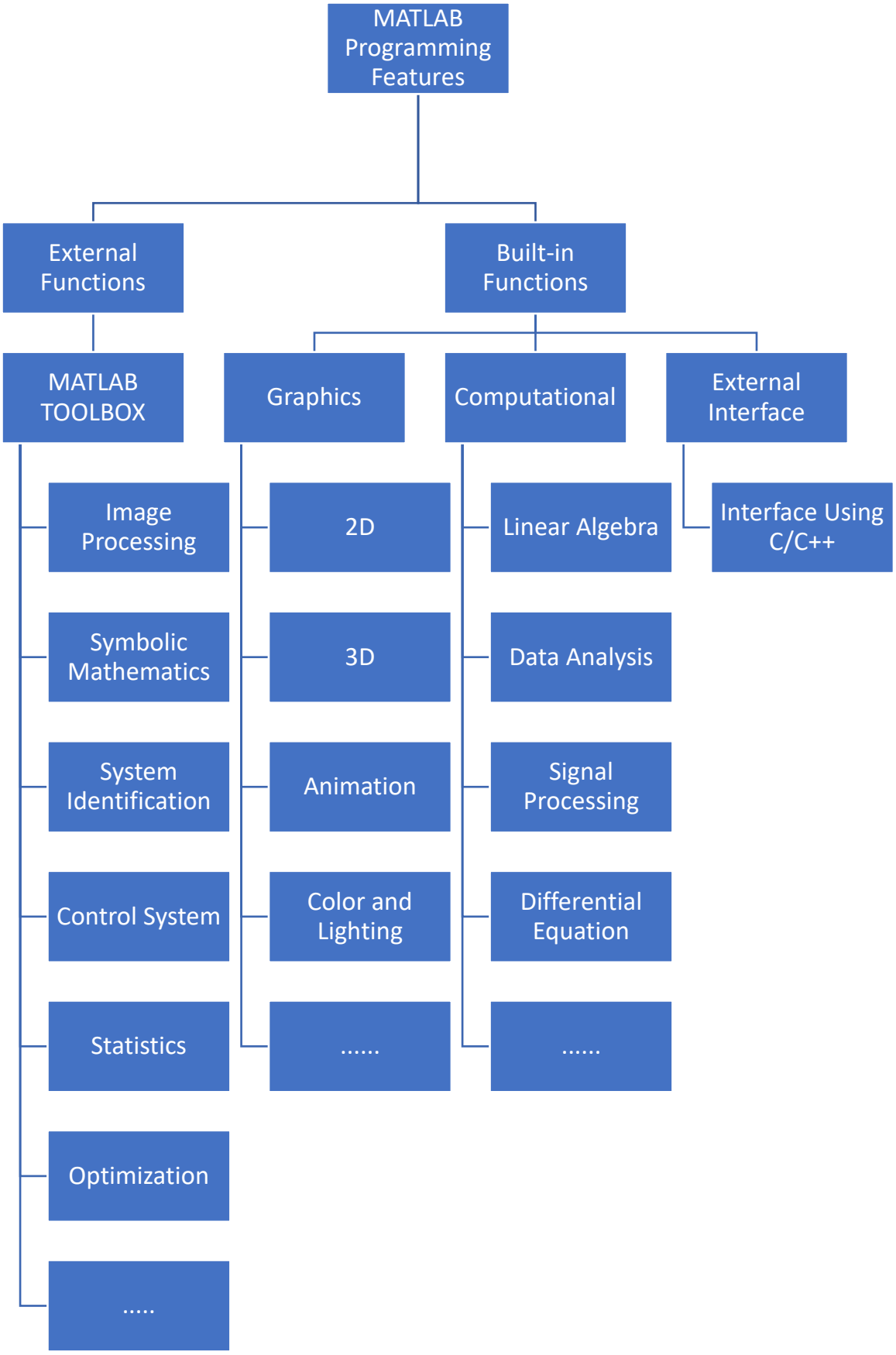

MATLAB Programming Features
External Functions
Built-in Functions
MATLAB TOOLBOX
Graphics
Computational
External Interface
Image Processing
Symbolic Mathematics
System Identification
Control System
Statistics
Optimization
.....
2D
3D
Animation
Color and Lighting
.....
Linear Algebra
Data Analysis
Signal Processing
Differential Equation
.....
Interface Using C/C++

UNIT – I

History of MATLAB

1. MATLAB was not a programming language but was a simple interactive calculator.
2. 1970 there was EISPACK (Matrix Eigon 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 Molar with Jack Little and Steve Bangret 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

SVKM'S NMIMS Deemed to be UNIVERSITY | NAVI MUMBAI

MATLAB
Lab1

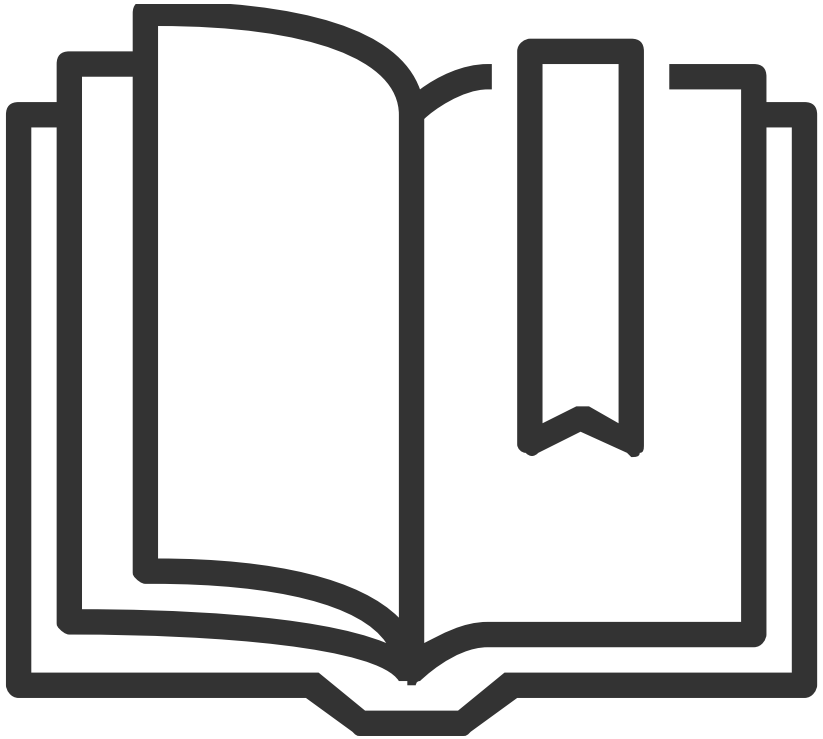
BTech (CSBS) -Semester VII

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= cosh2x-sinh2x; 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(∞)



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

*few figures/content have been prepared/referred from internet source/books

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

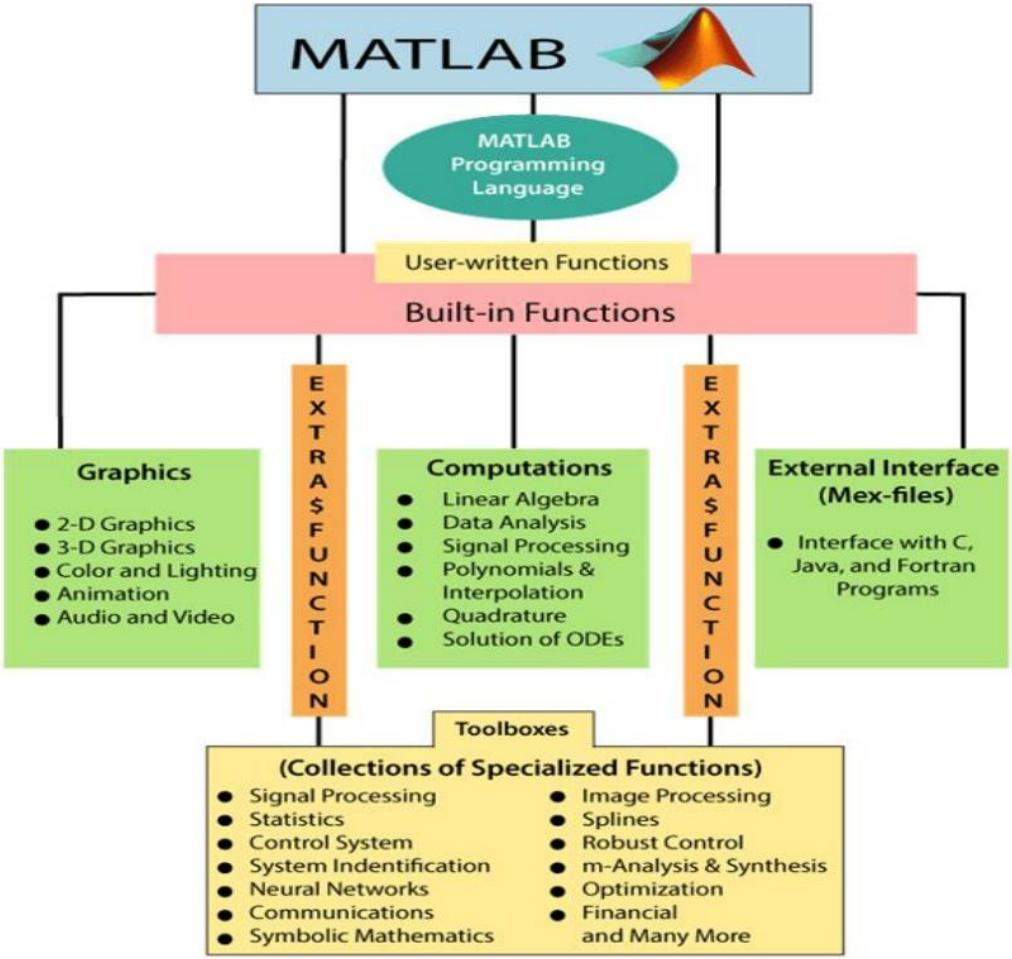
8. MATLAB and Simulink:

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



SVKM'S NMIMS Deemed to be UNIVERSITY | NAVI MUMBAI

MATLAB
Unit 1-Lecture 10

BTech (CSBS) -Semester VII

12 August 2022, 09:35AM

*few figures/content have been prepared/referred from internet source/books

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.

