Hint: use **ones** and **zeros**).
38. Given the two matrices:

$$\begin{array}{c} \text{A} \qquad \text{B} \\ \left[\begin{array}{ccc} 1 & 2 & 3 \end{array} \right] \quad \left[\begin{array}{ccc} 2 & 4 & 1 \end{array} \right] \\ \left[\begin{array}{ccc} 4 & -1 & 6 \end{array} \right] \quad \left[\begin{array}{ccc} 1 & 3 & 0 \end{array} \right] \end{array}$$

Perform the following operations:

A + B
A - B
A * B

39. The built-in function **clock** returns a vector that contains 6 elements: the first three are the current date (year, month, and day) and the last three represent the current time in hours, minutes, and seconds. The seconds is a real number, but all others are integers. Store the result from **clock** in a variable called **myc**. Then, store the first three elements from this variable in a variable **today** and the last three elements in a variable **now**. Use the **fix** function on the vector variable **now** to get just the integer part of the current time.
40. A vector **v** stores for several employees of the Green Fuel Cells Corporation, their hours worked one week followed for each by the hourly pay rate. For example, if the variable stores

```
>> v
v =
33.000010.500040.000018.000020.00007.5000
```

that means the first employee worked 33 hours at \$10.50 per hour, the second worked 40 hours at \$18 an hour, and so on. Write code that will separate this into two vectors, one that stores the hours worked and another that stores the hourly rates. Then, use the array multiplication operator to create a vector, storing in the new vector, the total pay for every employee.

41. A company is calibrating some measuring instrumentation and has measured the radius and height of one cylinder 8 separate times; they are in vector variables r and h . Find the volume from each trial, which is given by $\pi r^2 h$. Also use logical indexing first to make sure that all measurements were valid (>0).

```
>> r = [5.499 5.498 5.5 5.5 5.52 5.51 5.5 5.48];
>> h = [11.1 11.12 11.09 11.11 11.11 11.1 11.08 11.11];
```

42. For the following matrices A, B, and C:

$$A = \begin{bmatrix} 1 & 4 \\ 3 & 2 \end{bmatrix} \quad B = \begin{bmatrix} 2 & 1 & 3 \\ 1 & 5 & 6 \\ 3 & 6 & 0 \end{bmatrix} \quad C = \begin{bmatrix} 3 & 2 & 5 \\ 4 & 1 & 2 \end{bmatrix}$$

Give the result of $3*A$.

Give the result of $A*C$.

Are there any other matrix multiplications that can be performed? If so, list them.

43. For the following vectors and matrices A, B, and C:

$$A = \begin{bmatrix} 4 & 1 & -1 \\ 2 & 3 & 0 \end{bmatrix} \quad B = [1 \ 4] \quad C = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

Perform the following operations, if possible. If not, just say it can't be done!

$A * B$
 $B * C$
 $C * B$

44. The matrix variable *rainmat* stores the total rainfall in inches for some districts for the years 2010–13. Each row has the rainfall amounts for a given district. For example, if *rainmat* has the value:

```
>> rainmat
ans =
25 33 29 42
53 44 40 56
etc.
```

district 1 had 25 in. in 2010, 33 in 2011, etc. Write expression(s) that will find the number of the districts that had the highest total rainfall for the entire four year period.

45. Generate a vector of 20 random integers, each in the range from 50 to 100. Create a variable *evens* that stores all of the even numbers from the vector, and a variable *odds* that stores the odd numbers.
46. Assume that the function **diff** does not exist. Write your own expression(s) to accomplish the same thing for a vector.

47. Create a vector variable `vec`; it can have any length. Then, write assignment statements that would store the first half of the vector in one variable and the second half in another. Make sure that your assignment statements are general, and work whether `vec` has an even or odd number of elements (Hint: use a rounding function such as `fix`).



SVKM'S
NMIMS
Deemed to be UNIVERSITY

NAVI MUMBAI

MATLAB

Unit 4-Lecture 11

BTech (CSBS) -Semester VII

16 August 2022, 09:35AM



Basic plotting

- Overview,
- axis labels, and annotations,
- adding titles,
- specifying line styles and colours.
- creating simple plots,
- multiple data sets in one plot,



Overview

The most basic and perhaps most useful command for producing a 2-D plot is

```
plot(xvalues, yvalues, 'style-option')
```

where *xvalues* and *yvalues* are vectors containing the *x*- and *y*-coordinates of points on the graph and the *style-option* is an optional argument that specifies the color, the line style (e.g., solid, dashed, dotted), and the point-marker style (e.g., *o*, *+*, ***). All three style options can be specified together. The two vectors *xvalues* and *yvalues* MUST have the same length. Unequal length of the two vectors is the most common source of error in the plot command. The **plot** function also works with a single-vector argument, in which case the elements of the vector are plotted against row or column indices. Thus, for two column vectors *x* and *y* each of length *n*,



Overview

- `plot(x,y)` plots y versus x with a solid line (the default line style),
- `plot(x,y,'--')` plots y versus x with a dashed line (more on this below), and
- `plot(x)` plots the elements of x against their row index.

*For on-line help
type:
help graph2d*



Style Options

Color Style-option	Line Style-option	Marker Style-option
y yellow	- solid	+
m magenta	-- dashed	o circle
c cyan	:	*
r red	-. dash-dot	x x-mark
g green	none	.
b blue		^ up triangle
w white		s square
k black		d diamond, etc.



Style Options

Examples:

`plot(x,y,'r')`

plots y versus x with a red solid line,

`plot(x,y,:')`

plots y versus x with a dotted line,

`plot(x,y,'b--')`

plots y versus x with a blue dashed line, and

`plot(x,y,'+')`

plots y versus x as unconnected points marked by +.

When no style-option is specified, MATLAB uses a blue solid line by default.



Label and title

Plots may be annotated with `xlabel`, `ylabel`, `title`, and `text` commands.

The first three commands take string arguments, whereas the last one requires three arguments—`text(x-coordinate, y-coordinate, 'text')`, where the coordinate values are taken from the current plot. Thus,

<code>xlabel('Pipe Length')</code>	labels the <i>x</i> -axis with Pipe Length,
<code>ylabel('Fluid Pressure')</code>	labels the <i>y</i> -axis with Fluid Pressure,
<code>title('Pressure Variation')</code>	titles the plot with Pressure Variation, and
<code>text(2,6,'Note this dip')</code>	writes “Note this dip” at the location (2.0,6.0) in the plot coordinates.



Label and title

Plots may be annotated with `xlabel`, `ylabel`, `title`, and `text` commands.

The first three commands take string arguments, whereas the last one requires three arguments—`text(x-coordinate, y-coordinate, 'text')`, where the coordinate values are taken from the current plot. Thus,

<code>xlabel('Pipe Length')</code>	labels the <i>x</i> -axis with Pipe Length,
<code>ylabel('Fluid Pressure')</code>	labels the <i>y</i> -axis with Fluid Pressure,
<code>title('Pressure Variation')</code>	titles the plot with Pressure Variation, and
<code>text(2,6,'Note this dip')</code>	writes “Note this dip” at the location (2.0,6.0) in the plot coordinates.



Legend

`legend(string1, string2, ...)` produces legend using the text in *string*1, *string*2, etc., as labels,
`legend(LineStyle1, string1, ...)` specifies the line style of each label,
`legend(..., pos)` writes the legend outside the plot-frame if *pos* = -1 and inside if *pos* = 0, (there are other options for *pos* too), and
`legend off` deletes the legend from the plot.



Axis Control

Once a plot is generated, you can change the axes limits with the **axis** command. Typing

`axis([xmin xmax ymin ymax])`

changes the current axes limits to the specified new values *xmin* and *xmax* for the *x*-axis and *ymin* and *ymax* for the *y*-axis.

Examples:

```
axis([-5 10 2 22]);      sets the x-axis from -5 to 10, y-axis from 2 to 22,  
axy = [-5 10 2 22]; axis(axy);          same as above, and  
ax = [-5 10]; ay=[2 22]; axis([ax ay]);    same as above.
```



Axis Control

`axis('equal')`

sets equal scale on both axes,

`axis('square')`

sets the default rectangular frame to a square,

`axis('normal')`

resets the axis to default values,

`axis('axis')`

freezes the current axes limits, and

`axis('off')`

removes the surrounding frame and the tick marks.



Semi control of Axis

It is possible to control only part of the axes limits and let MATLAB set the other limits automatically. This is achieved by specifying the desired limits in the **axis** command along with **inf** as the values of the limits that you would like to be set automatically. For example,

`axis([-5 10 -inf inf])`

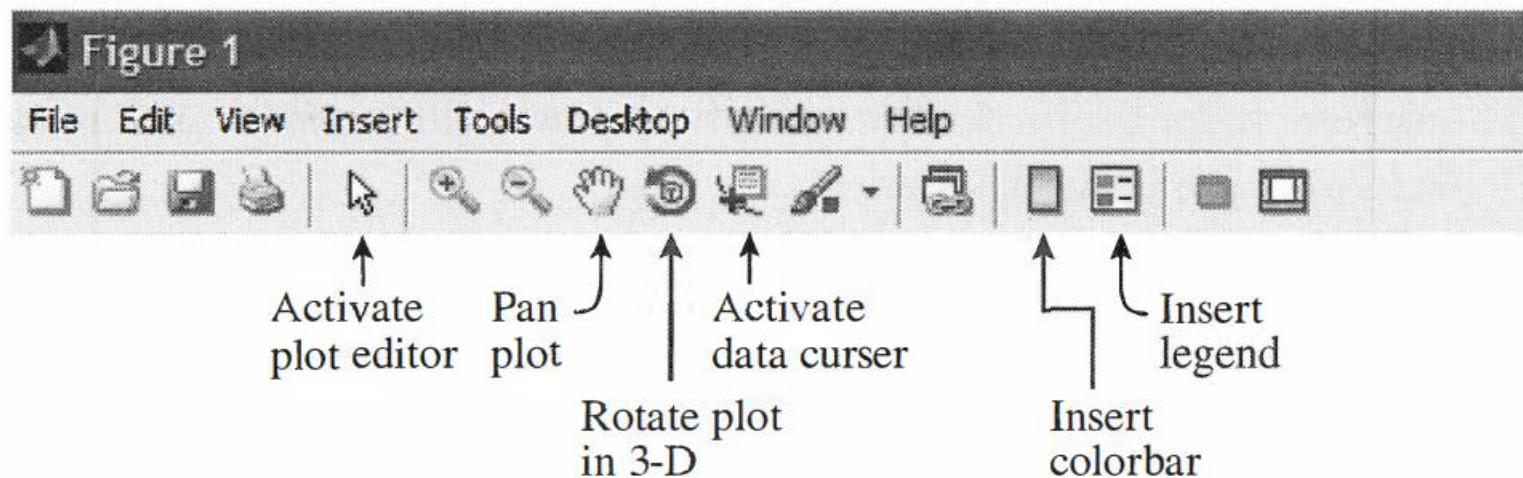
sets the *x*-axis limits at -5 and 10 and lets the *y*-axis limits be set automatically, and

`axis([-5 inf -inf 22])`

sets the lower limit of the *x*-axis and the upper limit of the *y*-axis, and leaves the other two limits to be set automatically.



Modify plot with Plot Editor



MATLAB

Unit 4-Lecture 12

BTech (CSBS) -Semester VII

26 August 2022, 09:35AM



Basic plotting

- Overview,
- axis labels, and annotations,
- **creating simple plots,**
- specifying line styles and colours
- adding titles,
- multiple data sets in one plot,



Overlay plot

Method 1: Using the `plot` command to generate overlay plots

```
plot(x1,y1, x2,y2,':', x3,y3,'o')
```

Method 2: Using the `hold` command to generate overlay plots

```
% - Script file to generate an overlay plot with the hold command -  
x = linspace(0,2*pi,100); % Generate vector x  
y1 = sin(x); % Calculate y1  
plot(x,y1) % Plot (x,y1) with solid line  
hold on % Invoke hold for overlay plots  
y2 = x; plot(x,y2,'--') % Plot (x,y2) with dashed line  
y3 = x - (x.^3)/6 + (x.^5)/120; % Calculate y3  
plot(x,y3,'o') % Plot (x,y3) as pts. marked by 'o'  
axis([0 5 -1 5]) % Zoom in with new axis limits  
hold off % Clear hold command
```



Overlay plot

Method 3: Using the line command to generate overlay plots

```
% -- Script file to generate an overlay plot with the line command --
% -----
% First, generate some data
t = linspace(0,2*pi,100);           % Generate vector t
y1 = sin(t);                      % Calculate y1, y2, y3
y2 = t;
y3 = t - (t.^3)/6 + (t.^5)/120;

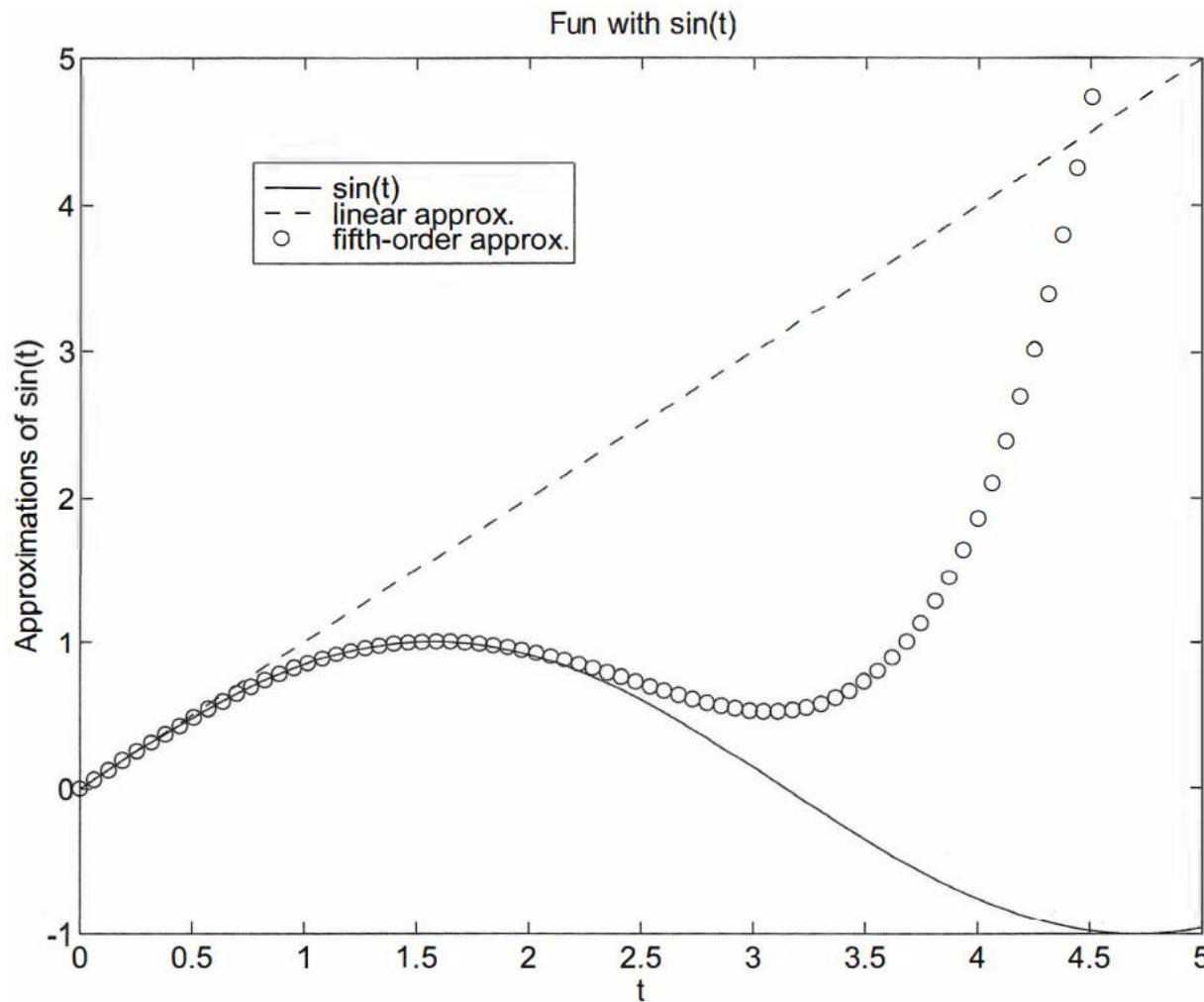
% Now, plot the three lines
plot(t,y1)                         % Plot (t,y1) with (default) solid line
line(t,y2,'linestyle','--')         % Add line (t,y2) with dashed line and
line(t,y3,'marker','o',...          % Add line (t,y3) plotted with circles--
    'linestyle', 'none')           % but no line
% Adjust the axes
axis([0 5 -1 5])                  % Zoom in with new axis limits

% Dress up the graph
xlabel('t')                         % Put x-label
ylabel('Approximations of sin(t)')   % Put y-label
title('Fun with sin(t)')            % Put title

legend('sin(t)', 'linear approx.', 'fifth-order approx.')
% add legend
```



Overlay plot





Specialized 2D plot

<code>area</code>	creates a filled area plot,
<code>bar</code>	creates a bar graph,
<code>barg</code>	creates a horizontal bar graph,
<code>comet</code>	makes an animated 2-D plot,
<code>compass</code>	creates arrow graph for complex numbers,
<code>contour</code>	makes contour plots,
<code>contourf</code>	makes filled contour plots,
<code>errorbar</code>	plots a graph and puts error bars,
<code>feather</code>	makes a feather plot,
<code>fill</code>	draws filled polygons of specified color,
<code>fplot</code>	plots a function of a single variable,



Specialized 2D plot

fplot	plots a function of a single variable,
hist	makes histograms,
loglog	creates plot with log scale on both the x -axis and the y -axis,
pareto	makes pareto plots,
pcolor	makes pseudocolor plot of a matrix,
pie	creates a pie chart,
plotyy	makes a double y -axis plot,
plotmatrix	makes a scatter plot of a matrix,
polar	plots curves in polar coordinates,
quiver	plots vector fields,
rose	makes angled histograms,
scatter	creates a scatter plot,



Specialized 2D plot

`semilogx`

`semilogy`

`stairs`

`stem`

makes semilog plot with log scale on the x -axis,
makes semilog plot with log scale on the y -axis,
plots a stair graph, and
plots a stem graph.



Specialized 2D plot

Function	Example Script	Output
fplot	$f(t) = t \sin t, 0 \leq t \leq 10\pi$ <code>fplot('x.*sin(x)',[0 10*pi])</code> Note that the function to be plotted must be written as a function of x .	
semilogx	$x = e^{-t}, y = t, 0 \leq t \leq 2\pi$ <code>t = linspace(0,2*pi,200); x = exp(-t); y = t; semilogx(x,y), grid</code>	

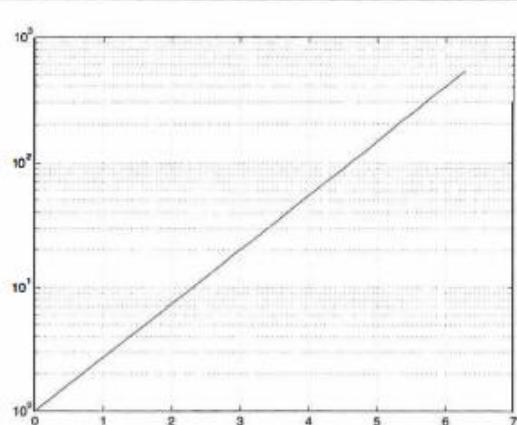


Specialized 2D plot

semilogy

$$x = t, y = e^t, 0 \leq t \leq 2\pi$$

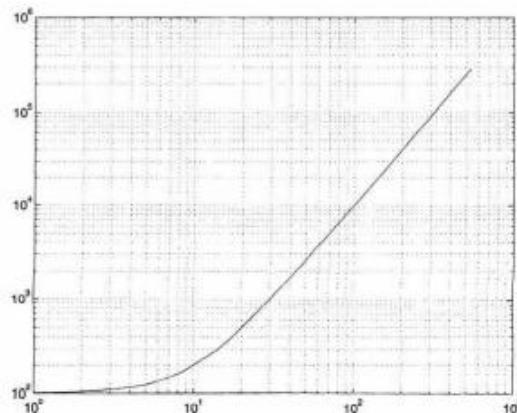
```
t = linspace(0,2*pi,200);
semilogy(t,exp(t))
grid
```



loglog

$$x = e^t, y = 100 + e^{2t}, 0 \leq t \leq 2\pi$$

```
t = linspace(0,2*pi,200);
x = exp(t);
y = 100 + exp(2*t);
loglog(x,y), grid
```





Specialized 2D plot

polar	$r^2 = 2 \sin 5t, 0 \leq t \leq 2\pi$ <pre>t = linspace(0,2*pi,200); r = sqrt(abs(2*sin(5*t))); polar(t,r)</pre>	
fill	$r^2 = 2 \sin 5t, 0 \leq t \leq 2\pi$ $x = r \cos t, y = r \sin t$ <pre>t = linspace(0,2*pi,200); r = sqrt(abs(2*sin(5*t))); x = r.*cos(t); y = r.*sin(t); fill(x,y,'k'), axis('square')</pre>	

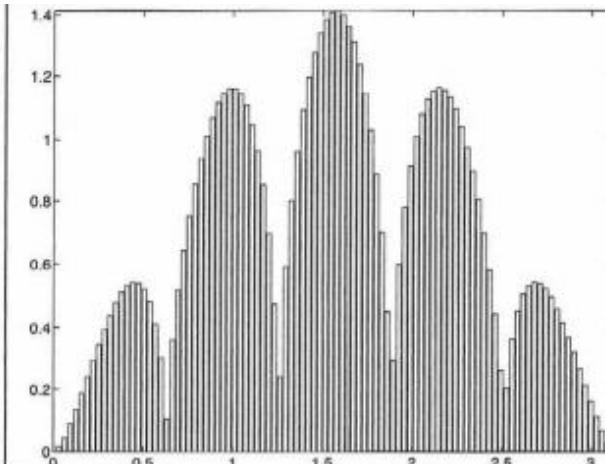


Specialized 2D plot

bar

$$\begin{aligned}r^2 &= 2 \sin 5t, \quad 0 \leq t \leq 2\pi \\y &= r \sin t\end{aligned}$$

```
t = linspace(0,2*pi,200);
r = sqrt(abs(2*sin(5*t)));
y = r.*sin(t);
bar(t,y)
axis([0 pi 0 inf]);
```

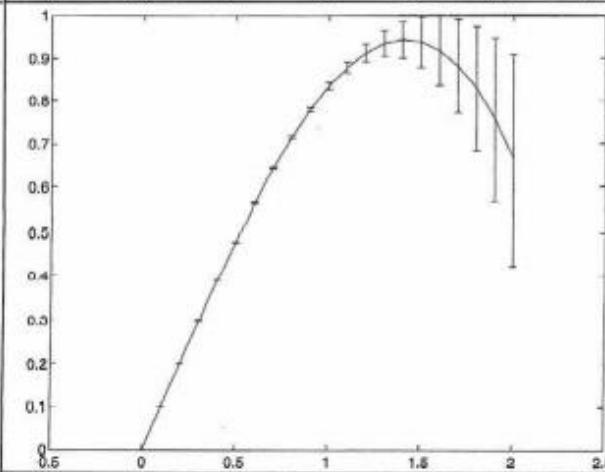


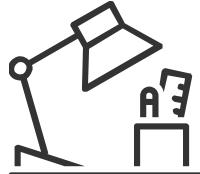
errorbar

$$f_{\text{approx}} = x - \frac{x^3}{3!}, \quad 0 \leq x \leq 2$$

$$\text{error} = f_{\text{approx}} - \sin x$$

```
x = 0:.1:2;
aprx2 = x - x.^3/6;
er = aprx2 - sin(x);
errorbar(x,aprx2,er)
```





Specialized 2D plot

barh	<p>World population by continents.</p> <pre>cont = char('Asia','Europe','Africa',... 'N. America','S. America'); pop = [3332;696;694;437;307]; barh(pop) for i=1:5, gtext(cont(i,:)); end xlabel('Population in millions') Title('World Population (1992)',... 'fontsize',18)</pre>	<table border="1"><caption>World Population (1992)</caption><thead><tr><th>Continent</th><th>Population (millions)</th></tr></thead><tbody><tr><td>Asia</td><td>3332</td></tr><tr><td>Europe</td><td>696</td></tr><tr><td>Africa</td><td>694</td></tr><tr><td>N. America</td><td>437</td></tr><tr><td>S. America</td><td>307</td></tr></tbody></table>	Continent	Population (millions)	Asia	3332	Europe	696	Africa	694	N. America	437	S. America	307
Continent	Population (millions)													
Asia	3332													
Europe	696													
Africa	694													
N. America	437													
S. America	307													
plotyy	$y_1 = e^{-x} \sin x, 0 \leq t \leq 10$ $y_2 = e^x$ <pre>x = 1:.1:10; y1 = exp(-x).*sin(x); y2 = exp(x); Ax = plotyy(x,y1,x,y2); hy1 = get(Ax(1),'ylabel'); hy2 = get(Ax(2),'ylabel'); set(hy1,'string','e^-x sin(x)'); set(hy2,'string','e^x');</pre>													



Specialized 2D plot

area	$y = \frac{\sin(x)}{x}, \quad -3\pi \leq x \leq 3\pi$ <pre>x = linspace(-3*pi,3*pi,100); y = -sin(x)./x; area(x,y) xlabel('x'), ylabel('sin(x)./x') hold on x1 = x(46:55); y1 = y(46:55); area(x1,y1,'facecolor','y')</pre>	
pie	<p>World population by continents.</p> <pre>cont = char('Asia','Europe','Africa',... 'N. America','S. America'); pop = [3332;696;694;437;307]; pie(pop) for i=1:5, gtext(cont(i,:)); end Title('World Population (1992)',... 'fontsize',18)</pre>	<p>World Population (1992)</p>



Using subplot to multiple plot

If you want to make a few plots and place the plots side by side (not overlay), use the `subplot` command to design your layout. The subplot command requires three integer arguments:

```
subplot(m,n,p)
```

MATLAB

Unit 4-Lecture 13

BTech (CSBS) -Semester VII

30 August 2022, 09:35AM



Basic plotting

- Overview,
- axis labels, and annotations,
- creating simple plots,
- specifying line styles and colours
- adding titles,
- multiple data sets in one plot,



Questions

Let's say that you want to plot these two equations in the same window:

$$\begin{aligned}y1 &= \cos(x) \\y2 &= x^2 - 1\end{aligned}$$



Steps for 2D Plots

1. Define your interval of interest, think of highest and lowest values, and a step.
2. Define your function $y = f(x)$. Take into account that you're working with arrays, not with scalars, use dot operators.
3. Use appropriate 2D built-in functions.



1. Define your Interval

Think:

- What values for x do I want to take into account? What steps in the array should I consider?



2. Define your Function(s)

Think of lower and upper values, and steps

`x = -1 : 0.1 : 1.5;`

`y1 = cos(x);`

`y2 = x.^2 - 1;`

Now, x, y1 and y2 are vectors with appropriate values.



3. Use 2D built-in Functions

You can use functions such as:

plot

stem

polar, compass, rose

loglog, semilogx, semilogy

area, fill

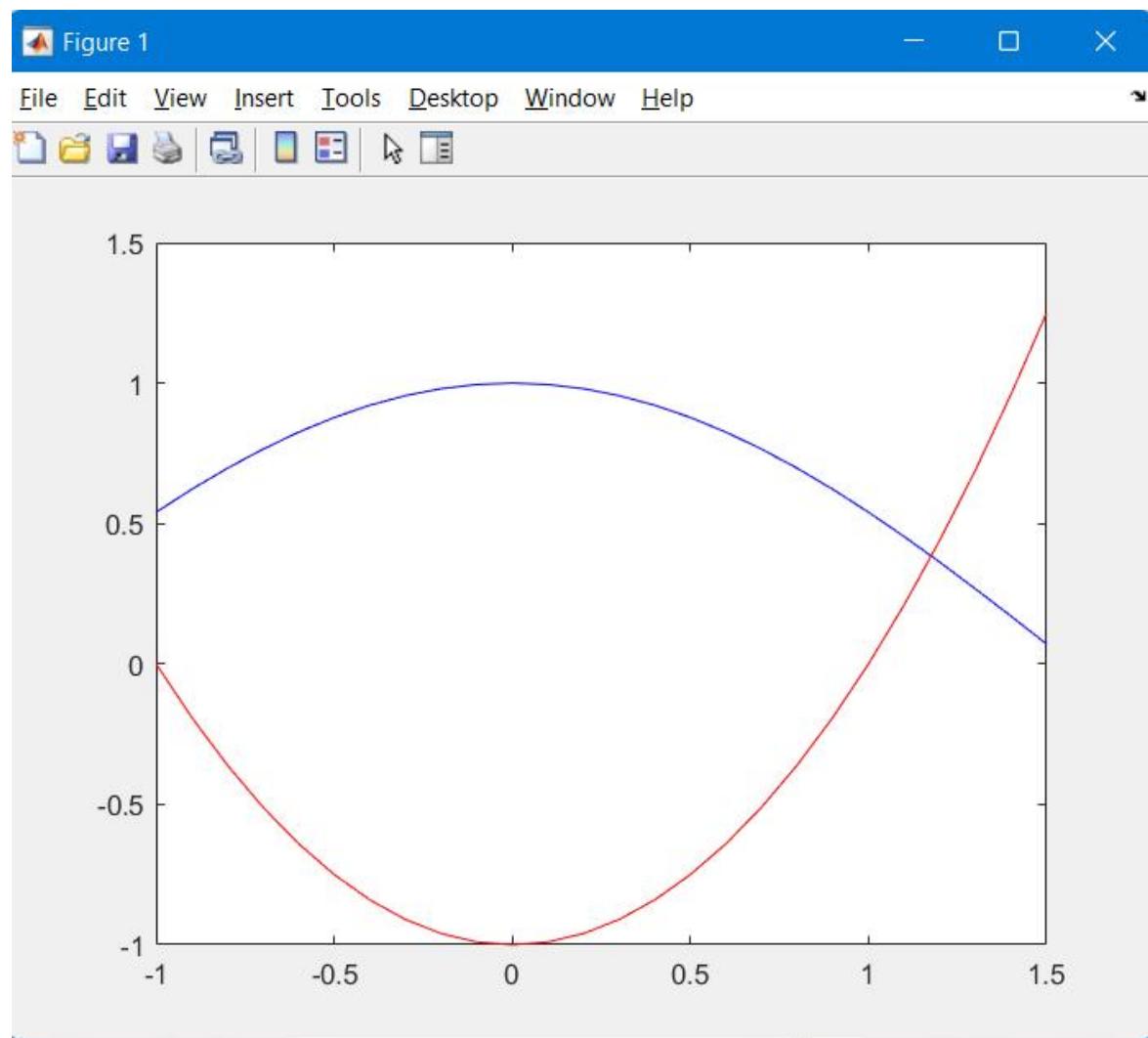
pie

hist, stairs



Solution

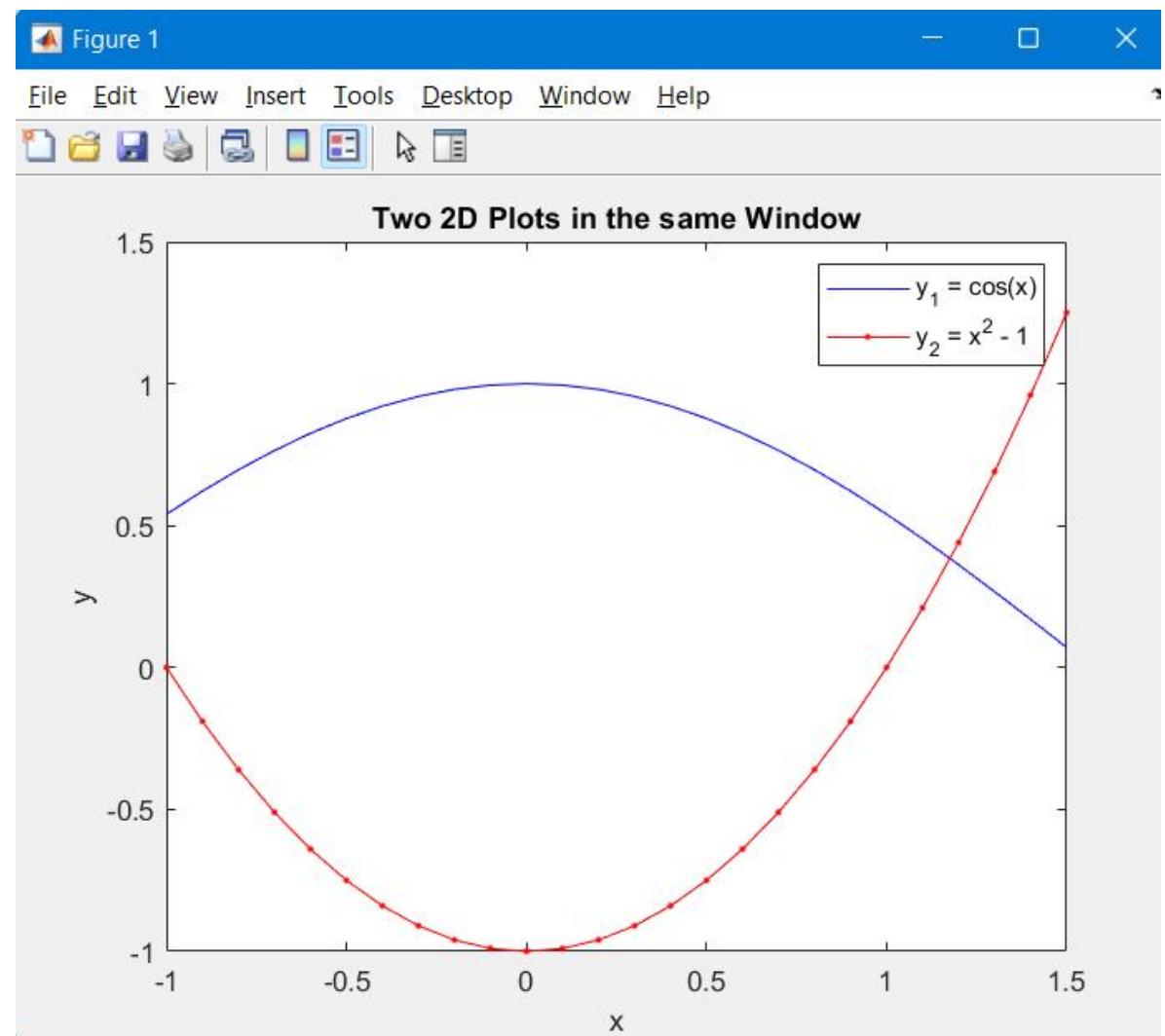
```
lab6.m  ×  +  
1      x = -1 : 0.1 : 1.5;  
2      y1 = cos(x);  
3      y2 = x.^2 - 1;  
4      plot(x, y1, 'b', x, y2, 'r')
```





Solution

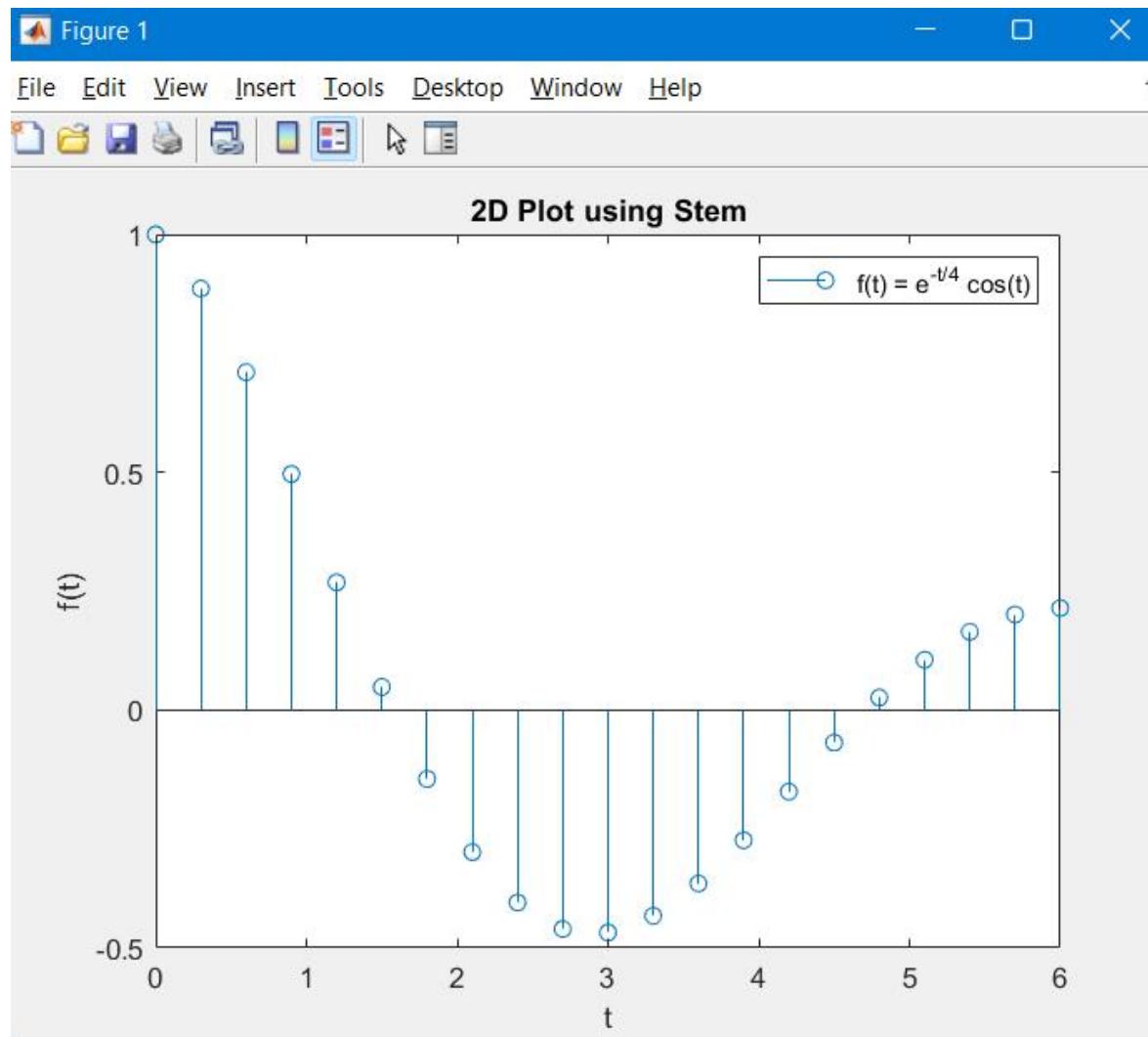
```
6 x = -1 : 0.1 : 1.5;
7 y1 = cos(x);
8 y2 = x.^2 - 1;
9 plot(x, y1, 'b', x, y2, 'r.-')
10 title('Two 2D Plots in the same Window')
11 legend('y_1 = cos(x)', 'y_2 = x^2 - 1')
12 xlabel('x')
13 ylabel('y')
```





Solution

```
15 t = 0 : .3 : 2*pi;
16 f = exp(-t/4) .* cos(t);
17 stem(t, f)
18 title('2D Plot using Stem')
19 legend('f(t) = e^{-t/4} cos(t)')
20 xlabel('t')
21 ylabel('f(t)')
22
```



MATLAB

Unit 4-Lecture 14

BTech (CSBS) -Semester VII

2 September 2022, 09:35AM



Basic plotting

- Overview,
- axis labels, and annotations,
- creating simple plots,
- specifying line styles and colours
- adding titles,
- multiple data sets in one plot,



fplot

- `fplot(@fun, lims)` - plots the function `fun` between the x-axis limits
- `lims = [xmin xmax ymin ymax]` – axis limits
- The function `fun(x)` must return a row vector for
- each element of vector `x`.



AXIS Control

1. axis scaling and appearance.
2. **axis([xmin xmax ymin ymax])**
3. Sets scaling for the x- and y-axes on the current plot.
4. **axis auto** - returns the axis scaling to its default, automatic mode
5. **axis off** - turns off all axis labeling, tick marks and background.
6. **axis on** - turns axis labeling, tick marks and background back on.
7. **axis equal** – makes both axes equal length



3D Plot

The general syntax for the `plot3` command is

```
plot3(x, y, z, 'style-option')
```

<code>plot3</code>	plots curves in space,
<code>stem3</code>	creates discrete data plot with stems in 3-D,
<code>bar3</code>	plots 3-D bar graph,
<code>bar3h</code>	plots 3-D horizontal bar graph,
<code>pie3</code>	makes 3-D pie chart,
<code>comet3</code>	makes animated 3-D line plot,
<code>fill3</code>	draws filled 3-D polygons,
<code>contour3</code>	makes 3-D contour plots,
<code>quiver3</code>	draws vector fields in 3-D,
<code>scatter3</code>	makes scatter plots in 3-D,
<code>mesh</code>	draws 3-D mesh surfaces (wire-frame),



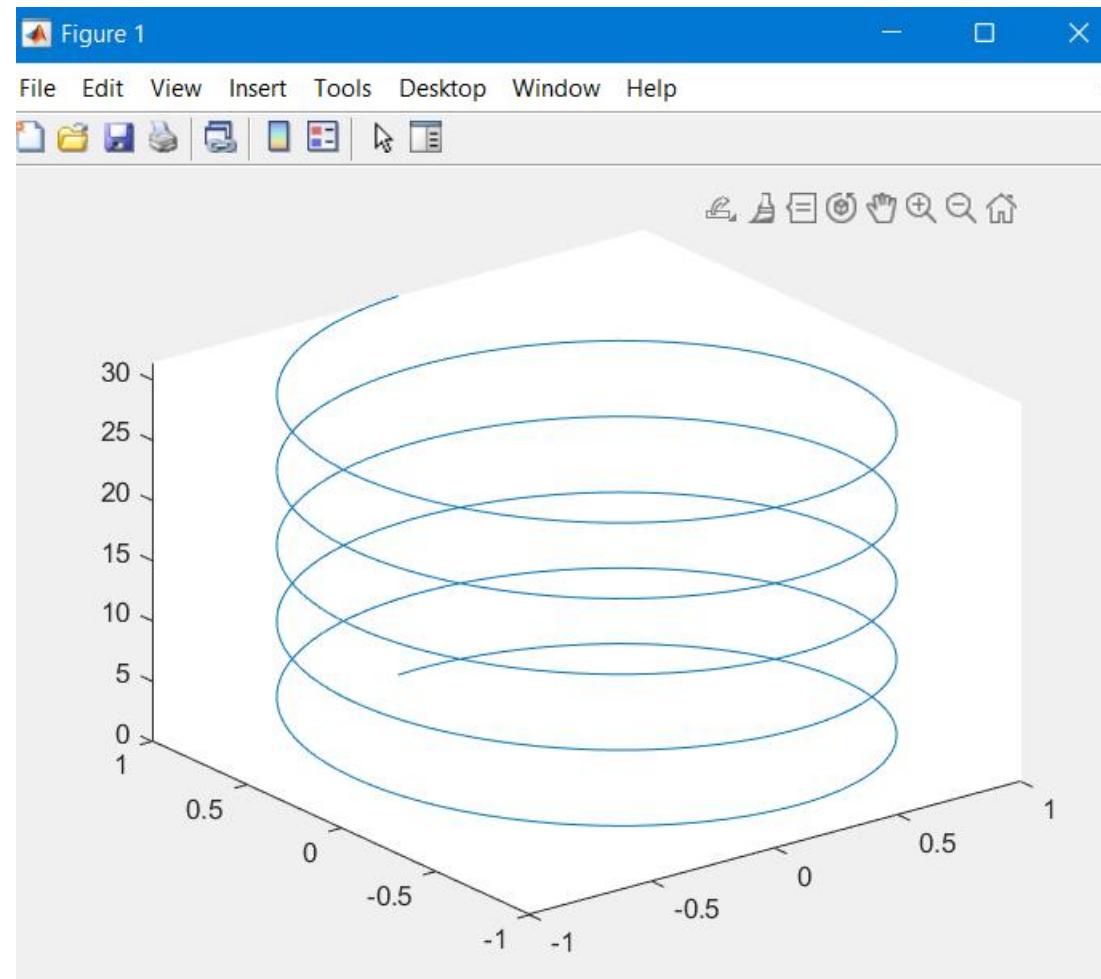
3D Plot

meshc	draws 3-D mesh surfaces along with contours,
meshz	draws 3-D mesh surfaces with reference plane curtains,
surf	creates 3-D surface plots,
surfc	creates 3-D surface plots along with contours,
surfl	creates 3-D surface plots with specified light source,
trimesh	mesh plot with triangles,
trisurf	surface plot with triangles,
slice	draws a volumetric surface with slices,
waterfall	creates a <i>waterfall</i> plot of 3-D data,
cylinder	generates a cylinder,
ellipsoid	generates an ellipsoid, and
sphere	generates a sphere.



