# CHE656 Course Slides

(9th Edition, 2016)

# Modeling in Chemical Engineering with MATLAB

Dr. Hong-ming Ku
Chemical Engineering Practice School
King Mongkut's University of Technology Thonburi



# Introduction

# What Is MATLAB?

- MATLAB = Matrix Laboratory
  - by The MathWorks, Inc. (www.mathworks.com)
- Originally developed for easy matrix manipulation
- Latest: Version R2016a (Version 9.0)
- Ours: Version R2012b
- Software program for numerical computations
  - Simple arithmetic and function calculations
  - Vectors and matrix manipulations



# What Is MATLAB? (Cont'd)

- Equations solving
  1. Linear algebraic equations
  2. Nonlinear algebraic equations
  3. Ordinary differential equations (ODEs)
  4. Partial differential equations (PDEs)
- Programming
- Plotting

# Getting Started

- On PCs, click on the MATLAB icon in Desktop

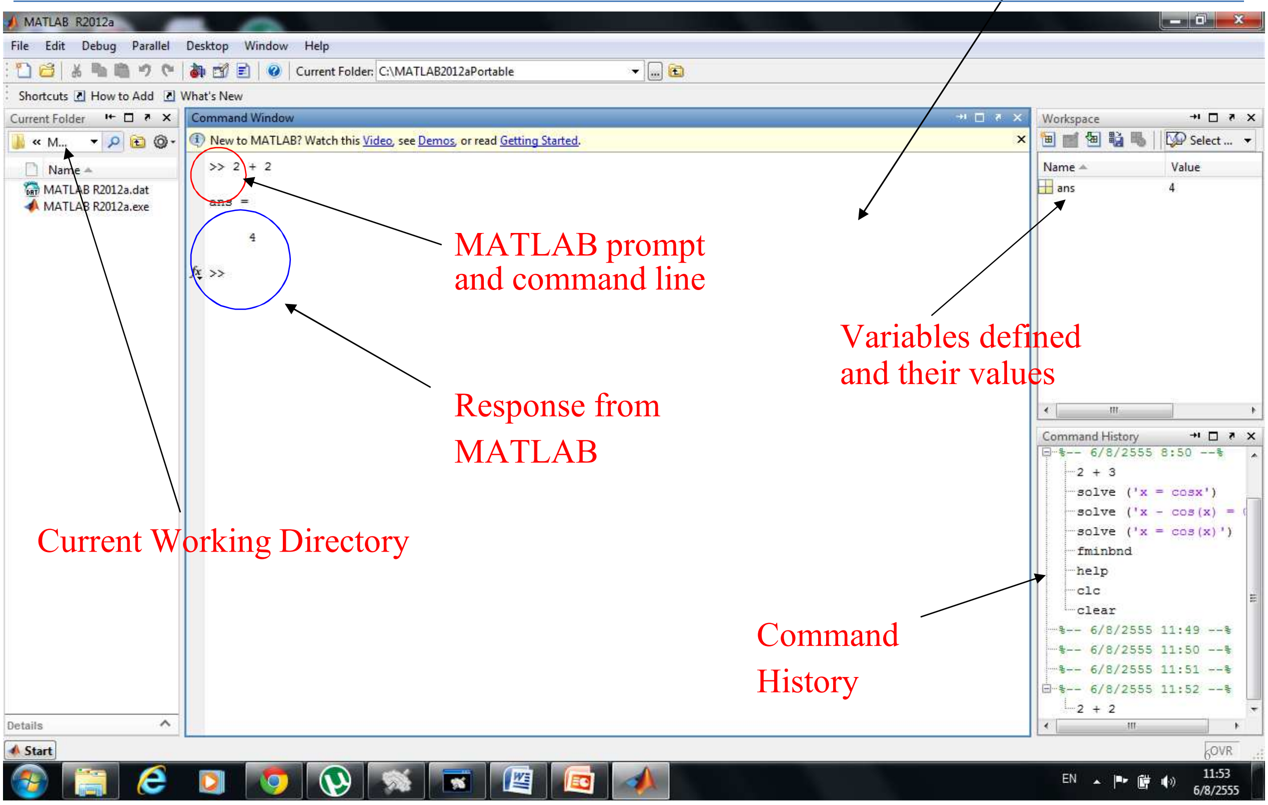
- Terminating a MATLAB session:

  1. Click on the "Close Window" button
  2. Select Exit MATLAB from the File pulldown menu
  3. Press Cntrl+Q on the command line
  4. Type exit or quit at the command line
  5. Cntrl+C will interrupt a MATLAB task but will not exit the program



# Getting Started (Cont'd)

Command Window



# Getting Help in MATLAB

- Very extensive set of help at the command line:

| | |
|---|---|
| demo | Opens Help browser to MATLAB examples |
| help *topic* | Display on-line help on a topic (with syntax and examples) at command line<br>Type help to view all topics |
| doc<br>helpwin<br>help | Online help and comprehensive hypertext documentation and trouble-shooting |



# Lookfor vs. Help

- The lookfor command searches for functions based on a keyword in the first line of help text
- For example, MATLAB does not have a function named "inverse":

  >> help inverse

  inverse.m not found. => response from MATLAB

  >> lookfor inverse => will find many matches

# Simple Arithmetic Capabilities

| | |
|---|---|
| >> clc | % Clear the screen |
| >> clear | % Clear all the variables in session |
| >> 2 + 3 | % Simple addition |

ans =

5

| | |
|---|---|
| >> 2*3 | % Simple multiplication |

ans =

6



# Simple Arithmetic Capabilities (Cont'd)

| | |
|---|---|
| >> 3 / 6 | % Simple division |
| >> 2^3 | % Exponentiation of power |
| >> 10 / (3+2) | % More complex expression |

ans =

2

Arithmetic Operators:

| | | | |
|---|---|---|---|
| + | Addition | / | Division |
| - | Subtraction | \ | Left division |
| * | Multiplication | ^ | Power |
| ( ) | Specify evaluation order by the degree of nesting | | |