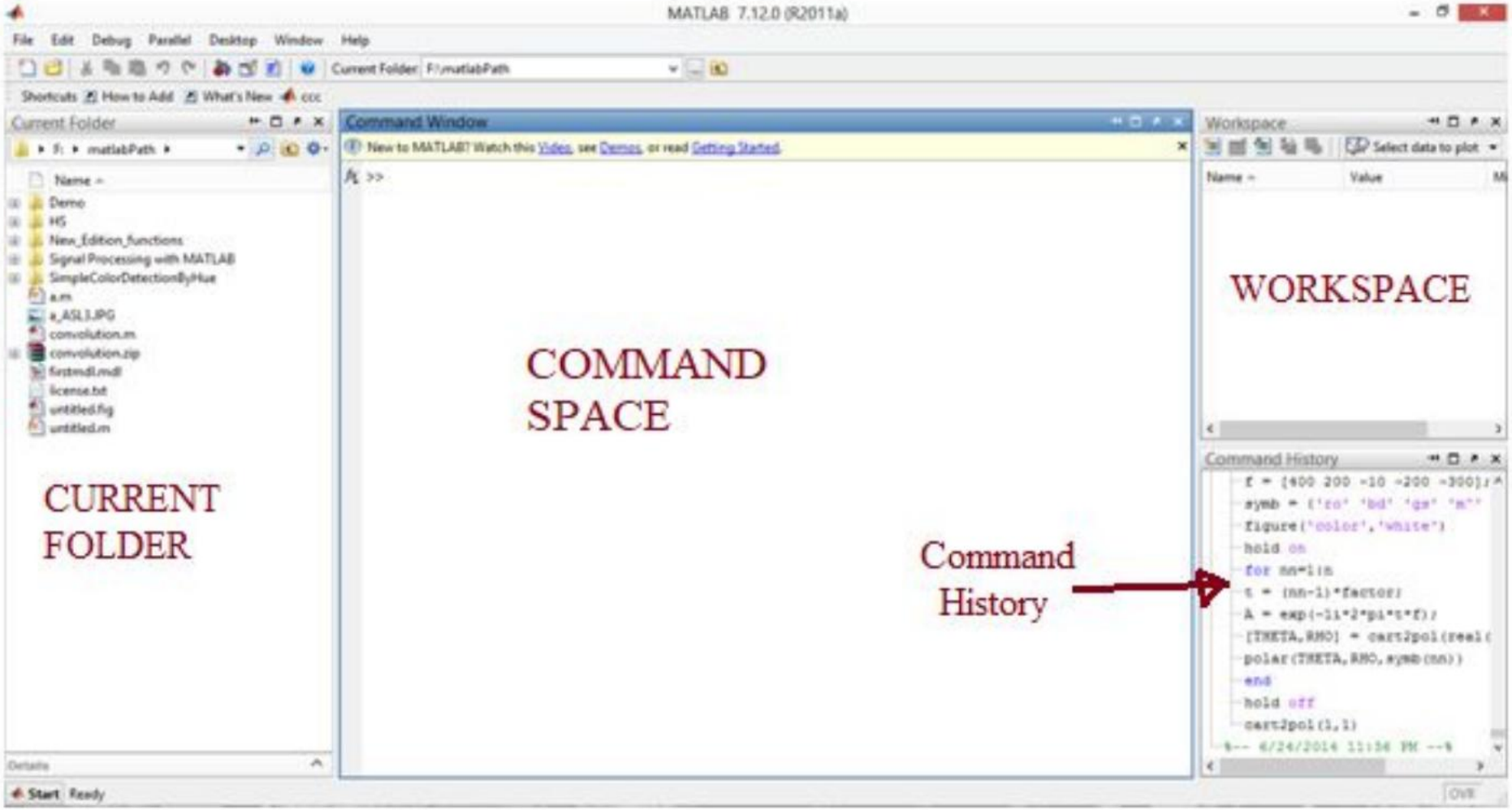
MATLAB good programming practices

- 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



SVKM'S NMIMS Deemed to be UNIVERSITY | NAVI MUMBAI

MATLAB
Unit 4-Lecture 11

BTech (CSBS) -Semester VII

16 August 2022, 09:35AM

*few figures/content have been prepared/referred from internet source/books

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	+	plus sign
`m`	magenta	`--`	dashed	o	circle
`c`	cyan	`:`	dotted	*	asterisk
`r`	red	`-.`	dash-dot	x	x-mark
`g`	green	`none`	no line	.	point
`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,

`xlabel('Pipe Length')`	labels the x-axis with `Pipe Length`,
`ylabel('Fluid Pressure')`	labels the y-axis with `Fluid Pressure`,
`title('Pressure Variation')`	titles the plot with `Pressure Variation`, and
`text(2,6,'Note this dip')`	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,

`xlabel('Pipe Length')`	labels the x-axis with `Pipe Length`,
`ylabel('Fluid Pressure')`	labels the y-axis with `Fluid Pressure`,
`title('Pressure Variation')`	titles the plot with `Pressure Variation`, and
`text(2,6,'Note this dip')`	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 *string1*, *string2*, 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

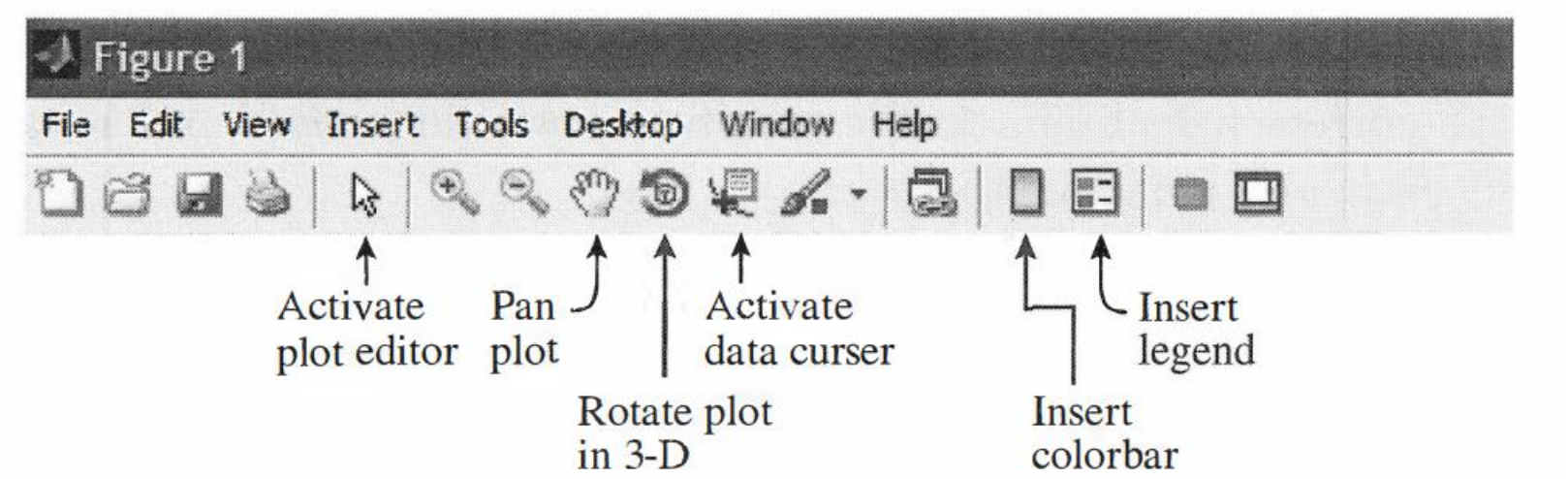
`axis('equal')`	sets equal scale on both axes,
`axis('square')`	sets the default rectangular frame to a square,
`axis('normal')`	resets the axis to default values,
`axis('axis')`	freezes the current axes limits, and
`axis('off')`	removes the surrounding frame and the tick marks.

