18AIS101J-INTRODUCTION TO MATLAB FOR ARTIFICIAL INTELLIGENCE

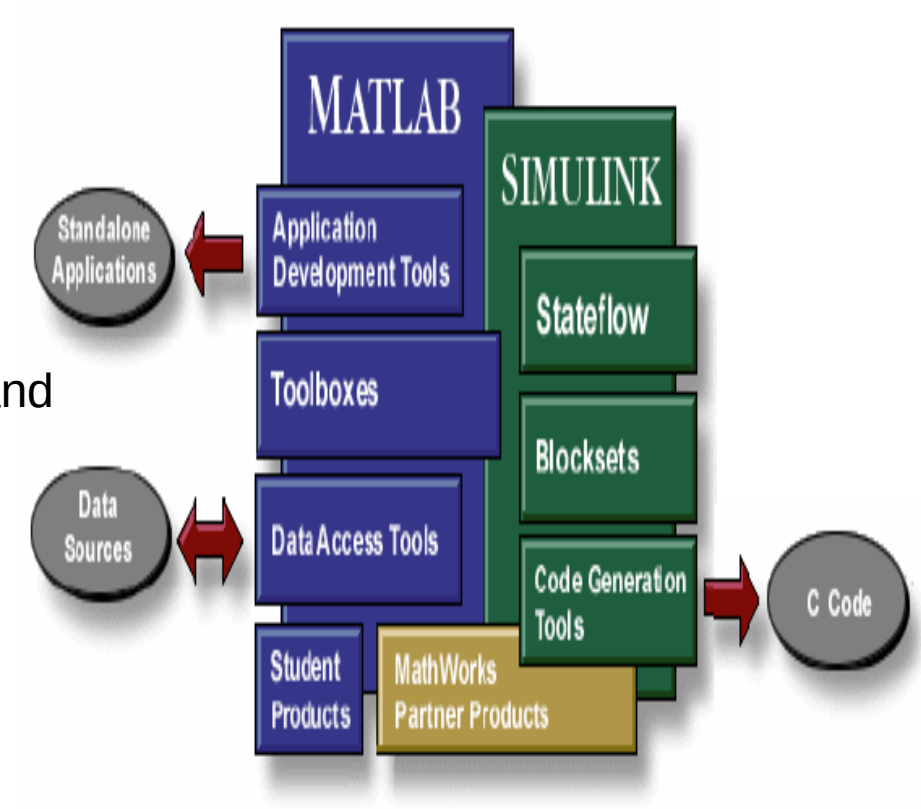
What is MATLAB?

- MATLAB(MATrix LABoratory)
- A High level programming language .
- Assembly/ Machine coding: Low level
- C,C++, Java, Javascript, Python,R…:High level (Interpreted Language)
- MATLAB, Basic,Javascript,Python, R…:High level(Interpreted Language)
- An interactive technical computing environment
- Algorithm Development
- Data Analysis
- Numerical Computation

MATLAB Product Family

Accelerating engineering design and discovery

- MATLAB for algorithm development and analysis
- Simulink for system-level design
- Stateflow –Simulation of event driven systems to complement Simulink



MATLAB Product Family: MATLAB

➢**Toolboxes for Modeling, Analysis, and Computation**

- Specific functionality for data analysis, modelling , optimization, math, and other capabilities.

➢**Toolboxes for Data Input/Output**

- MATLAB extensions for I/O of live and archived data

With general-purpose and application-specific sources

➢**MATLAB Deployment Tools**

- Deploy algorithms and applications to reports, web servers, and standalone applications

The MathWorks Product Family:Simulink

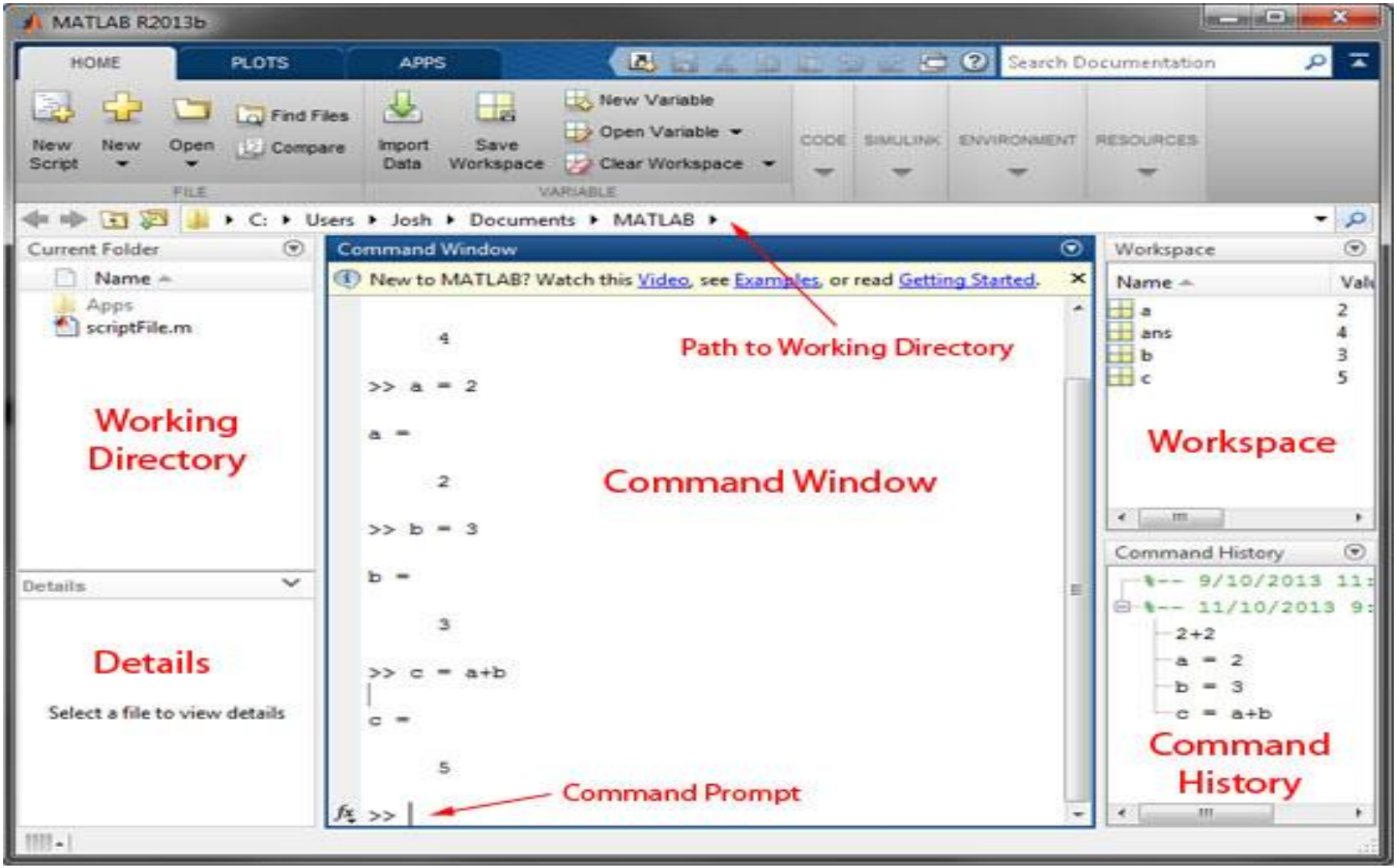
- **Blocksets**
- Block libraries for fixed-point, visualization, DSP, communications, and more.
- **Stateflow**
- Model and simulate reactive systems, state machines, and logic constructs.
- **Automatic Code Generation**
- For rapid prototyping, hardware-in-the-loop, and production embedded software.
- **Real-Time Systems**
- Target systems for rapid prototyping ,HIL and deployment.

Latest Release Highlights

- R2012b – new Desktop features (Toolstrip interface that replaced menus and toolbars, Apps gallery presenting apps from the MATLAB product family, redesigned Help system), command line suggestions; new Simulink Editor, smart signal routing and simulation tools
- R2013b – new types of data (table data container, categorical array)
- R2014a – new way to display Command History window (pop-up window rather than static window)
- R2014b – new graphics system, new types of data (datetime, duration, and calendarDuration), suggested corrections for syntax errors in the Command Window, packaging a sharing tool, big data analysis tools (datastore and others)
- R2016a – Live Editor to create and run live scripts with embedded output), App Designer

- R2016b – new types of data (timetable data container, timeseries objects, string array), working with missing data (fillmissing) and big data (tall)
- R2017a – MATLAB Online to use MATLAB through the web browser, working with outliers (filloutliers and others)
- R2017b – MATLAB Drive providing a common free cloud-based storage of 250Mb), plotting in geographic coordinates, wordcloud function, contextual hints for function arguments in Live Editor
- R2018a – improved graphic (axes, legend)
- R2018b – new plotting functions (xline, yline, geoaxes, stackedplot, scatterhistogram and others), axes toolbar, Deep Learning Toolbox
- R2019a – new tabular data reading functions (readmatrix and others), parallelplot, graphics export, object detection using you-only-look-once (YOLO) v2 detectors, Reinforcement Learning Toolbox
- R2019b – Simulink Toolstrip and other tools, Git integration with MATLAB, map-based data visualization, Live Editor Tasks, function argument validation, Navigation, Robotics System and ROS Toolboxes

MATLAB ENVIRONMENT



TYPES OF FILES

There are three different types of files in the MATLAB:

1.M-files

2.MAT-files

3.MEX-files

M-files

M-files are standard ASCII text files, with a (.m) extension to the filename. Any program written in a MATLAB editor is saved as M-files. These files are further classified as

- **Script files**

A script file consists of a sequence of normal MATLAB statements. If the file has the filename, say, **rotate.m**, then the MATLAB command **rotate** will cause the statements in the file to be executed. Variables in a script file are global and will change the value of variables of the same name in the environment of the current MATLAB session.

- **Function files**

Function files provide extensibility to MATLAB. You can create new functions specific to your problem which will then have the same status as other MATLAB functions. Variables in a function file are by default local. However, you can declare a variable to be global if you wish

MAT-files

MAT-files are binary MATLAB files with an (.mat) extension that store workspace variables. They are created by MATLAB when data is saved .

MEX-files

A MEX file is a function, created in MATLAB, that calls a C/C++ program or a Fortran subroutine. A MEX function behaves just like a MATLAB script or function.

To call a MEX function, use the name of the MEX file, without the file extension. The MEX file contains only one function or subroutine. The calling syntax depends on the input and output arguments defined by the MEX function.

Benefits of MATLAB

➡ It is easy to use due to its command line interface and file oriented structure.
➡MATLAB is platform independent and hence it can be installed on different Operating Systems such as Windows, Vista, Linux and Macintosh.
➡MATLAB has huge built-in library of functions for many predefined tasks. These functions are available as part of various toolkits which include signal processing, image processing, communications, control systems, neural networks etc.
➡It offers plotting and imaging related commands which are independent of devices.
➡MATLAB provides tool to develop GUI based applications.
➡Errors are easier to fix as it is interpreted language.
➡Matrix operations are easier and quick to perform. MATLAB can handle and manipulate large data sets. Hence it is used to develop and code many of algorithms quickly.
➡MATLAB is inexpensive software.