# Other Tidbits

```
>> 6 / 3 , 3 \ 6          % Use , to execute more than 1 operation
ans =
     2
ans =
     2
```

The semicolon ; will suppress the output but save the result

```
>> 2+3 ;    %Will produce no output but save the result in ans
>> ans      % Retrieve the result
ans =
     5
```



# Other Tidbits (Cont'd)

- Use up-arrow to recall previously entered commands ↑
- A statement can be continued onto the next line with 3 or more periods followed by a return

```
>> 2 + 3 …        % Use 3 periods to continue the next line
+ 10
ans =
      15
```

# Input and Output Format for Numbers

- All computations in MATLAB are done in double precision (16 digits)
- Uses conventional decimal notation
- Scientific notation uses the letter *e* to specify a power-of-ten scale factor
- Imaginary numbers use either *i* or *j* as a suffix



# Input and Output Format for Numbers

- Examples of legal numbers are:

| | | |
|---|---|---|
| 3 | -99 | 0.0001 |
| 9.6397238 | 1.60210e-20 | 6.02252e25 |
| 1i | -3.14159j | 3e5i |

- Format command is used to switch between different display formats.

# Display Output for Numbers with Format

| | |
|---|---|
| >> format | % Default. Same as “format short” |
| >> format short | % Scaled fixed point format with 5 digits |
| >> format long | % Scaled fixed point format with 15 digits |
| >> format short e<br>format shorte | % Floating point format with 5 digits |
| >> format long e<br>format longe | % Floating point format with 15 digits |
| >> format shorteng | % Engineering format that has at least 5 digits and a power that is a multiple of three |
| >> format longeng | % Engineering format that has exactly 16 significant digits and a power that is a multiple of three. |



# Display Output for Numbers with Format

```
>> pi                      % Display value of pi using default format
ans =
      3.1416

>> format long, pi         % Long, fixed format pi
ans =
      3.14159265358979
>> format shorte, pi       % Short, scientific notation for pi
ans =
     3.1416e+00
```

Use fprintf command to write formatted data to file or screen

Syntax: fprintf (fid, format, A, ……)

# The fprintf Command

Syntax: fprintf(fid, format, A, ……)

where fid = output filename; if blank, output is screen
format = format control of data
A = variable name (e.g. vector, matrix, etc.)

```
>> A = pi;
>> fprintf('%10.6f', A)      % print value of pi in fixed point
                             % format with a maximum of 10
                             % characters and 6 decimal places

3.141593
```



# The fprintf Command (Cont'd)

```
>> A = pi;, B =2*pi;
>> fprintf('%10.6f', A, B)
3.141593  6.283185

>> fprintf('%10.6f\n', A, B)        % \n forces a new line in output
3.141593
6.283185
```

Type 'help fprintf' to view more information about the the command and how to write to an output file.
Another useful command to display output is disp(x), where x could be an array or a string enclosed in ' '. The command displays the array without printing the array name.

# Predefined Variables

| | |
|---|---|
| ans | The most recent answer |
| i, j | Imaginary unit |
| pi | The value of *pi* (3.141592653) |
| Inf | Infinity |
| NaN | Not-a-Number (i.e. 0/0 or Infinity/Infinity) |



# Built-in Mathematical Functions

- MATLAB has many built-in mathematical functions
- Type "help elfun" and "help specfun" for a list of functions
- Some common ones are:

| | |
|---|---|
| abs(x) | Gives the absolute value of x |
| sqrt(x) | Gives the square root of x |
| exp(x) | Exponential of x |
| log(x) | Natural logarithm of x |
| log10(x) | Logarithm to the base 10 of x |

# Built-in Mathematical Functions (Cont'd)

| | |
|---|---|
| sin(x) | Sine of x, for x in radians |
| asin(x) | Arcsin(x) |
| csc(x) | Produces 1/sin(x) |
| round(x) | Gives the integer closest to x |
| real(x) | Gives the real part of a complex number |

```
>> x = exp(1)              % Numerical value of e
x =
   2.7183
```



# An Example

```
%
% Here is a simple sequence of expressions to compute
% the volume of a cylinder, given its radius and length.
%
>> radius = 2;                       % radius of cylinder
>> length = 4;                       % length of cylinder
>> volume = pi*radius^2*length       % volume of cylinder
volume =
   50.2655
```

# Writing a MATLAB Script File

- A script is an external text file containing a sequence of MATLAB statements.
  - Has the file extension .m
  - Very useful for running MATLAB non-interactively by executing many MATLAB statements with one Enter keystroke by typing the script filename.
  - The first character of the file name must be an alphabet, but the file name may contain numerals.
  - Must make sure the file name does not coincide with built-in MATLAB function names, e.g. sum, sin, mean.

# Writing a MATLAB Script File (Cont'd)

- Two simple ways to create a MATLAB script file:
  1. Use a text editor in Windows or use the built-in Editor in MATLAB by choosing New Script in the ribbon.
  2. Use MATLAB *diary* command to record an interactive session.

     >> diary filename

     >> (some MATLAB commands)

     >> (some MATLAB output)

     >> diary off

     Then edit the file to delete MATLAB output, including incorrect commands and any error messages. Save the file again with the extension .m.

# Example of a Script File

- Create a script file named “Volume.m”

```
clear
clc
radius = 2;
length = 4;
volume = pi*radius^2*length;
fprintf (‘The volume of the cylinder = %4.2f \n’, volume)
```

- Notice that the file name of a script is case-sensitive.

