COMP 478/6771 (FALL 2020)

Digital Image Processing

Introduction to MATLAB

Part I

Instructor: Prof. Yiming Xiao

Tutors:

Materials provided by Dr. T. D. Bui

MATLAB

- Name stands for **mat**rix **lab**oratory
- Interactive system for doing technical computations
- First version written in the 1970s!
- Evolved into a successful commercial software package
- Integrates computation, visualization, and programming.

MATLAB

- Language of technical computing
- Matrix-based system for performing mathematical and engineering calculations.
  - MATLAB has only one data type: **matrix**, or a rectangular array of numbers.
  - All variables handled in MATLAB are matrices.
- Has an extensive set of routines for obtaining graphical outputs.

MATLAB

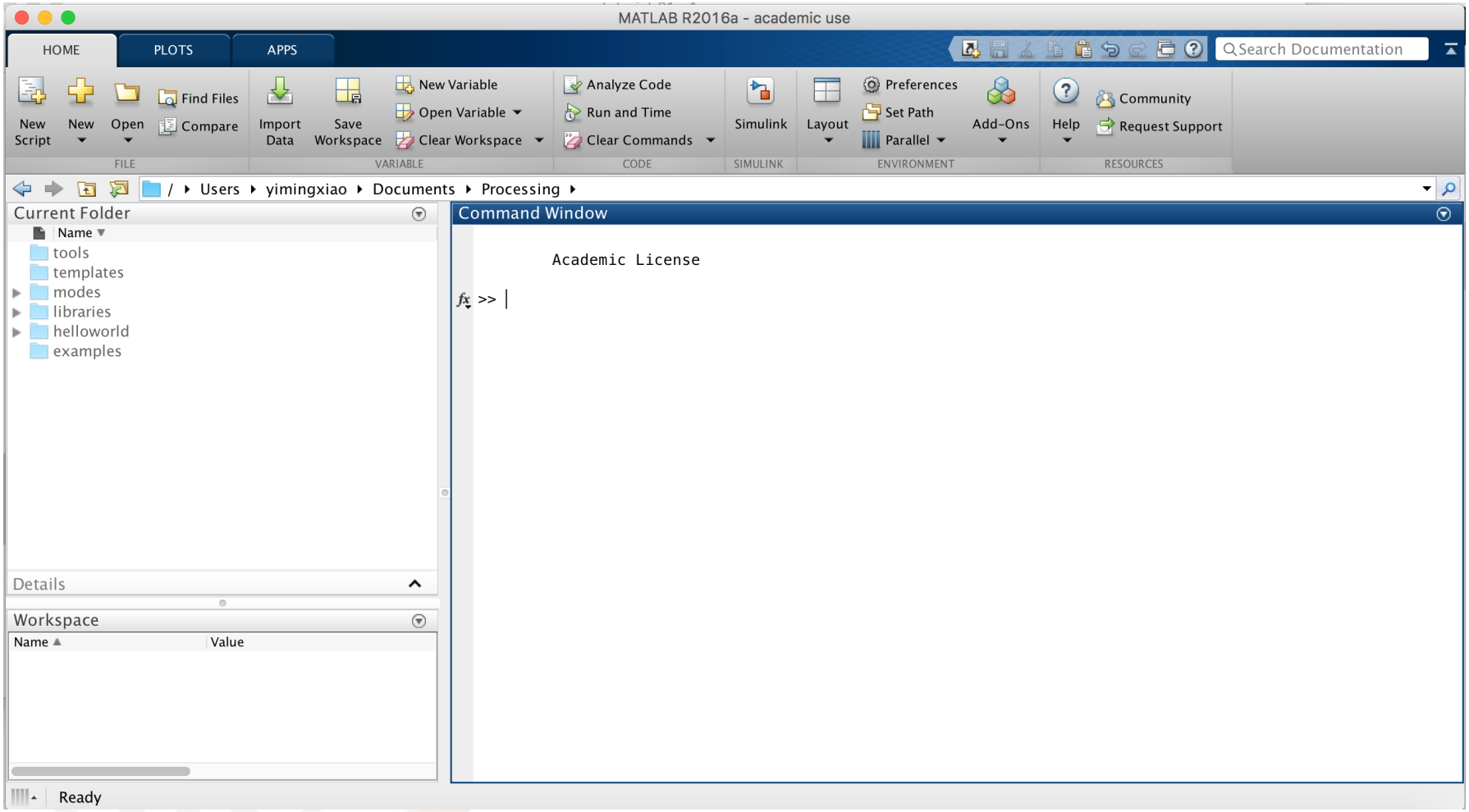
- In university environments
  - is the standard instructional tool for introductory and advanced courses in mathematics, engineering, and science
- In industry
  - is the tool of choice for high-productivity research, development, and analysis.