3D Plot: Question 1

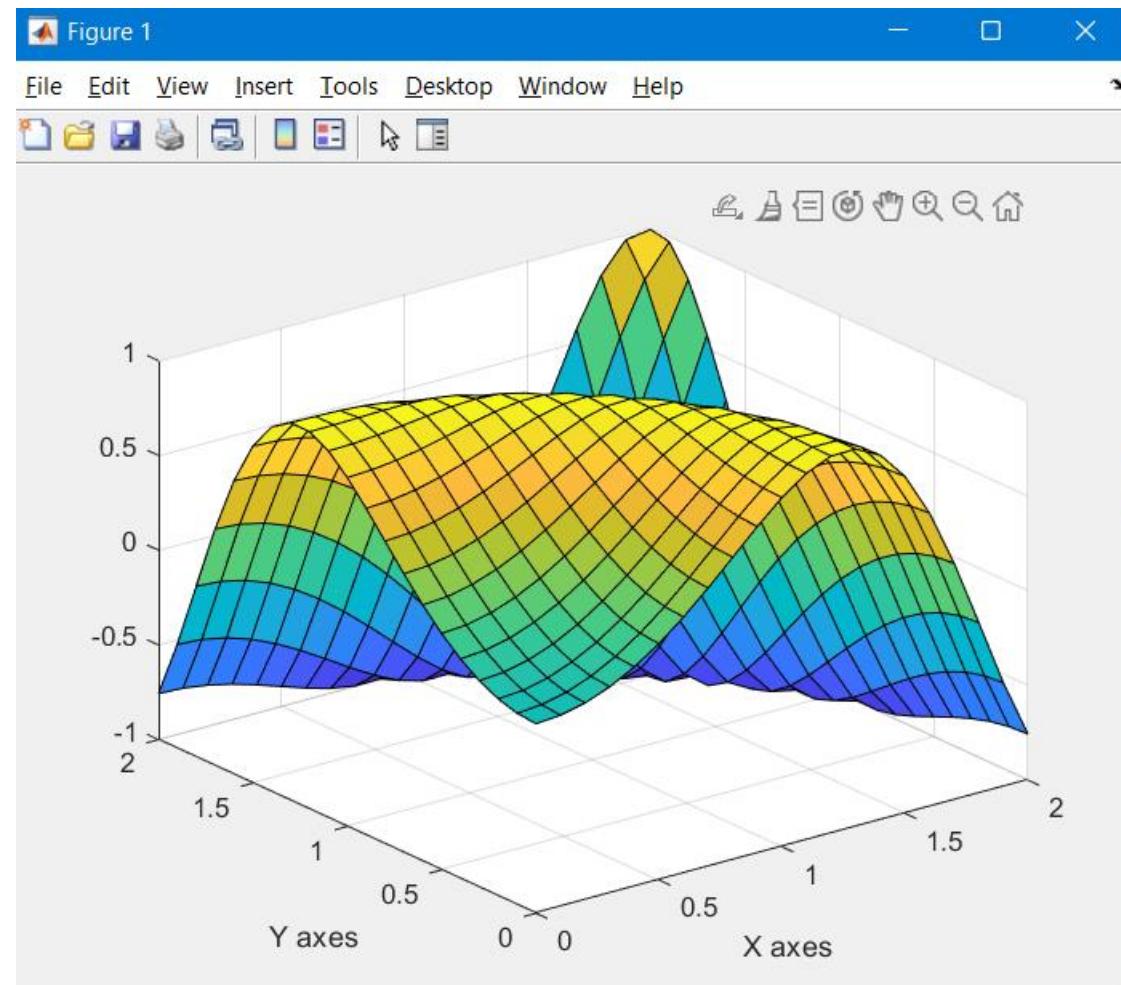
```
22  
23 t = 0:pi/50:10*pi;  
24 plot3(sin(t),cos(t),t)  
25
```





Surface Plot: Question 2

```
--  
26 x = 0:0.1:2;  
27 y = 0:0.1:2;  
28 [xx, yy] = meshgrid(x,y);  
29 zz=sin(xx.^2+yy.^2);  
30 surf(xx,yy,zz)  
31 xlabel('X axes')  
32 ylabel('Y axes')  
33
```



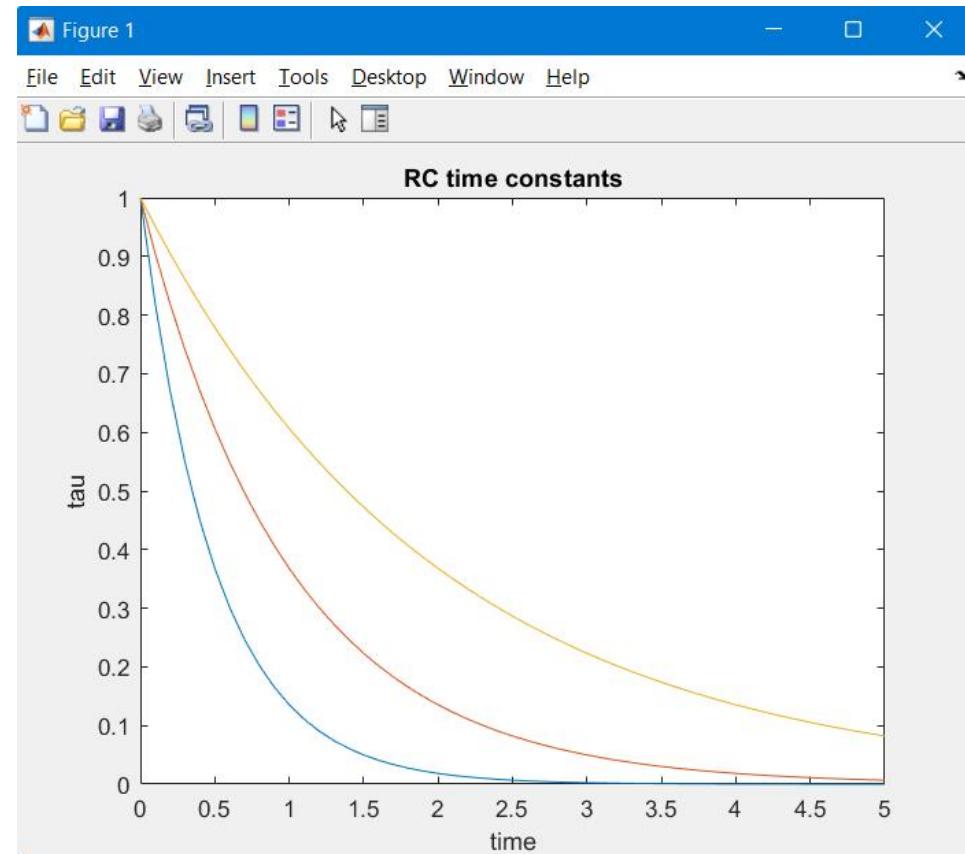


Question 3

Plot voltage vs time for various RC time constants

$$\frac{V}{V_0} = e^{-t/\tau}$$

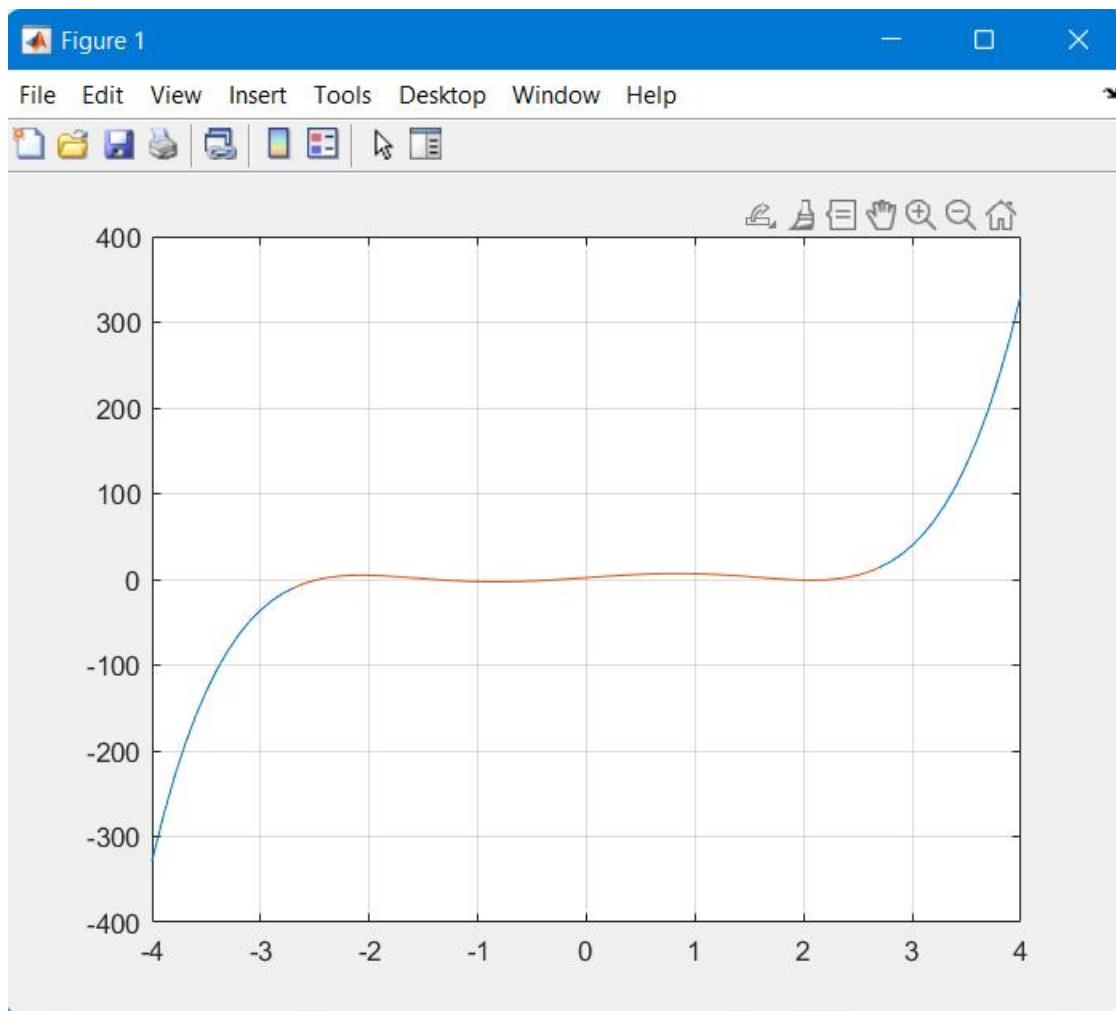
```
34 time = 0:0.1:5;
35 tau = [0.5 1.0 2.0];
36 [TIME TAU] = meshgrid(time,tau);
37 V = exp(-TIME./TAU);
38 plot(time,V)
39 xlabel('time')
40 ylabel('tau')
41 title('RC time constants')
```





Question 4

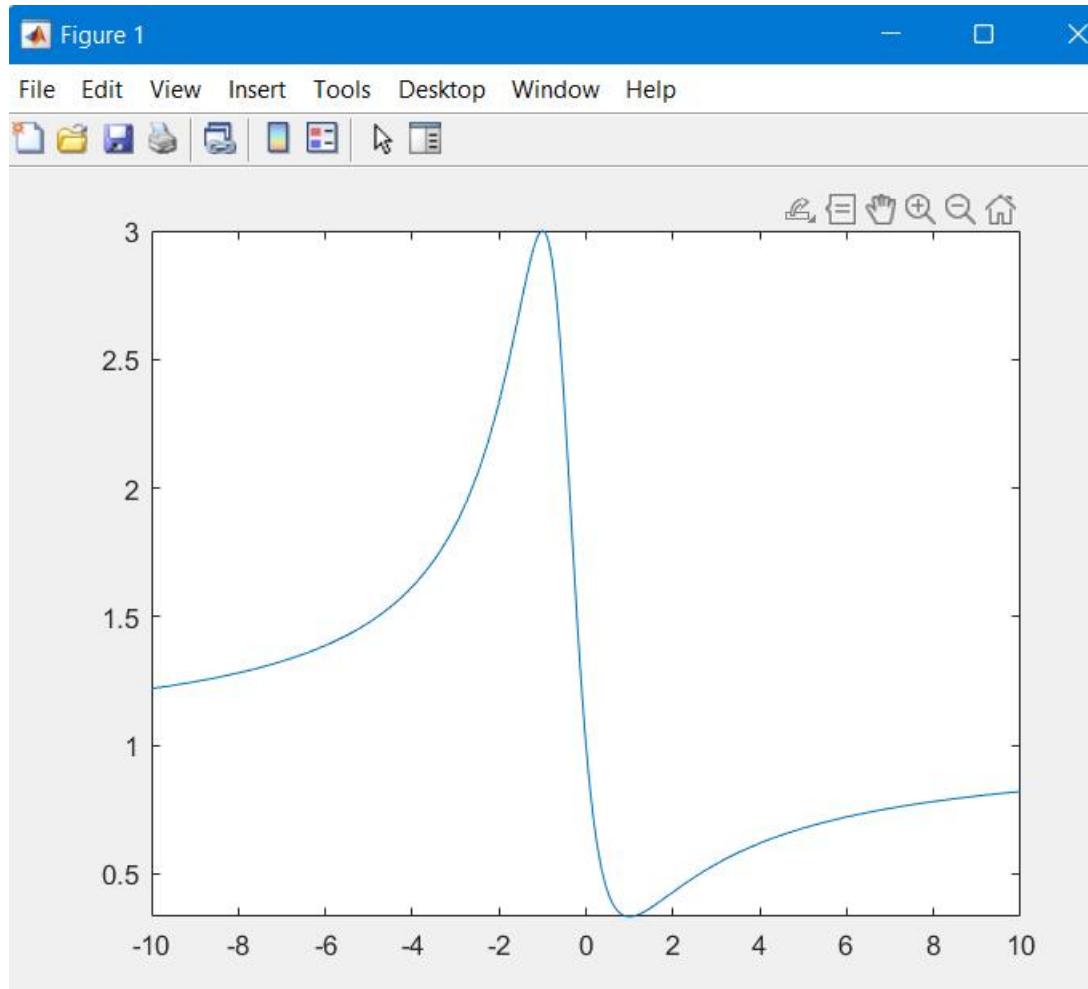
```
43 x1=-4:0.1:4;
44 x2 = -2.7:0.1:2.7;
45 f1 = 0.6*x1.^5-5*x1.^3+9*x1+2;
46 f2= 0.6*x2.^5-5*x2.^3+9*x2+2;
47 plot(x1,f1,x2,f2)
48 grid on
```





Question 5

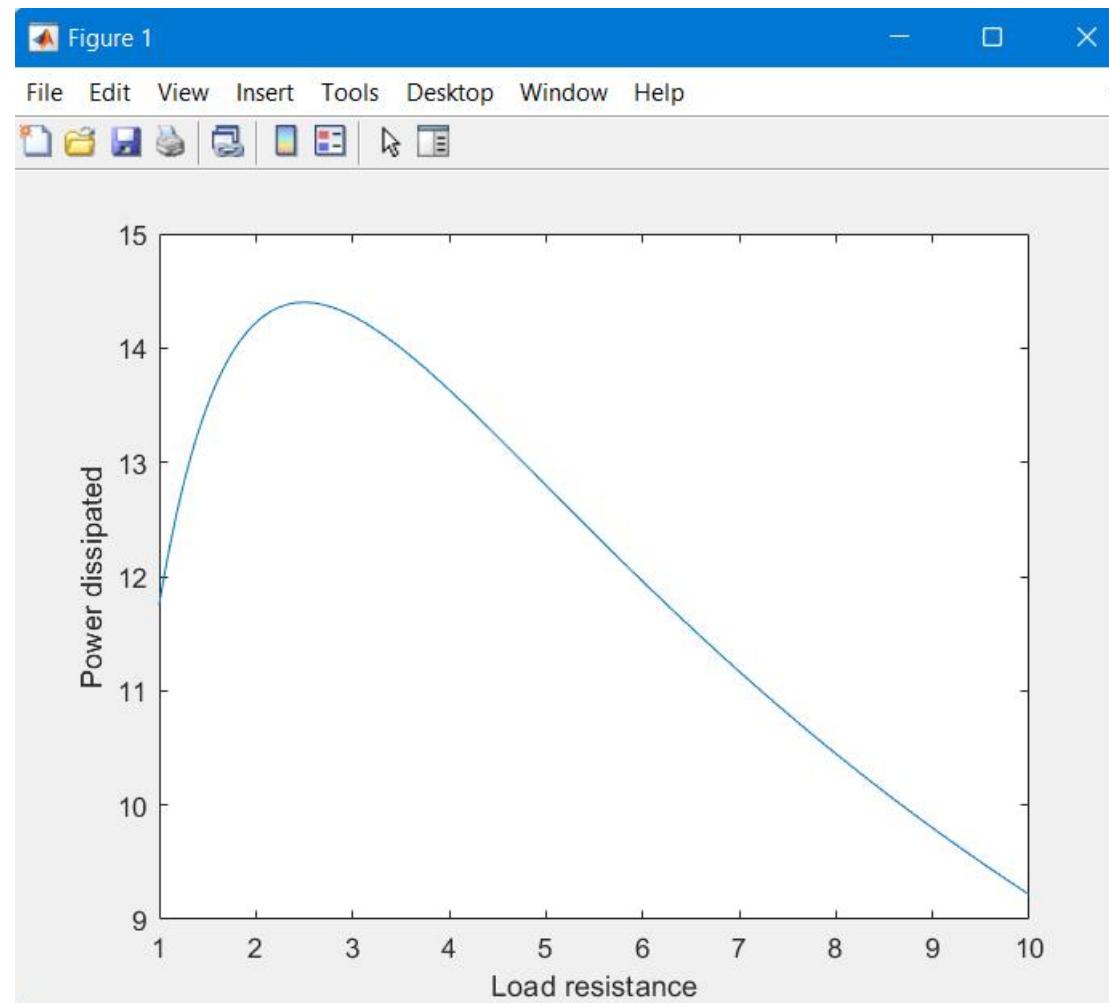
```
50 f=@(x)(x^2-x+1)/(x^2+x+1);  
51 l=[-10 10];  
52 fplot(f,l)
```





Question 6

```
54 RL = 1:0.01:10;
55 Vs = 12;
56 Rs = 2.5;
57 P = (Vs^2*RL)./(RL+Rs).^2;
58 plot(RL,P)
59 xlabel('Load resistance')
60 ylabel('Power dissipated')
```



MATLAB

Unit 5-Lecture 15

BTech (CSBS) -Semester VII

6 September 2022, 09:35AM



Introduction to programming

- 1) Introduction,
- 2) M-File Scripts,
- 3) script side-effects,
- 4) M-File functions,
- 5) anatomy of a M- File function,
- 6) input and output arguments,
- 7) input to a script file,
- 8) output commands.



Algorithm

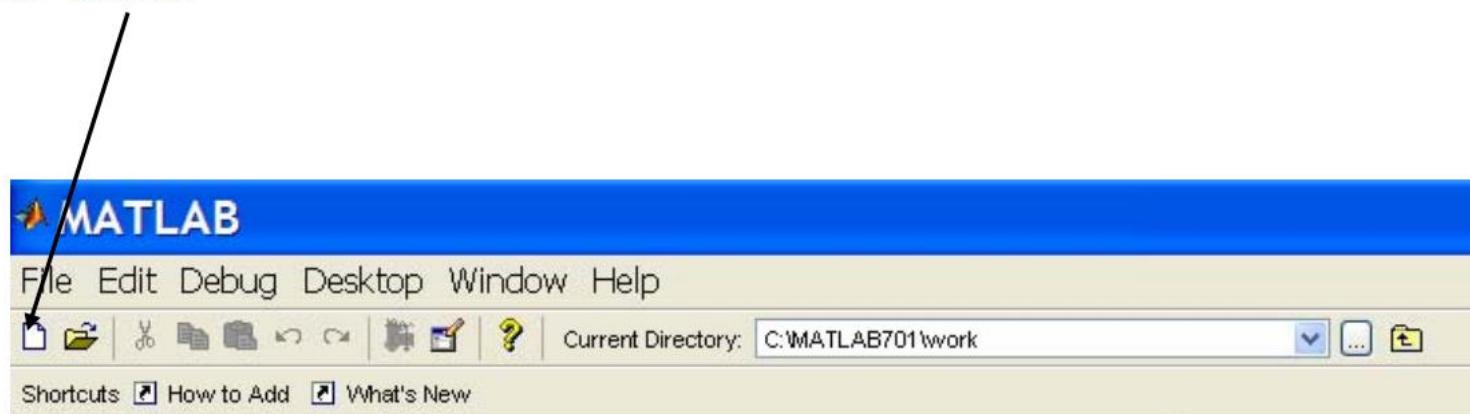
- An **algorithm** is a sequence of steps needed to solve a problem.
- In a **modular** approach, problem is broken into separate steps and then it is refined until results are manageable.
- The basic algorithm involves 3 steps:
 - Get the input: eg. the radius
 - Calculate the result: eg. the area
 - Display the output

This is known as ***top to down*** design



MATLAB script

- Scripts are
 - collection of commands executed in sequence
 - written in the MATLAB editor
 - saved as MATLAB files (.m extension)
- To create an MATLAB file from command-line
 - » `edit helloWorld.m`
- or click





Script-editor

Line numbers

MATLAB file path

* Means that it's not saved

Real-time error check

Debugging tools

```
Editor - C:\Documents and Settings\Danilo\My Documents\MATLAB\coinToss.m*  
File Edit Text Go Cell Tools Debug Desktop Window Help  
1 % coinToss.m  
2 % a script that flips a fair coin and displays the output  
3  
4 if rand<0.5 % if a random number is less than .5 say heads  
5 disp('HEADS');  
6 else % if greater than 0.5 say tails  
7 disp('TAILS');  
8 end
```

Possible breakpoints

Courtesy of The MathWorks, Inc. Used with permission.



Script-editor

- **COMMENT!**
 - Anything following a **%** is seen as a comment
 - The first contiguous comment becomes the script's help file
 - Comment thoroughly to avoid wasting time later
- Note that scripts are somewhat static, since there is no input and no explicit output
- All variables created and modified in a script exist in the workspace even after it has stopped running



Script-exercise

```
% This is a MATLAB script file.  
% It has been saved as "g13.m".  
  
load g13.dat; %Load data file  
voltage = g13( :, 4); %Extract volts vector  
time = .005*[1:length(voltage)]; %Create time vector  
plot (time, voltage) %Plot volts vs time  
xlabel ('Time in Seconds') % Label x axis  
ylabel ('Voltage') % Label y axis  
title ('Bike Strain Gage Voltage vs Time')  
grid %Put a grid on graph
```



Script-Question

Write a script to calculate the circumference of circle ($C=2\pi r$).
Comment the script.



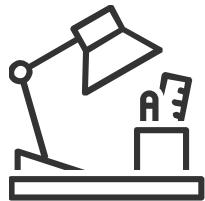
Documentation

circlescript.m

```
% This script calculates the area of a circle  
  
% First the radius is assigned  
radius = 5  
  
% The area is calculated based on the radius  
area = pi * (radius ^2)
```

```
>> help circlescript  
This script calculates the area of a circle
```

The first comment at the beginning of the script describes what the script does; this is sometimes called a ***block comment***. Then, throughout the script, comments describe different parts of the script (not usually a comment for every line, however!). Comments don't affect what a script does, so the output from this script would be the same as for the previous version.



Example of a script file

Let us write a script file to solve the following system of linear equations1 :

$$\begin{bmatrix} 5 & 2r & r \\ 3 & 6 & 2r - 1 \\ 2 & r - 1 & 3r \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 2 \\ 3 \\ 5 \end{Bmatrix}$$

or $Ax = b$. Clearly, A depends on the parameter r . We want to find the solution of the equation for various values of the parameter r . We also want to find, say, the determinant of matrix A in each case.



Example of a script file

```
%----- This is the script file 'solvex.m' -----
% It solves equation (4.1) for x and also calculates det(A).

A = [5 2*r r; 3 6 2*r-1; 2 r-1 3*r]; % create matrix A
b = [2;3;5]; % create vector b
det_A = det(A) % find the determinant
x = A\b % find x
```



Example of a script file

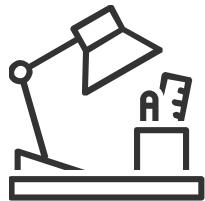
Let us now execute the script in MATLAB.

```
>> clear all % clear the workspace
>> r = 1; % specify a value of r
>> solvex % execute the script file solvex.m
```

```
det_A =
64
x =
-0.0312
0.2344
1.6875
>> who
```

This is the output. The values of the variables *det_A* and *x* appear on the screen because there is no semi-colon at the end of the corresponding lines in the script file.

Check the variables in the workspace.



Precautions

- NE VER name a script file the same as the name o.f a variable it computes.
- The name of a script file must begin with a letter. The rest of the characters may include digits and the underscore character.
- You may give long names but MATLAB will take only the first 19 character.

eg. proj ecL23C.m, cee213_hw5_1 .m but proj ect.23C.m and cee2 13_hw5.1.m are not valid names.



Function Files

A function file is also an M-file, like a script file, except that the variables in a function file are all local .A function file begins with a function definition line, which has a well-defined list of inputs and outputs. Without this line, the file becomes a script file. The syntax of the function definition line is as follows:

```
function [output variables] = function_name(input variables);
```



Examples of Function Files

Function Definition Line

```
function [rho,H,F] = motion(x,y,t);  
function [theta] = angleTH(x,y);  
function theta = THETA(x,y,z);  
function [] = circle(r);  
function circle(r);
```

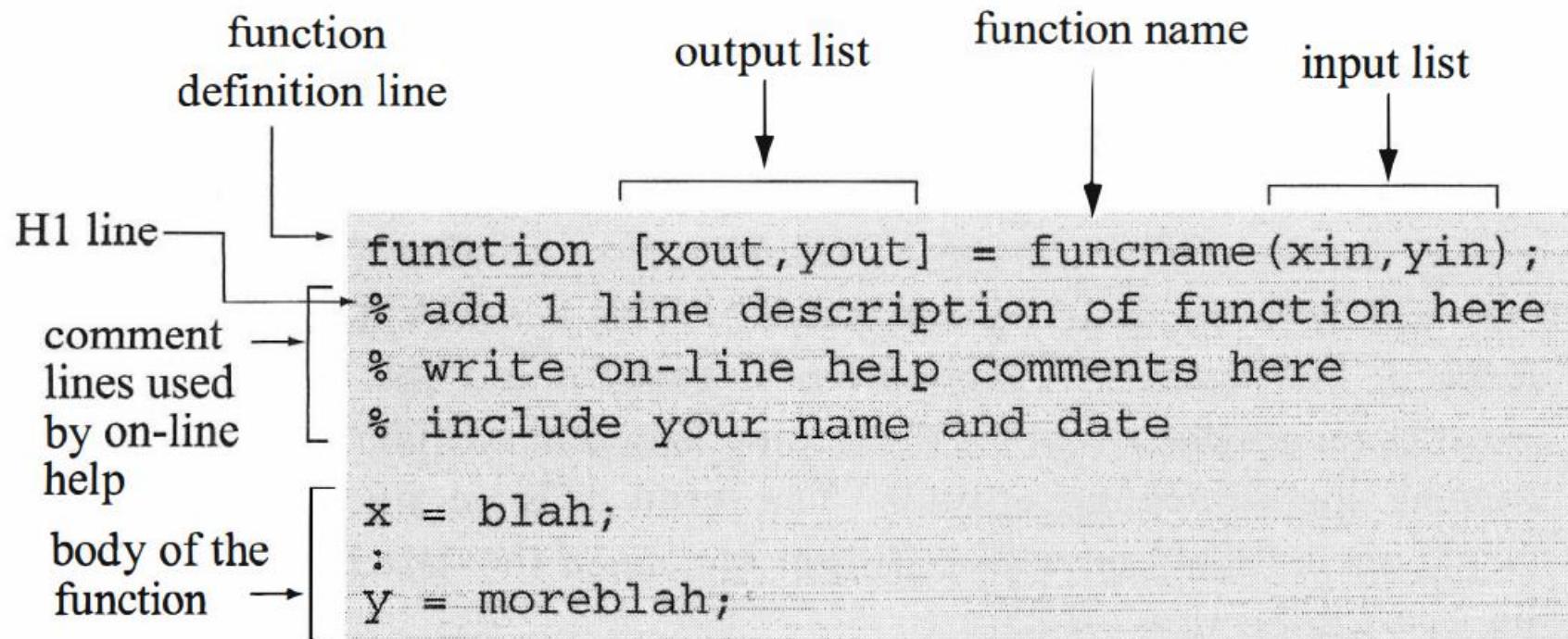
File Name

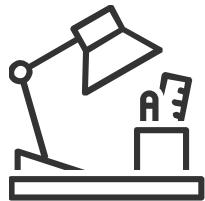
```
motion.m  
angleTH.m  
THETA.m  
circle.m  
circle.m
```

Caution: The first word in the function definition line, *function*, *must be typed in lowercase*. A common mistake is to type it as *Function*.



Anatomy of Function Files





Executing Function Files

This is the full syntax of calling a function. Both the output and input list are specified in the call. For example, if the function definition line of a function reads

```
function [rho ,H,F] = motion (x ,y,t) ;
```

then all the following commands represent legal call (execution) statements:



Executing Function Files

- `[r,angmom,force]=motion(xt,yt,time);` The input variables *xt*, *yt*, and *time* must be defined before executing this command.
- `[r,h,f]=motion(rx,ry,[0:100]);` The input variables *rx* and *ry* must be defined beforehand; the third input variable *t* is specified in the call statement.
- `[r,h,f]=motion(2,3.5,0.001);` All input values are specified in the call statement.
- `[radius,h]=motion(rx,ry);` Call with partial list of input and output. The third input variable must be assigned a default value inside the function if it is required in calculations. The output corresponds to the first two elements of the output list of `motion`.



Example

```
>> clear all

>> [detA, y] = solvexf(1); % take r=1 and execute solvexf.m

>> detA % display the value of detA

detA =
64

>> y % display the value of y

y =
-0.0312
0.2344
1.6875

>> who

Your variables are:

detA y
```

Values of *detA* and *y* will be automatically displayed if the semi-colon at the end of the function command is omitted.

Note that only *detA* and *y* are in the workspace; no *A*, *b*, or *x*.

MATLAB

Unit 5-Lecture 16

BTech (CSBS) -Semester VII

13 September 2022, 09:35AM



Introduction to programming

- 1) Introduction,
- 2) M-File Scripts,
- 3) script side-effects,
- 4) M-File functions,
- 5) anatomy of a M- File function,
- 6) input and output arguments,
- 7) input to a script file,
- 8) output commands.



Introduction to programming

1) What is an m-file?

An m-file, or script file, is a simple text file where you can place MATLAB commands. When the file is run, MATLAB reads the commands and executes them exactly as it would if you had typed each command sequentially at the MATLAB prompt. All m-file names must end with the extension '.m' (e.g. test.m). If you create a new m-file with the same name as an existing m-file, MATLAB will choose the one which appears first in the path order (type `help path` in the command window for more information). To make life easier, choose a name for your m-file which doesn't already exist. To see if a `filename.m` already exists, type `help filename` at the MATLAB prompt.

2) Why use m-files?

For simple problems, entering your requests at the MATLAB prompt is fast and efficient. However, as the number of commands increases or trial and error is done by changing certain variables or values, typing the commands over and over at the MATLAB prompt becomes tedious. M-files will be helpful and almost necessary in these cases.



Introduction to programming

3) How to create, save or open an m-file?

If you are using PC or Mac:

- To create an m-file, choose New from the File menu and select Script. This procedure brings up a text editor window in which you can enter MATLAB commands.
- To save the m-file, simply go to the File menu and choose Save (remember to save it with the '.m' extension). To open an existing m-file, go to the File menu and choose Open.

If you are using Unix:

- To create an m-file, use your favorite text editor (pico, nedit, vi, emacs, etc.) to create a file with .m extension (e.g. filename.m).

4) How to run the m-file?

- After the m-file is saved with the name filename.m in the current MATLAB folder or directory, you can execute the commands in the m-file by simply typing filename at the MATLAB command window prompt.
- If you don't want to run the whole m-file, you can just copy the part of the m-file that you want to run and paste it at the MATLAB prompt.



Script M-file: Creating M-file

Quite often we need to be able to calculate the value of a function $y=f(x)$ for any value of x . Obviously, it is not practical to change value of x each time. For this purpose MATLAB has a special type of M-file, called M-function.

> The following script M-file finds the value of the function at $y= \sin x + x^3$ at $x = 3$.

```
% exercisefscript.m  
  
x = 3  
  
y = sin(x) + x^3
```

> Run this M-file by typing the following in the *Command Window*:

```
exercisefscript
```

> Then update the M-file to find the value of $f(x)$ for $x = 4, 5, 6$.



Properties of Function M-Files

A MATLAB file of a special format that contains code with optional inputs and outputs is called function M-file.

Some advantages:

- Functions can be called from inside of other script and function M-files.
- Inputs allow variable values to be modified when calling the function (eg from the Command Window).
- Outputs can be assigned to other variables at the Command Window or within a separate M-file.

Disadvantages:

- A slight disadvantage with a function M-file is that you must follow the prescribed format, or else it won't run correctly. Once you get the hang of that, you will see they are often very useful.



Properties of Function M-Files

- The following function M-file finds the value of the function $f(x) = \sin x + x^3$ for any value of x . Type it in and save as **exercise1func.m** in your *Current Directory*

```
% <insert your name and the date here>

% exercisefunc.m

% input: x

% output: p, solved in terms of x

function p = exercisefunc(x) %Note special format!

p = sin(x) + x^3
```

- Call this M-file from the *Command Window* using the following command and then update it to find the value of for $p(x)$ for $x=3, 4, 5, 6$.

```
exercisefunc(3) % returns value for x = 3
```

- Consider the case, when the variable x is an array $[3 4 5 6]$. Modify your function M-file so, that it will be able to work with arrays.



Constructing Function M-files

Function M-files allow you to define, construct and store your own functions.

First, let's recall some of the in-built MATLAB functions you have already used. If you type command `help <name of the function>` , such as `help abs`, in the Command Window, MATLAB will print description and correct syntax of this function. These functions have a few things in common and a few differences.

Function: `y = abs(x)`

Description: Assigns the value x to y if x is non-negative, or $-x$ if x is negative.

Input: 1 number: x

Output: 1 number: either x or $-x$



Constructing Function M-files

Function: `y=rem(a,b)`

Description: Assigns the remainder of a/b to y

Input: 2 numbers: a is the numerator, b is the denominator

Output: 1 number: remainder of a/b

Function: `plot(xArray,yArray)`

Description: Plots a graph, given an array of x-coordinates, $xArray$, and an array of y-coordinates, $yArray$.

Input: 2 arrays of equal length: $xArray$, $yArray$.

Note: there can be many additional optional inputs.

Output: A graph.



Constructing Function M-files

Exercise: Determining the Inputs & Outputs

- For the following function (which you may recall from the first practical), use MATLAB to help you write the description, inputs and outputs.

Function: `round(a)`

Description: _____

Inputs: _____

Outputs: _____



Constructing Function M-files

> Exercise: Creating a Customised Random Number Generator

> Create a function M-file called `myRand.m` that outputs a random number between the inputted values of `minRand` and `maxRand` by adding code in the starred lines.

The MATLAB `rand` function returns a random value between 0 and 1. You will need to use this function as well as calculating the scale and offset values.

Example: If you want to find a number between 3 and 10, your scale is 7 and your offset is 3.

```
% <insert your name and the date here>

% myRand.m

% inputs: minRand, maxRand

% outputs: y, a random number with value between

% minRand & maxRand
```



Constructing Function M-files

```
function y = myRand(minRand, maxRand)

% **calculate the scale**

% **calculate the offset**

% **calculate y, using your scale, offset and %MATLAB's rand function**
```

Note: Save your function in the *Current Directory*, otherwise you will need to switch to the directory you saved your function in or type in a full path.



Constructing Function M-files

➤ Type in the following lines to call this function from your Command Window and verify that it works.

```
myRand(1,10)
```

```
myRand(100,100+1)
```

```
myRand(3,pi)
```

```
myRand(20)
```

```
myRand(20,1)
```



Steps to Create

Steps for creating function M-files

1. The function name and its M-file name must be identical (except that the M-file must end with .m).
2. The first executable statement must be a function declaration of the form

```
function <outputVariables> = <functionName>(<inputVariables>)
```

► One of the following function declarations has been taken from the MATLAB `rem.m` function M-file. Determine which one must be correct declaration:

```
function rem = remainder(x,y)  
  
function out = rem(x,y)  
  
out = function rem(x,y)  
  
function out(x,y) = rem(x,y)
```



Excercise

Exercise: Writing Function Declarations

Write down the `<outputVariables>`, `<functionName>` and `<InputVariables>` for the two function M-files from the previous exercises and verify they match the function declaration statement.

Function M-file 1: $p(x) = \sin x + x^3$

Function Name: _____

Output Variables: _____

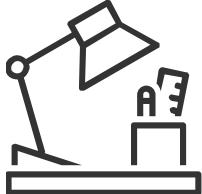
Input Variables: _____

Function M-file 2: Creating a Customised Random Number Generator

Function Name: _____

Output Variables: _____

Input Variables: _____



Exercise: Creating a Function M-file

Follow steps below to create a simple function M-file that computes and outputs the n^{th} power of 2, 2^n , where n is a number specified each time the function M-file is run.

- Q1. Create a blank M-file and save it as twoN.m
 - Q2. What should go at the top of every M-file? Add in header comments. This time, make sure you include the function description, inputs and outputs as well as your name and the date.
 - Q3. Type the function declaration into your file called twoN.m
 - a) replace <function Name> with twoN
 - b) decide upon an appropriate input variable name. In this case you may call it simply n.
 - c) decide upon an output variable name. The name y will be used in this case.
(In other examples you could use fn, f_n or another name of your choice as an output variable name.)
- Important: Every output variable must be assigned a value within your code.



Exercise: Creating a Function M-file

Now, you need to write your code. This function is pretty simple, so the code should only contain the following line: $y=2^n$

Note: if you have used different input/output variable names, you must change y to match your output variable name and n to match your input variable name.

Q5. OK, now you're ready to save and test your function M-file. After saving, make sure that your Current Directory matches the one you saved your M-file in.

Q6. Run the following lines in the Command Window to verify your function M-file works.

```
twoTo8 = twoN(8)
newNumber = twoN(5)
squareOfTwo = twoN(2)
twoN(9)
rootOfPower = twoN(5)^(1/2)
twoN %Why this does not work?
```



Exercise: Creating a Function M-file

Exercise: Writing your Own Function M-file

› Create a function M-file called `quadRoots` to find the roots of quadratic polynomials of the form $y=ax^2+bx+c$

Its inputs will be the coefficients a, b and c.

Its outputs will be the two roots, r_1 and r_2 and calculated by the formula:

$$r_1, r_2 = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Test `quadRoots` in the Command Window with the following three polynomials:

$$y = x^2 + 3x + 2 \text{ (ans } r_1=-1, r_2=-2\text{)}$$

$$y = x^2 + 6x + 10 \text{ (ans } r_1=-3+i, r_2=-3-i\text{)}$$

$$y = x^2 + 6x + 13 \text{ (ans } r_1=-3+2i, r_2=-3-2i\text{)}$$



Functions on Functions

Suppose a function $y = \text{demoFun}(x)$ has been defined in a function M-file `demoFun.m`

<code>fplot(@demoFun, [a b])</code>	Plots the function for $a \leq x \leq b$ without setting up arrays
<code>fzero(@demoFun, [a b])</code>	Finds one value of x for $\text{demoFun}(x) = 0$ provided that the signs of $\text{demoFun}(a)$ and $\text{demoFun}(b)$ are opposite.
<code>fzero(@demoFun, c)</code>	Finds one value of x for $\text{demoFun}(x) = 0$ by starting a search at $x = c$
<code>fminbnd(@demoFun, a, b)</code>	Finds the coordinates of a minimum point of $\text{demoFun}(x)$ at the interval $a \leq x \leq b$
<code>quadl(@demoFun, a, b)</code>	Finds an accurate value for $\int_a^b y(x) dx$ Note: <code>quadl</code> uses arrays, so therefore you must set your function up treating x as an array and so using the dot notation for operations.

Exercise: Using Functions

1. Create a function M-file called `myCubic.m` whose output is the value of the function $y = x^3 + 2x^2 - 5x - 8$

Input: `x`

Output: `y`

> Verify that this function works by checking that `myCubic(-5)=-58` and `myCubic(5)=142`

2. Create a script M-file called `cubicExercise.m` that contains the following five cells with code that:

a) plots `myCubic(x)` between the values of `[-5, 5]`,

b) finds a local minimum of `myCubic(x)`, located between `0` and `5`,

c) finds the all three roots of `myCubic(x)` using appropriate intervals

`[a, b]`

d) finds the value of the definite integral of `myCubic(x)` between `-5` and `5`.

Hint: You will need to use the array dot notation so you can use the `quadl` function to calculate the integral of `y`.

3. Make sure `cubicExercise.m` is marked up using cell formatting and publish it.

MATLAB

Unit 5-Lecture 17

BTech (CSBS) -Semester VII

16 September 2022, 09:35AM



Introduction to programming

- 1) Introduction,
- 2) M-File Scripts,
- 3) script side-effects,
- 4) M-File functions,
- 5) anatomy of a M- File function,
- 6) **input and output arguments,**
- 7) **input to a script file,**
- 8) **output commands.**



Kinds of M files

Script M-Files	Function M-Files
Do not accept input arguments or return output arguments	Can accept input arguments and return output arguments
Operate on data in the workspace	Internal variables are local to the function by default
Useful for automating a series of steps you need to perform many times	Useful for extending the MATLAB language for your application



Example

Factorial.m

```
n=10;  
factorial=1;  
for i=1:1:n  
    factorial=factorial*i;  
end
```



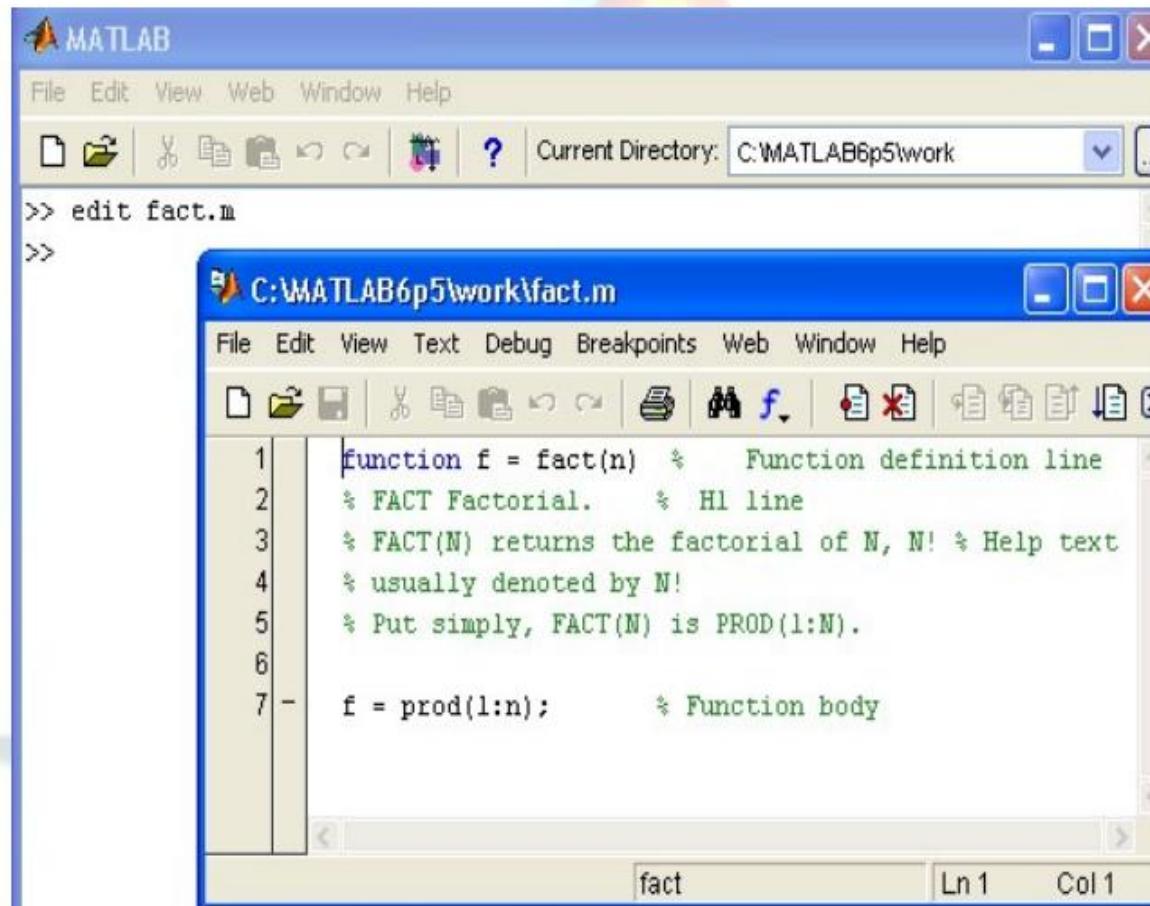
A screenshot of the MATLAB editor window showing the script `Factorial.m`. The window title is `C:\MATLAB6p5\work\Factorial.m`. The menu bar includes File, Edit, View, Text, Debug, Breakpoints, Web, Window, and Help. The toolbar contains various icons for file operations. The code editor displays the following MATLAB script:

```
1 | - n=10;  
2 | - factorial=1;  
3 | - for i=1:1:n  
4 | -     factorial=factorial*i;  
5 | - end
```



Accessing Text Editors

>> edit fact.m



The image shows the MATLAB desktop environment. The main window title is 'MATLAB'. The menu bar includes File, Edit, View, Web, Window, and Help. The toolbar contains various icons for file operations. The current directory is set to 'C:\MATLAB6p5\work'. A command window shows the command '>> edit fact.m' and its response '>>'. Below the command window is the MATLAB Editor window, titled 'C:\MATLAB6p5\work\fact.m'. The editor window has its own menu bar and toolbar. The code in 'fact.m' is as follows:

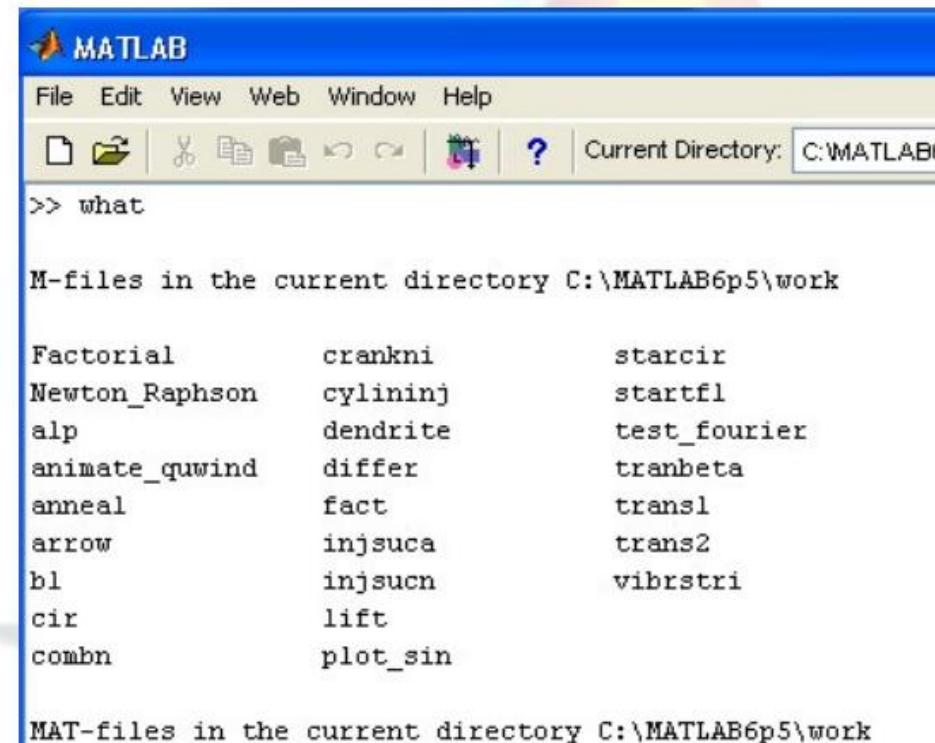
```
1 function f = fact(n) % Function definition line
2 % FACT Factorial. % Hl line
3 % FACT(N) returns the factorial of N, N! % Help text
4 % usually denoted by N!
5 % Put simply, FACT(N) is PROD(1:N).
6
7 f = prod(1:n); % Function body
```

The status bar at the bottom of the editor window shows 'fact' in the first field, 'Ln 1' in the second, and 'Col 1' in the third.



