58:110 Computer-Aided Engineering Spring 2005

Introduction to MATLAB

Department of Mechanical and industrial engineering

January 2005

The University of Iowa

Topics

- Introduction
- Running MATLAB and MATLAB Environment
- Getting help
- Variables
- Vectors, Matrices, and linear Algebra
- Mathematical Functions and Applications
- Plotting
- Programming
- M-files
- User Defined Functions

Introduction

- **What is MATLAB**

  MATLAB, which stands for MATrix LABoratory, is a powerful program that combines computation and visualization capability for science and engineering simulations.

- **MATLAB provides the user:**

  Manage variables

  Import and export data

  Perform calculations

  Generate Plots

  …………..

Running MATLAB

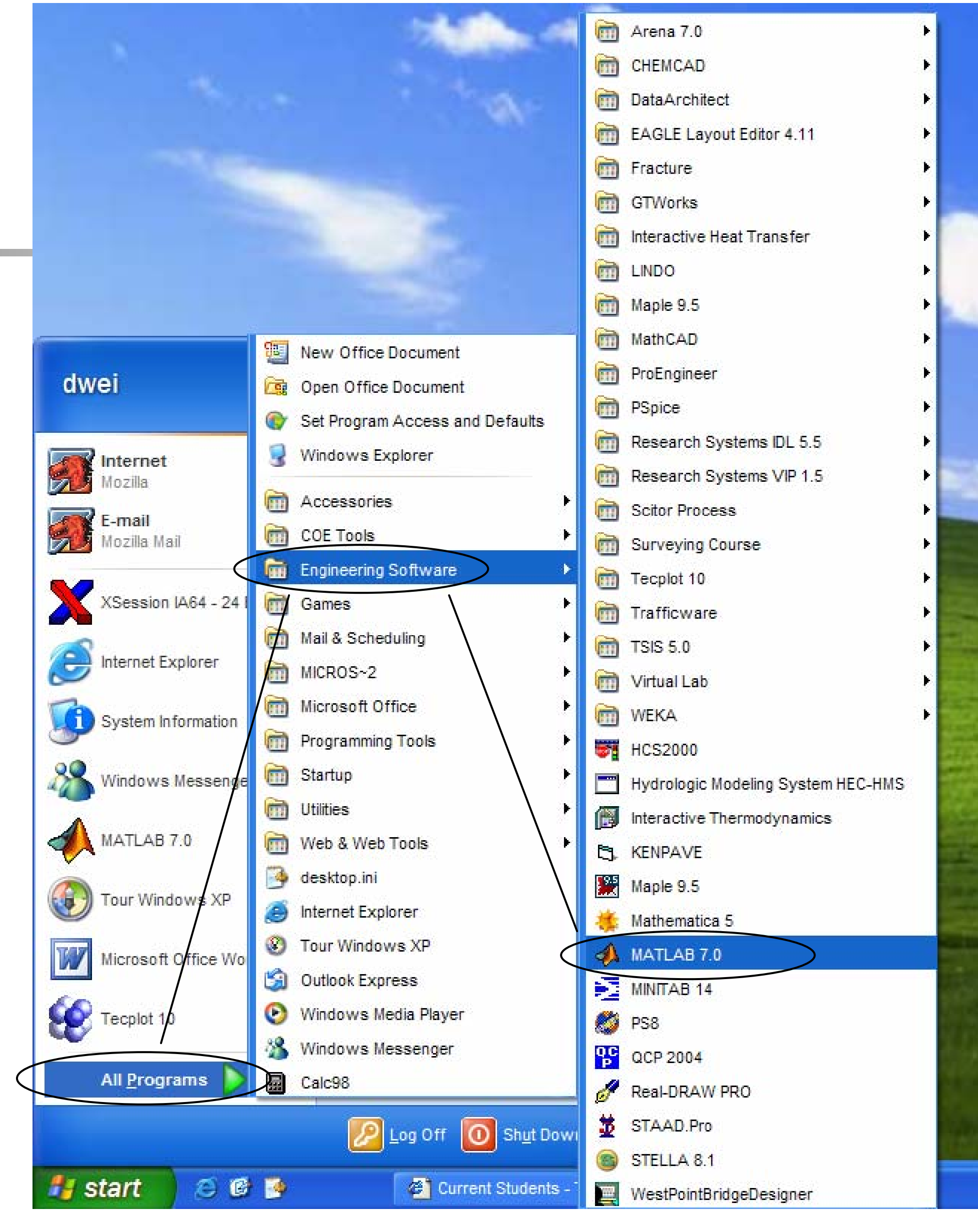
To run MATLAB:

Login any ICAEN PC with WIN XP

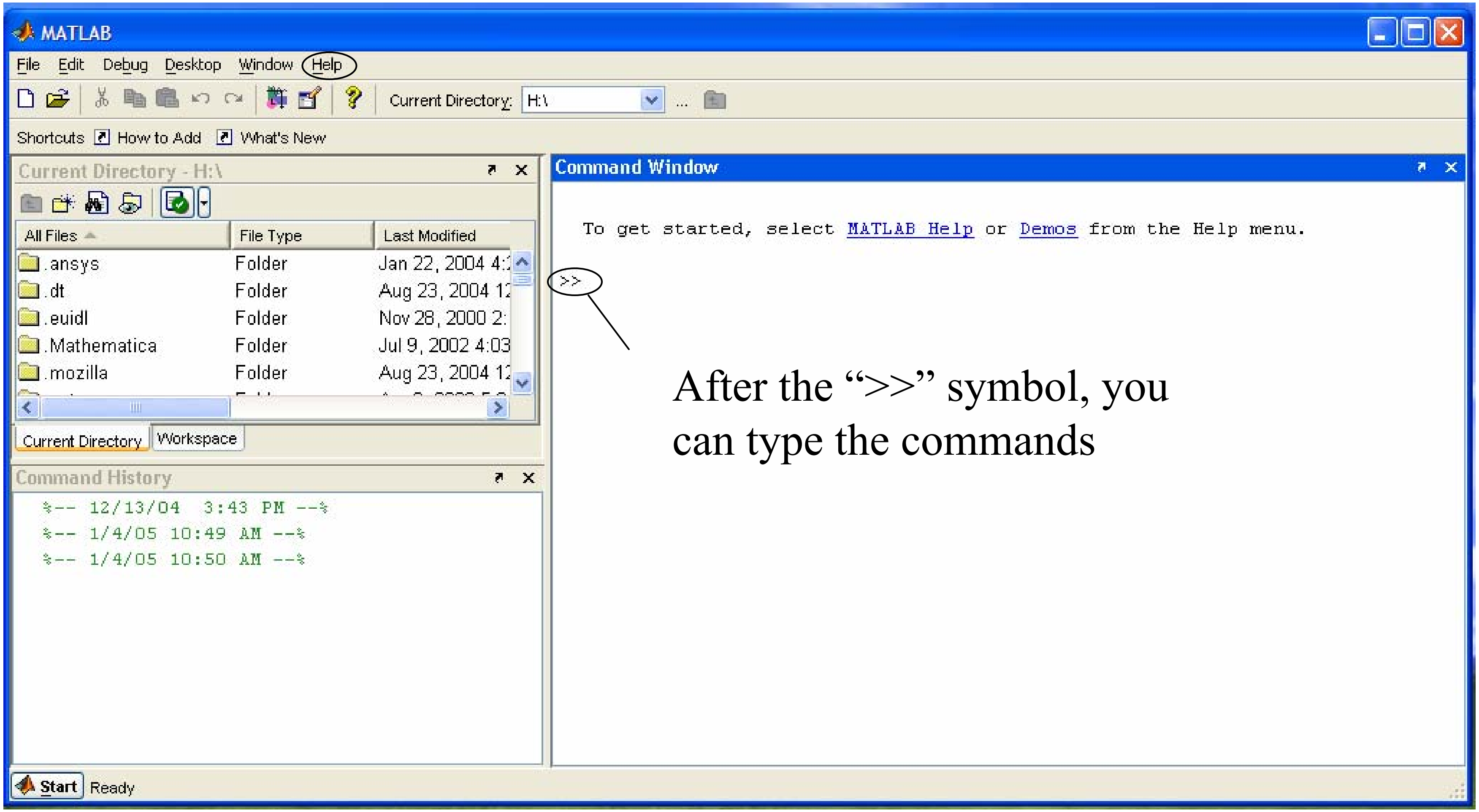
Start -> All Programs

-> Engineering Software

-> MATLAB 7.0



Main Working Windows



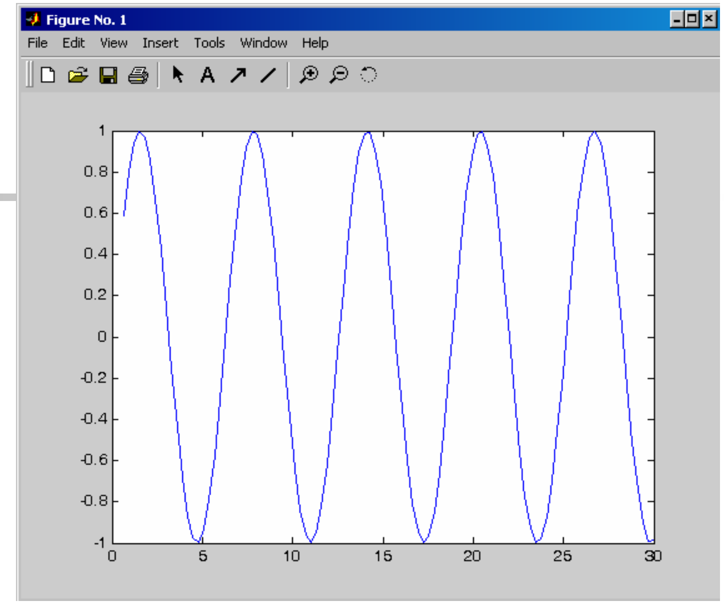
After the “>>” symbol, you can type the commands