MATLAB

- Typical uses:
  - Math and computation
  - Algorithm development
  - Data acquisition
  - Modeling, simulation, and prototyping
  - Data analysis, exploration, and visualization
  - Scientific and engineering graphics
  - Application development, including GUI building

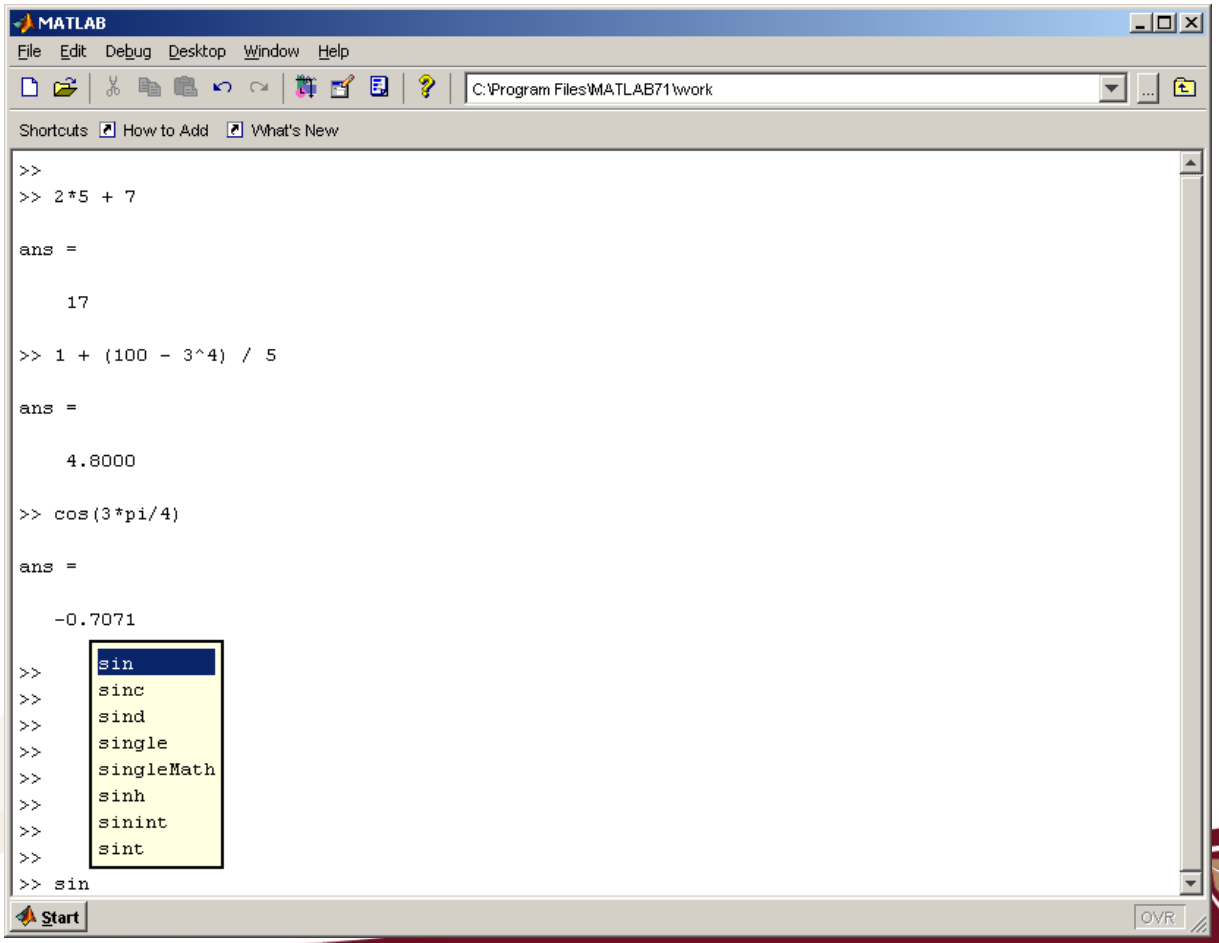
MATLAB

- Powerful toolboxes
  - Extending the environment to solve particular classes of problems:
    - Symbolic Math
    - Control Systems
    - Neural Networks
    - Signal / Image Processing
    - Partial Differential Equations
    - ...


MATLAB R2016a - academic use
HOME
PLOTS
APPS
Search Documentation
New Script
New
Open
Find Files
Compare
Import Data
Save Workspace
New Variable
Open Variable
Clear Workspace
Analyze Code
Run and Time
Clear Commands
Simulink
Layout
Preferences
Set Path
Parallel
Add-Ons
Help
Community
Request Support
FILE
VARIABLE
CODE
SIMULINK
ENVIRONMENT
RESOURCES
/ ▸ Users ▸ yimingxiao ▸ Documents ▸ Processing ▸
Current Folder
Name
tools
templates
modes
libraries
helloworld
examples
Details
Workspace
Name
Value
Command Window
Academic License
>>
Ready

Diving into MATLAB

- MATLAB as a calculator:



MATLAB Commands

- MATLAB is usually used in a command-driven mode.
- Some commonly-used commands:
  - `help`
    Displays help text in Command Window.
  - `clc`
    Clears the command window and homes the cursor
  - `who, whos`
    List the variables in the current workspace
  - `clear`
