

## **Experiment No.-1**

**Object:-** Solution of linear equations for under determined and over determined cases.

**Software Used:-** MATLAB 7.0

**Theory:-**

**Systems of Linear Equations:-**

**Computational Considerations**

One of the most important problems in technical computing is the solution of systems of simultaneous linear equations.

In matrix notation, the general problem takes the following form: Given two matrices  $A$  and  $B$ , does there exist a unique matrix  $X$  so that  $AX = B$  or  $XA = B$ ?

It is instructive to consider a 1-by-1 example. For example, does the equation

$$7x = 21$$

have a unique solution?

The answer, of course, is yes. The equation has the unique solution  $x = 3$ . The solution is easily obtained by division:

$$x = 21/7 = 3$$

The solution is *not* ordinarily obtained by computing the inverse of 7, that is  $7^{-1} = 0.142857\dots$ , and then multiplying  $7^{-1}$  by 21. This would be more work and, if  $7^{-1}$  is represented to a finite number of digits, less accurate. Similar considerations apply to sets of linear equations with more than one unknown;

the MATLAB software solves such equations without computing the inverse of the matrix.

Although it is not standard mathematical notation, MATLAB uses the division terminology familiar in the scalar case to describe the solution of a general system of simultaneous equations. The two division symbols, *slash*, /, and *backslash*, \, correspond to the two MATLAB functions `mldivide` and `mrdivide`. `mldivide` and `mrdivide` are used for the two situations where the unknown matrix appears on the left or right of the coefficient matrix:

$X = B/A$  Denotes the solution to the matrix equation  $XA = B$ .

$X = A \setminus B$  Denotes the solution to the matrix equation  $AX = B$ .

Think of "dividing" both sides of the equation  $AX = B$  or  $XA = B$  by  $A$ . The coefficient matrix  $A$  is always in the "denominator."

The dimension compatibility conditions for  $X = A \setminus B$  require the two matrices  $A$  and  $B$  to have the same number of rows. The solution  $X$  then has the same number of columns as  $B$  and its row dimension is equal to the column dimension of  $A$ . For  $X = B/A$ , the roles of rows and columns are interchanged.

In practice, linear equations of the form  $AX = B$  occur more frequently than those of the form  $XA = B$ . Consequently, the backslash is used far more frequently than the slash. The remainder of this section concentrates on the backslash operator; the corresponding properties of the slash operator can be inferred from the identity:

$$(B/A)' = (A' \setminus B')$$

The coefficient matrix  $A$  need not be square. If  $A$  is  $m$ -by- $n$ , there are three cases:

$m = n$  Square system. Seek an exact solution.

$m > n$  Over determined system. Find a least squares solution.

$m < n$  Underdetermined system. Find a basic solution with at most  $m$  nonzero components.

## **The mldivide Algorithm**

The mldivide operator employs different algorithms to handle different kinds of coefficient matrices. The various cases are diagnosed automatically by examining the coefficient matrix.

### **Permutations of Triangular Matrices**

mldivide checks for triangularity by testing for zero elements. If a matrix  $A$  is triangular, MATLAB software uses a substitution to compute the solution vector  $x$ . If  $A$  is a permutation of a triangular matrix, MATLAB software uses a permuted substitution algorithm.

### **Square Matrices**

If  $A$  is symmetric and has real, positive diagonal elements, MATLAB attempts a Cholesky factorization. If the Cholesky factorization fails, MATLAB performs a symmetric, indefinite factorization. If  $A$  is upper Hessenberg, MATLAB uses Gaussian elimination to reduce the system to a triangular matrix. If  $A$  is square but is neither permuted triangular, symmetric and positive definite, or Hessenberg, MATLAB performs a general triangular factorization using LU factorization with partial pivoting (see lu).

### **Rectangular Matrices**

If  $A$  is rectangular, mldivide returns a least-squares solution. MATLAB solves overdetermined systems with QR factorization (see qr). For an

underdetermined system, MATLAB returns the solution with the maximum number of zero elements.

The `mldivide` function reference page contains a more detailed description of the algorithm.

### General Solution

The general solution to a system of linear equations  $AX = b$  describes all possible solutions. You can find the general solution by:

1. Solving the corresponding homogeneous system  $AX = 0$ . Do this using the `null` command, by typing `null(A)`. This returns a basis for the solution space to  $AX = 0$ . Any solution is a linear combination of basis vectors.
2. Finding a particular solution to the nonhomogeneous system  $AX = b$ .

You can then write any solution to  $AX = b$  as the sum of the particular solution to  $AX = b$ , from step 2, plus a linear combination of the basis vectors from step 1.

The rest of this section describes how to use MATLAB to find a particular solution to  $AX = b$ , as in step 2.

### Square Systems

The most common situation involves a square coefficient matrix  $A$  and a single right-hand side column vector  $b$ .

#### Nonsingular Coefficient Matrix

If the matrix  $A$  is nonsingular, the solution,  $x = A \setminus b$ , is then the same size as  $b$ . For example:

```
A = pascal(3);
```

$$u = [3; 1; 4];$$

$$x = A \setminus u$$

$$x =$$

$$10$$

$$-12$$

$$5$$

It can be confirmed that  $A * x$  is exactly equal to  $u$ .

