Ex. No. 12 **PERFORMING CALCULATIONS WITH MATRICES CALCULATING STATISTICS WITH MATRIX DATA AND VISUALIZING MATRIX DATA**

MATRIX

In MATLAB, a matrix is a rectangular array of numbers. Special meaning is sometimes attached to 1-by-1 matrices, which are scalars, and to matrices with only one row or column, which are vectors. MATLAB has other ways of storing both numeric and nonnumeric data, but in the beginning, it is usually best to think of everything as a matrix. The operations in MATLAB are designed to be as natural as possible. Where other programming languages work with numbers one at a time, MATLAB allows you to work with entire matrices quickly and easily. A good example matrix, used throughout this book, appears in the Renaissance engraving Melencolia I by the German artist and amateur mathematician Albrecht Dürer.

Entering Matrices

Matrices can be entered into MATLAB in several different ways:

•Enter an explicit list of elements.
•Load matrices from external data files.
•Generate matrices using built-in functions.
•Create matrices with your own functions in M-files.

Start by entering Dürer's matrix as a list of its elements. Few basic conventions:

•Separate the elements of a row with blanks or commas.
•Use a semicolon, ; to indicate the end of each row.
•Surround the entire list of elements with square brackets, [ ].

To enter Dürer's matrix, simply type in the Command Window

```
A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
```

MATLAB displays the matrix you just entered:

```
A =
16     3     2    13
 5    10    11     8
 9     6     7    12
 4    15    14     1
```

This matrix matches the numbers in the engraving. Once you have entered the matrix, it is automatically remembered in the MATLAB workspace. You can refer to it

simply as A. Now that you have A in the workspace, take a look at what makes it so interesting.

Sum, transpose, and diag

The special properties of a magic square have to do with the various ways of summing its elements. If you take the sum along any row or column, or along either of the two main diagonals, you will always get the same number. Let us verify that using MATLAB. The first statement to try is

```
sum(A)
```

```
ans =
34     34     34     34
```

When you do not specify an output variable, MATLAB uses the variable ans, short for *answer*, to store the results of a calculation. You have computed a row vector containing the sums of the columns of A. Sure enough, each of the columns has the same sum, the *magic* sum, 34. How about the row sums?

MATLAB has a preference for working with the columns of a matrix, so the easiest way to get the row sums is to transpose the matrix, compute the column sums of the transpose, and then transpose the result. The transpose operation is denoted by an apostrophe or single quote, '. It flips a matrix about its main diagonal and it turns a row vector into a column vector.

So

```
A'
```

produces

```
ans =
16     5     9     4
3     10     6     15
2     11     7     14
13     8     12     1
```

and

```
sum(A')'
```

produces a column vector containing the row sums

```
ans =
34
```

```
34
34
34
```

The sum of the elements on the main diagonal is obtained with the sum and the diag functions:

```
diag(A)
```

produces

```
ans =
16
10
7
1
```

And

```
sum(diag(A))
```

produces

```
ans =
34
```

The other diagonal, the so-called *antidiagonal,* is not so important mathematically, so MATLAB does not have a ready-made function for it. But a function originally intended for use in graphics, fliplr, flips a matrix from left to right:

```
sum(diag(fliplr(A)))
```

```
ans =
34
```

Subscripts

The element in row i and column j of A is denoted by A(i,j). For example, A(4,2) is the number in the fourth row and second column. For our magic square, A(4,2) is 15. So to compute the sum of the elements in the fourth column of A, type

```
A(1,4) + A(2,4) + A(3,4) + A(4,4)
```

This produces

```
ans =
```

```
34
```

But is not the most elegant way of summing a single column. It is also possible to refer to the elements of a matrix with a single subscript, A(k). This is the usual way of referencing row and column vectors. But it can also apply to a fully two-dimensional matrix, in which case the array is regarded as one long column vector formed from the columns of the original matrix. So, for our magic square, A(8) is another way of referring to the value 15 stored in A(4,2).