    Removes all the variables from the workspace
  - `save,load`
    Saves/loads workspace variables to/from disk
  - `class`
    Returns the class of an object.
  - `clock, date, computer, …`

How to Use MATLAB?

- Command driven:
  - MATLAB processes single-line commands immediately and displays the result.
- Script driven:
  - MATLAB is also capable of executing sequences of command that are stored in files.
    - These source files are called m-files, having a .m extension.

Matrix Operators

+	Unary Plus/Addition
-	Unary Minus/Subtraction
*	Multiplication
^	Power
‘	Conjugate transpose
/ or \	Matrix division
./ or .\	Array division

Matrix Operators

Operation	M-File	Description
`a + b`	`plus(a,b)`	Binary addition
`a - b`	`minus(a,b)`	Binary subtraction
`-a`	`uminus(a)`	Unary minus
`+a`	`uplus(a)`	Unary plus
`a.*b`	`times(a,b)`	Element-wise multiplication
`a*b`	`mtimes(a,b)`	Matrix multiplication
`a./b`	`rdivide(a,b)`	Right elementwise division
`a.\b`	`ldivide(a,b)`	Left elementwise division
`a/b`	`mrdivide(a,b)`	Matrix right division
`a\b`	`mldivide(a,b)`	Matrix left division
`a.^b`	`power(a,b)`	Element-wise power
`a^b`	`mpower(a,b)`	Matrix power
`a'`	`ctranspose(a)`	Complex conjugate transpose
`a.'`	`transpose(a)`	Matrix transpose

Relational and Logical Operators

<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal
~=	Not equal
&	AND
|	OR
~	NOT

- Note that = is used in an assignment statement, while == is used in a relation.