If  $A$  and  $B$  are square and the same size,  $X = A \setminus B$  is also that size:

$$B = \text{magic}(3);$$

$$X = A \setminus B$$

$$X =$$

$$19 \quad -3 \quad -1$$

$$-17 \quad 4 \quad 13$$

$$6 \quad 0 \quad -6$$

It can be confirmed that  $A * X$  is exactly equal to  $B$ .

Both of these examples have exact, integer solutions. This is because the coefficient matrix was chosen to be `pascal(3)`, which has a determinant equal to 1.

### Singular Coefficient Matrix

A square matrix  $A$  is singular if it does not have linearly independent columns. If  $A$  is singular, the solution to  $AX = B$  either does not exist, or is not unique. The backslash operator,  $A \setminus B$ , issues a warning if  $A$  is nearly singular and raises an error condition if it detects exact singularity.

If  $A$  is singular and  $AX = b$  has a solution, you can find a particular solution that is not unique, by typing

$$P = \text{pinv}(A) * b$$

**P is a pseudoinverse of A. If  $AX = b$  does not have an exact solution, `pinv(A)` returns a least-squares solution.**

**For example:**

$$A = \begin{bmatrix} 1 & 3 & 7 \\ -1 & 4 & 4 \\ 1 & 10 & 18 \end{bmatrix}$$

**is singular, as you can verify by typing**

**`det(A)`**

**ans =**

**0**

**Exact Solutions.** For  $b = [5;2;12]$ , the equation  $AX = b$  has an exact solution, given by

**`pinv(A)*b`**

**ans =**

**0.3850**

**-0.1103**

**0.7066**

**Verify that `pinv(A)*b` is an exact solution by typing**

**`A*pinv(A)*b`**

**ans =**

**5.0000**

**2.0000**

**12.0000**

**Least-Squares Solutions.** On the other hand, if  $b = [3;6;0]$ ,  $AX = b$  does not have an exact solution. In this case, `pinv(A)*b` returns a least squares solution. If you type

**`A*pinv(A)*b`**

**ans =**

**-1.0000**

**4.0000**

**2.0000**

**you do not get back the original vector b.**

You can determine whether  $AX = b$  has an exact solution by finding the row reduced echelon form of the augmented matrix  $[A \ b]$ . To do so for this example, enter

```
rref([A b])
```

```
ans =
```

```
1.0000    0  2.2857    0
      0  1.0000  1.5714    0
      0    0    0  1.0000
```

Since the bottom row contains all zeros except for the last entry, the equation does not have a solution. In this case,  $\text{pinv}(A)$  returns a least-squares solution.

**Result:** - Verified by executing the m-file

## Experiment No.-2

**Object:-** Matrices in the MATLAB Environment .

**Software Used:-** MATLAB 7.0

The MATLAB environment uses the term *matrix* to indicate a variable containing real or complex numbers arranged in a two-dimensional grid. An *array* is, more generally, a vector, matrix, or higher-dimensional grid of numbers. All arrays in MATLAB are rectangular, in the sense that the component vectors along any dimension are all the same length.

MATLAB has dozens of functions that create different kinds of matrices. There are two functions you can use to create a pair of 3-by-3 example matrices for use throughout this chapter. The first example is symmetric:

```
A = pascal(3)
```

```
A =
```

```
1 1 1
1 2 3
1 3 6
```

The second example is not symmetric:

```
B = magic(3)
```

```
B =
```

```
8 1 6
3 5 7
4 9 2
```

Another example is a 3-by-2 rectangular matrix of random integers:

```
C = fix(10*rand(3,2))
```

```
C =
```

```
9 4
2 8
```

6 7

A column vector is an  $m$ -by-1 matrix, a row vector is a 1-by- $n$  matrix, and a scalar is a 1-by-1 matrix. The statements

$$\mathbf{u} = [3; 1; 4]$$

$$\mathbf{v} = [2 \ 0 \ -1]$$

$$s = 7$$

produce a column vector, a row vector, and a scalar:

$$\mathbf{u} =$$

3

1

4

$$\mathbf{v} =$$

2 0 -1

$$s =$$

7

For more information about creating and working with matrices, see in **MATLAB Programming Fundamentals**.