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



SVKM'S NMIMS Deemed to be UNIVERSITY | NAVI MUMBAI

MATLAB
Unit 4-Lecture 12

BTech (CSBS) -Semester VII

26 August 2022, 09:35AM

*few figures/content have been prepared/referred from internet source/books

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

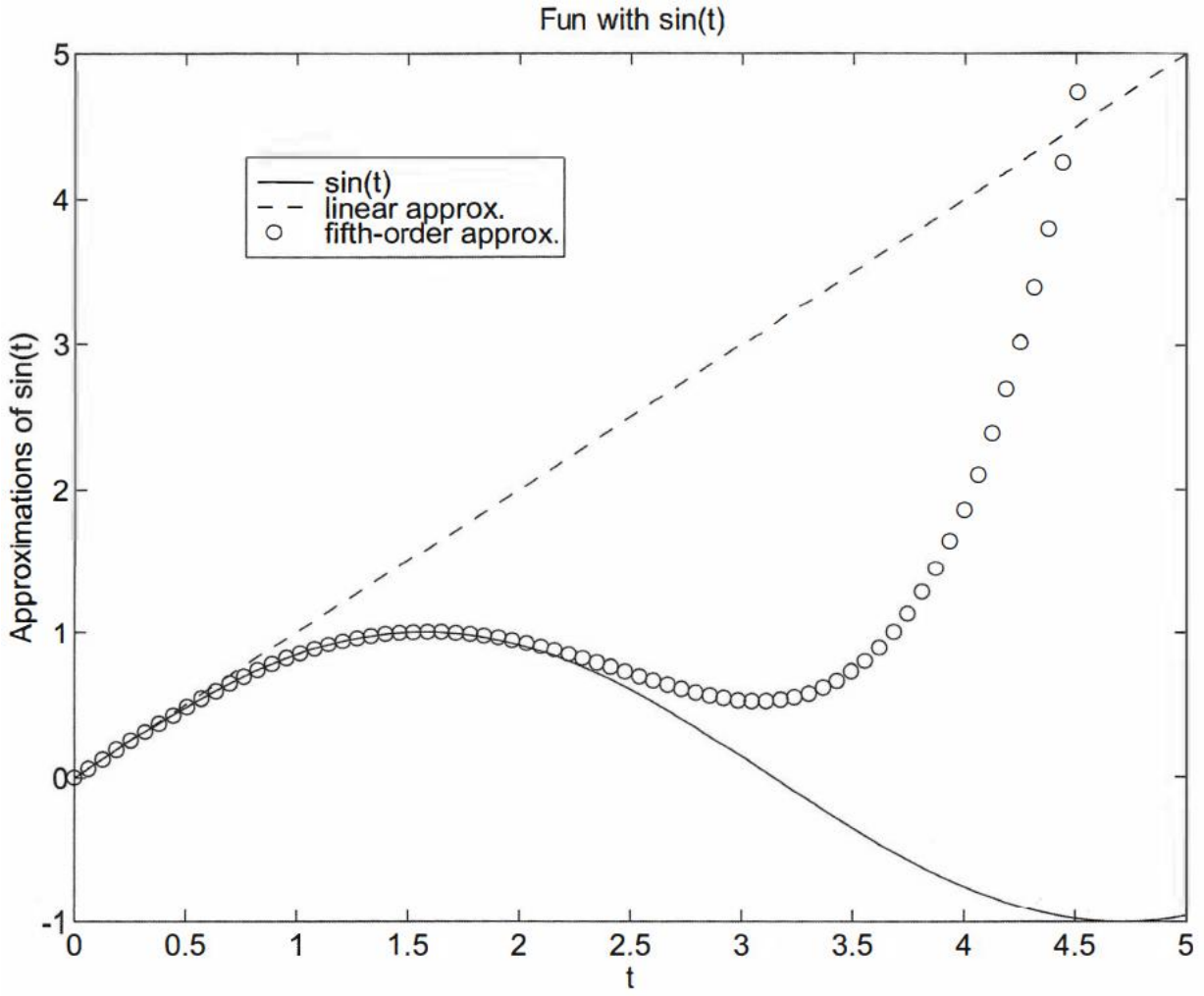
```
% -- Script file to generate an overlay plot with the line command --
% -----------------
% First, generate some data
t = linspace(0,2*pi,100);       % Generate vector t
y1 = sin(t);                    % Calculate y1, y2, y3
y2 = t;
y3 = t - (t.^3)/6 + (t.^5)/120;

% 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

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

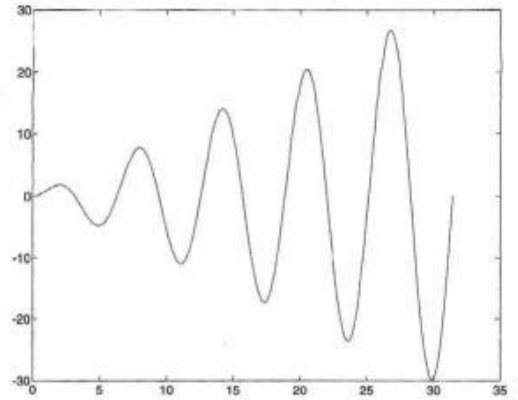
Specialized 2D plot

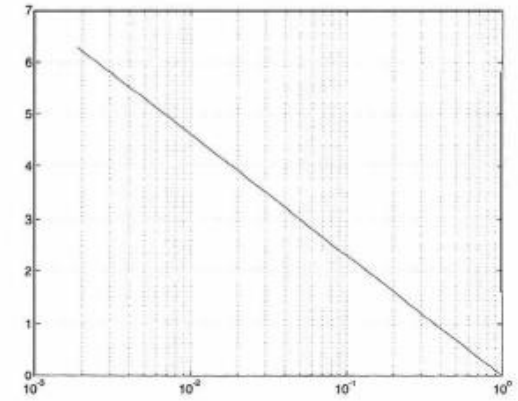
`fplot`	plots a function of a single variable,
`hist`	makes histograms,
`loglog`	creates plot with log scale on both the x-axis and the y-axis,
`pareto`	makes pareto plots,
`pcolor`	makes pseudocolor plot of a matrix,
`pie`	creates a pie chart,
`plotyy`	makes a double y-axis plot,
`plotmatrix`	makes 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

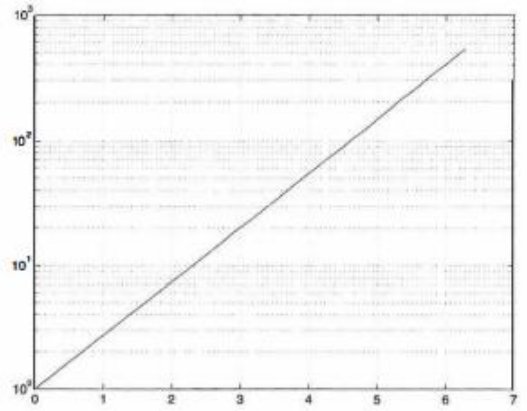
`semilogx`	makes semilog plot with log scale on the x-axis,
`semilogy`	makes semilog plot with log scale on the y-axis,
`stairs`	plots a stair graph, and
`stem`	plots a stem graph.

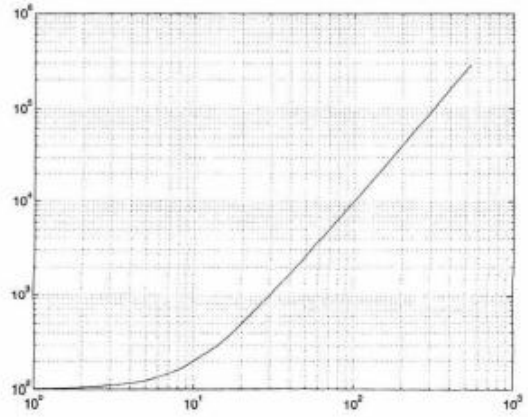
Specialized 2D plot

Function	Example Script	Output
fplot	$f(t) = t\sin t, \ \ 0 \le t \le 10\pi$ `fplot('x.*sin(x)',[0 10*pi])` Note that the function to be plotted must be written as a function of x.	

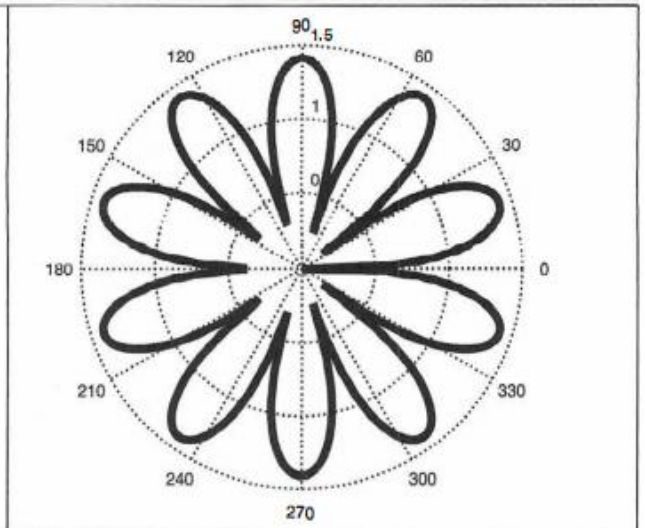
semilogx	$x = e^{-t}, \ \ y = t, \ 0 \le t \le 2\pi$
	`t = linspace(0,2*pi,200);`
	`x = exp(-t); y = t;`
	`semilogx(x,y), grid`
		

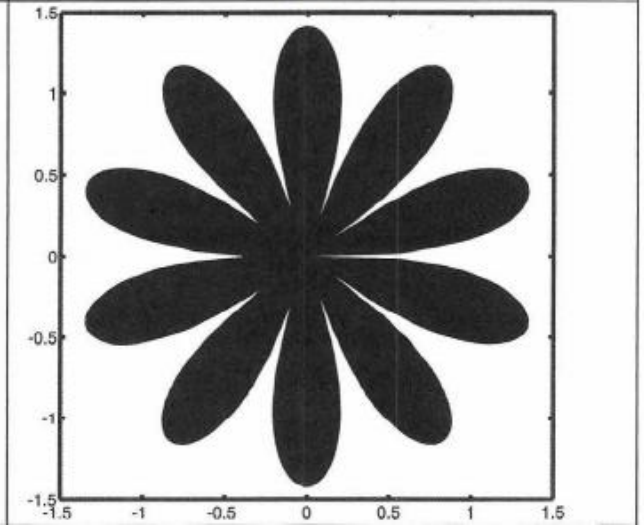
Specialized 2D plot

semilogy	$x = t,\ y = e^t,\ \ 0 \le t \le 2\pi$
	`t = linspace(0,2*pi,200);`
	`semilogy(t,exp(t))`
	`grid`
		

loglog	$x = e^t,\ y = 100 + e^{2t},\ \ 0 \le t \le 2\pi$
	`t = linspace(0,2*pi,200);`
	`x = exp(t);`
	`y = 100 + exp(2*t);`
	`loglog(x,y), grid`
		

Specialized 2D plot

polar	$r^2 = 2\sin 5t, \quad 0 \le t \le 2\pi$
	`t = linspace(0,2*pi,200);`
	`r = sqrt(abs(2*sin(5*t)));`
	`polar(t,r)`
		

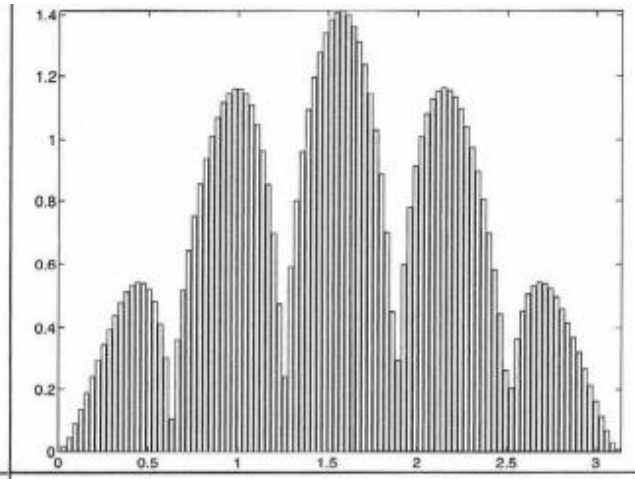
fill	$r^2 = 2\sin 5t, \quad 0 \le t \le 2\pi$
	$x = r\cos t, \quad y = r\sin t$
	`t = linspace(0,2*pi,200);`
	`r = sqrt(abs(2*sin(5*t)));`
	`x = r.*cos(t);`
	`y = r.*sin(t);`
	`fill(x,y,'k'),`
	`axis('square')`
		

Specialized 2D plot

bar

$$r^2 = 2\sin 5t, \quad 0 \le t \le 2\pi$$
$$y = r\sin t$$

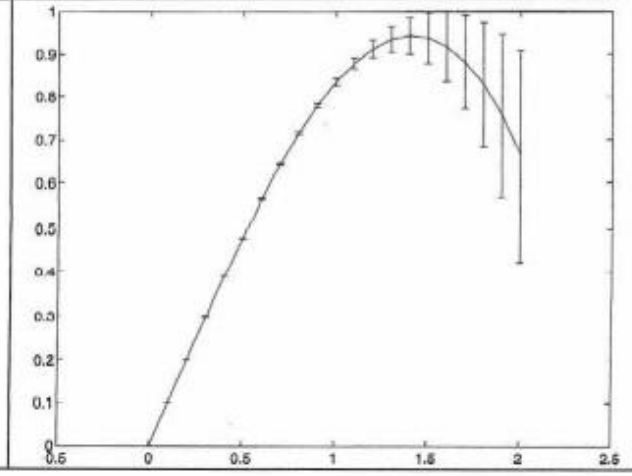
```
t = linspace(0,2*pi,200);
r = sqrt(abs(2*sin(5*t)));
y = r.*sin(t);
bar(t,y)
axis([0 pi 0 inf]);
```



errorbar

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

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

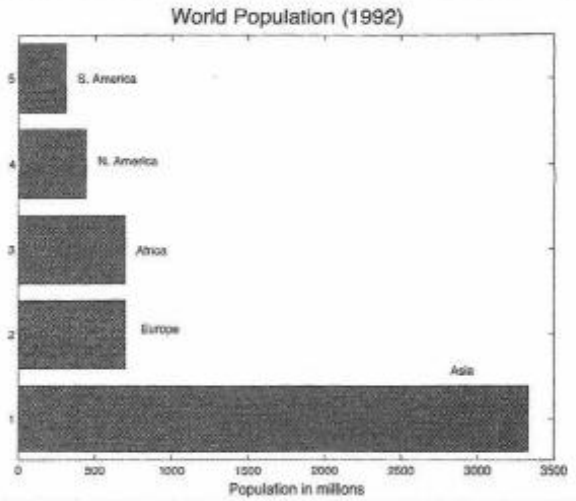


Specialized 2D plot

barh

World population by continents.

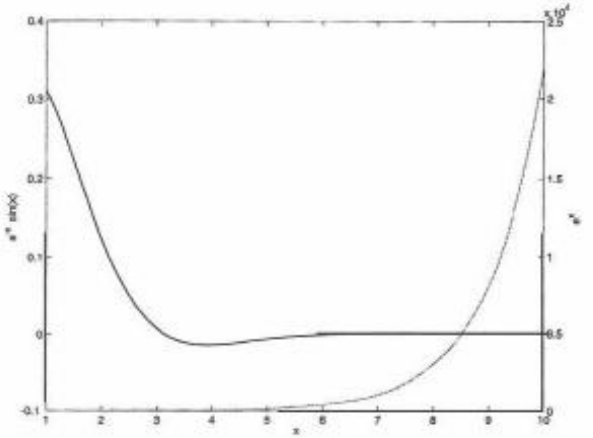
```
cont = char('Asia','Europe','Africa',...
     'N. America','S. America');
pop = [3332;696;694;437;307];
barh(pop)
for i=1:5,
      gtext(cont(i,:));
end
xlabel('Population in millions')
Title('World Population (1992)',
      'fontsize',18)
```



plotyy

$$y_1 = e^{-x}\sin x,\ 0 \le t \le 10$$
$$y_2 = e^{x}$$