Misc Operators

[]	Used to form vectors and matrices
()	Arithmetic expression precedence
,	Separate subscripts and function arguments
;	End rows; suppress printing
:	Subscripting, vector generation
!	Execute operating system command
%	Comment

Semicolon Operator

- Use of semicolon
  - Suppressing printing
    - If an statement is terminated with a semicolon, printing is suppressed.
      - The command is still executed, but the result is not displayed.
  - Entering matrices
    - A semicolon is used to indicate the end of a row, except for the last row.

Colon Operator

- Use of colon
  - Creating vectors
  - Subscripting matrices
  - Specifying iterations
    - Example:

```
t = 1:5
```

Generates a row vector containing the numbers from 1 to 5 with unit increment that is:

```
t =
     1   2   3   4   5
```

Colon Operator

- Use of colon
  - An increment other than unity can be used. For example:

```
t = 1:0.5:3
```

  will result in:

```
t =
    1.0000  1.5000  2.0000  2.5000  3.0000
```

  - Negative increments may also be used.
- There are also some commands for generating sequential data such as `linspace` or `logspace`.

Functions (Math. Commands)

- Some built-in and commonly used mathematical functions:

```
sin, cos, tan, asin, atan, …
log, log10, log2, exp, …
sqrt, abs, sign, …
real, imag, conj, angle, …
```

Vectors in MATLAB

- Vectors are essentially 1×*n* or *n*×1 matrices.
- Vectors are used to hold ordinary 1-*D* sampled data signals or sequences.
- There is two types of vectors in MATLAB:
  - Row vector
  - Column vector

Vectors in MATLAB

- Entering vectors into MATLAB:
  - No dimension or type statements are needed.
  - One way is to enter the vector as an explicit list of elements separated by blank, spaces or commas.
  - Example:

```
x = [1 2 3 -4 -5]
```

or

```
x = [1, 2, 3, -4, -5]
```

- A row vector can be turned into a column vector by transposition.
- Another way of entering a column vector is to use semicolons or newlines as the element separator.

Entering Matrices Into MATLAB

- A matrix

$$A = \begin{bmatrix} 1.2 & 10 & 15 \\ 3 & 5.5 & 2 \\ 4 & 6.8 & 7 \end{bmatrix}$$

  may be entered into MATLAB as follows:

```
A = [1.2 10 15; 3 5.5 2; 4 6.8 7]
```

- Again, the values must be entered within brackets.
- As with vectors, the elements of any row must be separated by blanks (or commas).
- The end of each row, except for the last one, is indicated by a semicolon.

Entering Matrices Into MATLAB

- As another example, a matrix

$$C = \begin{bmatrix} 1 & e^{-0.02} \\ \sqrt{2} & 3 \end{bmatrix}$$

may be entered as:

```
C = [1 exp(-0.02); sqrt(2) 3]
```

After the carriage return, the following matrix will be seen on the screen:

```
C =
    1.0000    0.9802
    1.4142    3.0000
```

Generating Vectors/Matrices

- Utility functions:
  - `linspace, logspace`
  - `ones, zeros, eye, diag, rand, randn, magic`
- Examples:

```
x = linspace(-10,10,5);
w = logspace(-1,1,10);
I = eye(5);
A = ones(3,4);
B = diag([ones(1,5)]);
```

Matrix Constructors

Function	Description
`ones`	Create a matrix or array of all ones.
`zeros`	Create a matrix or array of all zeros.
`eye`	Create a matrix with ones on the diagonal and zeros elsewhere.
`diag`	Create a diagonal matrix from a vector.
`magic`	Create a square matrix with rows, columns, and diagonals that add up to the same number.
`rand`	Create a matrix or array of uniformly distributed random numbers.
`randn`	Create a matrix or array of normally distributed random numbers and arrays.