### Adding and Subtracting Matrices

Addition and subtraction of matrices is defined just as it is for arrays, element by element. Adding A to B and then subtracting A from the result recovers B:

$$\mathbf{A} = \text{pascal}(3);$$

$$\mathbf{B} = \text{magic}(3);$$

$$\mathbf{X} = \mathbf{A} + \mathbf{B}$$

$$\mathbf{X} =$$

9 2 7

4 7 10

5 12 8

$$\mathbf{Y} = \mathbf{X} - \mathbf{A}$$

$$\mathbf{Y} =$$

8 1 6

3 5 7

4 9 2

Addition and subtraction require both matrices to have the same dimension, or one of them be a scalar. If the dimensions are incompatible, an error results:

```
C = fix(10*rand(3,2))
```

```
X = A + C
```

(Error using ==> +Matrix dimensions must agree.)

```
w = v + s
```

```
w =
```

9 7 6

### Vector Products and Transpose

A row vector and a column vector of the same length can be multiplied in either order. The result is either a scalar, the *inner* product, or a matrix, the *outer product* :

```
u = [3; 1; 4];
```

```
v = [2 0 -1];
```

```
x = v*u
```

```
x =
```

2

```
X = u*v
```

```
X =
```

6 0 -3

2 0 -1

8 0 -4

For real matrices, the *transpose* operation interchanges  $a_{ij}$  and  $a_{ji}$ . MATLAB uses the apostrophe operator (') to perform a complex conjugate transpose, and uses the dot-apostrophe operator (.' ) to transpose without conjugation. For matrices containing all real elements, the two operators return the same result.

The example matrix A is *symmetric*, so A' is equal to A. But B is not symmetric:

```
B = magic(3);
```

```
X = B'
```

**X =**

$$\begin{bmatrix} 8 & 3 & 4 \\ 1 & 5 & 9 \\ 6 & 7 & 2 \end{bmatrix}$$

**Transposition turns a row vector into a column vector:**

$$\mathbf{x} = \mathbf{v}'$$

$$\mathbf{x} =$$

$$2$$

$$0$$

$$-1$$

**If x and y are both real column vectors, the product  $\mathbf{x}*\mathbf{y}$  is not defined, but the two products**

$$\mathbf{x}'*\mathbf{y} \quad \text{and} \quad \mathbf{y}'*\mathbf{x}$$

**are the same scalar. This quantity is used so frequently, it has three different names: *inner product*, *scalar product*, or *dot product*.**

**For a complex vector or matrix, z, the quantity  $\mathbf{z}'$  not only transposes the vector or matrix, but also converts each complex element to its complex conjugate. That is, the sign of the imaginary part of each complex element changes. So if**

$$\mathbf{z} = [1+2i \ 7-3i \ 3+4i; \ 6-2i \ 9i \ 4+7i]$$

$$\mathbf{z} =$$

$$1.0000 + 2.0000i \quad 7.0000 - 3.0000i \quad 3.0000 + 4.0000i$$

$$6.0000 - 2.0000i \quad 0 + 9.0000i \quad 4.0000 + 7.0000i$$

**then**

$$\mathbf{z}'$$

$$\text{ans} =$$

$$1.0000 - 2.0000i \quad 6.0000 + 2.0000i$$

$$7.0000 + 3.0000i \quad 0 - 9.0000i$$

$$3.0000 - 4.0000i \quad 4.0000 - 7.0000i$$

**The unconjugated complex transpose, where the complex part of each element retains its sign, is denoted by  $\mathbf{z}.'$ :**

```

z.'
ans =
    1.0000 + 2.0000i    6.0000 - 2.0000i
    7.0000 - 3.0000i         0 + 9.0000i
    3.0000 + 4.0000i    4.0000 + 7.0000i

```

For complex vectors, the two scalar products  $x^*y$  and  $y^*x$  are complex conjugates of each other, and the scalar product  $x^*x$  of a complex vector with itself is real.

### Multiplying Matrices

Multiplication of matrices is defined in a way that reflects composition of the underlying linear transformations and allows compact representation of systems of simultaneous linear equations. The matrix product  $C = AB$  is defined when the column dimension of  $A$  is equal to the row dimension of  $B$ , or when one of them is a scalar. If  $A$  is  $m$ -by- $p$  and  $B$  is  $p$ -by- $n$ , their product  $C$  is  $m$ -by- $n$ . The product can actually be defined using MATLAB **for** loops, **colon** notation, and vector dot products:

```

A = pascal(3);
B = magic(3);
m = 3; n = 3;
for i = 1:m
    for j = 1:n
        C(i,j) = A(i,:)*B(:,j);
    end
end
end

```

MATLAB uses a single asterisk to denote matrix multiplication. The next two examples illustrate the fact that matrix multiplication is not commutative;  $AB$  is usually not equal to  $BA$ :

```

X = A*B
X =
    15    15    15
    26    38    26
    41    70    39

```

```

Y = B*A

```

**Y =**

**15 28 47**  
**15 34 60**  
**15 28 43**

**A matrix can be multiplied on the right by a column vector and on the left by a row vector:**

$$\mathbf{u} = [3; 1; 4];$$

$$\mathbf{x} = \mathbf{A} * \mathbf{u}$$

**x =**

**8**  
**17**  
**30**

$$\mathbf{v} = [2 \ 0 \ -1];$$

$$\mathbf{y} = \mathbf{v} * \mathbf{B}$$

**y =**

**12 -7 10**

**Rectangular matrix multiplications must satisfy the dimension compatibility conditions:**

$$\mathbf{C} = \text{fix}(10 * \text{rand}(3,2));$$

$$\mathbf{X} = \mathbf{A} * \mathbf{C}$$

**X =**

**17 19**  
**31 41**  
**51 70**

$$\mathbf{Y} = \mathbf{C} * \mathbf{A}$$

**(Error using ==> \* Inner matrix dimensions must agree.)**

**Anything can be multiplied by a scalar:**

$$\mathbf{s} = 7;$$

$$\mathbf{w} = \mathbf{s} * \mathbf{v}$$

**w =**

14 0 -7

## Eigenvalue Decomposition:-

An *eigenvalue* and *eigenvector* of a square matrix  $A$  are, respectively, a scalar  $\lambda$  and a nonzero vector  $v$  that satisfy