CHARACTER SET

A set of characters used to construct words, statements, etc.,

1.Alphabets

- supports all the alphabets from the English language. Lower and upper case letters together support 52 alphabets.
- lower case letters - a to z
- UPPER CASE LETTERS - A to Z

2.Numerals

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

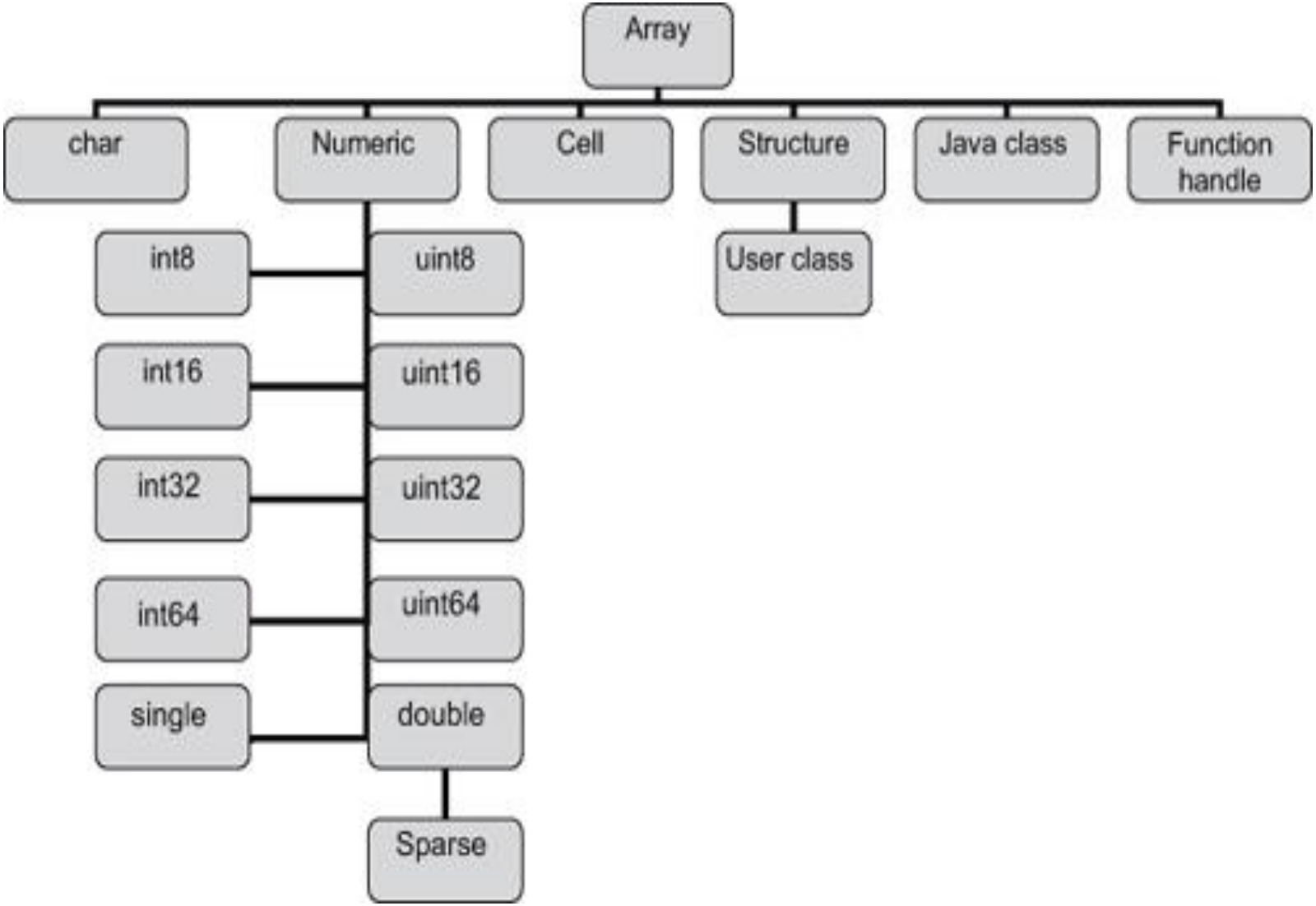
3.Special Characters

A special character is a character that is not an alphabetic or numeric character. Punctuation marks and other symbols are examples of special characters.

4.White space characters

Space, tab, line feed (newline), carriage return, form feed, and vertical tab characters are called "white-space characters"

DATA TYPES


Array
char
Numeric
Cell
Structure
Java class
Function handle
int8
uint8
int16
uint16
int32
uint32
int64
uint64
single
double
Sparse
User class

The basic data type (also called a class) in MATLAB is the array or matrix. There are 15 fundamental data types in MATLAB. Each of these data types is in the build of a matrix or array.

Datatype	Example
int8, uint8, int16, uint16, int32, uint32, int64, uint64	uint16(65000)
single	3 * 10^38
double	3 * 10^300 5 + 6i
logical	magic(4) > 10
char	'Hello'

cell array	a{1,1} = 12;
	a{1,2} = 'Red';
	a{1,3} = magic(4);

structure	a.day = 12;
	a.color = 'Red';
	a.mat = magic(3);

function handle	@sin
user class	polynom([0 -2 -5])
Java class	java.awt.Frame

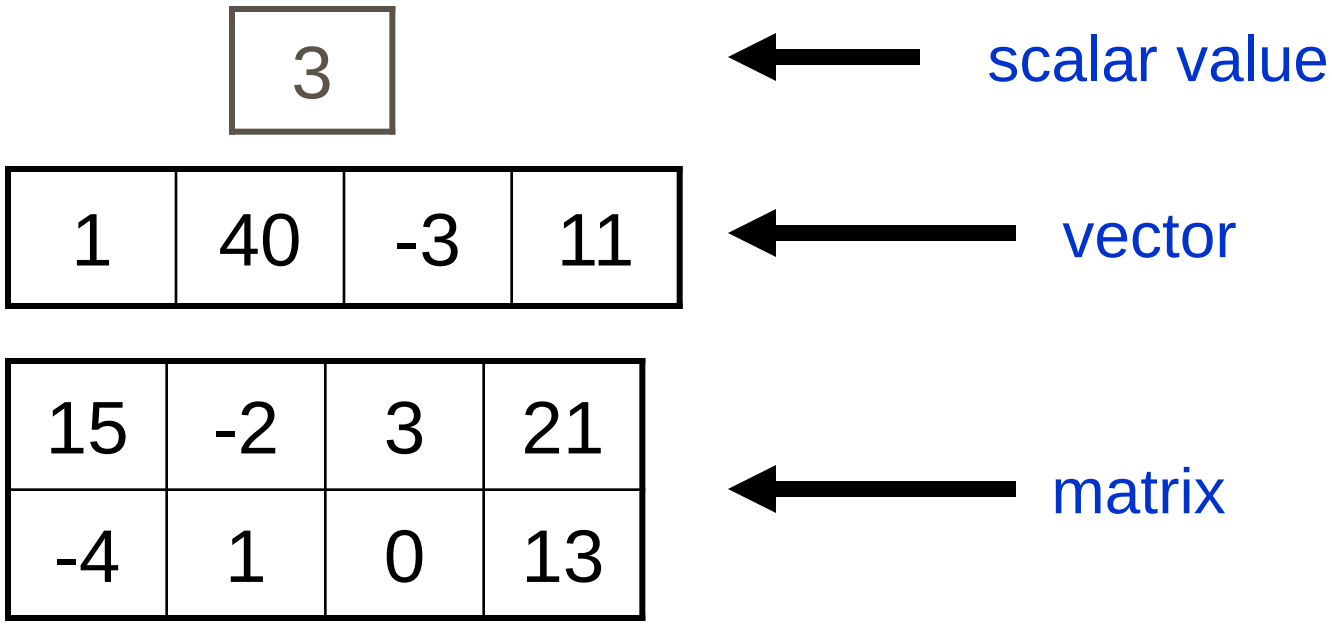
Review

- A program can be input
  - Typing commands using the command line (lines starting with "»" on the MATLAB desktop)
  - As a series of commands using a file (a special file called M-file)
- If a command is followed by a semicolon (;), result of the computation is not shown on the command window

2.1 Variables and Arrays

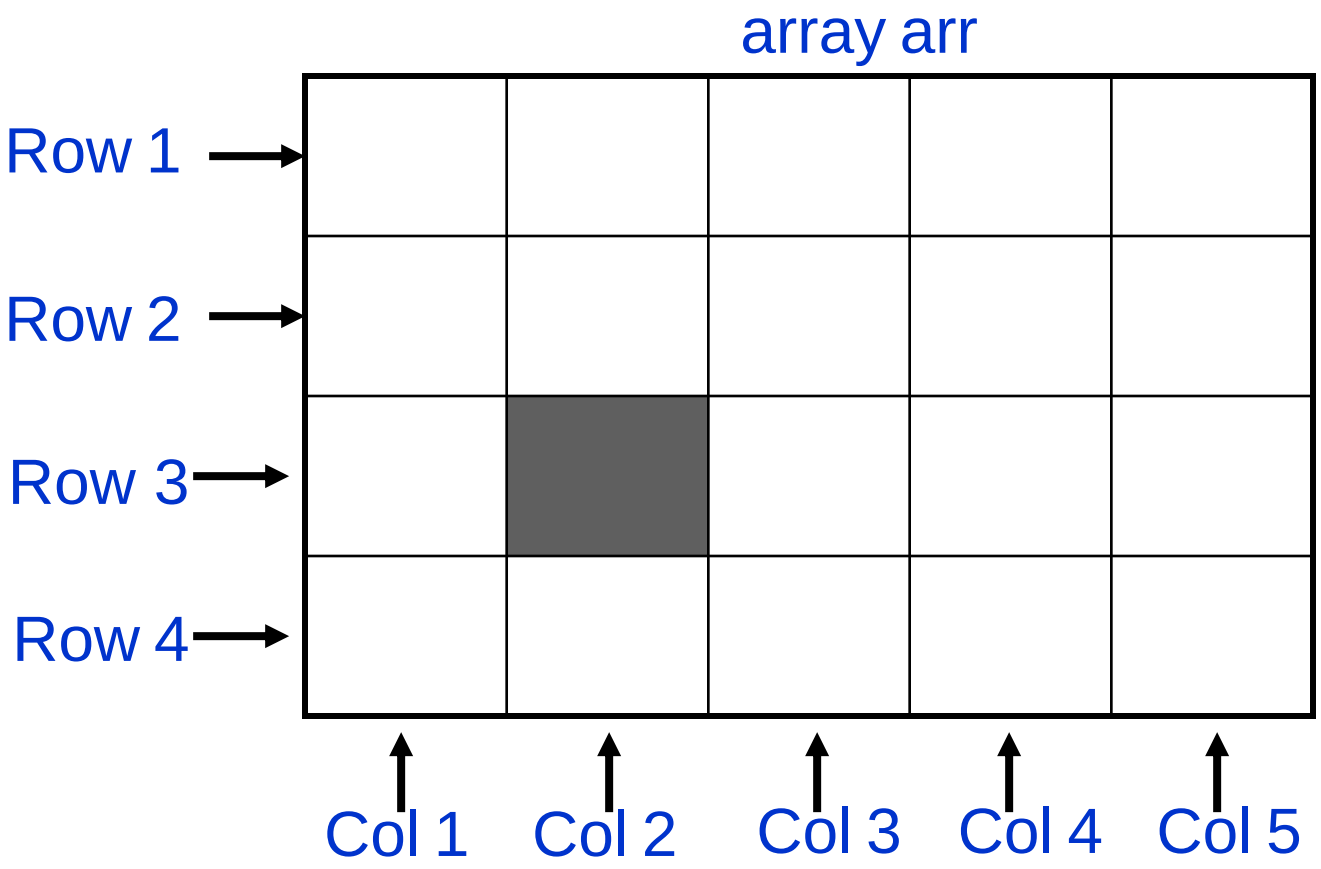
- The fundamental unit of data is array

  An array is a collection of data values organized into rows and columns.