Vector/Matrix Functions

- Basic commands:

`length`	Returns length of a vector
`size`	Returns number of rows/columns of a matrix
`ndims`	Returns number of dimensions a matrix
`numel`	Returns number of elements of a matrix

- Utility/Math functions:

`reshape`	Reshapes a matrix
`sum`	Returns sum of elements
`min/max`	Returns minimum/maximum of elements

`inv, det, ...`

Variables in MATLAB

- A convenient feature of MATLAB is that variables need not be dimensioned before they are used.
  - A variable's dimensions are generated automatically upon the first use of the variable, and
- The dimensions of the variables can be altered later if necessary.
- Example:

```
>> A = [1 2 3; 4 5 6; 7 8 9];
>> x = [1 2 3];
>> whos
   …
>> x = [1 2 3 4 5];
>> A = x' * x
```

Variables in MATLAB

- Examples of variable names:
  - Legal:

```
averageCost          average_cost
n5                   N5
Left2Pay
```

  - Illegal:
    - Syntactically:

```
average-Cost         average cost
2pay                 %x
@sign
```

    - Semantically:
      - Function Names:

```
sin, cos, abs, …
```

      - Special Names:

```
ans, eps, pi, i,j, …
```

Primitive Numeric Types

C Type	Equivalent MATLAB Type
char, byte	int8
unsigned char, byte	uint8
Short	int16
int, long	int32
unsigned int, unsigned long	uint32
float	single
double	double

Complex Number Type

- MATLAB has a built-in support for complex numbers.
- Complex numbers in MATLAB are simply represented as *A* ± *B*i or *A* ± *B*j, where *A* is the real part and *B* is the imaginary part.
  - i and j are used to represent complex numbers. That's why you shouldn't use them as ordinary variable names.
- Useful functions:
  - `complex, real, imag, conj, angle`
- Example:

```
>> m = sqrt(-3)
>> n = 4 + 7i % == complex(4,7)
>> m*n
```

More About Matrices

- MATLAB is a matrix-based computing environment.
- Matrix is the most basic data structure in MATLAB.
- All data is stored in the form of a matrix or a multidimensional array.
  - Even a single numeric value is stored as a 1-by-1 matrix:

```
>> a = 5;
>> size(a)
```

Creating Matrices

- Create a row in the matrix by entering elements within brackets. Separate each element with a comma or space:

```
row = [E1, E2, ..., Em]
```

- To start a new row, terminate the current row with a semicolon:

```
A = [row1; row2; ...; rown]
```

Concatenating Matrices

- Joining one or more matrices to make a new matrix
- The expression C = [A B] horizontally concatenates matrices A and B.
- The expression C = [A; B] vertically concatenates them.
- Example:

```
A = ones(2, 5) * 6; % 2-by-5 matrix of 6's
B = rand(3, 5); % 3-by-5 random matrix
C = [A; B] % vertically concatenate A and B
```

- We can use the function `cat` for concatenating along arbitrary dimension.

Numeric Sequences

- Useful in constructing and indexing into matrices and arrays.
- MATLAB provides a special operator to assist in creating them
  - The colon operator: (*first*:*last*) generates a 1-by-n matrix (or a row vector) of sequential numbers from the first value to the last.
  - Examples:

```
>> A = 10:15
>> A = 1:6.3
>> A = 10:5:15
```

```
>> A = -2.5:2.5
>> A = 9:1
>> A = 9:-1:1
```

Matrix Indexing

- Accessing single elements
  - specify the row first and the column second:

```
A(row, column)
```

- Accessing multiple elements
  - Subscript expressions involving colons refer to portions of a matrix.

```
A(1:m, n)
```

  - Example:

```
>> A = magic(4);
>> A(1,4) + A(2,4) + A(3,4) + A(4,4)
>> sum(A(1:4,4))
>> sum(A(:,4))
```

Vector/Matrix Indexing

- The `end` keyword
  - Designate the last element in a particular dimension of an array.
  - Example:

```
>> V = [3 7 2];
>> V(end + 1) = 8;
```

- The colon operator for specifying all elements
  - colon by itself refers to **all** the elements in a row or column of a matrix.
  - Example

```
>> sum(A(:, 2))
>> A(:)
```

