المملكة العربية السعودية

وزارة التعليم

MINISTRY OF EDUCATION

هام

لكل المهتمين و المهتمات

بدروس و مراجع الجامعية

مدونة المناهج السعودية eduschool40.blog

BMT 227 MATLAB Lab 1
Due: 04/10/2018 at 04:00 PM

Understanding the MATLAB Environment

MATLAB environment behaves like a super-complex calculator. You can enter commands at the >> command prompt.

The **Editor Window** is used for writing and editing programs. This window is opened by clicking on the New Script icon in the Toolstrip, or by clicking on the New icon and then selecting Script from the menu that opens. Here, in this window or we call it Script, you can type and edit your commands. Script files are also called M-files because the extension .m is used when they are saved.

Start by opening MATLAB → New Script (choose the appropriate name file

Basic Syntax

As any programing language, MATLAB also has its own syntax, which play the rules as grammar in English language. If the syntax is not correct, such statement can't be executed.

The semicolon (;): If a semicolon is typed at the end of a command, the output of the command is not displayed.

Typing % : When the symbol % (percent) is typed at the beginning of a line, the line is designated as a comment. This means that when the Enter key is pressed the line is not executed

Variables in MATLAB

Defining scalar variables: A variable is a name made of a letter or a combination of several letters (and digits) that is assigned a numerical value.

Rules About Variable Names

1. Must begin with a letter.
2. Can be up to 63 characters long.
3. Can contain letters, digits, and the underscore character.
4. Cannot contain punctuation characters (e.g., period, comma, semicolon).
5. MATLAB is case-sensitive: it distinguishes between uppercase and lowercase letters. For example, AA, Aa, aA, and aa are the names of four different variables.
6. No spaces are allowed between characters (use the underscore where a space is desired).
7. Avoid using the name of a built-in function for a variable (i.e., avoid using cos, sin, exp, sqrt, etc.). Once a function name is used to for a variable name, the function cannot be used.

Useful Commands for Managing Variables:

Command	Outcome
clear	Removes all variables from the memory.
clear x y z	Removes only variables x, y, and z from the memory.
who	Displays a list of the variables currently in the memory.
whos	Displays a list of the variables currently in the memory and their sizes together with information about their bytes and class
clc	clears the command Window

Arithmetic Operations with Scalars:

Operation	Symbol	Example
Addition	+	5 + 3
Subtraction	–	5 – 3
Multiplication	*	5 * 3
Right division	/	5 / 3
Left division	\	5 \ 3 = 3 / 5
Exponentiation	^	5 ^ 3 (means $5^3 = 125$)

For scalars, the left division is the inverse of the right division. The left division, however, is mostly used for operations with arrays.

Order of Precedence

MATLAB executes the calculations according to the order of precedence displayed below. This order is the same as used in most calculators.

Precedence	Mathematical Operation
First	Parentheses. For nested parentheses, the innermost are executed first.
Second	Exponentiation.
Third	Multiplication, division (equal precedence).
Fourth	Addition and subtraction.

Elementary Math built-in Functions

Function	Description	Example

`sqrt(x)`	Square root.	`>> sqrt(81)`
		`ans =`
		`9`

`nthroot(x,n)`	Real *n*th root of a real number *x*. (If *x* is negative *n* must be an odd integer.)	`>> nthroot(80,5)`
		`ans =`
		`2.4022`

`exp(x)`	Exponential (e^x).	`>> exp(5)`
		`ans =`
		`148.4132`

Table 1-3: Elementary math functions (Continued)

Function	Description	Example

`abs(x)`	Absolute value.	`>> abs(-24)`
		`ans =`
		`24`

`log(x)`	Natural logarithm.
	Base *e* logarithm (ln).
		`>> log(1000)`
		`ans =`
		`6.9078`

`log10(x)`	Base 10 logarithm.	`>> log10(1000)`
		`ans =`
		`3.0000`

`factorial(x)`	The factorial function *x*!
	(*x* must be a positive integer.)
		`>> factorial(5)`
		`ans =`
		`120`

Table 1-4: Trigonometric math functions

Function	Description	Example

`sin(x)`
`sind(x)`
	Sine of angle *x* (*x* in radians).
	Sine of angle *x* (*x* in degrees).
		`>> sin(pi/6)`
		`ans =`
		`0.5000`

`cos(x)`
`cosd(x)`
	Cosine of angle *x* (*x* in radians).
	Cosine of angle *x* (*x* in degrees).
		`>> cosd(30)`
		`ans =`
		`0.8660`

`tan(x)`
`tand(x)`
	Tangent of angle *x* (*x* in radians).
	Tangent of angle *x* (*x* in degrees).
		`>> tan(pi/6)`
		`ans =`
		`0.5774`

`cot(x)`
`cotd(x)`
	Cotangent of angle *x* (*x* in radians).
	Cotangent of angle *x* (*x* in degrees).
		`>> cotd(30)`
		`ans =`
		`1.7321`

Task 1 (To be Submitted)

1. Solve the following problems

Define the variable x as $x = 6.7$, then evaluate:

(*a*) $0.01x^5 - 1.4x^3 + 80x + 16.7$ (*b*) $\sqrt{x^3 + e^x - 51/x}$

Define the variable t as $t = 3.2$, then evaluate:

(*a*) $56t - 9.81\frac{t^2}{2}$ (*b*) $14e^{-0.1t}\sin(2\pi t)$

Creating a one-dimensional array (vector)

First method:

- The **vector** is created by typing the elements (numbers) inside square brackets [ ].
  - Row vector: To create a row vector type the elements with a space or a comma between the elements inside the square brackets.

```
variable_name = [ type vector elements ]
```

  - Column vector: To create a column vector type the left square bracket [and then enter the elements with a semicolon between them, or press the Enter key after each element. Type the right square bracket] after the last element.

Second method:

- Creating a vector with constant spacing by specifying the first term, the spacing, and the last term.
  - In a vector with constant spacing, the difference between the elements is the same. For example, in the vector v = 2 4 6 8 10, the spacing between the elements is 2. A vector in which the first term is **m**, the spacing is **q**, and the last term is **n** is created by typing:

```
variable_name = [m:q:n]   or   variable_name = m:q:n
```

(The brackets are optional.)

Third method:

- Creating a vector with linear (equal) spacing by specifying the first and last terms, and the number of terms.
  - A vector with **n** elements that are linearly (equally) spaced in which the first element is ***xi*** and the last element is ***xf*** can be created by typing the linspace command (MATLAB determines the correct spacing):

```
variable_name = linspace(xi,xf,n)
```

First element — Last element — Number of elements

Array addressing

The address of an element in a vector is its position in the row (or column). For a vector named ve, ve (k) refers to the element in position k. The first position is 1.

Task 2 (To be Submitted):

1. Create three vectors using the three methods above: Name them: Vec1, Vec2, and Vec3. All the vectors should include numbers from 10 to 20 by increment of 2, thus each vector includes 10 numbers.
2. Evaluate a sine function (sin(t)) using Vec1 vector for time t.

Creating a Two-dimensional array (MATRIX)

A *m x n* matrix has **m** rows and **n** columns, and **m** by **n** is called the size of the matrix. A matrix is created by assigning the elements of the matrix to a variable. This is done by typing the elements, row by row, inside square brackets [ ]. First type the left bracket [then type the first row, separating the elements with spaces or commas. To type the next row, type a semicolon or press Enter. Type the right bracket] at the end of the last row.

```
variable_name=[1st row elements; 2nd row elements; 3rd
               row elements; ... ; last row elements]
```

Notes about variables in MATLAB:

1. **All variables in MATLAB are arrays. A scalar is an array with one element, a vector is an array with one row or one column of elements, and a matrix is an array with elements in rows and columns.**

Matrix addressing

The address of an element in a matrix is its position, defined by the row number and the column number where it is located. For a matrix assigned to a variable **ma**, **ma**(k,p) refers to the element in row **k** and column **p**

The Transpose Operator:

The transpose operator, when applied to a vector, switches a row (column) vector to a column (row) vector. When applied to a matrix, it switches the rows (columns) to columns (rows). The transpose operator is applied by typing a single quote ' following the variable to be transposed.

Using a colon : In addressing arrays

```
>> v=[4 15 8 12 34 2 50 23 11]
v =
     4    15     8    12    34     2    50    23    11
>> u=v(3:7)
u =
     8    12    34     2    50
>>
```

A vector v is created.

A vector u is created from the elements 3 through 7 of vector v.

Using a colon : In addressing a matrix:

$A(:,n)$	Refers to the elements in all the rows of column n of the matrix A.
$A(n,:)$	Refers to the elements in all the columns of row n of the matrix A.
$A(:,m:n)$	Refers to the elements in all the rows between columns m and n of the matrix A.
$A(m:n,:)$	Refers to the elements in all the columns between rows m and n of the matrix A.

Task 3 (To be Submitted):

1. Create a column vector with 9 equally spaced elements in which the first element is -34 and the last element is -7. (A column vector can be created by the transpose of a row vector.)

Basic Polynomials applications using vectors

In MATLAB, polynomials are represented by a row vector in which the elements are the coefficients $a_n, a_{n-1}, \ldots, a_1, a_0$. The first element is the coefficient of the x with the highest power. The vector has to include all the coefficients, including the ones that are equal to 0. For example:

Polynomial	**MATLAB representation**
$8x + 5$	p = [8 5]
$2x^2 - 4x + 10$	d = [2 –4 10]
$6x^2 - 150$, MATLAB form: $6x^2 + 0x - 150$	h = [6 0 –150]
$5x^5 + 6x^2 - 7x$, MATLAB form: $5x^5 + 0x^4 + 0x^3 + 6x^2 - 7x + 0$	c = [5 0 0 6 –7 0]

Roots of a Polynomial

MATLAB has a function, called `roots`, that determines the root, or roots, of a polynomial. The form of the function is:

```
r = roots(p)
```

`r` is a column vector with the roots of the polynomial.

`p` is a row vector with the coefficients of the polynomial.

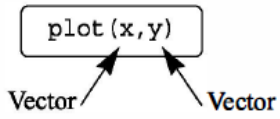
Task 4 (To be Submitted):

1. **Find the roots of the following polynomials:**
   a. $5x^2 + x^3 + 25 = 0$
   b. $3x + x^2 = 15$

Basic Plotting in MATLAB

- **The plot command:**

The `plot` command is used to create two-dimensional plots. The simplest form of the command is:



- The arguments x and y are each a vector (one-dimensional array). The two vectors must have the same number of elements.
- **Formatting a Plot Using Commands**

The `xlabel` and `ylabel` commands:

Labels can be placed next to the axes with the `xlabel` and `ylabel` command which have the form:

```
xlabel('text as string')
ylabel('text as string')
```

The `title` command:

A title can be added to the plot with the command:

```
title('text as string')
```

- **Plotting multiple graphs in the same plot**
  - There are three methods to plot multiple graphs in one figure. One is by using the plot command, the second is by using the *hold on* and *hold off* commands, and the third is by using the line command. We are going to use *hold on* and *hold off* commands for this course.

Task 5 (To be Submitted):

1. Create a time vector "t" from 0 to 0.5 sec using 2000 elements.
2. Define a variable f=100.
3. Evaluate the function y=sin(f*t).
4. Plot the function y.
5. Label and title your graph.

- **Addition, Multiplication, and Division of Polynomials**
- **Symbolic Math**
- **Derivatives of Polynomials**
- **Integration**

❖ Addition, Multiplication, and Division of Polynomials

8.1.3 Addition, Multiplication, and Division of Polynomials

Addition:

Two polynomials can be added (or subtracted) by adding (subtracting) the vectors of the coefficients. If the polynomials are not of the same order (which means that the vectors of the coefficients are not of the same length), the shorter vector has to be modified to be of the same length as the longer vector by adding zeros (called padding) in front. For example, the polynomials

$f_1(x) = 3x^6 + 15x^5 - 10x^3 - 3x^2 + 15x - 40$ and $f_2(x) = 3x^3 - 2x - 6$ can be added by:

```
>> p1=[3 15 0 -10 -3 15 -40];
>> p2=[3 0 -2 -6];
>> p=p1+[0 0 0 p2]
p =
     3    15     0    -7    -3    13   -46
```

Three 0s are added in front of `p2`, since the order of `p1` is 6 and the order of `p2` is 3.

Multiplication:

Two polynomials can be multiplied using the MATLAB built-in function `conv`, which has the form:

```
c = conv(a,b)
```

`c` is a vector of the coefficients of the polynomial that is the product of the multiplication.

`a` and `b` are the vectors of the coefficients of the polynomials that are being multiplied.

- The two polynomials do not have to be of the same order.
- Multiplication of three or more polynomials is done by using the `conv` function repeatedly.

For example, multiplication of the polynomials $f_1(x)$ and $f_2(x)$ above gives:

```
>> pm=conv(p1,p2)
pm =
     9    45    -6   -78   -99    65   -54   -12   -10   240
```

which means that the answer is:

$9x^9 + 45x^8 - 6x^7 - 78x^6 - 99x^5 + 65x^4 - 54x^3 - 12x^2 - 10x + 240$

Division:

A polynomial can be divided by another polynomial with the MATLAB built-in function `deconv`, which has the form:

```
[q,r] = deconv(u,v)
```

`q` is a vector with the coefficients of the quotient polynomial.
`r` is a vector with the coefficients of the remainder polynomial.