2.1 Variables and Arrays

- Data values within an array are accessed by including the name of the array followed by subscripts in parentheses.



The shaded element in this array would be addressed as arr(3, 2).

Array	Size
$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$	This is a 3*2 matrix, containing 6 elements.
$B = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}$	This is a 1*4 matrix, known as a row vector, containing 4 elements.
$C = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$	This is a 3*1 matrix, known as a column vector, containing 3 elements.

A(2, 1) = 3, and C(2) = 2

2.1 Variables and Arrays

- Arrays can be classified as either vectors or matrices.
- The term “vector” is usually used to describe an array with only one dimension;
- The term “matrix” is usually used to describe an array with two or more dimensions.
- The size of an array:

2.1 Variables and Arrays

- Variable is a name given to a reserved location in memory
  - class_code = 111;
  - number_of_students = 65;
  - name = ‘Chongqing University of Posts and Telecommunications';
  - radius = 5;
  - area = pi * radius^2;

2.1 Variables and Arrays

- MATLAB variable names
  - must begin with a letter
  - can contain any combination of letters, numbers and underscore (_)
  - must be unique in the first 31 characters
- MATLAB is case sensitive

  "name", "Name" and "NAME" are considered different variables
- Never use a variable with the same name as a MATLAB command
- Naming convention: use lowercase letters

2.1 Variables and Arrays

- Use meaningful names for variables
- number_of_students
- exchange_rate

2.1 Variables and Arrays

- data dictionary:
- The most common types of MATLAB variables are double and char:
- Double: consist of scalars or array of 64-bit double-precision floating-point numbers.10^{-308}~10^{308}
- Char:consist of scalars or array of 16-bit values.

2.1 Variables and Arrays

- MATLAB: weakly typed language
- C: strongly typed language

2.2 Initializing Variables

- Three common ways:

1. Initialization using assignment statements;
2. Input data from the keyboard;
3. Read data from a file.

2.2 Initializing Variables

- 2.2.1 Initialization using assignment statements
  - General form: var = expression
    - x = 5

      x =

          5
    - y = x + 1

      y =

          6
    - vector = [1 2 3 4]

      vector =

          1   2   3   4

2.2 Initializing Variables

```
» matrix = [ 1 2 3; 4 5 6 ]
  matrix =
       1     2     3
       4     5     6
» matrix = [ 1 2 3; 4 5 ]
  ??? Error
» a = [ 5 (2+4) ]
  a =
       5     6
```

2.2 Initializing Variables

- 2.2.2 Initialization using shortcut statements
  - colon operator → first:increment:last

```
» x = 1:2:10
x =
     1     3     5     7     9
» y = 0:0.1:0.5
y =
     0    0.1    0.2    0.3    0.4    0.5
```

2.2 Initializing Variables

- Transpose operator → '

```
>> u = [ 1:3 ]'
  u =
       1
       2
       3
>> v = [ u u ]
  v =
       1     1
       2     2
       3     3
```

```
>> v = [ u'; u' ]
  v =
       1     2     3
       1     2     3
```

2.2 Initializing Variables

2.2.3 Initialization using built-in functions

- zeros()
  - `x = zeros(2)`

```
x =
     0     0
     0     0
```

  - `z = zeros(2,3)`

```
z =
     0     0     0
     0     0     0
```

- ones(), size(), length()

`y = zeros(1,4)`

```
y =
     0     0     0     0
```

`t = zeros( size(z) )`

```
t =
     0     0     0
     0     0     0
```

2.2 Initializing Variables

2.2.4 Initialization using keyboard input

- input()
  - value = input('Enter an value: ')

    Enter an value: 1.25

    value =

        1.2500
  - name = input('What is your name: ', 's')

    What is your name: Selim

    name =

    Selim

2.3 Multidimensional Arrays

- Initialization
  - » c(:,:,1)=[1 2 3; 4 5 6];
    c(:,:,2)=[7 8 9; 10 11 12];
  - » whos

```
  Name      Size                Bytes  Class
  c         2x3x2                  96  double array
```

  - » c

```
c(:,:,1) =
     1     2     3
     4     5     6
c(:,:,2) =
     7     8     9
    10    11    12
```

2.3 Multidimensional Arrays

- Storing in Memory
  - Two-dimensional : Column major order
  - Multidimensional Arrays : array subscript incremented

    first, second, third, …

    (1,1,1), (2,1,1), (1,2,1), (2,2,1), (1,1,2), (2,1,2), (1,2,2), (2,2,2)
- Accessing Multidimensional arrays with a Single Subscript
  - The elements will be accessed in the order in which they were allocated in memory
    - c(5)

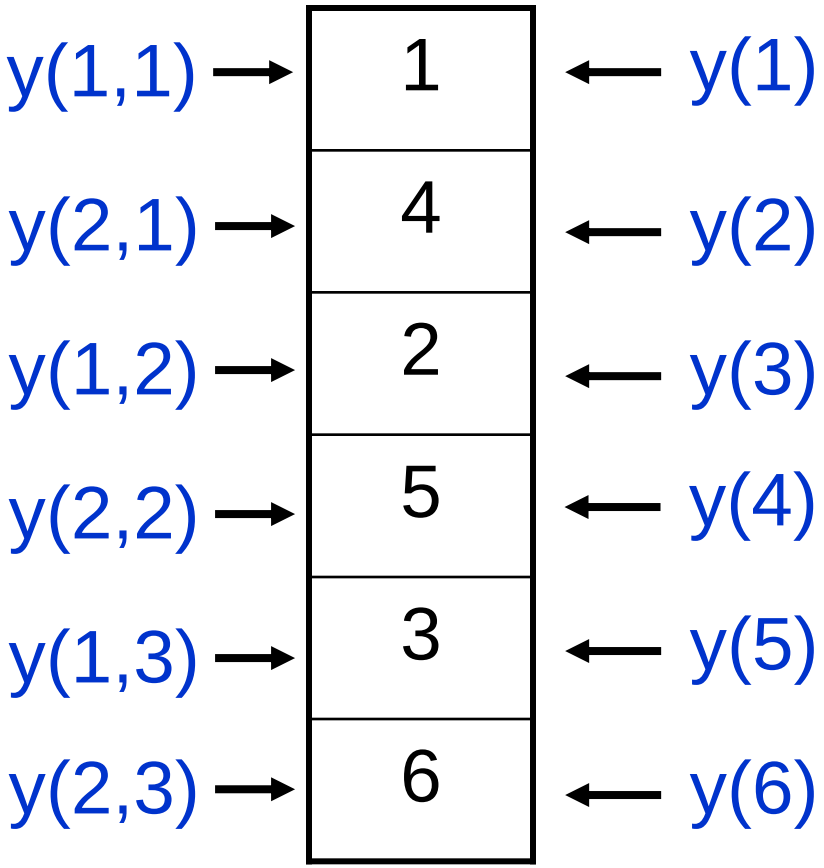
2.4 Subarrays

- Array indices start from 1

```
» x = [ -2 0 9 1 4 ];
» x(2)
ans =
     0
» x(4)
ans =
     1
» x(8)
??? Error
» x(-1)
??? Error
```

2.4 Subarrays

```
» y = [ 1 2 3; 4 5 6 ];
  » y(1,2)
    ans =
        2
  » y(2,1)
    ans =
        4
  » y(2)
    ans =
        4
```



(column major order)

2.4 Subarrays

```
» y = [ 1 2 3; 4 5 6 ];
» y(1,:)
ans =
     1     2     3
» y(:,2)
ans =
     2
     5
» y(2,1:2)
ans =
     4     5
» y(1,2:end)
ans =
     2     3
» y(:,2:end)
ans =
     2     3
     5     6
```

2.4 Subarrays

```
» x = [ -2 0 9 1 4 ];
  » x(2) = 5
    x =
        -2     5     9     1     4
  » x(4) = x(1)
    x =
        -2     5     9    -2     4
  » x(8) = -1
    x =
        -2     5     9    -2     4     0     0    -1
```

2.4 Subarrays

```
» y = [ 1 2 3; 4 5 6 ];
» y(1,2) = -5
y =
     1    -5     3
     4     5     6
» y(2,1) = 0
y =
     1    -5     3
     0     5     6
» y(1,2:end) = [ -1 9 ]
y =
     1    -1     9
     0     5     6
```

2.4 Subarrays

```
» y = [ 1 2 3; 4 5 6; 7 8 9 ];
  » y(2:end,2:end) = 0
    y =
         1     2     3
         4     0     0
         7     0     0
  » y(2:end,2:end) = [ -1 5 ]
    ??? Error
  » y(2,[1 3]) = -2
    y =
         1     2     3
        -2     0    -2
         7     0     0
```

2.5 Special Values

- pi: π value up to 15 significant digits
- i, j: sqrt(-1)
- Inf: infinity (such as division by 0)
- NaN: Not-a-Number (such as division of zero by zero)
- clock: current date and time as a vector
- date: current date as a string (e.g. 20-Feb-2008)
- eps: epsilon
- ans: default variable for answers

2.6 Displaying Output Data

2.6.1 Changing the data format

- value = 12.345678901234567

format short → 12.3457

long → 12.34567890123457

short e → 1.2346e+001

long e → 1.234567890123457e+001

rat → 1000/81

compact

loose

2.6 Displaying Output Data

2.6.2 The disp(array) function

```
disp( 'Hello' );
```

Hello

```
disp(5);
```

5

```
disp( [ ‘重庆邮电大学’ ] );
```

重庆邮电大学

```
name = 'Selim';
disp( [ 'Hello ' name ] );
```

Hello Selim

2.6 Displaying Output Data

2.6.2 The num2str() and int2str() functions

```
» d = [ num2str(25) '-Feb-' num2str(2008) ];
» disp(d);
  25-Feb-2008
» x = 23.11;
» disp( [ 'answer = ' num2str(x) ] );
  answer = 23.11
» disp( [ 'answer = ' int2str(x) ] );
  answer = 23
```

2.6 Displaying Output Data

2.6.3 The fprintf(*format*, *data*) function

- %d integer
- %f floating point format
- %e exponential format
- \n skip to a new line
- \g either floating point format or exponential format,whichever is shorter.