If you try to use the value of an element outside of the matrix, it is an error: t = A(4,5)
Index exceeds matrix dimensions. On the other hand, if you store a value in an element outside of the matrix, the size increases to accommodate the newcomer:

```
X = A;
X(4,5) = 17
```

```
X =
16     3     2    13     0
 5    10    11     8     0
 9     6     7    12     0
 4    15    14     1    17
```

The Colon Operator

The colon, :, is one of the most important MATLAB operators. It occurs in several different forms. The expression

```
1:10
```

is a row vector containing the integers from 1 to 10:

```
1     2     3     4     5     6     7     8     9    10
```

To obtain non unit spacing, specify an increment. For example,

```
100:-7:50
```

is

```
100    93    86    79    72    65    58    51
```

and

```
0:pi/4:pi
```

is

```
0    0.7854    1.5708    2.3562    3.1416
```

Subscript expressions involving colons refer to portions of a matrix: A(1:k,j) is the first k elements of the jth column of A. So

```
sum(A(1:4,4))
```

Computes the sum of the fourth column. But there is a better way. The colon by itself refers to *all* the elements in a row or column of a matrix and the keyword end refers to the *last* row or column. So

```
sum(A(:,end))
```

computes the sum of the elements in the last column of A:

```
ans =
34
```

The magic sum for a 4-by-4 square equal to 34? If the integers from 1 to 16 are sorted into four groups with equal sums, that sum must be

```
sum(1:16)/4
```

```
ans =
34
```

The magic Function

MATLAB actually has a built-in function that creates magic squares of almost any size. Not surprisingly, this function is named magic:

```
B = magic(4)
```

```
B =
16     2     3    13
5    11    10     8
9     7     6    12
4    14    15     1
```

This matrix is almost the same as the one in the Dürer engraving and has all the same "magic" properties; the only difference is that the two middle columns are exchanged. To make this B into Dürer's A, swap the two middle columns:

```
A = B(:,[1 3 2 4])
```

```
A =
```

```
16     3     2    13
 5    10    11     8
 9     6     7    12
 4    15    14     1
```

Expressions

Like most other programming languages, MATLAB provides mathematical *expressions*, but unlike most programming languages, these expressions involve entire matrices. The building blocks of expressions are:

Variables

MATLAB does not require any type declarations or dimension statements. When MATLAB encounters a new variable name, it automatically creates the variable and allocates the appropriate amount of storage. If the variable already exists, MATLAB changes its contents and, if necessary, allocates new storage. For example,

```
num_students = 25
```

creates a 1-by-1 matrix named num_students and stores the value 25 in its single element.

Variable names consist of a letter, followed by any number of letters, digits, or underscores. MATLAB uses only the first 31 characters of a variable name. MATLAB is case sensitive; it distinguishes between uppercase and lowercase letters. A and a are *not* the same variable. To view the matrix assigned to any variable, simply enter the variable name.

Numbers

MATLAB uses conventional decimal notation, with an optional decimal point and leading plus or minus sign, for numbers. *Scientific notation* uses the letter e to specify a power-of-ten scale factor. *Imaginary numbers* use either i or j as a suffix. Some examples of legal numbers are:

```
3 -99 0.0001
9.6397238 1.60210e-20 6.02252e23
1i -3.14159j 3e5i
```

Functions

MATLAB provides a large number of standard elementary mathematical functions, including abs, sqrt, exp, and sin. Taking the square root or logarithm of a negative number is not an error; the appropriate complex result is produced automatically. MATLAB also provides many more advanced mathematical functions, including Bessel

and gamma functions. Most of these functions accept complex arguments. For a list of the elementary mathematical functions, type help elfun. For a list of more advanced mathematical and matrix functions, type help specfun help elmat

- + Addition
- - Subtraction
- * Multiplication
- / Division
- \ Left division (described in "Matrices and Linear
- Algebra" in the MATLAB documentation)
- ^ Power
- ' Complex conjugate transpose
- ( ) Specify evaluation order

Some of the functions, like sqrt and sin, are *built in*. Built-in functions are part of the MATLAB core so they are very efficient, but the computational details are not readily accessible. Other functions, like gamma and sinh, are implemented in M-files. There are some differences between built-in functions and other functions.

For example, for built-in functions, you cannot see the code. For other functions, you can see the code and even modify it if you want. Several special functions provide values of useful constants. Infinity is generated by dividing a nonzero value by zero, or by evaluating well defined mathematical expressions that *overflow*, i.e., exceed realmax.