```
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 ');
```

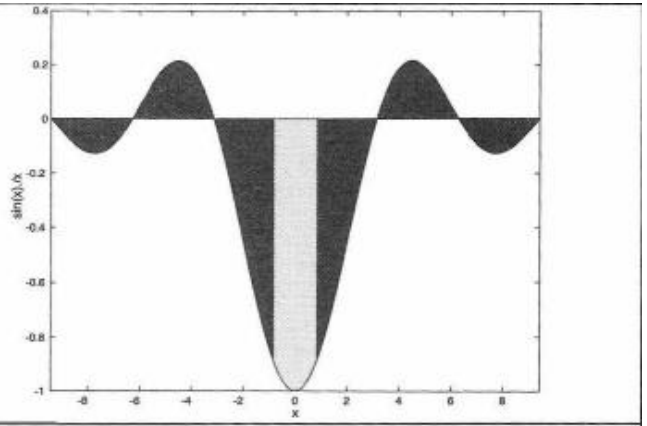


Specialized 2D plot

area

$$y = \frac{\sin(x)}{x}, \quad -3\pi \le x \le 3\pi$$

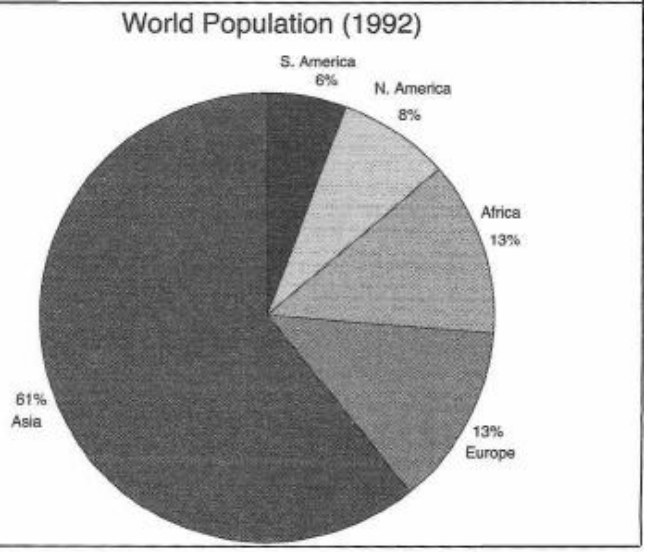
```
x = linspace(-3*pi,3*pi,100);
y = -sin(x)./x;
area(x,y)
xlabel('x'), ylabel('sin(x)./x')
hold on
x1 = x(46:55); y1 = y(46:55);
area(x1,y1,'facecolor','y')
```



pie

World population by continents.

```
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)
```



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)
```

SVKM'S NMIMS Deemed to be UNIVERSITY | NAVI MUMBAI

MATLAB
Unit 4-Lecture 13

BTech (CSBS) -Semester VII

30 August 2022, 09:35AM

*few figures/content have been prepared/referred from internet source/books

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:

$$y1 = \cos(x)$$
$$y2 = x^2 - 1$$

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 intoaccount? 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