`u` is a vector with the coefficients of the numerator polynomial.
`v` is a vector with the coefficients of the denominator polynomial.

For example, dividing $2x^3+9x^2+7x-6$ by $x+3$ is done by:

```
>> u=[2 9 7 -6];
>> v=[1 3];
```

```
>> [a b]=deconv(u,v)
a =
     2     3    -2
b =
     0     0     0     0
```

The answer is: $2x^2+3x-2$.

Remainder is zero.

An example of division that gives a remainder is $2x^6-13x^5+75x^3+2x^2-60$ divided by x^2-5:

```
>> w=[2 -13 0 75 2 0 -60];
>> z=[1 0 -5];
>> [g h]=deconv(w,z)
g =
     2   -13    10    10    52
h =
     0     0     0     0     0    50   200
```

The quotient is: $2x^4-13x^3+10x^2+10x+52$.

The remainder is: $50x+200$.

The answer is: $2x^4-13x^3+10x^2+10x+52+\frac{50x+200}{x^2-5}$.

Task 1:

Use MATLAB to carry out the following multiplication of polynomials:

$$x(x-1.7)(x+0.5)(x-0.7)(x+1.5)$$

Plot the polynomial for $-1.6 \le x \le 1.8$.

Divide the polynomial $-10x^6-20x^5+9x^4+10x^3+8x^2+11x-3$ by the polynomial $2x^2+4x-1$.

Divide the polynomial
$-0.24x^7+1.6x^6+1.5x^5-7.41x^4-1.8x^3-4x^2-75.2x-91$ by the polynomial $-0.8x^3+5x+6.5$.

All materials are taken from: *Gilat, Amos. MATLAB: an Introduction with Applications. John Wiley & Sons, 2015.*

❖ Symbolic Math

11.1.1 Creating Symbolic Objects

Symbolic objects can be variables or numbers. They can be created with the `sym` and/or `syms` commands. A single symbolic object can be created with the `sym` command:

```
object_name = sym('string')
```

where the string, which is the symbolic object, is assigned to a name. The string can be:

- A single letter or a combination of several letters (no spaces). Examples: `'a'`, `'x'`, `'yad'`.

```
>> a=sym('a')
a =
a
>> bb=sym('bb')
bb =
bb
>> x=sym('x');
>>
```

Create a symbolic object a and assign it to a.

The display of a symbolic object is not indented.

The symbolic variable x is created but not displayed, since a semicolon is typed at the end of the command.

Several symbolic variables can be created in one command by using the `syms` command, which has the form:

```
syms variable_name variable_name variable_name
```

The command creates symbolic objects that have the same names as the symbolic variables. For example, the variables *y*, *z*, and *d* can all be created as symbolic variables in one command by typing:

```
>> syms y z d
>> y
y =
y
```

The variables created by the `syms` command are not displayed automatically. Typing the name of the variable shows that the variable was created.

When the `syms` command is executed, the variables it creates are not displayed automatically—even if a semicolon is not typed at the end of the command.

11.1.2 Creating Symbolic Expressions

Symbolic expressions are mathematical expressions written in terms of symbolic variables. Once symbolic variables are created, they can be used for creating symbolic expressions. The symbolic expression is a symbolic object (the display is not indented). The form for creating a symbolic expression is:

```
Expression_name = Mathematical expression
```

A few examples are:

```
>> syms a b c x y
>> f=a*x^2+b*x + c
f =
a*x^2 + b*x + c
```

Define a, b, c, x, and y as symbolic variables.

Create the symbolic expression $ax^2 + bx + c$ and assign it to f.

The display of the symbolic expression is not indented.

When a symbolic expression, which includes mathematical operations that can be executed (addition, subtraction, multiplication, and division), is entered, MATLAB executes the operations as the expression is created. For example:

```
>> g=2*a/3+4*a/7-6.5*x+x/3+4*5/3-1.5
```

$\frac{2a}{3}+\frac{4a}{7}-6.5x+\frac{x}{3}+4\cdot\frac{5}{3}-1.5$ is entered.

```
g =
(26*a)/21 - (37*x)/6 + 31/6
```

$\frac{26a}{21}-\frac{37x}{6}+\frac{31}{6}$ is displayed.

11.2.2 The `simplify` *and* `simple` *Commands*

The `simplify` and `simple` commands are both general tools for simplifying the form of an expression. The `simplify` command uses built-in simplification rules to generate a simpler form of the expression than the original. The `simple` command is programmed to generate a form of the expression with the least number of characters. Although there is no guarantee that the form with the least number of characters is the simplest, in actuality this is often the case.

The `simplify` command:

The `simplify` command uses mathematical operations (addition, multiplication, rules of fractions, powers, logarithms, etc.) and functional and trigonometric identities to generate a simpler form of the expression. The format of the `simplify` command is:

```
simplify(S)
```

where either `S` is the name of the existing expression to be simplified, or an expression to be simplified can be typed in for `S`.

11.3 SOLVING ALGEBRAIC EQUATIONS

A single algebraic equation can be solved for one variable, and a system of equations can be solved for several variables with the `solve` function.

Solving a single equation:

An algebraic equation can have one or several symbolic variables. If the equation has one variable, the solution is numerical. If the equation has several symbolic variables, a solution can be obtained for any of the variables in terms of the others. The solution is obtained by using the `solve` command, which has the form

`h = solve(eq)` or `h = solve(eq,var)`

- The argument `eq` can be the name of a previously created symbolic expression, or an expression that is typed in. When a previously created symbolic expression `S` is entered for `eq`, or when an expression that does not contain the = sign is typed in for `eq`, MATLAB solves the equation `eq` = 0.
- An equation of the form $f(x) = g(x)$ can be solved by typing the equation (including the = sign) as a string for `eq`.
- If the equation to be solved has more than one variable, the `solve(eq)` command solves for the default symbolic variable (see Section 11.1.3). A solution for any of the variables can be obtained with the `solve(eq,var)` command by typing the variable name for `var`.
- If the user types `solve(eq)`, the solution is assigned to the variable `ans`.
- If the equation has more than one solution, the output `h` is a symbolic column vector with a solution at each element. The elements of the vector are symbolic objects. When an array of symbolic objects is displayed, each row is enclosed with square brackets (see the following examples).

The following examples illustrate the use of the `solve` command.

```
>> syms a b x y z
>> h=solve(exp(2*z)-5)
h =
log(5)/2
>> S=x^2-x-6
S =
x^2-x-6
>> k=solve(S)
k =
 -2
  3
```

Define a, b, x, y, and z as symbolic variables.

Use the `solve` command to solve $e^{2z} - 5 = 0$.

The solution is assigned to `h`.

Create the symbolic expression $x^2 - x - 6$, and assign it to `S`.

Use the `solve(S)` command to solve $x^2 - x - 6 = 0$.

The equation has two solutions. They are assigned to `k`, which is a column vector with symbolic objects.

```
>> solve('cos(2*y)+3*sin(y)=2')
ans =
     pi/2
     pi/6
  (5*pi)/6
>> T= a*x^2+5*b*x+20
T =
a*x^2+5*b*x+20
>> solve(T)
ans =
 -(5*b+5^(1/2)*(5*b^2-16*a)^(1/2))/(2*a)
 -(5*b-5^(1/2)*(5*b^2-16*a)^(1/2))/(2*a)
>> M = solve(T,a)
M =
-(5*b*x+20)/x^2
```

Use the `solve` command to solve $\cos(2y) + 3\sin(y) = 2$. (The equation is typed as a string in the command.)

The solution is assigned to `ans`.

Create the symbolic expression $ax^2 + 5bx + 20$, and assign it to `T`.

Use the `solve(S)` command to solve $T = 0$.

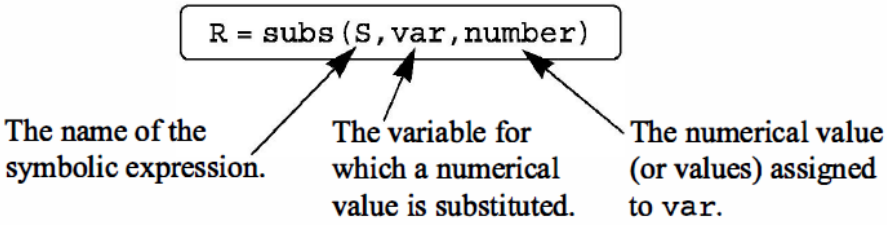
The equation $T = 0$ is solved for the variable x, which is the default variable.

Use the `solve(eq,var)` command to solve $T = 0$.

The equation $T = 0$ is solved for the variable a.

Substituting a numerical value for one symbolic variable:

A numerical value (or values) can be substituted for one symbolic variable when a symbolic expression has one or more symbolic variables. In this case the `subs` command has the form:



- `number` can be one number (a scalar), or an array with many elements (a vector or a matrix).

- The value of `S` is calculated for each value of `number` and the result is assigned to `R`, which will have the same size as `number` (scalar, vector, or matrix).
- If `S` has one variable, the output `R` is numerical. If `S` has several variables and a numerical value is substituted for only one of them, the output `R` is a symbolic expression.

An example with an expression that includes one symbolic variable is:

```
>> syms x
>> S=0.8*x^3+4*exp(0.5*x)
S =
4*exp(x/2) + (4*x^3)/5
>> SD=diff(S)
SD =
2*exp(x/2)+(12*x^2)/5)
>> subs(SD, x, 2)
ans =
   15.0366
>> SDU=subs(SD, x, [2:0.5:4])
SDU =
   15.0366   21.9807   30.5634   40.9092   53.1781
```

Define x as a symbolic variable.

Assign to S the expression $0.8x^3 + 4e^{(0.5x)}$.

Use the `diff(S)` command to differentiate S.

The answer $2e^{x/2} + 12x^2/5$ is assigned to SD.

Use the `subs` command to substitute $x = 2$ in SD.

The value of SD is displayed.

Use the `subs` command to substitute $x = [2, 2.5, 3, 3.5, 4]$ (vector) in SD.

The values of SD (assigned to SDU) for each value of x are displayed in a vector.

Task 2

Define x as a symbolic variable and create the two symbolic expressions

$$S_1 = x^2(x-6) + 4(3x-2) \text{ and } S_2 = (x+2)^2 - 8x$$

Use symbolic operations to determine the simplest form of each of following expressions:

(*a*) $S_1 \cdot S_2$ (*b*) $\frac{S_1}{S_2}$ (*c*) $S_1 + S_2$

(*d*) Use the `subs` command to evaluate the numerical value of the result from part (*c*) for $x = 5$.

Define x and y as symbolic variables and create the two symbolic expressions

$$S = x + \sqrt{x}y^2 + y^4 \text{ and } T = \sqrt{x} - y^2$$

Use symbolic operations to determine the simplest form of $S \cdot T$. Use the `subs` command to evaluate the numerical value of the result for $x = 9$ and $y = 2$.

❖ Derivatives of Polynomials

Method 1:

8.1.4 Derivatives of Polynomials

The built-in function `polyder` can be used to calculate the derivative of a single polynomial, a product of two polynomials, or a quotient of two polynomials, as shown in the following three commands.

`k = polyder(p)`	Derivative of a single polynomial. `p` is a vector with the coefficients of the polynomial. `k` is a vector with the coefficients of the polynomial that is the derivative.
`k = polyder(a,b)`	Derivative of a product of two polynomials. `a` and `b` are vectors with the coefficients of the polynomials that are multiplied. `k` is a vector with the coefficients of the polynomial that is the derivative of the product.
`[n d] = polyder(u,v)`	Derivative of a quotient of two polynomials. `u` and `v` are vectors with the coefficients of the numerator and denominator polynomials. `n` and `d` are vectors with the coefficients of the numerator and denominator polynomials in the quotient that is the derivative.

The only difference between the last two commands is the number of output arguments. With two output arguments MATLAB calculates the derivative of the quotient of two polynomials. With one output argument, the derivative is of the product.

For example, if $f_1(x) = 3x^2 - 2x + 4$, and $f_2(x) = x^2 + 5$, the derivatives of $3x^2 - 2x + 4$, $(3x^2 - 2x + 4)(x^2 + 5)$, and $\frac{3x^2 - 2x + 4}{x^2 + 5}$ can be determined by:

```
>> f1= 3 -2 4];
>> f2=[1 0 5];
>> k=polyder(f1)
k =
     6    -2
>> d=polyder(f1,f2)
d =
    12    -6    38   -10
>> [n d]=polyder(f1,f2)
n =
     2    22   -10
d =
     1     0    10     0    25
```

Creating the vectors of coefficients of f_1 and f_2.

The derivative of f_1 is: $6x - 2$.

The derivative of $f_1 * f_2$ is: $12x^3 - 6x^2 + 38x - 10$.

The derivative of $\frac{3x^2 - 2x + 4}{x^2 + 5}$ is: $\frac{2x^2 + 22x - 10}{x^4 + 10x^2 + 25}$.