Computing Matrix Functions

- Norms
  - The norm of a matrix is a scalar that gives some measure of the size of the matrix.
  - Several different definitions are commonly used, One is:

    norm(A) = largest singular value of A
  - Similarly, several definitions are available for the norm of a vector. One commonly used definition is:

    norm(x) = sum(abs(x).^2)^0.5
  - MATLAB function: `norm`

Computing Matrix Functions

- Characteristic Equation
  - The roots of the characteristic equation of a square matrix **A** are the same as the eigenvalues of **A**.
  - The characteristic equation of **A** is computed with the function poly(**A**).
  - Example

```
>> A = [0 1 0; 0 0 1; -6 -11 -6];
>> p = poly(A)
p =
    1.0000   6.0000   11.0000   6.0000
```

This is the MATLAB representation of the characteristic equation

$$s^3 + 6s^2 + 11s + 6 = 0.$$

Computing Matrix Functions

- Note that polynomials are represented as row vectors containing the polynomial coefficients in descending order.
  - That is, in the previous example: p = [1 6 11 6].
- The roots of the characteristic equation p = 0 can be obtained with the function `roots`:

```
>> r = roots(p)
r =
    -3.0000
    -2.0000
    -1.0000
```

Computing Matrix Functions

- Note that the commands ploy and roots could be combined into a single expression:
  ```
  roots(poly(A))
  ```
- The roots of characteristic equation may be reassembled back into the original polynomial with the function poly.

```
>> r = [-3 -2 -1];
>> q = poly(r)
q =
     1  6  11  6
```

Computing Matrix Functions

- Addition or subtraction of polynomials
  - If the two polynomials are of the same order, simply add the vectors that describe their coefficients.
  - If the polynomials are of different order (*n* and *m*, where *m* < *n*), then add *n*-*m* zeros to the left side of the coefficient vector of the lower order polynomial.
  - Example:

```
>> a = [3 10 25 36 50];
>> b = [0 0 1 2 10];
>> a+b
ans =
       3   10   26   38   60
```

Computing Matrix Functions

- Eigenvalues and eigenvectors
  - If **A** is an $n \times n$ matrix, then the n numbers λ that satisfy $\mathbf{Ax} = \lambda\mathbf{x}$ are the eigenvalues of **A**.
  - Eigenvalues of **A** are obtained with the function `eig(A)`, which returns the eigenvalues in a column vector.
  - Example:

```
>> A = [0 1 0; -1 0 2; 3 0 5];
>> eig(A)
ans =
      5.2130
      -0.1065 + 1.4487i
      -0.1065 - 1.4487i
```

Computing Matrix Functions

- Eigenvalues and eigenvectors
  - MATLAB functions may have single or multiple-output arguments.
  - The function `eig(A)` for example, produces a column vector consisting of the eigenvalues of **A**, while the double-assignment statement `[X,D] = eig(A)`, produces eigenvalues or eigen vectors.
    - The diagonal elements of the diagonal matrix **D** are the eigenvalues, and the columns of **X** are the corresponding eigenvectors such that **AX** = **XD**.

Computing Matrix Functions

- Convolution (product of polynomials)
  - The product of polynomials is the convolution of the coefficients.
  - In MATLAB, the product of the polynomials a(s) and b(s) can be obtained with the function conv(a,b).
  - Example

```
>> a = [3 10 25 36 50] % 3s^4 + 10s^3 + … + 50
>> b = [1 2 10]        % s^2 + 2s + 10
>> % Define the product of a and b as c
>> c = conv(a,b)
c =
     3   16   75  186  372  460  500
>> % which is 3s^6 + 16s^5 + … + 500.
```

Computing Matrix Functions

- Polynomial evaluation
  - If ***p*** is a vector representing a polynomial, then `polyval`(***p***,*s*) is the value of the polynomial evaluated at *s*.
  - Example
    - To evaluate the polynomial *p*(*s*) = 3*s*^2 + 2*s* + 1 at *s* = 5, enter the following commands:

```
>> p = [3 2 1];
>> polyval(p,5)
```

Plotting

- MATLAB has an extensive set of routines for obtaining graphical outputs.
- The function `plot` creates *2D x-y* plots.
  - Logarithmic or polar plots are created simply by substituting the word `loglog`, `semilogx`, `semilogy` or `polar` for `plot`.
  - All such commands are used the same way. They affect only how the axis is scaled and how the data are displayed.