Not-a-number is generated by trying to evaluate expressions like 0/0 or Inf-Inf that do not have well defined mathematical values. The function names are not reserved. It is possible to overwrite any of them with a new variable, such as eps = 1.e-6 and then use that value in subsequent calculations. The original function can be restored with clear eps

pi 3.14159265…
i Imaginary unit,
j Same as i
eps Floating-point relative precision,
realmin Smallest floating-point number,
realmax Largest floating-point number,
Inf Infinity
NaN Not-a-number
–1
∑ 2 –52 =
2 –1022
(2 – ∑)21023

Examples of Expressions

You have already seen several examples of MATLAB expressions. Here are a few more examples, and the resulting values:

```
rho = (1+sqrt(5))/2
```

```
rho =
1.6180
```

```
a = abs(3+4i)
```

```
a =
5
```

```
z = sqrt(besselk(4/3,rho-i))
```

```
z =
0.3730+ 0.3214i
```

```
huge = exp(log(realmax))
```

```
huge =
1.7977e+308
```

```
toobig = pi*huge
```

```
toobig =
Inf
```

Generating Matrices

MATLAB provides four functions that generate basic matrices. Here are some examples:

a. zeros : All zeros

```
Z = zeros(2,4)
```

```
Z =
0     0     0     0
```

```
0     0     0     0
```

b. ones : All ones

```
F = 5*ones(3,3)
```

```
F =
5     5     5
5     5     5
5     5     5
```

c. rand : Uniformly distributed random elements

```
N = fix(10*rand(1,10))
```

```
N =
8     9     1     9     6     0     2     5     9     9
```

d. randn : Normally distributed random elements

```
R = randn(4,4)
```

```
R =
  -1.3499    0.7147    1.4090    0.7172
   3.0349   -0.2050    1.4172    1.6302
   0.7254   -0.1241    0.6715    0.4889
  -0.0631    1.4897   -1.2075    1.0347
```

The load Function

The load function reads binary files containing matrices generated by earlier MATLAB sessions, or reads text files containing numeric data. The text file should be organized as a rectangular table of numbers, separated by blanks, with one row per line, and an equal number of elements in each row. For example, outside of MATLAB, create a text file containing these four lines:

```
16.0   3.0    2.0   13.0
5.0    10.0   11.0   8.0
9.0    6.0    7.0    12.0
4.0    15.0   14.0   1.0
```

Store the file under the name magik.dat. Then the statement load magik.dat reads the file and creates a variable, magik, containing our example matrix. An easy way to read data into MATLAB in many text or binary formats is to use Import Wizard.

M-Files

You can create your own matrices using *M-files*, which are text files containing MATLAB code. Use the MATLAB Editor or another text editor to create a file containing the same statements you would type at the MATLAB command line. Save the file under a name that ends in .m. For example, create a file containing these five lines:

```
A = [ ...
16.0   3.0    2.0   13.0
5.0    10.0   11.0  8.0
9.0    6.0    7.0   12.0
4.0    15.0   14.0  1.0 ];
```

Store the file under the name magik.m. Then the statement magik reads the file and creates a variable, A, containing our example matrix.

Creating Matrices

A matrix is a two-dimensional array of numbers. In MATLAB, a matrix is created by entering each row as a sequence of space or comma separated elements, and end of a row is demarcated by a semicolon.

For example, let us create a 3-by-3 matrix as:

```
m = [1 2 3; 4 5 6; 7 8 9]
```

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

For example, let us create a 4-by-5 matrix *a*:

```
a = [ 1 2 3 4 5; 2 3 4 5 6; 3 4 5 6 7; 4 5 6 7 8]
```

```
a =
1     2     3     4     5
2     3     4     5     6
3     4     5     6     7
```

```
4       5       6       7       8
```

Referencing the Elements of a Matrix

To reference an element in the mth row and nth column, of a matrix *mx*, we write: mx(m, n);

For example, to refer to the element in the 2nd row and 5th column, of the matrix *a*, as created in the last section, we type:

```
a = [ 1 2 3 4 5; 2 3 4 5 6; 3 4 5 6 7; 4 5 6 7 8];
a(2,5)
```

```
ans =
6
```

To reference all the elements in the mth column we type A(:,m).