Method 2 (Treated as a symbolic expression)

11.4 DIFFERENTIATION

Symbolic differentiation can be carried out by using the `diff` command. The form of the command is:

`diff(S)` or `diff(S,var)`

- Either `S` can be the name of a previously created symbolic expression, or an expression can be typed in for `S`.
- In the `diff(S)` command, if the expression contains one symbolic variable, the differentiation is carried out with respect to that variable. If the expression contains more than one variable, the differentiation is carried out with respect to the default symbolic variable (Section 11.1.3).
- In the `diff(S,var)` command (which is used for differentiation of expressions with several symbolic variables) the differentiation is carried out with respect to the variable `var`.
- The second or higher (*n*th) derivative can be determined with the `diff(S,n)` or `diff(S,var,n)` command, where `n` is a positive number. $n = 2$ for the second derivative, $n = 3$ for the third, and so on.

Some examples are:

```
>> syms x y t
>> S=exp(x^4);
>> diff(S)
ans =
4*x^3*exp(x^4)
>> diff((1-4*x)^3)
ans =
-12*(1-4*x)^2
>> R=5*y^2*cos(3*t);
>> diff(R)
ans =
10*y*cos(3*t)
>> diff(R,t)
ans =
-15*y^2*sin(3*t)
>> diff(S,2)
ans =
12*x^2*exp(x^4)+16*x^6*exp(x^4)
```

Define `x`, `y`, and `t` as symbolic variables.

Assign to `S` the expression e^{x^4}.

Use the `diff(S)` command to differentiate `S`.

The answer $4x^3e^{x^4}$ is displayed.

Use the `diff(S)` command to differentiate $(1-4x)^3$.

The answer $-12(1-4x)^2$ is displayed.

Assign to `R` the expression $5y^2\cos(3t)$.

Use the `diff(R)` command to differentiate `R`.

MATLAB differentiates `R` with respect to y (default symbolic variable); the answer $10y\cos(3t)$ is displayed.

Use the `diff(R,t)` command to differentiate `R` w.r.t. t.

The answer $-15y^2\sin(3t)$ is displayed.

Use `diff(S,2)` command to obtain the second derivative of `S`.

The answer $12x^2e^{x^4} + 16x^6e^{x^4}$ is displayed.

Task 3 (using the two methods above)

1- Find the differentiation for the following equation:

$$3x^4+4x^2-20$$

❖ Integration

11.5 INTEGRATION

Symbolic integration can be carried out by using the `int` command. The command can be used for determining indefinite integrals (antiderivatives) and definite integrals. For indefinite integration the form of the command is:

`int(S)` or `int(S,var)`

- Either `S` can be the name of a previously created symbolic expression, or an expression can be typed in for `S`.
- In the `int(S)` command, if the expression contains one symbolic variable, the integration is carried out with respect to that variable. If the expression contains more than one variable, the integration is carried out with respect to the default symbolic variable (Section 11.1.3).
- In the `int(S,var)` command, which is used for integration of expressions with several symbolic variables, the integration is carried out with respect to the variable `var`.

Some examples are:

```
>> syms x y t
>> S=2*cos(x)-6*x;
>> int(S)
ans =
2*sin(x)-3*x^2
>> int(x*sin(x))
ans =
sin(x)-x*cos(x)
>>R=5*y^2*cos(4*t);
>> int(R)
ans =
(5*y^3*cos(4*t))/3
>> int(R,t)
ans =
(5*y^2*sin(4*t))/4
```

Define `x`, `y`, and `t` as symbolic variables.

Assign to `S` the expression $2\cos(x)-6x$.

Use the `int(S)` command to integrate `S`.

The answer $2\sin(x)-3x^2$ is displayed.

Use the `int(S)` command to integrate $x\sin(x)$.

The answer $\sin(x)-x\cos(x)$ is displayed.

Assign to `R` the expression $5y^2\cos(4t)$.

Use the `int(R)` command to integrate `R`.

MATLAB integrates `R` with respect to y (default symbolic variable); the answer $5y^3\cos(4t)/3$ is displayed.

Use the `int(R,t)` command to integrate `R` w.r.t. t.

The answer $5y^2\sin(4t)/4$ is displayed.

For definite integration the form of the command is:

`int(S,a,b)` or `int(S,var,a,b)`

- `a` and `b` are the limits of integration. The limits can be numbers or symbolic variables.

For example, determination of the definite integral $\int_0^{\pi}(\sin y - 5y^2)dy$ with MATLAB is:

```
>> syms y
>> int(sin(y)-5*y^2,0,pi)
ans =
2 - (5*pi^3)/3
```

- It is possible also to use the `int` command by typing the expression to be integrated as a string without having the variables in the expression first created as symbolic objects. However, the variables in the integrated expression do not exist as independent symbolic objects.
- Integration can sometimes be a difficult task. A closed-form answer may not exist, or if it exists, MATLAB might not be able to find it. When that happens MATLAB returns `int(S)` and the message `Explicit integral could not be found.`

Task 4

16. Evaluate the following indefinite integrals:

(a) $I = \int \frac{x^3}{\sqrt{1-x^2}}\,dx$ (b) $I = \int x^2 \cos x\,dx$

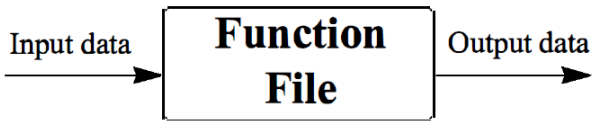
1- Evaluate the definite integral for the following integral:

I= ∫_{-1}^{3} 3x^4 − 4^2 + 10x − 25 dx

- **User-Defined Functions and Function Files**
- **Polynomials, Curve Fitting, and Interpolation**
- **2-D plots**

❖ User-Defined Functions and Function Files

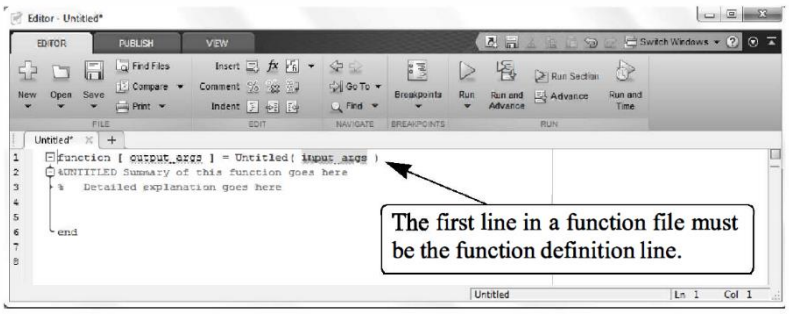
A user-defined function is a MATLAB program that is created by the user, saved as a function file, and then used like a built-in function. The function can be a simple single mathematical expression or a complicated and involved series of calculations. In many cases it is actually a subprogram within a computer program. The main feature of a function file is that it has an input and an output. This means that the calculations in the function file are carried out using the input data, and the results of the calculations are transferred out of the function file by the output. The input and the output can be one or several variables, and each can be a scalar, a vector, or an array of any size. Schematically, a function file can be illustrated by:



7.1 CREATING A FUNCTION FILE

Function files are created and edited, like script files, in the Editor/Debugger Window. This window is opened from the Command Window. In the Toolstrip select **New**, then select **Function**. Once the Editor/Debugger Window opens, it looks like that shown in Figure 7-1. The editor contains several pre-typed lines that outline the structure of a function file. The first line is the function definition line, which is followed by comments the describe the function. Next comes the program (the empty lines 4 and 5 in Figure 7-1), and the last line is an `end` statement, which is optional. The structure of a function file is described in detail in the next section.

Note: The Editor/Debugger Window can also be opened (as was described in Chapter 1) by clicking on the **New Script** icon in the Toolstrip, or by clicking **New** in the Toolstrip and then selecting **Script** from the menu that open. The win-



7.2 *Structure of a Function File*

The structure of a typical complete function file is shown in Figure 7-2. This particular function calculates the monthly payment and the total payment of a loan. The inputs to the function are the amount of the loan, the annual interest rate, and the duration of the loan (number of years). The output from the function is the monthly payment and the total payment.

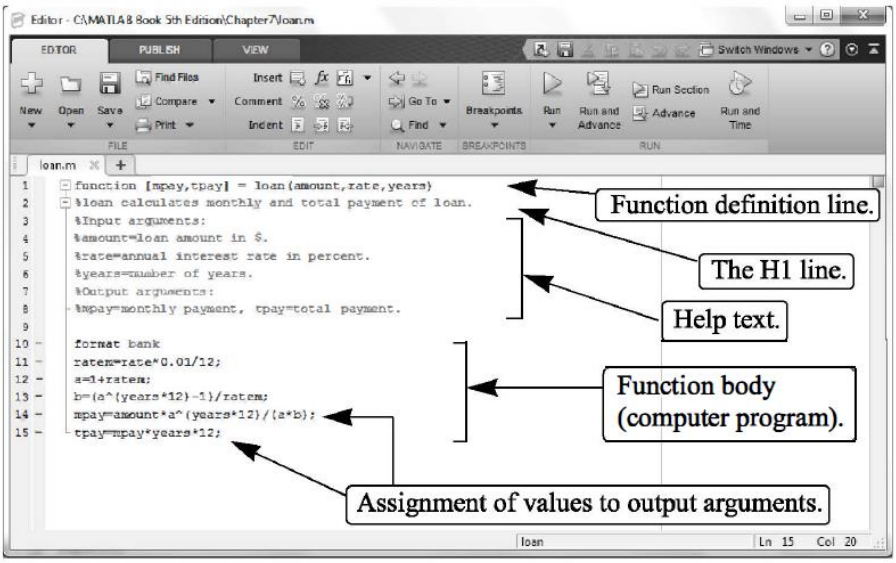


Figure 7-2: Structure of a typical function file.

7.2.1 Function Definition Line

The first executable line in a function file ***must*** be the function definition line. Otherwise the file is considered a script file. The function definition line:

- Defines the file as a function file
- Defines the name of the function
- Defines the number and order of the input and output arguments

The form of the function definition line is:

```
function [output arguments] = function_name(input arguments)
```

The word "function" must be the first word, and must be typed in lowercase letters.

A list of output arguments typed inside brackets.

The name of the function.

A list of input arguments typed inside parentheses.

The word "function," typed in lowercase letters, must be the first word in the function definition line. On the screen the word function appears in blue. The function name is typed following the equal sign. The name can be made up of letters, digits, and the underscore character (the name cannot include a space). The rules for the name are the same as the rules for naming variables described in Section 1.6.2. It is good practice to avoid names of built-in functions and names of variables already defined by the user or predefined by MATLAB.

Examples of function definition line:

Function definition line	**Comments**
function [mpay,tpay] = loan(amount,rate,years)	Three input arguments, two output arguments.
function [A] = RectArea(a,b)	Two input arguments, one output argument.
function A = RectArea(a,b)	Same as above; one output argument can be typed without the brackets.
function [V, S] = SphereVolArea(r)	One input variable, two output variables.
function trajectory(v,h,g)	Three input arguments, no output arguments.

7.4 SAVING A FUNCTION FILE

A function file must be saved before it can be used. This is done, as with a script file, by choosing **Save as . . .** from the **File** menu, selecting a location (many students save to a flash drive), and entering the file name. It is highly recommended that the file be saved with a name that is identical to the function name in the function definition line. In this way the function is called (used) by using the function name. (If a function file is saved with a different name, the name it is saved under must be used when the function is called.) Function files are saved with the extension .m. Examples:

Function definition line	File name
function [mpay,tpay] = loan(amount,rate,years)	loan.m
function [A] = RectArea(a,b)	RectArea.m
function [V, S] = SphereVolArea(r)	SphereVolArea.m
function trajectory(v,h,g)	trajectory.m

Task 1:(2.5 pts)

Sample Problem 7-1: User-defined function for a math function

Write a function file (name it `chp7one`) for the function $f(x) = \frac{x^4\sqrt{3x+5}}{(x^2+1)^2}$. The input to the function is x and the output is $f(x)$. Write the function such that x can be a vector. Use the function to calculate:
(*a*) $f(x)$ for $x = 6$.
(*b*) $f(x)$ for $x = 1, 3, 5, 7, 9,$ and 11.

Submit both your function file, and your script file.

❖ Polynomials, Curve Fitting, and Interpolation

9.2 FINDING A MINIMUM OR A MAXIMUM OF A FUNCTION

In many applications there is a need to determine the local minimum or maximum of a function of the form $y = f(x)$. In calculus the value of x that corresponds to a local minimum or maximum is determined by finding the zero of the derivative of the function. The value of y is determined by substituting the x into the function. In MATLAB the value of x where a one-variable function $f(x)$ within the interval $x_1 \le x \le x_2$ has a minimum can be determined with the `fminbnd` command which has the form:

```
x = fminbnd(function,x1,x2)
```

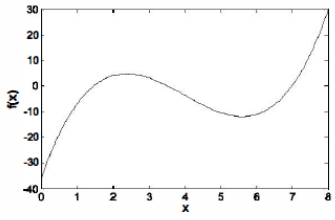
The value of x where the function has a minimum. The function. The interval of x.

