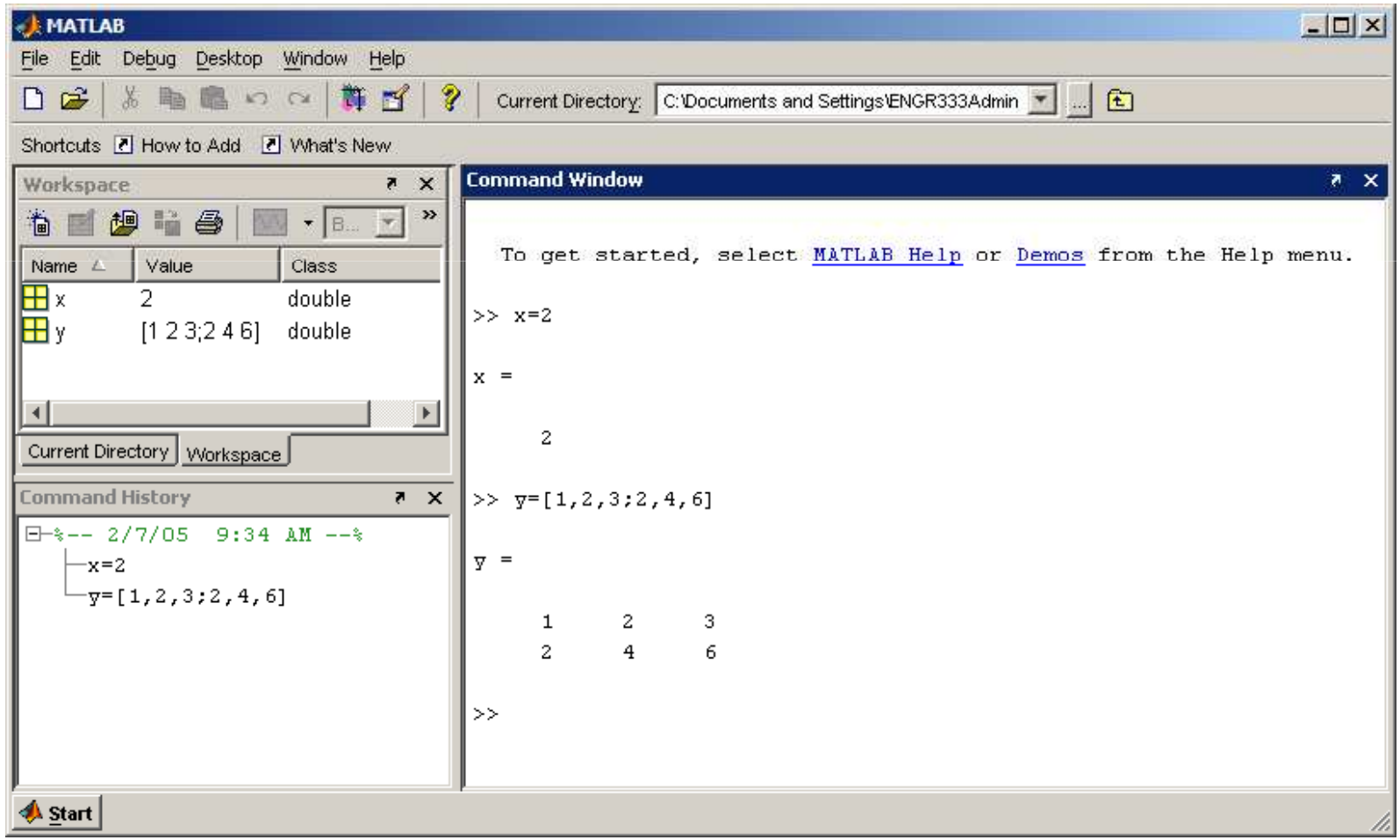
How to Use MATLAB

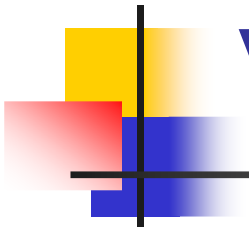
A Brief Introduction

MATLAB Working Environments



Some Useful Commands

- help % list all the topics
- clear % remove all the data in current session
- ; (semicolon) % prevent commands from outputing results
- % (percent sign) % comments line
- clc % clears the screen



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 =
        0    0.7854    1.5708    2.3562    3.1416
>> y = [ 0; 0.25*pi; 0.5*pi; 0.75*pi; pi ]
y =
        0
   0.7854
   1.5708
   2.3562
   3.1416
```

x is a row vector.

y is a column vector.

Vectors (con't…)

- 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)

start is the starting index, increment is the amount to add to each successive index, and end is the ending index. A shortened format (start : end) may be used if increment is 1.

- Example:

```
>> x(1:3)
ans =
         0    0.7854    1.5708
```

← 1st to 3rd elements of vector x

NOTE: MATLAB index starts at 1.

Vectors (con't…)

Some useful commands:

x = start:end	create row vector x starting with start, counting by one, ending at end
x = start:increment:end	create row vector x starting with start, counting by increment, ending at or before end
linspace(start,end,number)	create row vector x starting with start, ending at end, having number elements
length(x)	returns the length of vector x
y = x'	transpose of vector x
dot (x, y)	returns the scalar dot 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 operations to all elements of the array.

- Example:

```
>> f = [ 1 2; 3 4]
f =
     1     2
     3     4
>> g = 2*f – 1
g =
     1     3
     5     7
```

Each element in the array f is multiplied by 2, then subtracted by 1.

Array Operations (con't…)

- **Element-by-Element Array-Array Mathematics.**

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

- **Example:**

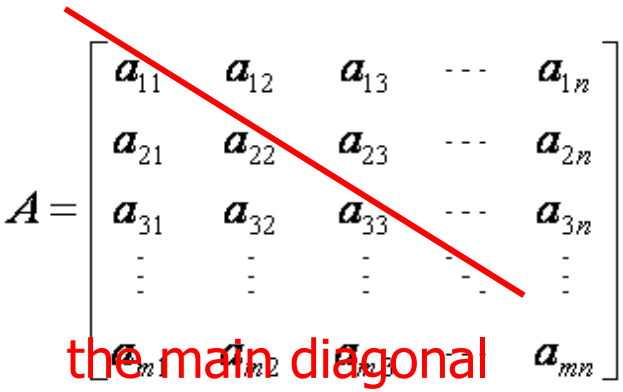
```
>> x = [ 1 2 3 ];
>> y = [ 4 5 6 ];
>> z = x .* y
z =
     4    10    18
```

Each element in x is multiplied by the corresponding element in y.

Matrices

- A Matrix array is two-dimensional, having both multiple rows and multiple columns, similar to vector arrays:
  - 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.

A is an m x n matrix.



•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 (con't…)

- Matrix Addressing:
  -- *matrixname(row, column)*
  -- **colon** may be 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
```

recall:

f =

1	2	3
4	5	6

h =

2	4	6
1	3	5

Matrices (con't…)

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 (con't…)

more commands

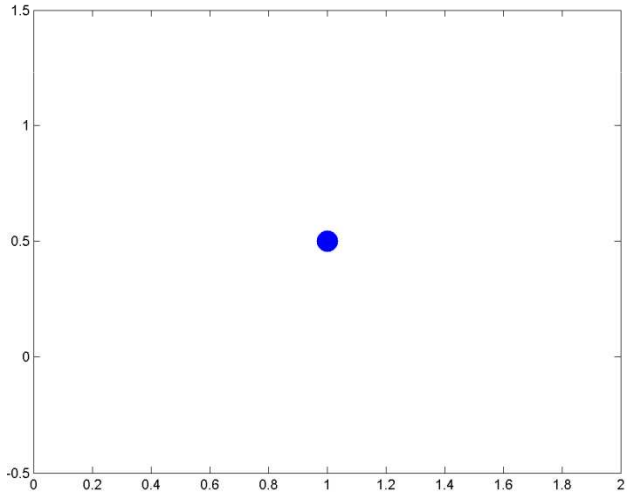
Transpose	B = A'
Identity Matrix	eye(n) ➔ returns an n x n identity matrix eye(m,n) ➔ returns an m x n matrix with ones on the main diagonal and zeros elsewhere.
Addition and subtraction	C = A + B C = A – B
Scalar Multiplication	B = αA, where α is a scalar.
Matrix Multiplication	C = A*B
Matrix Inverse	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.

A, B, C are matrices, and m, n, α are scalars.

Plotting

- For more information on 2-D plotting, type **help graph2d**
- Plotting a point:
  >> plot (*variablename, 'symbol')*

- Example : Complex number
  >> z = 1 + 0.5j;
  >> plot (z, '.')