Let us create a column vector v, from the elements of the 4th row of the matrix a:

```
a = [ 1 2 3 4 5; 2 3 4 5 6; 3 4 5 6 7; 4 5 6 7 8];
v = a(:,4)
```

```
v =
4
5
6
7
```

You can also select the elements in the mth through nth columns, for this we write: a(:,m:n)

Let us create a smaller matrix taking the elements from the second and third columns:

```
a = [ 1 2 3 4 5; 2 3 4 5 6; 3 4 5 6 7; 4 5 6 7 8];
a(:, 2:3)
```

```
ans =
2       3
3       4
4       5
5       6
```

In the same way, you can create a sub-matrix taking a sub-part of a matrix.

```
a = [ 1 2 3 4 5; 2 3 4 5 6; 3 4 5 6 7; 4 5 6 7 8];
```

```
a(:, 2:3)
```

```
ans =
2        3
3        4
4        5
5        6
```

In the same way, you can create a sub-matrix taking a sub-part of a matrix.
For example, let us create a sub-matrix *sa* taking the inner subpart of a:

3 4 5
4 5 6

To do this, write:

```
a = [ 1 2 3 4 5; 2 3 4 5 6; 3 4 5 6 7; 4 5 6 7 8];
sa = a(2:3,2:4)
```

```
sa =
3        4        5
4        5        6
```

Deleting a Row or a Column in a Matrix

You can delete an entire row or column of a matrix by assigning an empty set of square braces [] to that row or column. Basically, [] denotes an empty array.

For example, let us delete the fourth row of a:

```
a = [ 1 2 3 4 5; 2 3 4 5 6; 3 4 5 6 7; 4 5 6 7 8];
a( 4 , : ) = []
```

MATLAB will execute the above statement and return the following result:

```
a =
1        2        3        4        5
2        3        4        5        6
3        4        5        6        7
```

Next, let us delete the fifth column of a:

```
a = [ 1 2 3 4 5; 2 3 4 5 6; 3 4 5 6 7; 4 5 6 7 8];
a(: , 5)=[]
```

MATLAB will execute the above statement and return the following result:

```
a =
1    2    3    4
2    3    4    5
3    4    5    6
4    5    6    7
```

Example

In this example, let us create a 3-by-3 matrix m, then we will copy the second and third rows of this matrix twice to create a 4-by-3 matrix.

Create a script file with the following code:

```
a = [ 1 2 3 ; 4 5 6; 7 8 9];
new_mat = a([2,3,2,3],:)
```

When you run the file, it displays the following result:

```
new_mat =
4    5    6
7    8    9
4    5    6
7    8    9
```

Matrix Operations

In this section, let us discuss the following basic and commonly used matrix operations:

- Addition and Subtraction of Matrices
- Division of Matrices
- Scalar Operations of Matrices
- Transpose of a Matrix
- Concatenating Matrices
- Matrix Multiplication
- Determinant of a Matrix
- Inverse of a Matrix

Addition and Subtraction of Matrices

You can add or subtract matrices. Both the operand matrices must have the same number of rows and columns.

Example

Create a script file with the following code:

```
a = [ 1 2 3 ; 4 5 6; 7 8 9];
b = [ 7 5 6 ; 2 0 8; 5 7 1];
c = a + b
d = a - b
```

```
c =
8     7     9
6     5    14
12    15    10

d =
-6    -3    -3
2     5    -2
2     1     8
```

Division (Left, Right) of Matrix

You can divide two matrices using left (\) or right (/) division operators. Both the operand matrices must have the same number of rows and columns.

Example

Create a script file with the following code:

```
a = [ 1 2 3 ; 4 5 6; 7 8 9];
b = [ 7 5 6 ; 2 0 8; 5 7 1];
c = a / b
d = a \ b
```

When you run the file, it displays the following result:

```
c =
-0.5254    0.6864    0.6610
 -0.4237    0.9407    1.0169
 -0.3220    1.1949    1.3729

d =
1.0e+17 *

  -0.3603   -0.5404    0.4053
```

```
  0.7206    1.0809   -0.8106
 -0.3603   -0.5404    0.4053
```

Scalar Operations of Matrices

When you add, subtract, multiply or divide a matrix by a number, this is called the **scalar operation**. Scalar operations produce a new matrix with same number of rows and columns with each element of the original matrix added to, subtracted from, multiplied by or divided by the number.