- The function can be entered as a string expression, or as a function handle, in the same way as with the `fzero` command. See Section 9.1 for details.
- The value of the function at the minimum can be added to the output by using the option

```
[x fval]=fminbnd(function,x1,x2)
```

  where the value of the function at `x` is assigned to the variable `fval`.
- Within a given interval, the minimum of a function can either be at one of the end points of the interval or at a point within the interval where the slope of the function is zero (local minimum). When the `fminbnd` command is executed, MATLAB looks for a local minimum. If a local minimum is found, its value is compared to the value of the function at the end points of the interval. MATLAB returns the point with the actual minimum value for the interval.

For example, consider the function $f(x) = x^3 - 12x^2 + 40.25x - 36.5$, which is plotted in the interval $0 \le x \le 8$ in the figure on the right. It can be observed that there is a local minimum between 5 and 6, and that the absolute minimum is at $x = 0$. Using the `fminbnd` command with the interval $3 \le x \le 8$ to find the location of the local minimum and the value of the function at this point gives:




```
>> [x fval]=fminbnd('x^3-12*x^2+40.25*x-36.5',3,8)
x =
    5.6073
fval =
  -11.8043
```

The local minimum is at $x = 5.6073$. The value of the function at this point is –11.8043.

Notice that the `fminbnd` command gives the local minimum. If the interval is changed to $0 \le x \le 8$, `fminbnd` gives:

```
>> [x fval]=fminbnd('x^3-12*x^2+40.25*x-36.5',0,8)
x =
     0
fval =
  -36.5000
```

The minimum is at $x = 0$. The value of the function at this point is –36.5.

For this interval the `fminbnd` command gives the absolute minimum which is at the end point $x = 0$.

- The `fminbnd` command can also be used to find the maximum of a function. This is done by multiplying the function by –1 and finding the minimum. For example, the maximum of the function $f(x) = xe^{-x} - 0.2$ (from Sample Problem 9-1) in the interval $0 \le x \le 8$ can be determined by finding the minimum of the function $f(x) = -xe^{-x} + 0.2$ as shown below:

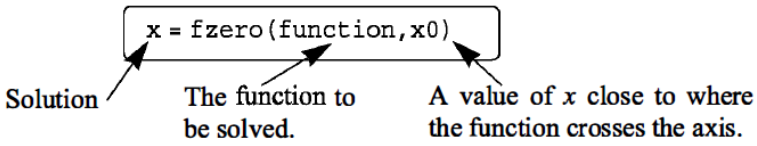
```
>> [x fval]=fminbnd('-x*exp(-x)+0.2',0,8)
x =
    1.0000
fval =
   -0.1679
```

The maximum is at $x = 1.0$. The value of the function at this point is 0.1679.

9.1 SOLVING AN EQUATION WITH ONE VARIABLE

An equation with one variable can be written in the form $f(x) = 0$. A solution to the equation (also called a root) is a numerical value of x that satisfies the equation. Graphically, a solution is a point where the function $f(x)$ crosses or touches the x axis. An exact solution is a value of x for which the value of the function is exactly zero. If such a value does not exist or is difficult to determine, a numerical solution can be determined by finding an x that is very close to the solution. This is done by the iterative process, where in each iteration the computer determines a value of x that is closer to the solution. The iterations stop when the difference in x between two iterations is smaller than some measure. In general, a function can have zero, one, several, or an infinite number of solutions.

In MATLAB a zero of a function can be determined with the command (built-in function) `fzero` with the form:

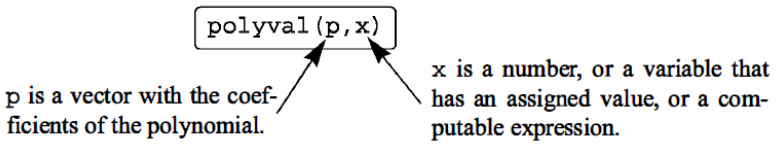


The built-in function `fzero` is a MATLAB function function (see Section 7.9), which means that it accepts another function (the function to be solved) as an input argument.

- The function has to be written in a standard form. For example, if the function to be solved is $xe^{-x} = 0.2$, it has to be written as $f(x) = xe^{-x} - 0.2 = 0$. If this function is entered into the `fzero` command as a string, it is typed as: `'x*exp(-x)-0.2'`.
- When a function is entered as an expression (string), it cannot include predefined variables. For example, if the function to be entered is $f(x) = xe^{-x} - 0.2$, it is not possible to define `b=0.2` and then enter `'x*exp(-x)-b'`.
- `x0` can be a scalar or a two-element vector. If it is entered as a scalar, it has to be a value of x near the point where the function crosses (or touches) the x axis. If `x0` is entered as a vector, the two elements have to be points on opposite sides of the solution. If $f(x)$ crosses the x axis, then $f(x0(1))$ has a different sign than $f(x0(2))$. When a function has more than one solution, each solution can be determined separately by using the `fzero` function and entering values for `x0` that are near each of the solutions.

8.1.1 Value of a Polynomial

The value of a polynomial at a point x can be calculated with the function `polyval` that has the form:



`x` can also be a vector or a matrix. In such a case the polynomial is calculated for each element (element-by-element), and the answer is a vector, or a matrix, with the corresponding values of the polynomial.

Sample Problem 8-1: Calculating polynomials with MATLAB

For the polynomial $f(x) = x^5 - 12.1x^4 + 40.59x^3 - 17.015x^2 - 71.95x + 35.88$:

(*a*) Calculate $f(9)$.

(*b*) Plot the polynomial for $-1.5 \le x \le 6.7$.

Solution

The problem is solved in the Command Window.

(*a*) The coefficients of the polynomials are assigned to vector `p`. The function `polyval` is then used to calculate the value at $x = 9$.

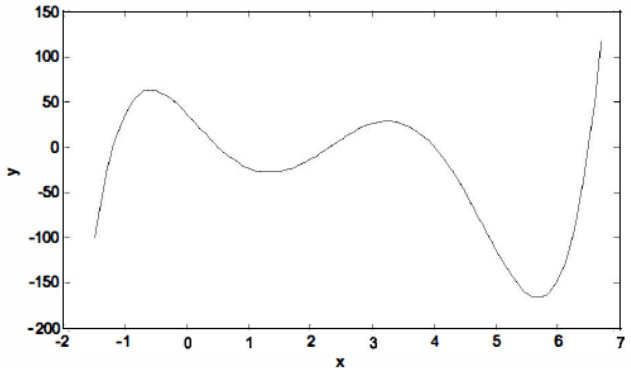
```
>> p = [1 -12.1 40.59 -17.015 -71.95 35.88];
>> polyval(p,9)
ans =
  7.2611e+003
```

(*b*) To plot the polynomial, a vector `x` is first defined with elements ranging from –1.5 to 6.7. Then a vector `y` is created with the values of the polynomial for every element of `x`. Finally, a plot of `y` vs. `x` is made.

```
>> x=-1.5:0.1:6.7;
>> y=polyval(p,x);
>> plot(x,y)
```

Calculating the value of the polynomial for each element of the vector `x`.

The plot created by MATLAB is presented below (axis labels were added with the Plot Editor).



8.2 CURVE FITTING

Curve fitting, also called regression analysis, is a process of fitting a function to a set of data points. The function can then be used as a mathematical model of the data. Since there are many types of functions (linear, polynomial, power, exponential, etc.), curve fitting can be a complicated process. Many times one has some idea of the type of function that might fit the given data and will need only to determine the coefficients of the function. In other situations, where nothing is known about the data, it is possible to make different types of plots that provide information about possible forms of functions that might fit the data well. This section describes some of the basic techniques for curve fitting and the tools that MATLAB has for this purpose.

8.2.1 Curve Fitting with Polynomials; The `polyfit` Function

Polynomials can be used to fit data points in two ways. In one the polynomial passes through all the data points, and in the other the polynomial does not necessarily pass through any of the points but overall gives a good approximation of the data. The two options are described below.

Polynomials that pass through all the points:

When n points (x_i, y_i) are given, it is possible to write a polynomial of degree $n-1$ that passes through all the points. For example, if two points are given it is possible to write a linear equation in the form of $y = mx + b$ that passes through the points. With three points, the equation has the form of $y = ax^2 + bx + c$. With n points the polynomial has the form $a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_1x + a_0$. The coefficients of the polynomial are determined by substituting each point in the polynomial and then solving the n equations for the coefficients. As will be shown later in this section, polynomials of high degree might give a large error if they are used to estimate values between data points.

Polynomials that do not necessarily pass through any of the points:

When n points are given, it is possible to write a polynomial of degree less than $n-1$ that does not necessarily pass through any of the points but that overall approximates the data. The most common method of finding the best fit to data points is the method of least squares. In this method, the coefficients of the polynomial are determined by minimizing the sum of the squares of the residuals at all the data points. The residual at each point is defined as the difference between the value of the polynomial and the value of the data. For example, consider the case of finding the equation of a straight line that best fits four data points as shown in Figure 8-1. The points are (x_1,y_1), (x_2,y_2), (x_3,y_3), and (x_4,y_4), and the polyno-

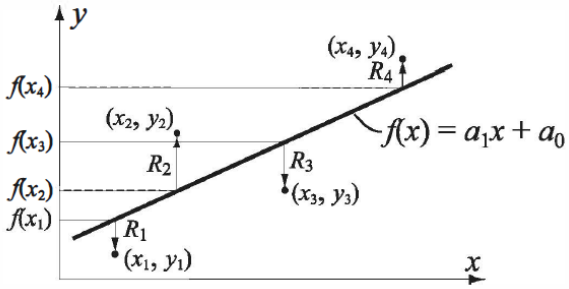


Figure 8-1: Least squares fitting of first-degree polynomial to four points.

mial of the first degree can be written as $f(x) = a_1x + a_0$. The residual, R_i, at each point is the difference between the value of the function at x_i and y_i, $R_i = f(x_i) - y_i$. An equation for the sum of the squares of the residuals R_i of all the points is given by:

$$R = [f(x_1) - y_1]^2 + [f(x_2) - y_2]^2 + [f(x_3) - y_3]^2 + [f(x_4) - y_4]^2$$

or, after substituting the equation of the polynomial at each point, by:

$$R = [a_1x_1 + a_0 - y_1]^2 + [a_1x_2 + a_0 - y_2]^2 + [a_1x_3 + a_0 - y_3]^2 + [a_1x_4 + a_0 - y_4]^2$$

At this stage R is a function of a_1 and a_0. The minimum of R can be determined by taking the partial derivative of R with respect to a_1 and a_0 (two equations) and equating them to zero:

$$\frac{\partial R}{\partial a_1} = 0 \quad \text{and} \quad \frac{\partial R}{\partial a_0} = 0$$

This results in a system of two equations with two unknowns, a_1 and a_0. The solution of these equations gives the values of the coefficients of the polynomial that best fits the data. The same procedure can be followed with more points and higher-order polynomials. More details on the least squares method can be found in books on numerical analysis.

Curve fitting with polynomials is done in MATLAB with the `polyfit` function, which uses the least squares method. The basic form of the `polyfit` function is:

```
p = polyfit(x,y,n)
```

`p` is the vector of the coefficients of the polynomial that fits the data.

`x` is a vector with the horizontal coordinates of the data points (independent variable).
`y` is a vector with the vertical coordinates of the data points (dependent variable).
`n` is the degree of the polynomial.

For the same set of m points, the `polyfit` function can be used to fit polynomials of any order up to $m-1$. If $n=1$ the polynomial is a straight line, if $n=2$ the polynomial is a parabola, and so on. The polynomial passes through all the points if $n = m-1$ (the order of the polynomial is one less than the number of points). It should be pointed out here that a polynomial that passes through all the points, or polynomials with higher order, do not necessarily give a better fit overall. High-order polynomials can deviate significantly between the data points.

Figure 8-2 shows how polynomials of different degrees fit the same set of data points. A set of seven points is given by (0.9, 0.9), (1.5, 1.5), (3, 2.5), (4, 5.1),

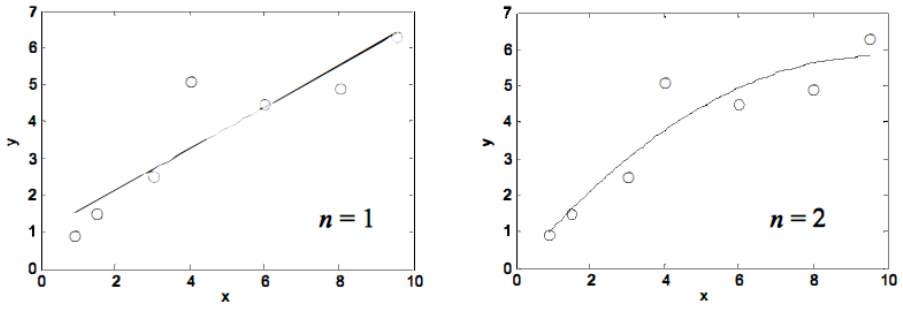


Figure 8-2: Fitting data with polynomials of different order.