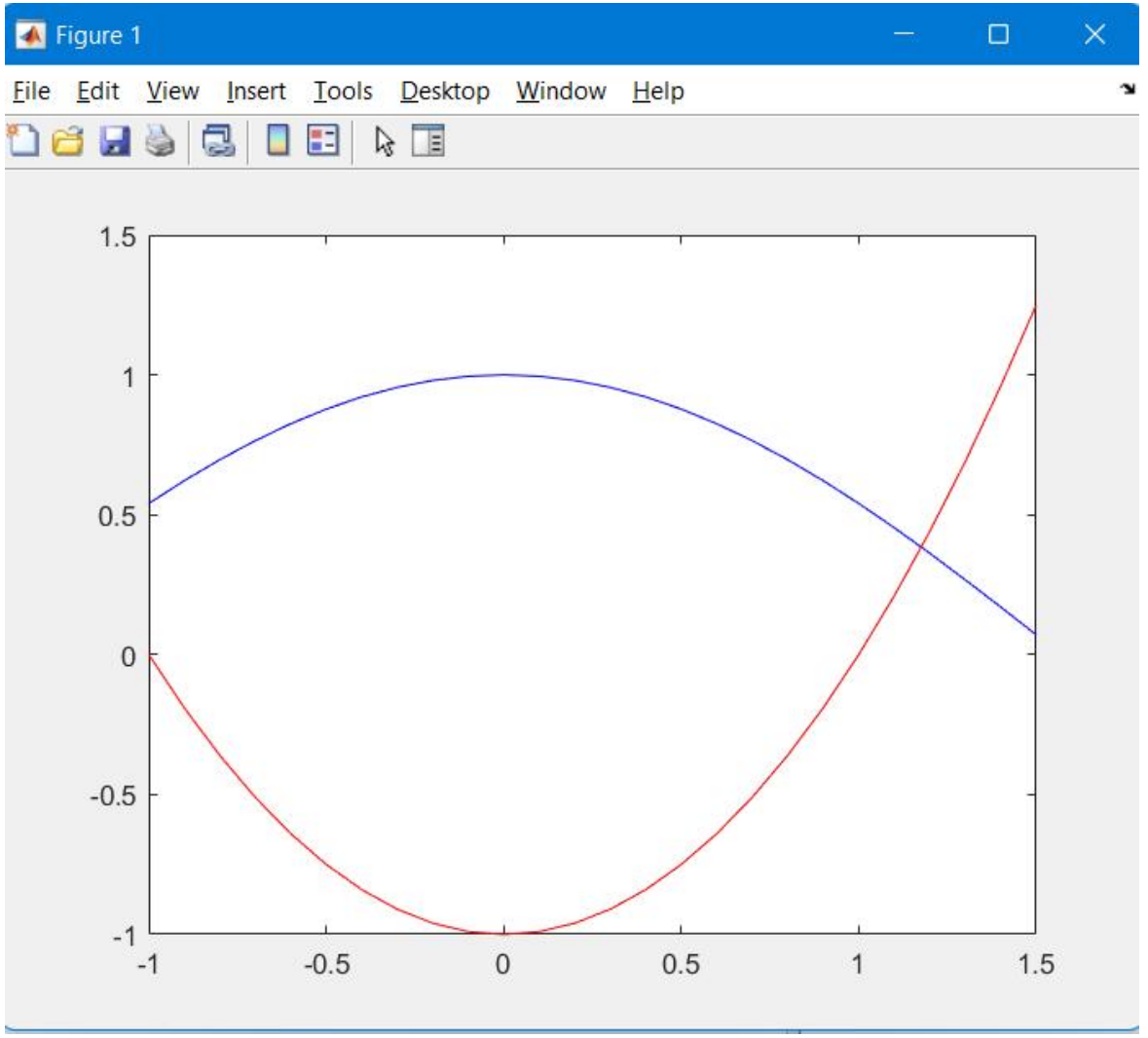
pie

hist, stairs

Solution

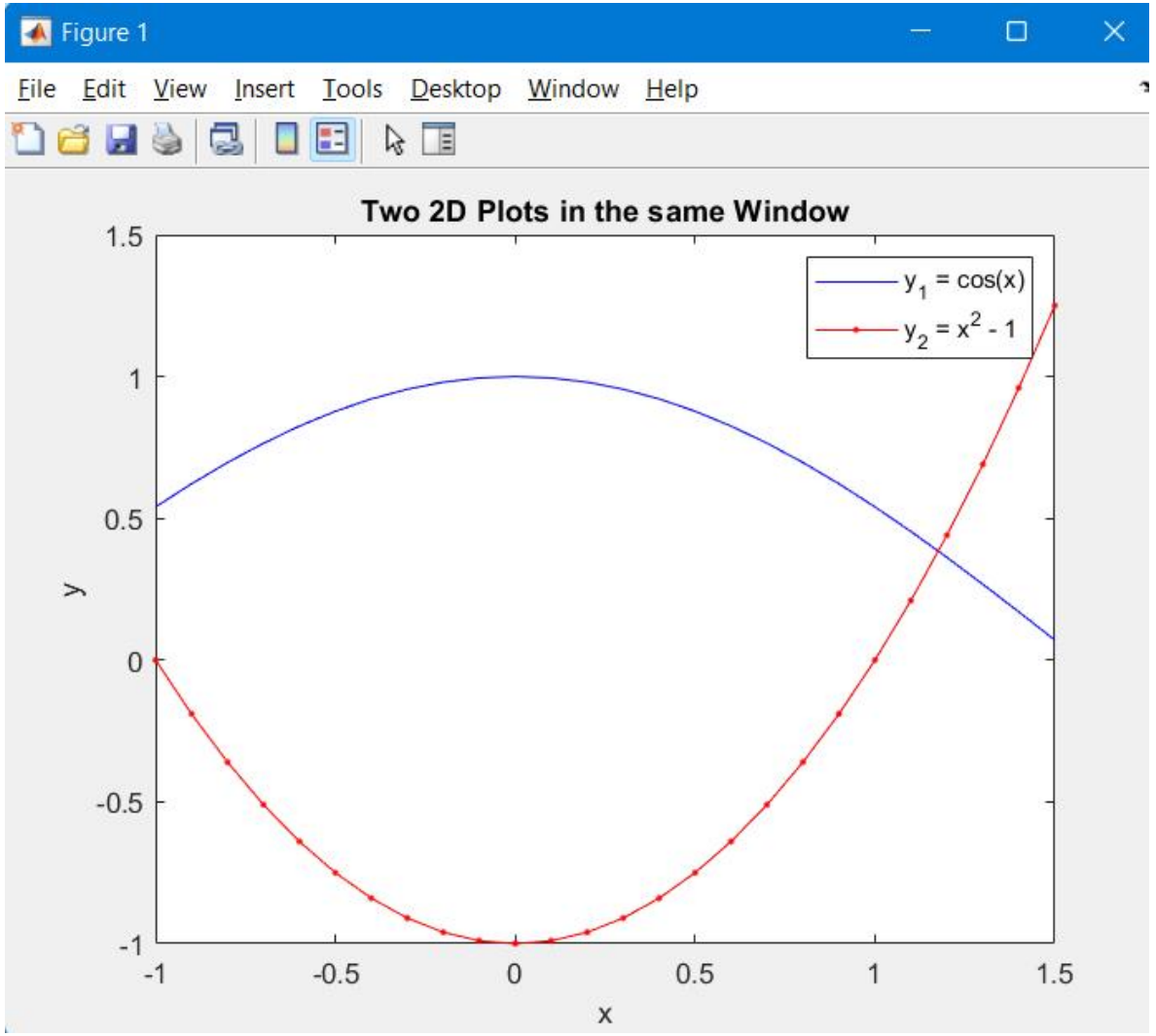
lab6.m

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



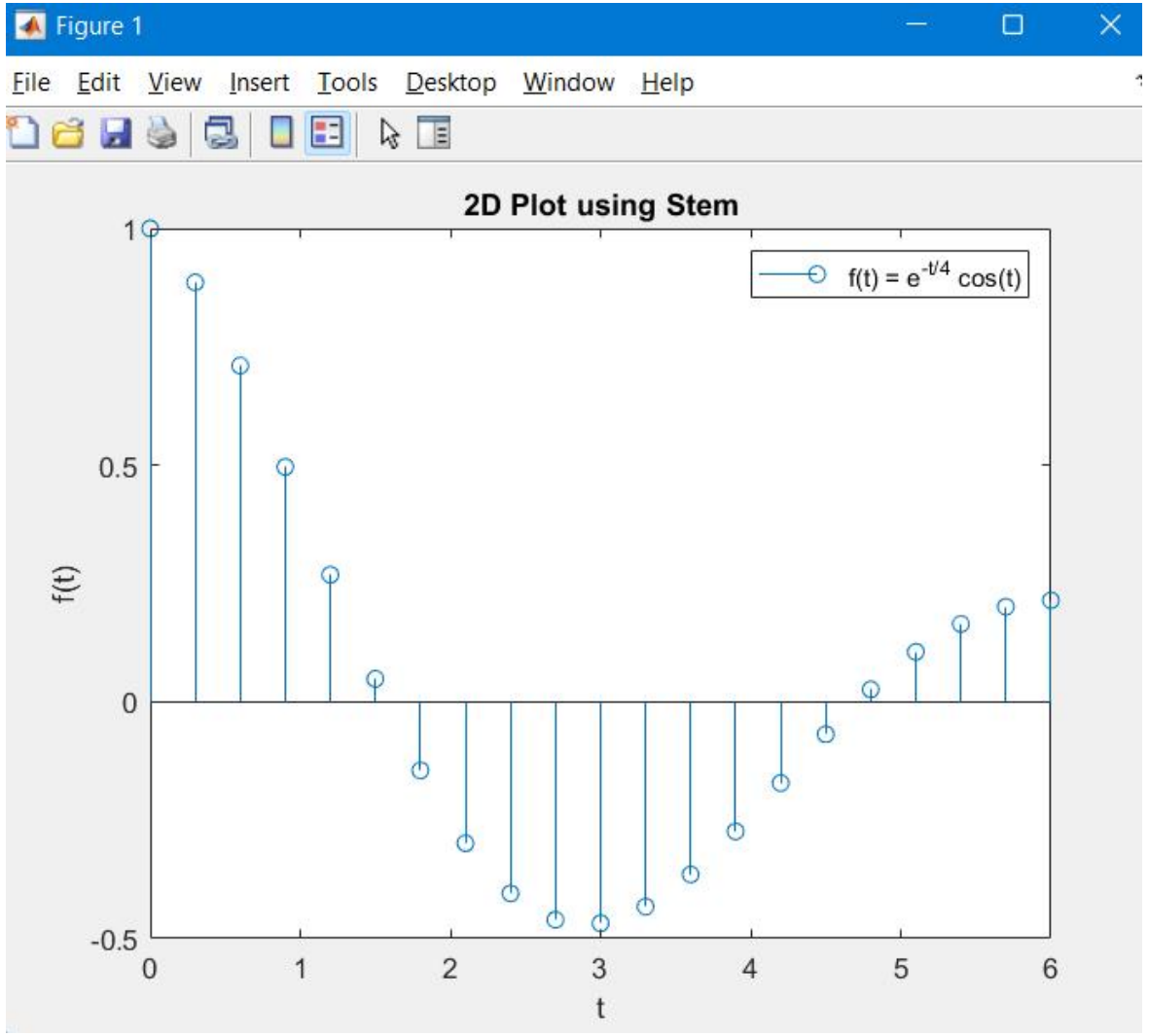
Solution

```
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
```



SVKM'S NMIMS Deemed to be UNIVERSITY | NAVI MUMBAI

MATLAB
Unit 4-Lecture 14

BTech (CSBS) -Semester VII

2 September 2022, 09:35AM

*few figures/content have been prepared/referred from internet source/books

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')
```

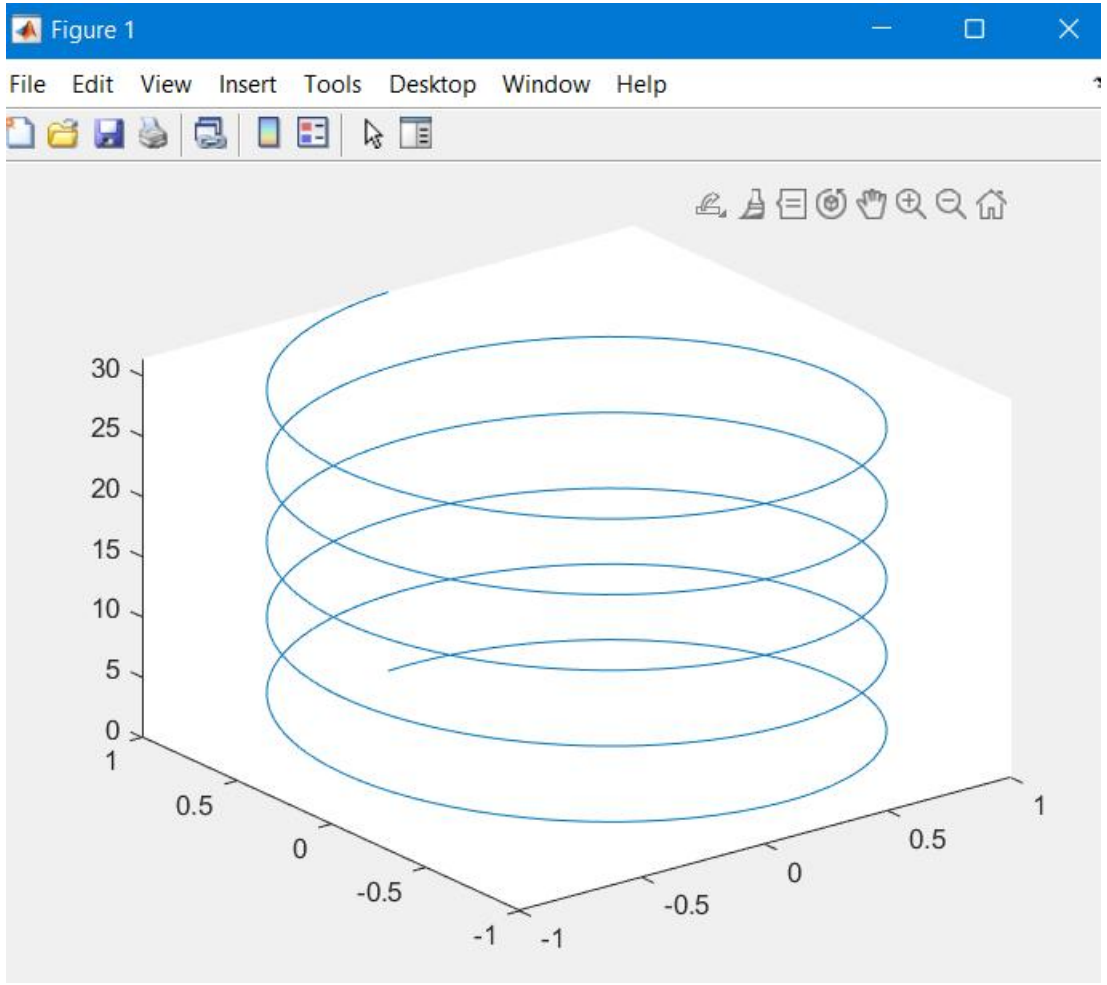
`plot3`	plots curves in space,
`stem3`	creates discrete data plot with stems in 3-D,
`bar3`	plots 3-D bar graph,
`bar3h`	plots 3-D horizontal bar graph,
`pie3`	makes 3-D pie chart,
`comet3`	makes animated 3-D line plot,
`fill3`	draws filled 3-D polygons,
`contour3`	makes 3-D contour plots,
`quiver3`	draws vector fields in 3-D,
`scatter3`	makes scatter plots in 3-D,
`mesh`	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 *waterfall* plot of 3-D data,
`cylinder`	generates a cylinder,
`ellipsoid`	generates an ellipsoid, and
`sphere`	generates a sphere.

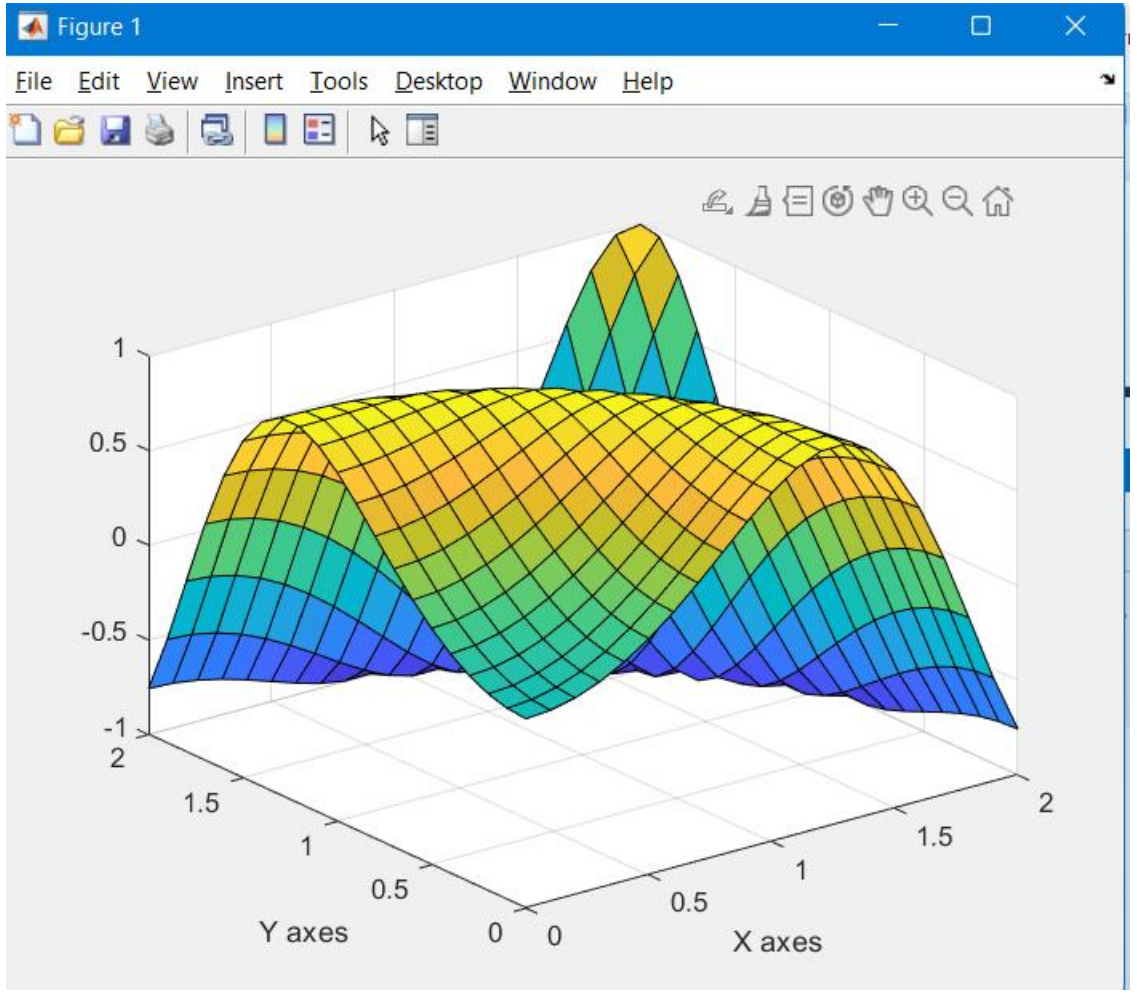
3D Plot: Question 1