Example

Create a script file with the following code:

```
a = [ 10 12 23 ; 14 8 6; 27 8 9];
b = 2;
c = a + b
d = a - b
e = a * b
f = a / b
```

```
c =
12      14      25
16      10      8
29      10      11

d =
8       10      21
12      6       4
25      6       7

e =
20      24      46
28      16      12
54      16      18

f =
5.0000          6.0000          11.5000
7.0000          4.0000          3.0000
13.5000         4.0000          4.5000
```

Transpose of a Matrix : The transpose operation switches the rows and columns in a matrix. It is represented by a single quote(').

Example

Create a script file with the following code:

```
a = [ 10 12 23 ; 14 8 6; 27 8 9]
b = a'
```

When you run the file, it displays the following result:

```
a =
10      12      23
14       8       6
27      8       9

b =
10      14      27
12       8       8
23      6       9
```

Concatenating Matrices

You can concatenate two matrices to create a larger matrix. The pair of square brackets '[]' is the concatenation operator.

MATLAB allows two types of concatenations:

- Horizontal concatenation
- Vertical concatenation

When you concatenate two matrices by separating those using commas, they are just appended horizontally. It is called horizontal concatenation. Alternatively, if you concatenate two matrices by separating those using semicolons, they are appended vertically. It is called vertical concatenation.

Example

Create a script file with the following code:

```
a = [ 10 12 23 ; 14 8 6; 27 8 9]
b = [ 12 31 45 ; 8 0 -9; 45 2 11]
c = [a, b]
d = [a; b]
```

```
a =
10      12      23
14       8       6
27       8       9

b =
12      31      45
8       0       -9
45      2       11

c =
10      12      23      12      31      45
14      8       6       8       0       -9
27      8       9       45      2       11

d =
10      12      23
14      8       6
27       8      9
12      31      45
8       0       -9
45      2       11
```

Matrix Multiplication

Consider two matrices A and B. If A is an m x n matrix and B is an n x p matrix, they could be multiplied together to produce an m x n matrix C. Matrix multiplication is possible only if the number of columns n in A is equal to the number of rows n in B. In matrix multiplication, the elements of the rows in the first matrix are multiplied with corresponding columns in the second matrix. Each element in the (i, j)th position, in the resulting matrix C, is the summation of the products of elements in ith row of first matrix with the corresponding element in the jth column of the second matrix. Matrix multiplication in MATLAB is performed by using the * operator.

Example

Create a script file with the following code:

```
a = [ 1 2 3; 2 3 4; 1 2 5]
b = [ 2 1 3 ; 5 0 -2; 2 3 -1]
```

```
prod = a * b
```

```
a =
1     2     3
2     3     4
1     2     5

b =
2     1     3
5     0    -2
2     3    -1

prod =
18    10    -4
27    14    -4
22    16    -6
```

Determinant of a Matrix

Determinant of a matrix is calculated using the **det** function of MATLAB. Determinant of a matrix A is given by det(A).

Example

Create a script file with the following code:

```
a = [ 1 2 3; 2 3 4; 1 2 5]
det(a)
```

```
a =
1     2     3
2     3     4
1     2     5

ans =
-2
```

Inverse of a Matrix

The inverse of a matrix A is denoted by A−1 such that the following relationship holds:

$AA^{-1} = A^{-1}A = 1$

The inverse of a matrix does not always exist. If the determinant of the matrix is zero, then the inverse does not exist and the matrix is singular. Inverse of a matrix in MATLAB is calculated using the **inv** function. Inverse of a matrix A is given by inv(A).