# Vector and Matrix Manipulations

# Matrices and Vectors

## Vectors and One-Dimensional Arrays

1. Row Vector

```
>> a = [1 3 9 25 1]        % Syntax for a row vector with
                           %  elements separated by a space
>> a = [1, 3, 9, 25, 1]    % Syntax for a row vector with
                           % elements separated by a comma
a  =
      1     3     9     25     1
```



# Matrices and Vectors (Cont'd)

2. Column Vector

```
>> b = [1; 3; 2; 5]    % Syntax for a column vector with
                       % elements separated by a semicolon
b =
      1
      3
      2
      5
```

# Some Vector Operations/Manipulations

>> a(2)     % Determine the value of the 2nd element of the vector

ans =

3

>> length(a)     % Determine the number of elements in vector

ans =

5

>> a(7) = 49     % Add an additional element to the vector a

a =

1 3 9 25 1 0 49



# Some Vector Operations/Manipulations

>> a(6) = 16   % Change the 6th element of the vector

a =

1 3 9 25 1 16 49

Many of the functions introduced can be applied to a vector

>> sqrt(a)     % Determine the square root of each element

ans =

1.0000 1.7321 3.0000 5.0000 1.0000
4.0000 7.0000

Other useful functions are:

min(a), max(a), mean(a), median(a)

# Some Vector Operations/Manipulations

```
>> c = [2 4 5 3]'     % c is the transpose of the row vector
c =
    2
    4
    5
    3
>> 3*b – c     % array operations can be performed on each element
ans =
     1
     5
     1
    12
```



# Some Vector Operations/Manipulations

Arrays can be combined

```
>> [c ; b]          % Join two column vectors to form a new one
ans =
     2
     4
     5
     3
     1
     3
     2
     5
```

# Some Vector Operations/Manipulations

When division, exponentiation, or other operators are involved, the syntax is to put a period '.' before the operator without any spacing:

```
>> a./2                    % Divide each array element by 2
ans =
      0.5000    1.5000    4.5000    12.5000    0.5000
8.0000    24.5000

>> b'.*c'      % Form product of the individual elements,
```

i.e. $[b_1c_1, b_2c_2, \ldots, b_nc_n]$

```
ans =
      2    12    10    15
```



# Some Vector Operations/Manipulations

```
>> (b'.*c').^2          % Another example of exponentiation and .
ans =
      4    144    100    225
```

Vector inner and outer products:

```
>> c'*b      % Form inner product of 2 vectors → a scalar
ans =
      39
```

# Some Vector Operations/Manipulations

```
>> b*c'     % Form the outer product of 2 vectors → a matrix
ans =
     2     4     5     3
     6    12    15     9
     4     8    10     6
    10    20    25    15
```

## Matrices:

Some basic conventions:

1. Separate the element of a row with a blanks or commas



# Matrices (Cont'd)

2. Use semicolons ; to indicate the end of each row

3. Surround the entire list of elements with square brackets, [ ]

```
>> A = [1 2 3; 5 7 4]          % Entering a 2×3 matrix

A =
     1     2     3
     5     7     4

>> A(2,1)     % Access element of second row, first column

ans =
     5
```

# Matrices (Cont'd)

Consider a larger matrix:

>> B = [2 3 1 5 7; 3 5 1 6 7; 8 3 2 1 4; 5 7 10 3 4]

B =

| | | | | |
|---|---|---|---|---|
| 2 | 3 | 1 | 5 | 7 |
| 3 | 5 | 1 | 6 | 7 |
| 8 | 3 | 2 | 1 | 4 |
| 5 | 7 | 10 | 3 | 4 |

Sub-matrices can be extracted from B using the colon operator

The syntax is: (start_row:end_row, start_column:end_column)



# Matrices (Cont'd)

>> B_submatrix = B(2:3, 2:4) % Extract a 2×3 sub-matrix

B_submatrix =

| | | |
|---|---|---|
| 5 | 1 | 6 |
| 3 | 2 | 1 |

>> A(:, 3) = [ ] % Delete the third column of matrix A

A =

| | |
|---|---|
| 1 | 2 |
| 5 | 7 |

>> A(:, 3) = [3; 4] % Add another column to A

A =

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 5 | 7 | 4 |

# Matrices (Cont'd)

Some useful functions for manipulating matrices:

diag(A) - Produces the diagonal of matrix A
inv(A) - Finds the inverse of matrix A
eig(A) - Computes the eigenvalues of matrix A
eye(n) - Generates an $n \times n$ identity matrix
zeros(n, m) - Generates an $n \times m$ matrix of zeros
ones(n, m) - Generates an $n \times m$ matrix of ones

Matrix manipulations can be used to solve a system of algebraic equations!!!



# Example of the Use of Matrices

To solve a Stoichiometric Balance Problem:

$$x_1 CH_4 + x_2 O_2 \text{ ---> } x_3 CO_2 + x_4 H_2O$$ (combustion of methane)

The balance equations are:

$$x_1 = x_3 \;,\; 4x_1 = 2x_4 \;,\; 2x_2 = 2x_3 + x_4$$

3 equations but 4 unknowns ==> set $x_1 = 1$

## Example of the Use of Matrices (Cont'd)

The matrix form is:

$$\begin{pmatrix} 1 & 0 & -1 & 0 \\ 4 & 0 & 0 & -2 \\ 0 & 2 & -2 & -1 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

The solution from MATLAB is:

$CH_4 + 2O_2$ ---> $CO_2 + 2H_2O$



# Solving Nonlinear Algebraic Equations

# Solving Nonlinear Equations

❑ There are 3 important MATLAB functions for solving nonlinear equations: f(x) = 0

1. roots → special function to solve for polynomial roots

2. solve → generalized *symbolic* solver for roots of a set of nonlinear equations

3. fsolve → generalized *numerical* solver for roots of a set of nonlinear equations



# Syntax of Roots Function

❑ Syntax of roots is:

ROOTS(*C*) computes the roots of the polynomial whose coefficients are the elements of the vector *C*.