Display Windows

- **Graphic (Figure) Window**

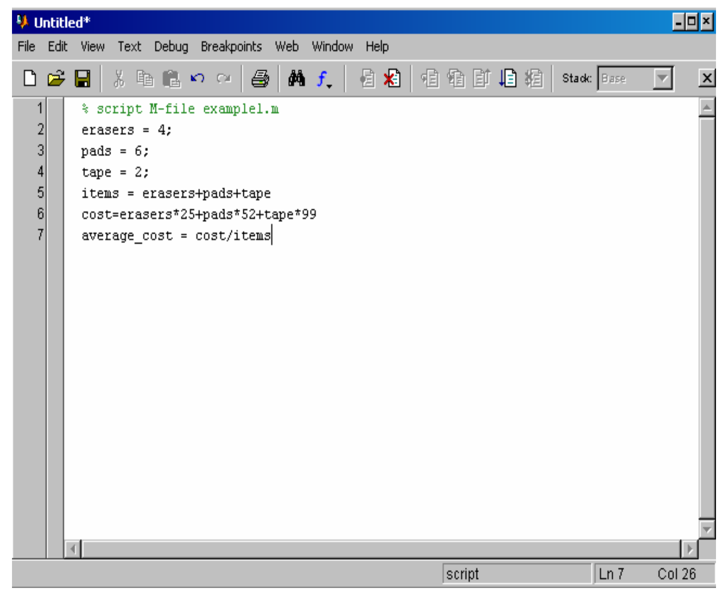
  Displays plots and graphs

  Created response to graphics commands



- **M-file editor/debugger window**

  Create and edits scripts of commands called M-files



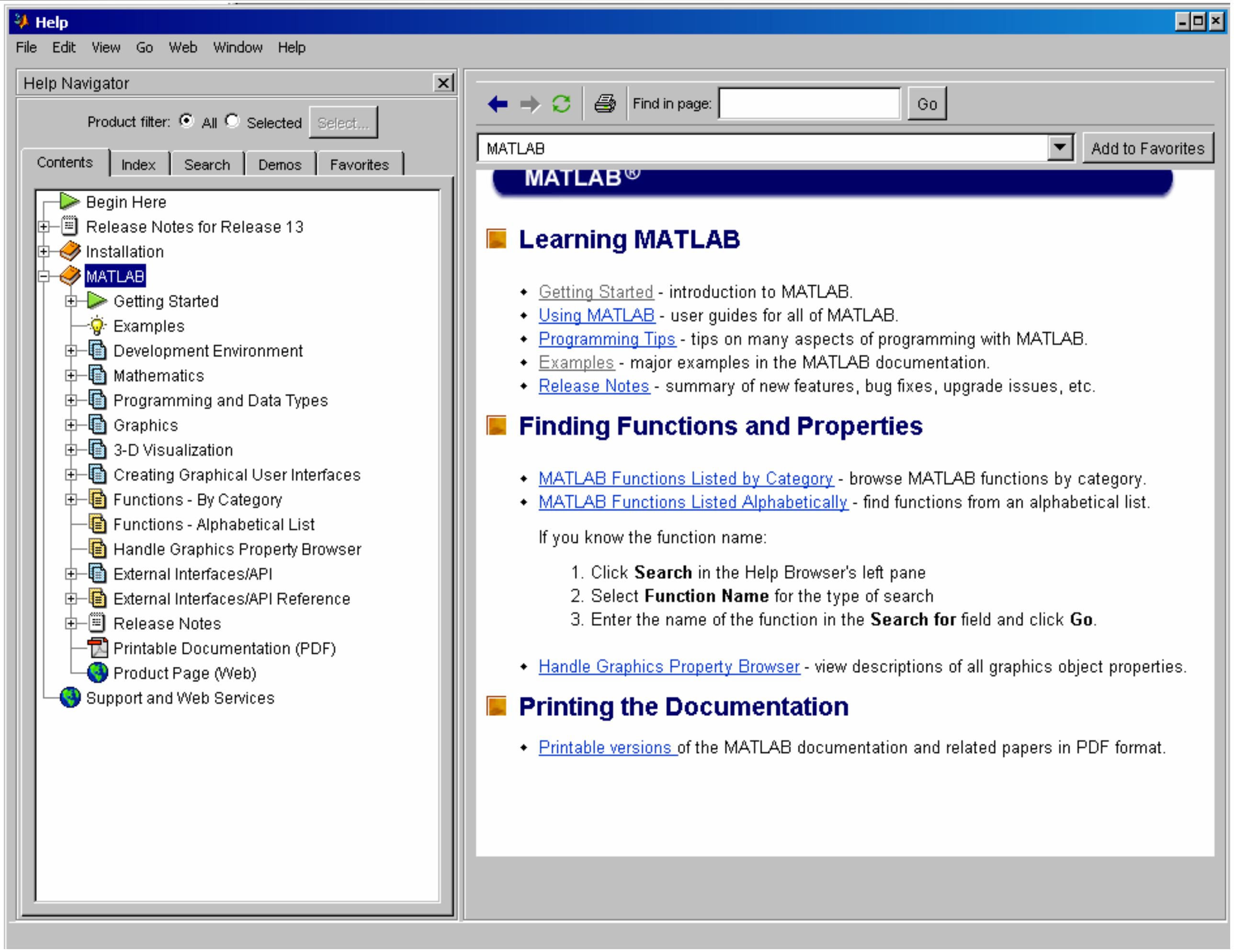
Getting Help

To get help:

MATLAB main menu

-> Help

-> MATLAB Help



Getting Help

- Type one of the following commands in the command window:
  - **help –** lists all the help topic
  - **help** *topic* – provides help for the specified topic
  - **help** *command* – provides help for the specified command
  - **helpwin** – opens a separate help window for navigation
  - **Lookfor** *keyword* – search all M-files for *keyword*
- Online resource

Variables

- Variable names:
  - Must start with a letter.
  - May contain only letters, digits, and the underscore “_”.
  - MATLAB is case sensitive, for example one & ONE are different variables.
  - MATLAB only recognizes the first 31 characters in a variable name.
- Assignment statement:
  - *Variable = number;*
  - *Variable = expression;*
- Example: >> t = 1234;
  - >> t = 1234
  - t =
  - 1234

Variables

- Special variables:
  - **ans**: default variable name for the result.
  - **pi**: π = 3.1415926 ……
  - **eps**: ε= 2.2204e-016, smallest value by which two numbers can differ
  - **inf**: ∞, infinity
  - **NAN** or **nan**: not-a-number
- Commands involving variables:
  - **who**: lists the names of the defined variables
  - **whos**: lists the names and sizes of defined variables
  - **clear**: clears all variables
  - **clear** *name*: clears the variable *name*
  - **clc**: clears the command window
  - **clf**: clears the current figure and the graph window

Vectors