Plotting (con't…)

- Plotting Curves:
  - **plot (x,y)** – generates 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 and y using a logarithmic scale for x and a linear scale for y
  - **semilogy (x,y)** – generate a plot of the values of x and y using a linear scale for x and a logarithmic scale for y.
  - **loglog(x,y)** – generate a plot of the values of x and y using logarithmic scales for both x and y
- Subplots:
  - **subplot (m, n, p)** – m by n grid of windows, with p specifying the current plot as the p^{th} window

Plotting (con't…)

- **Example: (polynomial function)**

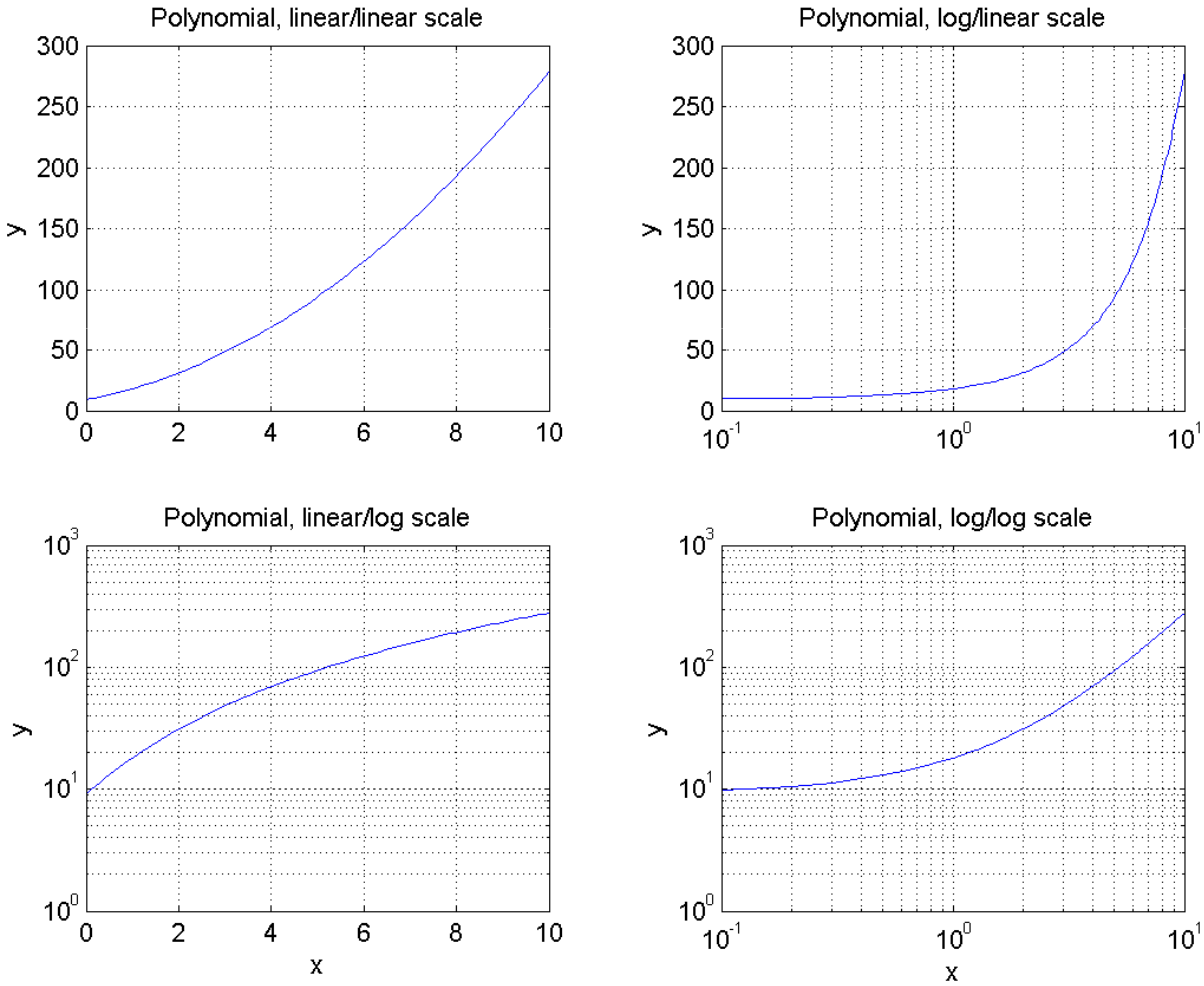
  plot the polynomial using linear/linear scale, log/linear scale, linear/log scale, & log/log scale:

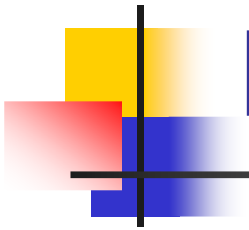
  $$y = 2x^2 + 7x + 9$$

```
% Generate the 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');
xlabel('x'), ylabel ('y'), grid;
subplot (2,2,4), loglog (x,y);
title ('Polynomial, log/log scale');
xlabel('x'), ylabel ('y'), grid;
```