Limitation:*it only displays the real portion of a complex value.*

2.6 Displaying Output Data

```
» fprintf( 'Result is %d', 3 );
  Result is 3
» fprintf( 'Area of a circle with radius %d is %f', 3, pi*3^2 );
  Area of a circle with radius 3 is 28.274334
» x = 5;
» fprintf( 'x = %3d', x );
  x =   5
» x = pi;
» fprintf( 'x = %0.2f', x );
  x = 3.14
» fprintf( 'x = %6.2f', x );
  x =   3.14
» fprintf( 'x = %d\ny = %d\n', 3, 13 );
  x = 3
  y = 13
```

2.7 Data Files

- save *filename var1 var2* ...
  - » save homework.mat x y → binary

  Including: name,type,size,value

  But annot be read by other programs

  - » save x.dat x –ascii → ascii

  But name and type are lost; and the resulting data file will be much larger

2.7 Data Files

- load *filename*
  - load filename.mat → binary
  - load x.dat –ascii → ascii

2.8 Scalar & Array Operations

- variable_name = expression;
  - addition a + b → a + b
  - subtraction a - b → a - b
  - multiplication a x b → a * b
  - division a / b → a / b
  - exponent a^b → a ^ b

2.8.1 Scalar & Array Operations

- Scalar-matrix operations

```
» a = [ 1 2; 3 4 ]
a =
     1     2
     3     4
» 2 * a
ans =
     2     4
     6     8
» a + 4
ans =
     5     6
     7     8
```

- Element-by-element operations

```
» a = [ 1 0; 2 1 ];
» b = [ -1 2; 0 1 ];
» a + b
ans =
     0     2
     2     2
» a .* b
ans =
    -1     0
     0     1
```

2.8.2 Array & Matrix Operations

- Matrix multiplication: C = A * B
- If
  - A is a p-by-q matrix
  - B is a q-by-r matrix

  then
  - C will be a p-by-r matrix where

$$C(i, j) = \sum_{k=1}^{q} A(i,k)B(k, j)$$

2.8.2 Array & Matrix Operations

- Matrix multiplication: C = A * B

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \qquad B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix}$$

$$C = \begin{bmatrix} (a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31}) & (a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32}) \\ (a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31}) & (a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32}) \\ (a_{31}b_{11} + a_{32}b_{21} + a_{33}b_{31}) & (a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32}) \end{bmatrix}$$

2.8.2 Array & Matrix Operations

- Examples

```
» a = [ 1 2; 3 4 ]

a =

     1     2
     3     4

» b = [ -1 3; 2 -1 ]

b =

    -1     3
     2    -1

» a .* b

ans =

    -1     6
     6    -4

» a * b

ans =

     3     1
     5     5
```

2.8.2 Array & Matrix Operations

- Examples

```
» a = [ 1 4 2; 5 7 3; 9 1 6 ]
a =
     1     4     2
     5     7     3
     9     1     6
» b = [ 6 1; 2 5; 7 3 ]
b =
     6     1
     2     5
     7     3
```

```
» c = a * b
c =
    28    27
    65    49
    98    32
» d = b * a
??? Error using ==> *
Inner matrix
dimensions must
agree.
```

2.8.2 Array & Matrix Operations

- Identity matrix: I
- A * I = I * A = A
- Examples

```
» a = [ 1 4 2; 5 7 3; 9 1 6 ];
» I = eye(3)
I =
     1     0     0
     0     1     0
     0     0     1
» a * I
ans =
     1     4     2
     5     7     3
     9     1     6
```

2.8.2 Array & Matrix Operations

- Inverse of a matrix: A^{-1}
- $AA^{-1} = A^{-1}A = I$
- Examples

```
» a = [ 1 4 2; 5 7 3; 9 1 6 ];
» b = inv(a)
b =
   -0.4382    0.2472    0.0225
    0.0337    0.1348   -0.0787
    0.6517   -0.3933    0.1461
» a * b
ans =
    1.0000         0         0
    0.0000    1.0000         0
         0   -0.0000    1.0000
```

2.8.2 Array & Matrix Operations

- Matrix left division: C = A \ B
- Used to solve the matrix equation AX = B where $X = A^{-1}B$
- In MATLAB, you can write
  - >> x = inv(a) * b
  - >> x = a \ b

    (second version is recommended)

2.8.2 Array & Matrix Operations

- Example: Solving a system of linear equations

$$\begin{aligned} 4x - 2y + 6z &= 8 \\ 2x + 8y + 2z &= 4 \\ 6x + 10y + 3z &= 0 \end{aligned} \quad \Rightarrow \quad \begin{bmatrix} 4 & -2 & 6 \\ 2 & 8 & 2 \\ 6 & 10 & 3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 8 \\ 4 \\ 0 \end{bmatrix}$$

```
» A = [ 4 -2 6; 2 8 2; 6 10 3 ];
» B = [ 8 4 0 ]';
» X = A \ B
X =
   -1.8049
    0.2927
    2.6341
```

2.8.2 Array & Matrix Operations

- Matrix right division: C = A / B
- Used to solve the matrix equation XA = B where $X = BA^{-1}$
- In MATLAB, you can write
  - `>> x = b * inv(a)`
  - `>> x = b / a`

    (second version is recommended)

2.9 Hierarchy of Operations

- x = 3 * 2 + 6 / 2
  - x = ?
- Processing order of operations is important
  1. parenthesis (starting from the innermost)
  2. exponentials (left to right)
  3. multiplications and divisions (left to right)
  4. additions and subtractions (left to right)
  - x = 3 * 2 + 6 / 2

    x = 9

2.9 Hierarchy of Operations

- Example p49(2.2)

- You should always ask yourself:
- will I easily understand this expression if I come back to it in six months?
- Can another programmer look at my code and easily understand what I am doing?
- *Use extra parentheses*
- *(2+4)/2*

2.10 Built-in Functions

- General form: result = function_name(input);
  - abs, sign
  - log, log10, log2
  - exp
  - sqrt
  - sin, cos, tan
  - asin, acos, atan
  - max, min
  - round, floor, ceil, fix
  - mod, rem
- help function_name

2.10 Built-in Functions

- Optional Results

  Can return more than one result. For example:

  » [maxval index] = max ([1 -5 6 -3])

  maxval = 6

  index = 3

- Using MATLAB Function with Array Input
  - You can call built-in functions with array inputs
  - The function is applied to all elements of the array
  - The result is an array with the same size as the input array

2.10 Built-in Functions

- Examples:
  - » x = [3 -2 9 4 -5 6 2];
  - » abs(x)

    ans =

    3 2 9 4 5 6 2
  - » sin([0 pi/6 pi/4 pi/3 pi/2])

    ans =

    0 0.5000 0.7071 0.8660 1.0000
  - » a = 1:5;
  - » log(a)

    ans =

    0 0.6931 1.0986 1.3863 1.6094
  - » sqrt(-2)

    ans =

    0 + 1.4142i

Example:

1. $$z_3 = \frac{10^{0.7a} - e^{0.5a}}{2} \sin(a + 3)$$

a=-3.0,-2.9,-2.8,…2.8,2.9,3.0

U=[0.89,1.20,3.09,4.27,7.71,8.99,7.92,9.70,10.41]
i=[0.028,0.040,0.100,0.145,0.118,0.258,0.299,0.257,0.308,0.345]

2.11 Introduction to Plotting

MATLAB's extensive,device-independent plotting capabilities are one of its most powerful features.

For example:

```
x=0:1:10;
y=x.^2-10.*x+15;
Plot(x,y)
```

2.11.1 Using Simple xy Plots

- Functions:
- title
- xlabel and ylabel
- grid on and grid off

For example:

```
x=0:1:10;
y=x.^2-10.*x+15;
plot(x,y);
title('plot of y=x.^2-10.*x+15')
xlabel('x')
ylabel('y')
grid on
```

2.11.2 Printing a Plot

- print <options> <filename>

(1)if no filename:

(2)if a filename is specified:

- print -dtiff my-image.tif
- file/export

.tiff:an image in tagged file format,can be imported into all of the important word processors on PC,Mzc,and UNIX platforms.

2.11.3 Multiple Plots

For example:

Plot function f(x)=sin(x) and its deribative on the same plot

$$\frac{d}{dt}\sin 2x = 2\cos 2x$$

- x=0:pi/100:2*pi;
- y1=sin(2*x);
- y2=2*cos(2*x);
- plot(x,y1,x,y2)

2.11.4 Line Color,Line Style,and Legends

These traits can be selected using an attribute character string after the x and y vectors in the plot function

The attribute character string can have up to three components:
First components specifying the color of the line;
The second components specifying the style of the marker;
The last components specifying the style of the line.

For example:

- **x=0:1:10;**
- **y=x.^2-10.*x+15;**
- **plot(x,y,'r-',x,y,'bo');**

- Legends can be created with the legend function.

legend(‘string1’,’ string2’…,pos

For example:

```
x=0:pi/100:2*pi;
y1=sin(2*x);
y2=2*cos(2*x);
plot(x,y1,'k-',x,y2,'b-');
title('plot of f(x)=sin(x) and its deribative')
xlabel('x')
ylabel('y')
Legend('f(x)','d/dx f(x)')
grid on
```

2.11.5 Logarithmic Scales

- 1. plot: linear axes;
- 2.semilogx:x-- Logarithmic axes
- y-- linear axes
- 3. semilogy:y-- Logarithmic axes
- x-- linear axes
- 4.loglog:both x and y data on logarithmic axes

For example:

Plot function $y = x\sin x$ **and** $s = \int_0^x (x\sin x)dx$ **on the same plot** $[0,4]$

- clf;dx=0.1;x=0:dx:4;y=x.*sin(x);
- s=cumtrapz(y)*dx;
- a=plotyy(x,y,x,s,'stem','plot');
- text(0.5,1.5,'\fontsize{14}\ity=xsinx')
- sint='{\fontsize{16}\int_{\fontsize{8}0}^{ x}}';
- ss=['\fontsize{14}\its=',sint,'\fontsize{14}\itxsinxdx'];
- text(2.5,3.5,ss)
- set(get(a(1),'Ylabel'),'String','被积函数 \ity=xsinx')
  set(get(a(2),'Ylabel'),'String',ss)
- xlabel('x')

3.5 Additional Plotting Feature: 3.5.1 Controlling axis Plotting Limits

Table 3.5 Forms of the `axis` Function / Command

Command	Description
`v = axis;`	This function returns a 4-element row vector containing `[xmin xmax ymin ymax]`, where `xmin`, `xmax`, `ymin`, and `ymax` are the current limits of the plot.
`axis ([xmin xmax ymin ymax]);`	This function sets the *x* and *y* limits of the plot to the specified values.
`axis equal`	This command sets the axis increments to be equal on both axes.
`axis square`	This command makes the current axis box square.
`axis normal`	This command cancels the effect of axis equal and axis square.
`axis off`	This command turns off all axis labeling, tick marks, and background.
`axis on`	This command turns on all axis labeling, tick marks, and background (default case).

