زانکۆی سەلاحەدین - هەولێر
Salahaddin University-Erbil

University of Salahaddin
College of Education
/Department of physics
Second Year

Chapter Two
Vectors and Matrices

Diyar A. Rasool

2019 – 2020

2.1.1 Vectors

- A **vector** is an ordered list of numbers.
- You can enter a vector of any length in MATLAB by typing a list of numbers, separated by **commas** and/or **spaces**, inside square brackets.
- For example: **>> z = [1,4,7,18]** **>> y = [4 -3 5 -2 8 1]**
- Vector of values running from 1 to 9:

>> x = 1:9 X = 1 2 3 4 5 6 7 8 9

- The increment can be specified as the **middle** of three argument:

>> x = 0:2:10 x = 0 2 4 6 8 10

- Increments can be **fractional or negative**, for example,

>> 0:0.1:1 or **100:-1:0**.

- **linspace(0,10,6)** ans = 0 2 4 6 8 10

2.1.1 Vectors

- The elements of the vector **x** can be **extracted** as **x(1)**, **x(2)**, etc. For example: >> x = 0: 7;

>> **x(3)** ans = 2 >> **x(4:7)** >> **x([4,7])**

- To **change** the vector **x** from a row vector to a column vector, put a **prime** (') after **x**: **>> x'**

>> x1=[5,3,1,23,11], min(x1), max(x1), mean(x1), sort(x1), sum(x1).

- You can perform **mathematical operations** on vectors.

for example, to **square** the elements of the vector **x**,

>> $x = 0: 2: 10$

>> x.^2 ans = 0 4 16 36 64 100;

- The **period(.)** in this expression says that the numbers in **x** should be squared individually, or *element-by-element*.

2.1.1 Vectors

- Typing **x^2** would tell MATLAB to use matrix multiplication to multiply **x** by itself and would produce an **error** message in this case.

- Similarly, you must type **.*** or **./** if you want to multiply or divide vectors element-by-element.

>> x.*y ans = 0 -6 20 -12 64 10

- Most MATLAB operations are, by **default**, performed element-by-element. **For example**, you do not type a period**(.)** before: **addition, subtraction** and **exp(x)** (the **matrix exponential** function is **expm**).

2.1.2 Matrices

- A **matrix** is a rectangular array of numbers.
- Row and column vectors are examples of matrices.
- **Example**: Write the following Matrix: $a = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 7 & 15 & 3 \\ 12 & 9 & 6 & 3 \end{pmatrix}$

Ans: $a = [1{:}\,4; 5,7,15,3; 12{:}\,{-3}{:}\,3];$

- **Note** that the matrix *elements* in any row are separated by **commas**, and the rows are separated by **semicolons**.
- The elements in a row can also be separated by **spaces**.

Example: **Extract**: >> **a(7), a(3,2), a(2,:), a(1:3,2:3), a([2 3],[1 3])**

- If two matrices **A** and **B** are the same size:

sum: A+B **add a scalar** (a single number): **A + c**

difference: A-B **subtracts: A - c**

2.1.2 Matrices

❖ Some commands:

>> zeros(1,3), ones(2,4), rand(3,5), randn(2,5), eye(n,m), det(A), size (A), Length (A).

>> Example: If v=[1, 2, 4, 5], w=[1;2;4;5] and A = [1,2,3;4,-5,6;5,-6,7], try to do the following steps:

>> v+2 >>B=A' >> A*B >>A+B >> B=A.'

>> A(2,3) >> A([2 3],[1 2]) >> B(:,2) = [ ]

>> B=A([3 2],[2 1])

>> B=[A(3,2);A(3,1);A(2,2);A(2,1)]

2.1.2 Matrices

Example: If A = [1,2,3;4,-5,6;5,-6,7]. Find the following:

>> A(1,:) >> A(1:2,:) >> A([1 2],:).