If *C* has N+1 components, the polynomial is *C*(1)**X*^N + *C*(2)**X*^(N-1) + ... + *C*(N)**X* + *C*(N+1).

# Example of Using Roots

Solve the following polynomial equation:

$$3x^4 + 2x^3 + x^2 + 4x - 6 = 0$$

```
>> c = [3 2 1 4 -6];
>> roots(c)
ans =
 -1.5476
  0.0435 + 1.2750i
  0.0435 - 1.2750i
  0.7940
```



# Syntax of the Function Solve

- The solve function can be used to solve nonlinear algebraic equations either symbolically or numerically if no analytical solution is available.

The most widely used syntax is (see help too):

solve('eqn1', 'eqn2', …, 'eqnN') or
solve('eqn1', 'eqn2', …, 'eqnN', 'var1', 'var2', …, 'varN')

# Some Examples of Using Solve

```
>> solve ('a*x^2+b*x+c=0', 'x')      % Produce an analytical
                                       solution
ans =
[ 1/2/a*(-b+(b^2-4*a*c)^(1/2))]
[ 1/2/a*(-b-(b^2-4*a*c)^(1/2))]
>> solve('x=cos(x)')                 % Produce a numerical result

ans =

0.7390851332151606416553120876738 7
```



# More Examples of Using Solve

Consider the following set of nonlinear equations:

$$x^2 + x - y^2 = 1 \quad \text{and} \quad y - \sin(x^2) = 0$$

```
>> xy = solve ('x^2+x-y^2-1=0', 'y - sin(x^2)=0')
```

```
xy =
   x: [1x1 sym]
   y: [1x1 sym]
>> xy.x
ans =
0.72595072614319203165298873673696
>> xy.y
ans =
0.50294650106250846284482819550095
```

```
[x y] = solve(……)
x =
0.72595…..
y =
0.50294…..
```

# Using the Double Command

- DOUBLE(X) returns the double precision value for X. If X is already a double precision array, DOUBLE has no effect.
- DOUBLE is very useful in converting symbolic numbers into double-precision numbers.

```
>> z = solve('3*x^2-4*x-10=0')
z =
[ 2/3+1/3*34^(1/2)]
[ 2/3-1/3*34^(1/2)]
>> double(z)
ans =
   2.6103
  -1.2770
```

```
>> format short
>> xy = solve ('x^2+x-y^2-1=0','y sin(x^2)=0');
         >> disp(xy.x)
0.725950726143192031652988736736 96
>> disp(xy.y)
0.502946501062508462844828195500 95
>> disp(double(xy.x))
0.7260
>> disp(double(xy.y))
0.5029
```



# Using Parameters in Solve Function

- How to pass parameter values computed earlier into the solve function? First just one unknown:

```
>> a = 4;
>> b = a/2;
>> syms x                         % define a symbolic variable
>> F = a*x-b*cos(x);
>> answer = solve(F);
>> disp(double(answer))
     0.4502
```

# Using Parameters in Solve Function

- Now solve for 2 unknowns from 2 nonlinear equations:

```
% Solve a*y-cos(z)=0 and y+b*log(z) = 0
>> syms y z
>> F1 = a*y-cos(z);
>> F2 = y+b*log(z);
>> yz = solve(F1, F2);
>> disp(double(yz.y)), disp(double(yz.z))
    0.1499
    0.9278
```



# Syntax of The Function fsolve

- The fsolve function solves a system of nonlinear equations of several variables.

- The most widely used syntax is (see help too):

  x = fsolve(fun, x0)

  where

  fun = an M-file function containing the system of nonlinear equations

  x0 = the initial guesses of the variables

## Example of Using fsolve

- Solve: $2x_1 - x_2 - \exp(-x_1) = 0$ and $-x_1 + 2x_2 - \exp(-x_2) = 0$ starting at $x_1 = -5$ and $x_2 = -5$
- First, write an M-file that computes F, the values of the equations at $x$.

```
function F = myfun(x)
F = [2*x(1) - x(2) - exp(-x(1)); -x(1) + 2*x(2) - exp(-x(2))];
>> x0 = [-5 -5];
>> x = fsolve(@myfun, x0)
x =
       0.5671   0.5671
```



# Solving Ordinary Differential Equations

# Solving ODEs in MATLAB

- The most widely used functions in MATLAB to solve a system of 1st-order ODEs are: ODE23 and ODE45

$$dy/dt = f(t, y) \qquad \text{s.t. } y(0) = a$$

- Based on the Runge-Kutta numerical method
- ODE23 is low-order while ODE45 is medium-order
- The higher the order, the more accurate the numerical algorithm



# Solving ODEs in MATLAB (Cont'd)

- A function is written for the ODEs as an M-file.

Example: Solve the following ODEs

$$dy_1/dt = 2y_1 - 0.001y_1y_2$$

$$dy_2/dt = -10y_2 + 0.002y_1y_2$$

$$\text{s.t. } y_1(0) = 5000$$
$$y_2(0) = 100$$

# Solving ODEs in MATLAB (Cont'd)

- The syntax of ODE23 and ODE45 is:

  [t, y] = ode23(odefun, tspan, y0)

where odefun is the name of the M-file containing the ODE functions; tspan is the length of simulation; y0 is the initial condition

Create an M-file called 'fxy.m', which contains the following code:

```
function fy = ode(t, y)
fy = zeros(2,1);        % Initialize fy as 2 × 1 matrix to zeros
fy(1) = 2*y(1)-0.001*y(1)*y(2);
fy(2) = -10*y(2)+0.002*y(1)*y(2);
```