Plotting

- If **x** and **y** are vectors of the same length, the command `plot(x,y)` plots the values in **y** versus the values in **x**.
- We can use the plot command with multiple arguments:

  `plot(X1, Y1, X2, Y2, …, Xn, Yn)`

  to plot multiple curves on a single graph. The variables Xi and Yi are pairs of vectors.
- Plotting more than one curve on a single graph may also be accomplished by the command `hold on`, or simply `hold`.

Plotting

- To plot a a function, we have to compute (sample) the function at a sufficiently large number of points and then join up the points by straight lines.
- Example:
  - Plot y = sin(3π*x*) for $0 \leq x \leq 1$

```
>> N = 10; h = 1/N; x
>> y = sin(3*pi*x);
>> plot(x,y);
```

Plotting

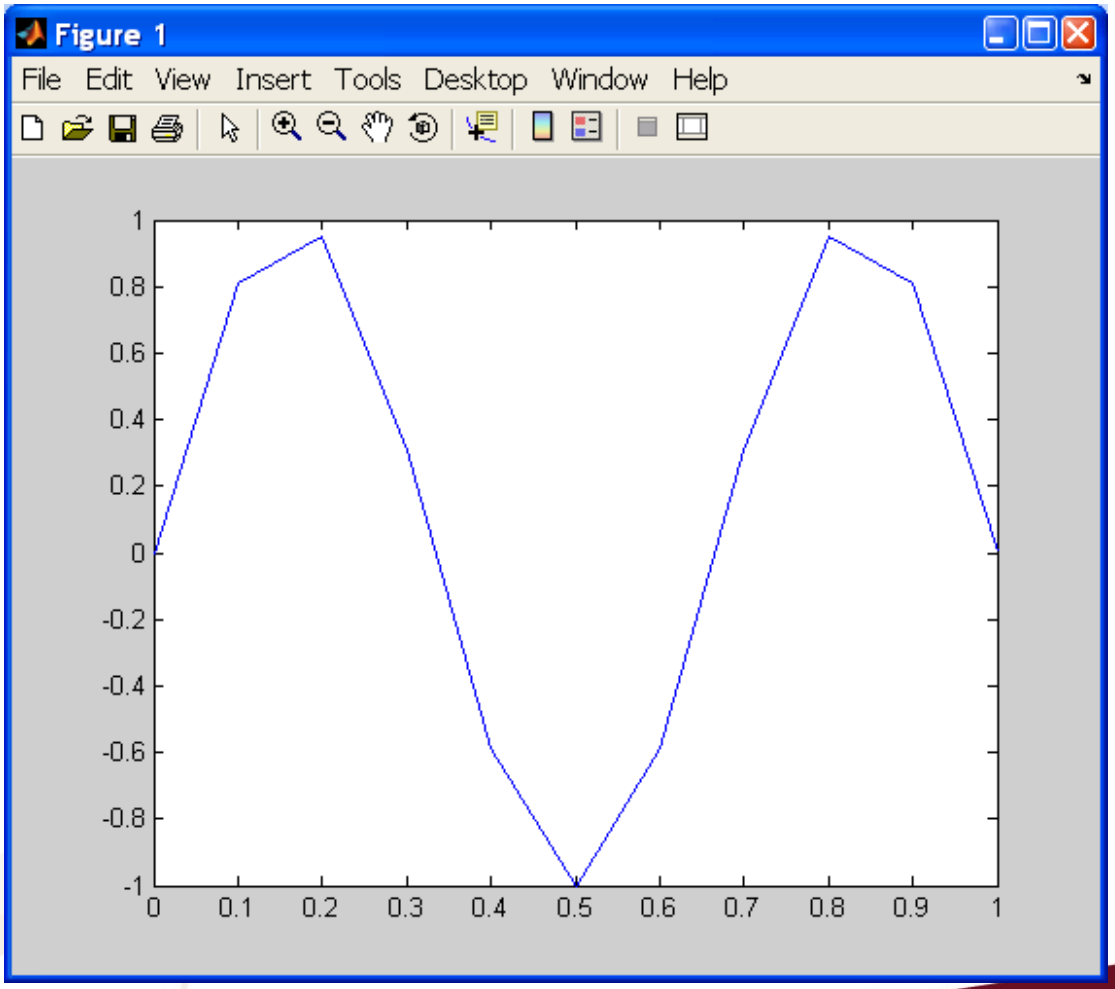


Figure 1
File Edit View Insert Tools Desktop Window Help
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1

Plotting

- Useful Functions:
  - `title`
    - Adds text at the top of the current axis.
  - `xlabel, ylabel`
    - Adds text beside the X/Y-axis on the current axis.
  - `grid`
    - Adds/removes grid lines to/from the current axes.
  - `subplot`
    - To show several plots in the same figure.

Symbolic Computations

- Symbolic Math Toolbox
  - Calculus
    - Differentiation, integration, limits, ...
  - Simplification
  - Variable-Precision Arithmetic
  - Transforms

  …
- Symbolic Object
  - Symbolic Variables, Expressions, Matrices

Symbolic Computations

- **sym**
  - Construct symbolic numbers, variables and objects.

```
>> x = sym('x');
>> delta = sym('1/10');
>> sqrt(2)
ans =
    1.4142
>> a = sqrt(sym(2))
a =
2^(1/2)
>> format long
>> ((sqrt(7)^(1/sqrt(3)))^sqrt(3))^2
ans =
7.000000000000001