Listing files

>> what (List the names of the files in your current directory)



The image shows a screenshot of the MATLAB interface. The title bar says 'MATLAB'. The menu bar includes 'File', 'Edit', 'View', 'Web', 'Window', and 'Help'. The toolbar has icons for file operations. The current directory is set to 'C:\MATLAB6'. The command window shows the command 'what' entered. The output lists M-files and MAT-files in the current directory 'C:\MATLAB6p5\work'. The M-files listed are: Factorial, crankni, starcir, Newton_Raphson, cylininj, startfl, alp, dendrite, test_fourier, animate_quwind, differ, tranbeta, anneal, fact, transl, arrow, injsuca, trans2, bl, injsucn, vibrstri, cir, lift, and combin. The MAT-files listed are: plot_sin.

```
>> what

M-files in the current directory C:\MATLAB6p5\work

Factorial      crankni      starcir
Newton_Raphson cylininj      startfl
alp             dendrite      test_fourier
animate_quwind differ       tranbeta
anneal          fact         transl
arrow           injsuca      trans2
bl              injsucn      vibrstri
cir              lift
combn          plot_sin

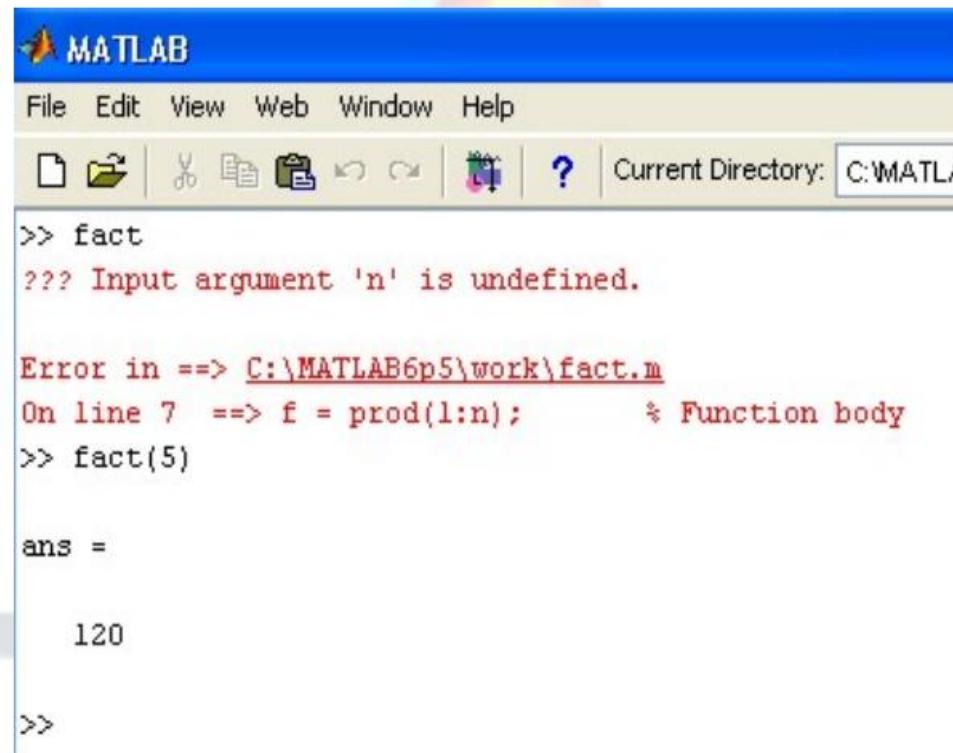
MAT-files in the current directory C:\MATLAB6p5\work
```



Calling function

Call the fact function

```
>> fact(5)
```



The screenshot shows the MATLAB desktop with the command window open. The window title is 'MATLAB'. The menu bar includes 'File', 'Edit', 'View', 'Web', 'Window', and 'Help'. The toolbar contains icons for file operations like Open, Save, and Print. The current directory is set to 'C:\MATLAB'. The command window displays the following text:

```
>> fact
??? Input argument 'n' is undefined.

Error in ==> C:\MATLAB6p5\work\fact.m
On line 7  ==> f = prod(1:n);      % Function body
>> fact(5)

ans =

120

>>
```



Some m-File Functions

Function	Description
run	Run script that is not on current path
type filename	lists the contents of the file given a full pathname
Edit fun	opens the file fun.m in a text editor
mfilename	Name of currently running M-file
namelengthmax	Return maximum identifier length
echo	Echoes the script file contents as they are executed



Some m-File Functions

Function	Description
input	Request user input
Disp (variablename)	Displays results without printing variable names
beep	Makes computer beep
eval	Interpret strings containing MATLAB expressions
feval	Evaluate function



Some m-File Functions

Function	Description
pause	Pauses and waits until user presses any keyboard key.
pause (n)	Pause (n) pauses for n seconds and then continues
waitForButtonpress	Pauses until user presses mouse button or any keyboard key
keyboard	Temporarily gives control to keyboard. <ul style="list-style-type: none">➤ The keyboard mode is terminated by executing the command RETURN➤ DBQUIT can also be used to get out of keyboard mode but in this case the invoking M-file is terminated.



Input Functions

The simplest input function in MATLAB is called **input**. The **input** function is used in an assignment statement. To call it, a string is passed that is the prompt that will appear on the screen, and whatever the user types will be stored in the variable named on the left of the assignment statement. For ease of reading the prompt, it is useful to put a colon and then a space after the prompt. For example,

```
>> rad = input ('Enter the radius: ')
Enter the radius: 5
rad =
5
```



Input Functions

If character or string input is desired, 's' must be added as a second argument to the **input** function:

```
>> letter = input('Enter a char: ', 's')
Enter a char: g
letter =
g
```

If the user enters only spaces or tabs before hitting the Enter key, they are ignored and an *empty string* is stored in the variable:

```
>> mychar = input('Enter a character: ', 's')
Enter a character:
mychar =
''
```



Input Functions

However, if blank spaces are entered before other characters, they are included in the string. In the next example, the user hits the space bar four times before entering "go." The **length** function returns the number of characters in the string.

```
>> mystr = input ('Enter a string: ', 's')
Enter a string:      go
mystr =
      go
>> length(mystr)
ans =
      6
```



Question

What would be the result if the user enters blank spaces after other characters? For example, the user here entered "xyz " (four blank spaces):

```
>> mychar = input('Enter chars: ', 's')
Enter chars: xyz
mychar =
xyz
```

Answer: The space characters would be stored in the string variable. It is difficult to see earlier, but is clear from the length of the string.

```
>> length(mychar)
ans =
7
```

The string can actually be seen in the Command Window by using the mouse to highlight the value of the variable; the xyz and four spaces will be highlighted.



Input

It is also possible for the user to type quotation marks around the string rather than including the second argument 's' in the call to the **input** function.

```
>> name = input ('Enter your name: ')
Enter your name: 'Stormy'
name =
Stormy
```

or

```
>> name = input ('Enter your name: ', 's')
Enter your name: 'Stormy'
name =
'Stormy'
>> length(name)
ans =
```



Input

```
>> num = input ('Enter a number: ')
Enter a number: t
Error using input
Undefined function or variable 't'.
```

```
Enter a number: 3
num =
3
```

MATLAB gave an *error message* and repeated the prompt. However, if *t* is the name of a variable, MATLAB will take its value as the input.

```
>> t = 11;
>> num = input ('Enter a number: ')
Enter a number: t
num =
11
```



Input

Separate **input** statements are necessary if more than one input is desired. For example,

```
>> x = input ('Enter the x coordinate: ');  
>> y = input ('Enter the y coordinate: ');
```

Normally in a script the results from **input** statements are suppressed with a semicolon at the end of the assignment statements.

It is also possible to enter a vector. The user can enter any valid vector, using any valid syntax such as square brackets, the colon operator, or functions such as **linspace**.

```
>> v = input ('Enter a vector: ')  
Enter a vector: [3 8 22]  
v =  
3 8 22
```



Output: `disp` and `fprintf`

Output statements display strings and/or the results of expressions, and can allow for *formatting* or customizing how they are displayed. The simplest output function in MATLAB is `disp`, which is used to display the result of an expression or a string without assigning any value to the default variable *ans*. However, `disp` does not allow formatting. For example,

```
>> disp('Hello')
Hello
```

```
>> disp(4 ^ 3)
64
```



Output: disp and fprintf

Formatted output can be printed to the screen using the **fprintf** function. For example,

```
>> fprintf('The value is %d, for sure!\n', 4 ^ 3)
The value is 64, for sure!
>>
```

To the **fprintf** function, first a string (called the *format string*) is passed that contains any text to be printed, as well as formatting information for the expressions to be printed. In this case, the **%d** is an example of format information.

The **%d** is sometimes called a *place holder* because it specifies where the value of the expression that is after the string, is to be printed. The character in the place holder is called the *conversion character*, and it specifies the type of value that is being printed. There are others, but what follows is a list of the simple place holders:

%d	integer (it stands for decimal integer)
%f	float (real number)
%c	character (one character)
%s	string of characters



Output: question

What do you think would happen if the newline character is omitted from the end of an **fprintf** statement?

Answer: Without it, the next prompt would end up on the same line as the output. It is still a prompt, and so an expression can be entered, but it looks messy as shown here.

```
>> fprintf('The value is %d, surely!', 4 ^ 3)
The value is 64, surely!>> 5 + 3
ans =
8
```

What would happen if you use the %d conversion character but you're trying to print a real number?

Answer: MATLAB will show the result using exponential notation

```
>> fprintf('%d\n', 1234567.89)
1.234568e+006
```

Note that with the **disp** function, however, the prompt will always appear on the next line:

```
>> disp('Hi ')
Hi
>>
```

Also, note that an ellipsis can be used after a string but not in the middle.

Note that if you want exponential notation, this is not the correct way to get it; instead, there are conversion characters that can be used. Use the **help** browser to see this option, as well as many others!



Output: question

How can you get a blank line in the output?

Answer: Have two newline characters in a row.

```
>> fprintf('The value is %d, \n\nOK! \n', 4 ^ 3)  
The value is 64,  
  
OK!
```

This also points out that the newline character can be anywhere in the string; when it is printed, the output moves down to the next line.

What do you think would happen if you tried to print 1234.5678 in a field width of 3 with 2 decimal places?

```
>> fprintf('%3.2f\n', 1234.5678)
```

Answer: It would print the entire 1234, but round the decimals to two places, that is,

1234.57

If the field width is not large enough to print the number, the field width will be increased. Basically, to cut the number off would give a misleading result, but rounding the decimal places does not change the number significantly.

MATLAB

Unit 5-Lecture 18

BTech (CSBS) -Semester VII

20 September 2022, 09:35AM



Introduction to programming

- 1) Introduction,
- 2) M-File Scripts,
- 3) script side-effects,
- 4) M-File functions,
- 5) anatomy of a M- File function,
- 6) **input and output arguments,**
- 7) input to a script file,
- 8) output commands.



Scripts with input and output

circleIO.m

```
% This script calculates the area of a circle
% It prompts the user for the radius

% Prompt the user for the radius and calculate
% the area based on that radius
fprintf('Note: the units will be inches.\n')
radius = input('Please enter the radius: ');
area = pi * (radius ^2);

% Print all variables in a sentence format
fprintf('For a circle with a radius of %.2f inches,\n',...
    radius)
fprintf('the area is %.2f inches squared\n',area)
```

Executing the script produces the following output:

```
>> circleIO
Note: the units will be inches.
Please enter the radius: 3.9
For a circle with a radius of 3.90 inches,
the area is 47.78 inches squared
```



Example

plotonepoint.m

```
% This is a really simple plot of just one point!  
  
% Create coordinate variables and plot a red '*'  
x = 11;  
y = 48;  
plot(x,y,'r*')  
  
% Change the axes and label them  
axis([9 12 35 55])  
xlabel('Time')  
ylabel('Temperature')  
  
% Put a title on the plot  
title('Time and Temp')
```



Example

plot2figs.m

```
% This creates 2 different plots, in 2 different
% Figure Windows, to demonstrate some plot features

clf
x = 1:5; % Not necessary
y1 = [2 11 6 9 3];
y2 = [4 5 8 6 2];
% Put a bar chart in Figure 1
figure(1)
bar(x,y1)
% Put plots using different y values on one plot
% with a legend
figure(2)
plot(x,y1,'k')
hold on
plot(x,y2,'ko')
grid on
legend('y1', 'y2')
```



Example

`sinncos.m`

```
% This script plots sin(x) and cos(x) in the same Figure Window
% for values of x ranging from 0 to 2*pi

clf
x = 0: 2*pi/40: 2*pi;
y = sin(x);
plot(x,y, 'ro')
hold on
y = cos(x);
plot(x,y, 'b+')
legend('sin', 'cos')
xlabel('x')
ylabel('sin(x) or cos(x)')
title('sin and cos on one graph')
```



Question

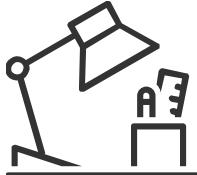
Sometimes files are not in the format that is desired. For example, a file "expresults.dat" has been created that has some experimental results, but the order of the values is reversed in the file:

```
4 53.4
3 44.3
2 50.0
1 55.5
```

How could we create a new file that reverses the order?

Answer: We can **load** from this file into a matrix, use the **flipud** function to "flip" the matrix up to down, and then **save** this matrix to a new file:

```
>> load expresults.dat
>> expresults
expresults =
    4.0000    53.4000
    3.0000    44.3000
    2.0000    50.0000
    1.0000    55.5000
>> correctorder = flipud(expresults)
correctorder =
    1.0000    55.5000
    2.0000    50.0000
    3.0000    44.3000
    4.0000    53.4000
>> save neworder.dat correctorder - ascii
```



Question

Could we pass a vector of radii to the *calcarea* function?

Answer: This function was written assuming that the argument was a scalar, so calling it with a vector instead would produce an error message:

```
>> calcarea(1:3)
Error using *
Inner matrix dimensions must agree.
```

```
Error in calcarea (line 6)
area = pi * rad * rad;
```

This is because the `*` was used for multiplication in the function, but `.*` must be used when multiplying vectors term by term. Changing this in the function would allow either scalars or vectors to be passed to this function:

calcarea.m

```
function area = calcareaii(rad)
% calcareaii returns the area of a circle
% The input argument can be a vector of radii
% Format: calcareaii(radVector)

area = pi * rad .* rad;
end
```

```
>> calcareaii(1:3)
ans =
3.1416 12.5664 28.2743
```

```
>> calcareaii(4)
ans =
50.2655
```

Note that the `.*` operator is only necessary when multiplying the radius vector by itself. Multiplying by `pi` is scalar multiplication, so the `.*` operator is not needed there. We could have also used:

```
area = pi * rad.^ 2;
```



Question

Nothing is technically wrong with the following function, but what about it does not make sense?

Answer; Why pass the third argument if it is not used?

fun.m

```
function out = fun(a,b,c)
out = a*b;
end
```



Practise Question

PRACTICE 3.1

Write a script to calculate the circumference of a circle ($C = 2\pi r$). Comment the script.

PRACTICE 3.2

Create a script that would prompt the user for a length, and then 'f' for feet or 'm' for meters, and store both inputs in variables. For example, when executed it would look like this (assuming the user enters 12.3 and then m):

```
Enter the length: 12.3
Is that f(eet) or m(eters)?: m
```

PRACTICE 3.3

Write a script to prompt the user separately for a character and a number, and print the character in a field width of 3 and the number left-justified in a field width of 8 with 3 decimal places. Test this by entering numbers with varying widths.

PRACTICE 3.4

Modify the script *plotonepoint* to prompt the user for the time and temperature, and set the axes based on these values.



Practise Question

PRACTICE 3.5

Modify the *plot2figs* script using the **axis** function so that all points are easily seen.

PRACTICE 3.6

Write a script that plots $\exp(x)$ and $\log(x)$ for values of x ranging from 0 to 3.5.

PRACTICE 3.7

Prompt the user for the number of rows and columns of a matrix, create a matrix with that many rows and columns of random integers, and write it to a file.



Practise Question

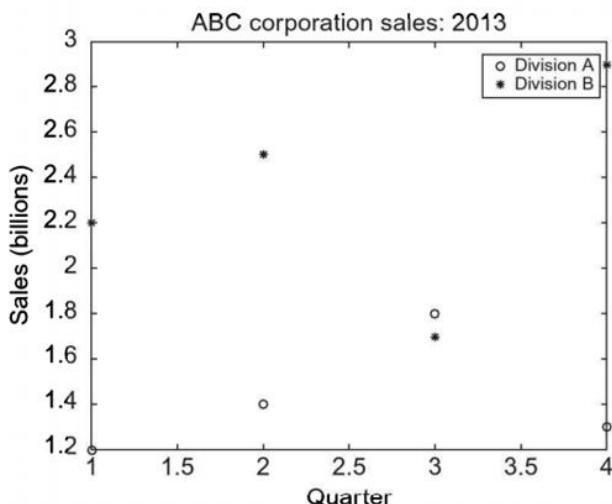
PRACTICE 3.8

The sales (in billions) for two separate divisions of the ABC Corporation for each of the four quarters of 2013 are stored in a file called "salesfigs.dat":

```
1.2 1.4 1.8 1.3
2.2 2.5 1.7 2.9
```

- First, create this file (just type the numbers in the Editor, and Save As "salesfigs.dat").
- Then, write a script that will

```
load the data from the file into a matrix
separate this matrix into 2 vectors.
create the plot seen in Fig. 3.7 (which uses black circles and stars as the plot symbols).
```





Practise Question

PRACTICE 3.9

Write a script that will prompt the user for the radius and height, call the function *conevol* to calculate the cone volume, and print the result in a nice sentence format. So, the program will consist of a script and the *conevol* function that it calls.

PRACTICE 3.10

For a project, we need some material to form a rectangle. Write a function *calcrectarea* that will receive the length and width of a rectangle in inches as input arguments, and will return the area of the rectangle. For example, the function could be called as shown, in which the result is stored in a variable and then the amount of material required is printed, rounded up to the nearest square inch.

```
>> ra = calcrectarea(3.1, 4.4)
ra =
13.6400

>> fprintf('We need %d sq in.\n', ceil(ra))
We need 14 sq in.
```

Unit 5: Introduction to programming

1. Using the top-down design approach, write an algorithm for making a sandwich.
2. Write a simple script that will calculate the volume of a hollow sphere,

$$\frac{4\pi}{3}(r_o^3 - r_i^3)$$

where r_i is the inner radius and r_o is the outer radius. Assign a value to a variable for the inner radius, and also assign a value to another variable for the outer radius. Then, using these variables, assign the volume to a third variable. Include comments in the script. Use **help** to view the comments in your script.

3. Write a statement that prompts the user for his/her favorite number.
4. Write a statement that prompts the user for his/her name.
5. Write an **input** statement that will prompt the user for a real number, and store it in a variable. Then, use the **fprintf** function to print the value of this variable using 2 decimal places.
6. Experiment, in the Command Window, with the **fprintf** function for real numbers. Make a note of what happens for each. Use **fprintf** to print the real number 12345.6789.
 - without specifying any field width
 - in a field width of 10 with 4 decimal places
 - in a field width of 10 with 2 decimal places
 - in a field width of 6 with 4 decimal places
 - in a field width of 2 with 4 decimal places
7. Experiment, in the Command Window, with the **fprintf** function for integers. Make a note of what happens for each. Use **fprintf** to print the integer 12345.
 - without specifying any field width
 - in a field width of 5
 - in a field width of 8
 - in a field width of 3
8. When would you use **disp** instead of **fprintf**? When would you use **fprintf** instead of **disp**?
9. Write a script called **echostring** that will prompt the user for a string, and will echo print the string in quotes:

```
>> echostring
Enter your string: hi there
Your string was: 'hi there'
```

10. If the lengths of two sides of a triangle and the angle between them are known, the length of the third side can be calculated. Given the lengths of two sides (b and c) of a triangle, and the angle between them α in degrees, the third side a is calculated as follows:

$$a^2 = b^2 + c^2 - 2 b c \cos(\alpha)$$

Write a script *thirdside* that will prompt the user and read in values for b , c , and α (in degrees), and then calculate and print the value of a , with 3 decimal places. The format of the output from the script should look exactly like this:

```
>> thirdside
Enter the first side: 2.2
Enter the second side: 4.4
Enter the angle between them: 50

The third side is 3.429
```

For more practice, write a function to calculate the third side, so the script will call this function.

11. Write a script that will prompt the user for a character, and will print it twice; once left-justified in a field width of 5, and again right-justified in a field width of 3.
12. Write a script *lumin* that will calculate and print the luminosity L of a star in Watts. The luminosity L is given by $L = 4\pi d^2 b$ where d is the distance from the sun in meters and b is the brightness in Watts/meters². Here is an example of executing the script:

```
>> lumin
This script will calculate the luminosity of a star.
When prompted, enter the star's distance from the sun
in meters, and its brightness in W/meters squared.

Enter the distance: 1.26e12
Enter the brightness: 2e-17
The luminosity of this star is 399007399.75 watts
```

13. A script “iotrace” has been written. Here’s what the desired output looks like:

```
>> iotrace
Please enter a number: 33
Please enter a character: x
Your number is 33.00
Your char is      x!
```

Fix this script so that it works as shown previously:

```
mynum = input ('Please enter a number:\n ');
mychar = input ('Please enter a character: ');
fprintf ('Your number is %.2f, mynum)
fprintf ('Your char is %c!\n', mychar)
```

14. Write a script that assigns values for the x coordinate and then y coordinate of a point, and then plot this using a green +.
15. Plot $\sin(x)$ for x values ranging from 0 to π (in separate Figure Windows):
using 10 points in this range
using 100 points in this range
16. When would it be important to use **legend** in a plot?
17. Why do we always suppress all assignment statements in scripts?
18. Atmospheric properties such as temperature, air density, and air pressure are important in aviation. Create a file that stores temperatures in degrees Kelvin at various altitudes. The altitudes are in the first column and the temperatures in the second. For example, it may look like this:
- | | |
|------|-----|
| 1000 | 288 |
| 2000 | 281 |
| 3000 | 269 |
19. Generate a random integer n , create a vector of the integers 1 through n in steps of 2, square them, and plot the squares.
20. Create a 3×6 matrix of random integers, each in the range of 50–100. Write this to a file called *randfile.dat*. Then, create a new matrix of random integers, but this time make it a 2×6 matrix of random integers, each in the range of 50–100. Append this matrix to the original file. Then, read the file (which will be to a variable called *randfile*) just to make sure that it worked!
21. A particular part is being turned on a lathe. The diameter of the part is supposed to be 20,000 mm. The diameter is measured every 10 min and the results are stored in a file called *partdiam.dat*. Create a data file to simulate this. The file will store the time in minutes and the diameter at each time. Plot the data.
22. Create a file called “*testtan.dat*” comprised of two lines with three real numbers on each line (some negative, some positive, in the –1 to 3 range). The file can be created from the Editor, or saved from a matrix. Then, **load** the file into a matrix and calculate the tangent of every element in the resulting matrix.

23. Write a function `calcrectarea` that will calculate and return the area of a rectangle. Pass the length and width to the function as input arguments.

Renewable energy sources such as biomass are gaining increasing attention. Biomass energy units include megawatt hours (MWh) and gigajoules (GJ). One MWh is equivalent to 3.6 GJ. For example, one cubic meter of wood chips produces 1 MWh.

24. Write a function *mwh_to_gj* that will convert MWh to GJ.
 25. List some differences between a script and a function.
 26. In quantum mechanics, the angular wavelength for a wavelength λ is defined as $\lambda/2\pi$. Write a function named *makeitangular* that will receive the wavelength as an input argument, and will return the angular wavelength.
 27. Write a *fives* function that will receive two arguments for the number of rows and columns, and will return a matrix with that size of all fives.
 28. Write a function *isdivby4* that will receive an integer input argument, and will return **logical 1** for **true** if the input argument is divisible by 4, or **logical false** if it is not.
 29. Write a function *isint* that will receive a number input argument *innum*, and will return 1 for **true** if this number is an integer, or 0 for **false** if not. Use the fact that *innum* should be equal to **int32(innum)** if it is an integer. Unfortunately, due to round-off errors, it should be noted that it is possible to get **logical 1** for **true** if the input argument is close to an integer. Therefore the output may not be what you might expect, as shown here.

```
>> isint(4)
ans =
1
>> isint(4.9999)
ans =
0
>> isint(4.99999999999999999999999999999999999999)
ans =
1
```

30. A Pythagorean triple is a set of positive integers (a, b, c) such that $a^2 + b^2 = c^2$. Write a function *ispythag* that will receive three positive integers (a, b, c) in that order and will return **logical 1** for **true** if they form a Pythagorean triple, or **0** for **false** if not.

31. A function can return a vector as a result. Write a function *vecout* that will receive one integer argument and will return a vector that increments from the value of the input argument to its value plus 5, using the colon operator. For example,

```
>> vecout(4)  
ans =  
4 5 6 7 8 9
```

32. Write a function that is called *pickone*, which will receive one input argument *x*, which is a vector, and will return one random element from the vector. For example,

```
>> pickone(4:7)
ans =
5
>> disp(pickone(-2:0))
-1
>> help pickone
pickone(x) returns a random element from vector x
```

33. The conversion depends on the temperature and other factors, but an approximation is that 1 in. of rain is equivalent to 6.5 in. of snow. Write a script that prompts the user for the number of inches of rain, calls a function to return the equivalent amount of snow, and prints this result. Write the function, as well!
34. In thermodynamics, the Carnot efficiency is the maximum possible efficiency of a heat engine operating between two reservoirs at different temperatures. The Carnot efficiency is given as

$$\eta = 1 - \frac{T_C}{T_H}$$

where T_C and T_H are the absolute temperatures at the cold and hot reservoirs, respectively. Write a script *carnot* that will prompt the user for the two reservoir temperatures in Kelvin, call a function to calculate the Carnot efficiency, and then print the corresponding Carnot efficiency to 3 decimal places. Also write the function.

35. Many mathematical models in engineering use the exponential function. The general form of the exponential decay function is:

$$y(t) = Ae^{-\tau t}$$

where A is the initial value at $t=0$, and τ is the time constant for the function. Write a script to study the effect of the time constant. To simplify the equation, set A equal to 1. Prompt the user for two different values for the time constant, and for beginning and ending values for the range of a *t* vector. Then, calculate two different *y* vectors using the above equation and the two time constants, and graph both exponential functions on the same graph within the range the user specified. Use a function to calculate *y*. Make one plot red. Be sure to label the graph and both axes. What happens to the decay rate as the time constant gets larger?

MATLAB

Unit 6-Lecture 20

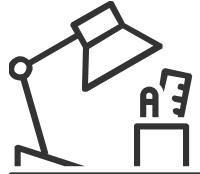
BTech (CSBS) -Semester VII

30 September 2022, 09:35AM



Control Flow and Operators

- 1) relational and logical operators
- 2) “if ... end” structure
- 3) “for ... end” loop
- 4) “while ... end” loop
- 5) other flow structures
- 6) operator precedence
- 7) saving output to a file



Relational operators

Relational operators compare the elements in two arrays and return logical true or false values to indicate where the relation holds.

<code>==</code>	Determine equality
<code>>=</code>	Determine greater than or equal to
<code>></code>	Determine greater than
<code><=</code>	Determine less than or equal to
<code><</code>	Determine less than
<code>~=</code>	Determine inequality
<code>isequal</code>	Determine array equality
<code>isequaln</code>	Determine array equality, treating NaN values as equal



Relational operators: Examples

Examples: If $x = [1 \ 5 \ 3 \ 7]$ and $y = [0 \ 2 \ 8 \ 7]$, then

$k = x < y$	results in $k = [0 \ 0 \ 1 \ 0]$	because $x_i < y_i$ for $i = 3$,
$k = x \leq y$	results in $k = [0 \ 0 \ 1 \ 1]$	because $x_i \leq y_i$ for $i = 3$ and 4 ,
$k = x > y$	results in $k = [1 \ 1 \ 0 \ 0]$	because $x_i > y_i$ for $i = 1$ and 2 ,
$k = x \geq y$	results in $k = [1 \ 1 \ 0 \ 1]$	because $x_i \geq y_i$ for $i = 1, 2$, and 4 ,
$k = x == y$	results in $k = [0 \ 0 \ 0 \ 1]$	because $x_i = y_i$ for $i = 4$, and
$k = x \sim= y$	results in $k = [1 \ 1 \ 1 \ 0]$	because $x_i \neq y_i$ for $i = 1, 2$, and 3 .



Logical operators

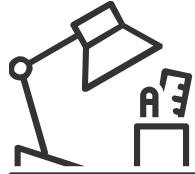
<code>&</code>	Find logical AND
<code>~</code>	Find logical NOT
<code> </code>	Find logical OR
<code>xor</code>	Find logical exclusive-OR
<code>all</code>	Determine if all array elements are nonzero or true
<code>any</code>	Determine if any array elements are nonzero
<code>false</code>	Logical 0 (false)
<code>find</code>	Find indices and values of nonzero elements
<code>islogical</code>	Determine if input is logical array
<code>logical</code>	Convert numeric values to logicals
<code>true</code>	Logical 1 (true)



Logical operators: Examples

Examples: For two vectors $x = [0 \ 5 \ 3 \ 7]$ and $y = [0 \ 2 \ 8 \ 7]$,

$m = (x > y) \& (x > 4)$ results in $m = [0 \ 1 \ 0 \ 0]$, because the condition is true only for x_2 ,
 $n = x \mid y$ results in $n = [0 \ 1 \ 1 \ 1]$, because either x_i or y_i is nonzero for $i = [2 \ 3 \ 4]$,
 $m = \sim(x \mid y)$ results in $m = [1 \ 0 \ 0 \ 0]$, which is the logical complement of $x \mid y$, and
 $p = \text{xor}(x, y)$ results in $p = [0 \ 0 \ 0 \ 0]$, because there is no such index i
for which x_i or y_i , but not both, is nonzero.



Logical operators: builtin functions

In addition to these logical operators, there are many useful built-in logical functions, such as:

<code>all</code>	true (= 1) if all elements of a vector are true, <i>Example:</i> <code>all(x<0)</code> returns 1 if each element of x is negative.
<code>any</code>	true (= 1) if any element of a vector is true, <i>Example:</i> <code>any(x)</code> returns 1 if any element of x is nonzero.
<code>exist</code>	true (= 1) if the argument (a variable or a function) exists,
<code>isempty</code>	true (= 1) for an empty matrix,
<code>isinf</code>	true for all infinite elements of a matrix,
<code>isfinite</code>	true for all finite elements of a matrix,
<code>isnan</code> ³	true for all elements of a matrix that are not a number (NaN), and
<code>find</code>	finds indices of nonzero elements of a matrix. <i>Examples:</i> <code>find(x)</code> returns [2 3 4] for $x=[0 \ 2 \ 5 \ 7]$ and <code>[r,c]=find(A>100)</code> returns the row and column indices i and j of A , in vectors r and c , for which $A_{ij} > 100$.

To complete this list of logical functions, we just mention `isreal`, `issparse`, `isstr`, and `ischar`.



Elementary math functions

Trigonometric functions

sin, sind	sine,	sinh	hyperbolic sine,
asin, asind	inverse sine,	asinh	inverse hyperbolic sine,
cos, cosd	cosine,	cosh	hyperbolic cosine,
acos,acosd	inverse cosine,	acosh	inverse hyperbolic cosine,
tan, tand	tangent,	tanh	hyperbolic tangent,
atan, atand	inverse tangent,	atanh	inverse hyperbolic tangent,
atan2	four-quadrant \tan^{-1} ,		
sec, secd	secant,	sech	hyperbolic secant,
asec, asecd	inverse secant,	asech	inverse hyperbolic secant,
csc, cscd	cosecant,	csch	hyperbolic cosecant,
acsc, acscd	inverse cosecant,	acsch	inverse hyperbolic cosecant,
cot, cotd	cotangent,	coth	hyperbolic cotangent,
acot, acotd	inverse cotangent, and	acoth	inverse hyperbolic cotangent.



Elementary math functions

The angles given to these functions as arguments must be in *radians* for `sin`, `cos`, etc., and in *degrees* for `sind`, `cosd`, etc. Thus, `sin(pi/2)` and `sind(90)` produce the same result. All of these functions, except `atan2`, take a single scalar, vector, or matrix as input argument. The function `atan2` takes two input arguments, `atan2(y,x)`, and produces the four-quadrant inverse tangent such that $-\pi \leq \tan^{-1} \frac{y}{x} \leq \pi$. This gives the angle a rectangular to polar conversion.

Examples: If $q=[0 \ pi/2 \ pi]$, $x=[1 \ -1 \ -1 \ 1]$, and $y=[1 \ 1 \ -1 \ -1]$, then

`sin(q)` gives $[0 \ 1 \ 0]$,

`sinh(q)` gives $[0 \ 2.3013 \ 11.5487]$,

`atan(y./x)` gives $[0.7854 \ -0.7854 \ 0.7854 \ -0.7854]$, and

`atan2(y,x)` gives $[0.7854 \ 2.3562 \ -2.3562 \ -0.7854]$.



Exponential Functions

exp

exponential,

Example: `exp(A)` produces a matrix with elements $e^{(A_{ij})}$.

So how do you compute e^A ? See the next section.

log

natural logarithm,

Example: `log(A)` produces a matrix with elements $\ln(A_{ij})$.

log10

base 10 logarithm,

Example: `log10(A)` produces a matrix with elements $\log_{10}(A_{ij})$.

sqrt

square root,

Example: `sqrt(A)` produces a matrix with elements $\sqrt{A_{ij}}$.

But what about \sqrt{A} ? See the next section.

nthroot

real n th root of real numbers,

Example: `nthroot(A, 3)` produces a matrix with elements $\sqrt[3]{A_{ij}}$.



Complex Functions

<code>abs</code>	absolute value, <i>Example</i> : <code>abs(A)</code> produces a matrix of absolute values $ A_{ij} $.
<code>angle</code>	phase angle, <i>Example</i> : <code>angle(A)</code> gives the phase angles of complex A .
<code>complex</code>	constructs complex numbers from given real and imaginary parts, <i>Example</i> : <code>complex(A,B)</code> produces $A + Bi$.
<code>conj</code>	complex conjugate, <i>Example</i> : <code>conj(A)</code> produces a matrix with elements \bar{A}_{ij} .
<code>imag</code>	imaginary part, <i>Example</i> : <code>imag(A)</code> extracts the imaginary part of A .
<code>real</code>	real part, <i>Example</i> : <code>real(A)</code> extracts the real part of A .



Round off Functions

fix

round toward 0,

Example: `fix([-2.33 2.66]) = [-2 2].`

floor

round toward $-\infty$,

Example: `floor([-2.33 2.66]) = [-3 2].`

ceil

round toward $+\infty$,

Example: `ceil([-2.33 2.66]) = [-2 3].`

mod

modulus after division; `mod(a,b)` is the same as `a-floor(a./b)*b`,

Example: `mod(26,5) = 1` and `mod(-26,5) = 4`.

round

round toward the nearest integer,

Example: `round([-2.33 2.66]) = [-2 3].`

rem

remainder after division, `rem(a,b)` is the same as `a-fix(a./b)*b`,

Example: If `a=[-1.5 7]`, `b=[2 3]`, then `rem(a,b) = [-1.5 1]`.

sign

signum function,

Example: `sign([-2.33 2.66]) = [-1 1].`



Matrix Functions

- | | |
|-----------------------|---|
| <code>expm(A)</code> | finds the exponential of matrix A , e^A , |
| <code>logm(A)</code> | finds $\log(A)$ such that $A = e^{\log(A)}$, and |
| <code>sqrtm(A)</code> | finds \sqrt{A} . |

MATLAB

Unit 6-Lecture 21

Operators Precedence

BTech (CSBS) -Semester VII

30 September 2022, 09:35AM



Control Flow and Operators

- 1) relational and logical operators
- 2) “if ... end” structure
- 3) “for ... end” loop
- 4) “while ... end” loop
- 5) other flow structures
- 6) operator precedence
- 7) saving output to a file



Operators Precedence

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

MATLAB

Unit 6-Lecture 22

“if ... end” structure

BTech (CSBS) -Semester VII

4 October 2022, 09:35AM



Control Flow and Operators

- 1) relational and logical operators
- 2) “if ... end” structure
- 3) “for ... end” loop
- 4) “while ... end” loop
- 5) other flow structures
- 6) operator precedence
- 7) saving output to a file



The “IF” statement

The if statement chooses whether another statement, or group of statements, is executed or not. The general form of the if statement is:

```
if condition
    action
end
```

For example, the following if statement checks to see whether the value of a variable is negative. If it is, the value is changed to a zero; otherwise, nothing is changed.

```
if num < 0
    num = 0
end
```



The “IF” statement: Example

sqrtifexamp.m

```
% Prompt the user for a number and print its sqrt  
  
num = input ('Please enter a number: ');\n\n% If the user entered a negative number, change it  
if num < 0  
    num = 0;  
end  
fprintf ('The sqrt of %.1f is %.1f\n', num, sqrt (num))
```

Here are two examples of running this script:

```
>> sqrtifexamp  
Please enter a number: -4.2  
The sqrt of 0.0 is 0.0
```

```
>> sqrtifexamp  
Please enter a number: 1.44  
The sqrt of 1.4 is 1.2
```



The “IF” statement: Example

`sqrtifexampi.m`

```
% Prompt the user for a number and print its sqrt  
  
num = input('Please enter a number: ');\n\n% If the user entered a negative number, tell  
% the user and change it  
if num < 0  
    disp('OK, we''ll use the absolute value')  
    num = abs(num);  
end  
fprintf('The sqrt of %.1f is %.1f\n', num, sqrt(num))
```

```
>> sqrtifexampi  
Please enter a number: -25  
OK, we'll use the absolute value  
The sqrt of 25.0 is 5.0
```



The “IF” statement: Question

Assume that we want to create a vector of increasing integer values from *mymin* to *mymax*. We will write a function *createvec* that receives two input arguments, *mymin* and *mymax*, and returns a vector with values from *mymin* to *mymax* in steps of one. First, we would make sure that the value of *mymin* is less than the value of *mymax*. If not, we would need to exchange their values before creating the vector. How would we accomplish this?

Answer: To exchange values, a third variable, a temporary variable, is required. For example, let's say that we have two variables, *a* and *b*, storing the values:

```
a = 3;  
b = 5;
```

To exchange values, we could *not* just assign the value of *b* to *a*, as follows:

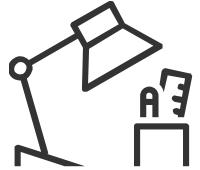
```
a = b;
```

If that were done, then the value of *a* (the 3), is lost! Instead, we need to assign the value of *a* first to a **temporary variable** so that the value is not lost. The algorithm would be:

- assign the value of *a* to *temp*
- assign the value of *b* to *a*
- assign the value of *temp* to *b*

```
>> temp = a;  
>> a = b  
a =  
5  
>> b = temp  
b =  
3
```

Now, for the function. An if statement is used to determine whether or not the exchange is necessary.



The “IF” statement: Question

createvec.m

```
function outvec = createvec (mymin, mymax)
% createvec creates a vector that iterates from a
% specified minimum to a maximum
% Format of call: createvec (minimum, maximum)
% Returns a vector

% If the "minimum" isn't smaller than the "maximum",
% exchange the values using a temporary variable
if mymin > mymax
    temp = mymin;
    mymin = mymax;
    mymax = temp;
end

% Use the colon operator to create the vector
outvec = mymin:mymax;
end
```

Examples of calling the function are:

```
>> createvec (4, 6)
ans =
    4    5    6
```

```
>> createvec (7, 3)
ans =
    3    4    5    6    7
```



“if-else” Statement

The if statement chooses whether or not an action is executed. Choosing between two actions, or choosing from among several actions, is accomplished using if-else, nested if-else, and switch statements.

The if-else statement is used to choose between two statements, or sets of statements. The general form is:

```
if condition
    action1
else
    action2
end
```



“if-else” Statement: Example

For example, to determine and print whether or not a random number in the range from 0 to 1 is less than 0.5, an if-else statement could be used:

```
if rand < 0.5
    disp('It was less than .5!')
else
    disp('It was not less than .5!')
end
```



“if-else” Statement: Example

One application of an if-else statement is to check for errors in the inputs to a script (this is called *error-checking*). For example, an earlier script prompted the user for a radius, and then used that to calculate the area of a circle. However, it did not check to make sure that the radius was valid (e.g., a positive number). Here is a modified script that checks the radius:

checkradius.m

```
% This script calculates the area of a circle
% It error-checks the user's radius
radius = input('Please enter the radius: ');
if radius <= 0
    fprintf('Sorry; %.2f is not a valid radius\n',radius)
else
    area = calcarea(radius);
    fprintf('For a circle with a radius of %.2f,',radius)
    fprintf(' the area is %.2f\n',area)
end
```



“if-else” Statement: Example

Examples of running this script when the user enters invalid and then valid radii are shown as follows:

```
>> checkradius
```

```
Please enter the radius: -4
```

```
Sorry; -4.00 is not a valid radius
```

```
>> checkradius
```

```
Please enter the radius: 5.5
```

```
For a circle with a radius of 5.50, the area is 95.03
```

```
>> if radius <= 0
```

```
    error('Sorry; %.2f is not a valid radius\n', radius)
```

```
end
```

```
Sorry; -4.00 is not a valid radius
```



“if-else” Statement: Example

Examples of running this script when the user enters invalid and then valid radii are shown as follows:

```
>> checkradius
```

```
Please enter the radius: -4
```

```
Sorry; -4.00 is not a valid radius
```

```
>> checkradius
```

```
Please enter the radius: 5.5
```

```
For a circle with a radius of 5.50, the area is 95.03
```

```
>> if radius <= 0
```

```
    error('Sorry; %.2f is not a valid radius\n', radius)
```

```
end
```

```
Sorry; -4.00 is not a valid radius
```



Nested “if-else” Statement

```
y = 1    if    x < -1
y = x2  if    -1 ≤ x ≤ 2
y = 4    if    x > 2
```

The value of y is based on the value of x , which could be in one of three possible ranges. Choosing which range could be accomplished with three separate if statements, is as follows:

```
if x < -1
    y = 1;
end
if x >= -1 && x <=2
    y = x2;
end
if x > 2
    y = 4;
end
```



Nested “if-else” Statement

Above example could also be written as below:

```
if x < -1
    y = 1;
else
    % If we are here, x must be >= -1
    % Use an if-else statement to choose
    % between the two remaining ranges
    if x <= 2
        y = x^2;
    else
        % No need to check
        % If we are here, x must be > 2
        y = 4;
    end
end
```



Nested “if-else” Statement

THE PROGRAMMING CONCEPT

In some programming languages, choosing from multiple options means using nested if-else statements. However, MATLAB has another method of accomplishing this using the elseif clause.

THE EFFICIENT METHOD

To choose from among more than two actions, the elseif clause is used. For example, if there are n choices (where $n > 3$ in this example), the following general form would be used:



Nested “if-else” Statement

THE EFFICIENT METHOD—CONT'D

```
if condition1
    action1
elseif condition2
    action2
elseif condition3
    action3
% etc: there can be many of these
else
    actionn    % the nth action
end
```

The actions of the if, elseif, and else clauses are naturally bracketed by the reserved words if, elseif, else, and end.



“elseif” Statement

For example, the previous example could be written using the elseif clause, rather than nesting if-else statements:

```
if x < -1
    y = 1;
elseif x <= 2
    y = x^2;
else
    y = 4;
end
```



“elseif” Statement

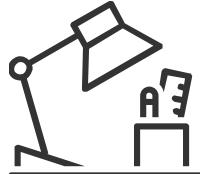
This could be implemented in a function that receives a value of x and returns the corresponding value of y :

`calcY.m`

```
function y = calcY(x)
% calcY calculates y as a function of x
% Format of call: calcY(x)
% y = 1          if x < -1
% y = x^2        if -1 <= x <= 2
% y = 4          if x > 2

if x < -1
    y = 1;
elseif x <= 2
    y = x^2;
else
    y = 4;
end
end
```

```
>> x = 1.1;
>> y = calcY(x)
y =
1.2100
```



“elseif” Statement

How could you write a function to determine whether an input argument is a scalar, a vector, or a matrix?

Answer: To do this, the **size** function can be used to find the dimensions of the input argument. If both the number of rows and columns is equal to 1, then the input argument is scalar. If, however, only one dimension is 1, the input argument is a vector (either a row or column vector). If neither dimension is 1, the input argument is a matrix. These three options can be tested using a nested **if-else** statement. In this example, the word ‘scalar,’ ‘vector,’ or ‘matrix’ is returned from the function.

Note that there is no need to check for the last case: if the input argument isn’t a scalar or a vector, it must be a matrix! Examples of calling this function are:

```
>> findargtype(33)
ans =
scalar

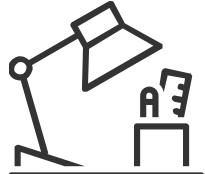
>> disp(findargtype(2:5))
vector

>> findargtype(zeros(2, 3))
ans =
matrix
```

findargtype.m

```
function outtype = findargtype(inputarg)
% findargtype determines whether the input
% argument is a scalar, vector, or matrix
% Format of call: findargtype(inputArgument)
% Returns a string

[r c] = size(inputarg);
if r == 1 && c == 1
    outtype = 'scalar';
elseif r == 1 || c == 1
    outtype = 'vector';
else
    outtype = 'matrix';
end
end
```



“elseif” Statement

letgrade.m

```
function grade = letgrade(quiz)
% letgrade returns the letter grade corresponding
% to the integer quiz grade argument
% Format of call: letgrade(integerQuiz)
% Returns a character

% First, error-check
if quiz < 0 || quiz > 10
    grade = 'X';

% If here, it is valid so figure out the
% corresponding letter grade
elseif quiz == 9 || quiz == 10
    grade = 'A';
elseif quiz == 8
    grade = 'B';
elseif quiz == 7
    grade = 'C';
elseif quiz == 6
    grade = 'D';
else
    grade = 'F';
end
end
```

Three examples of calling this function are:

```
>> quiz = 8;
>> lettergrade = letgrade(quiz)
lettergrade =
B

>> quiz = 4;
>> letgrade(quiz)
ans =
F

>> lg = letgrade(22)
lg =
X
```



Practise question

PRACTICE 4.1

Write an `if` statement that would print “Hey, you get overtime!” if the value of a variable `hours` is greater than 40. Test the `if` statement for values of `hours` less than, equal to, and greater than 40. Will it be easier to do this in the Command Window or in a script?

PRACTICE 4.3

Modify the function `findargtype` to return either ‘scalar,’ ‘row vector,’ ‘column vector,’ or ‘matrix,’ depending on the input argument.

PRACTICE 4.4

Modify the original function `findargtype` to use three separate `if` statements instead of a nested `if-else` statement.