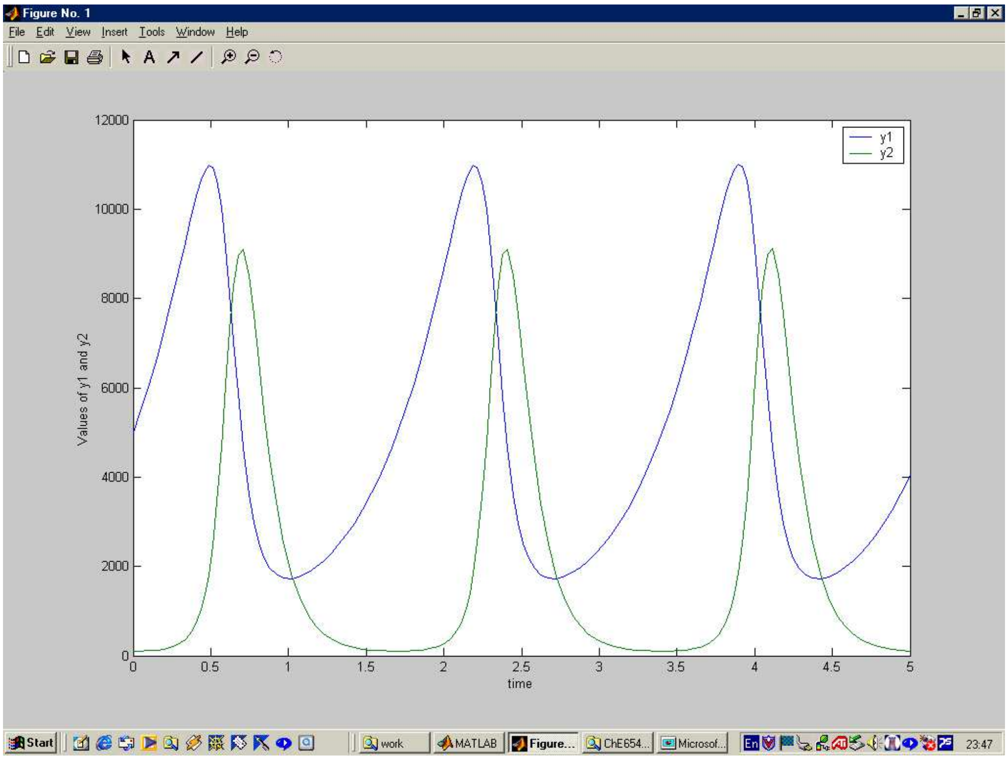


# Solving ODEs in MATLAB (Cont'd)

The solution of the ODEs can now be obtained by entering the following MATLAB commands, or put them into a script file:

```
>> simtime = 5;                 % Length of simulation
>> inity = [5000, 100];         % Initial values at t=0
>> [t, y] = ode23('fxy', simtime, inity)        % Solve the ODEs
>> plot(t,y);
>> xlabel('time')
>> ylabel('Values of y1 and y2')
>> legend('y1', 'y2')
```

# Solving ODEs in MATLAB (Cont'd)






# Plotting in MATLAB

MATLAB has extensive facilities for displaying vectors and matrices as graph, as well as annotating and printing these graphs.

```
>> x = [0 1 2 3 4 5 6 7 8 9 10];        % Setting the x values
>> y = x.^2;                            % y = x^2
>> plot(x,y)                                % Plot of a quadratic
>> title('Graph of a Quadratic')        % Put in a title for the graph
>> xlabel('Values of x')                    % Label the x-axis
>> ylabel('y = x^2')                    % Label the y-axis
>> legend('y')                      % Put in a legend for multiple lines
```

# Solving Higher-Order ODEs

- For higher-order ODEs (e.g. 2nd-order, 3rd-order, etc.), must reduce them to a system of 1st-order ODEs.
- There are 2 kinds of higher-order ODE problems:
  - Initial-value problems (IVPs)
  - Boundary-value problems (BVPs)

$y'' + 3y' - xy = \sin(x), \qquad y'(0) = 0, y(0) = 1 \qquad$ => IVP

$y'' - xy' + y = \exp(-x), \qquad y'(0) = 0, y(1) = 2 \qquad$ => BVP

$y''' + y'' + 3y' - y = 0, \qquad y''(0) = 0, y'(0) = 1, y(2) = 5 \qquad$ => BVP



# Reducing Higher-Order ODEs

- Consider the 2nd order ODE:

$$d^2y/dt^2 = 3\, dy/dt + 6\, y - \cos(t), \qquad y'(0) = 0, y(0) = 1$$

The ODE can be converted into a pair of 1st-order ODEs:

Define $x = dy/dt$ so that

$$dx/dt = 3\, x + 6\, y - \cos(t) \qquad (1)$$

$$dy/dt = x \qquad (2)$$

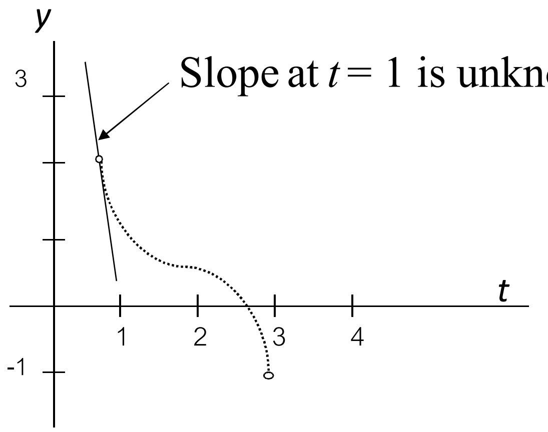
subject to $x(0) = 0, y(0) = 1$

# Solving Boundary-Value Problems

❑ Shooting Method - Trial and Error

Consider the following 2nd-order ODE:

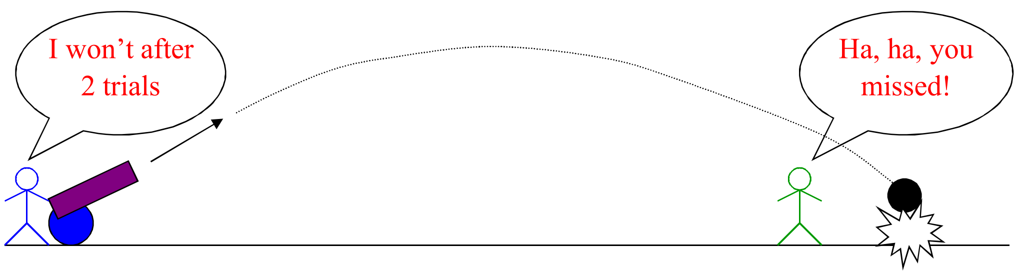
$$d^2y/dt^2 - (1 - t/5)\,y = t, \qquad y(1) = 2,\ y(3) = -1$$






# Shooting Method (Cont'd)

❑ Based on the mechanics of an artillery problem




❑ Solve the ODE as an IVP by guessing the slope $y'(1)$ to get $y(3)$.

- If $y(3) > -1$, then the guess is too high. Guess a lower value for $y'$.

- If $y(3) < -1$, then the guess is too low. Guess a higher value for $y'$.

- After 2 trials, linearly interpolate or extrapolate for a third trial.

# Shooting Method (Cont'd)

- The formula for linear interpolation/extrapolation is:

$$y'(1) = G1 + \frac{G2 - G1}{R2 - R1}(D - R1)$$

where G1 = first guess at initial slope
G2 = second guess at initial slope
R1 = first result at endpoint (using G1)
R2 = second result at endpoint (using G2)
D = desired value at the endpoint

Note: The third trial always gives the correct results if the ODE is *linear* => An ODE is linear if the coefficients of each derivative term and the forcing function are not functions of $y$.



# Shooting Method in MATLAB

- First reduce the 2nd-order ODE into a pair of 1st-order ODEs:

$dy/dt = x$ and $dx/dt - (1 - t/5)\,y = t$, $y(1) = 2, y(3) = -1$

- MATLAB m-file: fshoot.m

```
function fy = ode(t, y)
fy = zeros(2,1);
fy(1) = y(2);
fy(2) = (1-t/5)*y(1) + t;
```

# Shooting Method in MATLAB (Cont'd)

- First trial => guess $y'(1) = x(1) = -1.5$

```
clc
clear
simtime = [1:0.2:3];
inity = [2, -1.5];
[t, y] = ode45('fshoot', simtime, inity);
```

Run from **t** = 1 to **t** = 3 with Δ**t** = 0.2

**y** **x** or **y′**

| | |
|---|---|
| 2.0000 | -1.5000 |
| 1.7514 | -0.9886 |
| 1.6043 | -0.4814 |
| 1.5597 | 0.0389 |
| 1.6218 | 0.5876 |
| 1.7976 | 1.1783 |
| 2.0967 | 1.8227 |
| 2.5309 | 2.5310 |
| 3.1139 | 3.3116 |
| 3.8608 | 4.1706 |
| 4.7876 | 5.1119 |

**y**(**t**=3) which is > -1.0 wanted

so **y′**(1) is too large



# Shooting Method in MATLAB (Cont'd)

- Second trial => guess $y'(1) = x(1) = -3.0$

```
clc
clear
simtime = [1:0.2:3];
inity = [2, -3.0];
[t, y] = ode45('fshoot', simtime, inity);
```

Run from **t** = 1 to **t** = 3 with Δ**t** = 0.2

**y** **x** or **y′**

| | |
|---|---|
| 2.0000 | -3.0000 |
| 1.4498 | -2.5118 |
| 0.9921 | -2.0719 |
| 0.6192 | -1.6598 |
| 0.3275 | -1.2580 |
| 0.1163 | -0.8512 |
| -0.0118 | -0.4259 |
| -0.0520 | 0.0299 |
| 0.0029 | 0.5266 |
| 0.1620 | 1.0732 |
| 0.4360 | 1.6773 |

**y**(**t**=3) is > -1.0

so **y′**(1) is still too large

# The Complete MATLAB File

```
% Shooting Method to solve a 2nd-order ODE
clc
clear
% first trial
simtime = [1:0.2:3];
g1 = -1.5;
inity = [2, g1];
[t, y] = ode45['fshoot', simtime, inity)
r1 = y(11,1);
% second trial
g2 = -3.0;
inity = [2, g2];
[t, y] = ode45('fshoot', simtime, inity)
r2 = y(11,1);
% third trial and the solution
g3 = g1 + (g2-g1)/(r2-r1)*(-1-r1);
inity = [2, g3];
[t, y] = ode45('fshoot', simtime, inity)
```

Output:

| | |
|---|---|
| 2.0000 | -3.4950 |
| 1.3503 | -3.0145 |
| 0.7900 | -2.5967 |
| 0.3088 | -2.2204 |
| -0.0997 | -1.8671 |
| -0.4385 | -1.5209 |
| -0.7076 | -1.1679 |
| -0.9043 | -0.7955 |
| -1.0237 | -0.3925 |
| -1.0586 | 0.0511 |
| -1.0000 | 0.5439 |

y(t=3)



# Programming in MATLAB

# Programming in MATLAB

- MATLAB is both a powerful programming language as well as an interactive computational environment
- Files that contain code in the MATLAB language are called M-files (file names must end with the extension '.m')
- There are 2 kinds of M-files:
  - Scripts, a simple text file where you can place MATLAB commands.
  - Functions, which can accept input arguments and return output arguments



# The IF Condition Statement

- The IF statement evaluates a logical expression and executes a group of statements when the expression is true.