Controlling axis Plotting Limits

- e.g.(eg3_5_1.m)
  - x=-2*pi:pi/20:2*pi;

    y=sin(x);

    plot(x,y);

    title('plot of sin(x) vs x');

    grid on;
  - axis([0 pi 0 1]);

3.5.2 Plotting Multiple Plots on the Same Axes

- hold on; hold off
- e.g.(eg3_5_2.m)

```
x=-pi:pi/20:pi;
y1=sin(x);
y2=cos(x);
plot(x,y1,'b-');
hold on;
plot(x,y2,'k--');
hold off;
legend('sin x','cos x');
```

3.5.3 Creating Multiple Figures

- figure(n)
- e.g.(eg3_5_3.m)

```
figure(1)
x=0:0.05:2;
y1=exp(x);
plot(x,y1);
figure(2);
y2=exp(-x);
plot(x,y2);
```

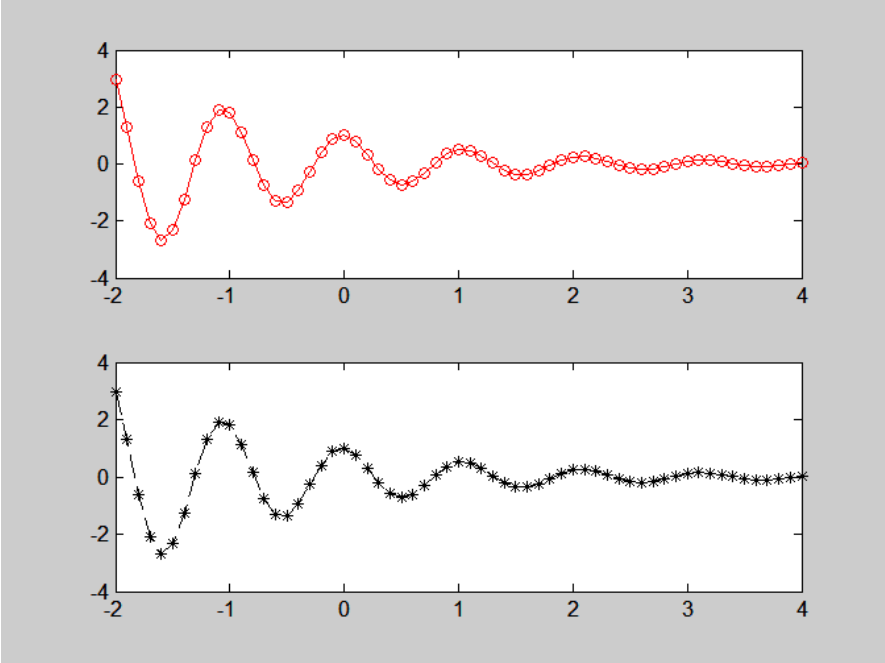
3.5.4 Subplots

- **subplot(m,n,p)**
- **e.g.(eg3_5_4.m)**

```
figure(1);
subplot(2,1,1)
x=-pi:pi/20:pi;
y=sin(x);
plot(x,y);
title('subplot 1 title');
subplot(2,1,2)
x=-pi:pi/20:pi;
y=cos(x);
plot(x,y);
title('subplot 2 title');
```

Subplots

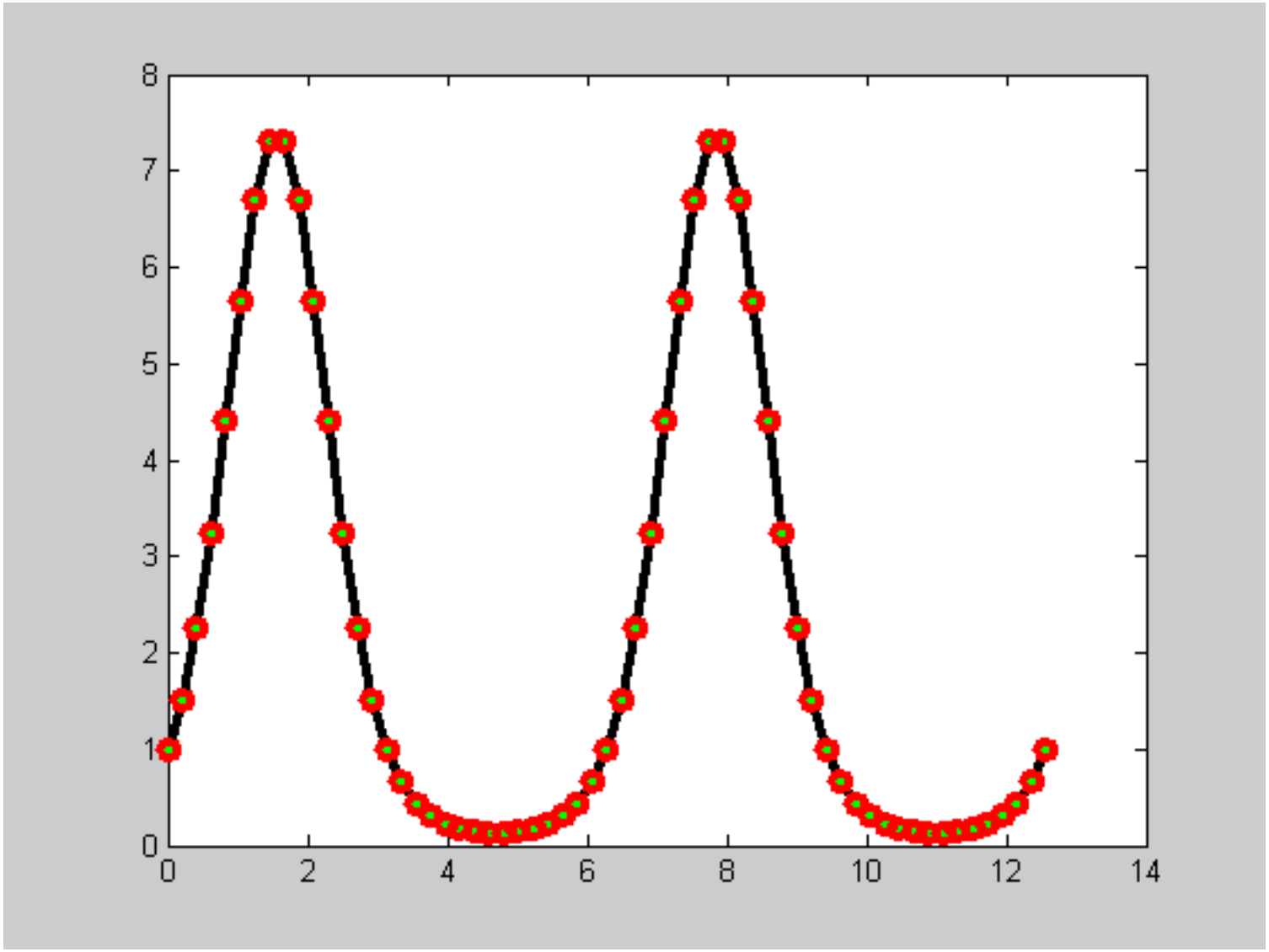
```
x = -2:0.1:4;
y = 3.5 .^ (-0.5*x) .*
    cos(6*x);
figure(1);
subplot(2,1,1);
plot(x,y,'r-o');
subplot(2,1,2);
plot(x,y,'k--*');
```



3.5.5 Enhanced Control of Plotted Lines

- Plot(x,y,'PropertyName',value,…)
  - LineWidth
  - MarkerEdgeColor
  - MarkerFaceColor
  - MarkerSize
- e.g.(eg3_5_5.m)

```
x=0:pi/15:4*pi;
y=exp(2*sin(x));
plot(x,y,'-ko', 'LineWidth', 3.0, 'MarkerSize', 6,
   'MarkerEdgeColor','r','MarkerFaceColor','g')
```


0
1
2
3
4
5
6
7
8
0
2
4
6
8
10
12
14

3.5.6 Enhanced Control of Text Strings

- **Text Strings**

  titles, axis labels,etc

- **Stream modifiers**
  - **\bf - Boldface**
  - **\it - Italics**
  - **\rm - Restore normal font**
  - **\fontname{fontname} – Specify the font name to use**
  - **\fontsize{fontsize} – Specify font size**
  - **_{xxx} – Subscripts**
  - **^{xxx} - Superscripts**

- **Table 3.6: Greek and Mathematical symbols**
- **e.g. String**

  \tau_{ind} versus \omega_{itm}

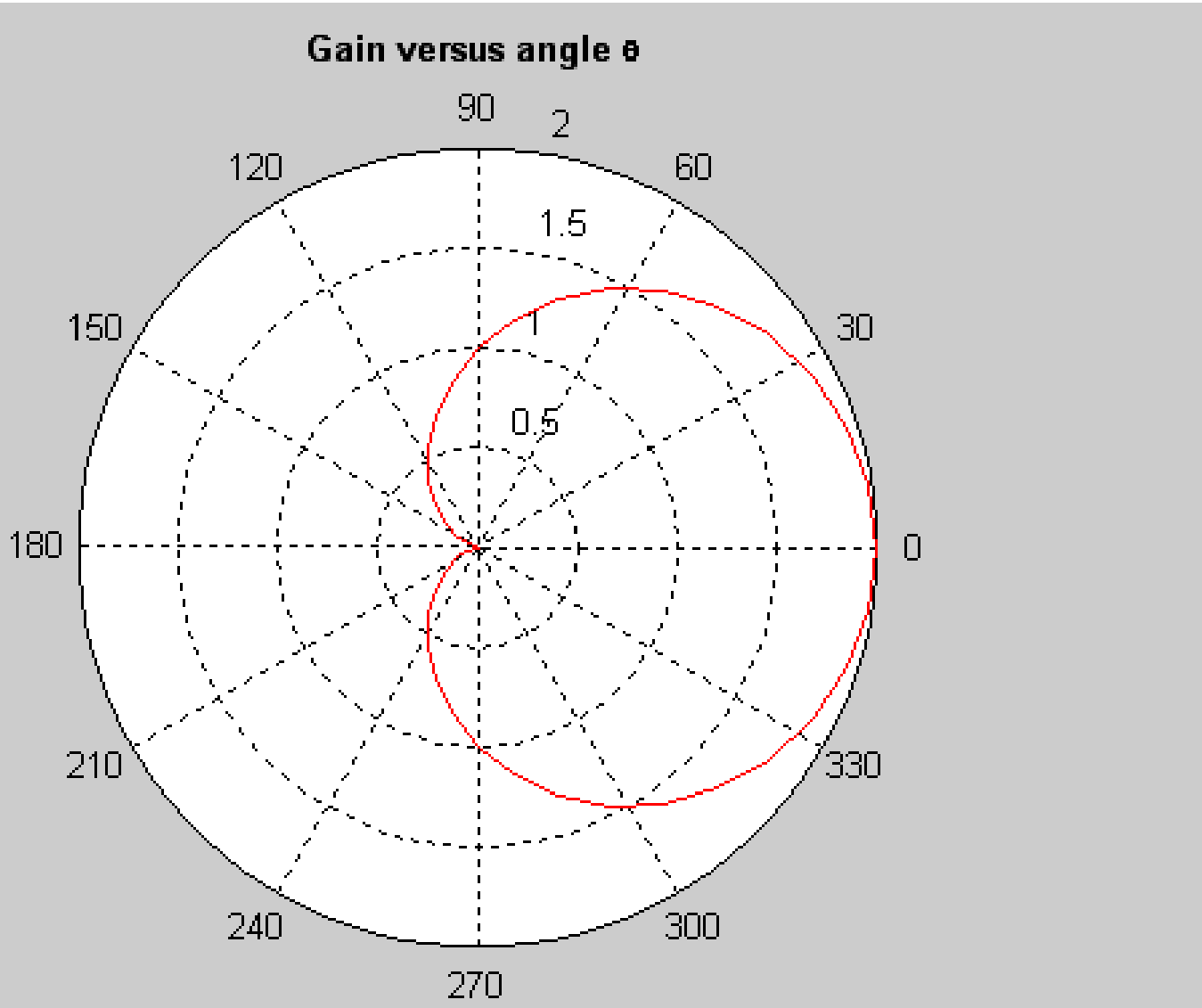
  \theta varies from 0\circle to 90\circ