Practise question

PRACTICE 4.2

Write a script *printsindegorrad* that will:

- prompt the user for an angle
- prompt the user for (r)adians or (d)egrees, with radians as the default
- if the user enters 'd,' the **sind** function will be used to get the sine of the angle in degrees; otherwise, the **sin** function will be used. Which sine function to use will be based solely on whether the user entered a 'd' or not ('d' means degrees, so **sind** is used; otherwise, for any other character the default of radians is assumed so **sin** is used)
- print the result.

Here are examples of running the script:

```
>> printsindegorrad
Enter the angle: 45
(r)adians (the default) or (d)egrees: d
The sin is 0.71
```

```
>> printsindegorrad
Enter the angle: pi
(r)adians (the default) or (d)egrees: r
The sin is 0.00
```

MATLAB

Unit 6-Lecture 23

“for ... end” structure

BTech (CSBS) -Semester VII

7 October 2022, 09:35AM



Control Flow and Operators

- 1) relational and logical operators
- 2) “if ... end” structure
- 3) “for ... end” loop
- 4) “while ... end” loop
- 5) other flow structures
- 6) operator precedence
- 7) saving output to a file



Question

Loop through the matrix and assign each element a new value. Assign 2 on the main diagonal, -1 on the adjacent diagonals and 0 everywhere else.

Given: nrows=4; nclos=6; A=ones(nrows,nclos);

```
for c = 1:ncols
    for r = 1:nrows

        if r == c
            A(r,c) = 2;
        elseif abs(r-c) == 1
            A(r,c) = -1;
        else
            A(r,c) = 0;
        end

    end
end
A
```

A = 4x6

2	-1	0	0	0	0
-1	2	-1	0	0	0
0	-1	2	-1	0	0
0	0	-1	2	-1	0



example using “any”

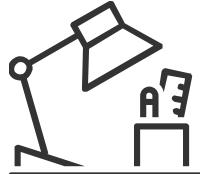
```
limit = 0.75;  
A = rand(10,1)
```

A = 10x1

0.8147
0.9058
0.1270
0.9134
0.6324
0.0975
0.2785
0.5469
0.9575
0.9649

```
if any(A > limit)  
    disp('There is at least one value above the limit.')  
else  
    disp('All values are below the limit.')  
end
```

There is at least one value above the limit.



example using “isequal”

Test Arrays for Equality

Compare arrays using `isequal` rather than the `==` operator to test for equality, because `==` results in an error when the arrays are different sizes.

Create two arrays.

```
A = ones(3,4)  
B = rand(3,4)
```

If `size(A)` and `size(B)` are the same, concatenate the arrays; otherwise, display a warning and return an empty array.

```
if isequal(size(A),size(B))  
    C = [A; B]    % append  
else  
    disp('A and B are not the same size.')  
    C = [];  
end
```

```
A = 3x4  
1 1 1 1  
1 1 1 1  
1 1 1 1  
  
B = 3x4  
0.8055 0.2399 0.4899 0.7127  
0.5767 0.8865 0.1679 0.5005  
0.1829 0.0287 0.9787 0.4711  
  
C = 6x4  
1.0000 1.0000 1.0000 1.0000  
1.0000 1.0000 1.0000 1.0000  
1.0000 1.0000 1.0000 1.0000  
0.8055 0.2399 0.4899 0.7127  
0.5767 0.8865 0.1679 0.5005  
0.1829 0.0287 0.9787 0.4711
```



Evaluate multiple condition in an expression

Q: Determine if a value falls within a specified range.

```
x = 10;  
minVal = 2;  
maxVal = 6;  
  
if (x >= minVal) && (x <= maxVal)  
    disp('Value within specified range.')  
elseif (x > maxVal)  
    disp('Value exceeds maximum value.')  
else  
    disp('Value is below minimum value.')  
end
```

Value exceeds maximum value.



Evaluate multiple condition in an expression

Compare Vectors Containing NaN Values

Create three vectors containing NaN values.

```
A1 = [1 NaN NaN]  
A2 = [1 NaN NaN]  
A3 = [1 NaN NaN]
```

Compare the vectors for equality.

```
tf = isequaln(A1,A2,A3)
```

The result is logical 1 (true) because `isequaln` treats the NaN values as equal to each other.

```
A1 = 1x3  
1   NaN   NaN
```

```
A2 = 1x3  
1   NaN   NaN
```

```
A3 = 1x3  
1   NaN   NaN
```

```
tf = logical  
1
```



Evaluate multiple condition in an expression

Compare Vectors Containing NaN Values

Create three vectors containing NaN values.

```
A1 = [1 NaN NaN]  
A2 = [1 NaN NaN]  
A3 = [1 NaN NaN]
```

Compare the vectors for equality.

```
tf = isequaln(A1,A2,A3)
```

The result is logical 1 (true) because `isequaln` treats the NaN values as equal to each other.

```
A1 = 1×3  
1   NaN   NaN
```

```
A2 = 1×3  
1   NaN   NaN
```

```
A3 = 1×3  
1   NaN   NaN
```

```
tf = logical  
1
```



The “for” loop

The general form of the for loop is:

```
for loopvar = range
    action
end
```

where *loopvar* is the loop variable, “range” is the range of values through which the loop variable is to iterate, and the action of the loop consists of all statements up to the end. Just like with if statements, the action is indented to make it easier to see. The range can be specified using any vector, but normally the easiest way to specify the range of values is to use the colon operator.



The “for” loop: Question

How could you print this column of integers (using the programming method):

0
50
100
150
200

Answer: In a loop, you could print these values starting at 0, incrementing by 50 and ending at 200. Each is printed using a field width of 3.

```
>> for i = 0:50:200  
    fprintf('%3d\n', i)  
end
```



The “for” loop that don’t use iterator

```
for i = 1:3
    fprintf('I will not chew gum\n')
end
```

produces the output:

```
I will not chew gum
I will not chew gum
I will not chew gum
```

The variable *i* is necessary to repeat the action three times, even though the value of *i* is not used in the action of the loop.

What would be the result of the following for loop?

```
for i = 4:2:8
    fprintf('I will not chew gum\n')
end
```

then 8 instead of 1, 2, 3. As the loop variable is not used in the action, this is just another way of specifying that the action should be repeated three times. Of course, using 1:3 makes more sense!

Answer: Exactly the same output as above! It doesn't matter that the loop variable iterates through the values 4, then 6,



Input for “for” loop

`forecho.m`

```
% This script loops to repeat the action of
% prompting the user for a number and echo-printing it

for iv = 1:3
    inputnum = input('Enter a number: ');
    fprintf('You entered %.1f\n', inputnum)
end
```

```
>> forecho
Enter a number: 33
You entered 33.0
Enter a number: 1.1
You entered 1.1
Enter a number: 55
You entered 55.0
```



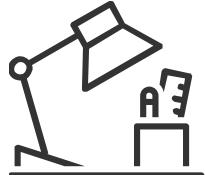
Find sum and product

sumnnums.m

```
% sumnnums calculates the sum of the n numbers
% entered by the user

n = randi([3 10]);
runsum = 0;
for i = 1:n
    inputnum = input('Enter a number: ');
    runsum = runsum + inputnum;
end
fprintf('The sum is %.2f\n', runsum)
```

```
>> sumnnums
Enter a number: 4
Enter a number: 3.2
Enter a number: 1.1
The sum is 8.30
```



Preallocating vector

forgenvec.m

```
% forgenvec creates a vector of length n
% It prompts the user and puts n numbers into a vector

n = randi([4 8]);
numvec = zeros(1, n);
for iv = 1:n
    inputnum = input('Enter a number: ');
    numvec(iv) = inputnum;
end
fprintf('The vector is: \n')
disp(numvec)
```

```
>> forgenvec
Enter a number: 44
Enter a number: 2.3
Enter a number: 11
The vector is:
44.0000  2.3000  11.0000
```

It is very important to notice that the loop variable iv is used as the index into the vector.



Question

QUICK QUESTION!

If you need to just print the sum or average of the numbers that the user enters, would you need to store them in a vector variable?

Answer: No. You could just add each to a running sum as you read them in a loop.

QUICK QUESTION!

What if you wanted to calculate how many of the numbers that the user entered were greater than the average?

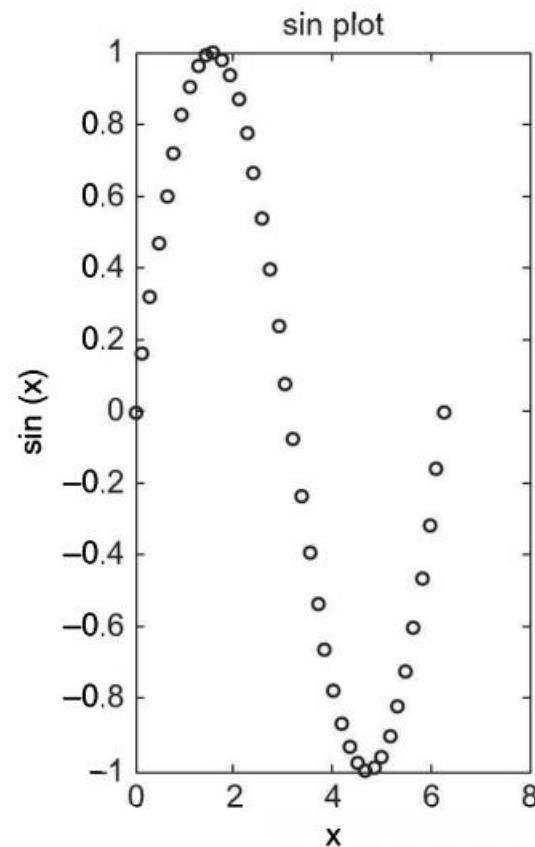
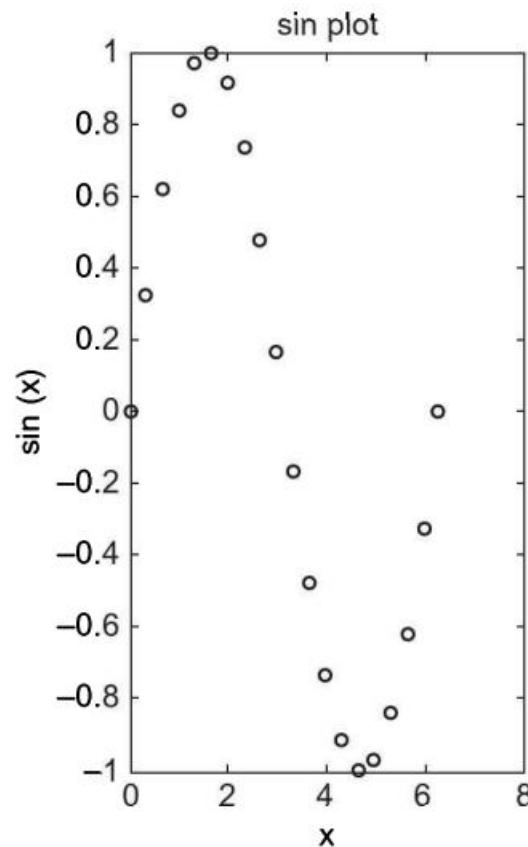
Answer: Yes, then you would need to store them in a vector because you would have to go back through them to count how many were greater than the average (or, alternatively, you could go back and ask the user to enter them again!!).



“for” loop-subplot

subplotex.m

```
% Demonstrates subplot using a for loop
for i = 1:2
    x = linspace(0,2*pi,20*i);
    y = sin(x);
    subplot(1,2,i)
    plot(x,y, 'ko')
    xlabel('x')
    ylabel('sin(x)')
    title('sin plot')
end
```





Nested “for” loop

The general form of a nested for loop is as follows:

```
for loopvarone = rangeone      ← outer loop
    % actionone includes the inner loop
    for loopvartwo = rangetwo    ← inner loop
        actiontwo
    end
end
```



Nested “for” loop

- For every row of output:
 - Print the required number of stars
 - Move the cursor down to the next line (print '\n')

printstars.m

```
% Prints a box of stars
% How many will be specified by two variables
% for the number of rows and columns

rows = 3;
columns = 5;
% loop over the rows
for i=1:rows
    % for every row loop to print *'s and then one \n
    for j=1:columns
        fprintf('*')
    end
    fprintf('\n')
end
```

Executing the script displays the output:

```
>> printstars
*****
*****
*****
```



Nested “for” loop

How could this script be modified to print a triangle of stars instead of a box such as the following:

```
*  
**  
***
```

Answer: In this case, the number of stars to print in each row is the same as the row number (e.g., one star is printed in row 1, two stars in row 2, and so on). The inner for loop does not loop to columns, but to the value of the row loop variable (so we do not need the variable *columns*):

`printtristar.m`

```
% Prints a triangle of stars  
% How many will be specified by a variable  
% for the number of rows  
rows = 3;  
for i=1:rows  
    % inner loop just iterates to the value of i  
    for j=1:i  
        fprintf('*')  
    end  
    fprintf('\n')  
end
```



Nested “for” loop

printloopvars.m

```
% Displays the loop variables
for i = 1:3
    for j = 1:2
        fprintf('i=%d, j=%d\n', i, j)
    end
    fprintf('\n')
end
```

>> *printloopvars*

i=1, j=1

i=1, j=2

i=2, j=1

i=2, j=2

i=3, j=1

i=3, j=2



Nested “for” loop

multtable.m

```
function outmat = multtable(rows, columns)
% multtable returns a matrix which is a
% multiplication table
% Format: multtable(nRows, nColumns)

% Preallocate the matrix
outmat = zeros(rows,columns);
for i = 1:rows
    for j = 1:columns
        outmat(i,j) = i*j;
    end
end
end
```

>> multtable(3,5)

ans =

1	2	3	4	5
2	4	6	8	10
3	6	9	12	15



Nested “for” loop

createmulttab.m

```
% Prompt the user for rows and columns and
% create a multiplication table to store in
% a file "mymulttable.dat"

num_rows = input('Enter the number of rows: ');
num_cols = input('Enter the number of columns: ');
multmatrix = multtable(num_rows, num_cols);
save mymulttable.dat multmatrix -ascii
```

```
>> createmulttab
Enter the number of rows: 6
Enter the number of columns: 4
```

```
>> load mymulttable.dat
```

```
>> mymulttable
mymulttable =
    1     2     3     4
    2     4     6     8
    3     6     9    12
    4     8    12    16
    5    10    15    20
    6    12    18    24
```

MATLAB

Unit 6-Lecture 24

“while ... end" loop

BTech (CSBS) -Semester VII

7 October 2022, 09:35AM



Control Flow and Operators

- 1) relational and logical operators
- 2) “if ... end” structure
- 3) “for ... end” loop
- 4) “while ... end” loop
- 5) other flow structures
- 6) operator precedence
- 7) saving output to a file



“while” loop

```
while condition
    action
end
```

factgthigh.m

```
function facgt = factgthigh(high)
% factgthigh returns the first factorial > input
% Format: factgthigh(inputInteger)

i=0;
fac=1;
while fac <= high
    i=i+1;
    fac = fac * i;
end
facgt = fac;
end
```

```
>> factgthigh(5000)
ans =
5040
```



Multiple condition in “while” loop

```
while x >= 0 && x <= 100
```

As another example, continuing the action of a loop may be desired as long as at least one of two variables is in a specified range:

```
while x < 50 || y < 100
```



Input in “while” loop

whileposnum.m

```
% Prompts the user and echo prints the numbers entered
% until the user enters a negative number

inputnum=input ('Enter a positive number: ');
while inputnum >= 0
    fprintf('You entered a %d.\n\n',inputnum)
    inputnum = input ('Enter a positive number: ');
end
fprintf('OK!\n')
```

```
>> whileposnum
Enter a positive number: 6
You entered a 6.

Enter a positive number: -2
OK!
```



Input in “while” loop

As we have seen previously, MATLAB will give an error message if a character is entered rather than a number.

```
>> whileposnum
Enter a positive number: a
Error using input
Undefined function or variable 'a'.

Error in whileposnum (line 4)
inputnum=input ('Enter a positive number: ') ;
Enter a positive number: - 4
OK!
```

However, if the character is actually the name of a variable, it will use the value of that variable as the input. For example:

```
>> a = 5;
>> whileposnum
Enter a positive number: a
You entered a 5.

Enter a positive number: - 4
OK!
```



Counting in “while” loop

countposnum.m

```
% Prompts the user for positive numbers and echo prints as
% long as the user enters positive numbers

% Counts the positive numbers entered by the user
counter=0;
inputnum=input('Enter a positive number: ');
while inputnum >= 0
    fprintf('You entered a %d.\n\n',inputnum)
    counter = counter+1;
    inputnum = input('Enter a positive number: ');
end
fprintf('Thanks, you entered %d positive numbers.\n',counter)
```

```
>> countposnum
Enter a positive number: 4
You entered a 4.

Enter a positive number: 11
You entered a 11.

Enter a positive number: -4
Thanks, you entered 2 positive numbers.
```



Error checking input in “while” loop

readonenum.m

```
% Loop until the user enters a positive number  
  
inputnum=input('Enter a positive number: ');\nwhile inputnum < 0  
    inputnum = input('Invalid! Enter a positive number: ');\nend  
fprintf('Thanks, you entered a %.1f \n',inputnum)
```

An example of running this script follows:

```
>> readonenum  
Enter a positive number: -5  
Invalid! Enter a positive number: -2.2  
Invalid! Enter a positive number: c  
Error using input  
Undefined function or variable 'c'.  
Error in readonenum (line 5)  
    inputnum = input('Invalid! Enter a positive number: ');\nInvalid! Enter a positive number: 44  
Thanks, you entered a 44.0
```



Question

How could we vary the previous example so that the script asks the user to enter positive numbers n times, where n is an integer defined to be 3?

Answer: Every time the user enters a value, the script checks and in a while loop keeps telling the user that it's

`readnnums.m`

```
% Loop until the user enters n positive numbers
n=3;
fprintf('Please enter %d positive numbers\n\n',n)
for i=1:n
    inputnum=input('Enter a positive number: ');
    while inputnum < 0
        inputnum = input('Invalid! Enter a positive number: ');
    end
    fprintf('Thanks, you entered a %.1f \n',inputnum)
end
```

invalid until a valid positive number is entered. By putting the error-check in a for loop that repeats n times, the user is forced eventually to enter three positive numbers, as shown in the following.

```
>> readnnums
Please enter 3 positive numbers

Enter a positive number: 5.2
Thanks, you entered a 5.2
Enter a positive number: 6
Thanks, you entered a 6.0
Enter a positive number: -7.7
Invalid! Enter a positive number: 5
Thanks, you entered a 5.0
```



Question

MATLAB has a **cumsum** function that will return a vector of all of the running sums of an input vector. However, many other languages do not, so how could we write our own?

Answer: Essentially, there are two programming methods that could be used to simulate the **cumsum** function. One method is to start with an empty vector and extend the vector by adding each running sum to it as the running sums are calculated. A better method is to preallocate the vector to the correct size and then change the value of each element to be successive running sums.

`myveccumsum.m`

```
function outvec = myveccumsum(vec)
% myveccumsum imitates cumsum for a vector
% It preallocates the output vector
% Format: myveccumsum(vector)

outvec = zeros(size(vec));
runsum = 0;
for i = 1:length(vec)
    runsum = runsum + vec(i);
    outvec(i) = runsum;
end
end
```

An example of calling the function follows:

```
>> myveccumsum([5 9 4])
ans =
    5 14 18
```



Practise question

PRACTICE 5.1

Write a for loop that will print a column of five *'s.

PRACTICE 5.2

Write a script *prodnums* that is similar to the *sumnums* script but will calculate and print the product of the numbers entered by the user.



Practise question

PRACTICE 5.3

For each of the following (they are separate), determine what would be printed. Then, check your answers by trying them in MATLAB.

```
mat = [7 11 3; 3:5];
[r, c] = size(mat);
for i = 1:r
    fprintf('The sum is %d\n', sum(mat(i,:)))
end
-----
for i = 1:2
    fprintf('%d: ', i)
    for j = 1:4
        fprintf('%d ', j)
    end
    fprintf('\n')
end
```



Practise question

PRACTICE 5.4

Write a function *mymatmin* that finds the minimum value in each column of a matrix argument and returns a vector of the column minimums. An example of calling the function follows:

```
>> mat = randi(20, 3, 4)
```

```
mat =
```

```
15    19    17    5
 6    14    13   13
 9     5     3   13
```

```
>> mymatmin(mat)
```

```
ans =
```

```
6    5    3    5
```



Practise question

PRACTICE 5.5

Write a script *avenegnum* that will repeat the process of prompting the user for negative numbers, until the user enters a zero or positive number, as just shown. Instead of echo printing them, however, the script will print the average (of just the negative numbers). If no negative numbers are entered, the script will print an error message instead of the average. Use the programming method. Examples of executing this script follow:

```
>> avenegnum
Enter a negative number: 5
No negative numbers to average.
```

```
>> avenegnum
Enter a negative number: -8
Enter a negative number: -3
Enter a negative number: -4
Enter a negative number: 6
The average was -5.00
```

MATLAB

Unit 6-Lecture 25

Other flow structure

BTech (CSBS) -Semester VII

11 October 2022, 09:35AM



Control Flow and Operators

- 1) relational and logical operators
- 2) “if ... end” structure
- 3) “for ... end” loop
- 4) “while ... end” loop
- 5) other flow structures
- 6) operator precedence
- 7) saving output to a file



Other flow structures

Repeat Statements Until Expression Is False

Use a while loop to calculate factorial(10).

```
n = 10;
f = n;
while n > 1
    n = n-1;
    f = f*n;
end
disp(['n! = ' num2str(f)])
```

n! = 3628800

% num2str - converts numeric array to character array



Switch,case, otherwise

Syntax

```
switch switch_expression
    case case_expression
        statements
    case case_expression
        statements
    ...
    otherwise
        statements
end
```

Description

`switch switch_expression, case case_expression, end` evaluates an expression and chooses to execute one of several groups of statements. Each choice is a case.

The switch block tests each case until one of the case expressions is true. A case is true when:

- For numbers, `case expression == switch expression`.
- For character vectors, `strcmp(case_expression, switch_expression) == 1`.



Example

Compare Single Values

Display different text conditionally, depending on a value entered at the command prompt.

```
n = input('Enter a number: ');\n\nswitch n\n    case -1\n        disp('negative one')\n    case 0\n        disp('zero')\n    case 1\n        disp('positive one')\n    otherwise\n        disp('other value')\nend
```

At the command prompt, enter the number 1.

positive one

Repeat the code and enter the number 3.

other value

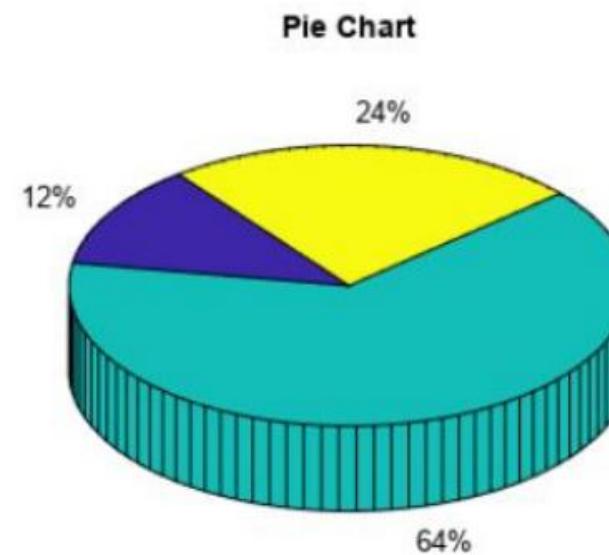


Compare against multiple value

Determine which type of plot to create based on the value of `plottype`. If `plottype` is either 'pie' or 'pie3', create a 3-D pie chart. Use a cell array to contain both values.

```
x = [12 64 24];
plottype = 'pie3';

switch plottype
    case 'bar'
        bar(x)
        title('Bar Graph')
    case {'pie','pie3'}
        pie3(x)
        title('Pie Chart')
    otherwise
        warning('Unexpected plot type. No plot created.')
end
```



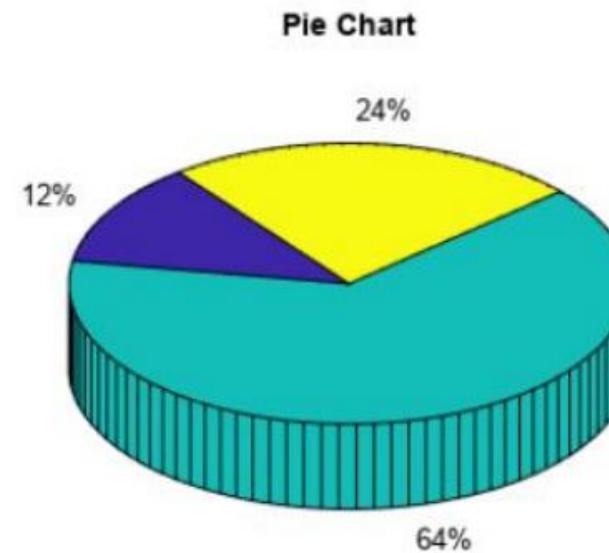


Compare against multiple value

Determine which type of plot to create based on the value of `plottype`. If `plottype` is either 'pie' or 'pie3', create a 3-D pie chart. Use a cell array to contain both values.

```
x = [12 64 24];
plottype = 'pie3';

switch plottype
    case 'bar'
        bar(x)
        title('Bar Graph')
    case {'pie','pie3'}
        pie3(x)
        title('Pie Chart')
    otherwise
        warning('Unexpected plot type. No plot created.')
end
```



MATLAB

Unit 7-Lecture 26

Debugging M-files

BTech (CSBS) -Semester VII

14 October 2022, 09:35AM



Debugging M-files

- 1) Preparing for debugging,
- 2) Examining values,
- 3) Debugging process
- 4) setting breakpoints
- 5) running with breakpoints
- 6) correcting and ending debugging,
- 7) correcting an M- file



Debug MATLAB Code Files

You can diagnose problems in your MATLAB® code files by debugging your code interactively in the Editor and Live Editor or programmatically by using debugging functions in the Command Window. There are several ways to debug your code:

- 1) Display output by removing semicolons.
- 2) Run the code to a specific line and pause by clicking the Run to Here button .
- 3) Step into functions and scripts while paused by clicking the Step In button .
- 4) Add breakpoints to your file to enable pausing at specific lines when you run your code.



Debug MATLAB Code Files

Before you begin debugging, to avoid unexpected results, save your code files and make sure that the code files and any files they call exist on the search path or in the current folder. MATLAB handles unsaved changes differently depending on where you are debugging from:

- 1) Editor — If a file contains unsaved changes, MATLAB saves the file before running it.
- 2) Live Editor — MATLAB runs all changes in a file, whether they are saved or not.
- 3) Command Window — If a file contains unsaved changes, MATLAB runs the saved version of the file. You do not see the results of your changes.



Display Output: Example

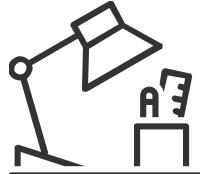
For example, suppose that you have a script called **plotRand.m** that plots a vector of random data and draws a horizontal line on the plot at the mean.

```
n = 50;  
r = rand(n,1);  
plot(r)  
  
m = mean(r);  
hold on  
plot([0,n],[m,m])  
hold off  
title('Mean of Random Uniform Data')
```

Command Window

```
>> plotRand  
  
r =  
  
0.9631  
0.5468  
0.5211  
0.2316  
0.4889  
0.6241  
0.6791
```

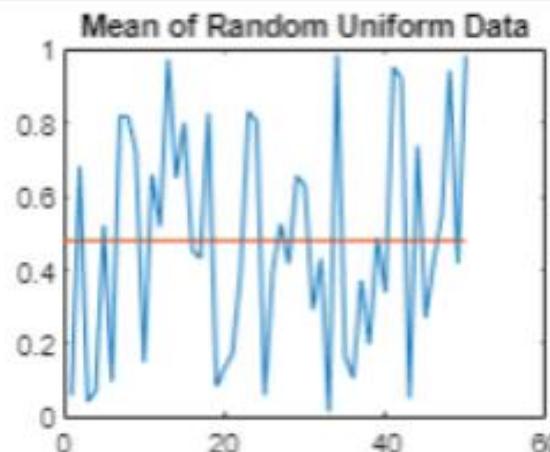
fx



Display Output: Example

```
1 n = 50;
2 r = rand(n,1)
3 plot(r)
4
5 m = mean(r);
6 hold on
7 plot([0,n],[m,m])
8 hold off
9 title('Mean of Random Uniform Data')
```

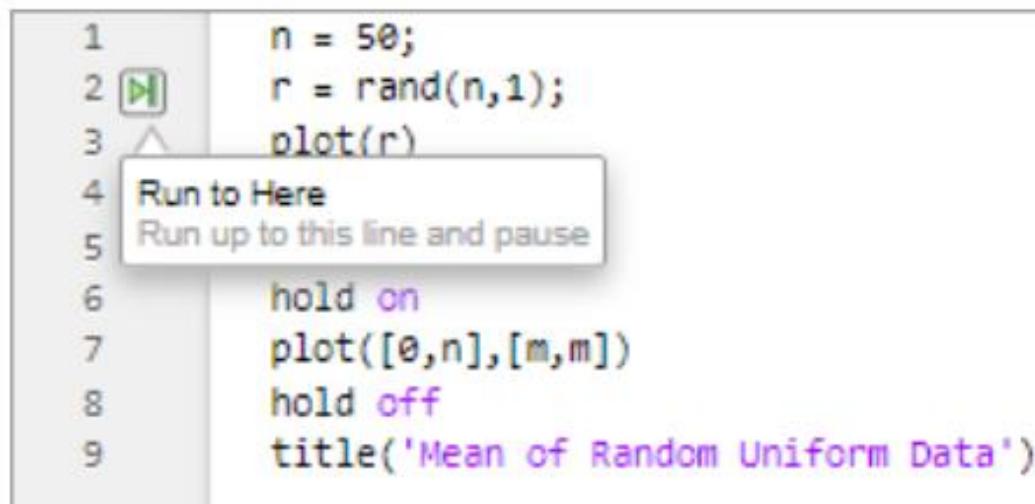
```
r = 50x1
0.0596
0.6820
0.0424
0.0714
0.5216
0.0967
0.8181
0.8175
0.7224
0.1499
:
:
```





Debug Using Run to Here

When debugging, the Run to Here button  becomes the Continue to Here button  In functions and classes, running to a specified line and then pausing is only available when debugging using the Continue to Here button .



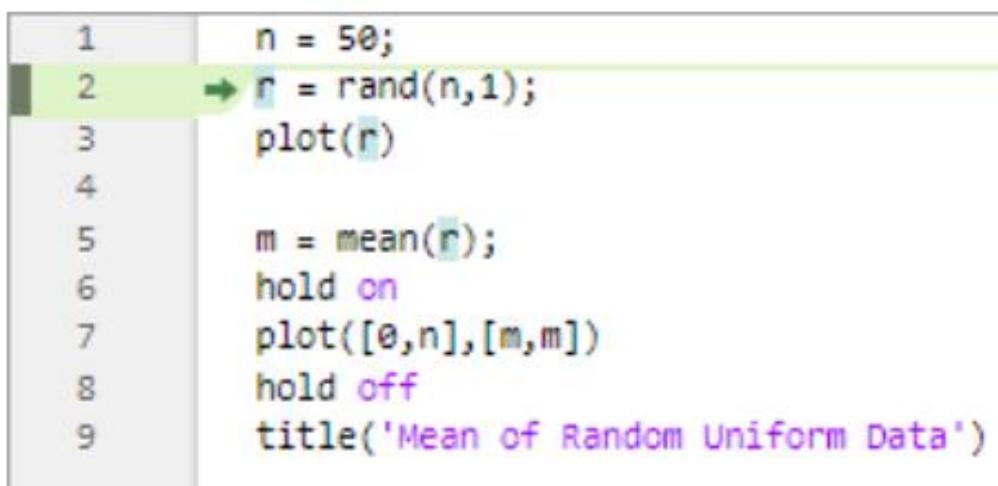
```
1 n = 50;
2 
3 plot(r)
4 Run to Here
5 Run up to this line and pause
6 hold on
7 plot([0,n],[m,m])
8 hold off
9 title('Mean of Random Uniform Data')
```



Debug Using Run to Here

When MATLAB pauses, multiple changes occur:

- 1) The Run button in the **Editor** or Live Editor tab changes to a Continue button.
- 2) The prompt in the Command Window changes to K>> indicating that MATLAB is in debug mode and that the keyboard is in control.
- 3) MATLAB indicates the line at which it is paused by using a green arrow and green highlighting.



```
1 n = 50;
2 r = rand(n,1);
3 plot(r)
4
5 m = mean(r);
6 hold on
7 plot([0,n],[m,m])
8 hold off
9 title('Mean of Random Uniform Data')
```



Debug Using Run to Here

The line at which MATLAB is paused does not run until after you continue running the code. To continue running the code, click the  **Continue** button. MATLAB continues running the file until it reaches the end of the file or a breakpoint. You also can click the Continue to Here button  to the left of the line of code that you want to continue running to.

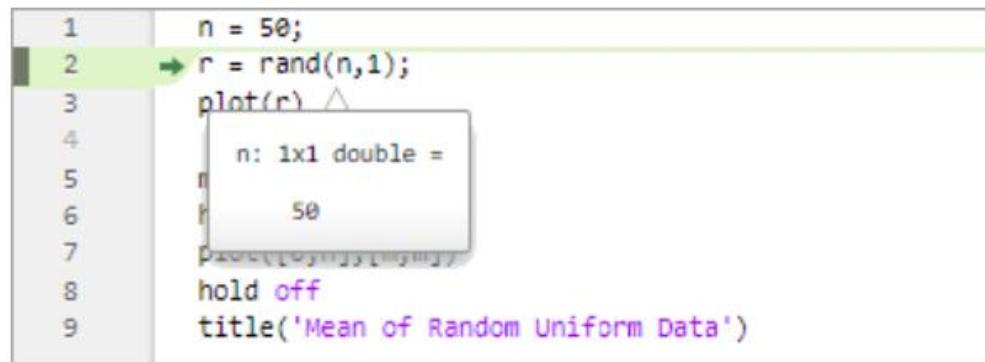
To continue running the code line-by-line, on the **Editor** or **Live Editor** tab, click  **Step**. MATLAB executes the current line at which it is paused and the pauses at the next line.

```
1 n = 50;
2 r = rand(n,1);
3 plot(r)
4
5 m = mean(r);
6 hold on
7 plot([0,n],[m,m])
8 hold off
9 title('Mean of Random Uniform Data')
```



View Variable Value While Debugging

To view the value of a variable while MATLAB is paused, place your cursor over the variable. The current value of the variable appears in a data tip. The data tip stays in view until you move the cursor. To disable data tips, go to the **View** tab and click the  **Datatips** button off.



The screenshot shows the MATLAB Editor with a script containing the following code:

```
1 n = 50;
2 r = rand(n,1);
3 plot(r) ▲
4
5 n: 1x1 double =
6
7 r
8
9 hold off
10 title('Mean of Random Uniform Data')
```

A data tip is displayed over the variable `n` in line 2. The tip shows the variable's value: `n: 1x1 double = 50`.

You also can view the value of a variable by typing the variable name in the Command Window. For example, to see the value of the variable `n`, type `n` and press **Enter**. The Command Window displays the variable name and its value. To view all the variables in the current workspace, use the Workspace browser.



Pause a Running File

You can pause long running code while it is running to check on the progress and ensure that it is running as expected. To pause running code, go to the **Editor** or **Live Editor** tab and click the  **Pause** button. MATLAB pauses at the next executable line, and the  **Pause** button changes to a  **Continue** button. To continue running the code, press the  **Continue** button.



Note

Clicking the  **Pause** button can cause MATLAB to pause in a file outside your own code.



Examine Values While Debugging

When debugging a code file, you can view the value of any variable currently in the workspace while MATLAB® is paused. If you want to determine whether a line of code produces the expected result or not, examining values is useful. If the result is as expected, you can continue running the code or step to the next line. If the result is not as you expect, then that line, or a previous line, might contain an error.



View Variable Value

There are several ways to view the value of a variable while debugging:

- Workspace browser — The Workspace browser displays all variables in the current workspace. The Value column of the Workspace browser shows the current value of the variable.

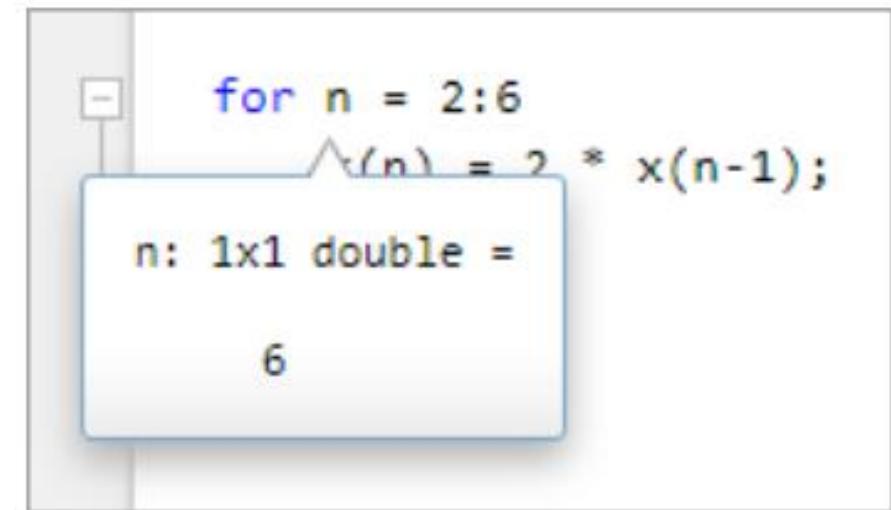
Name	Value	Class
n	6	double
x	[1,2,4,8,16,32,1,1,1,1]	double



View Variable Value

To view more details, double-click the variable. The Variables Editor opens, displaying the content for that variable. You also can use the `openvar` function to open a variable in the Variables Editor.

- **Editor and Live Editor** — To view the value of a variable in the Editor and Live Editor, place your cursor over the variable. The current value of the variable appears in a data tip. The data tip stays in view until you move the cursor. If you have trouble getting the data tip to appear, click the line containing the variable, and then move the pointer next to the variable.



A screenshot of a MATLAB code editor showing a data tip for the variable `n`. The code in the editor is:

```
for n = 2:6
    x(n) = 2 * x(n-1);
```

The data tip, which is a light blue box, displays the following information:

`n: 1x1 double =`
6



View Variable Value Outside Current Workspace

You also can use the **dbstack** function to view the current workspace in the Command Window:

To examine the values of variables outside of the current workspace, select a different workspace. In the Editor or Live Editor, select a workspace from the drop-down list to the right of the function call stack at the top of the file.



```
dbstack
> In mean (line 48)
In plotRand (line 5)
```

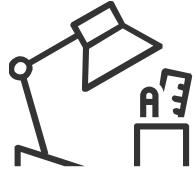


View Variable Value Outside Current Workspace

You also can use the **dbup** and **dbdown** functions in the Command Window to select the previous or next workspace in the function call stack. To list the variables in the current workspace, use **who** or **whos**. If you attempt to view the value of a variable in a different workspace while MATLAB is in the process of overwriting it, MATLAB displays an error in the Command Window.

```
K>> x
Variable "x" is inaccessible. When a variable appears on both sides of an assignment
statement, the variable may become temporarily unavailable during processing.
```

The error occurs whether you select the workspace by using the drop-down list to the right of the function call stack or the **dbup** command.



Debug by Using Keyboard Shortcuts or Functions

Action	Description	Keyboard Shortcut	Function
Continue ➤	Continue running file until the end of the file is reached or until another breakpoint is encountered.	F5	<code>dbcont</code>
Step ⏪	Run the current line of code.	F10 (Shift+Command+O on macOS systems)	<code>dbstep</code>
Step In ⏴	Run the current line of code, and, if the line contains a call to another function, step into that function.	F11 (Shift+Command+I on macOS systems)	<code>dbstep in</code>
Step Out ⏵	After stepping in, run the rest of the called function, leave the called function, and then pause.	Shift+F11 (Shift+Command+U on macOS systems)	<code>dbstep out</code>
Stop ■	End debugging session.	Shift+F5	<code>dbquit</code>
Set Breakpoint	Set a breakpoint at the current line, if no breakpoint exists.	F12	<code>dbstop</code>
Clear Breakpoint	Clear the breakpoint at the current line.	F12	<code>dbclear</code>



Function Call Stack

When you step into a called function or file, MATLAB displays the list of the functions it executed before pausing at the current line. The list, also called the **function call stack**, is shown at the top of the file and displays the functions in order, starting on the left with the first called script or function, and ending on the right with the current script or function in which MATLAB is paused.



For each function in the function call stack, there is a corresponding workspace. Workspaces contain variables that you create within MATLAB or import from data files or other programs. Variables that you assign through the Command Window or create by using scripts belong to the base workspace. Variables that you create in a function belong to their own function workspace.



HW

dbcont	
dbclear	
dbstack	
dbstatus	
dbstop	
dbstep	

MATLAB

Unit 7-Lecture 27

Debugging process , setting breakpoints

BTech (CSBS) -Semester VII

14 October 2022, 09:35AM



Debugging M-files

- 1) preparing for debugging,
- 2) examining values,
- 3) Debugging process
- 4) setting breakpoints
- 5) running with breakpoints
- 6) correcting and ending debugging
- 7) correcting an M- file



Build Process

- Before you can build an executable program or shared library for a model, choose and set up a compiler or IDE and configure the target environment.
- Several methods are available for initiating the build process.
- Tooling is available for reloading, rebuilding, and relocating generated code.
- If your system includes referenced models, reduce build time and control whether the code generator regenerates code for the top model.
- To improve the speed of code execution, consider using available profiling capabilities.

Functions

▼ Initiate Build Process	
<code>packNGo</code>	Package generated code in ZIP file for relocation
<code>rtw_precompile_libs</code>	Build model libraries without building model
<code>codebuild</code>	Compile and link generated code
<code>rtwrebuild</code>	Rebuild generated code from model
<code>slbuild</code>	Build standalone executable file or model reference target for model



Build Process

▼ Get or Modify Build Process Controls

<code>coder.buildstatus.close</code>	Close Build Status window
<code>coder.buildstatus.open</code>	Open Build Status window
<code>RTW.getBuildDir</code>	Get build folder information from model build information
<code>Simulink.fileGenControl</code>	Specify root folders for files generated by diagram updates and model builds
<code>switchTarget</code>	Select target for model configuration set



Debugging Approaches

There are probably a lot of ways to debug programs. These include:

- 1) editing the code and removing semicolons or adding a **keyboard** statement at judicious locations
- 2) **setting a breakpoint** at a particular line and stepping through code from there
- 3) using a variant of setting a breakpoint by using **dbstop if error**
- 4) seeing if the **mlint code analyser** can help (also reachable from the Tools menu)
- 5) comparing variants of the code using the **File and Folder Comparisons** tool or **visdiff** for command-line access



Set Breakpoint

Setting breakpoints pauses the execution of your MATLAB® program so that you can examine values where you think an issue might have occurred. You can set breakpoints interactively in the Editor or Live Editor, or by using functions in the Command Window.

There are three types of breakpoints:

- 1) Standard
- 2) Conditional
- 3) Error



Set Breakpoint

By default, when MATLAB reaches a breakpoint, it opens the file containing the breakpoint.

To disable this option:

1. From the **Home** tab, in the **Environment** section, click  **Preferences**.
2. In the Preferences window, select **MATLAB > Editor/Debugger**.
3. Clear the **Automatically open file when MATLAB reaches a breakpoint** option and click **OK**.



Standard Breakpoints

A standard breakpoint pauses at a specific line in a file. To set a standard breakpoint, click the gray area to the left of the executable line where you want to set the breakpoint. Alternatively, you can press the F12 key to set a breakpoint at the current line. If you attempt to set a breakpoint at a line that is not executable, such as a comment or a blank line, MATLAB sets it at the next executable line.

```
1 n = 50;
2 r = rand(n,1);
3 plot(r)
4
5 m = mean(r);
6 hold on
7 plot([0,n],[m,m])
8 hold off
9 title('Mean of Random Uniform Data')
```



Standard Breakpoints

To set a standard breakpoint programmatically, use the **dbstop** function.

```
dbstop in plotRand at 3
```

When debugging a file that contains a loop, set the breakpoint inside the loop to examine the values at each increment of the loop. Otherwise, if you set the breakpoint at the start of the loop, MATLAB pauses at the loop statement only once. For example, this code creates an array of ten ones and uses a for loop to perform a calculation on items two through six of the array:

```
x = ones(1:10);  
  
for n = 2:6  
    x(n) = 2 * x(n-1);  
end
```



Standard Breakpoints

For MATLAB to pause at each increment of the for loop (a total of five times), set a breakpoint at line four.

A screenshot of a MATLAB code editor. On the left, a vertical line of numbers 3, 4, and 5 is displayed. The number 4 is highlighted with a red box and a small red circle with a white 'b' is positioned to its right, indicating it is a breakpoint. To the right of the numbers is the MATLAB code:

```
for n = 2:6
    x(n) = 2 * x(n-1);
end
```



Conditional Breakpoints

- To set a conditional breakpoint, right-click the gray area to the left of the executable line where you want to set the breakpoint and select Set Conditional Breakpoint.
- If a breakpoint already exists on that line, select Set/Modify Condition. In the dialog box that opens, enter a condition and click OK.
- A condition is any valid MATLAB expression that returns a logical scalar value.
- **Example:** Set a conditional breakpoint at line four with the condition $n \geq 4$. When you run the code, MATLAB runs through the for loop twice and pauses on the third iteration at line four when n is 4. If you continue running the code, MATLAB pauses again at line four on the fourth iteration when n is 5, and then once more, when n is 6.



Conditional Breakpoints: Example

Code:

```
x = ones(1:10)  
  
for n = 2:6  
    x(n) = 2 * x(n-1);  
end
```

Result:

```
1 x = ones(1:10);  
2  
3  
4  
5 for n = 2:6  
    x(n) = 2 * x(n-1);  
end
```

Stop the file:

```
dbstop in myprogram at 6 if n>=4
```



Error Breakpoints

To set an error breakpoint, on the **Editor** tab, click  **Run** and select from these options:

- **Pause on Errors** to pause on all errors.
- **Pause on Warnings** to pause on all warnings.
- **Pause on NaN or Inf** to pause on NaN (not-a-number) or Inf (infinite) values.

```
dbstop if error
```

```
dbstop if caught error MATLAB:ls:InputsMustBeStrings
```

MATLAB

Unit 7-Lecture 27

Debugging process , setting breakpoints

BTech (CSBS) -Semester VII

14 October 2022, 09:35AM



Debugging M-files

- 1) preparing for debugging,
- 2) examining values,
- 3) Debugging process
- 4) setting breakpoints
- 5) running with breakpoints
- 6) correcting and ending debugging
- 7) correcting an M- file



Build Process

- Before you can build an executable program or shared library for a model, choose and set up a compiler or IDE and configure the target environment.
- Several methods are available for initiating the build process.
- Tooling is available for reloading, rebuilding, and relocating generated code.
- If your system includes referenced models, reduce build time and control whether the code generator regenerates code for the top model.
- To improve the speed of code execution, consider using available profiling capabilities.