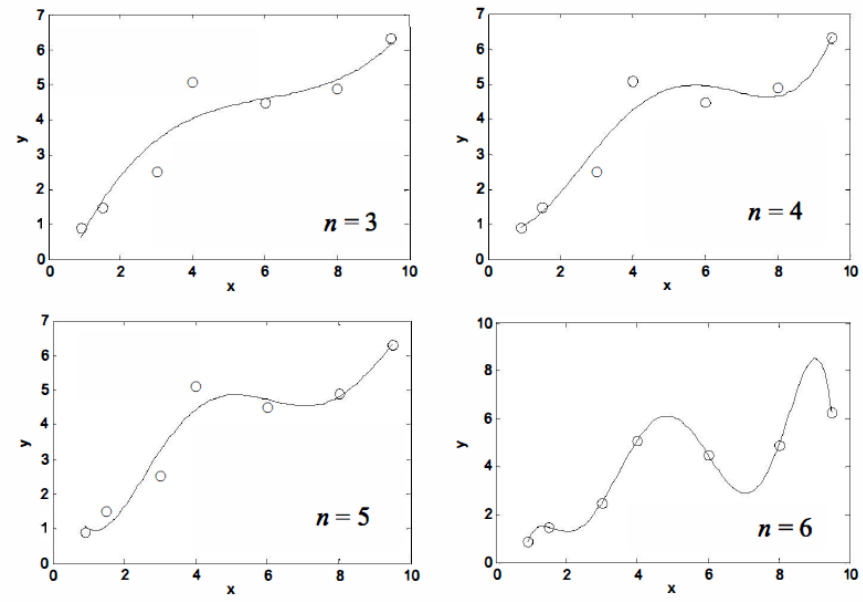


Figure 8-2: Fitting data with polynomials of different order. (Continued)

(6, 4.5), (8, 4.9), and (9.5, 6.3). The points are fitted using the `polyfit` function with polynomials of degrees 1 through 6. Each plot in Figure 8-2 shows the same data points, marked with circles, and a curve-fitted line that corresponds to a polynomial of the specified degree. It can be seen that the polynomial with $n = 1$ is a straight line, and that with $n = 2$ is a slightly curved line. As the degree of the polynomial increases, the line develops more bends such that it passes closer to more points. When $n = 6$, which is one less than the number of points, the line passes through all the points. However, between some of the points, the line deviates significantly from the trend of the data.

The script file used to generate one of the plots in Figure 8-2 (the polynomial with $n = 3$) is shown below. Note that in order to plot the polynomial (the line), a new vector xp with small spacing is created. This vector is then used with

```
x=[0.9 1.5 3 4 6 8 9.5];
y=[0.9 1.5 2.5 5.1 4.5 4.9 6.3];
p=polyfit(x,y,3)
xp=0.9:0.1:9.5;
yp=polyval(p,xp)
plot(x,y,'o',xp,yp)
xlabel('x'); ylabel('y')
```

Create vectors x and y with the coordinates of the data points.

Create a vector p using the `polyfit` function.

Create a vector xp to be used for plotting the polynomial.

Create a vector yp with values of the polynomial at each xp.

A plot of the seven points and the polynomial.

One-dimensional interpolation:

In a linear interpolation, the line between two data points has a constant slope, and there is a change in the slope at every point. A smoother interpolation curve can be obtained by using quadratic or cubic polynomials. In these methods, called quadratic splines and cubic splines, a second-, or third-order polynomial is used to interpolate between every two points. The coefficients of the polynomial are determined by using data from points that are adjacent to the two data points. The theoretical background for the determination of the constants of the polynomials is beyond the scope of this book and can be found in books on numerical analysis.

One-dimensional interpolation in MATLAB is done with the `interp1` (the last character is the numeral one) function, which has the form:

```
yi = interp1(x,y,xi,'method')
```

`yi` is the interpolated value.

`x` is a vector with the horizontal coordinates of the input data points (independent variable).
`y` is a vector with the vertical coordinates of the input data points (dependent variable).
`xi` is the horizontal coordinate of the interpolation point (independent variable).

Method of interpolation, typed as a string (optional).

- The vector `x` must be monotonic (with elements in ascending or descending order).
- `xi` can be a scalar (interpolation of one point) or a vector (interpolation of many points). `yi` is a scalar or a vector with the corresponding interpolated values.
- MATLAB can do the interpolation using one of several methods that can be specified. These methods include:

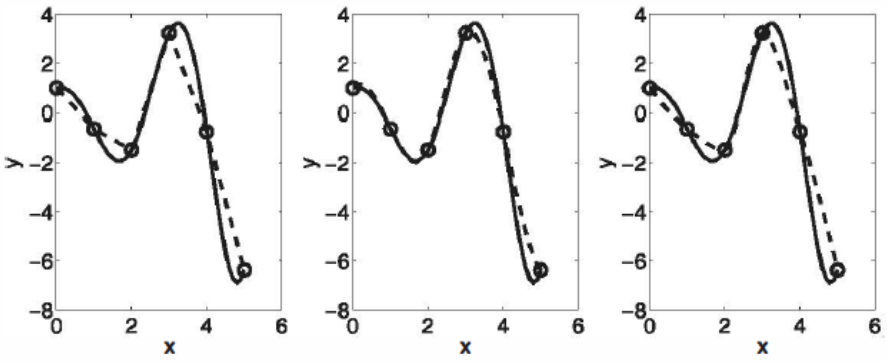
`'nearest'`	returns the value of the data point that is nearest to the interpolated point.
`'linear'`	uses linear spline interpolation.
`'spline'`	uses cubic spline interpolation.
`'pchip'`	uses piecewise cubic Hermite interpolation, also called `'cubic'`

- When the `'nearest'` and the `'linear'` methods are used, the value(s) of `xi` must be within the domain of `x`. If the `'spline'` or the `'pchip'` methods are used, `xi` can have values outside the domain of `x` and the function `interp1` performs extrapolation.
- The `'spline'` method can give large errors if the input data points are nonuniform such that some points are much closer together than others.
- Specification of the method is optional. If no method is specified, the default is `'linear'`.

Sample Problem 8-3: Interpolation

The following data points, which are points of the function $f(x) = 1.5^x \cos(2x)$, are given. Use linear, spline, and pchip interpolation methods to calculate the value of y between the points. Make a figure for each of the interpolation methods. In the figure show the points, a plot of the function, and a curve that corresponds to the interpolation method.

x	0	1	2	3	4	5
y	1.0	–0.6242	–1.4707	3.2406	–0.7366	–6.3717



Solution

The following is a program written in a script file that solves the problem:

```
x=0:1.0:5;                          Create vectors x and y with coordinates of the data points.
y=[1.0 -0.6242 -1.4707 3.2406 -0.7366 -6.3717];
xi=0:0.1:5;                         Create vector xi with points for interpolation.
yilin=interp1(x,y,xi,'linear');     Calculate y points from linear interpolation.
yispl=interp1(x,y,xi,'spline');     Calculate y points from spline interpolation.
yipch=interp1(x,y,xi,'pchip');      Calculate y points from pchip interpolation.
yfun=1.5.^xi.*cos(2*xi);            Calculate y points from the function.
subplot(1,3,1)
plot(x,y,'o',xi,yfun,xi,yilin,'--');
subplot(1,3,2)
plot(x,y,'o',xi,yfun,xi,yispl,'--');
subplot(1,3,3)
plot(x,y,'o',xi,yfun,xi,yipch,'--');
```

Task 2: (5 pts)

Refer to LMS for Task 2 details

❖ Two-Dimensional Plots

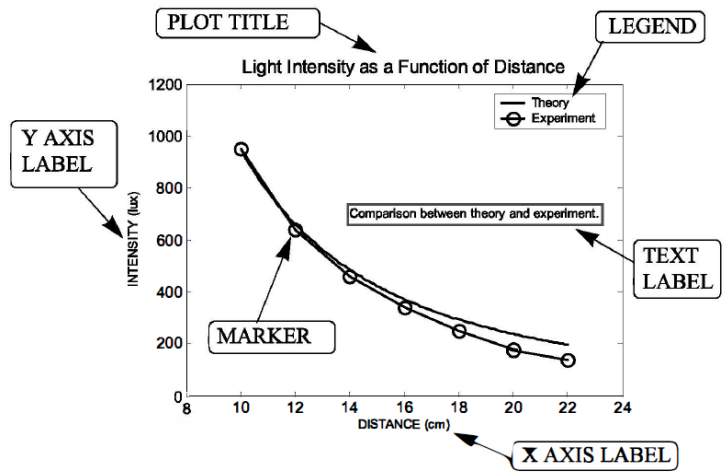
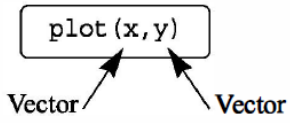


Figure 5-1: Example of a formatted two-dimensional plot.

5.1 THE `plot` COMMAND

The `plot` command is used to create two-dimensional plots. The simplest form of the command is:



The arguments `x` and `y` are each a vector (one-dimensional array). The two vectors ***must*** have the same number of elements. When the `plot` command is executed, a figure is created in the Figure Window. If not already open, the Figure

```
>> x=[1.1  1.8  3.2  5.5  7  7.5  8  10];
>> y=[2  6.5  7  7  5.5  4  6  8];
>> plot(x,y)
```

Once the `plot` command is executed, the Figure Window opens and the plot is displayed, as shown in Figure 5-2.

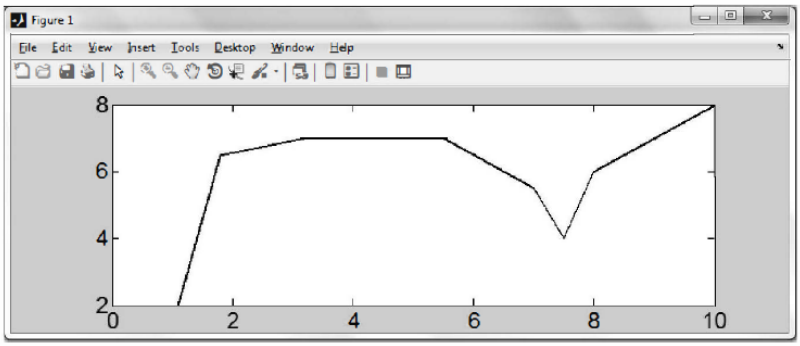


Figure 5-2: The Figure Window with a simple plot.

The plot appears on the screen in blue, which is the default line color.

The `plot` command has additional, optional arguments that can be used to specify the color and style of the line and the color and type of markers, if any are desired. With these options the command has the form:

```
plot(x,y,'line specifiers','PropertyName',PropertyValue)
```

Vector Vector — (Optional) Specifiers that define the type and color of the line and markers. — (Optional) Properties with values that can be used to specify the line width, and a marker's size and edge, and fill colors.

Line Specifiers:

Line specifiers are optional and can be used to define the style and color of the line and the type of markers (if markers are desired). The line style specifiers are:

Line Style	Specifier
solid (default)	-
dashed	--

Line Style	Specifier
dotted	:
dash-dot	-.

The line color specifiers are:

Line Color	Specifier
red	r
green	g
blue	b
cyan	c

Line Color	Specifier
magenta	m
yellow	y
black	k
white	w

The marker type specifiers are:

Marker Type	Specifier		Marker Type	Specifier
plus sign	+		square	s
circle	o		diamond	d
asterisk	*		five-pointed star	p
point	.		six-pointed star	h
cross	x		triangle (pointed left)	<
triangle (pointed up)	^		triangle (pointed right)	>
triangle (pointed down)	v			

Notes about using the specifiers:

- The specifiers are typed inside the `plot` command as strings.
- Within the string the specifiers can be typed in any order.
- The specifiers are optional. This means that none, one, two, or all three types can be included in a command.

Some examples:

`plot(x,y)`	A blue solid line connects the points with no markers (default).
`plot(x,y,'r')`	A red solid line connects the points.
`plot(x,y,'--y')`	A yellow dashed line connects the points.
`plot(x,y,'*')`	The points are marked with * (no line between the points).
`plot(x,y,'g:d')`	A green dotted line connects the points that are marked with diamond markers.

5.2 THE `fplot` COMMAND

The `fplot` command plots a function with the form $y = f(x)$ between specified limits. The command has the form:

```
fplot('function',limits,'line specifiers')
```

- The function to be plotted.
- The domain of x and, optionally, the limits of the y axis.
- Specifiers that define the type and color of the line and markers (optional).

`'function'`: The function can be typed directly as a string inside the command. For example, if the function that is being plotted is $f(x) = 8x^2 + 5\cos(x)$, it is typed as: `'8*x^2+5*cos(x)'`. The functions can include MATLAB built-in functions and functions that are created by the user (covered in Chapter 6).

- The function to be plotted can be typed as a function of any letter. For example, the function in the previous paragraph can be typed as `'8*z^2+5*cos(z)'` or `'8*t^2+5*cos(t)'`.

- The function cannot include previously defined variables. For example, in the function above it is not possible to assign 8 to a variable, and then use the variable when the function is typed in the `fplot` command.

`limits`: The limits argument is a vector with two elements that specify the domain of x [`xmin,xmax`], or a vector with four elements that specifies the domain of x and the limits of the y-axis [`xmin, xmax, ymin, ymax`].

`line specifiers`: The line specifiers are the same as in the `plot` command. For example, a plot of the function $y = x^2 + 4\sin(2x) - 1$ for $-3 \le x \le 3$ can be created with the `fplot` command by typing:

```
>> fplot('x^2+4*sin(2*x)-1',[-3 3])
```

in the Command Window. The figure that is obtained in the Figure Window is shown in Figure 5-6.