- A row vector in MATLAB can be created by an explicit list, starting with a left bracket, entering the values separated by spaces (or commas) and closing the vector with a right bracket.
- A column vector can be created the same way, and the rows are separated by semicolons.
- Example:
  - >> x = [0 0.25*pi 0.5*pi 0.75*pi pi]
  - x = **x is a row vector**.
  - 0   0.7854   1.5708   2.3562   3.1416
  - y=[0; 0.25*pi; 0.5*pi; 0.75*pi; pi]
  - y = **y is a column vector**.
  - 0
  - 0.7854
  - 1.5708
  - 2.3562
  - 3.1416

Vectors

- Vector Addressing- A vector element is addressed in MATLAB with an integer index enclosed in parentheses.
- Example:
  - >> x(3)
  - ans =
  - 1.5708 **<- 3rd element of vector X**
- The colon notation may be used to address a block of elements
- **(start:increment:end)**
- Example:
  - >> x(1:2:5)
  - ans =
  - 0 1.5708 3.1416

Vectors

- Some useful commands:

x = start:end	Create row vector x starting with start, counting by 1 , ending at end
x = start:increment:end	Create row vector x starting with start, counting by increment, ending at or before end
x = linspace(start,end,number)	Create linearly spaced row vector x starting with start, ending at end, having number elements
x = logspace(start,end,number)	Create logarithmically spaced row vector x starting with start, ending with end, having number elements
length(x)	Returns the length of vector x
y = x'	Transpose of vector x
dot(x,y),cross(x,y)	Returns the scalar dot and vector cross product of the vector x and y

Array Operations

- Scalar-Array Mathematics
  - For addition, subtraction, multiplication, and division of an array by a scalar, simply apply the operation to all elements of the array
- Example:
  - >> f = [1 2; 3 4]
  - f =
  - 1 2
  - 3 4
  - >> g = pi * f / 3 + 0.8
  - g =
  - 1.8472 2.8944
  - 3.9416 4.9888