Transpose	B = A'
Identity Matrix	eye(n) ➔ returns an n x n identity matrix eye(m,n) ➔ returns an m x n matrix with ones on the main diagonal and zeros elsewhere.
Addition and subtraction	C = A + B C = A – B
Scalar Multiplication	B = αA, where α is a scalar.
Matrix Multiplication	C = A*B
Matrix Inverse	B = inv(A), A must be a square matrix in this case. rank (A) ➔ returns the rank of the matrix A.
Matrix Powers	B = A.^2 ➔ squares each element in the matrix C = A * A ➔ computes A*A, and A must be a square matrix.
Determinant	det (A), and A must be a square matrix.

Examples

1. If a = [1 2 3 4 5; 2 3 4 5 6; 3 4 5 6 7; 4 5 6 7 8]. Find:

a(2,5) = 6. a(:,4) = 4 5 6 7 8. a(:, 2:3) =

2	3
3	4
4	5
5	6

a(2:3,2:4) =

3	4	5
4	5	6

2. From $f = x^2 e^{-ax} \sin(\pi x)$. Find f, from $x = 1$ to $x = 10$ by typing a $= 1$.

Ans:

>> x=linspace(1,10,10); >> a=1; >> $f = x.^{\wedge}2 .* \exp(-a * x) . * \sin(\text{pi} * \text{x})$;

2.2 Matlab Functions

- In MATLAB you will use both **built-in functions** and **functions** that you create yourself.

Built-in Functions

- **MATLAB has many built-in functions:**

1. These include: **sqrt**, **cos**, **sin**, **tan**, **log**, **exp** and **atan** (for arctan).

2. Specialized mathematical functions like: **gamma, erf,** and **besselj**.

3. MATLAB also has several **built-in constants**, including: **pi** (the number π), **i** (the complex number i $=\sqrt{-1}$), and **Inf** (∞).

2.2 Matlab Functions

- The following table lists some commonly used functions, where variables x and y can be numbers, vectors, or matrices.

`cos(x)`	Cosine	`abs(x)`	Absolute value
`sin(x)`	Sine	`sign(x)`	Signum function
`tan(x)`	Tangent	`max(x)`	Maximum value
`acos(x)`	Arc cosine	`min(x)`	Minimum value
`asin(x)`	Arc sine	`ceil(x)`	Round towards $+\infty$
`atan(x)`	Arc tangent	`floor(x)`	Round towards $-\infty$
`exp(x)`	Exponential	`round(x)`	Round to nearest integer
`sqrt(x)`	Square root	`rem(x)`	Remainder after division
`log(x)`	Natural logarithm	`angle(x)`	Phase angle
`log10(x)`	Common logarithm	`conj(x)`	Complex conjugate

- There are also some constants which are listed here:

`pi`	The π number, $\pi = 3.14159\ldots$
`i,j`	The imaginary unit i, $\sqrt{-1}$
`Inf`	The infinity, ∞
`NaN`	Not a number

2.2 Matlab Functions

❖ Functions can be written in two different ways:

```
>> g = @(x, y) x^2 + y^2; g(1, 2) ;    ans =  5
>> g1 = inline('x^2 + y^2', 'x', 'y'); g1(1, 2);     ans =  5;
```

Example: Use built in functions at the points (1:5,*2*) and (1:5,2:6) for the function of $g = x^2 + y^2$.