$$Av = \lambda v$$

With the eigenvalues on the diagonal of a diagonal matrix  $\Lambda$  and the corresponding eigenvectors forming the columns of a matrix  $V$ , you have

$$AV = V\Lambda$$

If  $V$  is nonsingular, this becomes the eigenvalue decomposition

$$A = V\Lambda V^{-1}$$

A good example is provided by the coefficient matrix of the [ordinary differential equation](#) in the previous section:

$$A = \begin{bmatrix} 0 & -6 & -1 \\ 6 & 2 & -16 \\ -5 & 20 & -10 \end{bmatrix}$$

The statement

$$\text{lambda} = \text{eig}(A)$$

produces a column vector containing the eigenvalues. For this matrix, the eigenvalues are complex:

$$\text{lambda} = \begin{bmatrix} -3.0710 \\ -2.4645+17.6008i \\ -2.4645-17.6008i \end{bmatrix}$$

The real part of each of the eigenvalues is negative, so  $e^{\lambda t}$  approaches zero as  $t$  increases. The nonzero imaginary part of two of the eigenvalues,  $\pm\omega$ , contributes the oscillatory component,  $\sin(\omega t)$ , to the solution of the differential equation.

With two output arguments, `eig` computes the eigenvectors and stores the eigenvalues in a diagonal matrix:

$$[V,D] = \text{eig}(A)$$

$$\begin{aligned}
 \mathbf{V} = & \\
 & -0.8326 \quad 0.2003 - 0.1394i \quad 0.2003 + 0.1394i \\
 & -0.3553 \quad -0.2110 - 0.6447i \quad -0.2110 + 0.6447i \\
 & -0.4248 \quad -0.6930 \quad -0.6930
 \end{aligned}$$

$$\begin{aligned}
 \mathbf{D} = & \\
 & -3.0710 \quad 0 \quad 0 \\
 & 0 \quad -2.4645+17.6008i \quad 0 \\
 & 0 \quad 0 \quad -2.4645-17.6008i
 \end{aligned}$$

The first eigenvector is real and the other two vectors are complex conjugates of each other. All three vectors are normalized to have Euclidean length,  $\text{norm}(v,2)$ , equal to one.

The matrix  $\mathbf{V} \cdot \mathbf{D} \cdot \text{inv}(\mathbf{V})$ , which can be written more succinctly as  $\mathbf{V} \cdot \mathbf{D} / \mathbf{V}$ , is within roundoff error of  $\mathbf{A}$ . And,  $\text{inv}(\mathbf{V}) \cdot \mathbf{A} \cdot \mathbf{V}$ , or  $\mathbf{V} \setminus \mathbf{A} \cdot \mathbf{V}$ , is within roundoff error of  $\mathbf{D}$ .

### Multiple Eigenvalues

Some matrices do not have an eigenvector decomposition. These matrices are not diagonalizable. For example:

$$\mathbf{A} = \begin{bmatrix} 6 & 12 & 19 \\ -9 & -20 & -33 \\ 4 & 9 & 15 \end{bmatrix}$$

For this matrix

$$[\mathbf{V}, \mathbf{D}] = \text{eig}(\mathbf{A})$$

produces

$$\begin{aligned}
 \mathbf{V} = & \\
 & -0.4741 \quad -0.4082 \quad -0.4082 \\
 & 0.8127 \quad 0.8165 \quad 0.8165 \\
 & -0.3386 \quad -0.4082 \quad -0.4082
 \end{aligned}$$

$$\begin{aligned}
 \mathbf{D} = & \\
 & -1.0000 \quad 0 \quad 0
 \end{aligned}$$

$$\begin{bmatrix} 0 & 1.0000 & 0 \\ 0 & 0 & 1.0000 \end{bmatrix}$$

**There is a double eigenvalue at  $\lambda = 1$ . The second and third columns of  $V$  are the same. For this matrix, a full set of linearly independent eigenvectors does not exist.**

**Result:** - Verified by executing the m-file

## Experiment No.-3

**Object:** - Determination of Eigen values and eigenvectors of a square matrix.

**Software Used:-** MATLAB 7.0

### **Eigenvalue Decomposition:-**

An *eigenvalue* and *eigenvector* of a square matrix  $A$  are, respectively, a scalar  $\lambda$  and a nonzero vector  $v$  that satisfy

$$Av = \lambda v$$

With the eigenvalues on the diagonal of a diagonal matrix  $\Lambda$  and the corresponding eigenvectors forming the columns of a matrix  $V$ , you have

$$AV = V\Lambda$$

If  $V$  is nonsingular, this becomes the eigenvalue decomposition

$$A = V\Lambda V^{-1}$$

A good example is provided by the coefficient matrix of the **ordinary differential equation** in the previous section:

$$A = \begin{bmatrix} 0 & -6 & -1 \\ 6 & 2 & -16 \\ -5 & 20 & -10 \end{bmatrix}$$