Functions

▼ Initiate Build Process	
<code>packNGo</code>	Package generated code in ZIP file for relocation
<code>rtw_precompile_libs</code>	Build model libraries without building model
<code>codebuild</code>	Compile and link generated code
<code>rtwrebuild</code>	Rebuild generated code from model
<code>slbuild</code>	Build standalone executable file or model reference target for model



Build Process

▼ Get or Modify Build Process Controls

<code>coder.buildstatus.close</code>	Close Build Status window
<code>coder.buildstatus.open</code>	Open Build Status window
<code>RTW.getBuildDir</code>	Get build folder information from model build information
<code>Simulink.fileGenControl</code>	Specify root folders for files generated by diagram updates and model builds
<code>switchTarget</code>	Select target for model configuration set



Debugging Approaches

There are probably a lot of ways to debug programs. These include:

- 1) editing the code and removing semicolons or adding a **keyboard** statement at judicious locations
- 2) **setting a breakpoint** at a particular line and stepping through code from there
- 3) using a variant of setting a breakpoint by using **dbstop if error**
- 4) seeing if the **mlint code analyser** can help (also reachable from the Tools menu)
- 5) comparing variants of the code using the **File and Folder Comparisons** tool or **visdiff** for command-line access



Set Breakpoint

Setting breakpoints pauses the execution of your MATLAB® program so that you can examine values where you think an issue might have occurred. You can set breakpoints interactively in the Editor or Live Editor, or by using functions in the Command Window.

There are three types of breakpoints:

- 1) Standard
- 2) Conditional
- 3) Error



Set Breakpoint

By default, when MATLAB reaches a breakpoint, it opens the file containing the breakpoint.

To disable this option:

1. From the **Home** tab, in the **Environment** section, click  **Preferences**.
2. In the Preferences window, select **MATLAB > Editor/Debugger**.
3. Clear the **Automatically open file when MATLAB reaches a breakpoint** option and click **OK**.



Standard Breakpoints

A standard breakpoint pauses at a specific line in a file. To set a standard breakpoint, click the gray area to the left of the executable line where you want to set the breakpoint. Alternatively, you can press the F12 key to set a breakpoint at the current line. If you attempt to set a breakpoint at a line that is not executable, such as a comment or a blank line, MATLAB sets it at the next executable line.

```
1 n = 50;
2 r = rand(n,1);
3 plot(r)
4
5 m = mean(r);
6 hold on
7 plot([0,n],[m,m])
8 hold off
9 title('Mean of Random Uniform Data')
```



Standard Breakpoints

To set a standard breakpoint programmatically, use the **dbstop** function.

```
dbstop in plotRand at 3
```

When debugging a file that contains a loop, set the breakpoint inside the loop to examine the values at each increment of the loop. Otherwise, if you set the breakpoint at the start of the loop, MATLAB pauses at the loop statement only once. For example, this code creates an array of ten ones and uses a for loop to perform a calculation on items two through six of the array:

```
x = ones(1:10);  
  
for n = 2:6  
    x(n) = 2 * x(n-1);  
end
```



Standard Breakpoints

For MATLAB to pause at each increment of the for loop (a total of five times), set a breakpoint at line four.

A screenshot of a MATLAB code editor. On the left, a vertical line of numbers 3, 4, and 5 is displayed. The number 4 is highlighted with a red box and a small red circle with a white 'b' is positioned to its right, indicating it is a breakpoint. To the right of the numbers is the MATLAB code:

```
for n = 2:6
    x(n) = 2 * x(n-1);
end
```



Conditional Breakpoints

- To set a conditional breakpoint, right-click the gray area to the left of the executable line where you want to set the breakpoint and select Set Conditional Breakpoint.
- If a breakpoint already exists on that line, select Set/Modify Condition. In the dialog box that opens, enter a condition and click OK.
- A condition is any valid MATLAB expression that returns a logical scalar value.
- **Example:** Set a conditional breakpoint at line four with the condition $n \geq 4$. When you run the code, MATLAB runs through the for loop twice and pauses on the third iteration at line four when n is 4. If you continue running the code, MATLAB pauses again at line four on the fourth iteration when n is 5, and then once more, when n is 6.



Conditional Breakpoints: Example

Code:

```
x = ones(1:10)  
  
for n = 2:6  
    x(n) = 2 * x(n-1);  
end
```

Result:

```
1 x = ones(1:10);  
2  
3  
4  
5 for n = 2:6  
    x(n) = 2 * x(n-1);  
end
```

Stop the file:

```
dbstop in myprogram at 6 if n>=4
```



Error Breakpoints

To set an error breakpoint, on the **Editor** tab, click  **Run** and select from these options:

- **Pause on Errors** to pause on all errors.
- **Pause on Warnings** to pause on all warnings.
- **Pause on NaN or Inf** to pause on NaN (not-a-number) or Inf (infinite) values.

```
dbstop if error
```

```
dbstop if caught error MATLAB:ls:InputsMustBeStrings
```

MATLAB

Unit 7-Lecture 28 and 29

Debugging M-files

BTech (CSBS) -Semester VII

18 October 2022, 09:35AM



Control Flow and Operators

- 1) preparing for debugging,
- 2) examining values,
- 3) Debugging process
- 4) setting breakpoints
- 5) running with breakpoints
- 6) correcting an M- file
- 7) correcting and ending debugging



Running with breakpoints

Code:

```
local_max.m x +  
1 [-] function [vals,locs]=local_max(v)  
2 n=length(v);  
3 vals=[];  
4 locs=[];  
5  
6 [-] for m=2:n  
7 if v(m)-v(m-1)> 0 && v(m)-v(m+1)>0  
8     vals=[vals;v(m)];  
9     locs=[locs;m];  
10    end  
11 end  
12 figure; plot (v);  
13 hold on;  
14 plot (locs,vals,'ro');
```



Running with breakpoints

Run:

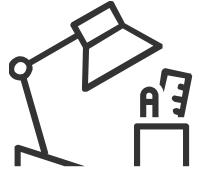
Command Window

```
>> local_max  
Not enough input arguments.
```

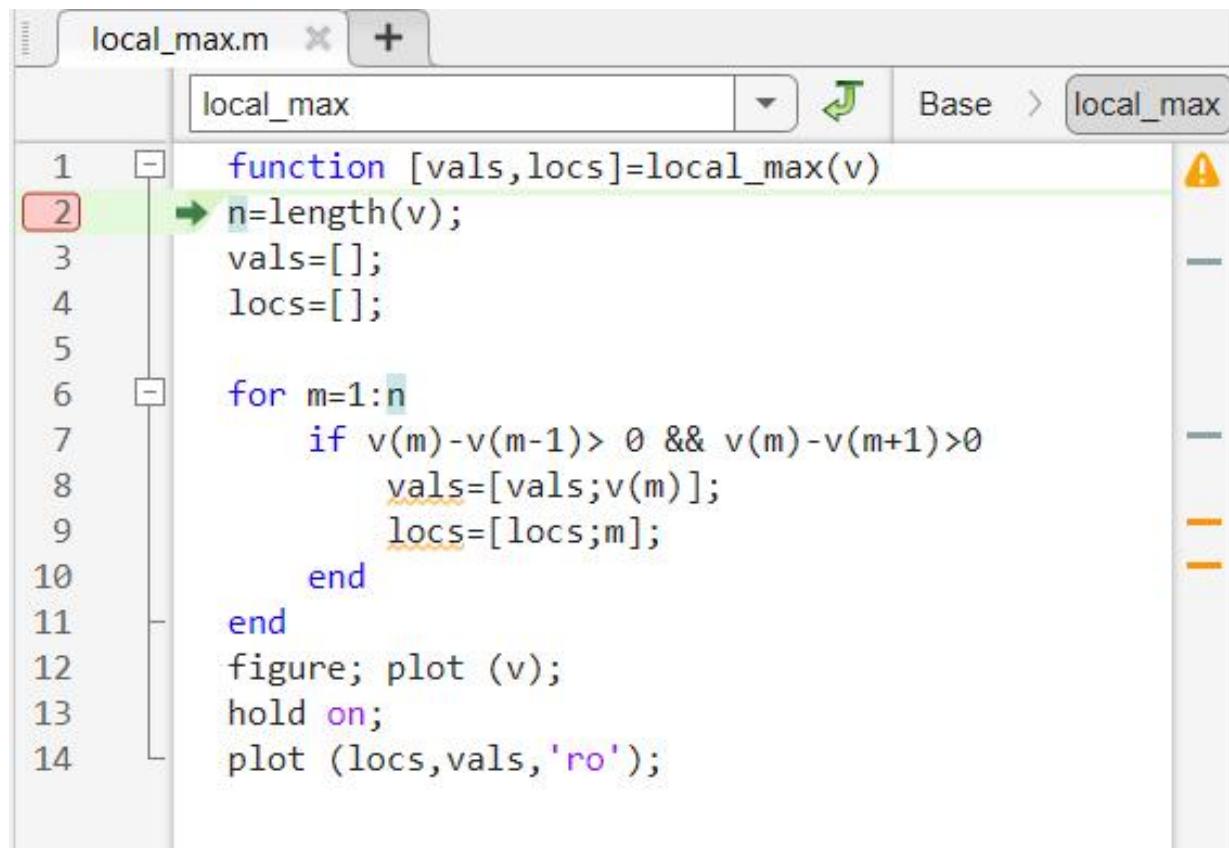
```
Error in local_max (line 2)  
n=length(v);
```

Run
some
value:

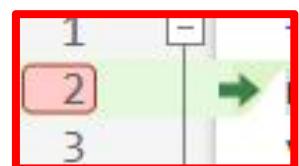
```
>> a=rand(10,1);  
>> [V,L]=local_max(a);  
Array indices must be positive integers or logical values.  
  
Error in local_max (line 7)  
if v(m)-v(m-1)> 0 && v(m)-v(m+1)>0
```



Set the breakpoint



```
1 function [vals,locs]=local_max(v)
2 n=length(v);
3 vals=[];
4 locs=[];
5
6 for m=1:n
7     if v(m)-v(m-1)> 0 && v(m)-v(m+1)>0
8         vals=[vals;v(m)];
9         locs=[locs;m];
10    end
11 end
12 figure; plot (v);
13 hold on;
14 plot (locs,vals,'ro');
```



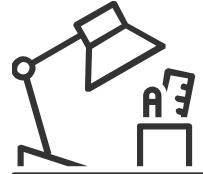
This means the code is not executed yet

Result:

```
>> a=rand(10,1);
>> [v,L]=local_max(a);
2 n=length(v);
fx K>> |
```

All workspace variable are gone:

Workspace - local_max	
Name	Value
v	[0.6557;0.0357;0.8491;0.934...



Execute line: Press Step

Step

local_max.m

```
1 function [vals,locs]=local_max(v)
2 n=length(v);
3 vals=[];
4 locs=[];
5
6 for m=1:n
7     if v(m)-v(m-1)> 0 && v(m)-v(m+1)>0
8         vals=[vals;v(m)];
9         locs=[locs;m];
10    end
11 end
12 figure; plot (v);
13 hold on;
14 plot (locs,vals,'ro');
```

Workspace is declared with n values

Workspace - local_max	
Name	Value
n	10
v	[0.6557;0.0357;0.8491;0.934...



Execute line: Press Step



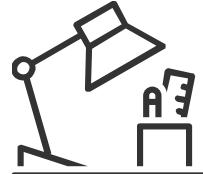
again and again till error line

local_max.m

```
1 function [vals,locs]=local_max(v)
2 n=length(v);
3 vals=[];
4 locs=[];
5
6 for m=1:n
7     if v(m)-v(m-1)> 0 && v(m)-v(m+1)>0
8         vals=[vals;v(m)];
9         locs=[locs;m];
10    end
11 end
12 figure; plot (v);
13 hold on;
14 plot (locs,vals,'ro');
```

Workspace is declared with n values

Workspace - local_max	
Name	Value
n	10
v	[0.6557;0.0357;0.8491;0.934...
vals	[]



Execute line: Press Step



again and again till error line

Editor - C:\Users\91887\Desktop\Onedrive\OneDrive - Indian Institute o... X

local_max.m +

local_max

local_max

```
1
2
3
4
5
6 for m=1:n
7     if v(m)-v(m-1)> 0 && v(m)-v(m+1)>0
8         vals=[vals;v(m)];
9         locs=[locs;m];
10    end
11 end
12 figure; plot (v);
13 hold on;
14 plot (locs,vals,'ro');
```

Workspace is declared with n values

Workspace - local_max

Name	Value
locs	[]
n	10
v	[0.6557;0.0357;0.8491;0.934...
vals	[]



Execute line: Press Step



again and again till error line

local_max.m

```
1 function [vals,locs]=local_max(v)
2 n=length(v);
3 vals=[];
4 locs=[];
5
6 for m=1:n
7     if v(m)-v(m-1)> 0 && v(m)-v(m+1)>0
8         vals=[vals;v(m)];
9         locs=[locs;m];
10    end
11 end
12 figure; plot (v);
13 hold on;
14 plot (locs,vals,'ro');
```

Workspace is declared with n values

Workspace - local_max

Name	Value
locs	[]
m	1
n	10
v	[0.6557;0.0357;0.8491;0.934...
vals	[]



Error line

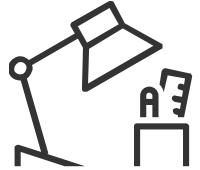
local_max.m

```
1 function [vals,locs]=local_max(v)
2 n=length(v);
3 vals=[];
4 locs=[];
5
6 for m=1:n
7     if v(m)-v(m-1)> 0 && v(m)-v(m+1)>0
8         vals=[vals;v(m)];
9         locs=[locs;m];
10    end
11 end
12 figure; plot (v);
13 hold on;
14 plot (locs,vals,'ro');
```



Error

```
>> a=rand(10,1);
>> [V,L]=local_max(a);
2 n=length(v);
K>> v(m)-v(m-1)> 0 && v(m)-v(m+1)>0
Array indices must be positive integers or logical values.
```



Breaking the Error line furthur

```
local_max.m x + local_max use
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

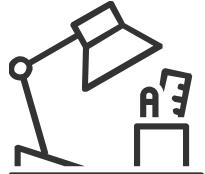
```
function [vals,locs]=local_max(v)
n=length(v);
vals=[];
locs=[];
for m=1:n
    if v(m)-v(m-1)> 0 && v(m)-v(m+1)>0
        vals=[vals;v(m)];
        locs=[locs;m];
    end
end
figure; plot (v);
hold on;
plot (locs,vals,'ro');
```

The code is a MATLAB script named 'local_max.m'. It defines a function that takes a vector 'v' as input and returns two vectors: 'vals' (local maxima) and 'locs' (their indices). The script uses a for loop to iterate through the elements of 'v'. For each element 'm', it checks if it is a local maximum (i.e., if the element is greater than its neighbors). If it is, it adds the value to 'vals' and the index to 'locs'. The script then plots the original vector 'v' and the identified local maxima 'vals' at their respective indices 'locs'.

Error

K>> v (m) -v (m-1) > 0

Array indices must be positive integers or logical values.



Checking individual terms

```
local_max.m x + local_max 1 2 3 4 5 6 7 8 9 10 11 12 13 14
```

```
function [vals,locs]=local_max(v)
n=length(v);
vals=[];
locs=[];
for m=1:n
    if v(m)-v(m-1)> 0 && v(m)-v(m+1)>0
        vals=[vals;v(m)];
        locs=[locs;m];
    end
end
figure; plot (v);
hold on;
plot (locs,vals,'ro');
```

Run:

```
K>> v (m)
ans =
0.66
```



Checking individual terms

```
local_max.m x + local_max
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

```
function [vals,locs]=local_max(v)
n=length(v);
vals=[];
locs=[];
for m=1:n
    if v(m)-v(m-1)> 0 && v(m)-v(m+1)>0
        vals=[vals;v(m)];
        locs=[locs;m];
    end
end
figure; plot (v);
hold on;
plot (locs,vals,'ro');
```

Run:

```
K>> v(m-1)
Array indices must be positive integers or logical values.
```

Since, value of m in line 6 starts from 1. so $v(m-1)$ will be $1-1=0$.

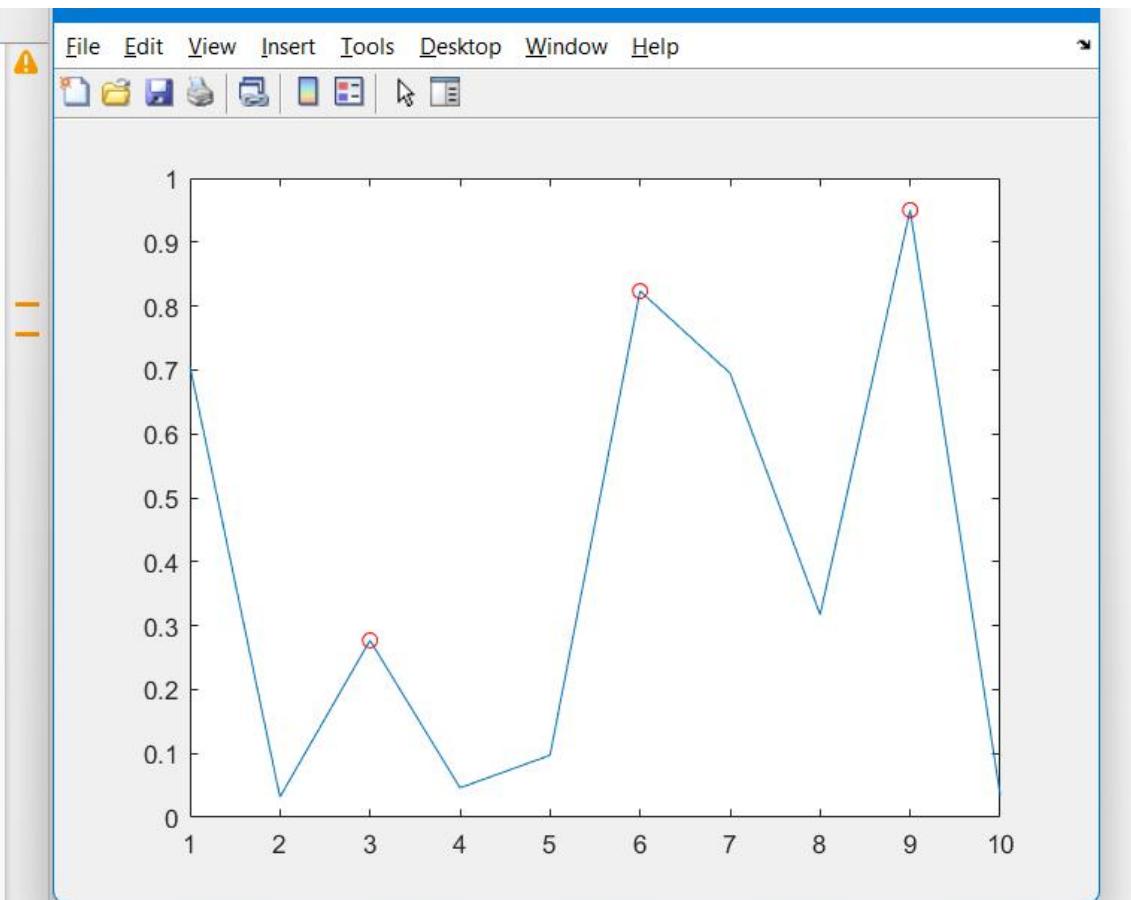
Thus there is an error and need to be corrected.



Correct the code

Quit debugger here, come to the normal .m file, remove breakpoint and run the code

```
local_max.m ✘ +  
1 function [vals,locs]=local_max(v)  
2 n=length(v);  
3 vals=[];  
4 locs=[];  
5  
6 for m=2:n  
7     if v(m)-v(m-1)> 0 && v(m)-v(m+1)>0  
8         vals=[vals;v(m)];  
9         locs=[locs;m];  
10    end  
11 end  
12 figure; plot (v);  
13 hold on;  
14 plot (locs,vals, 'ro');
```





Check the values

- write m under for loop
- quit debugger
- stop breakpoint
- save the file
- execute the code

```
local_max.m  ✘ +  
1  [-] function [vals,locs]=local_max(v)  
2      n=length(v);  
3      vals=[];  
4      locs=[];  
5  
6  [-] for m=2:n  
7      m  
8          if v(m)-v(m-1)> 0 && v(m)-v(m+1)>0  
9              vals=[vals;v(m)];  
10             locs=[locs;m];  
11         end  
12     end  
13     figure; plot (v);  
14     hold on;  
15     plot (locs,vals,'ro');
```



Execute the code

```
>> [V,L]=local_max(a);
```

```
m =
```

```
2.00
```

```
m =
```

```
3.00
```

```
m =
```

```
4.00
```

```
m =
```

```
5.00
```

```
m =
```

```
6.00
```

```
m =
```

```
7.00
```

```
m =
```

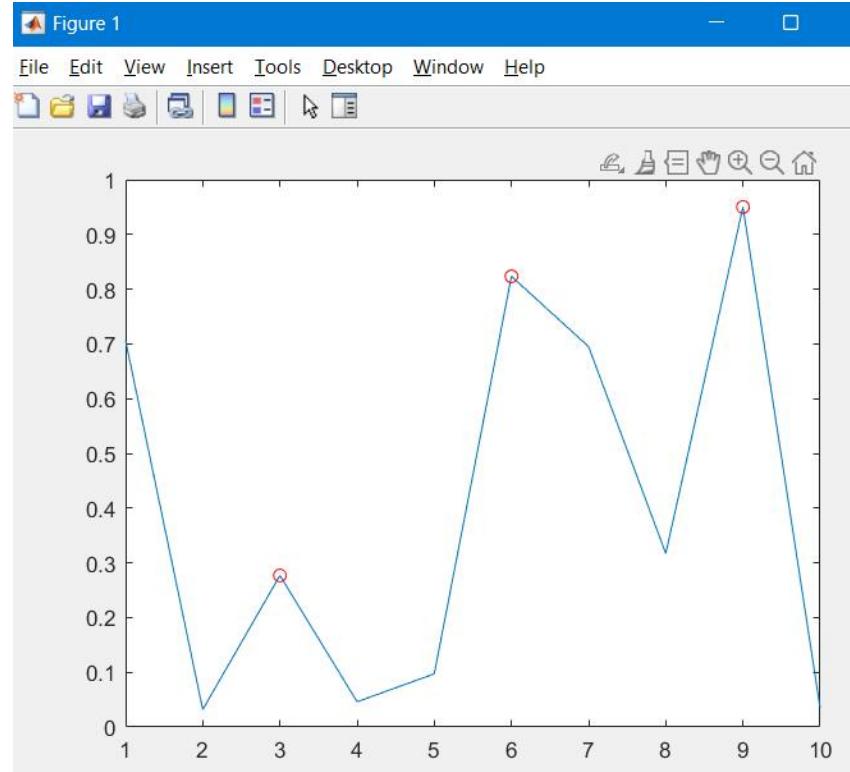
```
8.00
```

```
m =
```

```
9.00
```

```
m =
```

```
10.00
```



Code is executed properly, is there any problem?

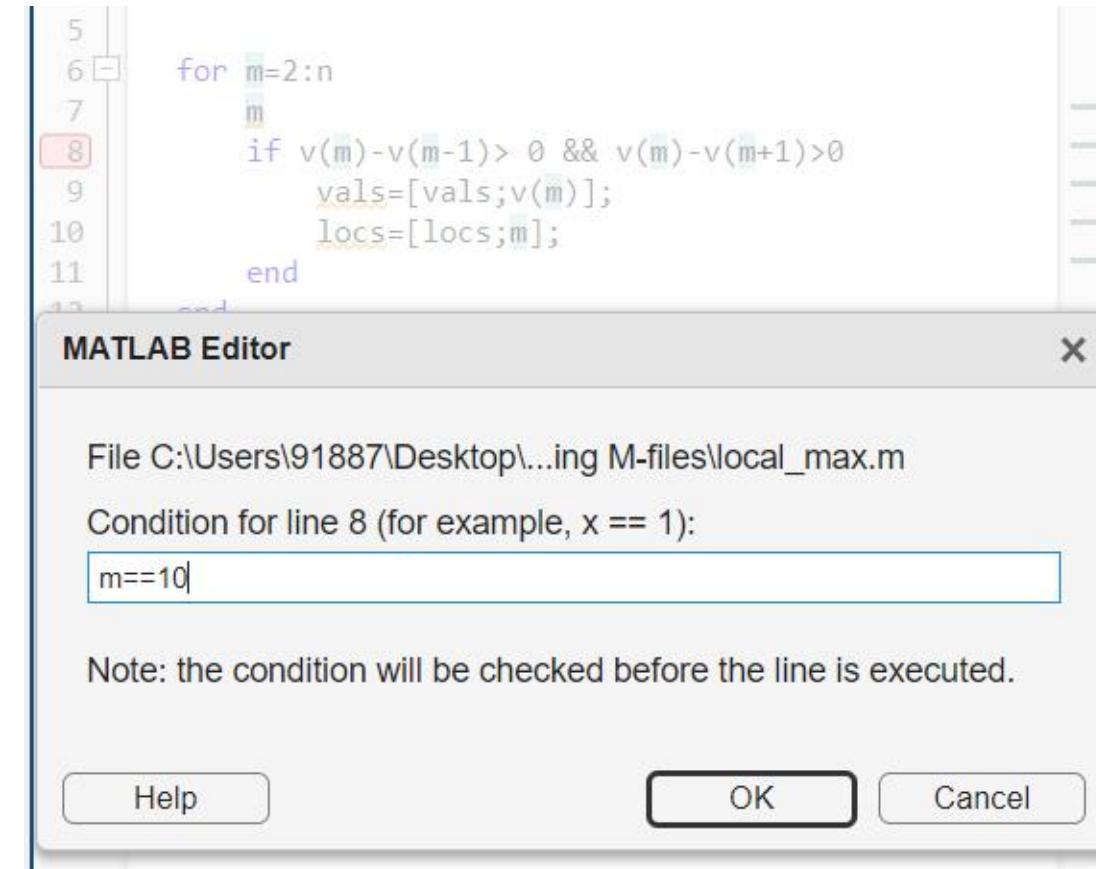


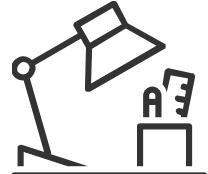
Set a condition breakpoint

```
local_max.m x +  
1 function [vals,locs]=local_max(v)  
2 n=length(v);  
3 vals=[];  
4 locs=[];  
5  
6 for m=2:n  
7 m  
8 if v(m)-v(m-1)>0 && v(m)-v(m+1)>0  
9     vals=[vals;v(m)];  
10    locs=[locs;m];  
11 end  
12 end
```

Line 8 is highlighted with a red box. A context menu is open at this line, showing the following options:

- Set/Modify Condition...
- Disable Breakpoint
- Clear Breakpoint
- Disable All Breakpoints in File
- Clear All Breakpoints
- Clear All Breakpoints in File
- Show Code Folding Margin





Execute file

local_max.m

```
function [vals,locs]=local_max(v)
n=length(v);
vals=[];
locs=[];
for m=2:n
    m
    if v(m)-v(m-1)> 0 && v(m)-v(m+1)>0
        vals=[vals;v(m)];
        locs=[locs;m];
    end
end
figure; plot (v);
hold on;
plot (locs,vals,'ro');
```

m = 5.00

m = 6.00

m = 7.00

m = 8.00

m = 9.00

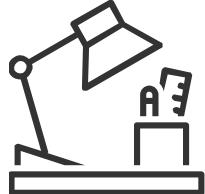
m = 10.00

8 if v(m)-v(m-1)> 0 && v(m)-v(m+1)>0

fx K>>

Workspace - local_max

Name	Value
locs	[3;6;9]
m	10
n	10
v	[0.7060;0.0318;0.2769;0.046...
vals	[0.2769;0.8235;0.9502]



Error found in line 8 again

```
8         if v(m)-v(m-1)> 0  && v(m)-v(m+1)>0
K>> v(m+1)
Index exceeds the number of array elements. Index must not exceed 10.
```

Now correct the code at line 6, the new code will be



New code

```
local_max.m  ✘ +  
1  [-] function [vals,locs]=local_max(v)  
2      n=length(v);  
3      vals=[];  
4      locs=[];  
5  
6  [-]     for m=2:n-1|  
7      |     m  
8      |     if v(m)-v(m-1)> 0 && v(m)-v(m+1)>0  
9      |     |     vals=[vals;v(m)];  
10     |     |     locs=[locs;m];  
11     |     end  
12   |   end  
13   |   figure; plot (v);  
14   |   hold on;  
15   |   plot (locs,vals,'ro');
```

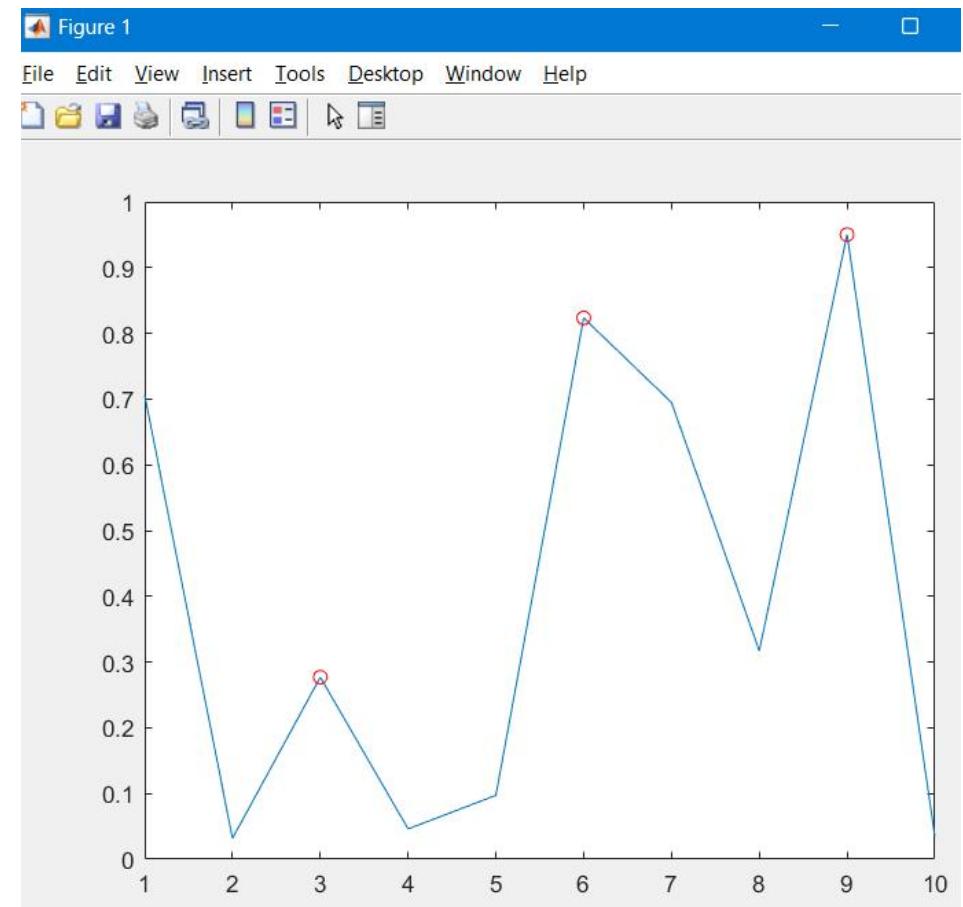
- quit debugger
- stop breakpoint
- save the file
- execute the code



Final code and result

```
local_max.m +  
1 function [vals,locs]=local_max(v)  
2 n=length(v);  
3 vals=[];  
4 locs=[];  
5  
6 for m=2:n-1  
7 m  
8 if v(m)-v(m-1)> 0 && v(m)-v(m+1)>0  
9 vals=[vals;v(m)];  
10 locs=[locs;m];  
11 end  
12 end  
13 figure; plot (v);  
14 hold on;  
15 plot (locs,vals, 'ro');
```

```
Command Window  
m = 4.00  
m = 5.00  
m = 6.00  
m = 7.00  
m = 8.00  
m = 9.00
```



MATLAB

Unit 7-Lecture 30

correcting and ending debugging

BTech (CSBS) -Semester VII

18 October 2022, 09:35AM



Control Flow and Operators

- 1) Debugging process,
- 2) preparing for debugging,
- 3) setting breakpoints,
- 4) running with breakpoints,
- 5) examining values,
- 6) correcting an M- file
- 7) **correcting and ending debugging,**



End Debugging Session

After you identify a problem, to end the debugging session, go to the **Editor** or **Live Editor** tab and click  Stop. After you end debugging, the normal `>>` prompt in the Command Window reappears in place of the `K>>` prompt. You no longer can access the function call stack.

To avoid confusion, make sure to end your debugging session every time you are done debugging. If you make changes to a file and save it while debugging, MATLAB ends the debugging session. If MATLAB becomes unresponsive when it pauses, press **Ctrl+C** to end debugging.



Quit debug mode

Syntax

```
dbquit  
dbquit all
```

dbquit terminates debug mode. The Command Window then displays the standard prompt (>>). The file being executed is not completed and no result is returned. All breakpoints remain in effect. If MATLAB® is in debug mode for more than one function, **dbquit** only terminates debugging for the active function.

For example, if you debug file1 and also debug file2, then running **dbquit** terminates debugging for file2, while file1 remains in debug mode until you run **dbquit** again. However, if you debug file3 and step into file4, then running **dbquit** terminates debugging for both file.

dbquit all ends debugging for all files simultaneously.



Quit debug mode

Create a file, `buggy.m`, that contains these statements.

```
function z = buggy(x)
n = length(x);
z = (1:n)./x;
```

Create a second file, `buggy2.m`, that contains these statements.

```
function z2 = buggy2(y)
m = length(y);
z2 = (1:m).*y;
```

Set breakpoints in `buggy` and `buggy2` and run both files. MATLAB pauses at the first line in `buggy` and `buggy2`.

```
dbstop in buggy
dbstop in buggy2
buggy(5)
buggy2(5)
```



Quit debug mode

Call the `dbstack` command to check the debugging status.

```
dbstack
```

```
In buggy2 (line 2)  
In buggy (line 2)
```

Quit debugging. MATLAB ends debugging for `buggy2`, while `buggy` remains in debug mode.

```
dbquit  
dbstack
```

```
In buggy (line 2)
```

Run `dbquit` again to exit debug mode for `buggy`.

Alternatively, `dbquit all` ends debugging for both files simultaneously.

Roll. No. A016	Name: Varun Khadayate
Class B.Tech CsBs	Batch: 1
Date of Experiment: 29-07-2021	Subject: IT/WS

1. Create a 3×5 matrix of random real numbers. Delete the third row.

```
>> mat = rand(3,5)

mat =

0.3238  0.7131  0.2204  0.7464  0.6632
0.2118  0.2151  0.2079  0.4527  0.3126
0.4927  0.4089  0.6518  0.8642  0.8317

>> mat(3,:) = []

mat =

0.3238  0.7131  0.2204  0.7464  0.6632
0.2118  0.2151  0.2079  0.4527  0.3126
```

2. Given the matrix:

```
>> mat = randi([1 20], 3,5) mat =
6      17      7      13      17
17      5      4      10      12
6      19      6      8      11
```

Why wouldn't this work:

```
mat(2:3, 1:3) = ones(2)
```

Because the left and right sides are not the same dimensions.

3. Create a three-dimensional matrix with dimensions $2 \times 4 \times 3$ in which the first "layer" is all 0s, the second is all 1s and the third is all 5s. Use size to verify the dimensions.

```
>> mat3d = zeros(2,4,3)
```

```
mat3d(:,:,1) =
```

```
0  0  0  0
0  0  0  0
```

```
mat3d(:,:,2) =
```

```
0  0  0  0
0  0  0  0
```

```
mat3d(:,:,3) =
```

```
0 0 0 0  
0 0 0 0
```

```
>> mat3d(:,:,2) = 1
```

```
mat3d(:,:,1) =
```

```
0 0 0 0  
0 0 0 0
```

```
mat3d(:,:,2) =
```

```
1 1 1 1  
1 1 1 1
```

```
mat3d(:,:,3) =
```

```
0 0 0 0  
0 0 0 0
```

```
>> mat3d(:,:,3) = 5
```

```
mat3d(:,:,1) =
```

```
0 0 0 0  
0 0 0 0
```

```
mat3d(:,:,2) =
```

```
1 1 1 1  
1 1 1 1
```

```
mat3d(:,:,3) =
```

```
5 5 5 5  
5 5 5 5
```

```
>> mat3d
```

```
mat3d(:,:,1) =
```

```
0 0 0 0
0 0 0 0
```

```
mat3d(:,:,2) =
```

```
1 1 1 1
1 1 1 1
```

```
mat3d(:,:,3) =
```

```
5 5 5 5
5 5 5 5
```

4. Create a vector x which consists of 20 equally spaced points in the range from $-\pi$ to $+\pi$. Create a y vector which is $\sin(x)$.

```
>> x = linspace(-pi,pi,20)

x =

Columns 1 through 12

-3.1416   -2.8109   -2.4802   -2.1495   -1.8188   -1.4881   -
1.1574   -0.8267   -0.4960   -0.1653    0.1653    0.4960

Columns 13 through 20

 0.8267    1.1574    1.4881    1.8188    2.1495    2.4802
2.8109    3.1416

>> y = sin(x)

y =

Columns 1 through 12

-0.0000   -0.3247   -0.6142   -0.8372   -0.9694   -0.9966   -
0.9158   -0.7357   -0.4759   -0.1646    0.1646    0.4759

Columns 13 through 20

 0.7357    0.9158    0.9966    0.9694    0.8372    0.6142
0.3247    0.0000
```

5. Create a 3×5 matrix of random integers, each in the inclusive range from -5 to 5. Get the sign of every element.

```
>> mat = randi([-5,5], 3,5)

mat =
```

```

-2    -2    4    5    -2
 5    -3    2    -2    3
-3     4    4    5    -4

>> sign(mat)

ans =

```

```

-1    -1    1    1    -1
 1    -1    1    -1    1
-1     1    1    1    -1

```

6. Find the sum $3+5+7+9+11$.

```

>> sum(3:2:11)

ans =

```

35

7. Find the sum of the first n terms of the harmonic series where n is an integer variable greater than one.

$1 + 1/2 + 1/3 + 1/4 + \dots$

```
>> n = 4
```

$n =$

4

```
>> sum(1./(1:n))
```

$ans =$

2.0833

8. Find the following sum by first creating vectors for the numerators and denominators:

$$\frac{3}{1} + \frac{5}{2} + \frac{7}{3} + \frac{9}{4}$$

```
>> num = 3:2:9
```

$num =$

3 5 7 9

```
>> denom = 1:4
```

```
denom =  
  
1 2 3 4  
  
>> fracs = num ./ denom  
  
fracs =  
  
3.0000 2.5000 2.3333 2.2500  
  
>> sum(fracs)  
  
ans =  
  
10.0833
```

9. Create a matrix and find the product of each row and column using prod.

```
>> mat = randi([1, 30], 2,3)  
  
mat =  
  
13 9 9  
28 23 2  
  
>> prod(mat)  
  
ans =  
  
364 207 18  
  
>> prod(mat,2)  
  
ans =  
  
1053  
1288
```

10. Write a relational expression for a vector variable that will verify that the last value in a vector created by cumsum is the same as the result returned by sum.

```
>> vec = 2:3:17

vec =

2 5 8 11 14 17

>> cv = cumsum(vec)

cv =

2 7 15 26 40 57

>> sum(vec) == cv(end)

ans =

logical

1
```

11. Create a vector of five random integers, each in the inclusive range from -10 to 10. Perform each of the following:

```
>> vec = randi([-10, 10], 1,5)

vec =

-3 10 -1 3 2
```

a. subtract 3 from each element

```
>> vec-3

ans =

-6 7 -4 0 -1
```

b. count how many are positive

```
>> sum(vec > 0)

ans =

3
```

c. get the cumulative minimum

```
>> cummin(vec)
```

ans =

```
-3 -3 -3 -3 -3
```

12. Create a 3 x 5 matrix. Perform each of the following:

```
>> mat = randi([-10 10], 3,5)
```

mat =

```
1 7 10 10 -7
-4 -4 5 -2 10
-5 6 -3 5 9
```

a. Find the maximum value in each column.

```
>> max (mat)
```

ans =

```
1 7 10 10 10
```

b. Find the maximum value in each row.

```
>> max (mat, [], 2)
```

ans =

```
10
10
9
```

c. Find the maximum value in the entire matrix.

```
>> max (max (mat) )
```

ans =

```
10
```

d. Find the cumulative maxima.

```
>> cummax (mat)
```

ans =

```
1 7 10 10 -7
1 7 10 10 10
1 7 10 10 10
```

13. Find two ways to create a 3×5 matrix of all 100s (Hint: use ones and zeros).

```
>> ones(3,5)*100

ans =

100 100 100 100 100
100 100 100 100 100
100 100 100 100 100

>> zeros(3,5)+100

ans =

100 100 100 100 100
100 100 100 100 100
100 100 100 100 100
```

14. Given the two matrices:

	A			B	
1	2	3	2	4	1
4	-1	6	1	3	0

Perform the following operations:

```
>> A = [1,2,3;4,-1,6]

A =

1 2 3
4 -1 6

>> B = [2,4,1;1,3,0]

B =

2 4 1
1 3 0
```

a. $A + B$

```
>> A + B

ans =

3 6 4
5 2 6
```

b. $A - B$

```
>> A - B

ans =

-1 -2 2
3 -4 6
```

c. $A * B$

```
>> A .* B

ans =

 2     8     3
 4    -3     0
```

15. The built-in function `clock` returns a vector that contains 6 elements: the first three are the current date (year, month, day) and the last three represent the current time in hours, minutes, and seconds. The seconds is a real number, but all others are integers. Store the result from `clock` in a variable called `myc`. Then, store the first three elements from this variable in a variable `today` and the last three elements in a variable `now`. Use the `fix` function on the vector variable `now` to get just the integer part of the current time.

```
>> myc = clock

myc =

 1.0e+03 *
 2.0220    0.0070    0.0290    0.0160    0.0030    0.0567

>> today = myc(1:3)

today =

 2022      7      29

>> now = myc(4:end)

now =

 16.0000    3.0000    56.7430

>> fix(now)

ans =

 16      3      56
```

16. A vector `v` stores for several employees of the Green Fuel Cells Corporation their hours worked one week followed for each by the hourly pay rate. For example, if the variable stores

```
>> v
v =
 33.0000 10.5000 40.0000 18.0000 20.0000 7.5000
```

that means the first employee worked 33 hours at \$10.50 per hour, the second worked 40 hours at \$18 an hour, and so on. Write code that will separate this into two vectors, one that stores the hours worked and another that stores the hourly rates. Then, use the array multiplication operator to create a vector, storing in the new vector the total pay for every employee.

```

>> v = [33.0000 10.5000 40.0000 18.0000 20.0000 7.5000]

v =
33.0000 10.5000 40.0000 18.0000 20.0000 7.5000

>> hours = v(1:2:length(v))

hours =
33 40 20

>> payrate = v(2:2:length(v))

payrate =
10.5000 18.0000 7.5000

>> totpay = hours .* payrate

totpay =
346.5000 720.0000 150.0000

```

17. A company is calibrating some measuring instrumentation and has measured the radius and height of one cylinder 10 separate times; they are in vector variables r and h . Find the volume from each trial, which is given by $\pi r^2 h$. Also use logical indexing first to make sure that all measurements were valid (> 0).

```

>> r = [5.501 5.5 5.499 5.498 5.5 5.5 5.52 5.51 5.5 5.48];
>> h = [11.11 11.1 11.1 11.12 11.09 11.11 11.11 11.1 11.08 11.11];

>> r = [5.501 5.5 5.499 5.498 5.5 5.5 5.52 5.51 5.5 5.48]

r =
5.5010 5.5000 5.4990 5.4980 5.5000 5.5000
5.5200 5.5100 5.5000 5.4800

>> h = [11.11 11.1 11.1 11.12 11.09 11.11 11.11 11.1 11.08 11.11]

h =
11.1100 11.1000 11.1000 11.1200 11.0900 11.1100
11.1100 11.1000 11.0800 11.1100

>> vol = pi * r.^2 .* h

vol =
1.0e+03 *
1.0562 1.0549 1.0545 1.0560 1.0539 1.0558
1.0635 1.0587 1.0530 1.0482

```

18. For the following matrices A, B, and C:

$$A = \begin{bmatrix} 1 & 4 \\ 3 & 2 \end{bmatrix} \quad B = \begin{bmatrix} 2 & 1 & 3 \\ 1 & 5 & 6 \\ 3 & 6 & 0 \end{bmatrix} \quad C = \begin{bmatrix} 3 & 2 & 5 \\ 4 & 1 & 2 \end{bmatrix}$$

```
>> A = [1 4;3,2]  
  
A =  
  
1 4  
3 2  
  
>> B = [2 1 3;1 5 6;3 6 0]  
  
B =  
  
2 1 3  
1 5 6  
3 6 0  
  
>> C = [3 2 5;4 1 2]  
  
C =  
  
3 2 5  
4 1 2
```

a. Give the result of $3 \cdot A$.

```
>> 3 * A  
  
ans =  
  
3 12  
9 6
```

b. Give the result of $A \cdot C$.

```
>> A * C  
  
ans =  
  
19 6 13  
17 8 19
```

c. Are there any other matrix multiplications that can be performed? If so, list them.

```
>> C * B  
  
ans =  
  
23 43 21  
15 21 18
```

19. For the following vectors and matrices, A, B, and C:

$$A = \begin{bmatrix} 4 & 1 & -1 \\ 2 & 3 & 0 \end{bmatrix} \quad B = [1 \ 4] \quad C = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

```
>> A = [4 1 -1; 2 3 0]
```

A =

$$\begin{matrix} 4 & 1 & -1 \\ 2 & 3 & 0 \end{matrix}$$

```
>> B = [1 4]
```

B =

$$\begin{matrix} 1 & 4 \end{matrix}$$

```
>> C = [2; 3]
```

C =

$$\begin{matrix} 2 \\ 3 \end{matrix}$$

a. A * B

```
>> A * B
```

Error using *

Incorrect dimensions for matrix multiplication. Check that the number of columns in the first matrix matches the number of rows in the second matrix. To operate on each element of the matrix individually, use TIMES (.*) for elementwise multiplication.

b. B * C

```
>> B * C
```

ans =

$$14$$

c. C * B

```
>> C * B
```

ans =

$$\begin{matrix} 2 & 8 \\ 3 & 12 \end{matrix}$$

20. The matrix variable rainmat stores the total rainfall in inches for some districts for the years 2010-2013. Each row has the rainfall amounts for a given district. For example, if rainmat has the value:

```
>> rainmat
ans =
25 33 29 42
```

53 44 40 56
etc.

district 1 had 25 inches in 2010, 33 in 2011, etc. Write expression(s) that will find the number of the district that had the highest total rainfall for the entire four year period.

```
>> rainmat = [25 33 29 42; 53 44 40 56]
```

rainmat =

25 33 29 42
53 44 40 56

```
>> large = max(max(rainmat))
```

large =

56

```
>> linind = find(rainmat== large)
```

linind =

8

```
>> floor(linind/4)
```

ans =

2

21. Generate a vector of 20 random integers, each in the range from 50 to 100. Create a variable evens that stores all of the even numbers from the vector, and a variable odds that stores the odd numbers.

```
>> nums = randi([50, 100], 1, 20)
```

nums =

99 95 92 90 70 87 74 72 93 80 59
55 61 58 71 52 77 65 86 56

```
>> evens = nums(rem(nums, 2)==0)
```

evens =

92 90 70 74 72 80 58 52 86 56

```
>> odds = nums(rem(nums, 2)~==0)
```

odds =

99 95 87 93 59 55 61 71 77 65

22. Assume that the function diff does not exist. Write your own expression(s) to accomplish the same thing for a vector.

```
>> vec = [5 11 2 33 -4]

vec =
5      11      2      33      -4

>> v1 = vec(2:end)

v1 =
11      2      33      -4

>> v2 = vec(1:end-1)

v2 =
5      11      2      33

>> v1-v2

ans =
6      -9      31      -37
```

23. Create a vector variable vec; it can have any length. Then, write assignment statements that would store the first half of the vector in one variable and the second half in another. Make sure that your assignment statements are general, and work whether vec has an even or odd number of elements (Hint: use a rounding function such as fix).

```
>> vec = 1:9

vec =
1  2  3  4  5  6  7  8  9

>> fhalf = vec(1:fix(length(vec)/2))

fhalf =
1  2  3  4

>> shalf = vec(fix(length(vec)/2)+1:end)

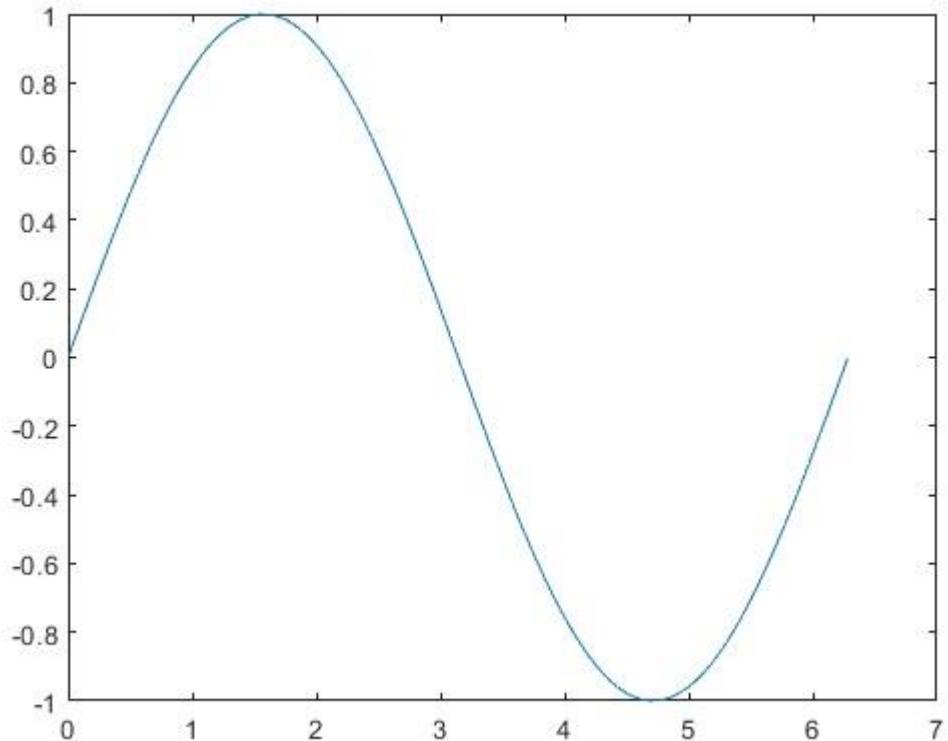
shalf =
5  6  7  8  9
```

Roll No.: A016	Name: Varun Khadayate
Class: B. Tech CsBs	Batch: 1
Date of Experiment: 18-09-2022	Subject: ITWS

Lab 5: Basic Plotting

- Creates a 2-D line plot of the data in y versus the corresponding values in x . Create x as a vector of linearly spaced values between 0 and 2π . Use an increment of $\pi/100$ between the values. Create y as sine values of x . Calling the plot function will display the x versus y data on a scaled viewing domain and range.

```
x = 0:pi/100:2*pi;
y = sin(x);
plot(x,y)
```



- Explore various specifications of plot command and put it in a tabular format.