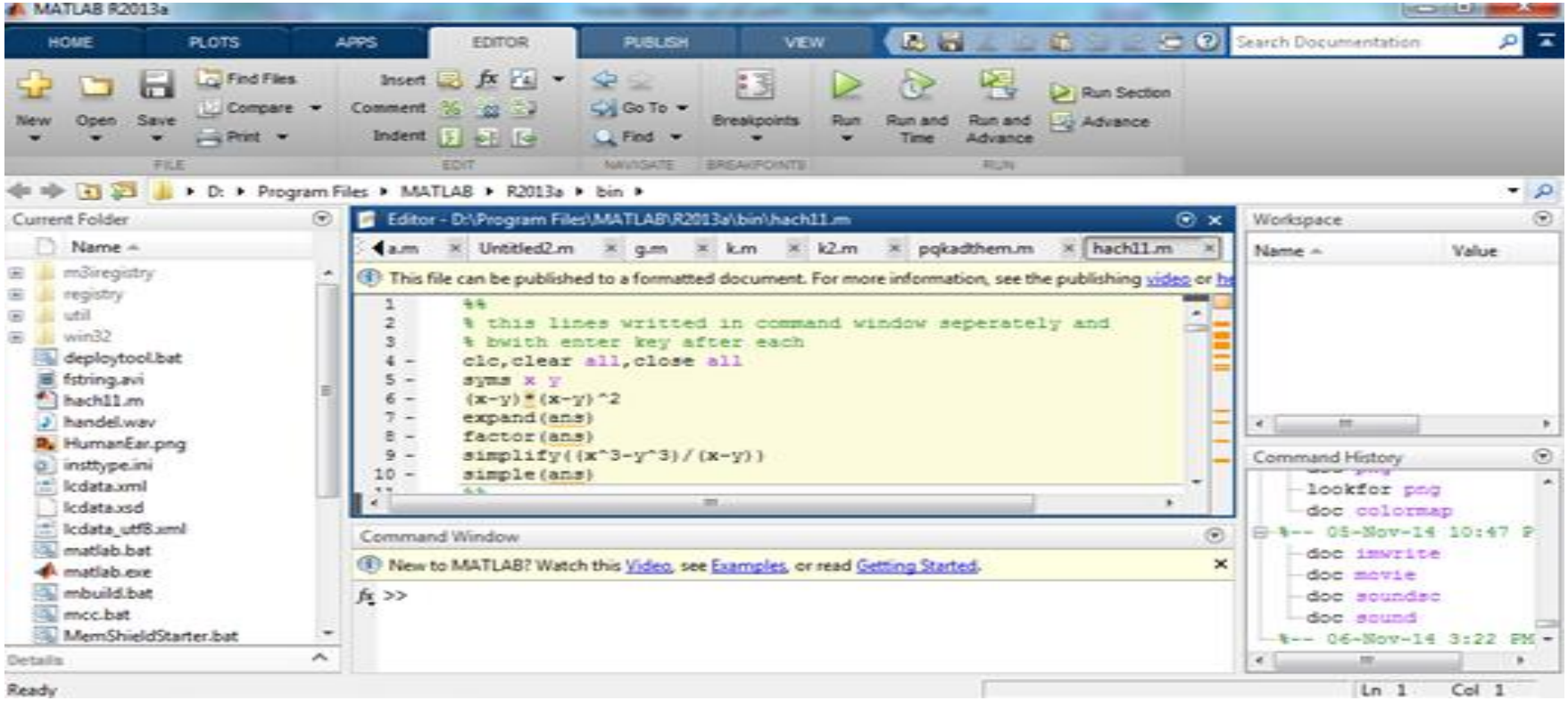
Ans:

```
>> g = @(x, y) x.^2 + y^2;   g(1:5, 2) ;
>>  g1 = inline('x.^2 + y.^2', 'x', 'y');      g1(1:5, 2:6) ;
```

```
>> f = inline('log(a*x)/(1+y^2)')
>> f = inline('log(a*x)/(1+y^2)','x','y','a')
>> f = inline('exp(-x)/(1+x^2)');
```

2.3 The MATLAB Interface The Desktop

- Windows are currently **visible** with the **Home Tab-Layout** at the top of the Desktop or changing position of windows.



- You can adjust the **size** of the windows by dragging their edges with the mouse.