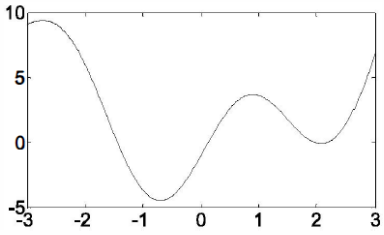


Figure 5-6: A plot of the function $y = x^2 + 4\sin(2x) - 1$.

5.3 PLOTTING MULTIPLE GRAPHS IN THE SAME PLOT

In many situations, there is a need to make several graphs in the same plot. This is shown, for example, in Figure 5-1 where two graphs are plotted in the same figure. There are three methods to plot multiple graphs in one figure. One is by using the `plot` command, the second is by using the `hold on` and `hold off` commands, and the third is by using the `line` command.

5.3.1 Using the `plot` Command

Two or more graphs can be created in the same plot by typing pairs of vectors inside the `plot` command. The command

```
plot(x,y,u,v,t,h)
```

creates three graphs—y vs. x, v vs. u, and h vs. t—all in the same plot. The vectors of each pair must be of the same length. MATLAB automatically plots the graphs in different colors so that they can be identified. It is also possible to add line specifiers following each pair. For example the command

```
plot(x,y,'-b',u,v,'--r',t,h,'g:')
```

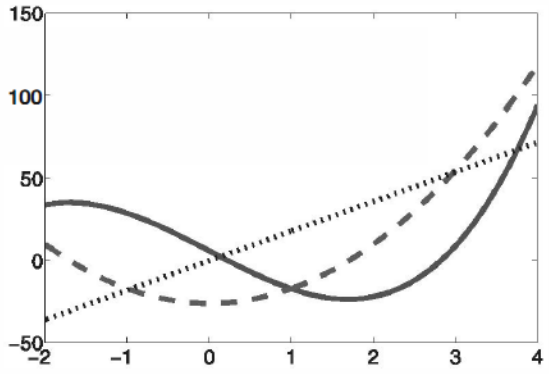
Sample Problem 5-1: Plotting a function and its derivatives

Plot the function $y = 3x^3 - 26x + 10$, and its first and second derivatives, for $-2 \le x \le 4$, all in the same plot.

Solution

The first derivative of the function is: $y' = 9x^2 - 26$.
The second derivative of the function is: $y'' = 18x$.
A script file that creates a vector x and calculates the values of y, y', and y'' is:

```
x=[-2:0.01:4];          Create vector x with the domain of the function.
y=3*x.^3-26*x+6;        Create vector y with the function value at each x.
yd=9*x.^2-26;           Create vector yd with values of the first derivative.
ydd=18*x;               Create vector ydd with values of the second derivative.
plot(x,y,'-b',x,yd,'--r',x,ydd,':k')
    Create three graphs, y vs. x, yd vs. x, and ydd vs. x, in the same figure.
```



5.3.2 *Using the* `hold on` *and* `hold off` *Commands*

To plot several graphs using the `hold on` and `hold off` commands, one graph is plotted first with the `plot` command. Then the `hold on` command is typed. This keeps the Figure Window with the first plot open, including the axis properties and formatting (see Section 5.4) if any was done. Additional graphs can be added with `plot` commands that are typed next. Each `plot` command creates a graph that is added to that figure. The `hold off` command stops this process. It returns MATLAB to the default mode, in which the `plot` command erases the previous plot and resets the axis properties.

As an example, a solution of Sample Problem 5-1 using the `hold on` and `hold off` commands is shown in the following script file:

```
x=[-2:0.01:4];
y=3*x.^3-26*x+6;
yd=9*x.^2-26;
ydd=18*x;
plot(x,y,'-b')
hold on
plot(x,yd,'--r')
plot(x,ydd,':k')
hold off
```

The first graph is created.

Two more graphs are added to the figure.

5.4 *Formatting a Plot*

The `plot` and `fplot` commands create bare plots. Usually, however, a figure that contains a plot needs to be formatted to have a specific look and to display information in addition to the graph itself. This can include specifying axis labels, plot title, legend, grid, range of custom axis, and text labels.

Plots can be formatted by using MATLAB commands that follow the `plot` or `fplot` command, or interactively by using the plot editor in the Figure Window. The first method is useful when a `plot` command is a part of a computer program (script file). When the formatting commands are included in the program, a formatted plot is created every time the program is executed. On the other hand, formatting that is done in the Figure Window with the plot editor after a plot has been created holds only for that specific plot, and will have to be repeated the next time the plot is created.

5.4.1 *Formatting a Plot Using Commands*

The formatting commands are entered after the `plot` or the `fplot` command. The various formatting commands are:

The `xlabel` and `ylabel` commands:

Labels can be placed next to the axes with the `xlabel` and `ylabel` command which have the form:

```
xlabel('text as string')
ylabel('text as string')
```

The `title` command:

A title can be added to the plot with the command:

```
title('text as string')
```

The `text` command:

A text label can be placed in the plot with the `text` or `gtext` commands:

```
text(x,y,'text as string')
gtext('text as string')
```

The `text` command places the text in the figure such that the first character is positioned at the point with the coordinates `x`, `y` (according to the axes of the figure). The `gtext` command places the text at a position specified by the user. When the command is executed, the Figure Window opens and the user specifies the position with the mouse.

The `legend` command:

The `legend` command places a legend on the plot. The legend shows a sample of the line type of each graph that is plotted, and places a label, specified by the user, beside the line sample. The form of the command is:

```
legend('string1','string2', ..... ,pos)
```

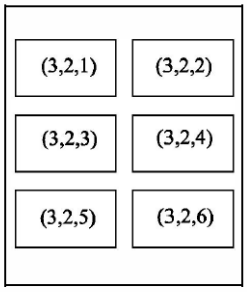
The strings are the labels that are placed next to the line sample. Their order corresponds to the order in which the graphs were created. The `pos` is an optional number that specifies where in the figure the legend is to be placed. The options are:

`pos = -1`	Places the legend outside the axes boundaries on the right side.
`pos = 0`	Places the legend inside the axes boundaries in a location that interferes the least with the graphs.
`pos = 1`	Places the legend at the upper-right corner of the plot (default).
`pos = 2`	Places the legend at the upper-left corner of the plot.
`pos = 3`	Places the legend at the lower-left corner of the plot.
`pos = 4`	Places the legend at the lower-right corner of the plot.

5.10 Putting Multiple Plots on the Same Page

Multiple plots can be created on the same page with the `subplot` command, which has the form:

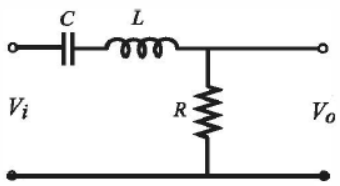
```
subplot(m,n,p)
```



The command divides the Figure Window (and the page when printed) into $m \times n$ rectangular subplots. The subplots are arranged like elements in an $m \times n$ matrix where each element is a subplot. The subplots are numbered from 1 through $m \cdot n$. The upper left subplot is numbered 1, and the lower right subplot is numbered $m \cdot n$. The numbers increase from left to right within a row, from the first row to the last. The command `subplot(m,n,p)` makes the subplot `p` current. This means that the next plot command (and any formatting commands) will create a plot (with the corresponding format) in this subplot. For example, the command `subplot(3,2,1)` creates six areas arranged in three rows and two columns as shown, and makes the upper left subplot current. An example of using the `subplot` command is shown in the solution of Sample Problem 5-2.

Task 3: (2.5 pts)

A bandpass filter passes signals with frequencies that are within a certain range. In this filter the ratio of the magnitudes of the voltages is given by



$$RV = \left|\frac{V_o}{V_i}\right| = \frac{\omega RC}{\sqrt{(1-\omega^2 LC)^2 + (\omega RC)^2}}$$

where ω is the frequency of the input signal. Given $R = 200\ \Omega$, $L = 8$ mH, and $C = 5\ \mu$F, make two plots of RV as a function of ω for $10 \le \omega \le 500000$. In the first plot use linear scale for both axis, and in the second plot use logarithmic scale for the horizontal (ω) axis, and linear scale for the vertical axis. Which plot provides a better illustration of the filter?

1. To do logarithmic scale, use the command

`semilogx(x,y)`	**Plots y versus x with a log (base 10) scale for the *x* axis and linear scale for the *y* axis.**

2. make sure to label your axis, and include a proper title for both graphs (both graphs should be in the same figure window)
3. Use red color line for the first graph, and black line for the second.

Name: **BMT 227 MATLAB**
ID # **KEY** **Introduction to Computer Systems**
Date: 10/11/2018 **Practical Quiz 1**

Q1 (1 point): Write a MATLAB code to evaluate the expression: (Remember, π is pi)
use MATLAB syntax!!!

$$\cos\left(\frac{7\pi}{9}\right) + \tan\left(\frac{7}{15}\pi\right)\sin(15°)$$

X=cos(7*pi/9)+tan(7*pi/15)*sind(15)
or
X=cos(7*pi/9)+tan(7/15*pi)*sind(15)

Q2 (1 point): Write MATLAB statements to find the roots of the polynomial:

$$3x + x^2 = 15$$

p=[1 3 -15]

r=roots(p)

Q3 (1 EC):

a. What is the function of *clc* command?

Clearing the command window

b. Create a time vector "T" from 0 to 0.5 sec using 2000 elements.

T=linspace(0,0.5,2000)

Name: **BMT 227 MATLAB**
ID # **KEY** **Introduction to Computer Systems**
Date: 10/18/2018 **Practical Quiz 2**

Q1 (1 point):

For the function $y = \frac{(x+5)^3}{x^2}$, calculate the value of y for the following values of x using element-by-element operations: 1, 2, 3, 4, 5, 6 .

Remember to use element-by-element operator!!!

X=linspace(1,6,6) or [1 2 3 4 5 6] or 1:6

Y= (x+5).^3 ./x.^2

Q2 (1 point): Write MATLAB statements to solve system of linear equations:

$$\begin{bmatrix} 4 & -2 \\ 2 & 8 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 8 \\ 4 \end{bmatrix}$$

```
A=[4 -2;2 8];
B=[8;4];
```

X= **A\B or inv(A)*B**

Q3 (1 EC):

The following two vectors are defined in MATLAB:

$v = [15, 8, -6]$ $u = [3, -2, 6]$

By hand (pencil and paper) write what will be displayed if the following commands are executed by MATLAB.

(*a*) `v./u` (*b*) `u'*v` (*c*) `u*v'`

a= [5 -4 -1]

c= [-7]

b=

45 24 -18
-30 -16 12
90 48 -36

Department of Biomedical Technology

INTRODUCTION TO COMPUTER SYSTEMS
BMT 227 Lab Manual

Lab 1: Introduction to MATLAB program

2018

Biomedical Technology
King Saud University
Riyadh, Kingdom of Saudi Arabia

Applications in the following operation

-basic operations (edit a command – save – open - run script, …)
-arrays or row vectors
-variables and matrices
-complex numbers
-polynomials and roots
-expressions and string functions
- plot a signal
-function M-files and script M-files

Important notes:

***Please **save** your work for all the practices
***Write the computer's number which you used during the MATLAB sessions.