```
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')
```

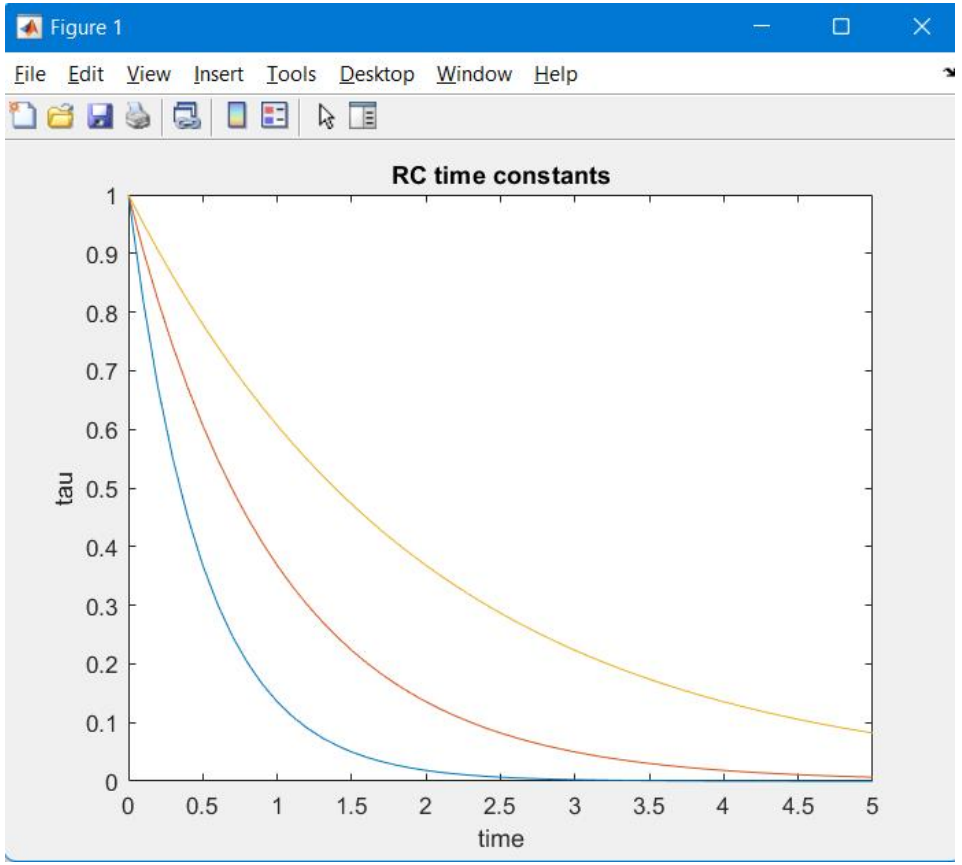


Question 3

Plot voltage vs time for various RC time constants

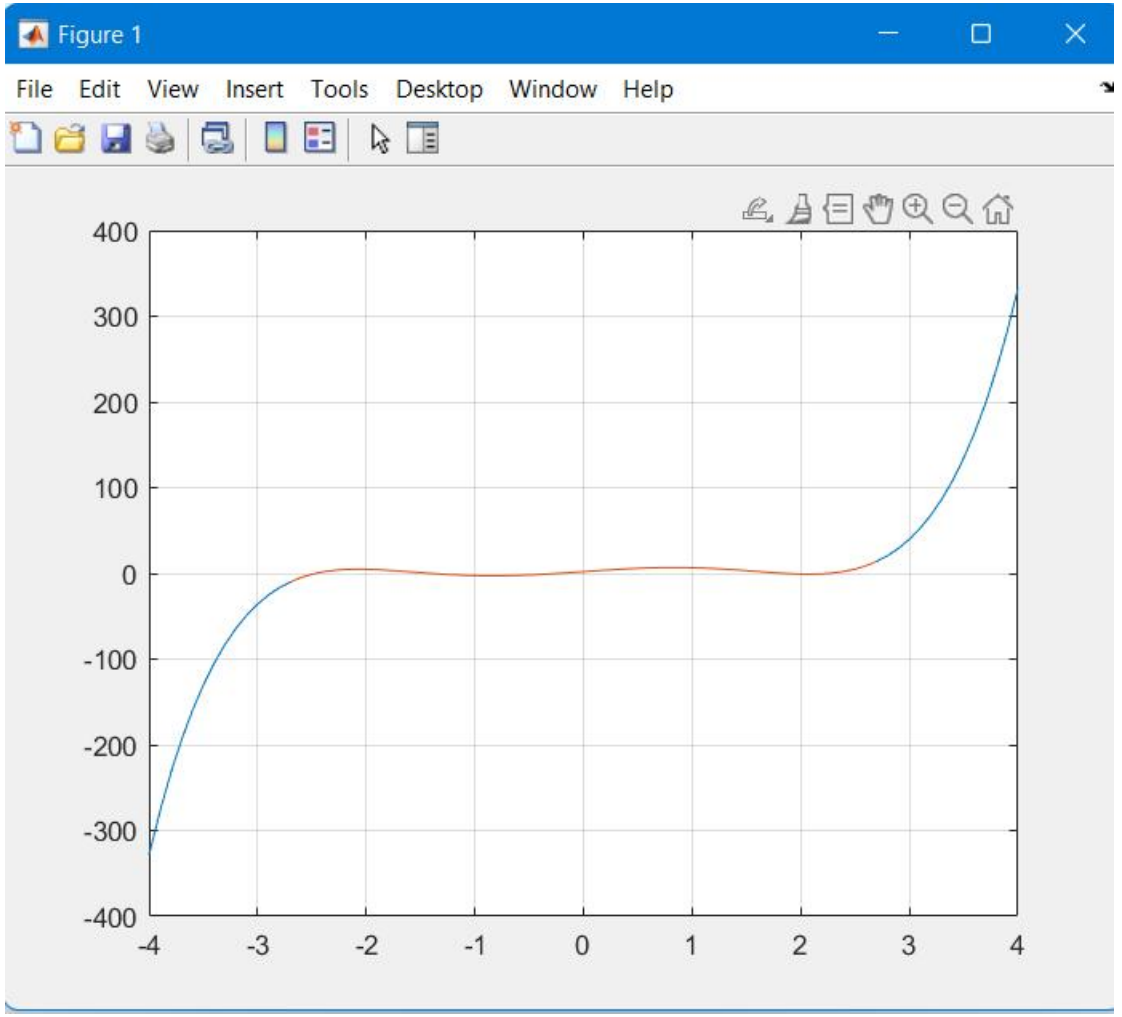
$$\frac{\vartheta}{V} = e^{-t/\tau}$$

```
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')
```



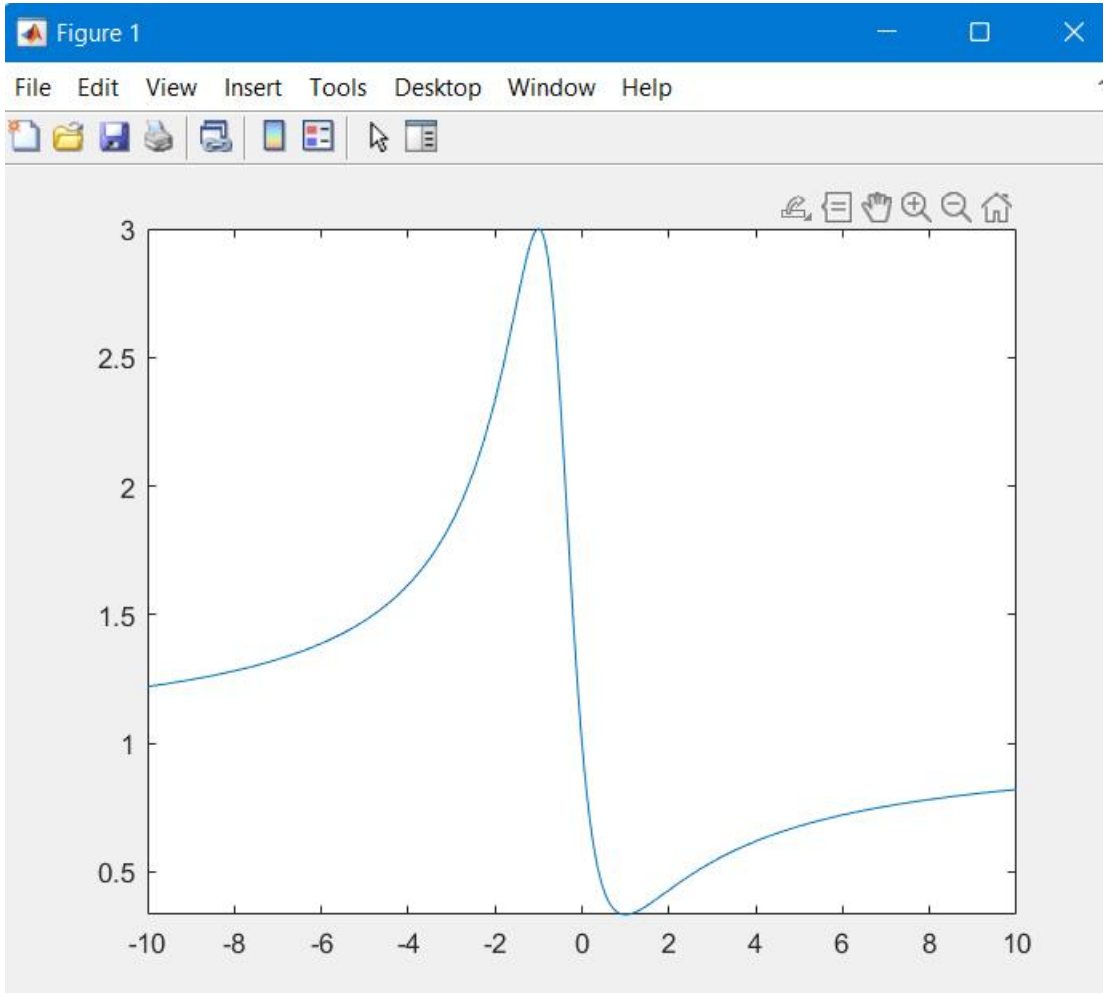
Question 4

```
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
```



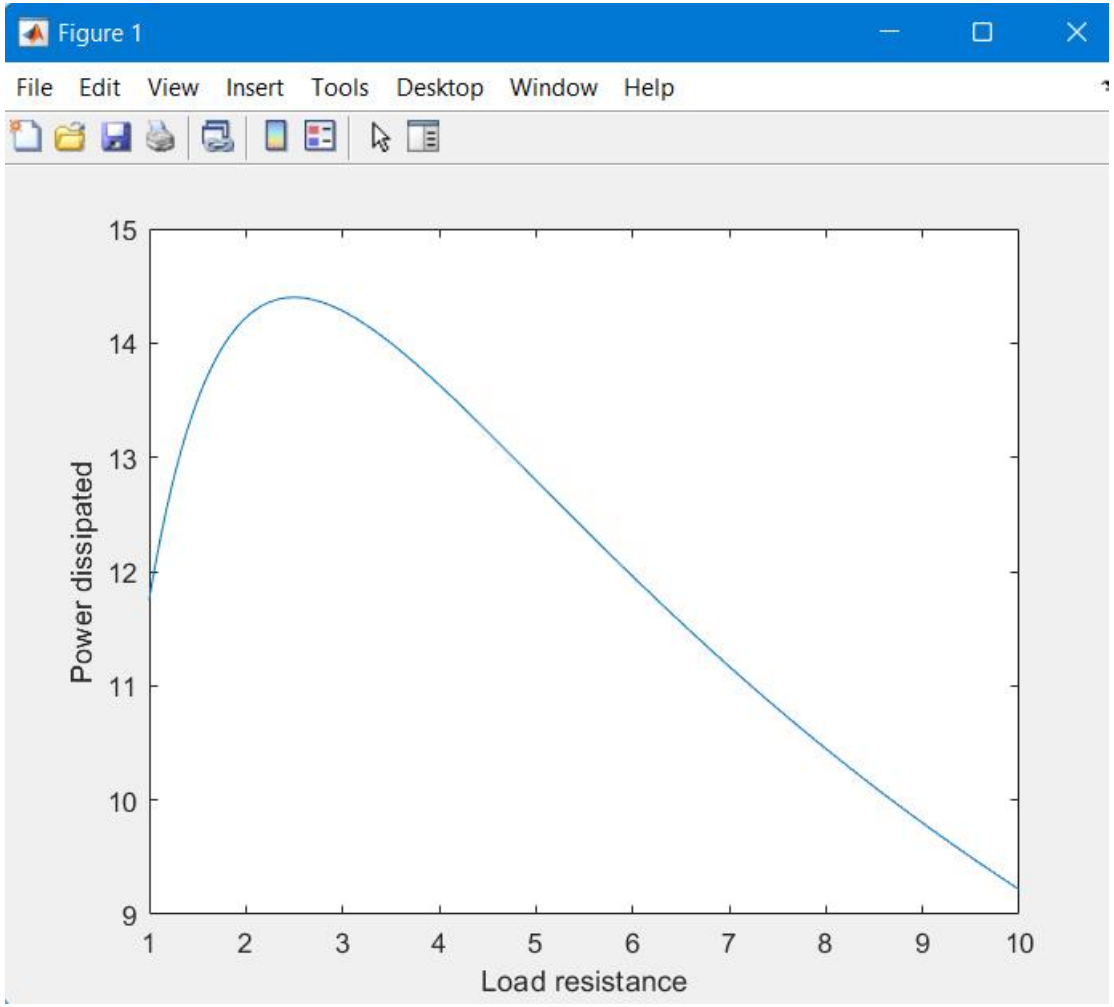
Question 5