2.3 The MATLAB Interface The Desktop

- The MATLAB **Toolstrip** includes **6 Tabs(Home, Plots, Apps, …)** and a **Quick Access *toolbar***; they contains icons that give quick access to some of the items.

- Some icons of **Toolstrip** also have **keyboard shortcuts**. You may find it useful to items you frequently use.

- Several of the windows contains **buttons** in the upper right-hand corner. The curved arrow will "**undock**" the window from the Desktop by click on  (you can return it to the Desktop by selecting **Dock** from the **same icon**).

Workspace

- The complete collection of defined variables(with their sizes) is referred to as the **Workspace**.

- If you double click on a variable, its contents will appear in a new window called the ***Variables***, which you can use to edit individual entries in a vector or matrix.

- If you need to **interrupt** a session and don't want to be forced to **re-compute** everything later, then you can **save** the current Workspace with **save workspace icon(in Home tab), or** typing **save myfile** (myfile.mat).

Save and Load a file

- To save **only** the values of the variables **x** and **y**, type:

  >> **save myfile x y**

- When you start a new session and want to **recover** the values of those variables, use **load**.

For example, typing **load myfile** restores the values of all the variables stored in the file myfile.mat:

Example:

```
savefile = 'pqfile.mat';
p = rand(1, 10);q = ones(10);
save(savefile, 'p', 'q')
load pqfile.mat
```

The Current Folder

❖ To **change** the **current folder** to folder of your project:

a. select a folder you have **used before** by clicking on the arrow at the right of the box(▼).

b. browse for a folder by clicking on the "Browse for folder" icon to the right of the box.

❖ You can create a **new folder**, say ProjectA, within it by:

1. >>**mkdir ProjectA**.

2. You can also **right-click** in the Current folder Browser and select **New Folder**.

3. click on the "**Actions**" icon  in the browser's toolbar and then chose new folder, wright ProjectA.

M – Files

❖ **M-files** are ordinary text files containing MATLAB commands. You can create and modify them using any text editor or word processor that is capable of saving files.

➢ **You can start *Editor/ Debugger and run* by:**

1. >> **edit** % to edit a new file **or** followed by the name of an existing M-file in the current folder.
2. **Home Tab**: **New script** or **New** icons, and **Open** icons.
3. Double-clicking on an M-file in the Current Folder Browser to open an existing M-file.
4. M-files can be **saved** in a file and then **run** them with a **single command**(its name without .m), mouse click on **run icon** in Editor tab or **F5**.

Types of M – files

1. Script M – file. **2. Function M – file.**

1. Script M – file: Let's start the scripting process. First, make a new script file in by clicking on the "New" command on the tool bar.

- Save this file as "test.m" (Matlab will automatically add the .m extension) in the directory where you want to store your Matlab scripts.
- After you've created the script file, enter the sample command *in the editor window, and save it*.

Example: Type **x = [0.1, 0.01, 0.001];** **y = sin(x)./x;**

- save as **task1.m** . The ".m" suffix is **mandatory**.
- The **output** will be displayed in the Command Window.

- **Note** that adding **comments** in M-files is explain what is being done in the calculation, or might interpret the results of the calculation. The percent sign **(%)** begins a comment; the rest of the line is not executed. (comments color is **green** to help distinguish them from commands, which appear in **black**.).

- Here is our new version of task1.m with a few comments added:

```
format long                    % turn on 15 digit display

 x = [0.1, 0.01, 0.001]; x     % These values illustrate the fact that the
limit of

y = sin(x)./x      % sin(x)/x as x approaches 0 is 1.
```

Cell Mode

❖ One can divide a script M-file into subunits called **cells**. This is especially **useful** if:

1. Your M-file is **long.**
2. if you are going to ***publish*** it
3. It can be a big help if you've made a **change** in just one cell and do not want to run the **whole** script all over again.