Example

Create a script file and type the following code:

```
a = [ 1 2 3; 2 3 4; 1 2 5]
inv(a)
```

```
a =
1     2     3
2     3     4
1     2     5

ans =
-3.5000    2.0000    0.5000
3.0000    -1.0000   -1.0000
-0.5000         0    0.5000
```

CODING*: Matrices (Multiplication, Addition and Inversion)*

```
%defining a matrix
a=[1 2 3;4 5 6;7 8 9]
%defined a matrix of order 3x3
%matrix addition: Two matrices can be added provided their dimensions match
a=[1 2 3;4 5 6;7 8 9];b=eye(3); %b is a unit matrix of order 3
c=a+b

%matriix multiplication two matrices can be multiplied iff the no.of
%columns of first matrix equals no. of or rows in 2nd matrix
a=[1 2 3;4 5 6] %matriix a is of order 2x3
%So matrix b must be of order 3xn where n>=1 so that the resultant matrix c=a*b will be of order 2xn
b=[1 2;3 4;5 6]

%is of order 2x3, now the resultant matrix c=a*b is valid and of order 2x2
c=[22 28
   49 64]
```

Inverse of a matrix: Inverse of a matrix a can be found out provided a is a square non-null matrix. i.e. determinant of a is not 0, The command for finding determinant is det(a).

```
a=[1 2 3;4 5 8;8 9 11];
b=det(a)
c=inv(a)
```

To know the characteristic equation of a matrix, cheq=poly(a) is used
To find eigen values and eigen vectors of a square matrix (non null)
[evec eval]=eig(a)

To extract specific rows and columns of a matrix a
b=a([1 2],:) extracts first second rows of a matrix
b=a([1 3],:) extracts first and third rows of a matrix
b=a(:,[1 2]) extracts first second columns of a matrix
b=a(:,[1 3]) extracts first and third columns of a matrix

To delete specific rows and columns of a matrix a
a([1],:)=[] deletes the first row of a matrix a
a(:,[1])=[] deletes the first column of a matrix a

To know the rank of a matrix the command is r=rank(a)

CALCULATING STATISTICS WITH MATRIX DATA

Consider a system of linear equations in two variables. General form

$$ax+by=h$$

$$cx+dy=k$$

where a, b, c, d, h, and k are real constants and neither a and b nor c and d are both zero.

Solving basic algebraic equations in MATLAB:

The **solve** function is used for solving algebraic equations. In its simplest form, the solve function takes the equation enclosed in quotes as an argument.

To solve for x in the equation x-5 = 0

```
solve('x-5=0')
```

MATLAB will execute the above statement and return the following result −

```
ans =
  5
```

```
y =solve('x-5 = 0')
```

MATLAB will execute the above statement and return the following result −

```
y =
  5
```

If the equation involves multiple symbols, then MATLAB by default assumes that we are solving for x.

The solve function has another form −

```
solve(equation, variable)
```

we can also mention the variable.

To solve the equation $v - u - 3t^2 = 0$, for v.

```
solve('v-u-3*t^2=0','v')
```

```
ans =
  3*t^2 + u
```

Solving Quadratic Equations in MATLAB

The **solve** function can also solve higher order equations. It is often used to solve quadratic equations. The function returns the roots of the equation in an array.

To solve the quadratic equation $x^2 - 7x + 12 = 0$.

```
eq='x^2 -7*x + 12 = 0';

s = solve(eq);

disp('The first root is: '),disp(s(1));

disp('The second root is: '),disp(s(2));
```

```
The first root is:

  3

The second root is:

  4
```

Solving Higher Order Equations in MATLAB

The **solve** function can also solve higher order equations. To solve a cubic equation as $(x-3)^2(x-7) = 0$

```
solve('(x-3)^2*(x-7)=0')
```

```
ans =
  3
  3
  7
```

In case of higher order equations, roots are long containing many terms. We can get the numerical value of such roots by converting them to double. The following example solves the fourth order equation $x^4 - 7x^3 + 3x^2 - 5x + 9 = 0$.

Create a script file and type the following code −

```
eq='x^4 - 7*x^3 + 3*x^2 - 5*x + 9 = 0';

s = solve(eq);

disp('The first root is: '),disp(s(1));

disp('The second root is: '),disp(s(2));

disp('The third root is: '),disp(s(3));

disp('The fourth root is: '),disp(s(4));



% converting the roots to double type

disp('Numeric value of first root'),disp(double(s(1)));

disp('Numeric value of second root'),disp(double(s(2)));

disp('Numeric value of third root'),disp(double(s(3)));

disp('Numeric value of fourth root'),disp(double(s(4)));
```

```
The first root is:
6.630396332390718431485053218985
 The second root is:
1.059780463302589629168277249885
 The third root is:
- 0.345088397846654030326665234486 75 - 1.077836295463017659683110926979 3*i
 The fourth root is:
- 0.345088397846654030326665234486 75 + 1.077836295463017659683110926979 3*i
Numeric value of first root
   6.6304
Numeric value of second root
   1.0598
Numeric value of third root
   -0.3451 - 1.0778i
Numeric value of fourth root
   -0.3451 + 1.0778i
```

Solving System of Equations in MATLAB

The **solve** function can also be used to generate solutions of systems of equations involving more than one variables.