The statement

$$\text{lambda} = \text{eig}(A)$$

produces a column vector containing the eigenvalues. For this matrix, the eigenvalues are complex:

$$\text{lambda} = \begin{bmatrix} -3.0710 \\ -2.4645+17.6008i \\ -2.4645-17.6008i \end{bmatrix}$$

The real part of each of the eigenvalues is negative, so  $e^{\lambda t}$  approaches zero as  $t$  increases. The nonzero imaginary part of two of the eigenvalues,  $\pm\omega$ ,

contributes the oscillatory component,  $\sin(\omega t)$ , to the solution of the differential equation.

With two output arguments, `eig` computes the eigenvectors and stores the eigenvalues in a diagonal matrix:

$$[V,D] = \text{eig}(A)$$

V =

$$\begin{bmatrix} -0.8326 & 0.2003 - 0.1394i & 0.2003 + 0.1394i \\ -0.3553 & -0.2110 - 0.6447i & -0.2110 + 0.6447i \\ -0.4248 & -0.6930 & -0.6930 \end{bmatrix}$$

D =

$$\begin{bmatrix} -3.0710 & 0 & 0 \\ 0 & -2.4645+17.6008i & 0 \\ 0 & 0 & -2.4645-17.6008i \end{bmatrix}$$

The first eigenvector is real and the other two vectors are complex conjugates of each other. All three vectors are normalized to have Euclidean length,  $\text{norm}(v,2)$ , equal to one.

The matrix  $V \cdot D \cdot \text{inv}(V)$ , which can be written more succinctly as  $V \cdot D / V$ , is within roundoff error of A. And,  $\text{inv}(V) \cdot A \cdot V$ , or  $V \setminus A \cdot V$ , is within roundoff error of D.

### Multiple Eigenvalues

Some matrices do not have an eigenvector decomposition. These matrices are not diagonalizable. For example:

$$A = \begin{bmatrix} 6 & 12 & 19 \\ -9 & -20 & -33 \\ 4 & 9 & 15 \end{bmatrix}$$

For this matrix

$$[V,D] = \text{eig}(A)$$

produces

V =

$$\begin{bmatrix} -0.4741 & -0.4082 & -0.4082 \end{bmatrix}$$

0.8127 0.8165 0.8165  
-0.3386 -0.4082 -0.4082

**D =**

-1.0000 0 0  
0 1.0000 0  
0 0 1.0000

**There is a double eigenvalue at  $\lambda = 1$ . The second and third columns of  $V$  are the same. For this matrix, a full set of linearly independent eigenvectors does not exist.**

**Result: - Verified by executing the m-file**

## Experiment No.-4

**Object:- Polynomial Curve Fitting.**

**Software Used:- MATLAB 7.0**

**Over determined Systems**

Over determined systems of simultaneous linear equations are often encountered in various kinds of curve fitting to experimental data. Here is a hypothetical example. A quantity  $y$  is measured at several different values of time,  $t$ , to produce the following observations:

$t$	$y$
0.0	0.82
0.3	0.72
0.8	0.63
1.1	0.60
1.6	0.55
2.3	0.50

Enter the data into MATLAB with the statements

```
t = [0 .3 .8 1.1 1.6 2.3]';
```

```
y = [.82 .72 .63 .60 .55 .50]';
```

Try modeling the data with a decaying exponential function:

$$y(t) = c_1 + c_2 e^{-t}$$

The preceding equation says that the vector  $y$  should be approximated by a linear combination of two other vectors, one the constant vector containing all

ones and the other the vector with components  $e^{-t}$ . The unknown coefficients,  $c_1$  and  $c_2$ , can be computed by doing a least-squares fit, which minimizes the sum of the squares of the deviations of the data from the model. There are six equations in two unknowns, represented by the 6-by-2 matrix:

$$E = [\text{ones}(\text{size}(t)) \exp(-t)]$$

E =

1.0000 1.0000

1.0000 0.7408

1.0000 0.4493

1.0000 0.3329

1.0000 0.2019

1.0000 0.1003

Use the backslash operator to get the least-squares solution:

$$c = E \backslash y$$

c =

0.4760

0.3413

In other words, the least-squares fit to the data is

$$y(t) = 0.4760 + 0.3413 e^{-t}$$

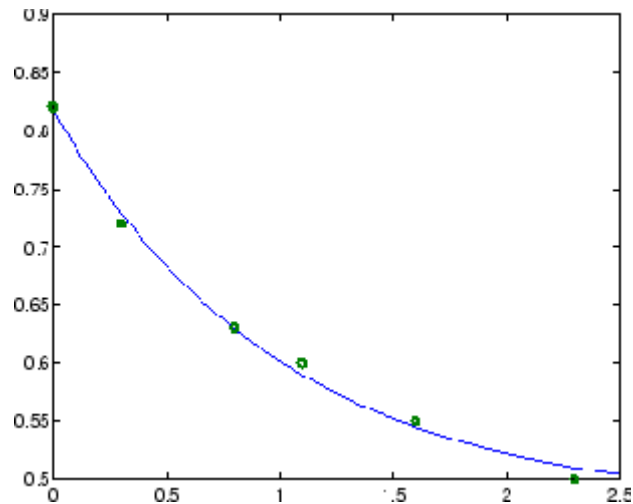
The following statements evaluate the model at regularly spaced increments in  $t$ , and then plot the result, together with the original data:

$$T = (0:0.1:2.5)';$$

$$Y = [\text{ones}(\text{size}(T)) \exp(-T)] * c;$$

$$\text{plot}(T, Y, '-', t, y, 'o')$$

$E * c$  is not exactly equal to  $y$ , but the difference might well be less than measurement errors in the original data.



A rectangular matrix  $A$  is *rank deficient* if it does not have linearly independent columns. If  $A$  is rank deficient, the least-squares solution to  $AX = B$  is not unique. The backslash operator,  $A \setminus B$ , issues a warning if  $A$  is rank deficient and produces a least-use the format command to display the solution in *rational* format. The particular solution is obtained with

```
format rat
```

```
p = R\b
```

```
p =
```

```
0
```

```
-3/7
```

```
0
```

```
29/7
```

One of the nonzero components is  $p(2)$  because  $R(:,2)$  is the column of  $R$  with largest norm. The other nonzero component is  $p(4)$  because  $R(:,4)$  dominates after  $R(:,2)$  is eliminated.

The complete solution to the underdetermined system can be characterized by adding an arbitrary vector from the null space, which can be found using the null function with an option requesting a rational basis:

$$Z = \text{null}(R, 'r')$$

$$Z =$$

$$\begin{array}{cc} -1/2 & -7/6 \end{array}$$

$$\begin{array}{cc} -1/2 & 1/2 \end{array}$$

$$\begin{array}{cc} 1 & 0 \end{array}$$

$$\begin{array}{cc} 0 & 1 \end{array}$$

It can be confirmed that  $R*Z$  is zero and that any vector  $x$  where

$$x = p + Z*q$$

for an arbitrary vector  $q$  satisfies  $R*x = b$ .

**Result:** - Verified by executing the m-file

## Experiment No.-5

**Object:** - Determination of roots of a polynomial.

**Software Used:-** MATLAB 7.0

**Theory:-**

### Representing Polynomials

MATLAB software represents polynomials as row vectors containing coefficients ordered by descending powers. For example, consider the equation

$$p(x) = x^3 - 2x - 5$$

This is the celebrated example Wallis used when he first represented Newton's method to the French Academy. To enter this polynomial into MATLAB, use

$$p = [1 \ 0 \ -2 \ -5];$$

### Roots

The roots function calculates the roots of a polynomial:

$$r = \text{roots}(p)$$

$$r =$$

$$2.0946$$

$$-1.0473 + 1.1359i$$

$$-1.0473 - 1.1359i$$

By convention, the MATLAB software stores roots in column vectors. The function poly returns to the polynomial coefficients:

$$p2 = \text{poly}(r)$$

$$p2 =$$

$$1 \ 8.8818e-16 \ -2 \ -5$$

**poly** and **roots** are inverse functions, up to ordering, scaling, and roundoff error.

## **poly** –

**Polynomial with specified roots**

### *Syntax*

**p = poly(A)**

**p = poly(r)**

### *Description*

**p = poly(A)** where **A** is an **n**-by-**n** matrix returns an **n+1** element row vector whose elements are the coefficients of the characteristic polynomial,  $\det(sl - A)$ . The coefficients are ordered in descending powers: if a vector **c** has **n+1** components, the polynomial it represents is  $c_1s^n + \dots + c_n s + c_{n+1}$

**p = poly(r)** where **r** is a vector returns a row vector whose elements are the coefficients of the polynomial whose roots are the elements of **r**.

### *Remarks*

Note the relationship of this command to

**r = roots (p)**

which returns a column vector whose elements are the roots of the polynomial specified by the coefficients row vector **p**. For vectors, **roots** and **poly** are inverse functions of each other, up to ordering, scaling, and roundoff error.