Department of Biomedical Technology

INTRODUCTION TO COMPUTER SYSTEMS
BMT 227 Lab Manual

Lab 2: Applications on MATLAB

2018

Biomedical Technology
King Saud University
Riyadh, Kingdom of Saudi Arabia

Use the commands which you learned in the 1st prac. to execute the following problems:-

1- Find the roots for the following polynomial equations.

a- $5x^2+x^3+25=0$

-5.7549+0.000 i
0.3774+ 2.0498i

>> roots(a) = [1 5 0 25]

b- $3x+x^2=15$

-1.5 ±3.5707i

>> b = [1 3 15]

>> roots(b) =

2- Write the next matrix

$$A=\begin{matrix} 5 & 3 & 2 \\ 4 & 9 & 12 \\ 30 & 40 & 50 \end{matrix}$$

Change it to be as following

$$B=\begin{matrix} 5 & 3 & 2 \\ 2 & 3 & 2 \\ 10 & 10 & 10 \end{matrix}$$

A(3,:)=p

Calculate the matrices

C=A.*B

E=A.*B

C=A.*B

51 44 36
158 159 146
730 710 640

25 9 4
8 27 24
300 400 500

D=A/B

>> D=A./B

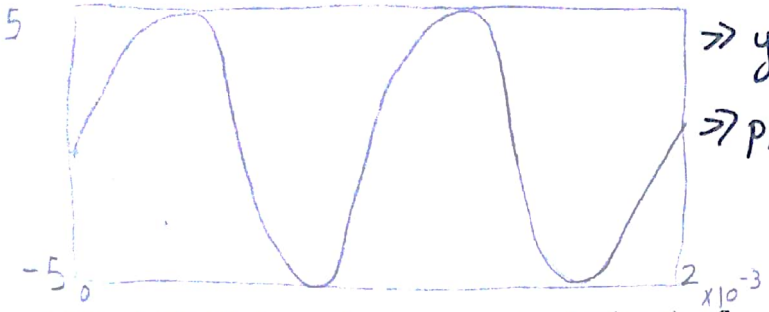
S=A./B

1 0 0
-2.67 -0.33 1.8
-6.67 -3.33 7

1 1 1
2 3 6
3 4 5

3- Plot the function x=5sinwt

f=1000HZ ; draw 2 cycles




```
>> t=0:0.000002:0.002;
>> y=5*sin(2*pi*1000*t);
>> plot(t,y);
```

4-Find GCD (greatest common divisor) of

a- 8 12 32 68

4

```
>> a=sym([8 12 32 68]);
>> lcm(a)
```

b- b-(5/2), (4/3), and (6/14)

```
>> gcd(a)
```

1/42

5-Find the LCM (least common multiple) of

a- 8 12 32 and 68

1632

b- b- (2/5), (3/4), and (14/6)

42

Department of Biomedical Technology

INTRODUCTION TO COMPUTER SYSTEMS
BMT 227 Lab Manual

Lab 3: Matrix operations and solving systems of linear algebraic equations

2018

Biomedical Technology
King Saud University
Riyadh, Kingdom of Saudi Arabia

B) Matrix Operations (addition, subtraction, scalar multiplication, multiplication, transpose)

1) Write the next matrices

$$\begin{matrix} 5 & 3 & 2 \\ 4 & 9 & 12 \\ 30 & 40 & 50 \end{matrix}$$

$$\begin{matrix} 1 & 2 & 3 \\ 4 & 6 & 8 \\ 10 & 20 & 30 \end{matrix}$$

2) Write the result for the following operations:

① Addition for the two matrices ~~a+b~~ a+b

② Subtraction for the two matrices a-b a-b

③ Multiplication for the two matrices ~~a*b~~ a*b

④ ***Array multiplication*** for the two matrices ~~a.*b~~ a.*b

①: $\begin{bmatrix} 6 & 5 & 5 \\ 8 & 15 & 20 \\ 40 & 60 & 80 \end{bmatrix}$

②: $\begin{bmatrix} 4 & 1 & -1 \\ 6 & 3 & 4 \\ 20 & 20 & 20 \end{bmatrix}$

③: $\begin{bmatrix} 37 & 68 & 99 \\ 160 & 302 & 444 \\ 690 & 1300 & 1910 \end{bmatrix}$

④: $\begin{bmatrix} 5 & 6 & 6 \\ 16 & 54 & 96 \\ 300 & 800 & 1500 \end{bmatrix}$

3) Write the next matrices

C=[5;3;7;1]

$$C = \begin{bmatrix} 5 \\ 3 \\ 7 \\ 1 \end{bmatrix} \qquad D = \begin{bmatrix} 6 & 2 & 3 & 4 \end{bmatrix}$$

Apply the Vector Operations, multiply two matrices

C*D

$$C*D = \begin{bmatrix} 30 & 10 & 15 & 20 \\ 18 & 6 & 9 & 12 \\ 42 & 14 & 21 & 28 \\ 6 & 2 & 3 & 4 \end{bmatrix}$$

4) Find the inverse of the following matrix:

$$b = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 6 & 8 \\ 12 & 14 & 16 \end{bmatrix} \qquad \gg (b).\backslash I$$

$$\begin{bmatrix} 1 & 0.5 & 0.3333 \\ 0.2500 & 0.1667 & 0.1250 \\ 0.0833 & 0.0714 & 0.0625 \end{bmatrix}$$

C)

1) Use the Matlab Program to find the solutions for the following linear algebraic equations:-

f

$$4x_1 - 2x_2 + x_3 = 7$$
$$x_1 + 1x_2 + 5x_3 = 10$$
$$-2x_1 + 3x_2 - x_3 = 2$$

P

النواتج \ الأعداد

f\P = [3.0244 2.9512 0.8049]

2) Use the Matlab Programmer to find the solutions for the following linear algebraic equations:-

t

$$x_1 + 2x_2 = 2$$
$$2x_1 + 2x_2 = 6$$

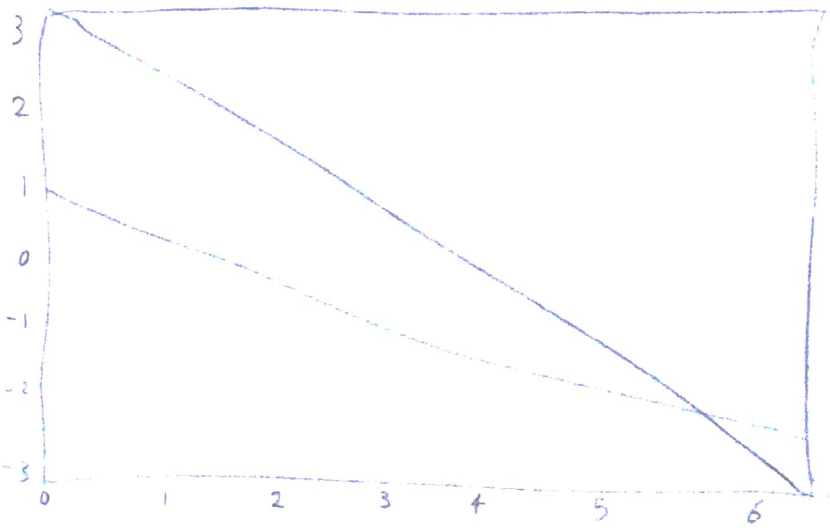
h

t\h = [4 -1]

back slash

Alt + \

3) Repeat the previous practice to find the solution by plot the two lines



A) **polynomial equations operations ; addition – subtraction – multiplication - division (long division) for :**

$a=x^3-xy^2+1$ and b= x+y

Apply the matlab commands for the pervious equations to find:

a+b , a-b, ab ,a/b

```
>> syms x y
>> a=[x^3-xy^2+1]
>> b=[x+y]
```

Results

```
>> a+b =
```

$x^3-xy^2+x+y+1$

```
>> a-b =
```

$x^3-xy^2-x-y+1$

```
>> a/b =
```

$(x^3-xy^2+1)/(x+y)$

```
>> a*b = (x+y) * (x^3-xy^2+1)
```

B) **Evaluate the following polynomial at several points:**

$$Z=5A^2+3B+50C$$

When;

A= [5,10,20]

B= [3,4,5]

C= [5,6,2]

```
>> a=[5 10 20];
>> b=[3 4 5];
>> c=[5 6 2];
>> syms z
>> z=[5.*a^2+3.*b+50.*c];
```

ans = 384 812 2115

C) 1- Evaluate the definite integral for the following integral: 49. A

$$I = \int_{-1}^{3} 3x^4 - 4^2 + 10x - 25\,dx$$

```
>> Z=[5 3 50];
>> a=[5 10 20];
>> polyval(z,a)

ans=
     190   590   2110

>> b=[3 4 5];
>> polyval(z,b)
 ans=  104   142   190
>> P=[3 0 -4 10 -25];
>> polyint(P,x)
ans = [ 3/5, 0, -4/3, 5, -25, x]
```

1- Find the differentiation for the following equation:

$$3x^4+4x^2-20$$

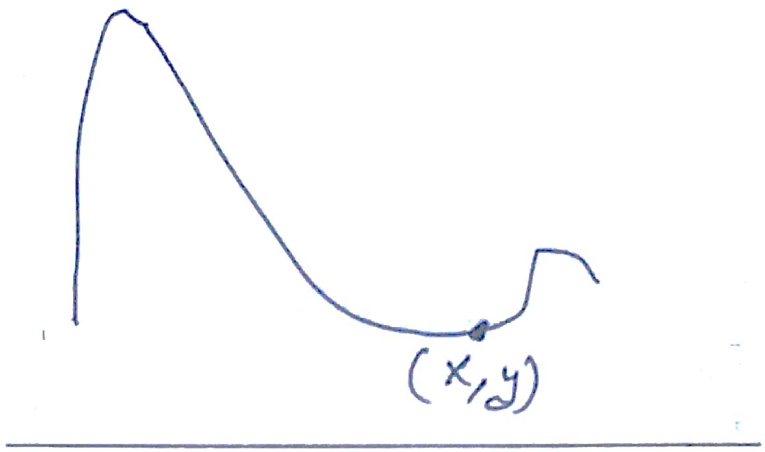
```
>> syms d
>> d=[3.*x.^4+4.*x.^2-20];
>> diff(d)
ans =  12*x^3+8*x
```

A) use fplot for the function defined by yourself.

- fplot('function1',[0 8])
- where function1 is a M-file function: fun 'function1'; fplot(fun,[0 8])
- function y function1(x)
- y 2*exp(-x).*sin(x);

min=(,)

max=(,)



- In many situations, we want to find the function extremes, the local maximum (peaks) and minimum (valleys).
- fmin(fun,x1,x2) can be used to find a local minimum of function f(x) defined by string fun in the interval x1 < x < x2.
- In order to find the local maximum, function f(x) should be replaced by -f(x).
- We have learned to use roots to find the zeros of a polynomial.
- fzero(fun,x0) can be used to find zero of a general function f(x) near x0.

- Apply the MatLab commands to find the maximum and minimum of the function y 2*exp(-x).*sin(x).

- Apply the MatLab commands to find the zeros of a polynomial y 2*exp(-x).*sin(x)

min(,)

max(,)

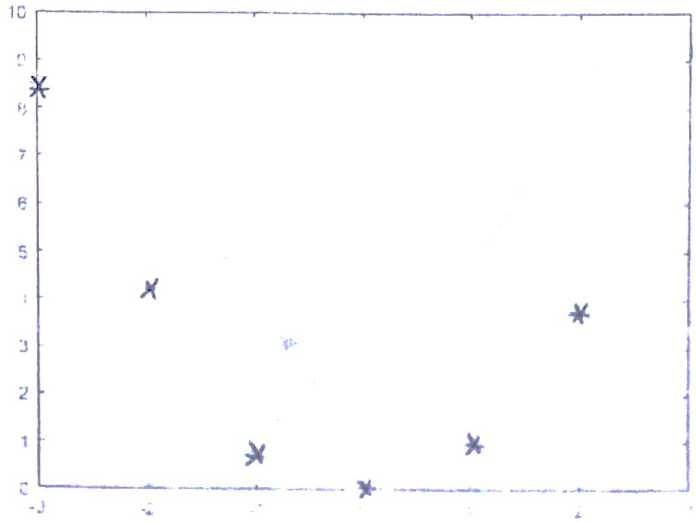
Zeros

B) **Using *polyfit* function**

x=[-3 -2 -1 0 1 2 3];
y=[8.5 4.3 .8 .1 1 3.8 9.2];

plot(x, y, '*')

Apply the MatLab commands to plot the data of x and y, then use the polyfit command to plot the same data.



C) Interpolation is defined as a way of estimating values of a function between those given by some set of data points
known temperature every hour for twelve hours. Find the temperature at 9.3 and 11.7 using *interp1*

Hours	Temp(C°)
1	5
2	8
3	9
4	15
5	25
6	29
7	31
8	30
9	22
10	25
11	27
12	24

```
x=0:8;
y=@(x) 2*exp(-x).*sin(x)
fplot('2*exp(-x).*sin(x)',[0 8])
grid on
b=fminbnd(y,0,8)
y2=y(b)
z='-(2*exp(-x).*sin(x))';
a=fminbnd(z,0,8)
y1=y(a)
xzero =fzero(y,8)
yzero=y(xzero)
x=[-3 -2 -1 0 1 2 3];
y=[8.5 4.3 .8 .1 1 3.8 9.2];
plot(x,y,'*'), grid on
p=polyfit(x,y,2)
x1=-3:.1:3;
y1=polyval(p,x1)
plot(x,y,'*',x1,y1),grid on
hours=1:12;
temps=[5 9 8 15 22 29 31 30 22 25 27 24];
y2=interp1(hours,temps,[9.3 11.7])
```

Mathematical Operations with Arrays

- Addition and Subtraction
  The operations + (addition) and - (subtraction) can be used to add (subtract) arrays of identical size (the same numbers of rows and columns) and to add (subtract) a scalar to an array.

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \end{bmatrix} \text{ and } B = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \end{bmatrix}$$

By adding A+B the results is:

$$\begin{bmatrix} (A_{11}+B_{11}) & (A_{12}+B_{12}) & (A_{13}+B_{13}) \\ (A_{21}+B_{21}) & (A_{22}+B_{22}) & (A_{23}+B_{23}) \end{bmatrix}$$

- Array Multiplication (Linear algebra rules)
  The multiplication operation ***** is executed by MATLAB according to the rules of linear algebra. This means that if **A** and **B** are two matrices, the operation **A* B** can be carried out only if the number of **columns in matrix A is equal to the number of rows in matrix B**. The result is a matrix that has the **same number of rows as A and the same number of columns as B.**

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \\ A_{41} & A_{42} & A_{43} \end{bmatrix} \text{ and } B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \\ B_{31} & B_{32} \end{bmatrix}$$

A*B=

$$\begin{bmatrix} (A_{11}B_{11}+A_{12}B_{21}+A_{13}B_{31}) & (A_{11}B_{12}+A_{12}B_{22}+A_{13}B_{32}) \\ (A_{21}B_{11}+A_{22}B_{21}+A_{23}B_{31}) & (A_{21}B_{12}+A_{22}B_{22}+A_{23}B_{32}) \\ (A_{31}B_{11}+A_{32}B_{21}+A_{33}B_{31}) & (A_{31}B_{12}+A_{32}B_{22}+A_{33}B_{32}) \\ (A_{41}B_{11}+A_{42}B_{21}+A_{43}B_{31}) & (A_{41}B_{12}+A_{42}B_{22}+A_{43}B_{32}) \end{bmatrix}$$

  - The multiplication of matrices is not commutative. This means that if A and B are both n x n, then A* B ≠ B *A
  - The power operation can be executed only with a square matrix.
- Array Multiplication (Element-by-element)
  - Element-by-element operations can be done only with arrays of the same size.