❖To start a new cell:

1. Insert a comment line "**title**" of the cell that follows starting with a double percent sign **%%** followed by a space or click on **Editor Tab\Insert** icon.
2. When you click somewhere in the M-file, the cell that contains that location will be **highlighted** in **pale yellow**.
3. You can evaluate that cell by: **Editor Tab\Run Section icon, Advance icon, Ctrl+Enter, or right click\ Evaluate Current Cell**.

Initializing Script M-files

- **In order for the results of a script M-file to be reproducible:**

1. The script should be **self-contained**(contained all needed)**.**

2.Unaffected by other variables that you might have defined elsewhere in the MATLAB session(**clear all**).

3. **Uncorrupted** by leftover graphics(close all, figures in the end of code).

Initializing Script M-files

- If you want the commands to be **displayed** along with the results, add the command **echo on** to the beginning of script and add **echo off** to the end of the script.

```
clear all   % remove old variable definitions
echo on   % display the input in the command window
format long    % turn on 15 digit display
x = [0.1, 0.01, 0.001];    % define the x values
y = sin(x)./x    % compute the desired quotients
 % These values illustrate the fact that the limit of
 % sin(x)/x as x approaches 0 is equal to 1.
   echo off
```

Example

- Create a script file, and type the following code:

```
a = 5;  b = 7;   c = a + b;

 d = c + sin(b) ;    e = 5 * d;

f = exp(-d)
```

2. Function M - File

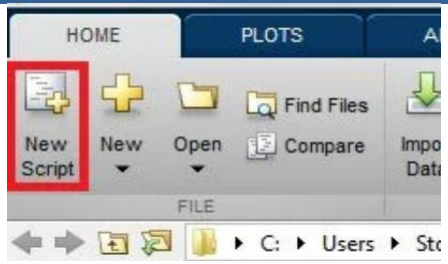
- If the **file** contains only **function definitions**, the first **function** is the main **function**, and is the **function** that **MATLAB** associates with the **file** name.

- **Functions** that follow the main **function** or script code are called local **functions**.

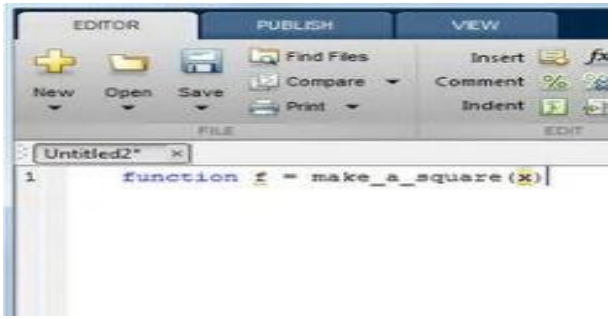
- Like a script M-file, a **function M-file** is a plain text file that should **reside** in your current directory or elsewhere in your MATLAB path.

Steps to do Function M – File

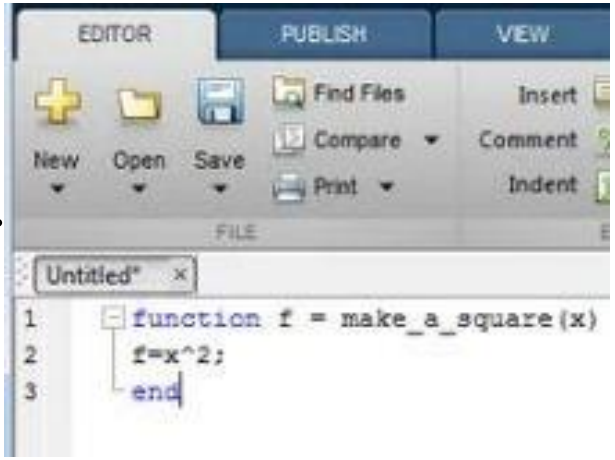
1. Open two script windows:



2. From a script window type function f = make_a_square(x)" into line1. The word "function" tells MATLAB that this script will be a function.