```
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')
```



SVKM'S NMIMS Deemed to be UNIVERSITY | NAVI MUMBAI

MATLAB
Unit 5-Lecture 15

BTech (CSBS) -Semester VII

6 September 2022, 09:35AM

*few figures/content have been prepared/referred from internet source/books

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 seperate steps and then it is refined until results are manageble.
- 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

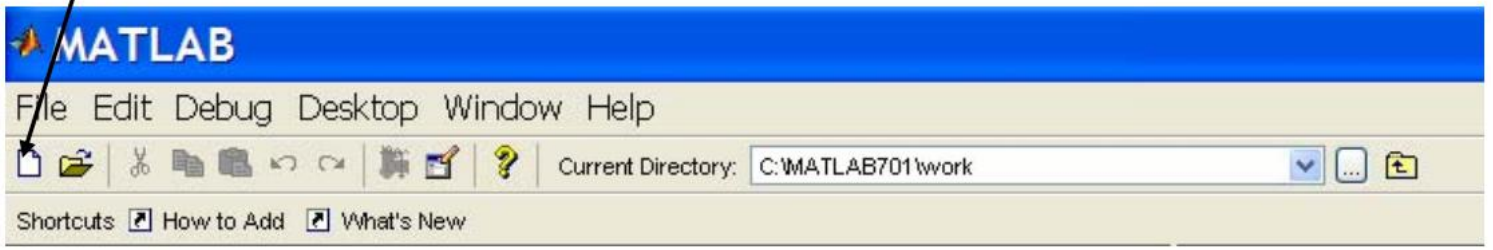
MATLAB script

- Scripts are
  - collection of commands executed in sequence
  - written in the MATLAB editor
  - saved as MATLAB files (.m extension)

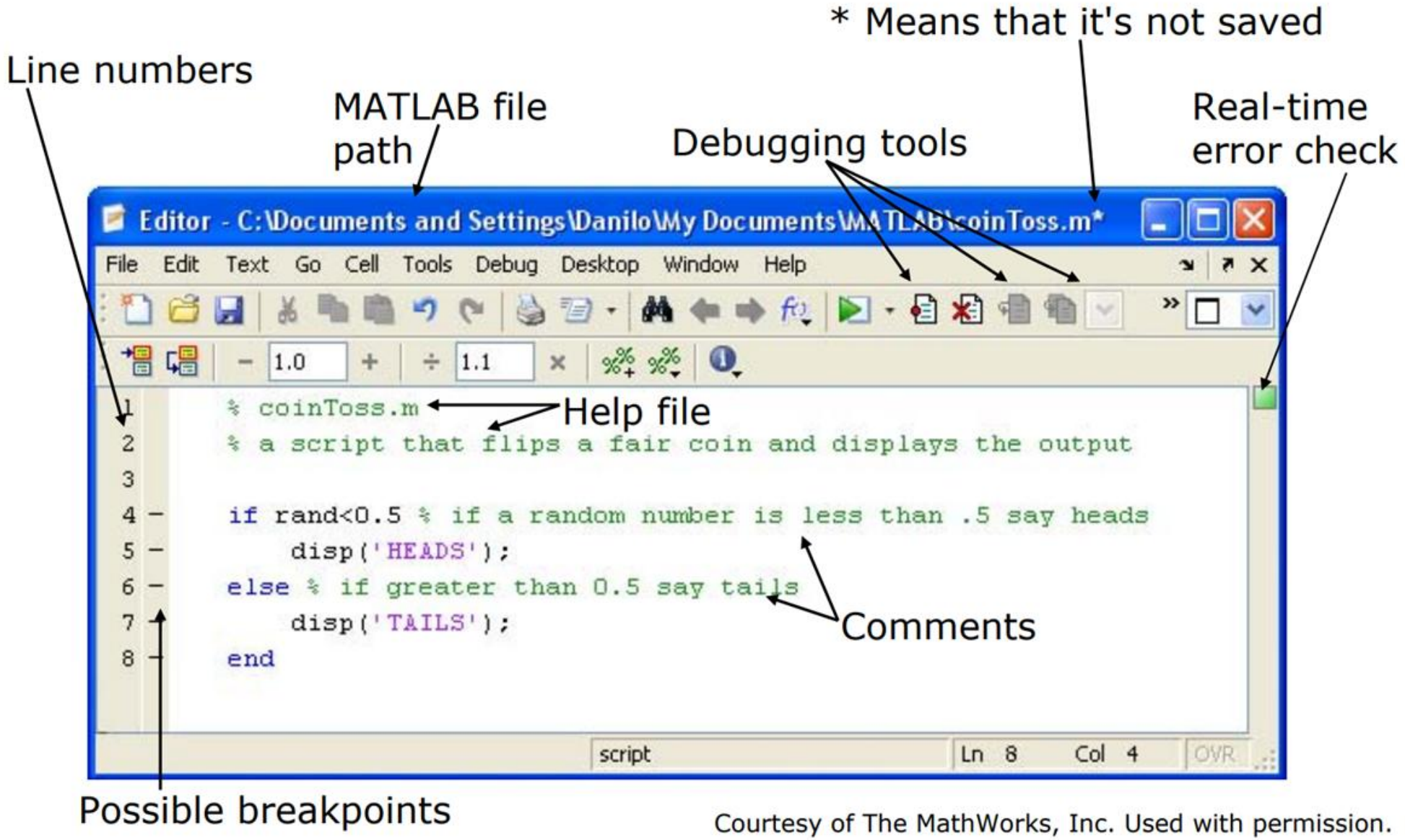
- To create an MATLAB file from command-line

```
» edit helloWorld.m
```

- or click



Script-editor



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-excercise

```
% 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:

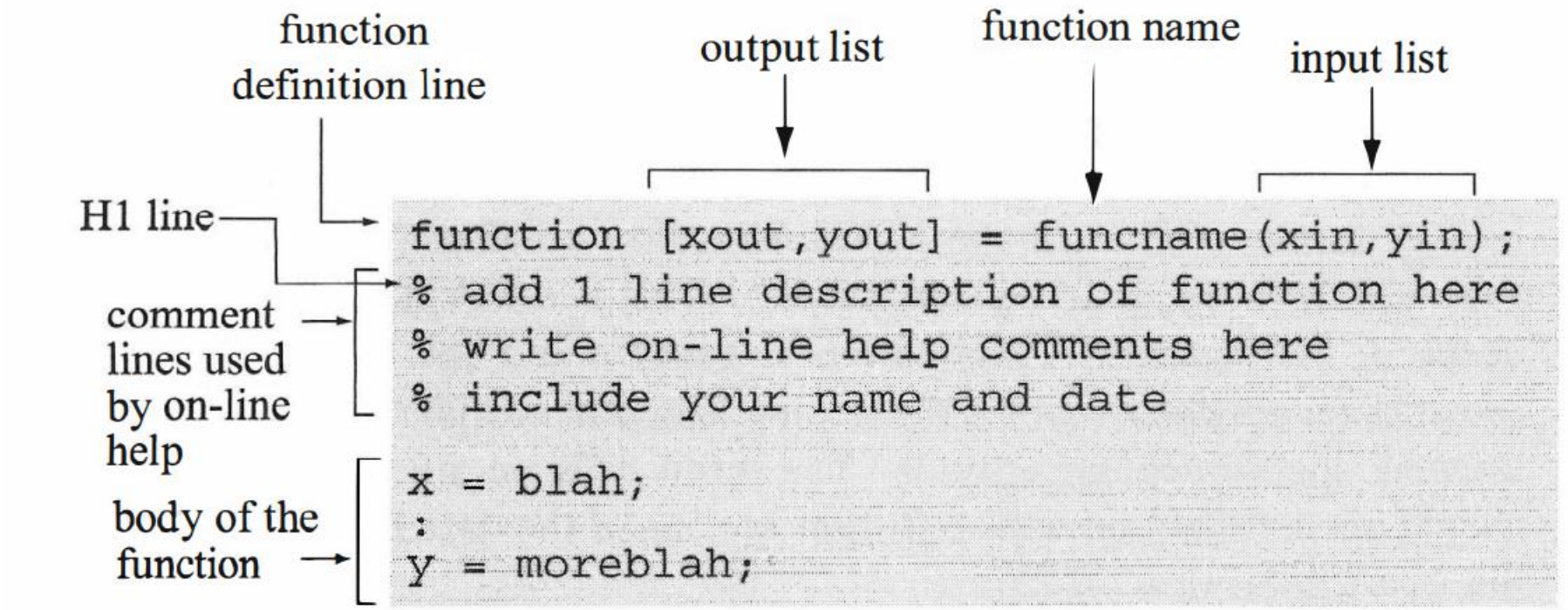
```
function [output variables] = function_name(input variables);
```

Examples of Function Files

Function Definition Line	*File Name*
`function [rho,H,F] = motion(x,y,t);`	motion.m
`function [theta] = angleTH(x,y);`	angleTH.m
`function theta = THETA(x,y,z);`	THETA.m
`function [] = circle(r);`	circle.m
`function circle(r);`	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]   =   mot ion (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*.

SVKM'S NMIMS Deemed to be UNIVERSITY | NAVI MUMBAI

MATLAB
Unit 5-Lecture 16

BTech (CSBS) -Semester VII

13 September 2022, 09:35AM

*few figures/content have been prepared/referred from internet source/books

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.

```
% exercise1script.m

x = 3

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

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

```
exercise1script
```

> 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>

% exercise1func.m

% input: x

% output: p, solved in terms of x

function p = exercise1func(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.

```
exercise1func(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:

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

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

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

Functions on Functions

Suppose a function y = demoFun(x) has been defined in a function M-file demoFun.m

`fplot(@demoFun,[a b])`	Plots the function for `a ≤ x ≤ b` without setting up arrays
`fzero(@demoFun,[a b])`	Finds one value of x for `demoFun(x) = 0` provided that the signs of `demoFun(a)` and `demoFun(b)` are opposite.
`fzero(@demoFun,c)`	Finds one value of x for `demoFun(x) = 0` by starting a search at `x = c`
`fminbnd(@demoFun,a,b)`	Finds the coordinates of a minimum point of `demoFun(x)` at the interval `a≤x≤b`
`quadl(@demoFun,a,b)`	Finds an accurate value for $\int_a^b y(x)\,dx$ Note: `quadl` 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 O 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.

SVKM'S NMIMS Deemed to be UNIVERSITY | NAVI MUMBAI

MATLAB
Unit 5-Lecture 17

BTech (CSBS) -Semester VII

16 Septemer 2022, 09:35AM

*few figures/content have been prepared/referred from internet source/books

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.