The general form of the IF statement is

```
IF expression
    statements
ELSEIF expression
    statements
ELSE
    statements
END
```

The ELSEIF and ELSE parts are optional. The valid operators in the expression are = =, <, < =, >, > =, and ~=.

# Example of IF Condition Statements

Given a positive integer number, determine if the number is divisible by 5.

The m-file is called "ifthenelse"

```
clc
clear
number = input('Please enter a positive integer number: ')
if number < 0
        fprintf ('Sorry, %5i is not a positive number \n', number)
elseif round(number) - number ~= 0
        fprintf ('Sorry, %10.5f is not an integer number \n', number)
elseif rem(number, 5) == 0
        fprintf ('%5i is divisible by 5 \n', number)
else
        fprintf ('%5i is not divisible by 5 \n', number)
        remainder = rem(number,5);
        fprintf ('%5i is the remainder \n', remainder)
end
```

Returns the remainder if not divisible by 5



# Example of IF Statements (Cont'd)

In MATLAB, type: ifthenelse

```
Please enter a positive integer number: -25
Sorry,   -25 is not a positive number
>>
Please enter a positive integer number: 15.23
Sorry,   15.23000 is not an integer number
>>
Please enter a positive integer number: 80
  80 is divisible by 5
>>
Please enter a positive integer number: 34
  34 is not divisible by 5
   4 is the remainder
>>
```

# The FOR Statement

- The FOR statement repeats a group of statements a fixed, predetermined number of times.

The general form of the FOR statement is

```
FOR variable = expr
    statements
 END
```

where expr is often of the form X:Y



# Example of FOR Loop Statements

Given a positive integer number *n*, calculate the sum of (1+2+3+…+*n*)

```
clc
clear
number = input('Please enter a positive integer number: ‘)
if number < 0
    fprintf (‘Sorry, %5i is not a positive number \n’, number)
else
    sum = 0;
    for i = 1:number
              sum = sum + i;
    end
    fprintf (‘The sum is  %8i \n’, sum)
end
```

The m-file is called “forloop”

```
In MATLAB, type: forloop
Please enter a positive integer number: 100
The sum is      5050
>>
```

# The WHILE and BREAK Statements

- The WHILE loop repeats a group of statements an indefinite number of times, under control of a logical condition.

The general form of the WHILE statement is

```
WHILE expression
    statements
END
```

- The BREAK statement lets you exit early from a FOR or WHILE loop. This prevents MATLAB from going into an infinite loop.



# Example of WHILE Statements

The Hi-Lo game:

Objective: Try to correctly guess an integer between 0 and 100 generated by the computer in as few trials as possible.

The m-file is called "whileloop"

```
clc
clear
myinteger = round(100*rand);
flag = 0;
while flag ==0
    fprintf ('\n')
    guess = input('Please guess an integer between 0 and 100 I have in mind: ');
```

# Example of WHILE Statements (Cont'd)

```
    if guess == myinteger
        flag = 1;
        fprintf ('\n')
        fprintf ('You guessed right!!!\n')
        fprintf ('My number is %3i \n', myinteger)
    elseif guess < myinteger
        fprintf ('Your number is too low.  Please guess again\n')
    else
        fprintf ('Your number is too high.  Please guess again\n')
    end
end
```



# Example of WHILE Statements (Cont'd)

```
Please guess an integer between 0 and 100 I have in mind: 50
Your number is too low.  Please guess again
Please guess an integer between 0 and 100 I have in mind: 75
Your number is too low.  Please guess again
Please guess an integer between 0 and 100 I have in mind: 88
Your number is too high.  Please guess again
Please guess an integer between 0 and 100 I have in mind: 82
Your number is too high.  Please guess again
Please guess an integer between 0 and 100 I have in mind: 79
Your number is too low.  Please guess again
Please guess an integer between 0 and 100 I have in mind: 81

You guessed right!!!
My number is  81
>>
```

# Workshops



## Workshop 1: Basic Calculations

Use MATLAB to carry out the following calculations:

(a) Solve the equation: $2x^2 – 5x – 20 = 0$, using the quadratic formula. Report your answers in 6 decimal places.

(b) What is the product of the two roots of the quadratic equation: $4x^2 + 3x + 13 = 0$. Report your answer in 4 decimal places.

(c) Compute the distance between two points, namely (2, -4, 9) and (-3, 1, -7), given in the Cartesian coordinates.

(d) Convert the Cartesian coordinates (4, 15) into the polar coordinates ($r$, $\theta$). Report your answers in 2 decimal places and show $\theta$ in both degree and radian.

# Workshop 1: Basic Calculations (Cont'd)

Use MATLAB to carry out the following calculations:

(e) A quick search on the Internet shows that the vapor pressure of acetone is given by:

$$\log_{10}(P^{VAP}) = 7.2316 - \frac{1277.03}{T + 237.23} \qquad \textbf{T} \text{ in } ^{\circ}\text{C and } \textbf{P} \text{ in mmHg}$$

Verify the accuracy of this vapor pressure at $T = 25$ °C by comparing it (in terms of relative % error with 5 decimal places) with the following vapor pressure equation reported by Ambrose, Sprake, *et al.* (1974):

$$\log_{10}(P^{VAP}) = 4.42448 - \frac{1312.253}{T - 32.445} \qquad \textbf{T} \text{ in Kelvin and } \textbf{P} \text{ in bar}$$

# Workshop 2: Matrix Manipulations

(a) Consider the following arrays:

$$\mathbf{A} = \begin{pmatrix} 1 & 4 & 2 \\ 2 & 4 & 100 \\ 7 & 9 & 7 \\ 3 & \pi & 42 \end{pmatrix} \qquad \mathbf{B} = \ln(\mathbf{A})$$

Use MATLAB to do the following (use "format short"):

- Select just the second row of **B**.
- Determine the sum of the second row of **B**.
- Multiply the second column of **B** and the first column of **A** (element-by-element)
- Determine the maximum value in the vector resulting from element-by-element multiplication of the second column of **B** with the first column of **A**.
- Determine the sum of the first row of **A** divided element-by-element by the first three elements of the third column of **B**.