Plotting (con't…)





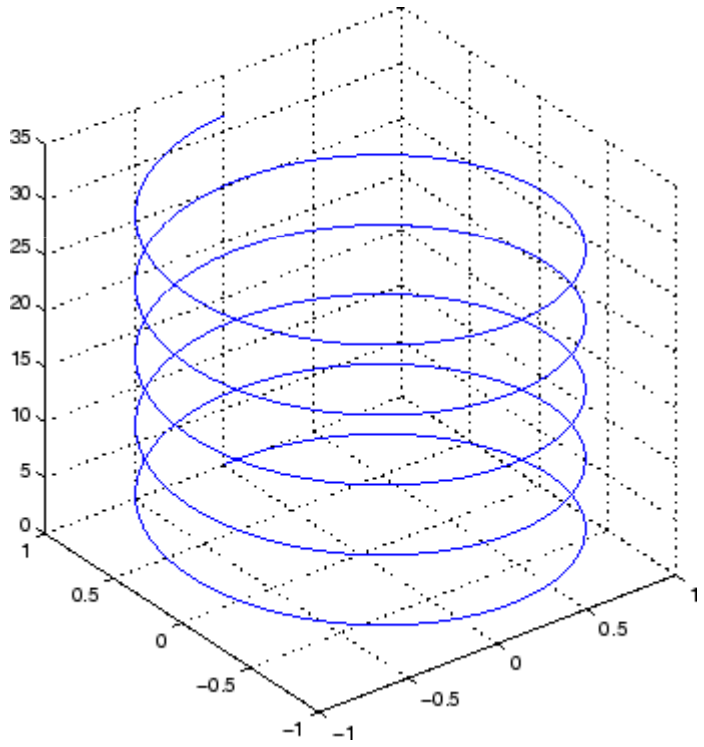
Plotting (con't…)

- Adding new curves to the existing graph:
- Use the **hold** command to add lines/points to an existing plot.
  - hold on – retain existing axes, add new curves to current axes. Axes are rescaled when necessary.
  - hold off – release the current figure window for new plots
- Grids and Labels:

Command	**Description**
grid on	Adds 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 in quotes
xlabel ('text')	labels horizontal (x) axis with text is quotes
ylabel ('text')	labels vertical (y) axis with text is quotes
text (x,y,'text')	Adds text in quotes to location (x,y) on the current axes, where (x,y) is in units from the current plot.

Plot3

```
t = 0:pi/50:10*pi;
plot3(sin(t),cos(t),t);
grid on
axis square
```



Flow Control

- 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 ('Below desired operating range.')
else
    disp ('Too cold – turn off equipment.')
end
```

Flow Control (con't…)

- The **switch** statement:

switch *expression*
    case *test expression 1*
        commands
    case *test expression 2*
        commands
    otherwise
        commands
end

- **Example:**

```
switch interval < 1
    case 1
      xinc = interval /10;
    case 0
      xinc = 0.1;
end
```

Loops

- **for** loop

  for *variable = expression*

  commands

  end

- **while** loop

  while *expression*

  commands

  end

- the **break** statement

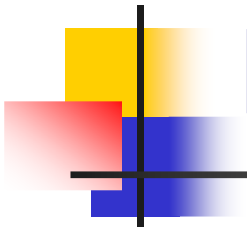
  break – is used to terminate the execution of the loop.

•**Example (for loop):**

```
for t = 1:5000
        y(t) = sin (2*pi*t/10);
end
```

•**Example (while loop):**

```
EPS = 1;
while ( 1+EPS) >1
        EPS = EPS/2;
end
EPS = 2*EPS
```



M-Files

- The M-file is a text file that consists a group of MATLAB commands.

  All MATLAB commands are M-files.

User-Defined Function

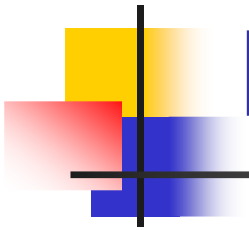
- Add the following command in the beginning of your m-file:

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
  -- 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.



Random Variable

- randn
- randi
- rand

Random Variable

```
v=25;                                %variance
m=10;                                %mean
x=sqrt(v)*randn(1, 1000) + m*ones(1, 1000);
figure;
plot (x);
grid;
xlabel ('Sample Index');
ylabel ('Amplitude');
title ('One thousands samples of a Gaussian random
    variable(mean=10, standard deviation=5)');
```

Exp2-Random Variable