  \bf{B}_{\itS}

3.5.7 Polar Plots

- polar(theta,r)
- e.g. script file: microphone.m

```
% Calculate gain versus angle
g = 0.5;
theta = 0:pi/20:2*pi;
gain = 2*g*(1+cos(theta));
% Plot gain
polar (theta,gain,'r-');
title ('\bfGain versus angle \theta');
```

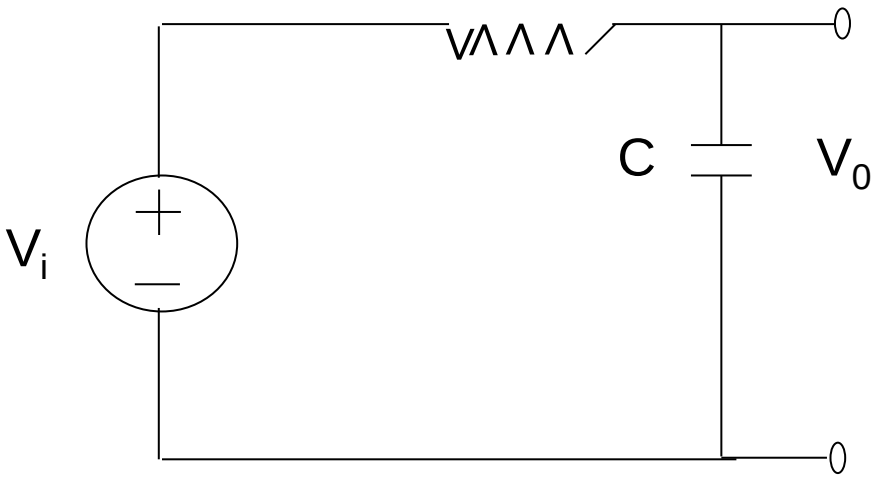

Gain versus angle θ
90
2
120
60
1.5
150
30
1
0.5
180
0
210
330
240
300
270

Plotting: Example 3.6 plot_fileter.m

Plot the amplitude and frequency response

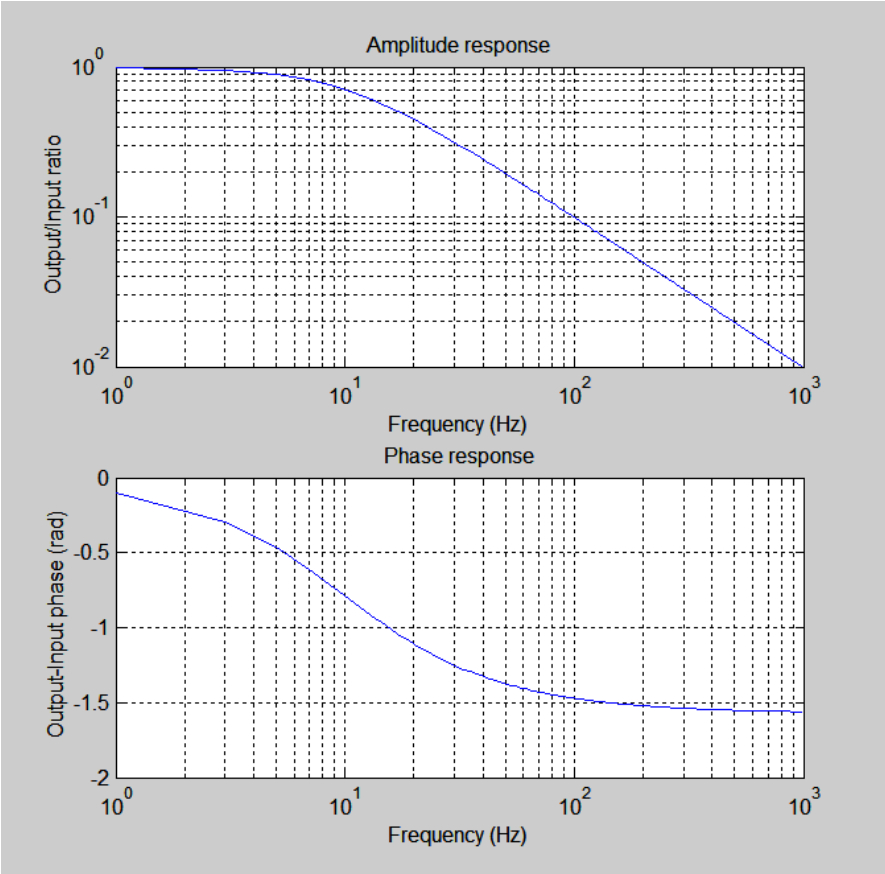
r = 16k

c = 1.0uF



A simple low-pass filter circuit

```
r = 16000; c = 1.0e-6;
f = 1:2:1000;
res = 1 ./ ( 1 + j*2*pi*f*r*c );
amp = abs(res);
phase = angle(res);
subplot(2,1,1);
loglog(f,amp);
title( 'Amplitude response' );
xlabel( 'Frequency (Hz)' );
ylabel( 'Output/Input ratio' );
grid on;
subplot(2,1,2);
semilogx(f,phase);
title( 'Phase response' );
xlabel( 'Frequency (Hz)' );
ylabel( 'Output-Input phase (rad)'
```



3.5.8 Annotating and Saving Plots

- Annotating
  - Editing tool
  - Text tool
  - Arrow tool
  - Line tool
- Saving
- Quiz 3.3

Plotting Summary

- plot(x,y)
  linear plot of vector y vs. vector x
- **title('text'), xlabel('text'), ylabel('text')**
  labels the figure, x-axis and y-axis
- grid on/off
  adds/removes grid lines
- legend('string1', 'string2', 'string3', ...)
  adds a legend using the specified strings
- hold on/off
  allows/disallows adding subsequent graphs to the current graph

Plotting Summary

- axis([xmin xmax ymin ymax])
  sets axes' limits
- v = axis
  returns a row vector containing the scaling for the current plot
- axis equal
  sets the same scale for both axes
- axis square
  makes the current axis square in size

Plotting Summary

- semilogy(x,y), semilogx(x,y), loglog(x,y)
  logarithmic plots of vector y vs. vector x
- figure(k)
  makes k the current figure
- subplot(m,n,p)
  breaks the figure window into an m-by-n matrix of small axes and selects the p^{th} axes for the current plot
- clf
  clears current figure

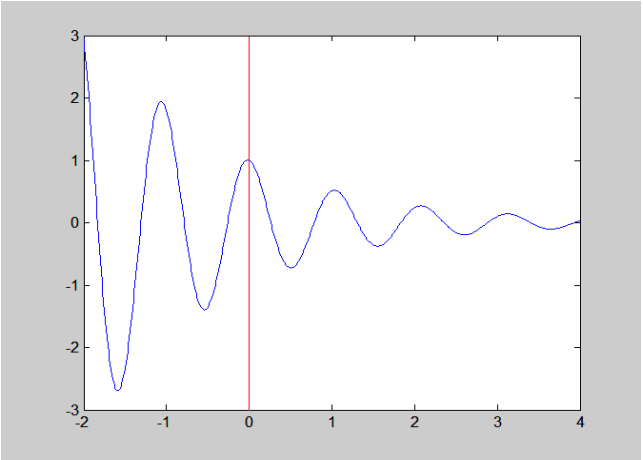
Two-Dimensional Plots

- Additional Types of 2D Plots: more than 20 types
- Example:
  - line
  - Stem
  - Stair
  - Bar
  - Pie
  - compass

Two-Dimensional Plots: examples

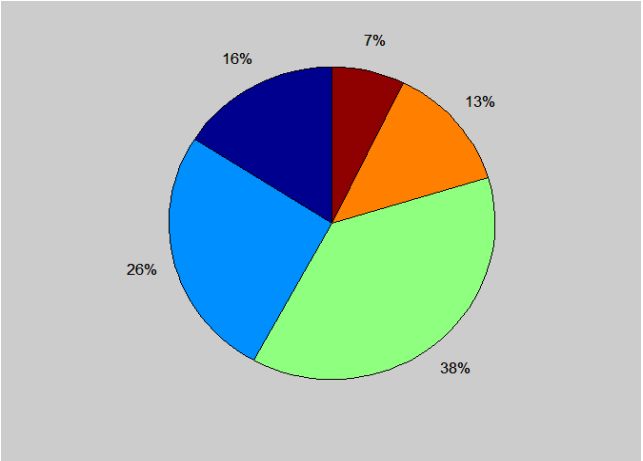
- Line plot

```
x = -2:0.01:4;
y = 3.5.^(-0.5*x).*cos(6*x);
plot(x,y);
line([0 0],[-3 3],'color','r');
```



- Pie plot

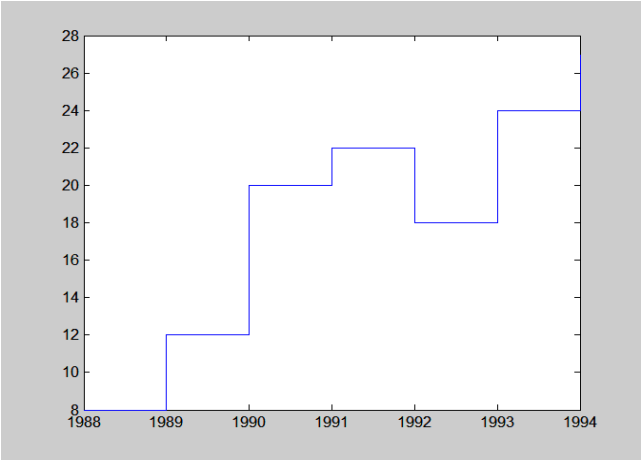
```
grades = [ 11 18 26 9 5 ];
pie(grades);
```