# Workshop 2 (Cont'd)

(b) Use MATLAB to determine the stoichiometric ratios of molecular species in the following reaction. You must find the lowest integer number for each stoichiometric coefficient.

$$a\ HIO_3 + b\ FeI_2 + c\ HCl \rightarrow d\ FeCl_3 + e\ ICl + f\ H_2O$$

where $HIO_3$ = Iodic Acid, $FeI_2$ = Ferrous Iodide,
$FeCl_3$ = Ferric Chloride, and ICl = Idodine Monochloride

**Answers**:

$a$ = ______ $b$ = ______ $c$ = ______ $d$ = ______
$e$ = ______ $f$ = ______

# Workshop 3: Molar Volume and Z from Redlich-Kwong-Soave Equation of State

The Redlich-Kwong-Soave equation of state contains 2 empirical parameters $a$ and $b$, and is given by:

$$P = \frac{RT}{\underline{V} - b} - \frac{a}{\underline{V}(\underline{V} + b)}$$ where

$$a = 0.42747[R^2\, T_C^{\,2}/P_C]\alpha(T)$$

$$b = 0.08664[R\, T_C/P_C]$$

$$\alpha(T) = [1 + m(1 - T_r^{\,1/2})]^2 \text{ and } T_r = T/T_C$$

$$m = 0.480 + 1.57w - 0.176w^2$$

$$w = -1.0 - \log10\, [P^{VAP}(T_r = 0.7)/P_C] = \text{Pitzer acentric factor}$$

# Workshop 3: Molar Volume and $Z$ from Redlich-Kwong-Soave Equation of State

The variables are defined by:

**P** = pressure in atm

**V** = molar volume in L/gmole

**T** = temperature in K

**R** = gas constant (0.08206 atm-L/gmole-K)

$\mathbf{T_C}$ = the critical temperature (405.5 K for ammonia)

$\mathbf{P_C}$ = the critical pressure (111.3 atm for ammonia)

$\mathbf{P}^{VAP}$ = vapor pressure (6.2 atm at $\mathbf{T_r}$ = 0.7 for ammonia)

Use MATLAB to answer the following questions:

(a) Calculate the molar volume and compressibility factor **Z** for gaseous ammonia at a pressure **P** = 56 atm and a temperature **T** = 450 K.

(b) Repeat the calculations for the following reduced pressures: $\mathbf{P_r}$ = 1, 2, 4, 10, and 20.



# Workshop 4: Solving an ODE

Write a MATLAB script file to solve the following 4th-order ODE using ode23:

$$d^4y/dt^4 = y + 7.5\sin(2t) + 16\sin^2 t - 14\cos^2 t + t^3$$

$$\text{s.t. } y(0) = 0,\ dy(0)/dt = 3,\ d^2y(0)/dt^2 = 6,\ d^3y(0)/dt^3 = -8$$

The above ODE has an analytical solution of:

$$y(t) = c_1 e^t + c_2\sin(2t) - c_3\cos^2(t) + c_4 t^3$$

# Workshop 4: Solving an ODE (Cont'd)

Make a plot of the numerical solution ($y$ versus $t$) from MATLAB. Then, compare your MATLAB solution with the analytical solution below by reporting the relative % differences. Run the simulation from $t = 0$ to $t = 1$ with an increment of 0.1. Include 6 decimal places in reporting all your numbers.

Note: You must do all your work in MATLAB, which includes determining the constants $c_1$, $c_2$, $c_3$, and $c_4$ in the analytical solution.



# Workshop 5: Newton's Method

Consider the following system of nonlinear equations:

$$f_1(x, y, z) = xyz - x^2 + y^2 - 1.34 = 0$$

$$f_2(x, y, z) = xy - z^2 - 0.09 = 0$$

$$f_3(x, y, z) = e^x - e^y + z - 0.41 = 0$$

Write a MATLAB program to do the following:

(a) Solve for the roots of the above equations using Newton's method. Use an initial guess of $(x, y, z) = (0, -1, 0)$. Accept the solution only when $|f_1|$, $|f_2|$, and $|f_3| \le 10^{-3}$.

# Workshop 5: Newton's Method (Cont'd)

(b) Solve the equations again using the function *solve* in MATLAB.

(c) Compare the % relative errors between the values of $x$, $y$, and $z$ obtained from Newton and from MATLAB. Report the errors with 5 decimal places.

Recall that the iterative formula for Newton's method is:

$$\boldsymbol{x}_{\mathbf{k+1}} = \boldsymbol{x}_{\mathbf{k}} - \boldsymbol{J}^{-1}(\boldsymbol{x}_{\mathbf{k}}) * \mathbf{f}(\boldsymbol{x}_{\mathbf{k}})$$

where $\boldsymbol{J}^{-1}$ is the inverse of the Jacobian matrix, $\boldsymbol{J}$

$$\boldsymbol{J} = \left\{ \begin{array}{lll} \partial f_1/\partial x_1 & \partial f_1/\partial x_2 & \ldots\ \partial f_1/\partial x_n \\ \partial f_2/\partial x_1 & \partial f_2/\partial x_2 & \ldots\ \partial f_2/\partial x_n \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots & & \\ \partial f_n/\partial x_1 & \partial f_n/\partial x_2 & \ldots\ \partial f_n/\partial x_n \end{array} \right\}$$