Array Operations

- Element-by-Element Array-Array Mathematics

operation	*Algebraic Form*	*MATLAB*
Addition	a + b	a + b
Subtraction	a – b	a – b
Multiplication	a × b	a .* b
Division	a ÷ b	a ./ b
Exponentiation	a^b	a .^ b

- Example:
  - >> x = [1 2 3];
  - >> y = [4 5 6];
  - >> z = x .* y
  - z =
  - 4 10 18

Matrices

- A matrix array is two-dimensional, having both mulitple rows and multiple columns.
  - It begins with [, and end with]
  - Spaces or commas are used to separate elements in a row
  - Semicolon or enter is used to separate rows
- Example:
  - >> f = [1 2 3; 4 5 6]
  - f =
  - 1 2 3
  - 4 5 6

  - >> h = [2 4 6
  - 1 3 5]
  - h =
  - 2 4 6
  - 1 3 5

Matrices

- Matrix Addressing:
  - Matrix name(row,column)
  - Colon maybe used in place of a row or column reference to select the entire row or column.
- Example:
  - >> f(2,3)
  - ans =
  - 6

  - >> h(:,1)
  - ans =
  - 2
  - 1

Matrices

- Some useful commands:

zeros(n)	Returns a n X n matrix of zeros
zeros(m,n)	Returns a m X n matrix of zeros
ones(n)	Returns a n X n matrix of ones
ones(m,n)	Returns a m X n matrix of ones
size(A)	For a m X n matrix A, returns the row vector [m,n] containing the number of rows and columns in matrix
length(A)	Returns the larger of the number of rows or columns in A

Matrices

- More commands:

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 = α A, where α is a scalar
Matrix Multiplication	C = A * B
Matrix Inverse	B = inv(A), A must be a square matrix in this case
Matrix powers	B = A * A , A must be a square matrix
Determinant	det(A), A must be a square matrix

Linear Equations

- Example: a system of 3 linear equations with 3 unknowns (x_1, x_2, x_3)
  - $3x_1 + 2x_2 - x_3 = 10$
  - $-x_1 + 3x_2 + 2x_3 = 5$
  - $x_1 - 2x_2 - x_3 = -1$
- Let:

$$\mathbf{A} = \begin{bmatrix} 3 & 2 & 1 \\ -1 & 3 & 2 \\ 1 & -1 & -1 \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 10 \\ 5 \\ -1 \end{bmatrix}$$

Then, the system can be described as:

$$\mathbf{Ax = b}$$

Linear Equations

- Solution by Matrix Inverse:

  Ax = b

  $\mathbf{A}^{-1}\,\mathbf{A}\mathbf{x} = \mathbf{A}^{-1}\,\mathbf{b}$

  Ax = b

MATLAB:

```
>> A = [3 2 -1; -1 3 2; 1 -1 -1];
>> b = [10;5;-1];
>> x = inv(A)*b
x =
   -2.0000
    5.0000
   -6.0000
```

- Solution by Matrix Division:
  - **Ax = b**
  - Can be solved by left division **b**÷**A**

MATLAB:

```
>> A = [3 2 -1; -1 3 2; 1 -1 -1];
>> b = [10;5;-1];
>> x =A \ b
x =
   -2.0000
    5.0000
   -6.0000
```