### *Examples*

**MATLAB** displays polynomials as row vectors containing the coefficients ordered by descending powers. The characteristic equation of the matrix

**A =**

**1 2 3**

**4 5 6**

**7 8 0**

is returned in a row vector by **poly**:

**p = poly (A)**

$$p = \\ 1 \quad -6 \quad -72 \quad -27$$

**The roots of this polynomial (eigenvalues of matrix A) are returned in a column vector by roots:**

$$r = \text{roots}(p)$$

$$r =$$

12.1229

-5.7345

-0.3884

**Result: - Verified by executing the m-file**

## **Experiment No.-6**

**Object:-** Determination of polynomial using method of least square curve Fitting

**Software Used:-** MATLAB 7.0

**Theory:-**

### **Representing Polynomials**

MATLAB software represents polynomials as row vectors containing coefficients ordered by descending powers. For example, consider the equation

$$p(x) = x^3 - 2x - 5$$

This is the celebrated example Wallis used when he first represented Newton's method to the French Academy. To enter this polynomial into MATLAB, use

$$p = [1 \ 0 \ -2 \ -5];$$

### **Evaluating Polynomials**

The polyval function evaluates a polynomial at a specified value. To evaluate p at  $s = 5$ , use

$$\text{polyval}(p,5)$$

ans =

110

It is also possible to evaluate a polynomial in a matrix sense. In this case

$p(s) = x^3 - 2x - 5$  becomes  $p(X) = X^3 - 2X - 5I$ , where  $X$  is a square

matrix and  $I$  is the identity matrix. For example, create a square matrix  $X$  and evaluate the polynomial  $p$  at  $X$ :

$$X = [2 \ 4 \ 5; -1 \ 0 \ 3; 7 \ 1 \ 5];$$

$$Y = \text{polyvalm}(p,X)$$

$Y =$

$$377 \ 179 \ 439$$

$$111 \ 81 \ 136$$

$$490 \ 253 \ 639$$

### Polynomial Curve Fitting

`polyfit` finds the coefficients of a polynomial that fits a set of data in a least-squares sense:

$$p = \text{polyfit}(x,y,n)$$

$x$  and  $y$  are vectors containing the  $x$  and  $y$  data to be fitted, and  $n$  is the degree of the polynomial to return. For example, consider the  $x$ - $y$  test data

$$x = [1 \ 2 \ 3 \ 4 \ 5];$$

$$y = [5.5 \ 43.1 \ 128 \ 290.7 \ 498.4];$$

A third degree polynomial that approximately fits the data is

$$p = \text{polyfit}(x,y,3)$$

$p =$

$$-0.1917 \ 31.5821 \ -60.3262 \ 35.3400$$

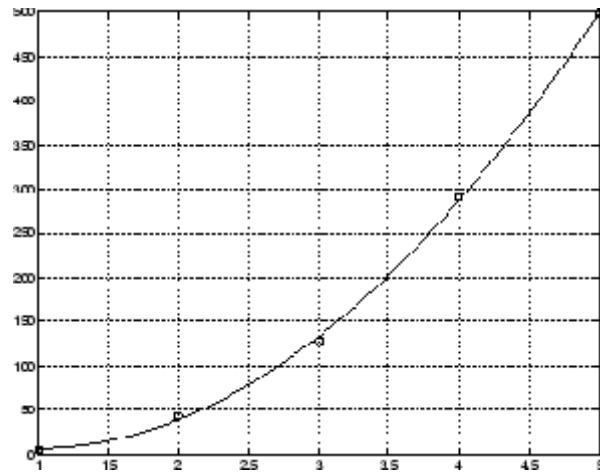
Compute the values of the `polyfit` estimate over a finer range, and plot the estimate over the real data values for comparison:

$$x2 = 1:.1:5;$$

$$y2 = \text{polyval}(p,x2);$$

`plot(x,y,'o',x2,y2)`

`grid on`



**Result:** - Verified by executing the m-file

## **Experiment No.-7**

**Object:-** Determination of polynomial fit, analyzing residuals, exponential fit and error bounds from the given data.

**Software Used:-** MATLAB 7.0

**Theory:-**

### **Representing Polynomials**

MATLAB software represents polynomials as row vectors containing coefficients ordered by descending powers. For example, consider the equation

$$p(x) = x^3 - 2x - 5$$

This is the celebrated example Wallis used when he first represented Newton's method to the French Academy. To enter this polynomial into MATLAB, use

$$p = [1 \ 0 \ -2 \ -5];$$

### **Evaluating Polynomials**

The polyval function evaluates a polynomial at a specified value. To evaluate p at  $s = 5$ , use

$$\text{polyval}(p,5)$$

$$\text{ans} = 110$$

It is also possible to evaluate a polynomial in a matrix sense. In this case  $p(s) = x^3 - 2x - 5$  becomes  $p(X) = X^3 - 2X - 5I$ , where  $X$  is a square matrix and  $I$  is the identity matrix. For example, create a square matrix  $X$  and evaluate the polynomial  $p$  at  $X$ :

$$X = [2 \ 4 \ 5; -1 \ 0 \ 3; 7 \ 1 \ 5];$$

$$Y = \text{polyvalm}(p,X)$$

$$Y =$$

$$377 \ 179 \ 439$$

$$111 \ 81 \ 136$$

$$490 \ 253 \ 639$$

## Derivatives

The `polyder` function computes the derivative of any polynomial. To obtain the derivative of the polynomial  $p = [1 \ 0 \ -2 \ -5]$ ,