Specification	Purpose	Syntax	Example
xlabel, ylabel	xlabel(<code>txt</code>) labels the x-axis of the current	<code>xlabel('string')</code> <code>ylabel('string')</code>	<code>xlabel('X-Axis')</code> <code>ylabel('Y-Axis')</code>

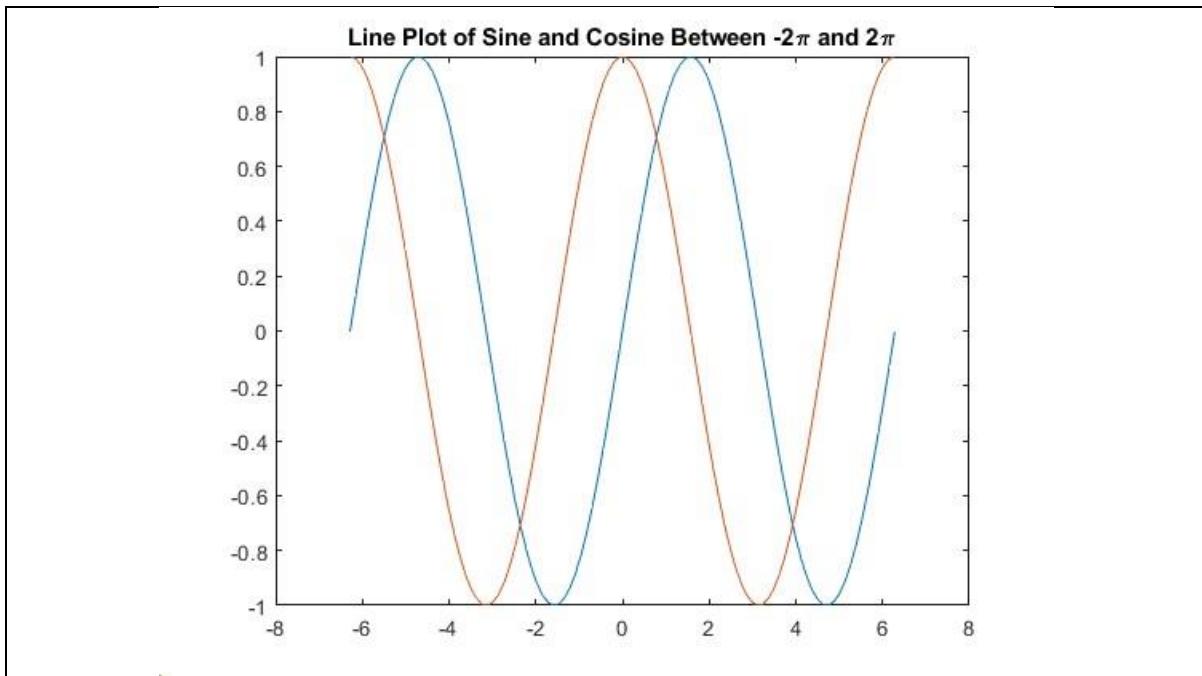
	axes or standalone visualization. <code>ylabel(txt)</code> labels the y-axis of the current axes or standalone visualization.		
<code>title</code>	<code>title(titleText)</code> adds the specified title to the current axes or standalone visualization.	<code>title('string')</code>	<code>title('Plot between X-axis and Y-axis')</code>
<code>legend</code>	The legend automatically updates when you add or delete data series from the axes. This command creates a legend in the current axes, which is returned by the <code>gca</code> command.	<code>legend</code>	<code>legend('x','y')</code>

3. Create `x` as 100 linearly spaced values between -2π and 2π . Create `y1` and `y2` as sine and cosec values of `x`. Plot both sets of data. Add a title to the chart by using the `title` function. To display the Greek symbol π , use `pi`.

```

x = linspace(-2*pi,2*pi,100);
y1 = sin(x);
y2 = cos(x);
figure
plot(x,y1,x,y2)
title('Line Plot of Sine and Cosine Between -2\pi and 2\pi')

```

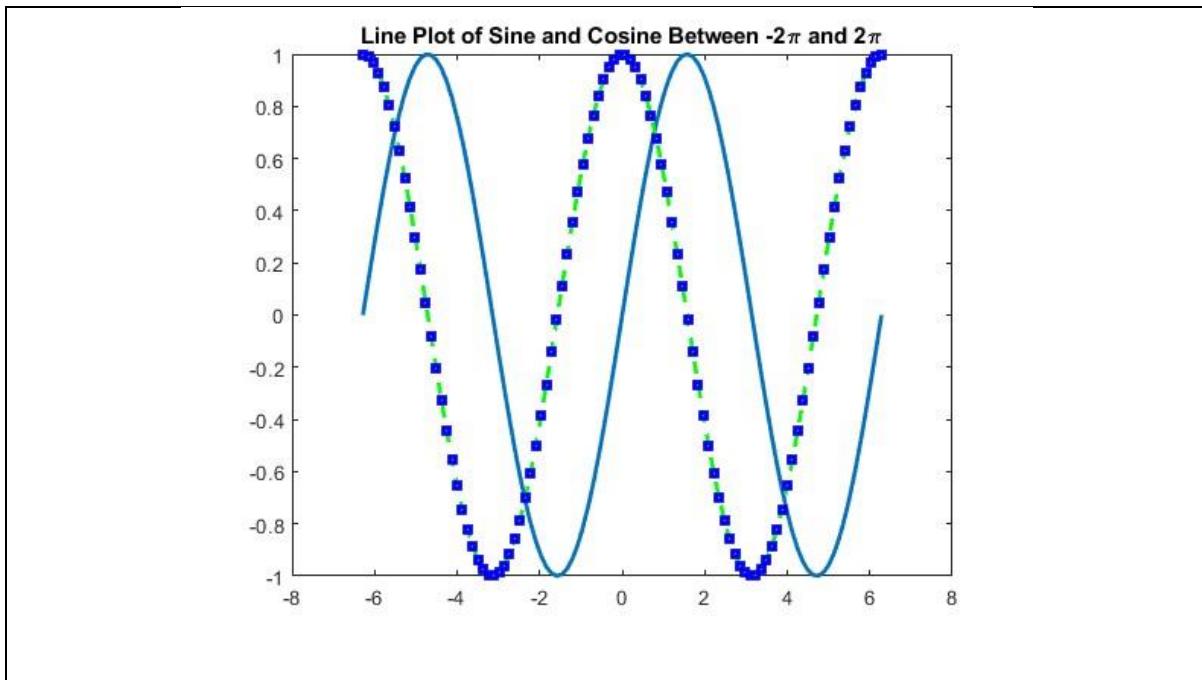


4. In the above plot, use the LineSpec option to specify a dashed green line with square markers. Use Name,Value pairs to specify the line width, point marker size, and point marker colors. Set the marker edge color to blue and set the marker face color using an RGB color value.

```

x = linspace(-2*pi,2*pi,100);
y1 = sin(x);
y2 = cos(x);
figure
plot(x,y1,x,y2, '--gs',...
    'LineWidth',2,...
    'MarkerSize',5,...
    'MarkerEdgeColor','b',...
    'MarkerFaceColor',[0.5,0.5,0.5])
title('Line Plot of Sine and Cosine Between -2\pi and 2\pi')

```



5. Display Multiple Plots in a Figure Window : Call the tiledlayout function to create a 2-by-1 tiled chart layout. Call the nexttile function to create an axes object and return the object as ax1. Create the top plot by passing ax1 to the plot function. Add a title and y-axis label to the plot by passing the axes to the title and ylabel functions. Repeat the process to create the bottom plot.

($y = \sin(3x)$ %for 1st plot)

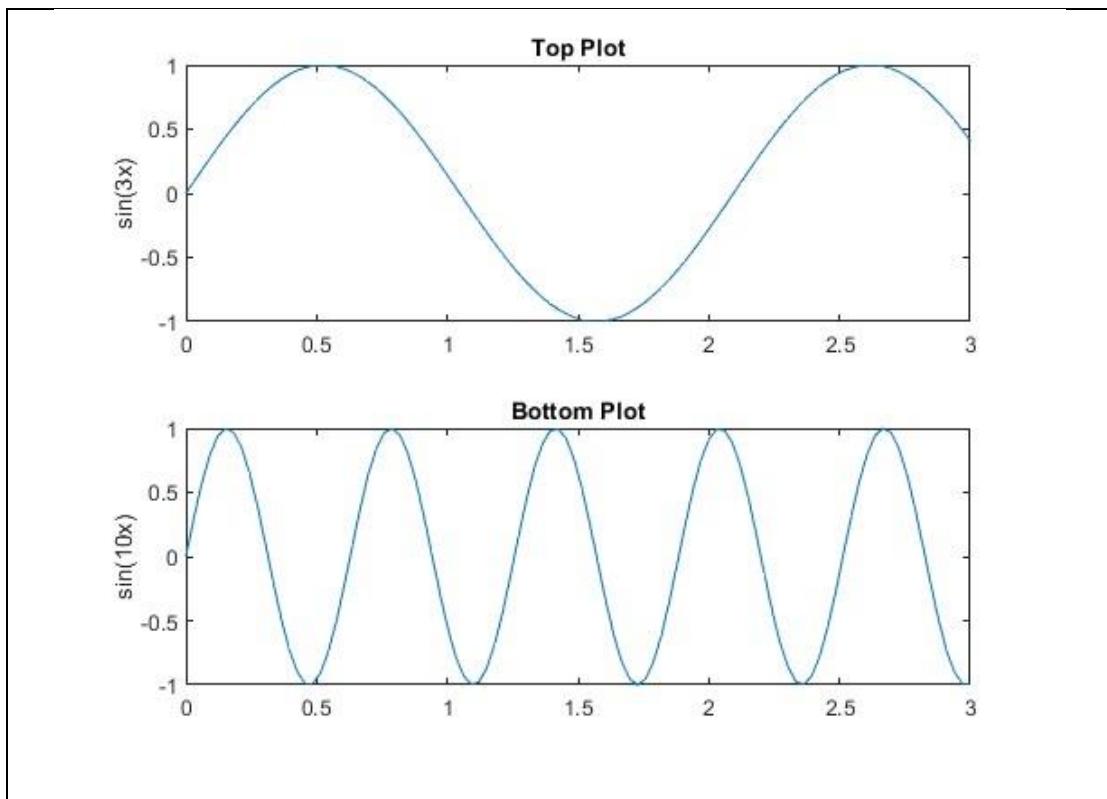
($y = \sin(10x)$ %for 2nd plot)

```

x = linspace(0,3);
y1 = sin(3*x);
y2 = sin(10*x);
tiledlayout(2,1)

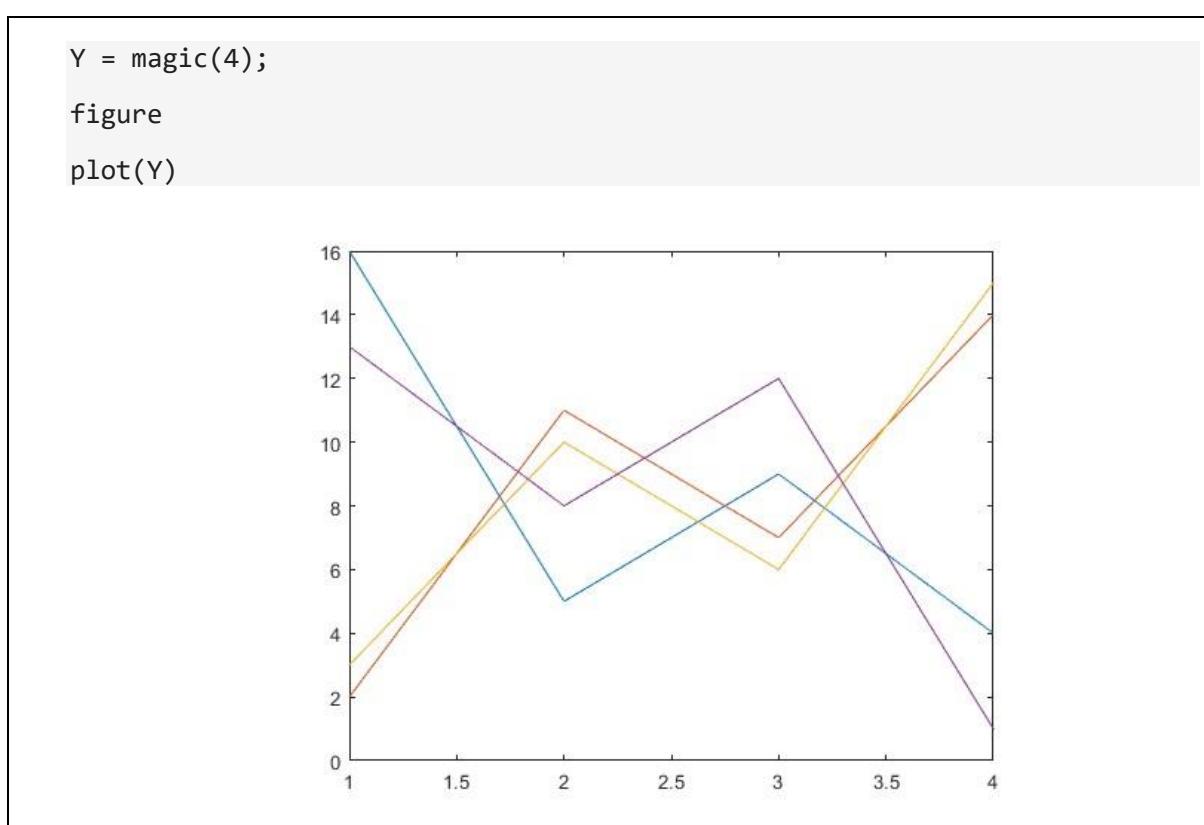
% Top plot
ax1 = nexttile;
plot(ax1,x,y1)
title(ax1,'Top Plot')
ylabel(ax1,'sin(3x)')

% Bottom plot
ax2 = nexttile;
plot(ax2,x,y2)
title(ax2,'Bottom Plot')
ylabel(ax2,'sin(10x)')
```



6. Define Y as the 4-by-4 matrix returned by the `magic` function. Create a 2-D line plot of Y . MATLAB® plots each matrix column as a separate line.

```
Y = magic(4);
figure
plot(Y)
```



Roll No.: A016	Name: Varun Khadayate
Class: B. Tech CsBs	Batch: 1
Date of Experiment: 18-09-2022	Subject: ITWS

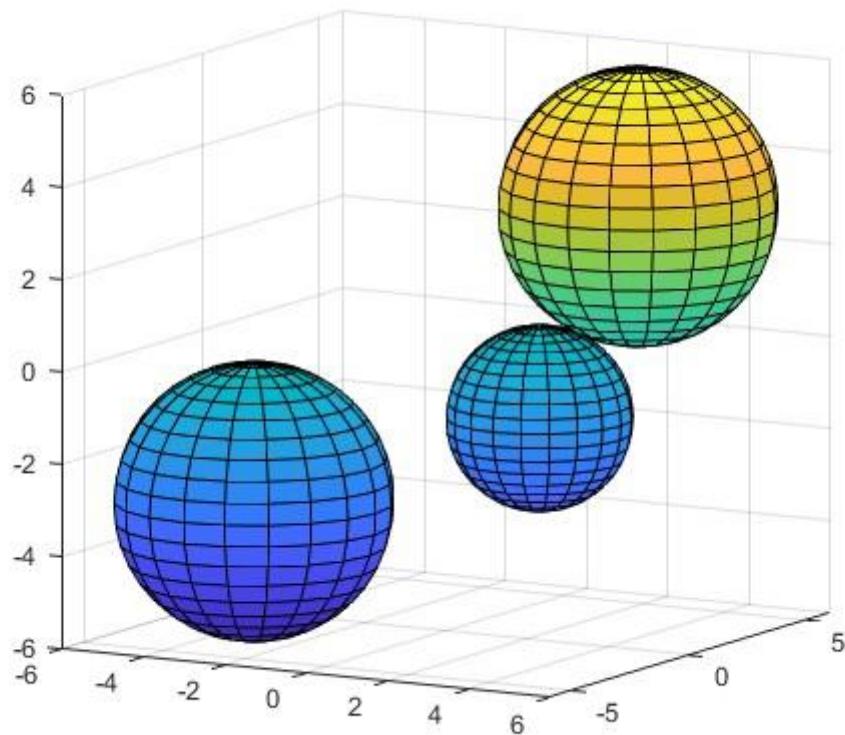
Lab 6: M-script file

1.

- a. Create a file called mysphere.mat
- b. Create and plot a sphere using sphere function. Sphere function creates a unit circle which can be scaled using surf function (hint: try different values of 'r' using surf function).
- c. Run and observe the output
- d. What is your observation?

While running the code with different value of r there are different sphere, and we can see that the dimension also changes. Below is the screenshot of 3 sphere in one graph

```
[x y z] = sphere;
a=[3 3 3 3;-3 -3 -3 3;0 4 -2 2];
s1=surf(x*a(1,4)+a(1,1),y*a(1,4)+a(1,2),z*a(1,4)+a(1,3));
hold on
s2=surf(x*a(2,4)+a(2,1),y*a(2,4)+a(2,2),z*a(2,4)+a(2,3));
s3=surf(x*a(3,4)+a(3,1),y*a(3,4)+a(3,2),z*a(3,4)+a(3,3));
daspect([1 1 1])
view(30,10)
```



2.

- a. Create an excel sheet consisting the following data :

Temperatures of Cities as on 20.6.2006		
City	Max.	Min.
Ahmedabad	38°C	29°C
Amritsar	37°C	26°C
Bangalore	28°C	21°C
Chennai	36°C	27°C
Delhi	38°C	28°C
Jaipur	39°C	29°C
Jammu	41°C	26°C
Mumbai	32°C	27°C

- b. Explore xlsread function using help
c. Read the excel file created in step a) using xlsread function
d. d) Create an M-File and write a script which finds the following:
i. Compare and identify the city that recorded the max temperature
ii. Plot the given data using bar graph

```
help xlsread
```

```
xlsread Read Microsoft Excel spreadsheet file.
```

```
*** xlsread is not recommended ***
```

```
See readtable, readmatrix, or readcell.
```

```
-----  
[NUM,TXT,RAW]=xlsread(FILE) reads data from the first worksheet in the Microsoft  
Excel spreadsheet file named FILE and returns the numeric data in array NUM.  
Optionally, returns the text fields in cell array TXT, and the unprocessed data  
(numbers and text) in cell array RAW.
```

```
[NUM,TXT,RAW]=xlsread(FILE,SHEET) reads the specified worksheet.
```

```
[NUM,TXT,RAW]=xlsread(FILE,SHEET,RANGE) reads from the specified SHEET and RANGE.  
Specify RANGE using the syntax 'C1:C2', where C1 and C2 are opposing corners of  
the region. Not supported for XLS files in BASIC mode.
```

```
[NUM,TXT,RAW]=xlsread(FILE,SHEET,RANGE,'basic') reads from the spreadsheet in  
BASIC mode, the default on systems without Excel for Windows. RANGE is supported  
for XLSX files only.
```

```
[NUM,TXT,RAW]=xlsread(FILE,RANGE) reads data from the specified RANGE of the  
first worksheet in the file. Not supported for XLS files in BASIC mode.
```

```
The following syntaxes are supported only on Windows systems with Excel software:
```

```
[NUM,TXT,RAW]=xlsread(FILE,-1) opens an Excel window to select data  
interactively.
```

[NUM,TXT,RAW,CUSTOM]=**xlsread**(FILE,SHEET,RANGE,'',FUNCTIONHANDLE) reads from the spreadsheet, executes the function associated with FUNCTIONHANDLE on the data, and returns the final results. Optionally, returns additional CUSTOM output, which is the second output from the function. **xlsread** does not change the data stored in the spreadsheet.

Input Arguments:

FILE Name of the file to read. SHEET Worksheet to read. One of the following:
* The worksheet name.
* Positive, integer-valued scalar indicating the worksheet index.

RANGE Character vector or string that specifies a rectangular portion of the worksheet to read. Not case sensitive. Use Excel A1 reference style. If you do not specify a SHEET, RANGE must include both corners and a colon character (:), even for a single cell (such as 'D2:D2').

'basic' Flag to request reading in BASIC mode, which is the default for systems without Excel for Windows. In BASIC mode, **xlsread**:
* Reads XLS, XLSX, XLSM, XLTX, and XLTM files only.
* Does not support an xlRange input when reading XLS files. In this case, use '' in place of xlRange.
* For XLS files, requires a name to specify the SHEET, and the name is case sensitive.
* Does not support function handle inputs.
* Imports all dates as Excel serial date numbers. Excel serial date numbers use a different reference date than MATLAB date numbers.

-1 Flag to open an interactive Excel window for selecting data. Select the worksheet, drag and drop the mouse over the range you want, and click OK. Supported only on Windows systems with Excel software.

FUNCTIONHANDLE

Handle to your custom function. When **xlsread** calls your function, it passes a range interface from Excel to provide access to the data. Your function must include this interface (of type 'Interface.Microsoft_Excel_5.0_Object_Library.Range', for example) both as an input and output argument.

Notes:

- * On Windows systems with Excel software, **xlsread** reads any file format recognized by your version of Excel, including XLS, XLSX, XLSB, XLSM, and HTML-based formats.
- * If your system does not have Excel for Windows, **xlsread** operates in BASIC mode (see Input Arguments).
- * **xlsread** imports formatted dates as character vectors (such as '10/31/96'), except in BASIC mode. In BASIC mode, **xlsread** imports all dates as serial date

numbers. Serial date numbers in Excel use different reference dates than date numbers in MATLAB. For information on converting dates, see the documentation on importing spreadsheets.

```
*****  
*** xlsread is not recommended ***  
*****
```

Compatibility Considerations:

If the spreadsheet columns are mixed types, but has homogenous types along the columns, use READTABLE.

For reading numeric data, replace the following:

```
NUM = xlsread(FILE);
```

Instead Use:

```
NUM = readmatrix(FILE);
```

For reading text data, replace the following:

```
[~,TXT] = xlsread(FILE);
```

Instead Use:

```
TXT = readmatrix(FILE,"OutputType","char");
```

When reading RAW data, replace the following:

```
[~,~,RAW] = xlsread(FILE);
```

Instead Use:

```
RAW = readcell(FILE);
```

For specifying sheet and range, replace the following:

```
... = xlsread(FILE,SHEET,RANGE);
```

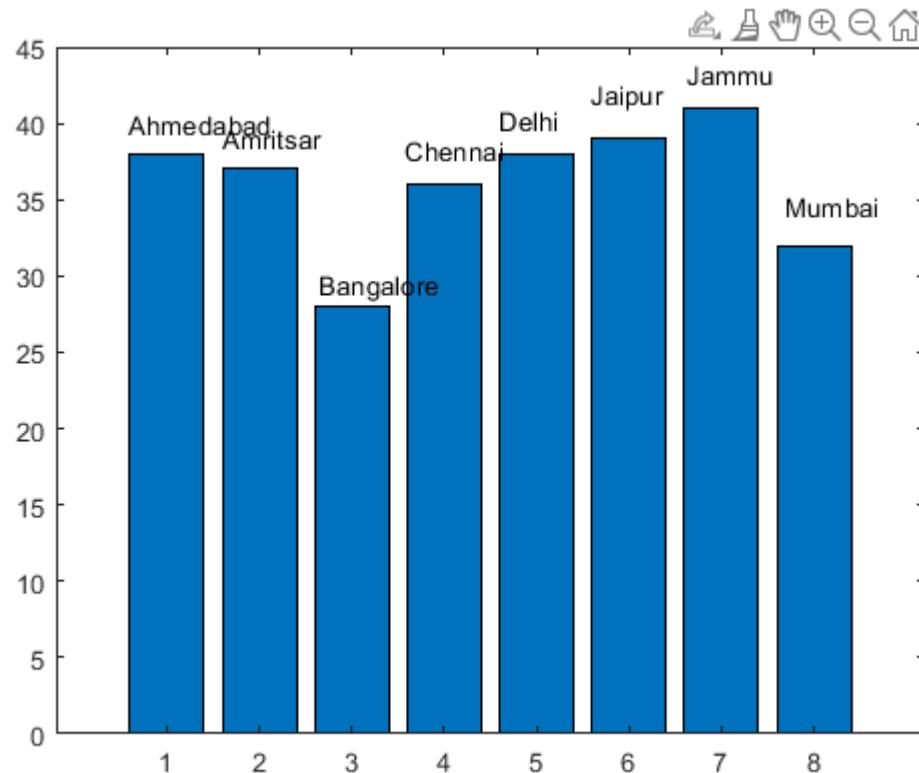
Instead use:

```
... = read<type>(FILE,"Sheet",SHEET,"RANGE",RANGE);
```

See also readtable, readmatrix, readcell, detectImportOptions

Documentation for xlsread

```
data = readtable("Lab_6_1.xlsx");
bar(data.Max)
cont =
char('Ahmedabad','Amritsar','Bangalore','Chennai','Delhi','Jaipur','Jammu',
'Mumbai');
for i = 1:8,
    gtext (cont (i,:));
end
```



Hence Jammu has recorded the Max Temperature of 41 Degrees

3.

- a. Create an excel file with the following data:

Roll No	Name	Accountancy	English	Maths	Economics	Business Studies
1	Akhilesh	97	36	47	13	34
2	Ruchi	69	85	86	51	53
3	Bhawna	19	72	41	53	40
4	Isha	76	68	46	11	22
5	Chetan	55	31	56	99	93
6	Neeti	84	57	68	30	31
7	Chanchal	18	46	51	63	22
8	Preeti	93	93	31	93	20
9	Richa	33	89	55	46	69
10	Manish	21	27	84	82	96
11	Karun	13	48	27	26	38
12	Madhur	85	74	26	53	84
13	Nitesh	28	31	27	77	17

- b. Read the file in matlab using xlsread
 c. Create a function tot_marks to calculate total marks of each Student.
 d. Create function avg to calculate average marks of each student.
 e. Plot the marks obtained in step c. using

```

data_1 = readtable("Lab_6_2.xlsx")

Warning: Column headers from the file were modified to make them valid MATLAB
identifiers before creating variable names for the table. The original column headers
are saved in the VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table
variable names.

data_1 = 13x7 table

```

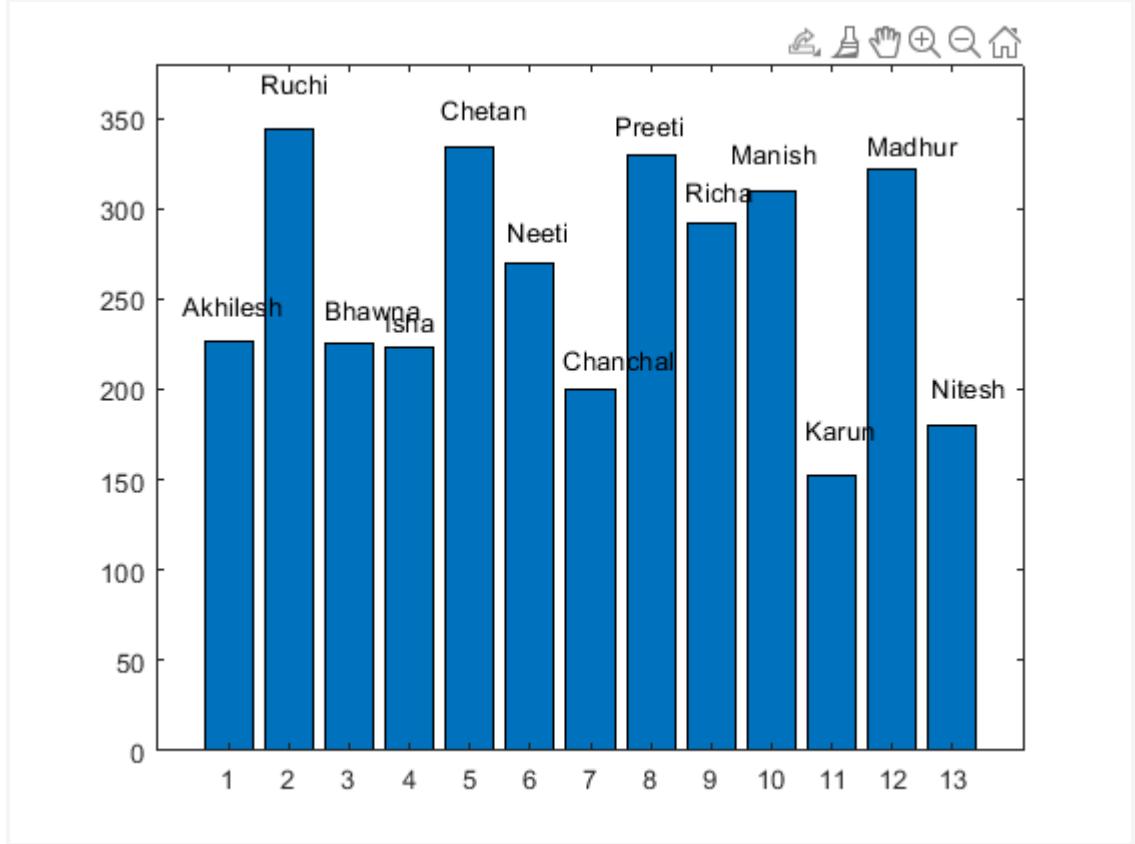
	RollNo	Name	Accountancy	English	Maths	Economics	BusinessStudies
1	1	'Akhilesh'	97	36	47	13	34
2	2	'Ruchi'	69	85	86	51	53
3	3	'Bhawna'	19	72	41	53	40
4	4	'Isha'	76	68	46	11	22
5	5	'Chetan'	55	31	56	99	93
6	6	'Neeti'	84	57	68	30	31
7	7	'Chanchal'	18	46	51	63	22
8	8	'Preeti'	93	93	31	93	20
9	9	'Richa'	33	89	55	46	69
10	10	'Manish'	21	27	84	82	96
11	11	'Karun'	13	48	27	26	38
12	12	'Madhur'	85	74	26	53	84
13	13	'Nitesh'	28	31	27	77	17

```

vars = ["Accountancy", "English", "Maths", "Economics", "BusinessStudies"];
data_1.TestMean = sum(data_1{:,vars},2);
bar(data_1.TestMean)
cont = char('Akhilesh', 'Ruchi', 'Bhawna', 'Isha', 'Chetan',...
    'Neeti', 'Chanchal', 'Preeti', 'Richa', 'Manish', 'Karun', 'Madhur', 'Nitesh');
ylim([0 380])
for i = 1:13,
    gtext (cont (i,:));

```

```
end
```



```
data_1.Average = mean(data_1{:,vars},2)
```

```
data_1 = 13x9 table
```

	RollNo	Name	Accountancy	English	Maths	Economics	BusinessStudies	TestMean	Average
1	1	'Akhilesh'	97	36	47	13	34	227	45.4000
2	2	'Ruchi'	69	85	86	51	53	344	68.8000
3	3	'Bhawna'	19	72	41	53	40	225	45
4	4	'Isha'	76	68	46	11	22	223	44.6000
5	5	'Chetan'	55	31	56	99	93	334	66.8000
6	6	'Neeti'	84	57	68	30	31	270	54
7	7	'Chanchal'	18	46	51	63	22	200	40
8	8	'Preeti'	93	93	31	93	20	330	66

9	9	'Rich a'	33	89	55	46	69	292	58.4 000
1 0	10	'Mani sh'	21	27	84	82	96	310	62
1 1	11	'Karu n'	13	48	27	26	38	152	30.4 000
1 2	12	'Mad hur'	85	74	26	53	84	322	64.4 000
1 3	13	'Nites h'	28	31	27	77	17	180	36

Roll No.: A016	Name: Varun Khadayate
Class: B. Tech CsBs	Batch: 1
Date of Experiment: 18-09-2022	Subject: ITWS

Lab 7: M-script file-extended

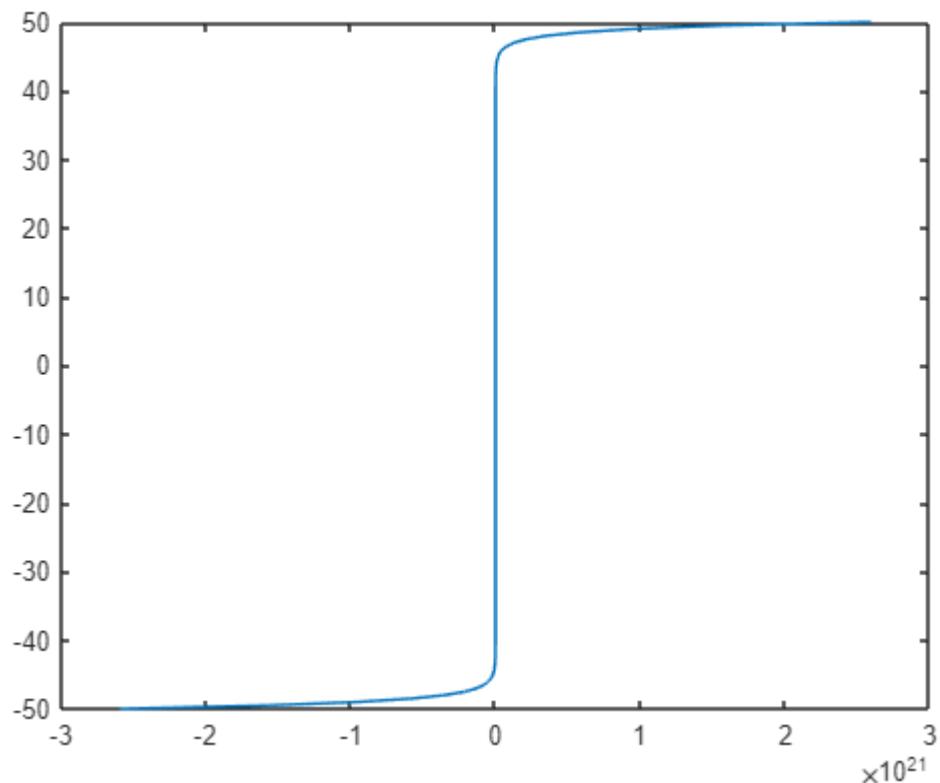
1.

Write three MATLAB functions to calculate the hyperbolic sine, cosine, and tangent functions:

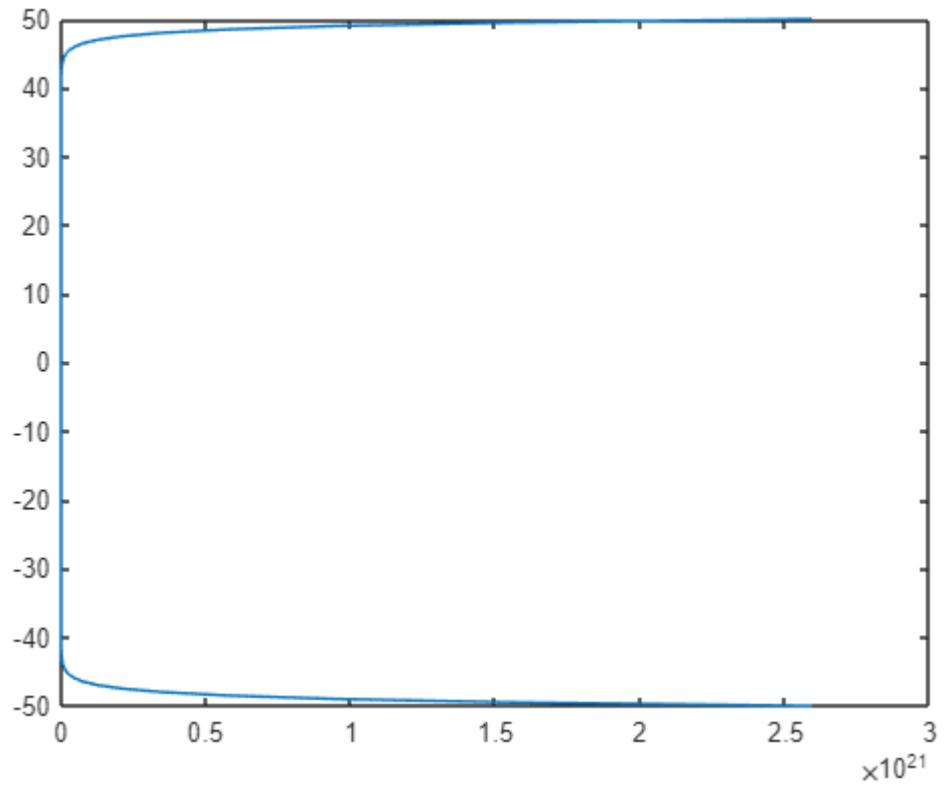
$$\sinh(x) = \frac{e^x - e^{-x}}{2} \quad \cosh(x) = \frac{e^x + e^{-x}}{2} \quad \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Use your functions to plot the shapes of the hyperbolic sine, cosine, and tangent functions.

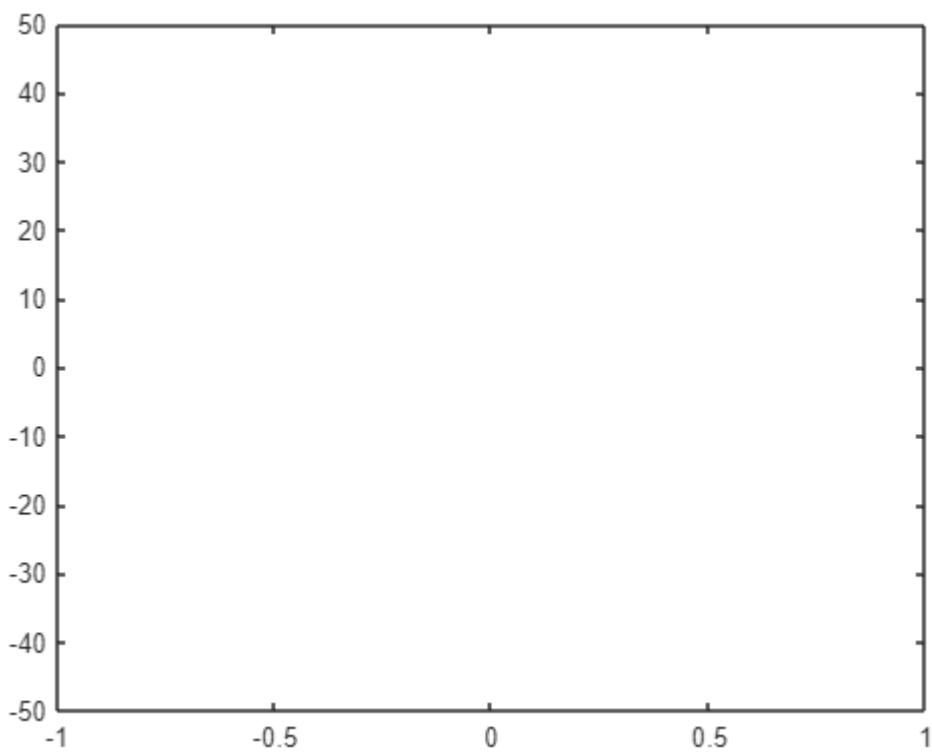
```
x = linspace(-50,50,200) ;
y_sinh = (exp(x) - exp(-x))/2;
figure;
plot(y_sinh,x);
```



```
x = linspace(-50,50,200) ;
y_cosh = (exp(x) + exp(-x))/2;
figure;
plot(y_cosh,x);
```



```
x = linspace(-50,50,200) ;
y_tanh = (exp(x) - exp(-x))/(exp(x) + exp(-x));
figure;
plot(y_tanh,x);
```



2.

The gravitational force F between two bodies of masses and is given by the equation

$$F = \frac{Gm_1m_2}{r^2}$$

where G is the gravitation constant ($6.672 \times 10^{11} \text{ N m}^2 / \text{kg}^2$), and m_1 and m_2 are the masses of the bodies in kilograms, and r is the distance between the two bodies. Write a function to calculate the gravitational force between two bodies given their masses and the distance between them. Test your function by determining the force on an 800 kg satellite in orbit 38,000 km above the Earth. (The mass of the Earth is $5.98 \times 10^{24} \text{ kg}$.)

```
m1 = 800;
m2 = 5.98E24;
r = 38000;
G = 6.672E-11;
force = G * m1 * m2 / r^2
Output
force = 2.2104e+08
```

3.