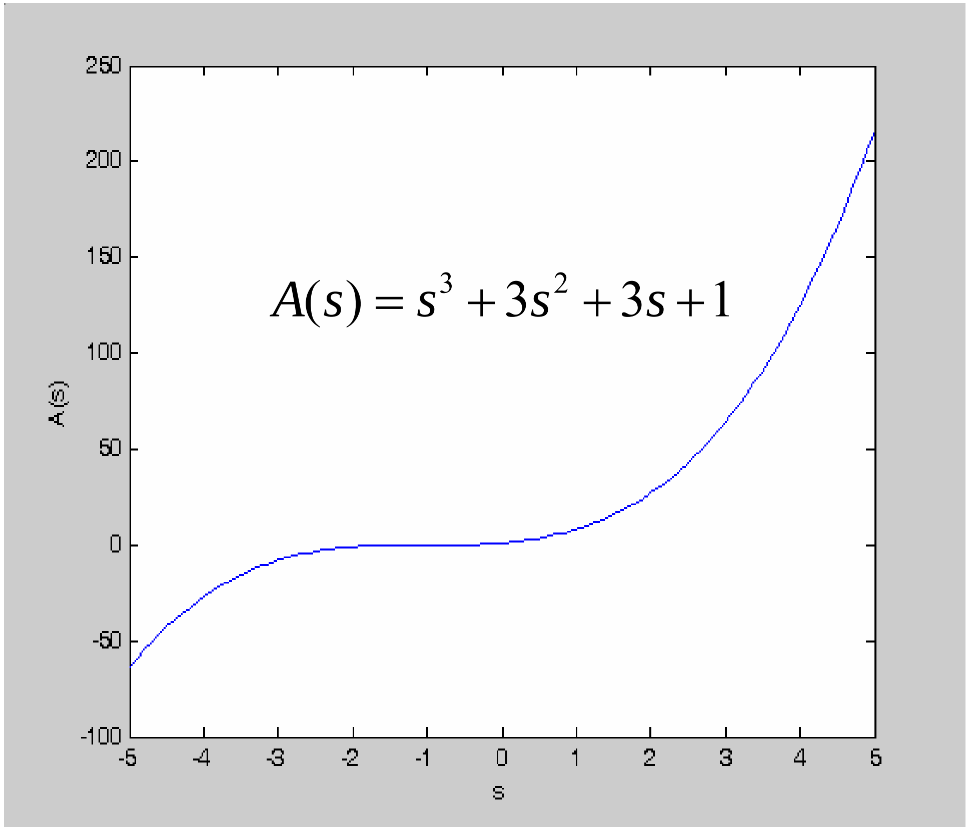
Polynomials

- The polynomials are represented by their coefficients in MATLAB
- Consider the following polynomial:

$$A(s) = s^3 + 3s^2 + 3s + 1$$

- For s is scalar: use scalar operations
  - A = s ^ 3 + 3 * s ^ 2 + 3 * s + 1;
- For s is a vector or a matrix: use array or element by element operation
  - A = s .^ 3 + 3 * s .^ 2 + 3 .* s + 1;
- Function **polyval(a,s):** evaluate a polynomial with coefficients in vector a for the values in s

Polynomials



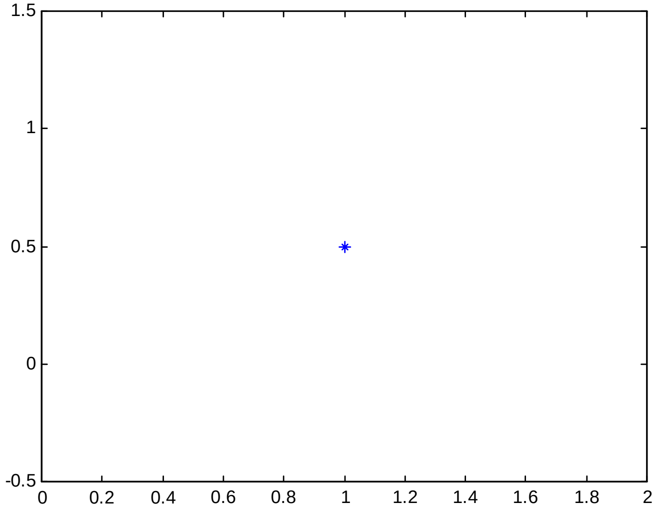
- MATLAB:
  - >> s = linspace(-5,5,100);
  - >> coeff = [1 3 3 1];
  - >> A = polyval(coeff,s);
  - >> plot(s,A)
  - >> xlabel('s')
  - >> ylabel('A(s)')

Polynomials

Operation	*MATLAB Command*	*Description*
Addition	**c = a + b**	Sum of polynomial A and B, the coefficient vectors must have the same length.
Scalar Multiple	**b = 3 * a**	Multiply the polynomial A by 3.
Polynomial Multiplication	**c = conv(a, b)**	Returns the coefficient vector for the resulting from the product of polynomial A and B.
Polynomial Division	**[q,r] = deconv(a,b)**	Returns the long division of A and B. q is the quotient polynomial coefficient, and r is the remainder polynomial coefficient.
Derivatives	**polyder(a)** **polyder(a,b)** **[n,d] = polyder(b,a)**	Returns the coefficients of the derivative of the polynomial A. Returns the coefficients of the derivative of the product of A and B. Returns the derivative of ratio B/A, represented as N/D.
Find Roots	**roots(a)**	Returns the roots of the polynomial A in column vector form.
Find Polynomials	**Poly(r)**	Returns the coefficient vector of the polynomial having roots r

Plotting

- For more information on 2-D plotting, type help graph2d
- Plotting a point:
  - >>plot (*variablename*, ‘*symbol*’)
- Example: Complex variable
- >>z = 1 + 0.5j;
- >>plot(z,‘*')
- Commands for axes:



Command	**Description**
axis([xmin xmax ymin ymax])	Define minimum and maximum values of the axes
axis square	Produce a square plot
axis equal	Equal scaling factors for both axes
axis normal	Turn off axis square, equal
axis (auto)	Return the axis to defaults