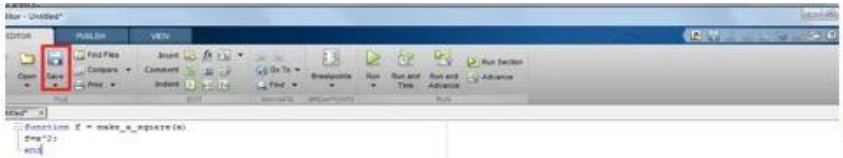


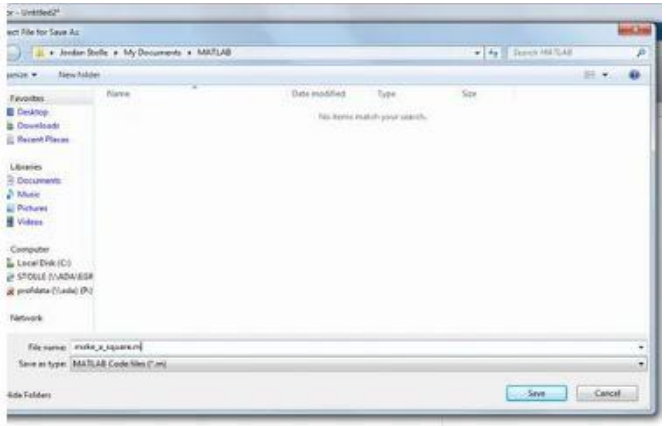
3. **Finishing the Function:** complete the code by entering "f=x^2;" on line 2 and "end" on line3. Typing end at the end of the function.



Steps to do Function M – File

4. Saving the function: Once your function is complete, save the function using the save button. The default name of the save file will be the same as the name of the function. **Make sure that you do not change this.**





5. From the second script type the value of x and at the end type f = make_a_square(x) in which you have been written at the beginning.

Examples

1. Use function M file for the following:

❖ **From first script type:**

```
function f=make_a_square(x)
f=x^2;
End
```

❖ **From the second script type:**

```
clc, clear all, close all
x=10;
f=make_a_square(x),      ans:
f= 100
```

2. Suppose that you want to find the smallest value of b for which $\sin(10^{-b})/(10^{-b})$ and 1 agree to 15 digits

```
function y=sinelimit(x)
y=sin(x)./x;
end
```

```
clc, clear all,close all
x=[0.1,0.01,0.001];
y=sinelimit(x)
```

2.4 Managing Variables

❖ We have four different types of MATLAB data:

1. Floating-point numbers (double array, scalar)
2. Strings(char array)
3. Symbolic expressions(sym object)
4. Functions(function handle array, inline object)

- The "**Bytes**" column shows how much computer memory is allocated to each variable.

```
>> whos
  Name        Size                Bytes  Class                Attributes

  a           1x1                     8  double
  ans         1x1                    60  sym
  b           1x2                     4  char
  c           1x1                    60  sym
  f           1x1                    16  function_handle
  f1          1x1                   822  inline
  m           1x9                    72  double
```

2.4 Managing Variables

- In a long MATLAB session it may be hard to remember the names and types of all the variables you have defined.

- Type **whos** to see a summary of the names and types, of your currently defined variables.

Example: Assign **a=pi**, **b='pi'**, **c=sym('pi')** and **x=1:9** type **whos a b c x.**

- Generally clear variables before starting a new calculation (**clear** or **clear all, clear x y)**. Otherwise values from a previous calculation can creep into the new calculation by accident.

- Even the **Workspace Browser** presents a graphical alternative to **whos** parameters that can be drawn.

2.5 Solving Equations

- You can solve algebraic equations, differential equations and solve equations involving variables with **solve** or **fzero**.
- The command **solve** can solve:

1. Higher-degree polynomial equations.
2. Equations involving more than one variable

- The input to solve can be **string** or **symbolic** expression
- Solve **algebraic equations** to get either exact analytic solutions or high-precision numeric solutions.
- For **analytic solutions**, use **solve**, for **numerical solutions**, use **vpasolve** and for **differential equations** use **dsolve**

Examples about analytic solutions

1. Solve the following equations:

>> solve('x^2 - 2*x - 4 = 0'); ans = 5^(1/2)+1 1-5^(1/2).