Element-by-element multiplication, division, or exponentiation of two vectors or matrices is entered in MATLAB by typing a period in front of the arithmetic operator.

Symbol	Description	Symbol	Description
.*	Multiplication	./	Right division
.^	Exponentiation	.\	Left Division

If two vectors a and b are $a = [a_1\ a_2\ a_3\ a_4]$ and $b = [b_1\ b_2\ b_3\ b_4]$, then element-by-element multiplication, division, and exponentiation of the two vectors gives:

$$a .* b = [a_1b_1\ a_2b_2\ a_3b_3\ a_4b_4]$$
$$a ./ b = [a_1/b_1\ a_2/b_2\ a_3/b_3\ a_4/b_4]$$
$$a .^ b = [(a_1)^{b_1}\ (a_2)^{b_2}\ (a_3)^{b_3}\ (a_4)^{b_4}]$$

All materials are taken from: *Gilat, Amos. MATLAB: an Introduction with Applications. John Wiley & Sons, 2015.*

If two matrices A and B are

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ B_{31} & B_{32} & B_{33} \end{bmatrix}$$

then element-by-element multiplication and division of the two matrices give:

$$A .* B = \begin{bmatrix} A_{11}B_{11} & A_{12}B_{12} & A_{13}B_{13} \\ A_{21}B_{21} & A_{22}B_{22} & A_{23}B_{23} \\ A_{31}B_{31} & A_{32}B_{32} & A_{33}B_{33} \end{bmatrix} \qquad A ./ B = \begin{bmatrix} A_{11}/B_{11} & A_{12}/B_{12} & A_{13}/B_{13} \\ A_{21}/B_{21} & A_{22}/B_{22} & A_{23}/B_{23} \\ A_{31}/B_{31} & A_{32}/B_{32} & A_{33}/B_{33} \end{bmatrix}$$

Element-by-element exponentiation of matrix A gives:

$$\text{A .\^{} } n = \begin{bmatrix} (A_{11})^n & (A_{12})^n & (A_{13})^n \\ (A_{21})^n & (A_{22})^n & (A_{23})^n \\ (A_{31})^n & (A_{32})^n & (A_{33})^n \end{bmatrix}$$

- Array Division
  - **Identity matrix:** The identity matrix is a square matrix in which the diagonal elements are 1s and the rest of the elements are 0s. An identity matrix can be created in MATLAB with the *eye* command.

$$\begin{bmatrix} 7 & 3 & 8 \\ 4 & 11 & 5 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 7 & 3 & 8 \\ 4 & 11 & 5 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 8 \\ 2 \\ 15 \end{bmatrix} = \begin{bmatrix} 8 \\ 2 \\ 15 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} 6 & 2 & 9 \\ 1 & 8 & 3 \\ 7 & 4 & 5 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 6 & 2 & 9 \\ 1 & 8 & 3 \\ 7 & 4 & 5 \end{bmatrix}$$

If a matrix A is square, it can be multiplied by the identity matrix, I, from the left or from the right:

$$AI = IA = A$$

  - **Inverse of a matrix:**

Inverse of a matrix:

The matrix B is the inverse of the matrix A if, when the two matrices are multiplied, the product is the identity matrix. Both matrices must be square, and the multiplication order can be BA or AB.

$$BA = AB = I$$

Obviously B is the inverse of A, and A is the inverse of B. For example:

$$\begin{bmatrix} 2 & 1 & 4 \\ 4 & 1 & 8 \\ 2 & -1 & 3 \end{bmatrix} \begin{bmatrix} 5.5 & -3.5 & 2 \\ 2 & -1 & 0 \\ -3 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 5.5 & -3.5 & 2 \\ 2 & -1 & 0 \\ -3 & 2 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 4 \\ 4 & 1 & 8 \\ 2 & -1 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The inverse of a matrix A is typically written as A^{-1}. In MATLAB the inverse of a matrix can be obtained either by raising A to the power of –1, A^{-1}, or with the `inv(A)` function. Multiplying the matrices above with MATLAB is shown below.

- ## Left division, \:

Left division is used to solve the matrix equation $AX = B$. In this equation X and B are column vectors. This equation can be solved by multiplying, on the left, both sides by the inverse of A:

$$A^{-1}AX = A^{-1}B$$

The left-hand side of this equation is X, since

$$A^{-1}AX = IX = X$$

So the solution of $AX = B$ is:

$$X = A^{-1}B$$

In MATLAB the last equation can be written by using the left division character:

$$X = A\backslash B$$

It should be pointed out here that although the last two operations appear to give the same result, the method by which MATLAB calculates X is different. In the first, MATLAB calculates A^{-1} and then uses it to multiply B. In the second (left division), the solution X is obtained *numerically* using a method that is based on Gauss elimination. The left division method is recommended for solving a set of linear equations, because the calculation of the inverse may be less accurate than the Gauss elimination method when large matrices are involved.

- ## Right division, /:

Right division, / :

The right division is used to solve the matrix equation $XC = D$. In this equation X and D are row vectors. This equation can be solved by multiplying, on the right, both sides by the inverse of C:

$$X \cdot CC^{-1} = D \cdot C^{-1}$$

which gives

$$X = D \cdot C^{-1}$$

In MATLAB the last equation can be written using the right division character:

$$X = D/C$$

Sample Problem 3-1: Solving three linear equations (array division)

Use matrix operations to solve the following system of linear equations.

$$4x - 2y + 6z = 8$$
$$2x + 8y + 2z = 4$$
$$6x + 10y + 3z = 0$$

Solution

Using the rules of linear algebra demonstrated earlier, the above system of equations can be written in the matrix form $AX = B$ or in the form $XC = D$:

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

Table 3-1: Built-in array functions

Function	Description	Example

`mean(A)`	If A is a vector, returns the mean value of the elements of the vector.	`>> A=[5 9 2 4];`
		`>> mean(A)`
		`ans =`
		`5`

`C=max(A)`
 	
`[d,n]=max(A)`
	If A is a vector, C is the largest element in A. If A is a matrix, C is a row vector containing the largest element of each column of A.
	
	If A is a vector, d is the largest element in A, and n is the position of the element (the first if several have the max value).
		`>> A=[5 9 2 4 11 6 11 1];`
		`>> C=max(A)`
		`C =`
		`11`
		
		`>> [d,n]=max(A)`
		`d =`
		`11`
		`n =`
		`5`

`min(A)`
 	
`[d,n]=min(A)`
	The same as `max(A)`, but for the smallest element.
	
	The same as `[d,n]=max(A)`, but for the smallest element.
		`>> A=[5 9 2 4];`
		`>> min(A)`
		`ans =`
		`2`

`sum(A)`	If A is a vector, returns the sum of the elements of the vector.	`>> A=[5 9 2 4];`
		`>> sum(A)`
		`ans =`
		`20`

`sort(A)`	If A is a vector, arranges the elements of the vector in ascending order.	`>> A=[5 9 2 4];`
		`>> sort(A)`
		`ans =`
		`2 4 5 9`

`median(A)`	If A is a vector, returns the median value of the elements of the vector.	`>> A=[5 9 2 4];`
		`>> median(A)`
		`ans =`
		`4.5000`

Table 3-1: Built-in array functions (Continued)

Function	Description	Example

`std(A)`	If A is a vector, returns the standard deviation of the elements of the vector.	`>> A=[5 9 2 4];`
		`>> std(A)`
		`ans =`
		`2.9439`

`det(A)`	Returns the determinant of a square matrix A.	`>> A=[2 4; 3 5];`
		`>> det(A)`
		`ans =`
		`-2`

`dot(a,b)`	Calculates the scalar (dot) product of two vectors a and b. The vectors can each be row or column vectors.	`>> a=[1 2 3];`
		`>> b=[3 4 5];`
		`>> dot(a,b)`
		`ans =`
		`26`

`cross(a,b)`	Calculates the cross product of two vectors a and b, (axb). The two vectors must have each three elements.	`>> a=[1 3 2];`
		`>> b=[2 4 1];`
		`>> cross(a,b)`
		`ans =`
		`-5 3 -2`

`inv(A)`	Returns the inverse of a square matrix A.	`>> A=[2 -2 1; 3 2 -1; 2 -3 2];`
		`>> inv(A)`
		`ans =`
		`0.2000 0.2000 0`
		`-1.6000 0.4000 1.0000`
		`-2.6000 0.4000 2.0000`

- **You are not required to memorize all the function in the above tables; however, you should familiarize yourself with them as they appear in practices.**

Example in class: Find the mean, inverse, dot product, sum, max, and min.

Solving System of Linear Equations

Linear algebra rules of array multiplication provide a convenient way for writing a system of linear equations. For example, the system of three equations with three unknowns

$$\begin{aligned} A_{11}x_1 + A_{12}x_2 + A_{13}x_3 &= B_1 \\ A_{21}x_1 + A_{22}x_2 + A_{23}x_3 &= B_2 \\ A_{31}x_1 + A_{32}x_2 + A_{33}x_3 &= B_3 \end{aligned}$$

can be written in a matrix form as

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \\ B_3 \end{bmatrix}$$

and in matrix notation as

$$AX = B \quad \text{where } A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}, X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \text{ and } B = \begin{bmatrix} B_1 \\ B_2 \\ B_3 \end{bmatrix}.$$

Examples in class:

- Solve the following system of five linear equations:

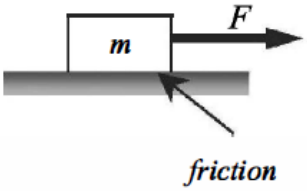
$$\begin{aligned} 2.5a - b + 3c + 1.5d - 2e &= 57.1 \\ 3a + 4b - 2c + 2.5d - e &= 27.6 \\ -4a + 3b + c - 6d + 2e &= -81.2 \\ 2a + 3b + c - 2.5d + 4e &= -22.2 \\ a + 2b + 5c - 3d + 4e &= -12.2 \end{aligned}$$

Hint: use left division

Sample Problem 3-3: Friction experiment (element-by-element calculations)

The coefficient of friction, μ, can be determined in an experiment by measuring the force F required to move a mass m. When F is measured and m is known, the coefficient of friction can be calculated by:

$$\mu = F/(mg) \quad (g = 9.81 \text{ m/s}^2).$$



Results from measuring F in six tests are given in the table below. Determine the coefficient of friction in each test, and the average from all tests.

Test	1	2	3	4	5	6
Mass m (kg)	2	4	5	10	20	50
Force F (N)	12.5	23.5	30	61	117	294

All materials are taken from: *Gilat, Amos. MATLAB: an Introduction with Applications. John Wiley & Sons, 2015.*

Tasks to be submitted for lab 2 grading:

Task 1:

. Create the following three matrices:

$$A = \begin{bmatrix} 1 & -3 & 5 \\ 2 & 2 & 4 \\ -2 & 0 & 6 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & -2 & 1 \\ 5 & 1 & -6 \\ 2 & 7 & -1 \end{bmatrix} \qquad C = \begin{bmatrix} -3 & 4 & -1 \\ 0 & 8 & 2 \\ -3 & 5 & 3 \end{bmatrix}$$

(*a*) Calculate $A + B$ and $B + A$ to show that addition of matrices is commutative.

(*b*) Calculate $A + (B + C)$ and $(A + B) + C$ to show that addition of matrices is associative.

(*c*) Calculate $3(A + C)$ and $3A + 5C$ to show that, when matrices are multiplied by a scalar, the multiplication is distributive.

(*d*) Calculate $A*(B + C)$ and $A*B + A*C$ to show that matrix multiplication is distributive.

Task 2:

. Use the matrices *A*, *B*, and *C* from the previous problem to answer the following:

(*a*) Does $A*B = B*A$?

(*b*) Does $A*(B*C) = (A*B)*C$?

(*c*) Does $(A*B)^t = A^t*B^t$? (^{t} means transpose)

(*d*) Does $(A + B)^t = A^t + B^t$?

Task 3:

- Solve the following system of linear equations

$$\begin{aligned} -4x + 3y + z &= -18.2 \\ 5x + 6y - 2z &= -48.8 \\ 2x - 5y + 4.5z &= 92.5 \end{aligned}$$

Task 4:

For the function $y = x^2 - e^{0.5x} + x$, calculate the value of y for the following values of x using element-by-element operations: $-3, -2, -1, 0, 1, 2, 3$.

Plot Y.Vs. X, then title and label your graph

All materials are taken from: *Gilat, Amos. MATLAB: an Introduction with Applications. John Wiley & Sons, 2015.*