Plotting

- Plotting curves:
  - **plot(x,y)** – generate a linear plot of the values of x (horizontal axis) and y (vertical axis)
  - **semilogx(x,y)** - generate a plot of the values of x (logarithmic scale) and y (linear scale)
  - **semilogy(x,y) -**
  - **loglog(x,y)** - generate a plot of the values of x and y (both logarithmic scale)
- Multiple curves
  - **plot(x,y,w,z)** – multiple curves can be plotted on the same graph: y vs. x and z vs. w
  - **legend(‘string1’,’string2’, …)** – used to distinguish between plots on the same graph
- Multiple figures
  - **figure(n)** – use in creation of multiple plot windows before the command **plot()**
  - **close** – closes the figure n window
  - **close all** – closes all the plot windows
- Subplots:
  - **subplot(m,n,p)** – m by n grid of windows, with p specifying the current plot as the pth window

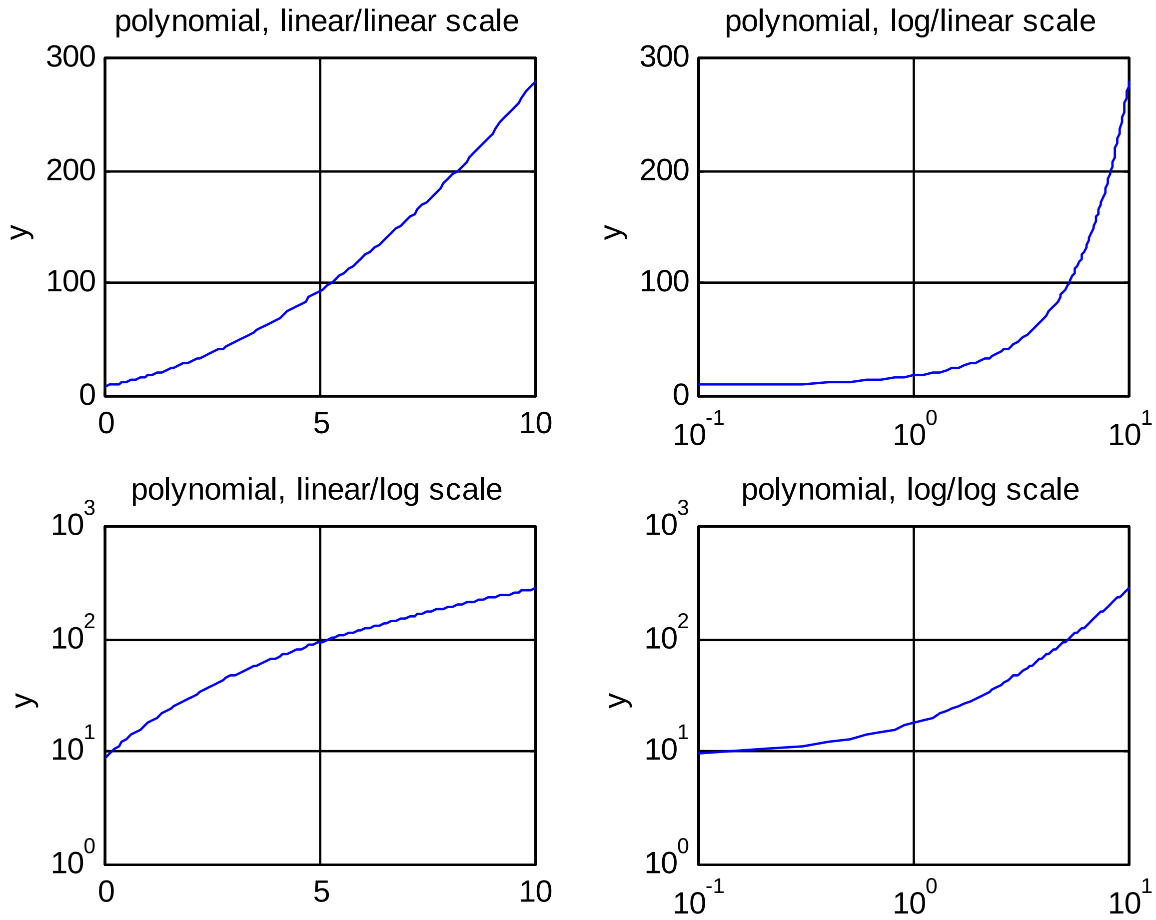
Plotting

- Example: (polynomial function)
  - Plot the polynomial using linear/linear, log/linear, linear/log, log/log scale

$$y = 2x^2 + 7x + 9$$

    - >>% generate te polynomial:
    - >>x=linspace(0,10,100);
    - >>y=2*x.^2+7*x+9;
    - >>% plotting the polynomial:
    - >>figure(1);
    - >>subplot(2,2,1),plot(x,y);
    - >>title('polynomial, linear/linear scale');
    - >>ylabel('y'),grid;
    - >>subplot(2,2,2),semilogx(x,y);
    - >>title('polynomial, log/linear scale');
    - >>ylabel('y'),grid;
    - >>subplot(2,2,3),semilogy(x,y);
    - >>title('polynomial, linear/log scale');
    - >>ylabel('y'),grid;
    - >>subplot(2,2,4),loglog(x,y);
    - >>title('polynomial, log/log scale');
    - >>ylabel('y'),grid;