Two-Dimensional Plots: examples

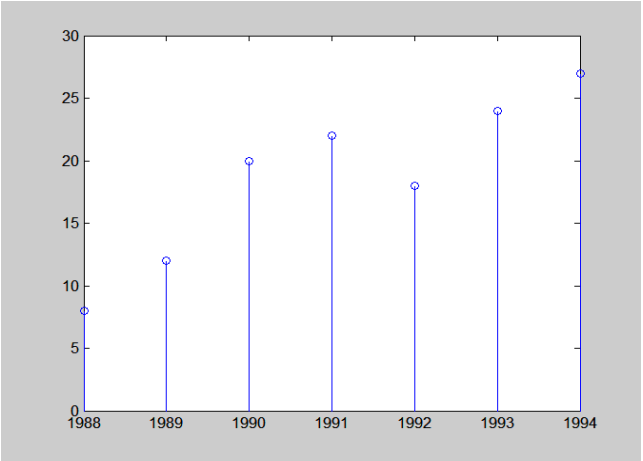
- Stairs plot

  y = 1988:1994;
  s = [8 12 20 22 18 24 27];
  stairs(y,s);



- Stem plot

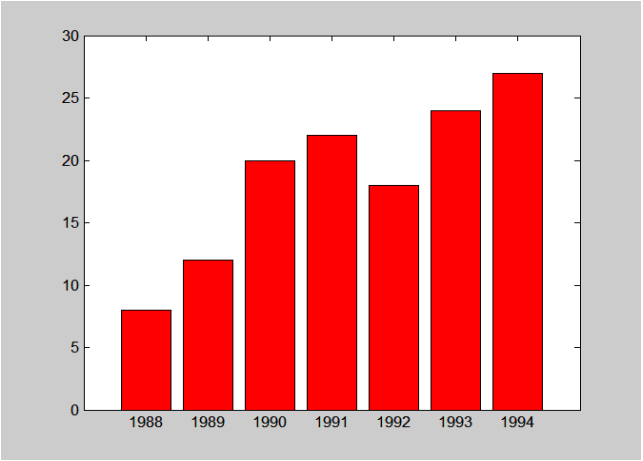
  y = 1988:1994;
  s = [8 12 20 22 18 24 27];
  stem(y,s);



Two-Dimensional Plots: examples

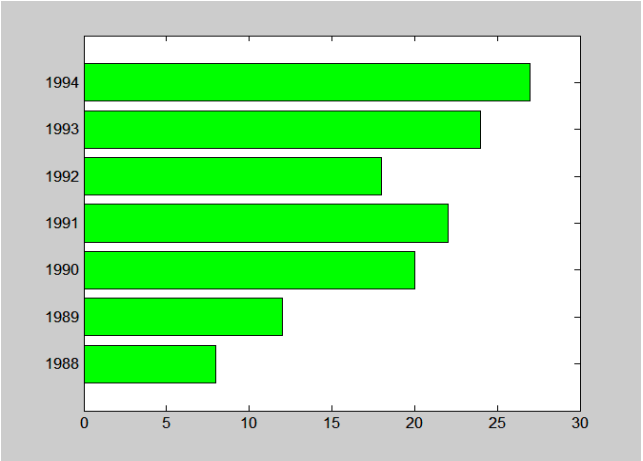
- Vertical bar plot

  y = 1988:1994;
  s = [8 12 20 22 18 24 27];
  bar(y,s,'r');



- Horizontal bar plot

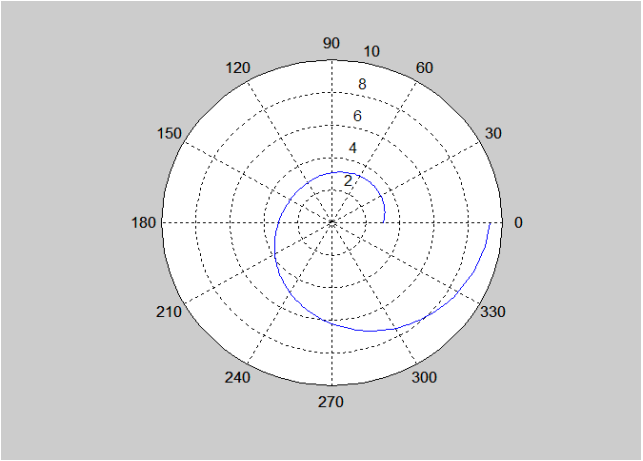
  y = 1988:1994;
  s = [8 12 20 22 18 24 27];
  barh(y,s,'g');



Two-Dimensional Plots: examples

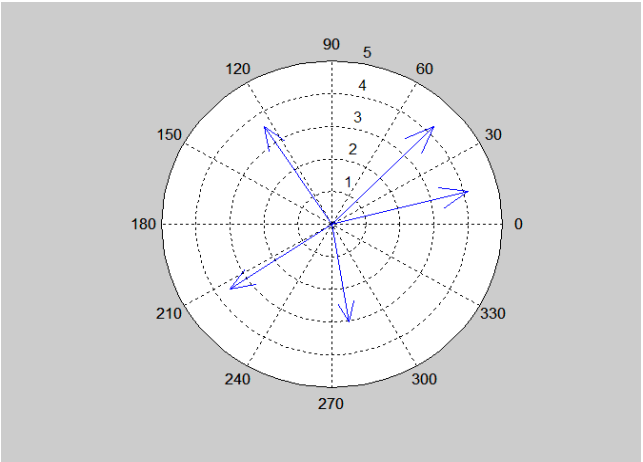
- Polar plot

  ```
  t = linspace(0,2*pi,200);
  r = 3 * cos(0.5*t).^2 + t;
  polar(t,r);
  ```



- Compass plot

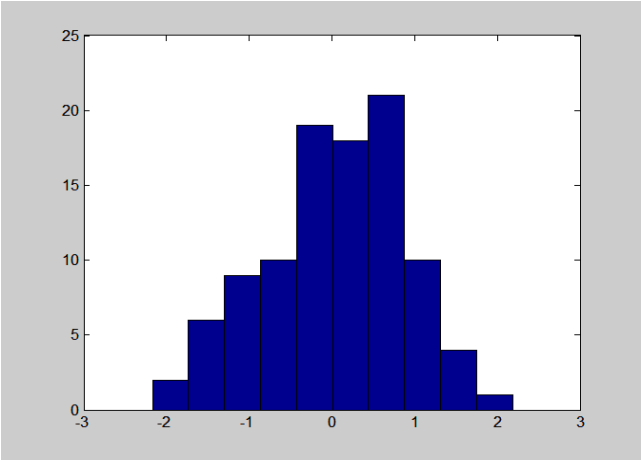
  ```
  u = [ 3 4 -2 -3 0.5 ];
  v = [ 3 1 3 -2 -3 ];
  compass(u,v);
  ```



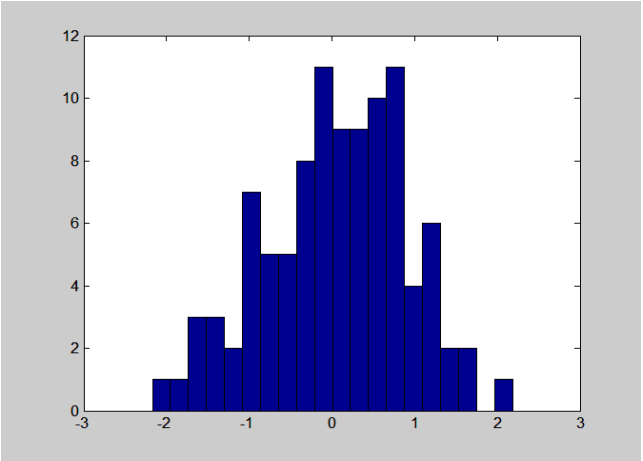
Two-Dimensional Plots: examples

- Histogram

  x = randn(1,100);
  hist(x,10);



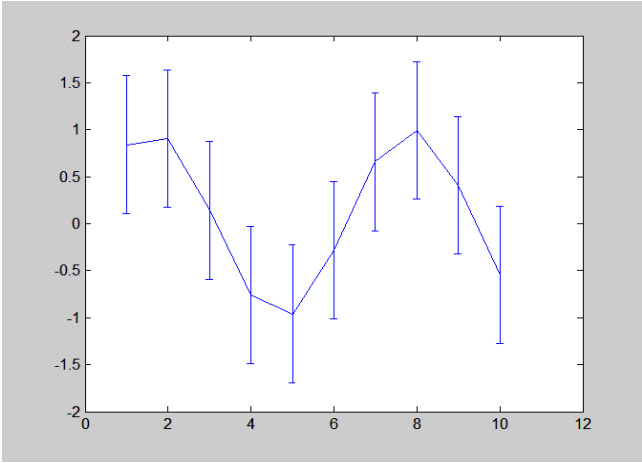
  hist(x,20);



Two-Dimensional Plots: examples

- Error bar plot

```
x = 1:10;
y = sin(x);
e = std(y) * ones(size(x));
errorbar(x,y,e);
```



Two-Dimensional Plots: Plotting Functions

- Easy to plot: plot a function directly, without the necessity of creating intermediate data arrays.
  - ezplot(fun);
  - ezplot(fun, [xmin xmax]);
  - ezplot(fun, [xmin xmax], figure);
- Example:

```
ezplot('sin(x)/x',[-4*pi 4*pi]);
title('Plot of sinx/x');
grid on;
```

- fplot

Three-Dimensional Plots: Line Plots

- plot3
- Example:
  - Test_plot.m

```
t = 0:0.1:10;
x = exp(-0.2*t) .* cos(2*t);
y = exp(-0.2*t) .* sin(2*t);
plot(x,y,'LineWidth',2);
title('\bfTwo-Dimensional
   Line Plot');
xlabel('\bfx');
ylabel('\bfy');
axis square;
grid on;
```

  - Test_plot3.m

```
t = 0:0.1:10;
x = exp(-0.2*t) .* cos(2*t);
y = exp(-0.2*t) .* sin(2*t);
plot3(x,y,t,'LineWidth',2);
title('\bfThree-Dimensional
   Line Plot');
xlabel('\bfx');
ylabel('\bfy');
zlabel('\bftime');
grid on;
```

Three-Dimensional Plots: Surface, Mesh, Contour

- Must create three equal-sized arrays
- Data prepared
  - [x,y]=meshgrid(xstart:xinc:xend, ystart:yinc:yend);