>> % Now try this:
>> sym(((sqrt(7)^(1/sqrt(3)))^sqrt(3))^2)
```

Symbolic Computations

- Arithmetic on symbolic objects is different from arithmetic on standard data types.

```
>> sym(2)/sym(5)
ans =
2/5
>> 2/5 + 1/3
ans =
0.7333
sym(2)/sym(5) + sym(1)/sym(3)
ans =
11/15
```

Symbolic Computations

```
>> syms x y
>> f = x^2+x*y-2*x+y-3;
>> factor(f)
ans =
...

>> R = cos(x)^2+sin(x)^2;
>> R = simple(R)
```

MATLAB Script Files

- Normal text files that contain Matlab commands.
  - have an extension `.m`
    - commonly known as m-files.
  - created and edited with M-file Editor:

```
>> edit example1.m
>> example1 % executes the scripts
```

MATLAB Script Files

- Useful Commands:
  - `pwd`
    - displays the current working directory.
  - `cd`
    - changes the current working directory.
  - `what`
    - lists the MATLAB specific files found in the current working directory.
  - `dir`
    - lists the files in a directory.

MATLAB Expressions

- Operators:
  - Arithmetic
    - +, -, *, /, ^, …
  - Relational
    - <, <=, >, >=, ==, ~=
  - Logical
    - Element-wise
      - &, |, ~
    - Short-circuit
      - &&, ||

MATLAB Expressions

- Advantage of Short-Circuiting
  - Evaluate an expression only when certain conditions are satisfied.

  Example:

  ```
  x = (b ~= 0) && (a/b > 18.5);
  ```

  avoids divide-by-zero errors when b equals 0.

Refrences

- MATLAB
  The Language of Technical Computing
  Programming *Version 7*

  http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/matlab_prog.pdf

  www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/getstart.pdf

Excellent Matlab tips from Dr. Kevin Murphy

http://www.cs.ubc.ca/~murphyk/Software/matlab_tips.html