- a) Write a program that will solve for the roots of a quadratic equation and display the answer on the screen. The inputs required by this program are the coefficients a,b and c (to be taken from the user) of the quadratic equation ($ax^2+bx+c = 0$)

```
a = input ('Enter the coefficient A: ');
b = input ('Enter the coefficient B: ');
c = input ('Enter the coefficient C: ');
d = b^2 - 4 * a * c;
x1 = ( -b + sqrt(d) ) / ( 2 * a );
x2 = ( -b - sqrt(d) ) / ( 2 * a );
fprintf(['The coefficient A: %d \n' ...
'The coefficient B: %d\n'...
'the coefficient C: %d\n'...
'The roots of this equation (%d)x^2 + (%d)x + (%d) are: %d and
%d'],a,b,c,a,b,c,x1,x2);
```

- b) Test the program for :

i. $x^2+5x+6 = 0$

```
The coefficient A: 1
The coefficient B: 5
the coefficient C: 6
The roots of this equation (1)x^2 + (5)x + (6) are: -2 and -3
```

ii. $x^2+2x+5 = 0$

```
The coefficient A: 1
The coefficient B: 2
the coefficient C: 5
The roots of this equation (1)x^2 + (2)x + (5) are: -1 and -1
```

Roll No.: A016	Name: Varun Khadayate
Class: B. Tech CsBs	Batch: 1
Date of Experiment: 20-09-2022	Subject: ITWS

Lab 8

CircleIO.m

```
% This script calculates the area of a circle
% It prompts the user for the radius

%Prompt the user for the radius and calculate the area based on that radius
fprintf('Note: the units will be inches. \n')
radius = input('Please enter the radius: ');
area = pi * (radius ^2) ;

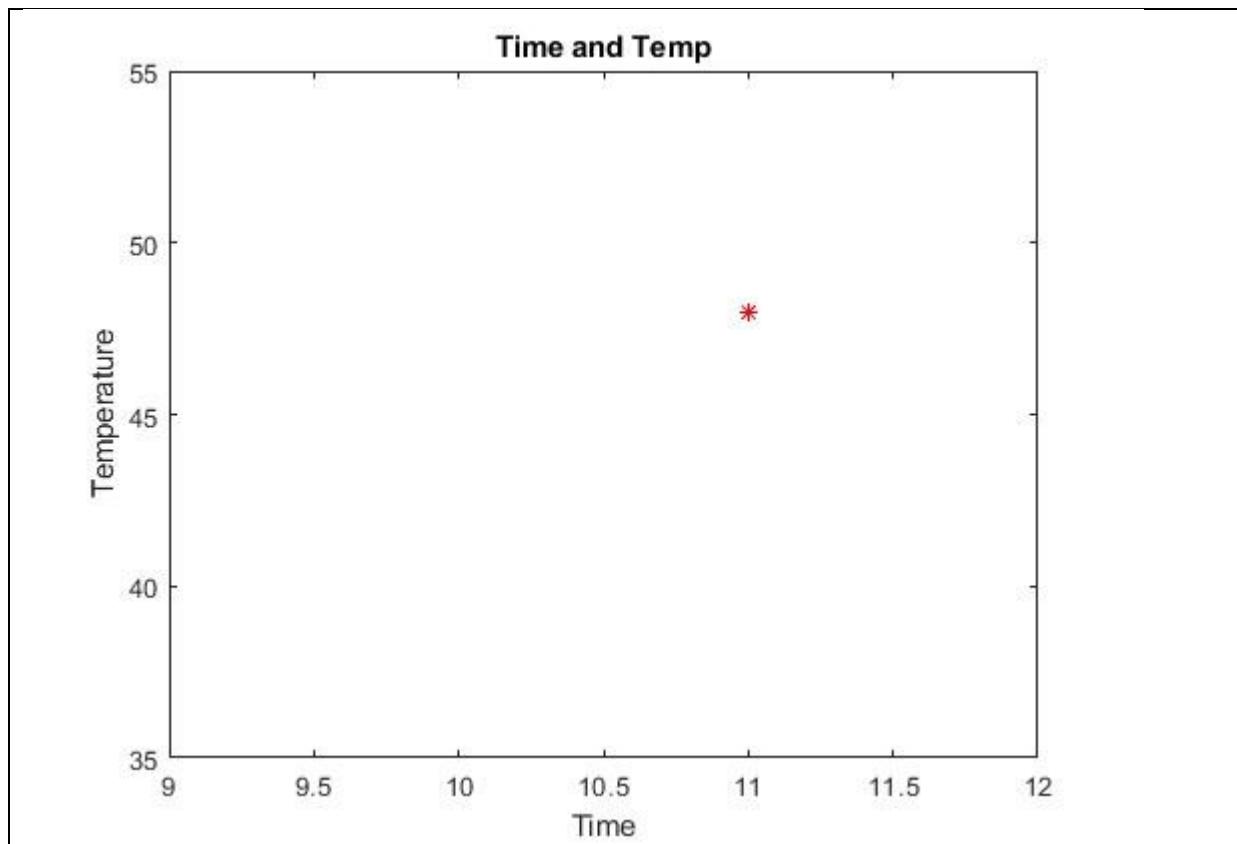
%Print all variables in a sentence format
fprintf('For a circle with a radius of %.2f inches, \n' ,radius)
fprintf('the area is %.2f inches squared \n' , area)

>> CircleIO
Note: the units will be inches.
Please enter the radius: 5
For a circle with a radius of 5.00 inches,
the area is 78.54 inches squared
```

plotonepoint.m

```
% This is a really simple plot of just one point !
% Create coordinate variables and plot a red '*'
x = 11;
y = 48;

plot(x, y, 'r*')
% Change the axes and label them
axis([9 12 35 55])
xlabel('Time')
ylabel('Temperature')
% Put a title on the plot
title('Time and Temp')
```



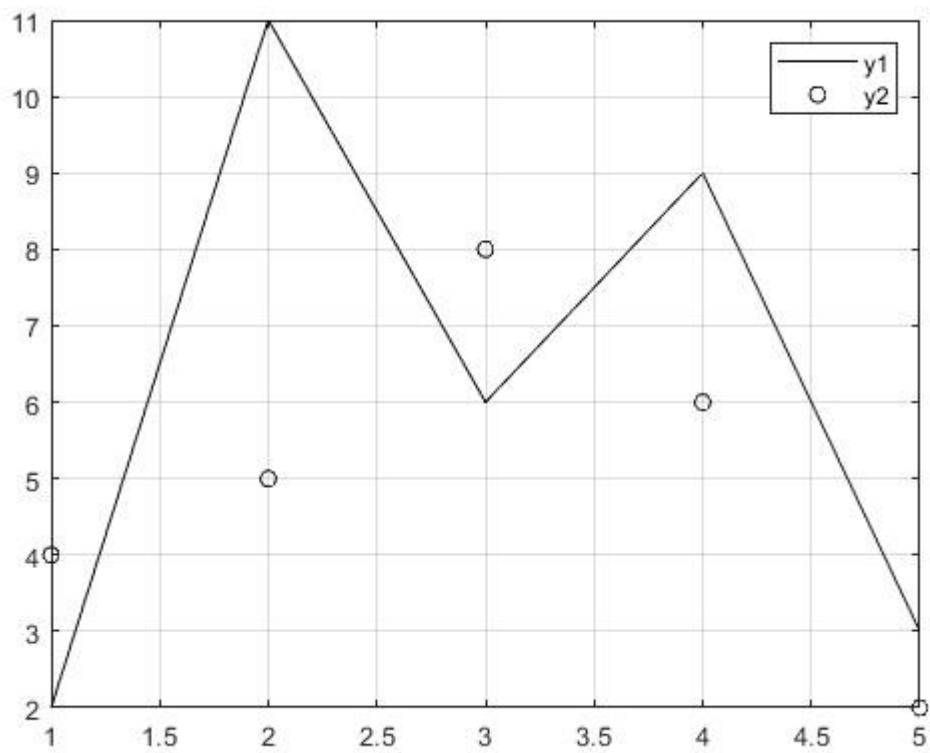
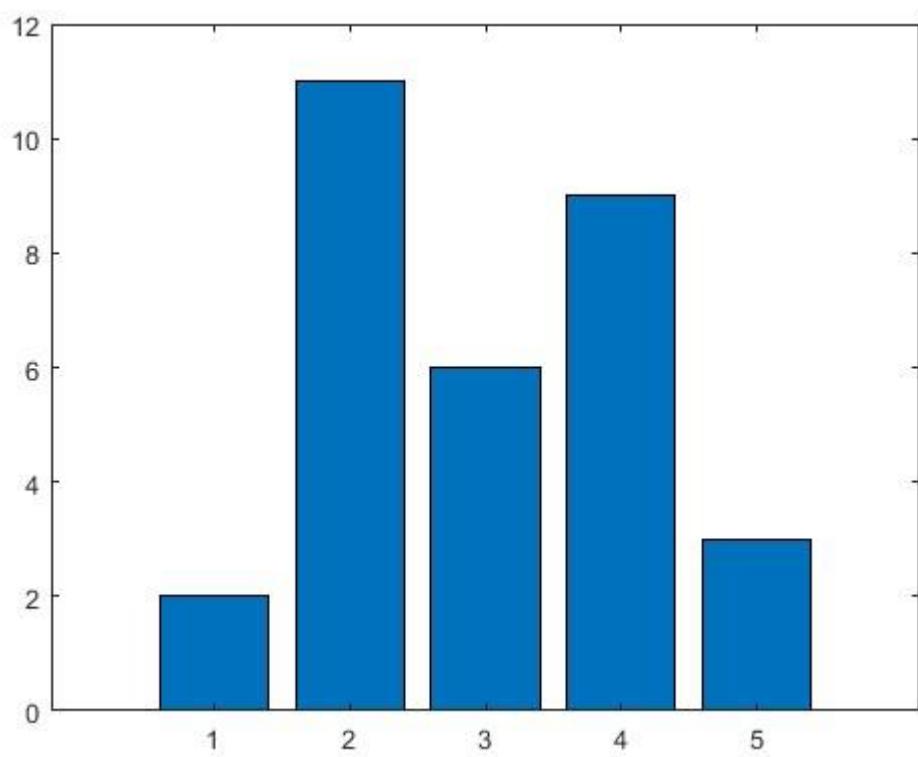
plot2figs.m

```
% This creates 2 different plots, in 2 different
% Figure Windows, to demonstrate some plot features
clf

x = 1:5; % Not necessary
y1 = [2 11 6 9 3] ;
y2 = [4 5 8 6 2] ;
% Put a bar chart in Figure 1

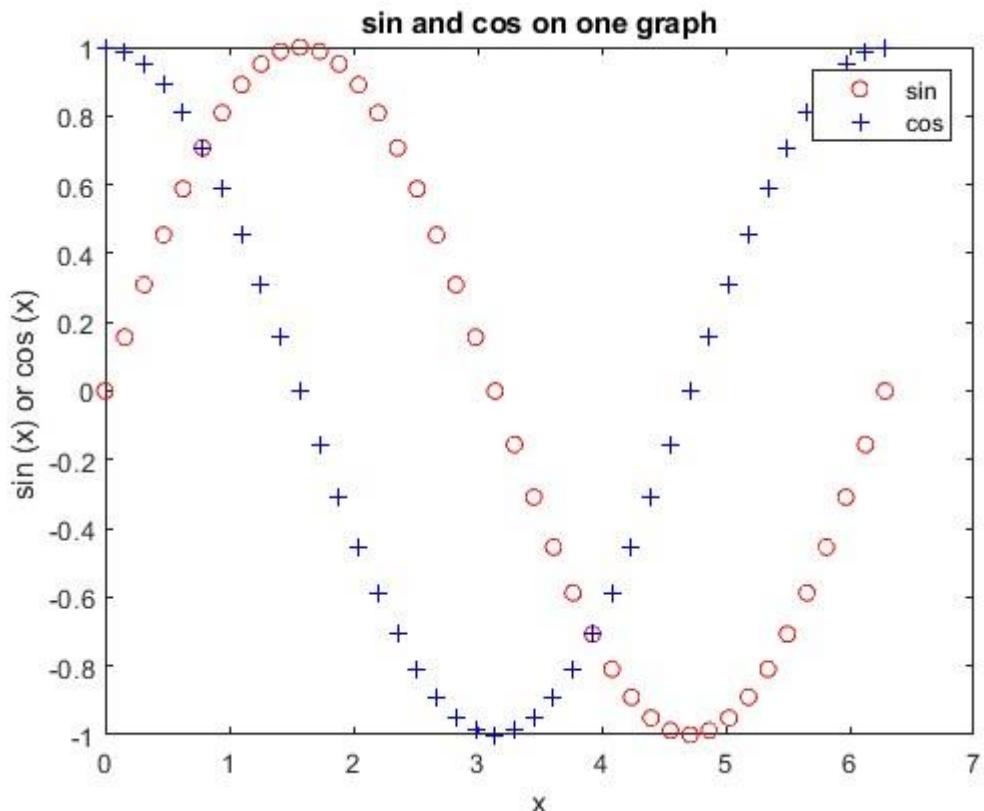
figure (1)
bar (x, y1)
% Put plots using different y values on one plot with a legend
figure (2)

plot (x, y1, 'k')
hold on
plot (x, y2, 'ko' )
grid on
legend ( 'y1' , 'y2' )
```



sinncos.m

```
% takes the character and the number and prints them
a=input('Enter a character--->','s');
n=input('Enter a number--->');
% display the character in a field width of 3.
fprintf('%3c',a);
fprintf('\n')
% display the left-justified number in a field width of 8 with 3 decimal places.
fprintf('-%8.3f\n',n);
```



Practice 3.1

Write a script to calculate the circumference of a circle ($C=2\pi r$). Comment the script.

```
% This script calculates the circumference of a circle
% It prompts the user for the radius

%Prompt the user for the radius and calculate the area based on that radius
fprintf('Note: the units will be inches. \n')
radius = input('Please enter the radius: ');
area = 2 * pi * radius ;

%Print all variables in a sentence format
fprintf('For a circle with a radius of %.2f inches, \n' ,radius)
fprintf('the Circle is %.2f inches \n' , area)
```

```
>> practice3_1
Note: the units will be inches.
Please enter the radius: 5
For a circle with a radius of 5.00 inches,
the Circle is 31.42 inches
```

Practice 3.2

Create a script that would prompt the user for a length, and then 'f' for feet or 'm' for meters, and store both inputs in variables. For example, when executed it would look like this (assuming the user enters 12.3 and then m):

```
Enter the length: 12.3
Is that f(eet) or m(eters)?: m
```

```
% This script records the length and the unit of that length
% It prompts the user for the radius

% Prompts the user for values and stores them
length=input ( 'Enter the value for length: ' );
units=input ( 'Enter the units for length (f or m) : ', 's');
>> practice3_2
Enter the value for length: 12
Enter the units for length (f or m) : m
```

Practice 3.3

Write a script to prompt the user separately for a character and a number, and print the character in a field width of 3 and the number left-justified in a field width of 8 with 3 decimal places. Test this by entering numbers with varying widths.

```
% takes the character and the number and prints them
a=input('Enter a character--->','s');
n=input('Enter a number--->');
% display the character in a field width of 3.
fprintf('%3c',a);
fprintf('\n')
% display the left-justified number in a field width of 8 with 3 decimal places.
fprintf('%-8.3f\n',n);

>> practice3_3
Enter a character--->varun
Enter a number--->123455678.123344556789
v a r u n
123455678.123
```

Practice 3.4

Modify the script *plotonepoint* to prompt the user for the time and temperature, and set the axes based on these values.

```
% This script plots one point (time, temp) from data
% acquired from the user.
% Ask the user for the data to plot and store in the
% variables time and temp
time = input('Enter the time in hours: ');

temp = input('Enter the temperature in degrees C: ');
plot(time,temp,"*")

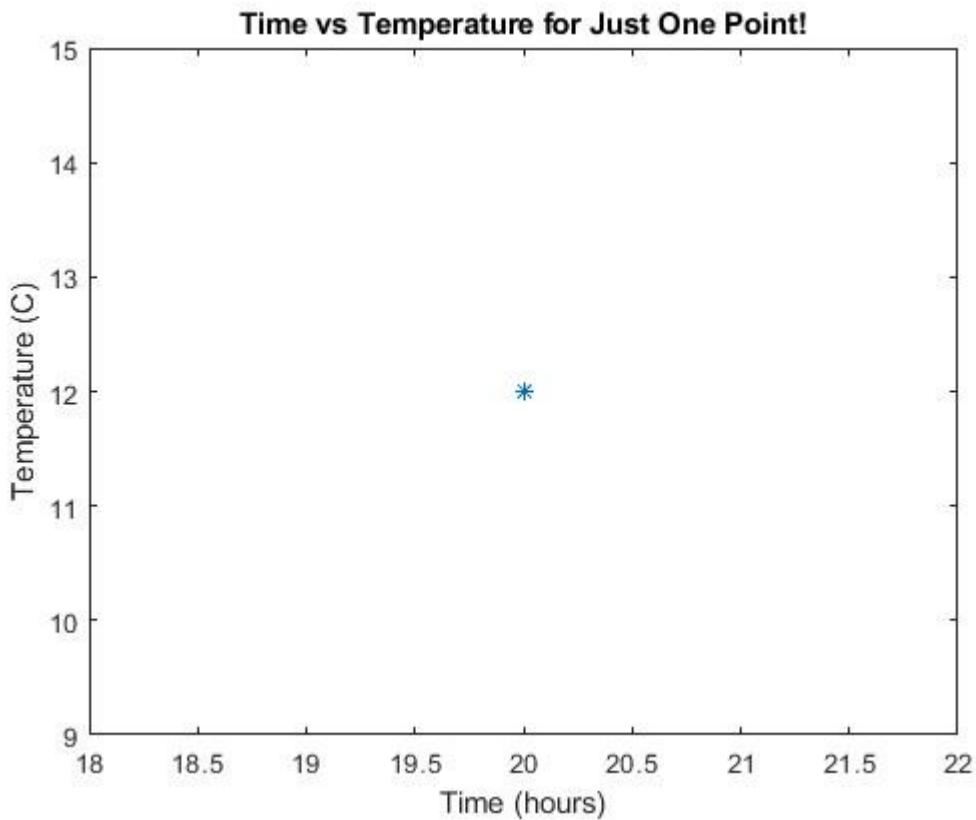
% Adjust the x and y axis limits

axis([time-2 time+2 temp-3 temp+3]);

% Label the axis
xlabel("Time (hours)")

ylabel("Temperature (C)")
% Add a title
title("Time vs Temperature for Just One Point!")
```

```
>> practice3_4
Enter the time in hours: 12
Enter the temperature in degrees C: 50
```

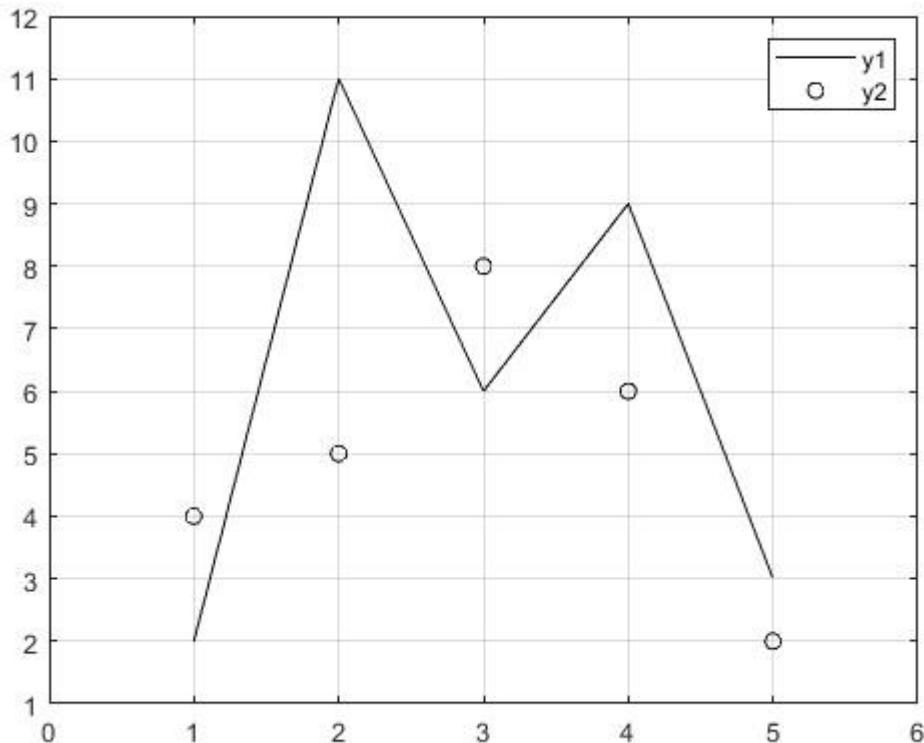


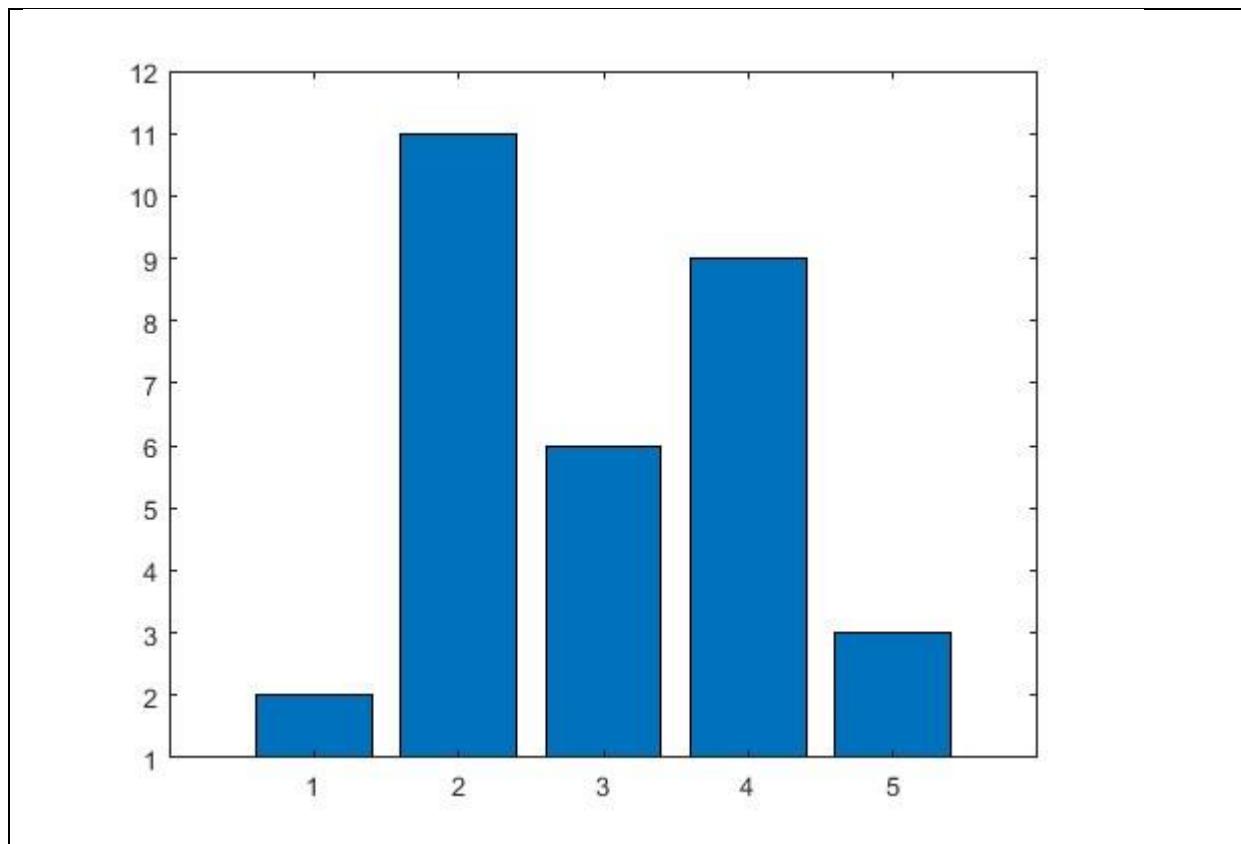
Practice3.5

Modify the `plot2figs` script using the `axis` function so that all points are easily seen.

```
% creates two different plots
x=1:5;
y1=[2 11 6 9 3];

y2=[4 5 8 6 2];
%Put a bar chart in Figure 1
figure(1)
bar(x,y1)
%Change the axis settings
axis([0 6 1 12]);
%Put plots using different y values on one plot with a legend
figure(2)
plot(x,y1, "k")
hold on
plot(x,y2, 'ko')
grid on
legend( 'y1', 'y2')
%Change the axis settings
axis([0 6 1 12]);
```

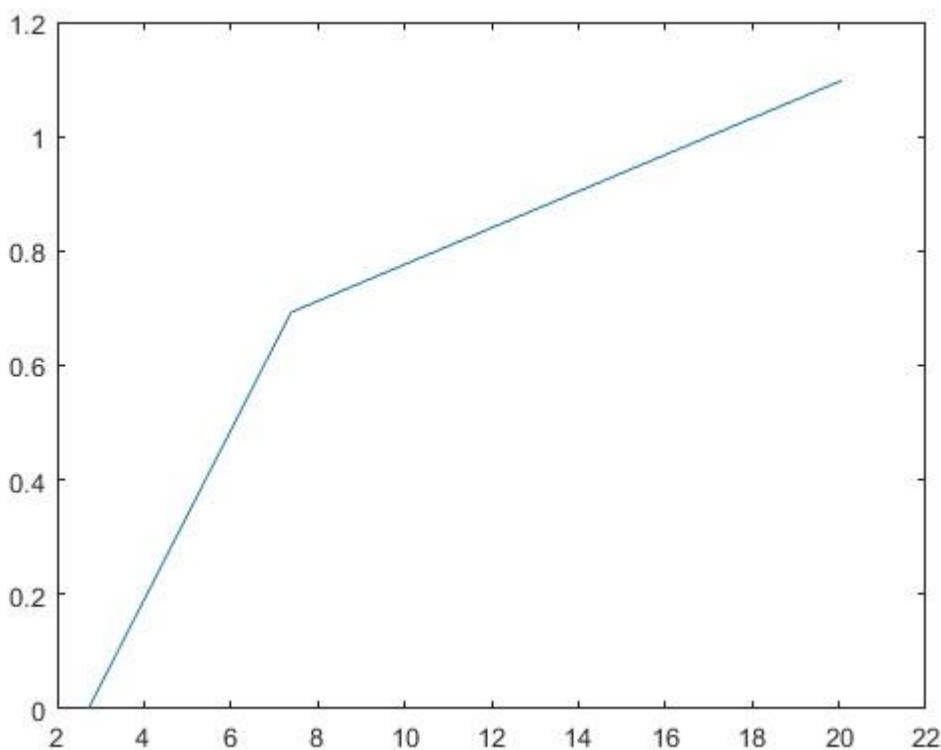




Practice 3.6

Write a script that plots $\exp(x)$ and $\log(x)$ for values of x ranging from 0 to 3.5.

```
x = 0:3.5;
plot(exp(x),log(x))
```



Practice 3.7

Prompt the user for the number of rows and columns of a matrix, create a matrix with that many rows and columns of random integers, and write it to a file.

```
% This script asks for user input for the number of rows
% and columns for a matrix. Then a matrix of that size is
% filled with random numbers and save in ASCII format to a

% file.
% Ask user for input
row = input('Enter the number of matrix rows: ');
col = input('Enter the number of matrix columns: ');

% Create the matrix of random numbers of specified size.
mat = randi(25, row,col)
% Save the matrix as an ASCII file.
save myMatrix.dat mat -ascii
```

```

>> practice3_7
Enter the number of matrix rows: 5
Enter the number of matrix columns: 5

mat =

```

11	1	19	1	18
23	22	10	7	8
20	24	17	2	24
24	17	5	3	1
17	19	18	21	11

Practice 3.8

The sales [in billions] for two separate divisions of the ABC Corporation for each of the four quarters of 2013 are stored in a file called "salesfigs.dat":

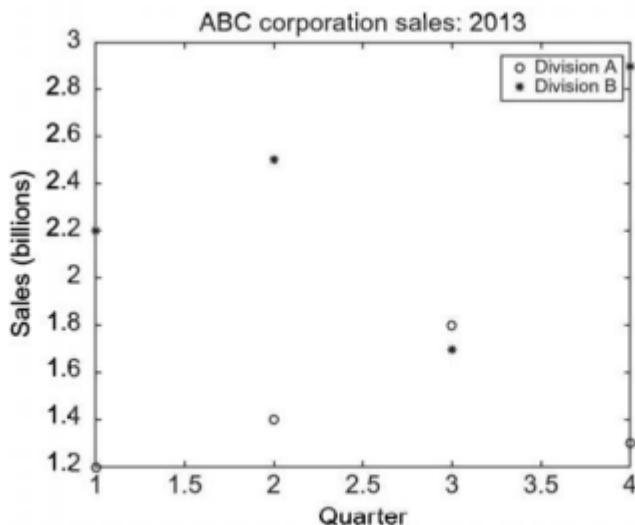
```

1.2 1.4 1.8 1.3
2.2 2.5 1.7 2.9

```

- First, create this file [just type the numbers in the Editor, and Save As "salesfigs.dat"].
- Then, write a script that will

load the data from the file into a matrix
 separate this matrix into 2 vectors.
 create the plot seen in Fig. 3.7 [which uses black circles and stars as the plot symbols].



```

row = [1.2 1.4 1.8 1.3];
col = [2.2 2.5 1.7 2.9];
mat = [row;col]
% Save the matrix as an ASCII file.
save salesfigs.dat mat -ascii
% Load .dat file
load salesfigs.dat
x=salesfigs(1,:)
y=salesfigs(2,:)

```

```

plot(1:numel(x),x,'o')
hold on
plot(1:numel(y),y,'.', 'MarkerSize',20)
ylabel('Sales(billion)')
title('ABC corporation Sales:2013')
legend('Division A','Division B')
hold off

```

```
>> practice3_8
```

```
mat =
```

```
1.2000 1.4000 1.8000 1.3000
2.2000 2.5000 1.7000 2.9000
```

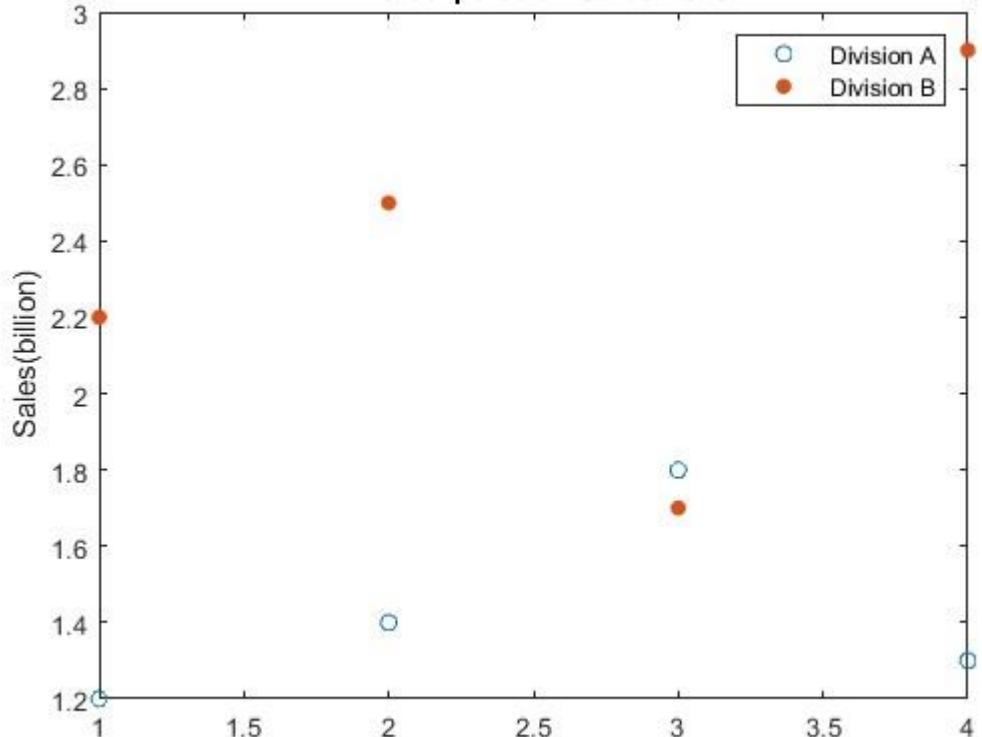
```
x =
```

```
1.2000 1.4000 1.8000 1.3000
```

```
y =
```

```
2.2000 2.5000 1.7000 2.9000
```

ABC corporation Sales:2013



Practice 3.9

Write a script that will prompt the user for the radius and height, call the function *conevol* to calculate the cone volume, and print the result in a nice sentence format. So, the program will consist of a script and the *conevol* function that it calls.

```
function [ V ] = conevol( r,h )
% CONEVOL finds the volume of a cone
% Format of call: conevol(r,h)

% Returns cone volume
V = pi * r^2 * h/3;
end

>> conevol(4,5)

ans =

    83.7758
```

Practice 3.10

For a project, we need some material to form a rectangle. Write a function *calrectarea* that will receive the length and width of a rectangle in inches as input arguments, and will return the area of the rectangle. For example, the function could be called as shown, in which the result is stored in a variable and then the amount of material required is printed, rounded up to the nearest square inch.

```
>> ra = calrectarea(3.1, 4.4)
ra =
    13.6400

>> fprintf('We need %d sq in.\n', ceil(ra))
We need 14 sq in.
```

```
function [ A ] = calrectarea( L,W )
% CALRECTAREA Finds the area of a rectangle rounded up to
% the nearest integer.
% Format of call: calRectArea(L,W)
% Returns rectangle area rounded up to the nearest integer
% Compute the area and round the result up to the nearest integer.
A = ceil( L * W );
end

>> calrectarea(4,4)

ans =

    16
```

Lab 9: If-Else statement

1. Whether a storm is a tropical depression, tropical storm, or hurricane is determined by the average sustained wind speed. In miles per hour, a storm is a tropical depression if the winds are less than 38 mph. It is a tropical storm if the winds are between 39 and 73 mph, and it is a hurricane if the wind speeds are \geq 74 mph. Write a script that will prompt the user for the wind speed of the storm, and will print which type of storm it is.

```
% Prints whether a storm is a tropical depression,  
% tropical storm, or hurricane based on wind speed  
  
wind = input('Enter the wind speed of the storm: ');  
if wind < 38  
    disp('Tropical depression')  
elseif wind >= 38 && wind < 73  
    disp('Tropical storm')  
else  
    disp('Hurricane')  
end
```

Tropical storm

2. In aerodynamics, the Mach number is a critical quantity. It is defined as the ratio of the speed of an object (e.g., an aircraft) to the speed of sound. If the Mach number is less than 1, the flow is subsonic; if the Mach number is equal to 1, the flow is transonic; if the Mach number is greater than 1, the flow is supersonic. Write a script that will prompt the user for the speed of an aircraft and the speed of sound at the aircraft's current altitude and will print whether the condition is subsonic, transonic, or supersonic.

```
% Prints whether the speed of an object is subsonic,  
% transonic, or supersonic based on the Mach number  
  
plane_speed = input('Enter the speed of the aircraft: ');  
sound_speed = input('Enter the speed of sound: ');  
mach = plane_speed/sound_speed;  
  
if mach < 1  
    disp('Subsonic')  
elseif mach == 1  
    disp('Transonic')  
else  
    disp('Supersonic')  
end
```

Supersonic

3. In a script, the user is supposed to enter either a 'y' or 'n' in response to a prompt. The user's input is read into a character variable called "letter". The script will print "OK, continuing" if the user enters either a 'y' or 'Y' or it will print "OK, halting" if the user enters a 'n' or 'N' or "Error" if the user enters anything else. Put this statement in the script first:

```
letter = input('Enter your answer: ', 's');
```

Write the script using a single nested if-else statement (elseif clause is permitted).

```
% Prompts the user for a 'y' or 'n' answer and responds
% accordingly, using an if-else statement
letter = input('Enter your answer: ', 's');
if letter == 'y' || letter == 'Y'
    disp('OK, continuing')
elseif letter == 'n' || letter == 'N'
    disp('OK, halting')
else
    disp('Error')
end
```

OK, continuing

4. Write a function *flipvec* that will receive one input argument. If the input argument is a row vector, the function will reverse the order and return a new row vector. If the input argument is a column vector, the function will reverse the order and return a new column vector. If the input argument is a matrix or a scalar, the function will return the input argument unchanged.

```
% function out = flipvec(vec)
% % Flips it if it's a vector, otherwise
% % returns the input argument unchanged
% % Format of call: flipvec(vec)
% % Returns flipped vector or unchanged
% [r, c] = size(vec);
% if r == 1 && c > 1
%     out = fliplr(vec);
% elseif c == 1 && r > 1
%     out = flipud(vec);
% else
%     out = vec;
% end
% end

% flipvec([1 2 3 4 5])
%
% ans =
```

```
%      5      4      3      2      1
%
% flipvec([1; 2; 3; 4; 5])
%
% ans =
%
%      5
%      4
%      3
%      2
%      1
```

5. Write a function `eqfn` that will calculate $f(x) = x^2 + \frac{1}{x}$ for all elements of x . Since division by 0 is not possible, if any element in x is zero, the function will instead return a flag of -99.

```
% function fofx = eqfn(x)
% if any(any(x==0))
%     fofx = -99;
% else
%     fofx = x.^2 + 1./x;
% end
% end

% eqfn(5)
%
% ans =
%
%      25.2000
```

for-end loop

6. In the Command Window, write a `for` loop that will iterate through the integers from 32 to 255. For each, show the corresponding character from the character encoding. Play with this! Try printing characters beyond the standard ASCII, in small groups. For example, print the characters that correspond to integers from 300 to 340.

```
for i = 32:255
    disp(char(i))
end
```

```
!
"
#
$%
%
```

&
,

(
)

*

+

,

-

.

/

0

1

2

3

4

5

6

7

8

9

:

;

<

=

>

?

@

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

[

\

]

^

-

a

b

c

d

e

f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x
y
z
{
|
}
~

í
¢
£
¤
¥

‐ § ..
© a « ‐
® - o ± 2 3 ‐
μ ¶ . ‐ 1
o » ¼ ½ ¾
‐ Ä Á Â Ã Ä Å
Æ Ç È É Æ Ë Ì Í Î Ï Ð Ñ Ò Ó Ô Õ × Ø Ù Ú Û Ý Þ ß à á â ã æ á

æ
ç
è
é
ê
ë
ì
í
î
ï
ð
ñ
ò
ó
ô
õ
ö
÷
ø
ù
ú
û
ý
þ
ÿ

7. Write a function *sumsteps2* that calculates and returns the sum of 1 to *n* in steps of 2, where *n* is an argument passed to the function. For example, if 11 is passed, it will return $1 + 3 + 5 + 7 + 9 + 11$. Do this using a for loop. Calling the function will look like this:

```
>> sumsteps2(11)
```

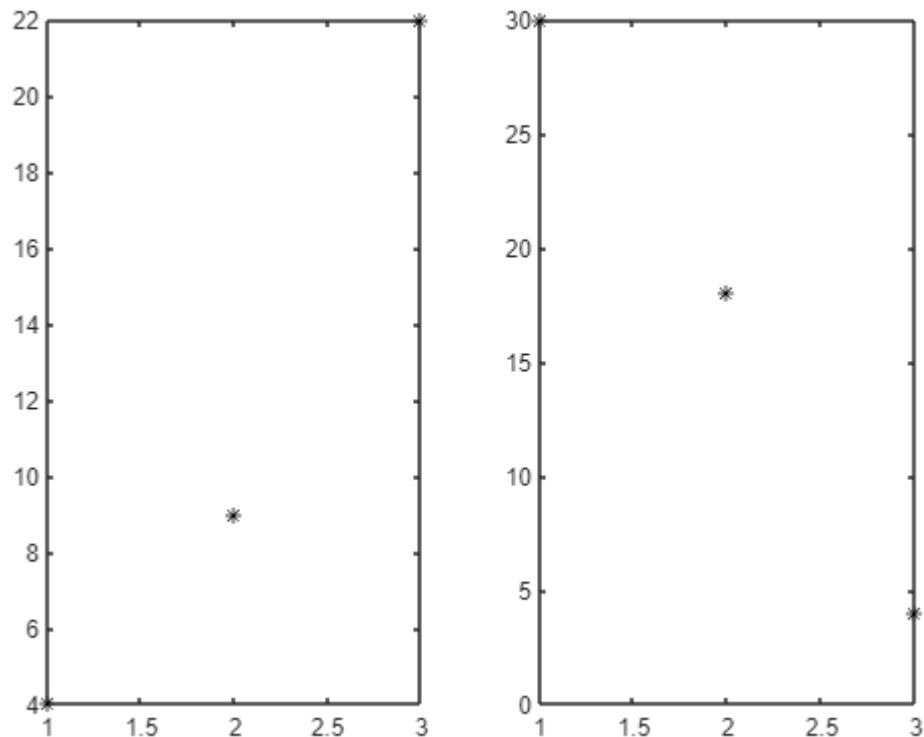
```
ans = 36
```

```
% function outsum = sumsteps2(n)
% sum from 1 to n in steps of 2
% Format of call: sumsteps2(n)
% Returns 1 + 3 + ... + n
%
% outsum = 0;
% for i = 1:2:n
%     outsum = outsum + i;
% end
% end

% sumsteps2(18)
%
% ans =
%
%     81
```

8. Write a script that will load data from a file into a matrix. Create the data file first, and make sure that there is the same number of values on every line in the file so that it can be loaded into a matrix. Using a `for` loop, it will then create a subplot for every row in the matrix, and will plot the numbers from each row element in the Figure Window.

```
row1 = [4 9 22];
row2 = [30 18 4];
mat = [row1;row2];
% Save the matrix as an ASCII file.
save xfile.dat mat -ascii
% load data from a file and plot data
% from each line in a separate plot in a subplot
load xfile.dat
[r, c] = size(xfile);
for i = 1:r
    subplot(1,r,i)
    plot(xfile(i,:), 'k*')
end
```



9. Write a script that will print the following multiplication table:

1

2 4

3 6 9

4 8 12 16

5 10 15 20 25

```
% Prints a multiplication table
rows = 5;
for i = 1:rows
%    for j = 1:i
%        fprintf('%d ', i*j)
%    end
%    fprintf('\n')
    linspace(i,i*i,i)
end
```

```
ans = 1
ans = 1×2
 2     4
ans = 1×3
 3     6     9
ans = 1×4
 4     8    12    16
ans = 1×5
 5    10    15    20    25
```

10. A machine cuts N pieces of a pipe. After each cut, each piece of pipe is weighed and its length is measured; these 2 values are then stored in a file called *pipe.dat* (first the weight and then the length on each line of the file). Ignoring units, the weight is supposed to be between 2.1 and 2.3, inclusive, and the length is supposed to be between 10.3 and 10.4, inclusive. The following is just the beginning of what will be a long script to work with these data. For now, the script will just count how many rejects there are. A reject is any piece of pipe that has an invalid weight and/or length. For a simple example, if N is 3 (meaning three lines in the file) and the file stores:

2.14 10.30

2.32 10.36

2.20 10.35

there is only one reject, the second one, as it weighs too much. The script would print: There were 1 rejects.

```
% Counts pipe rejects. Ignoring units, each pipe should be
% between 2.1 and 2.3 in weight and between 10.3 and 10.4
% in length
row1 = [2.44 10.40];
row2 = [2.23 10.36];
```

```
row3 = [2.20 12.35];
mat = [row1;row2;row3];
% Save the matrix as an ASCII file.
save pipe.dat mat -ascii
% read the pipe data and separate into vectors
load pipe.dat
weights = pipe(:,1);
lengths = pipe(:,2);
N = length(weights);
% the programming method of counting
count = 0;
for i=1:N
    if weights(i) < 2.1 || weights(i) > 2.3 || lengths(i) < 10.3 || lengths(i) > 10.4
        count = count + 1;
    end
end
fprintf('There were %d rejects.\n', count)
```

There were 2 rejects.

Roll No.: A016	Name: Varun Khadayate
Class: B. Tech CsBs	Batch: 1
Date of Experiment: 25-09-2022	Subject: ITWS

Lab 10: M Script Debugging

Code with error

```
function [vals, locs]=local_max(v)
n=length(v);
vals=[];
locs=[];
for m=1:n
    if v(m)-v(m-1)>0 && v(m)-v(m+1)>0
        vals = [vals;v(m)];
        locs = [locs;m];
    end
end
figure; plot(v);
hold on;
plot(locs,vals,'ro');
```

Running the file in the command window at first

```
>> a = rand(10,1);
>> [V,L] = local_max(a);
Array indices must be positive integers or logical
values.

Error in local_max (line 7)
    if v(m)-v(m-1)>0 && v(m)-v(m+1)>0
```

Error is received so will use step debugging

Setting Step debug in line 2 from where the file is running

2 | n=length(v);

Running the step debug in command window

```
>> a = rand(10,1);
>> [V,L] = local_max(a);
2 n=length(v);
K>>
```

Click on the step button to go to next step



You will find there is an error in line 7 as its shown in command window

```
>> a = rand(10,1);
>> [V,L] = local_max(a);
2 n=length(v);
Array indices must be positive integers or logical
values.

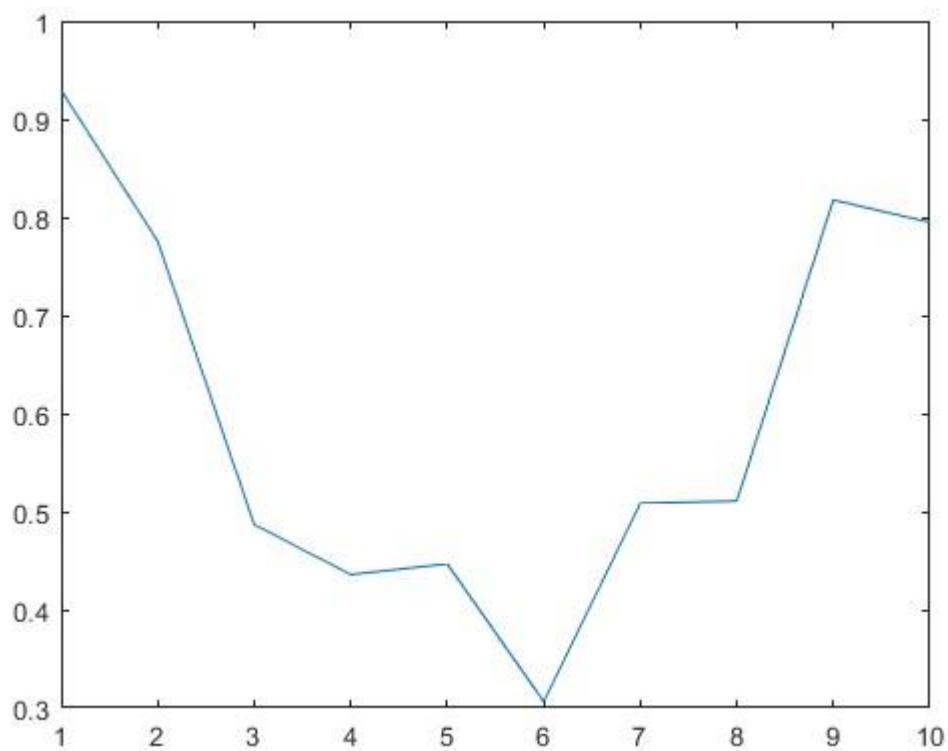
Error in local_max (line 7)
    if v(m)-v(m-1)>0 && v(m)-v(m+1)>0
```

Changing value from for $m=1:n$ to $m=2:n$

Once again setting the breakpoint in line 2 and running it

2 || **n=length(v);**

After doing it continue the above step by clicking on step button again and again till the graph is received like this



Roll. No. A016	Name: Varun Khadayate
Class B.Tech CsBs	Batch: 1
Date of Experiment: 19-07-2021	Subject: IT/WS

1. Launch MATLAB, create list of following variables in command window:

a) $m=10, n=25, p=43$

Ans:

```
>> m = 10
m =
10
>> n = 25
n =
25
>> p = 43
p =
43
```

b) $A=m^2, B=n^3, C=(A+B)*p$

Ans:

```
>> A = m^2
A =
100
>> B = n^3
B =
15625
>> C = (A + B)*p
C =
676175
```

c) $t=0.1, f=0.5, a=5, x=a*\sin(2\pi ft)$

Ans:

```
>> t = 0.1
t =
0.1000
>> f = 0.5
f =
0.5000
>> a = 5
a =
5
>> x = a*pi*f*t
x =
1.5451
```

d) $y=mx+C$

Ans:

```
>> y = m*x + C
y =
6.7619e+05
```

e) $k=(t^2+1)(t^2-1)$

Ans:

```
>> k = (t^2 + 1)*(t^2 - 1)
k =
-0.9999
```

2.