    [x,y]=meshgrid(-4:0.2:4,-4:0.2:4);
  - Z=f(x,y)

    z = exp(-0.5*(x.^2+0.5*(x-y).^2));
- mesh(x,y,z), surf(x,y,z), contour(x,y,z)

  e.g. test_contour2.m

3.5 Three-Dimensional Plots: Line Plots

- plot3
- Example:
  - Test_plot.m

```
t = 0:0.1:10;
x = exp(-0.2*t) .* cos(2*t);
y = exp(-0.2*t) .* sin(2*t);
plot(x,y,'LineWidth',2);
title('\bfTwo-Dimensional
   Line Plot');
xlabel('\bfx');
ylabel('\bfy');
axis square;
grid on;
```

  - Test_plot3.m

```
t = 0:0.1:10;
x = exp(-0.2*t) .* cos(2*t);
y = exp(-0.2*t) .* sin(2*t);
plot3(x,y,t,'LineWidth',2);
title('\bfThree-Dimensional
   Line Plot');
xlabel('\bfx');
ylabel('\bfy');
zlabel('\bftime');
grid on;
```

Three-Dimensional Plots: Surface, Mesh, Contour

- Must create three equal-sized arrays
- Data prepared
  - [x,y]=meshgrid(xstart:xinc:xend, ystart:yinc:yend);

    [x,y]=meshgrid(-4:0.2:4,-4:0.2:4);
  - Z=f(x,y)

    z = exp(-0.5*(x.^2+0.5*(x-y).^2));
- mesh(x,y,z), surf(x,y,z), contour(x,y,z)

  e.g. test_contour2.m

2.12 Examples

- Script file: temp_conversion.m
- Script file: calc_power.m
- Script file: c14_date.m

Example 2.3 – temperature conversion

- Design a MATLAB program that reads an input temperaturein degrees Fahrenheit, conberts it to an absolute temperature in kelvins,and writes out the result.

$$T(inKevins) = (\frac{5}{9}T(in^{0}F) - 32.0) + 273.15$$

Perform the following steps:

1. Prompt the user enter an input temperature in ^{o}F;

2. Read the input temperature;

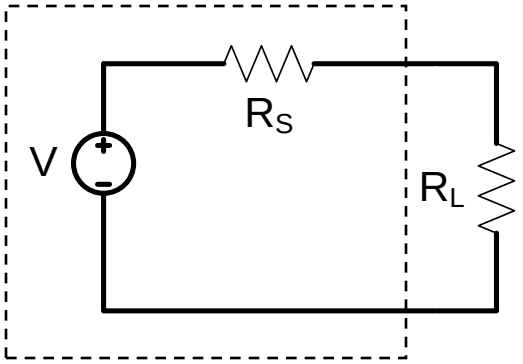
3. Calculate the temperature in kelvins from Equation

4. Write out the result,and stop

- **temp_f=input('Enter the input temperature in degree Fahrenhert:')**
- **temp_k=(5/9)*(temp_f-32)+273.15;**
- **fprintf('%6.2f degree Fahrenhert=%6.2f kelvins.\n',temp_f, temp_k);**

Examples 2.4 Maximum Power Transfer to a Load

Figure 2.9 shows a voltage source V=120V with an internal resistance Rs of 50Ω supplying a load of resistance R_L,. Find the value of load resistance R_L that will result in the maximum possible power being supplied by the source to the load.



Voltage source

Haw much power will be supplied in this case? Also, plot the power supplied to the load as a function of the load resistance R_L.

Examples 2.4 Maximum Power Transfer to a Load

In this program, we need to vary the load resistance R_L and compute the power supplied to the load at each value of R_L. The power supplied to the load resistance is given by the equation

$$P_L = I^2 R_L$$

where I is the current supplied to the load. The current supplied to the load can be calculated by Ohm's law.

$$I = \frac{V}{R_{TOT}} = \frac{V}{R_S + R_L}$$

Examples 2.4 Maximum Power Transfer to a Load

The program must perform the following steps.

1. Create an array of possible values for the load resistance R_L. The array will vary R_L from 1Ω to 100Ω in 1Ω steps.
2. Calculate the current for each value of R_L.
3. Calculate the power supplied to the load for each value of R_L.
4. Plot the power supplied to the load for each value of R_L, and determine the value of load resistance resulting in the maximum power.

Examples 2.4 Maximum Power Transfer to a Load

Now, we will demonstrate this program in MATLAB

Examples 2.4 Maximum Power Transfer to a Load

E:\重邮教案2008本科\matlab\chap2\M-File\calc_power.m

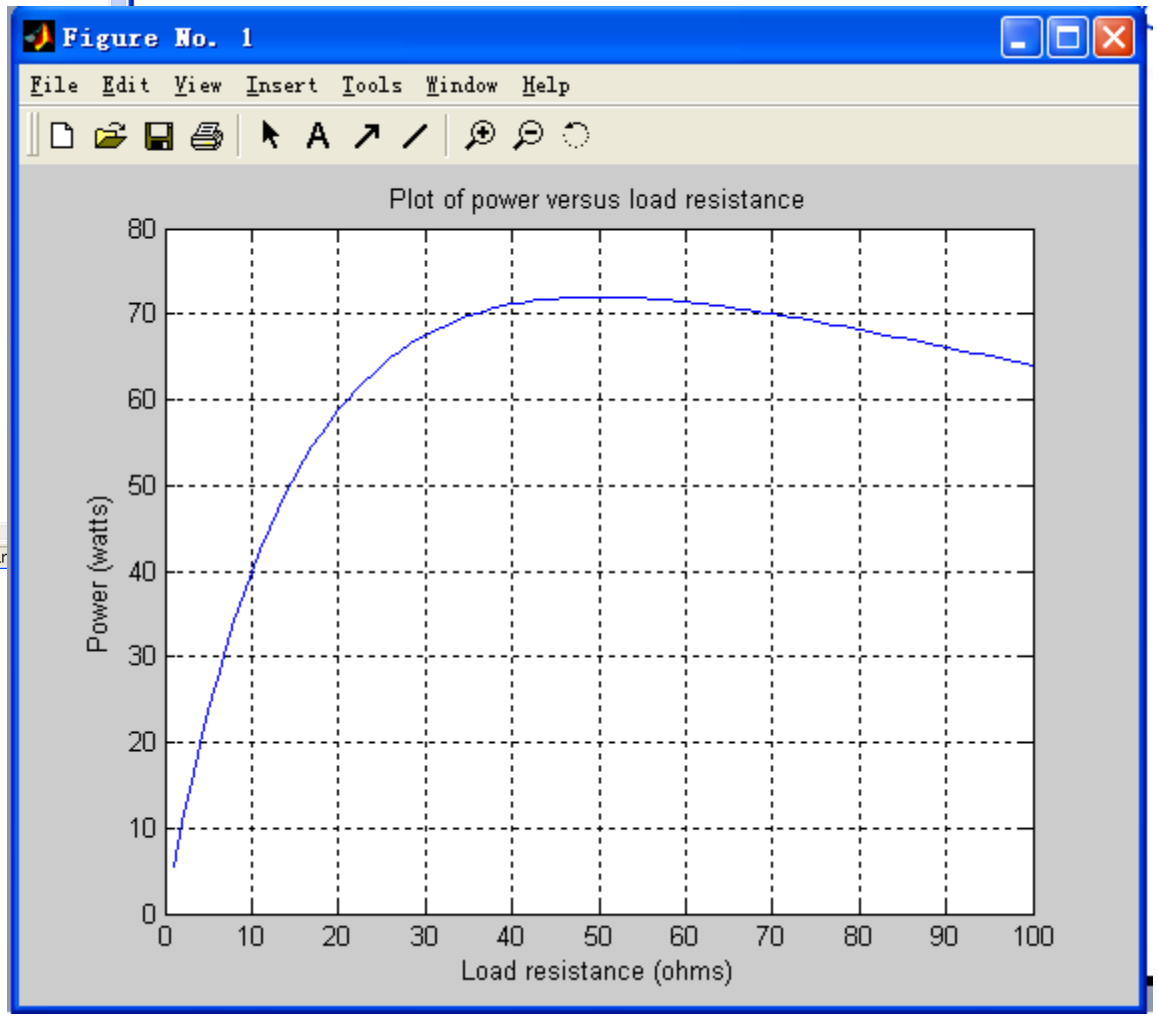
File Edit View Text Debug Breakpoints Web Window Help

```
% Script file: calc_power.m
%
% Purpose:
%   To calculate and plot the power supplied to a load as
%   as a function of the load resistance.
%
% Record of revisions:
%     Date       Programmer          Description of change
%     ====       ==========          =====================
%   01/03/04    S. J. Chapman        Original code
%   07/03/06    Maosuying
%
% Define variables:
%   amps      -- Current flow to load (amps)
%   pl        -- Power supplied to load (watts)
%   rl        -- Resistance of the load (ohms)
%   rs        -- Internal resistance of the power source (ohms)
%   volts     -- Voltage of the power source (volts)
%   maxpl     -- maximum power (watts)
%   rlmaxp    -- Resistance of the load when power is maximum (ohms)

% Set the values of source voltage and internal resistance
volts = 120;
rs = 50;

% Create an array of load resistances
rl = 1:1:100;
```

script



2.13 Debugging MATLAB Programs

- Syntax errors
  - Check spelling and punctuation
- Run-time errors
  - Check input data
  - Can remove “;” or add “disp” statements
- Logical errors
  - Use shorter statements
  - Check typos
  - Check units
  - Ask your friends, instructor, parents, …

Review: Useful Commands

help command	→ Online help
lookfor keyword	→ Lists related commands
which	→ Version and location info
clear	→ Clears the workspace
clc	→ Clears the command window
diary filename	→ Sends output to file
diary on/off	→ Turns diary on/off
who, whos	→ Lists content of the workspace
Ctrl+c	→ Aborts operation
...	→ Continuation
%	→ Comments

2.14 Summary

- Two data types: double and char
- Assignment statements
- Arithmetic calculations
- Intrinsic functions
- Input/output statements
- Data files
- Hierarchy of Operations

2.14 Summary of Good Programming Practice

- 1 .Use meaningful variable names whenever possible.
- 2. Create a data dictionary for each program .
- 3. Use only lowercase letters in variable names.
- 4. Use a semicolon at the end of all MATLAB assignment statements .
- 5. Save data in ASCII format or MAT-file format .
- 6. Save ASCII data files with a "dat" file extent and MAT-file with a “mat” extent.
- 7. Use parentheses as necessary to make your equations clear and easy to understand.
- 8. Always include the appropriate units with any values that you read or write in a program.

Homework

- Quiz 2.1 (P.30)
- Quiz 2.2 (P.38)
- Quiz 2.3 (P.44)
- Exercises
  - 1, 2, 3, 4, 5, 6, 7, 9, 11

plot: $\frac{x^2}{a^2} + \frac{y^2}{25 - a^2} = 1$

- th = [0:pi/50:2*pi]';
- a = [0.5:.5:4.5];
- X = cos(th)*a;
- Y = sin(th)*sqrt(25-a.^2);
- plot(X,Y)
- axis('equal')
- xlabel('x'), ylabel('y')
- title('A set of Ellipses')