Plotting


polynomial, linear/linear scale
polynomial, log/linear scale
polynomial, linear/log scale
polynomial, log/log scale
y
300
200
100
0
5
10
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}

Plotting

- Adding new curves to the exsiting graph
- Use the **hold** command to add lines/points to an existing plot
  - **hold on** – retain existing axes, add new curves to current axes.
  - **hold off** – release the current figure windows for new plots
- Grids and labels:

Command	**Description**
grid on	Add dashed grids lines at the tick marks
grid off	Removes grid lines (default)
Grid	Toggles grid status (off to on or on to off)
title('text')	Labels top of plot with text
xlabel('text')	Labels horizontal (x) axis with text
ylabel('text')	Labels vertical (y) axis with text
text(x,y,'text')	Adds text to location (x,y) on the current axes, where (x,y) is in units from the current plot

Programming

- Flow control and loops
- Simple **if** statement:
  - **if** *logical expression*
  - *commands*
  - **end**
- Example: (Nested)
  - **if** d < 50
  - count=count +1;
  - disp(d);
  - **if** b>d
  - b=0;
  - **end**
  - **end**
- Example: (**else** and **elseif** clauses)
  - **if** temperature >100
  - disp('Too hot – equipment malfunctioning.');
  - **elseif** temperature >90
  - disp('Normal operating range.');
  - **elseif** temperature > 75
  - disp('Below desired operating range.');
  - **else**
  - disp('Too cold – Turn off equipment.');
  - **end**

Programming

- The **switch** statement:
  - **switch** expression
  - **case** *test expression 1*
  - commands
  - **case** *test expression 2*
  - commands
  - **otherwise**
  - commands
  - **end**

- Example:

- **switch** interval
  - **case** 1
  - xinc = interval/10;
  - **case** 0
  - xinc = 0.1;
  - **otherwise**
  - disp('wrong value');
  - **end**

Programming

- Loops
- **for** loop
- **for** *variable = expression*
-     *commands*
- **end**
- **while** loop
- **while** *expression*
-     *commands*
- **end**

- **Example** (**for** loop):
- **for** t = 1: 5000
-     y(t) = sin (2*pi*t/10);
- **End**

- **Example (while** loop):
- **while** EPS>1
-     EPS=EPS/2;
- **end**

The **break** statement

break – is used to terminate the execution of the loop.

M-Files

- *Before, we have executed the commands in the command window. The more general way is to create a M-file.*
- The M-file is a text file that consists a group of MATLAB commands.
- MATLAB can open and execute the commands exactly as if they were entered at the MATLAB command window.
- To run the M-files, just type the file name in the command window. (make sure the current working directory is set correctly)

User-Defined Function

- Add the following command in the beginning of your m-file:
- **function** [output variables] = **function_name** (input variables);
  - Note: the function_name should be the same as your file name to avoid confusion.
- Calling your function:
  - A user-defined function is called by the name of the m-file, not the name given in the function definition.
  - Type in the m-file name like other pre-defined commands.
- Comments:
  - The first few lines should be comments, as they will be displayed if help is requested for the function name. the first comment line is reference by the lookfor command.

User-Defined Function

- Example (circle1.m)
  - function y = circle1(center,radius,nop,style)
  - % circle1 draws a circle with center defined as a vector 'center'
  - % radius as a scalar 'radius'. 'nop' is the number of points on the circle
  - % 'style' is the style of the point.
  - % Example to use: circle1([1 3],4,500, ':');
  - [m,n] = size(center);
  - if (~((m == 1) | (n == 1)) | (m == 1 & n == 1))
  -    error('Input must be a vector')
  - end
  - close all
  - x0=center(1);
  - y0=center(2);
  - t0=2*pi/nop;
  - axis equal
  - axis([x0-radius-1 x0+radius+1 y0-radius-1 y0+radius+1])
  - hold on
  - for i=1:nop+1
  -    pos1=radius*cos(t0*(i-1))+x0;
  -    pos2=radius*sin(t0*(i-1))+y0;
  -    plot(pos1,pos2,style);
  - end

User-Defined Function

- In command window:
  - >> help circle1

  - circle1 draws a circle with center defined as a vector 'center'
  - radius as a scalar 'radius'. 'nop' is the number of points on the circle
  - 'style' is the style of the point
  - Example to use: circle1([1 3],4,500,':');

- Example: plot a circle with center (3,1), radius 5 using 500 points and style '--':

  - >> circle1([3 1],5,500,'--');

- Result in the Figure window