(a) From Question1 make a new variable 'v', overwriting part (c), i.e.,

$$x=a*\sin(2\pi ft)$$

by adding $\cosh(t)$

Ans:

```
>> v = x + cosh(t)  
  
v =  
  
2.5501
```

(b) Create variable 'r', store value to it to find the area of circle :

$$A=\pi r^2$$

where 'r' is the radius of circle. Further, using the built in function **namelengthmax** , find the maximum number of character in "A".

[Hint: store value of 'r' as 10]

Ans:

```
>> r = 10  
  
r =  
  
10  
  
>> A = pi*r^2  
  
A =  
  
314.1593  
  
>> A = namelengthmax  
  
A =  
  
63
```

3. Explore the solve command using MATLAB help and find the solution for the problem given :
 $X + 1 = 2$, find X

Ans:

```
>> syms X  
>> solve(X + 1 == 2, X)  
  
ans =  
  
1
```

4. Complete the table using **help** command:

Ans:

Command name	Purpose								
whos	<p>>> help whos</p> <p>whos List current variables, long form. whos is a long form of WHO. It lists all the variables in the current workspace, together with information about their size, bytes, class, etc.</p> <p>In a nested function, variables are grouped into those in the nested function and those in each of the containing functions, each group separated by a line of dashes.</p> <p>whos GLOBAL lists the variables in the global workspace.</p> <p>whos -FILE FILENAME lists the variables in the specified .MAT file.</p> <p>whos ... VAR1 VAR2 restricts the display to the variables specified.</p> <p>The wildcard character '*' can be used to display variables that match a pattern. For instance, whos A* finds all variables in the current workspace that start with A.</p> <p>whos -REGEXP PAT1 PAT2 can be used to display all variables matching the specified patterns using regular expressions. For more information on using regular expressions, type "doc regexp" at the command prompt.</p> <p>Use the functional form of whos, such as whos('-file',FILE,V1,V2), when the filename or variable names are stored as a character vector or string scalar.</p> <p>S = whos(...) returns a structure with the fields:</p> <table> <tr> <td>name</td> <td>-- variable name</td> </tr> <tr> <td>size</td> <td>-- variable size</td> </tr> <tr> <td>bytes</td> <td>-- number of bytes allocated for the array</td> </tr> <tr> <td>class</td> <td>-- class of variable</td> </tr> </table>	name	-- variable name	size	-- variable size	bytes	-- number of bytes allocated for the array	class	-- class of variable
name	-- variable name								
size	-- variable size								
bytes	-- number of bytes allocated for the array								
class	-- class of variable								

	<p>global -- logical indicating whether variable is global</p> <p>sparse -- logical indicating whether value is sparse</p> <p>complex -- logical indicating whether value is complex</p> <p>nesting -- struct with the following two fields:</p> <ul style="list-style-type: none"> function -- name of function where variable is defined level -- nesting level of the function persistent -- logical indicating whether variable is persistent <p>You must use the functional form of whos when there is an output argument.</p> <p>Examples for pattern matching:</p> <ul style="list-style-type: none"> whos a* % Show variable names starting with "a" whos -regexp ^b\d{3}\\$ % Show variable names starting with "b" % and followed by 3 digits whos -file fname -regexp \d % Show variable names containing any % digits that exist in MAT-file fname <p>See also who, clear, clearvars, save, load.</p> <p>Documentation for whos</p>
clear	<p>>> help clear</p> <p>clear Clear variables and functions from memory.</p> <p>clear removes all variables from the workspace.</p> <p>clear VARIABLES does the same thing.</p> <p>clear GLOBAL removes all global variables.</p> <p>clear FUNCTIONS removes all compiled MATLAB and MEX-functions.</p> <p>Calling clear FUNCTIONS decreases code performance and is usually unnecessary.</p> <p>For more information, see the clear Reference page.</p> <p>clear ALL removes all variables, globals, functions and MEX links.</p> <p>clear ALL at the command prompt also clears the base import list.</p> <p>Calling clear ALL decreases code performance and is usually unnecessary.</p>

For more information, see the `clear` Reference page.

`clear IMPORT` clears the base import list. It can only be issued at the command prompt. It cannot be used in a function or a script.

`clear CLASSES` is the same as `clear ALL` except that class definitions are also cleared. If any objects exist outside the workspace (say in `userdata` or persistent in a locked program file) a warning will be issued and the class definition will not be cleared.

Calling `clear CLASSES` decreases code performance and is usually unnecessary.

If you modify a class definition, MATLAB automatically updates it.

For more information, see the `clear` Reference page.

`clear JAVA` is the same as `clear ALL` except that java classes on the dynamic java path (defined using `JAVACLASSPATH`) are also cleared.

`clear VAR1 VAR2 ...` clears the variables specified. The wildcard character '*' can be used to clear variables that match a pattern. For instance, `clear X*` clears all the variables in the current workspace that start with X.

`clear -REGEXP PAT1 PAT2` can be used to match all patterns using regular expressions. This option only clears variables. For more information on using regular expressions, type "doc regexp" at the command prompt.

If X is global, `clear X` removes X from the current workspace, but leaves it accessible to any functions declaring it global. `clear GLOBAL -REGEXP PAT` removes global variables that match regular expression patterns.

	<p>Note that to clear specific global variables, the GLOBAL option must come first. Otherwise, all global variables will be cleared.</p> <p>clear FUN clears the function specified. If FUN has been locked by MLOCK it will remain in memory. If FUN is a script or function that is currently executing, then it is not cleared. Use a partial path (see PARTIALPATH) to distinguish between different overloaded versions of FUN. For instance, 'clear inline/display' clears only the INLINE method for DISPLAY, leaving any other implementations in memory.</p> <p>Examples for pattern matching:</p> <pre>clear a* % Clear variables starting with "a" clear -regexp ^b\d{3}\\$ % Clear variables starting with "b" and % followed by 3 digits clear -regexp \d % Clear variables containing any digits</pre> <p>See also clearvars, who, whos, mlock, munlock, persistent, import.</p> <p>Documentation for clear Other functions named clear</p>
pwd	<pre>>> help pwd pwd Show (print) current working directory. pwd displays the current working directory. S = pwd returns the current directory in the string S. See also cd.</pre> <p>Documentation for pwd</p>
diary	<pre>>> help diary diary Save text of MATLAB session. diary FILENAME causes a copy of all subsequent command window input and most of the resulting command window output to be appended to the named file. If no file is specified, the file 'diary' is used.</pre>

diary OFF suspends it.
diary ON turns it back on.
diary, by itself, toggles the diary state.

Use the functional form of diary, such as
diary('file'),
when the file name is stored in a string.

See also save.

Documentation for diary

5. Given $\theta = 145$ degrees. A vector can be represented by its rectangular coordinates x and y or by its polar coordinates r and θ . Theta is measured in radians. The relationship between them is given by the equations:

$$\begin{aligned}x &= r * \cos(\theta) \\y &= r * \sin(\theta)\end{aligned}$$

Assign values for the polar coordinates to variables r and θ . Then, using these values, assign the corresponding rectangular coordinates to variables x and y .

Ans:

```
>> r = 10

r =

10

>> theta = 145

theta =

145

>> x = r * cos(theta)

x =

8.8386

>> y = r * sin(theta)

y =

4.6775
```

6. The combined resistance R_r of three resistors R_1 , R_2 , and R_3 in parallel is given by

$$R_r = \frac{1}{\frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3}}$$

Create variables for the three resistors and store values in each, and then calculate the combined resistance. (Hint: consider $R_1=50\Omega$, $R_2=25\Omega$ and $R_3=60\Omega$)

Ans:

```
>> R1 = 50
R1 =
50
>> R2 = 25
R2 =
25
>> R3 = 60
R3 =
60
>> R = (1/((1/R1)+(1/R2)+(1/R3)))
R =
13.0435
```

Roll. No. A016	Name: Varun Khadayate
Class B.Tech CsBs	Batch: 1
Date of Experiment: 25-07-2021	Subject: IT/WS

1. The sum of a geometric series

$$1+r+r^2+r^3+\dots+r^n = (1-r^N)/(1-r);$$

N=number of terms in a series.

Accept the value of r and n as input from keyboard. Verify the above equation

```
r = 2
r =
2
>> N = 20
N =
20
>> x = (1-r^N)/(1-r)
x =
1048575
>> y = sum(r.^0:N-1)
y =
1048575
>> logical(x == y)
ans =
logical
1
```

2. Accept a square matrix A of any size from keyboard.

```
>> A = [9,9,3;7,2,7;2,8,5]
```

```
A =
```

9	9	3
7	2	7
2	8	5

Find:

a. Size of A matrix

```
>> size(A)
```

```
ans =
```

3	3
---	---

b. Determinant of A matrix

```
>> det(A)
```

```
ans =
```

```
-447.0000
```

c. Display whether matrix A is singular or not

```
>> cond(A)
```

```
ans =
```

```
4.0913
```

d. Transpose of A matrix

```
>> B = transpose(A)
```

```
B =
```

9	7	2
9	2	8
3	7	5

e. Perform $A+A'$, $A-A'$

```
>> C = A + B
```

C =

```
18 16 5
16 4 15
5 15 10
```

```
>> D = A - B
```

D =

```
0 2 1
-2 0 -1
-1 1 0
```

f. Find inverse of A matrix

```
>> inv(A)
```

ans =

```
0.1029 0.0470 -0.1275
0.0470 -0.0872 0.0940
-0.1163 0.1208 0.1007
```

g. Perform A^*A' and $A.^*A'$

```
>> A*B
```

ans =

```
171 102 105
102 102 65
105 65 93
```

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

ans =

```
81 63 6
63 4 56
6 56 25
```

h. Find square of A matrix

```
>> F = A*A
```

F =

```
150 123 105
91 123 70
84 74 87
```

i. Find rank of A matrix

```
>> rank (A)

ans =
3
```

j. Find eigenvalues and eigenvectors of A matrix

```
>> eig(A)

ans =
17.5118
4.3526
-5.8644

>> [V,D] = eig(A)

V =
-0.7183 -0.6835 0.3698
-0.5284 0.1125 -0.7794
-0.4527 0.7213 0.5058

D =
17.5118 0 0
0 4.3526 0
0 0 -5.8644
```

3. Create a vector and a matrix with the following commands: $v=0:0.2:12$ and $M=[\sin(v); \cos(v)]$. Find the sizes of v and M and extract the first 10 elements of each row of the matrix and display them as column vectors.

```
>> v=0:0.2:12

v =

Columns 1 through 12

    0    0.2000    0.4000    0.6000    0.8000    1.0000    1.2000    1.4000    1.6000    1.8000
    2.0000    2.2000

Columns 13 through 24

    2.4000    2.6000    2.8000    3.0000    3.2000    3.4000    3.6000    3.8000    4.0000
    4.2000    4.4000    4.6000

Columns 25 through 36

    4.8000    5.0000    5.2000    5.4000    5.6000    5.8000    6.0000    6.2000    6.4000
    6.6000    6.8000    7.0000

Columns 37 through 48

    7.2000    7.4000    7.6000    7.8000    8.0000    8.2000    8.4000    8.6000    8.8000
    9.0000    9.2000    9.4000

Columns 49 through 60

    9.6000    9.8000   10.0000   10.2000   10.4000   10.6000   10.8000   11.0000   11.2000
   11.4000   11.6000   11.8000

Column 61

12.0000

>> M=[sin(v); cos(v)]

M =

Columns 1 through 12

    0    0.1987    0.3894    0.5646    0.7174    0.8415    0.9320    0.9854    0.9996    0.9738
    0.9093    0.8085
    1.0000    0.9801    0.9211    0.8253    0.6967    0.5403    0.3624    0.1700   -0.0292   -
    0.2272   -0.4161   -0.5885

Columns 13 through 24
```

```

0.6755 0.5155 0.3350 0.1411 -0.0584 -0.2555 -0.4425 -0.6119 -0.7568 -
0.8716 -0.9516 -0.9937
-0.7374 -0.8569 -0.9422 -0.9900 -0.9983 -0.9668 -0.8968 -0.7910 -0.6536 -
0.4903 -0.3073 -0.1122

```

Columns 25 through 36

```

-0.9962 -0.9589 -0.8835 -0.7728 -0.6313 -0.4646 -0.2794 -0.0831 0.1165
0.3115 0.4941 0.6570
0.0875 0.2837 0.4685 0.6347 0.7756 0.8855 0.9602 0.9965 0.9932
0.9502 0.8694 0.7539

```

Columns 37 through 48

```

0.7937 0.8987 0.9679 0.9985 0.9894 0.9407 0.8546 0.7344 0.5849
0.4121 0.2229 0.0248
0.6084 0.4385 0.2513 0.0540 -0.1455 -0.3392 -0.5193 -0.6787 -0.8111 -
0.9111 -0.9748 -0.9997

```

Columns 49 through 60

```

-0.1743 -0.3665 -0.5440 -0.6999 -0.8278 -0.9228 -0.9809 -1.0000 -0.9792 -
0.9193 -0.8228 -0.6935
-0.9847 -0.9304 -0.8391 -0.7143 -0.5610 -0.3853 -0.1943 0.0044 0.2030
0.3935 0.5683 0.7204

```

Column 61

```

-0.5366
0.8439

```

>> size(v)

ans =

```

1 61

```

>> size(M)

ans =

```

2 61

```

>> M(:,1:10)

ans =

```

0 0.1987 0.3894 0.5646 0.7174 0.8415 0.9320 0.9854 0.9996 0.9738
1.0000 0.9801 0.9211 0.8253 0.6967 0.5403 0.3624 0.1700 -0.0292 -
0.2272

```

4. The polar equation of a circle is given by $x=r \cos\theta$, $y=r \sin\theta$. Take $\theta= 0$ to 2π with step size of $\pi/16$ and plot the circle on x-y axis for given value of radius r. Give labels to axis and title to the figure. Make use of new figure and redraw the circle with distinct points shown by 'o' rather than a continuous plot. Now combine the two plots in new figure to show the line through the data points as well as the distinct data points.

```

>> theta = 0:pi/16:2*pi

theta =

Columns 1 through 12

0 0.1963 0.3927 0.5890 0.7854 0.9817 1.1781 1.3744 1.5708
1.7671 1.9635 2.1598

Columns 13 through 24

2.3562 2.5525 2.7489 2.9452 3.1416 3.3379 3.5343 3.7306 3.9270
4.1233 4.3197 4.5160

Columns 25 through 33

4.7124 4.9087 5.1051 5.3014 5.4978 5.6941 5.8905 6.0868 6.2832

>> x = r*cos(theta)

x =

Columns 1 through 12

2.0000 1.9616 1.8478 1.6629 1.4142 1.1111 0.7654 0.3902 0.0000 -
0.3902 -0.7654 -1.1111

Columns 13 through 24

-1.4142 -1.6629 -1.8478 -1.9616 -2.0000 -1.9616 -1.8478 -1.6629 -1.4142 -
1.1111 -0.7654 -0.3902

Columns 25 through 33

-0.0000 0.3902 0.7654 1.1111 1.4142 1.6629 1.8478 1.9616 2.0000

>> y = r*sin(theta)

y =

Columns 1 through 12

0 0.3902 0.7654 1.1111 1.4142 1.6629 1.8478 1.9616 2.0000
1.9616 1.8478 1.6629

```

Columns 13 through 24

```
1.4142 1.1111 0.7654 0.3902 0.0000 -0.3902 -0.7654 -1.1111 -1.4142 -  
1.6629 -1.8478 -1.9616
```

Columns 25 through 33

```
-2.0000 -1.9616 -1.8478 -1.6629 -1.4142 -1.1111 -0.7654 -0.3902 -0.0000
```

```
>> r = 10
```

```
r =
```

```
10
```

```
>> x = r*cos(theta)
```

```
x =
```

Columns 1 through 12

```
10.0000 9.8079 9.2388 8.3147 7.0711 5.5557 3.8268 1.9509 0.0000 -  
1.9509 -3.8268 -5.5557
```

Columns 13 through 24

```
-7.0711 -8.3147 -9.2388 -9.8079 -10.0000 -9.8079 -9.2388 -8.3147 -7.0711 -  
5.5557 -3.8268 -1.9509
```

Columns 25 through 33

```
-0.0000 1.9509 3.8268 5.5557 7.0711 8.3147 9.2388 9.8079 10.0000
```

```
>> y = r*sin(theta)
```

```
y =
```

Columns 1 through 12

```
0 1.9509 3.8268 5.5557 7.0711 8.3147 9.2388 9.8079 10.0000  
9.8079 9.2388 8.3147
```

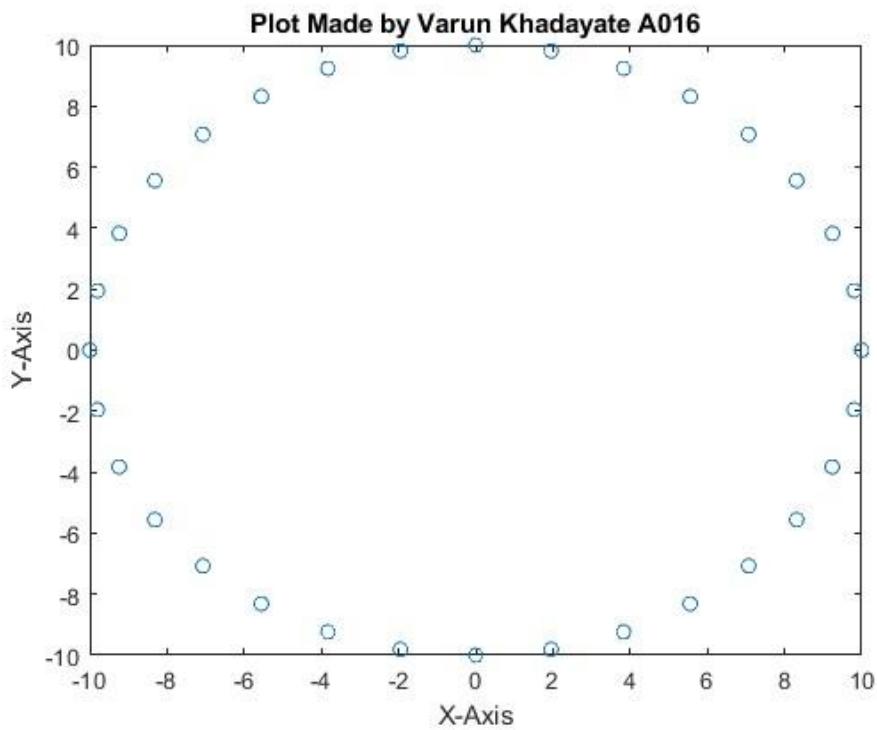
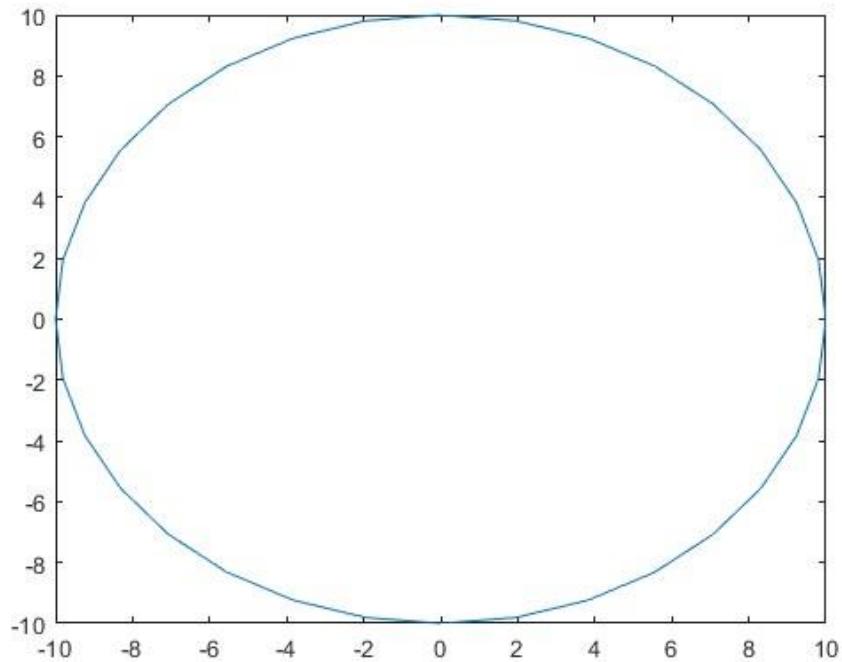
Columns 13 through 24

```
7.0711 5.5557 3.8268 1.9509 0.0000 -1.9509 -3.8268 -5.5557 -7.0711 -  
8.3147 -9.2388 -9.8079
```

Columns 25 through 33

```
-10.0000 -9.8079 -9.2388 -8.3147 -7.0711 -5.5557 -3.8268 -1.9509 -0.0000
```

```
>> plot(x,y)
>> plot(x,y)
>> plot(x,y,'O')
>> plot(x,y,'O')
>> xlabel('X-Axis')
>> ylabel('Y-Axis')
>> title('Plot Made by Varun Khadayate A016')
```



Roll. No. A016	Name: Varun Khadayate
Class B.Tech CsBs	Batch: 1
Date of Experiment: 29-07-2021	Subject: IT/WS

1. Create a variable `myage` and store your age in it. Subtract 2 from the value of the variable. Add 1 to the value of the variable. Observe the Workspace Window and Command History Window as you do this.

Command Window

```
>> myage = 21;
```

Workspace



Command Window

```
>> myage = myage - 2;
```

Workspace



Command Window

```
>> myage = myage + 1;
```

Workspace



2. Explain the difference between these two statements:

```
result = 9*2
```

```
result = 9*2;
```

Both will store 18 in the variable `result`. In the first, MATLAB will display this in the Command Window; in the second, it will not.

`result = 9*2`

Result shown in Command Window and in Worspace

```
>> result = 9*2

result =

18
```

```
result = 9*2;
```

Result shown in Workspace and not in Command Window

```
result 18
```

3. Use the built-in function `namelengthmax` to find out the maximum number of characters that you can have in an identifier name under your version of MATLAB.

```
>> namelengthmax
```

```
ans =
```

```
63
```

4. Create two variables to store a weight in pounds and ounces. Use `who` and `whos` to see the variables. Use `class` to see the types of the variables. Clear one of them and then use `who` and `whos` again.

```
>> pounds = 4;  
>> ounces = 3.3;  
>> who
```

Your variables are:

```
ans  myage  ounces  pounds  result
```

```
>> whos  
Name      Size      Bytes Class      Attributes  
ans      1x1      8 double  
myage    1x1      8 double  
ounces   1x1      8 double  
pounds   1x1      8 double  
result   1x1      8 double
```

```
>> clear pounds
```

```
>> who
```

Your variables are:

```
ans  myage  ounces  result
```

```
>> whos  
Name      Size      Bytes Class      Attributes  
ans      1x1      8 double  
myage    1x1      8 double  
ounces   1x1      8 double  
result   1x1      8 double
```

5. Explore the format command in more detail. Use help format to find options. Experiment with format bank to display dollar values.

```
>> format +
>> 1234

ans =

+

>> -1234

ans =

-

>> format bank
>> 33.4

ans =

33.40

>> 69.435

ans =

69.44
```

6. Find a format option that would result in the following output format:

```
>> 5/16 + 2/7
ans =
67/112
```

```
>> format rat
>> 5/16 + 2/7

ans =

67/112
```

7. Think about what the results would be for the following expressions, and then type them in to verify your answers.

```
25/5 * 5
4 + 3 ^ 2
(4 + 3) ^ 2
3 \ 12 + 5
4 - 2 * 3
```

```
>> 25/5*5
```

```
ans =  
25  
>> 4+3^2  
ans =  
13  
>> (4+3)^2  
ans =  
49  
>> 3/12+5  
ans =  
21/4  
>> 4-2*3  
ans =  
-2
```

8. Create a variable pounds to store a weight in pounds. Convert this to kilograms and assign the result to a variable kilos. The conversion factor is
1 kilogram = 2.2 lb.

```
>> pounds = 69  
pounds =  
69  
>> kilos = pounds / 2.2  
kilos =  
31.3636
```

9. Create a variable ftemp to store a temperature in degrees Fahrenheit (F). Convert this to degrees Celsius (C) and store the result in a variable ctemp. The conversion factor is

$$C = (F - 32) * 5/9.$$

```
>> f = 75
f =
75
>> c = (f - 32) * 5/9
c =
23.8889
```

10. The following assignment statements either contain at least one error, or could be improved in some way. Assume that radius is a variable that has been initialized. First, identify the problem, and then fix and/or improve them:

33 = number
my variable = 11.11;
area = 3.14 * radius^2;
x = 2 * 3.14 * radius;

33 = number

The variable is always on the left **number = 33**

my variable = 11.11;

Spaces are not allowed in variable names **my_variable = 11.11;**

area = 3.14 * radius^2;

Using pi is more accurate than 3.14 **area = pi * radius^2;**

x = 2 * 3.14 * radius;

x is not a descriptive variable name and Using pi is more accurate than 3.14

circumference = 2 * pi * radius;

11. Experiment with the functional form of some operators such as plus, minus, and times.

```
>> plus(6,9)
ans =
15
>> plus(6,-9)
ans =
```

```
-3

>> minus(1,7)

ans =

-6

>> minus(4,-6)

ans =

10

>> times(4,6)

ans =

24

>> times(6,-7)

ans =

-42
```

12. Generate a random

a. real number in the range (0, 20)

```
>> rand * 20

ans =

16.2945
```

b. real number in the range (20, 50)

```
>> rand*(50-20)+20

ans =

47.1738
```

c. integer in the inclusive range from 1 to 10

```
>> randi(10)

ans =

2
```

d. integer in the inclusive range from 0 to 10

```
>> randi([0, 10])

ans =

10
```

e. integer in the inclusive range from 50 to 100

```
>> randi([50, 100])

ans =

82
```

13. Get into a new Command Window, and type rand to get a random real number. Make a note of the number. Then, exit MATLAB and repeat this, again making a note of the random number; it should be the same as before. Finally, exit MATLAB and again get into a new Command Window. This time, change the seed before generating a random number; it should be different.

```
>> rand

ans =

0.0975

>> rng('shuffle')
>> rand

ans =

0.3191
```

14. What is the difference between x and 'x'?

In an expression, the first would be interpreted as the name of a variable, whereas 'x' is the character x.

15. Explain the difference between constants and variables.

Constants store values that are known and do not change. Variables are used when the value will change, or when the value is not known to begin with (e.g., the user will provide the value).

16. What would be the result of the following expressions?

```
'b' >= 'c' - 1
3 == 2 + 1
(3 == 2) + 1
xor(5 < 6, 8 > 4)
10 > 5 > 2
0 <= result <= 10
```

```
>> 'b' >= 'c' - 1
```

```
'b' >= 'c' - 1
```

```
↑
```

Error: Invalid text character. Check for unsupported symbol, invisible character, or pasting of non-ASCII characters.

```
>> 3 == 2 + 1
```

```
ans =
```

```
logical
```

```
1
```

```
>> xor(5 < 6, 8 > 4)
```

```
ans =
```

```
logical
```

```
0
```

```
>> 10 > 5 > 2
```

```
ans =
```

```
logical
```

```
0
```

```
>> result = 3^2 - 20;
```

```
>> result = 3^2 - 20
```

```
result =
```

```
-11  
>> 0 <= result <= 10  
ans =  
logical  
1
```

17. Create two variables x and y and store numbers in them. Write an expression that would be true if the value of x is greater than five or if the value of y is less than ten, but not if both of those are true.

```
>> x = 6  
x =  
6  
>> y = 9  
y =  
9  
>> xor(x > 5, y < 10)  
ans =  
logical  
0
```

18. Use the equality operator to verify that $3*10^5$ is equal to $3e5$.

```
>> 3*10^5 == 3e5  
ans =  
logical  
1
```

19. In the ASCII character encoding, the letters of the alphabet are in order: 'a' comes before 'b' and also 'A' comes before 'B'. However, which comes first - lower or uppercase letters?

```
>> int32('a')
```

```
ans =
```

```
int32
```

```
97
```

```
>> int32('A')
```

```
ans =
```

```
int32
```

```
65
```

20. Are there equivalents to intmin and intmax for real number types? Use help to find out.

```
>> help intmax
```

intmax Largest positive integer value.

X = intmax is the largest positive value representable in an int32.

Any value that is larger than intmax will saturate to intmax when cast to int32.

intmax('int32') is the same as intmax with no arguments.

intmax(CLASSNAME) is the largest positive value in the integer class CLASSNAME. Valid values of CLASSNAME are 'int8', 'uint8', 'int16', 'uint16', 'int32', 'uint32', 'int64' and 'uint64'.

intmax('like', Y) returns the largest positive value in the integer class with the same data type and complexity (real or complex) as the numeric variable Y.

See also intmin, realmax.

Documentation for intmax

```
>> help intmin
```

intmin Smallest integer value.

X = intmin is the smallest value representable in an int32.

Any value that is smaller than intmin will saturate to intmin when cast to int32.

intmin('int32') is the same as intmin with no arguments.

intmin(CLASSNAME) is the smallest value in the integer class CLASSNAME.

Valid values of CLASSNAME are 'int8', 'uint8', 'int16', 'uint16',

```
'int32', 'uint32', 'int64' and 'uint64'.
```

intmin('like', Y) returns the smallest value in the integer class with the same data type and complexity (real or complex) as the numeric variable Y.

See also intmax, realmin.

Documentation for intmin

```
>> realmin
```

```
ans =
```

```
2.2251e-308
```

```
>> realmin('double')
```

```
ans =
```

```
2.2251e-308
```

```
>> realmin('single')
```

```
ans =
```

```
single
```

```
1.1755e-38
```

```
>> realmax
```

```
ans =
```

```
1.7977e+308
```

21. Use intmin and intmax to determine the range of values that can be stored in the types uint32 and uint64.

```
>> intmin('uint32')
```

```
ans =
```

```
uint32
```

```
0
```

```
>> intmax('uint32')
```

```
ans =
```

```
uint32
4294967295
>> intmin('uint64')
ans =
uint64
0
>> intmax('uint64')
ans =
uint64
18446744073709551615
```

22. Use the cast function to cast a variable to be the same type as another variable.

```
>> vara = uint16(6 + 9)
vara =
uint16
15
>> varb = 4*7
varb =
28
>> class(varb)
ans =
'double'
>> varb = cast(varb, 'like', vara)
varb =
uint16
28
>> class(varb)
```

```
ans =
```

```
'uint16'
```

23. Use help elfun or experiment to answer the following questions:

a. Is fix(3.5) the same as floor(3.5)?

```
>> fix(3.5)
```

```
ans =
```

```
3
```

```
>> floor(3.5)
```

```
ans =
```

```
3
```

b. Is fix(3.4) the same as fix(-3.4)?

```
>> fix(3.4)
```

```
ans =
```

```
3
```

```
>> fix(-3.4)
```

```
ans =
```

```
-3
```

c. Is fix(3.2) the same as floor(3.2)?

```
>> fix(3.2)
```

```
ans =
```

```
3
```

```
>> floor(3.2)
```

```
ans =
```

```
3
```

d. Is fix(-3.2) the same as floor(-3.2)?

```
>> fix(-3.2)
```

ans =

-3

```
>> floor(-3.2)
```

ans =

-4

e. Is fix(-3.2) the same as ceil(-3.2)?

```
>> fix(-3.2)
```

ans =

-3

```
>> ceil(-3.2)
```

ans =

-3

24.

a. For what range of values is the function round equivalent to the function floor?

For positive numbers: when the decimal part is less than .5

For negative numbers: when the decimal part is greater than or equal to .5

b. For what range of values is the function round equivalent to the function ceil?

For positive numbers: when the decimal part is greater than or equal to .5

For negative numbers: when the decimal part is less than .5

25. Use help to determine the difference between the rem and mod functions.

```
>> help rem
```

rem Remainder after division.

rem(x,y) returns $x - \text{fix}(x/y) \cdot y$ if $y \approx 0$, carefully computed to avoid rounding error. If y is not an integer and the quotient x/y is within roundoff error of an integer, then n is that integer. The inputs x and y must be real and have compatible sizes. In the simplest cases, they can be the same size or one can be a scalar. Two inputs have compatible sizes if, for every dimension, the dimension sizes of the

inputs are either the same or one of them is 1.

By convention:

`rem(x,0)` is NaN.

`rem(x,x)`, for $x \approx 0$, is 0.

`rem(x,y)`, for $x \approx y$ and $y \approx 0$, has the same sign as x .

Note: `MOD(x,y)`, for $x \approx y$ and $y \approx 0$, has the same sign as y .

`rem(x,y)` and `MOD(x,y)` are equal if x and y have the same sign, but differ by y if x and y have different signs.

See also `mod`.

Documentation for `rem`

Other functions named `rem`

`>> help mod`

`mod` Modulus after division.

`mod(x,y)` returns $x - \text{floor}(x./y).*y$ if $y \approx 0$, carefully computed to avoid rounding error. If y is not an integer and the quotient $x./y$ is within roundoff error of an integer, then n is that integer. The inputs x and y must be real and have compatible sizes. In the simplest cases, they can be the same size or one can be a scalar. Two inputs have compatible sizes if, for every dimension, the dimension sizes of the inputs are either the same or one of them is 1.

The statement " x and y are congruent mod m " means $\text{mod}(x,m) == \text{mod}(y,m)$.

By convention:

`mod(x,0)` is x .

`mod(x,x)` is 0.

`mod(x,y)`, for $x \approx y$ and $y \approx 0$, has the same sign as y .

Note: `REM(x,y)`, for $x \approx y$ and $y \approx 0$, has the same sign as x .

`mod(x,y)` and `REM(x,y)` are equal if x and y have the same sign, but differ by y if x and y have different signs.

See also `rem`.

Documentation for `mod`

Other functions named `mod`

26. Find MATLAB expressions for the following

$$\sqrt{19}$$

`>> sqrt(19)`

`ans =`

```
4.3589
```

$$3^{1.2}$$

```
>> 3^1.2
```

```
ans =
```

```
3.7372
```

$$\tan(\pi)$$

```
>> tan(pi)
```

```
ans =
```

```
-1.2246e-16
```

27. Using only the integers 2 and 3, write as many expressions as you can that result in 9. Try to come up with at least 10 different expressions (Note: don't just change the order). Be creative! Make sure that you write them as MATLAB expressions. Use operators and/or built-in functions.

```
>> 3^2
```

```
ans =
```

```
9
```

```
>> 2^3+(3-2)
```

```
ans =
```

```
9
```

```
>> 3*3
```

```
ans =
```

```
9
```

```
>> 3^3-3*3*2
```

```
ans =
```

```
9
```

```
>> 2^3+abs(2-3)
```

```
ans =
```

```
9  
>> 2^3+sign(3)
```

```
ans =
```

```
9
```

```
>> 3/2^2*3
```

```
ans =
```

```
9
```

```
>> 2\3^2*3
```

```
ans =
```

```
9
```

```
>> sqrt(3^(2+2))
```

```
ans =
```

```
9
```

28. A vector can be represented by its rectangular coordinates x and y or by its polar coordinates r and θ . Theta is measured in radians. The relationship between them is given by the equations:

$$x = r * \cos(\theta)$$

$$y = r * \sin(\theta)$$

```
>> r = 46  
  
r =  
  
46  
  
>> theta = 0.7  
  
theta =  
  
0.7000  
  
>> x = r*cos(theta)  
  
x =
```

```
35.1827
```

```
>> y = r*sin(theta)
```

```
y =
```

```
29.6340
```

29. In special relativity, the Lorentz factor is a number that describes the effect of speed on various physical properties when the speed is significant relative to the speed of light. Mathematically, the Lorentz factor is given as:

$$\gamma = \frac{1}{\sqrt{1 - \frac{v^2}{c^2}}}$$

Use 3×10^8 m/s for the speed of light, c . Create variables for c and the speed v and from them a variable $lore$ for the Lorentz factor.

```
>> c = 3e8
```

```
c =
```

```
300000000
```

```
>> v = 2.9e8
```

```
v =
```

```
290000000
```

```
>> lore = 1 / sqrt(1 - v^2/c^2)
```

```
lore =
```

```
3.9057
```

30. A company manufactures a part for which there is a desired weight. There is a tolerance of N percent, meaning that the range between minus and plus $N\%$ of the desired weight is acceptable. Create a variable that stores a weight, and another variable for N (for example, set it to two). Create variables that store the minimum and maximum values in the acceptable range of weights for this part.

```
>> weight = 46.6917
```

```
weight =
```

```
46.6917
```

```

>> N = 7

N =

7

>> min = weight - weight*0.01*N

min =

43.4233

>> max = weight + weight*0.01*N

max =

49.9601

```

31. An environmental engineer has determined that the cost C of a containment tank will be based on the radius r of the tank:

$$C = \frac{32430 + 428\pi}{r}$$

Create a variable for the radius, and then for the cost

```

>> radius = 46;
>> c = 32430/radius + 428*pi*radius

c =

62556.68

```

32. A chemical plant releases an amount A of pollutant into a stream. The maximum concentration C of the pollutant at a point which is a distance x from the plant is:

$$C = \frac{A}{x} \sqrt{\frac{2}{\pi e}}$$

Create variables for the values of A and x , and then for C . Assume that the distance x is in meters. Experiment with different values for x .

```

>> A = 80000

A =

80000.00

```

```

>> x = 100
x =
100.00

>> C = A/x * sqrt(2/(pi*exp(1)))

C =
387.15

>> x = 1000;
>> C = A/x * sqrt(2/(pi*exp(1)))

C =
38.72

>> x = 20000;
>> C = A/x * sqrt(2/(pi*exp(1)))

C =
1.94

```

33. The geometric mean g of n numbers x_i is defined as the n^{th} root of the product of x_i :

$$g = \sqrt[n]{x_1 x_2 x_3 \dots x_n}$$

(This is useful, for example, in finding the average rate of return for an investment which is something you'd do in engineering economics). If an investment returns 15% the first year, 50% the second, and 30% the third year, the average rate of return would be $(1.15 * 1.50 * 1.30)^{\frac{1}{3}}$.) Compute this.

```

>> x1 = 1.15
x1 =
1.15

>> x2 = 1.5
x2 =
1.50

```

```
>> x3 = 1.3  
  
x3 =  
  
    1.30  
  
>> g = nthroot(x1*x2*x3, 3)  
  
g =  
  
    1.31
```

34. Use the **deg2rad** function to convert 180 degrees to radians.

```
>> deg2rad(180)  
  
ans =  
  
    3.14
```

Roll. No. A016	Name: Varun Khadayate
Class B.Tech CsBs	Batch: 1
Date of Experiment: 29-07-2021	Subject: IT/WS

1. If a variable has the dimensions 3×4 , could it be considered to be (Check all that apply):
 - a matrix
 - a row vector
 - a column vector
 - a scalar
2. If a variable has the dimensions 1×5 , could it be considered to be (Check all that apply):
 - a matrix
 - a row vector
 - a column vector
 - a scalar
3. If a variable has the dimensions 5×1 , could it be considered to be (Check all that apply):
 - a matrix
 - a row vector
 - a column vector
 - a scalar
4. If a variable has the dimensions 1×1 , could it be considered to be (Check all that apply):
 - a matrix
 - a row vector
 - a column vector
 - a scalar
5. Using the colon operator, create the following row vectors

2	3	4	5	6	7
1.1000	1.3000	1.5000	1.7000		
8	6	4	2		

```

>> 2:7
ans =
2.00    3.00    4.00    5.00    6.00    7.00
>> 1.1000:0.2000:1.7000

ans =
1.1000  1.3000  1.5000  1.7000
>> 8:-2:2
ans =
8   6   4   2

```

6. Using a built-in function, create a vector vec which consists of 20 equally spaced points in the range from $-\pi$ to $+\pi$.

```
>> vec = linspace(0,2*pi,20)

vec =

Columns 1 through 12

0 0.3307 0.6614 0.9921 1.3228 1.6535 1.9842 2.3149 2.6456 2.9762 3.3069
3.6376

Columns 13 through 20

3.9683 4.2990 4.6297 4.9604 5.2911 5.6218 5.9525 6.2832
```

7. Write an expression using linspace that will result in the same as $2:0.2:3$

```
>> linspace(2,3,6)

ans =

2.0000 2.2000 2.4000 2.6000 2.8000 3.0000
```

8. Using the colon operator and also the linspace function, create the following row vectors:

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

```
>> -5:-1

ans =

-5 -4 -3 -2 -1

>> linspace(-5,-1,5)

ans =

-5 -4 -3 -2 -1

>> 5:2:9

ans =

5 7 9

>> linspace(5,9,3)

ans =
```

```
5 7 9

>> 8:-2:4

ans =

8 6 4

>> linspace(8,4,3)

ans =

8 6 4
```

9. How many elements would be in the vectors created by the following expressions?
a. linspace(3,2000)

100 (always, by default)

b. logspace(3,2000)

50 (always, by default – although these numbers would get very large quickly; most would be represented as Inf)

10. Create a variable myend which stores a random integer in the inclusive range from 5 to 9.
Using the colon operator, create a vector that iterates from 1 to myend in steps of 3.

```
>> myend = randi([5, 9])
```

```
myend =
```

```
9
```

```
>> vec = 1:3:myend
```

```
vec =
```

```
1 4 7
```

11. Using the colon operator and the transpose operator, create a column vector myvec that has the values -1 to 1 in steps of 0.5.

```
>> rowVec = -1: 0.5: 1
```

```
rowVec =
```

```
-1.0000 -0.5000 0 0.5000 1.0000
```

```
>> rowVec'
```

```
ans =
```

```
-1.0000
-0.5000
 0
 0.5000
 1.0000
```

12. Write an expression that refers to only the elements that have odd-numbered subscripts in a vector, regardless of the length of the vector. Test your expression on vectors that have both an odd and even number of elements.

```
>> vec = 1:8

vec =
1 2 3 4 5 6 7 8

>> vec(1:2:end)

ans =
1 3 5 7

>> vec = 4:12

vec =
4 5 6 7 8 9 10 11 12

>> vec(1:2:end)

ans =
4 6 8 10 12
```

13. Generate a 2×4 matrix variable mat. Replace the first row with 1:4. Replace the third column (you decide with which values).

```
>> mat = [2:5; 1 4 11 3]

mat =
2 3 4 5
1 4 11 3

>> mat(1,:) = 1:4

mat =
1 2 3 4
1 4 11 3

>> mat(:,3) = [4;3]
```

```
mat =  
  
1 2 4 4  
1 4 3 3
```

14. Generate a 2×4 matrix variable *mat*. Verify that the number of elements is the product of the number of rows and columns

```
>> mat = randi(20,2,4)
```

```
mat =  
  
6 5 14 4  
15 1 9 6  
  
>> [r c] = size(mat)  
  
r =  
  
2  
  
c =  
  
4  
  
>> numel(mat) == r * c  
  
ans =  
  
logical  
  
1
```

15. Which would you normally use for a matrix: length or size? Why?

Size, because it tells you both the number of rows and columns.

16. When would you use length vs. size for a vector?

If you want to know the number of elements, you'd use length. If you want to figure out whether it's a row or column vector, you'd use size.

17. Generate a 2×3 matrix of random

a. real numbers, each in the range (0, 1)

```
>> rand(2,3)
```

```
ans =
```

```
0.0334 0.0782 0.7775  
0.0746 0.4518 0.1108
```

b. real numbers, each in the range (0, 10)

```
>> rand(2,3)*10

ans =

0.0534  6.6795  6.0724
9.5356  6.9165  9.2982
```

c. integers, each in the inclusive range from 5 to 20

```
>> randi([5, 20],2,3)

ans =

17  16  13
14  11  14
```

18. Create a variable rows that is a random integer in the inclusive range from 1 to 5. Create a variable cols that is a random integer in the inclusive range from 1 to 5. Create a matrix of all zeros with the dimensions given by the values of rows and cols.

```
>> rows = randi([1,5])

rows =

3

>> cols = randi([1,5])

cols =

2

>> zeros(rows,cols)

ans =

0  0
0  0
0  0
```

19. Create a matrix variable mat. Find as many expressions as you can that would refer to the last element in the matrix, without assuming that you know how many elements or rows or columns it has (i.e., make your expressions general).

```
>> mat = [12:15; 6:-1:3]

mat =

12  13  14  15
6   5   4   3

>> mat(end,end)
```

```
ans =  
3  
>> mat(end)  
ans =  
3  
>> [r c] = size(mat)  
r =  
2  
c =  
4  
>> mat(r,c)  
ans =  
3
```

20. Create a vector variable `vec`. Find as many expressions as you can that would refer to the last element in the vector, without assuming that you know how many elements it has (i.e., make your expressions general).

```
>> vec = 1:2:9  
vec =  
1 3 5 7 9  
>> vec(end)  
ans =  
9  
>> vec(numel(vec))  
ans =  
9  
>> vec(length(vec))
```

```
ans =  
9  
>> v = fliplr(vec)  
  
v =  
9 7 5 3 1  
>> v(1)  
  
ans =  
9
```

21. Create a 2×3 matrix variable mat. Pass this matrix variable to each of the following functions and make sure you understand the result: flip, fliplr, flipud, and rot90. In how many different ways can you reshape it?

```
>> mat = randi([1,20], 2,3)
```

```
mat =  
19 20 3  
8 19 10
```

```
>> flip(mat)
```

```
ans =  
8 19 10  
19 20 3
```

```
>> fliplr(mat)
```

```
ans =  
3 20 19  
10 19 8
```

```
>> flipud(mat)
```

```
ans =  
8 19 10  
19 20 3
```

```
>> rot90(mat)
```

```
ans =
```

```

3 10
20 19
19 8

>> rot90(rot90(mat))

ans =

10 19 8
3 20 19

>> reshape(mat,3,2)

ans =

19 19
8 3
20 10

>> reshape(mat,1,6)

ans =

19 8 20 19 3 10

>> reshape(mat,6,1)

ans =

19
8
20
19
3
10

```

22. What is the difference between `fliplr(mat)` and `mat = fliplr(mat)`?

`fliplr(mat)` stores the result in `ans` so `mat` is not changed.

`mat = fliplr(mat)` changes `mat`.

23. Use `reshape` to reshape the row vector `1:4` into a `2x2` matrix; store this in a variable named `mat`. Next, make `2x3` copies of `mat` using both `repelem` and `repmat`.

```
>> mat = reshape(1:4,2,2)
```

```
mat =
```

```
1 3
2 4
```

```
>> repelem(mat,2,3)
```

```
ans =  
  
1 1 1 3 3 3  
1 1 1 3 3 3  
2 2 2 4 4 4  
2 2 2 4 4 4
```

```
>> repmat(mat,2,3)
```

```
ans =  
  
1 3 1 3 1 3  
2 4 2 4 2 4  
1 3 1 3 1 3  
2 4 2 4 2 4
```