- The answer is of the exact (symbolic) solutions.
- To get numerical solutions, type **double (ans)**, or **vpa(ans)** to display more digits.

---

2. Solve $x^2 - 3x = -7$; >> syms x >> solve(x^2 - 3*x + 7)

ans=
```
3/2+1/2*i*19^(1/2)
3/2-1/2*i*19^(1/2)
```

- To get numerical solutions, type **double(ans)**, or **vpa(ans).**

---

3. Solve ($2x - \ln y = 1$ for y in terms of x),

type: **>> solve('2*x - log(y) – 1= 0', 'y')**

Examples

4. Solve these two equations: $x^2 - y = 2,\ y - 2x = 5$

Ans: >> **[x, y] = solve('x^2 - y = 2', 'y - 2*x = 5');**

```
x =                 y =
 1+2*2^(1/2)         7+4*2^(1/2)
 1-2*2^(1/2)         7-4*2^(1/2)
```

❖ You can **extract** first x and y values by typing:

>> **x(1)** ans = 1+2*2^(1/2) >> **y(1)** ans = 7+4*2^(1/2)

❖ Some equations **cannot** be solved symbolically, and in these cases **solve** tries to find a numerical answer.

5. >> solve('sin(x) = 2 - x')

Examples about Differential Equation

1. **Solve DE:** Solve this differential equation $\frac{dy}{dt} = ay$.

Ans: clc,clear all,close all

>> syms a y(t); >> eqn=diff(y,t)==a*y; >> dsolve(eqn) >> C3*exp(a*t).

2. Solve Higher-Order Differential Equation: Solve the equation $\frac{d^2y}{dt^2} = ay$.

Ans: >>syms a y(t); >> eqn=diff(y,t,2)==a*y; >> dsolve(eqn)

Ans: C3*exp(a^(1/2)*t) + C4*exp(-a^(1/2)*t).

3. Homework: solve $\frac{d^2y}{dt^2} = a^2y$.

Ans: C3*exp(-a*t) + C4*exp(a*t).

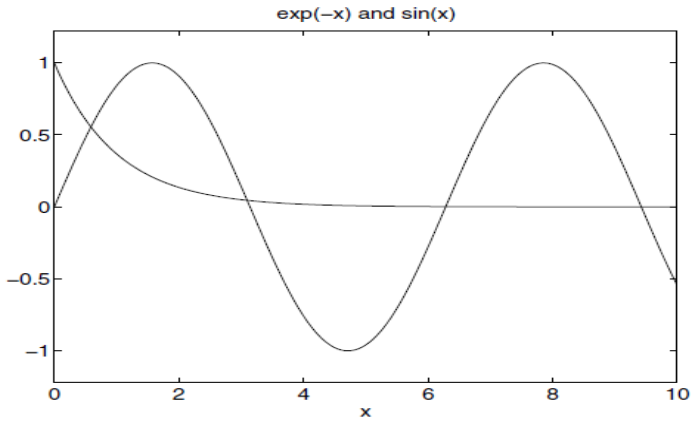
Fzero command

- The **fzero** command which looks for a zero of a given function ($e^{-x} - \sin(x) = 0$) near a specified value of x.

- To find an approximate solution near $x = 0.5$:

- In fact, the graphs of $\exp(-x)$ and $\sin(x)$ have intersections, each one of them represents a solution of the equation $e^{-x} = \sin(x)$.

Example: **>> h = @(x) exp(-x) - sin(x);** **>> fzero(h, 0.5)** ans= 0.5885



- Replace **0.5** with **3** to find the next solution, and so forth.