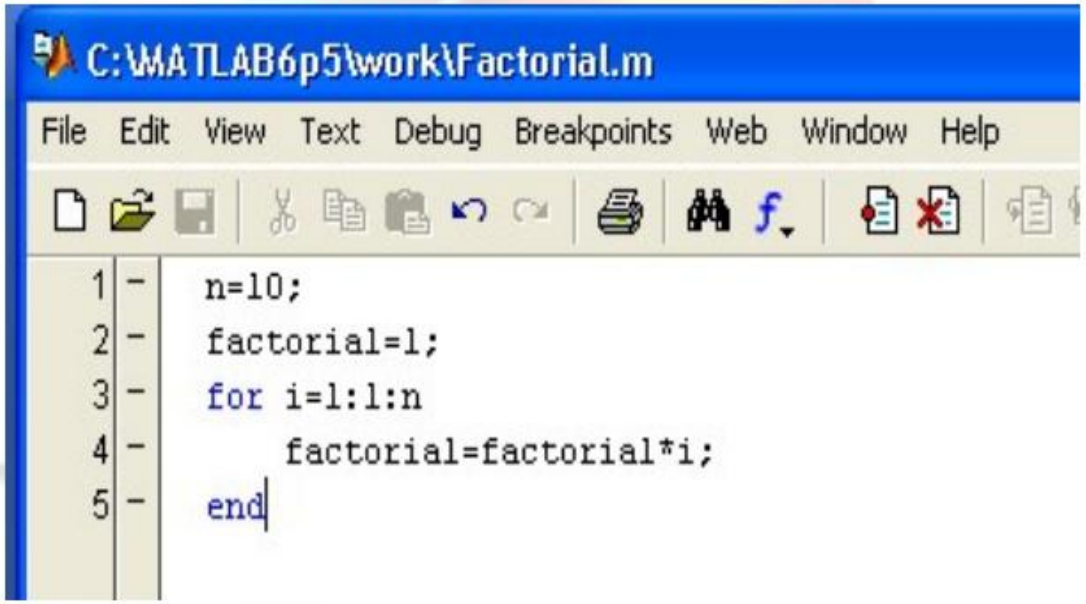
Kinds of M files

Script M-Files	Function M-Files
Do not accept input Do not accept input arguments or return arguments or return output arguments output arguments	Can Can accept accept input input arguments and return arguments and return output arguments 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

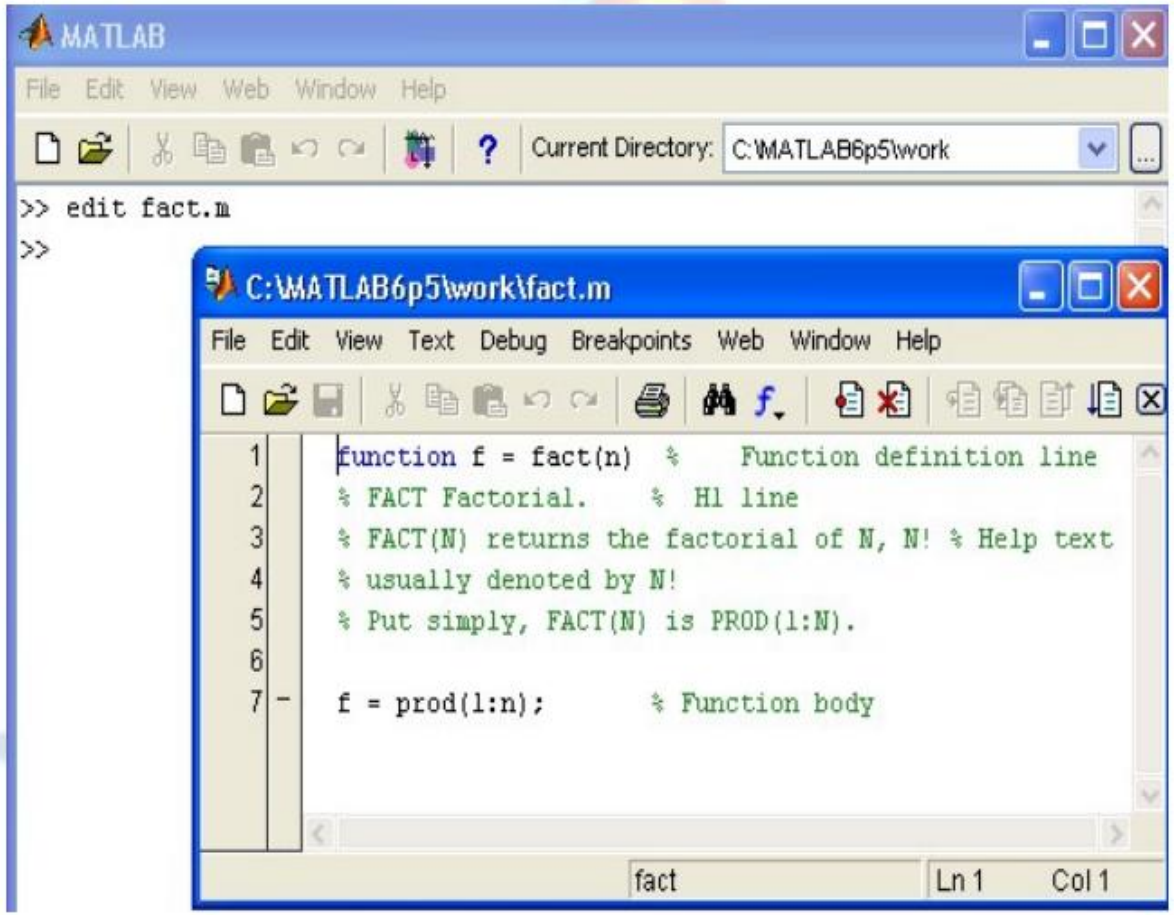
Factorial.m

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



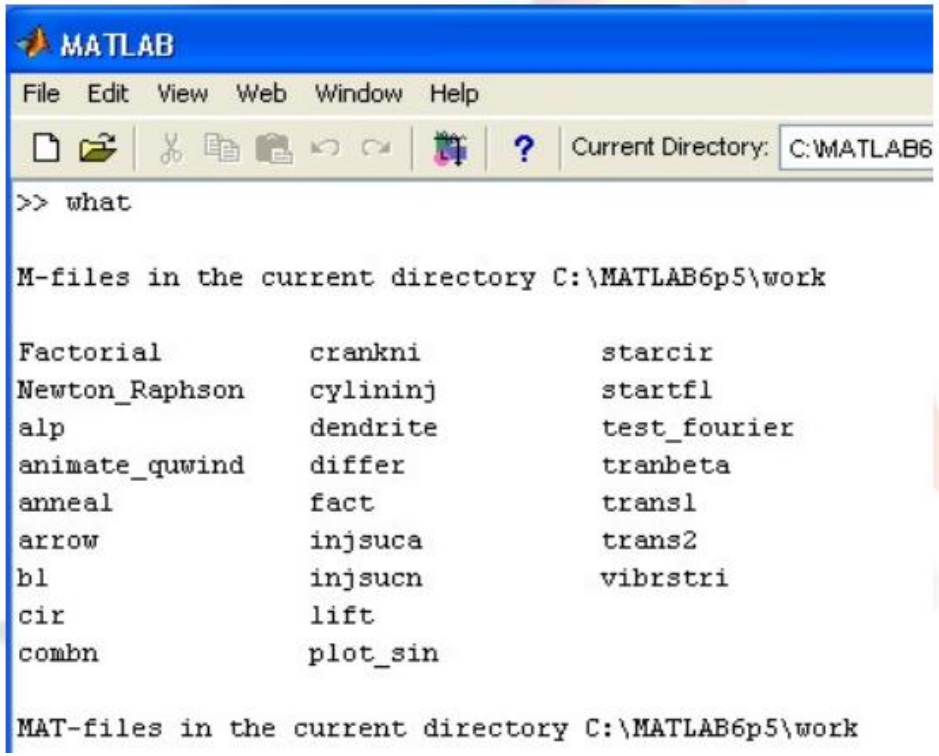
Accessing Text Editors

>> edit fact.m



Listing files

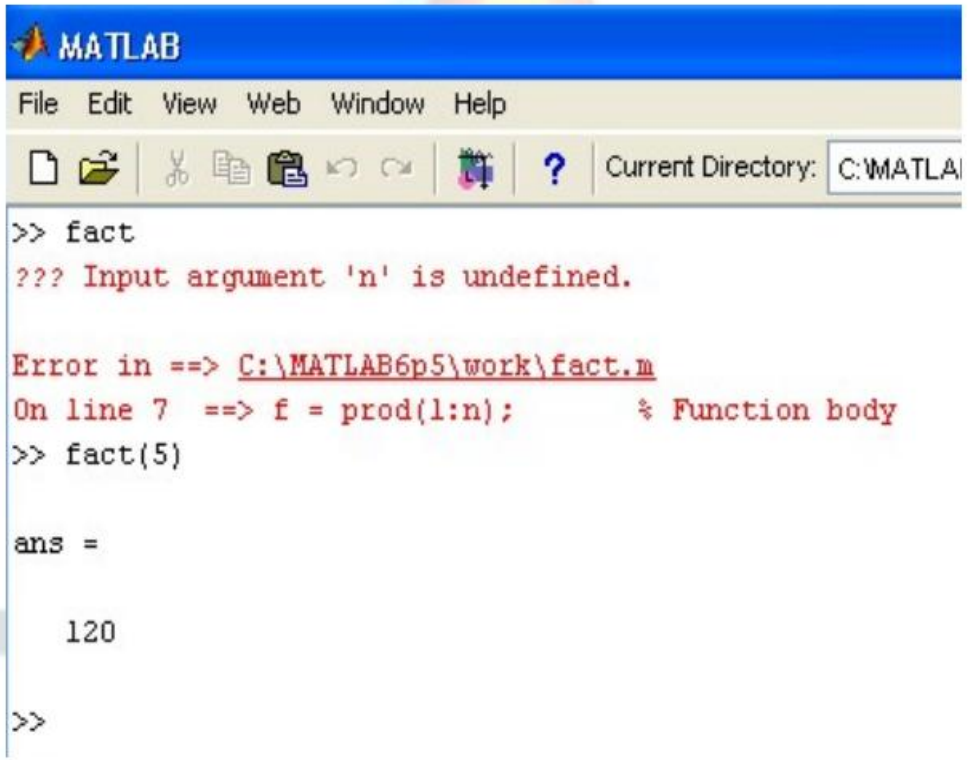
>> what (List the names of the files in your current directory)



Calling function

Call the fact function

>> fact(5)



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. ➢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 =
     8
```

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
```

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.

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 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.

SVKM'S NMIMS Deemed to be UNIVERSITY | NAVI MUMBAI

MATLAB
Unit 5-Lecture 18

BTech (CSBS) -Semester VII

20 Septemer 2022, 09:35AM

*few figures/content have been prepared/referred from internet source/books

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:

calcareaii.m

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

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

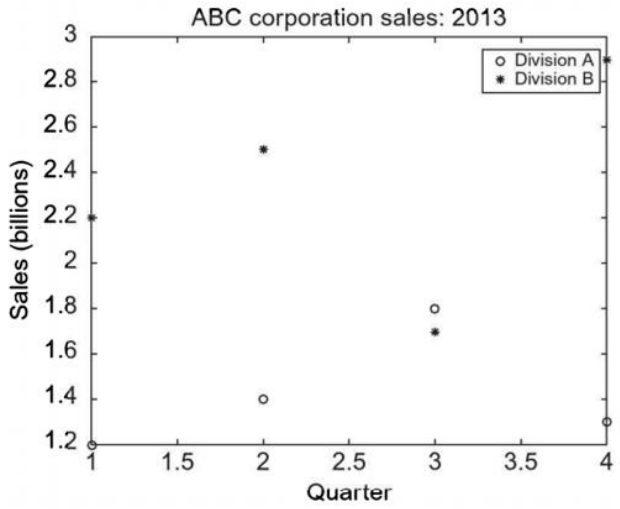
PRACTICE 3.8

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.
```