$$q = \text{polyder}(p)$$

$$q =$$

$$3 \ 0 \ -2$$

`polyder` also computes the derivative of the product or quotient of two polynomials. For example, create two polynomials  $a$  and  $b$ :

$$a = [1 \ 3 \ 5];$$

$$b = [2 \ 4 \ 6];$$

Calculate the derivative of the product  $a*b$  by calling `polyder` with a single output argument:

$$c = \text{polyder}(a,b)$$

$$c =$$

$$8 \ 30 \ 56 \ 38$$

Calculate the derivative of the quotient  $a/b$  by calling `polyder` with two output arguments:

$$[q,d] = \text{polyder}(a,b)$$

$$q = -2 \quad -8 \quad -2$$

$$d =$$

$$4 \quad 16 \quad 40 \quad 48 \quad 36$$

q/d is the result of the operation.

## Convolution

Polynomial multiplication and division correspond to the operations convolution and deconvolution. The functions conv and deconv implement these operations.

Consider the polynomials  $a(s) = s^2 + 2s + 3$  and  $b(s) = 4s^2 + 5s + 6$ . To compute their product,

$$a = [1 \ 2 \ 3]; \quad b = [4 \ 5 \ 6];$$

$$c = \text{conv}(a,b)$$

$$c = 4 \quad 13 \quad 28 \quad 27 \quad 18$$

Use deconvolution to divide  $a(s)$  back out of the product:

$$[q,r] = \text{deconv}(c,a)$$

$$q =$$

$$4 \quad 5 \quad 6$$

$$r =$$

$$0 \quad 0 \quad 0 \quad 0 \quad 0$$

## Partial Fraction Expansions

RESIDUE finds the partial fraction expansion of the ratio of two polynomials. This is particularly useful for applications that represent systems in transfer function form. For polynomials  $b$  and  $a$ , if there are no multiple roots,

$$\frac{b(s)}{a(s)} = \frac{r_1}{s-p_1} + \frac{r_2}{s-p_2} + \dots + \frac{r_n}{s-p_n} + k_s$$

where  $r$  is a column vector of residues,  $p$  is a column vector of pole locations, and  $k$  is a row vector of direct terms. Consider the transfer function

$$\frac{-4s + 8}{s^2 + 6s + 8}$$

$$b = [-4 \ 8];$$

$$a = [1 \ 6 \ 8];$$

$$[r,p,k] = \text{residue}(b,a)$$

$$r = -12$$

$$8$$

$$p = -4$$

$$-2$$

$$k =$$

Given three input arguments ( $r$ ,  $p$ , and  $k$ ), `residue` converts back to polynomial form:

$$[b2,a2] = \text{residue}(r,p,k)$$

$$b2 = -4 \ 8$$

$$a2 = 1 \ 6 \ 8$$

## Polynomial Curve Fitting

`polyfit` finds the coefficients of a polynomial that fits a set of data in a least-squares sense:

$$p = \text{polyfit}(x,y,n)$$

**x** and **y** are vectors containing the *x* and *y* data to be fitted, and **n** is the degree of the polynomial to return. For example, consider the *x-y* test data

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

```
y = [5.5 43.1 128 290.7 498.4];
```

**A third degree polynomial that approximately fits the data is**

```
p = polyfit(x,y,3)
```

```
p =
```

```
-0.1917 31.5821 -60.3262 35.3400
```

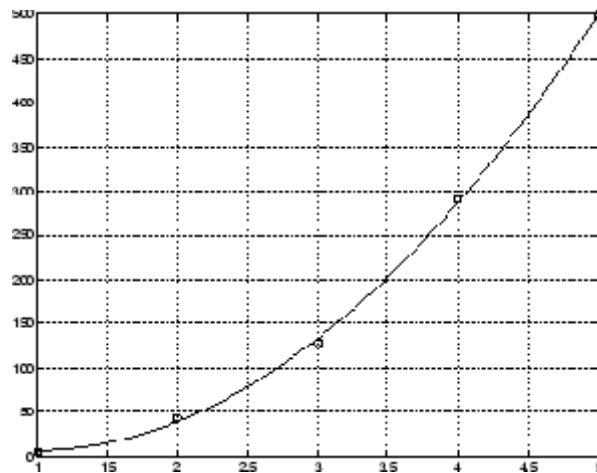
**Compute the values of the polyfit estimate over a finer range, and plot the estimate over the real data values for comparison:**

```
x2 = 1:.1:5;
```

```
y2 = polyval(p,x2);
```

```
plot(x,y,'o',x2,y2)
```

```
grid on
```



## Characteristic Polynomials

The `poly` function also computes the coefficients of the characteristic polynomial of a matrix:

```
A = [1.2 3 -0.9; 5 1.75 6; 9 0 1];
```

```
poly(A)
```

```
ans =
```

```
1.0000 -3.9500 -1.8500 -163.2750
```

The roots of this polynomial, computed with `roots`, are the *characteristic roots*, or eigenvalues, of the matrix `A`. (Use `eig` to compute the eigenvalues of a matrix directly.)

**Result:** - Verified by executing the m-file