To solve the equations −

5x + 9y = 5

3x – 6y = 4

Create a script file and type the following code −

```
s =solve('5*x + 9*y = 5','3*x - 6*y = 4');

s.x

s.y
```

```
ans =

   22/19

ans =

   -5/57
```

In same way, you can solve larger linear systems.

x + 3y -2z = 5
3x + 5y + 6z = 7
2x + 4y + 3z = 8

EXPANDING AND COLLECTING FUNCTION

The expand and the collect function expands and collects an equation respectively. Expand and collect function expands and collects an equation.

```
syms x   %symbolic variable x
syms y   %symbolic variable x
% expanding equations
expand((x-5)*(x+9))
expand((x+2)*(x-3)*(x-5)*(x+7))
expand(sin(2*x))
expand(cos(x+y))

% collecting equations
collect(x^3*(x-7))
collect(x^4*(x-3)*(x-5))
A =[5,9;3,-6];
b =[5;4];
A \ b
ans =
   1.157895
  -0.087719
ans =
   x^2 + 4*x - 45
ans =
   x^4 + x^3 - 43*x^2 + 23*x + 210
ans =
   2*cos(x)*sin(x)
ans =
   cos(x)*cos(y) - sin(x)*sin(y)
ans =
```

```
  x^4 - 7*x^3
ans =
  x^6 - 8*x^5 + 15*x^4
```

Factorization and Simplification of Algebraic Expressions

The **factor** function factorizes an expression and the **simplify** function simplifies an expression.

Example

```
syms x
syms y
factor(x^3- y^3)
factor([x^2-y^2,x^3+y^3])
simplify((x^4-16)/(x^2-4))
```

When you run the file, it displays the following result −

```
ans =
  (x - y)*(x^2 + x*y + y^2)
ans =
  [ (x - y)*(x + y), (x + y)*(x^2 - x*y + y^2)]
ans =
  x^2 + 4
```

SOLVING LINEAR EQUATIONS

Aim: To solve a simultaneous linear equation involving three variables, algebraic equation Linear equation

A linear equation Ax=b is solvable provided matrix A is not null or if the rank(A,b)=rank(A) x is a coefficient (column) matrix.

x=[x y z……]'; b is the resultant matrix

To solve the system of equations

3x+4y+5z=12; 4x-3y+2z=3 and 5x+2y-3z=4, the MATLAB code is

[x y z]=solve('3*x+4*y+5*z=12','4*x-3*y+2*z=3','5*x+2*y-3*z=4')

Another way of doing it is:

syms x y z

A=[3 4 5;4 -3 2;5 2 -3]; b=[12 3 4]';

x=A\b or x=inv(A)*b

REAL WORLD FOOD PROCESSING PROBLEM

Manufacturing: Production Scheduling in a dairy engineering machinery plant

An industry wishes to produce three types of souvenirs involving dairy equipments: types A, B, and C. To manufacture a type-A souvenir requires 2 minutes on machine I, 1 minute on machine II, and 2 minutes on machine III. A type-B souvenir requires 1 minute on machine I, 3 minutes on machine II, and 1 minute on machine III. A type-C souvenir requires 1 minute on machine I and 2 minutes each on machines II and III. There are 3 hours available on machine I, 5 hours available on machine II, and 4 hours available on machine III for processing the order. How many souvenirs of each type should Ace Novelty make in order to use all of the available time? Formulate and solve the problem using MATLAB.

2x+y+z=180
x+3y+2z=300
2x+y+2z=240

```
s=solve('2*x+y+z=180','x+3*y+2*z=300','2*x+y+2*z=240');
s.x
ans =
36
>>s.y
ans =
48

>>s.z
ans =
```

60
`syms x y z
a=[2 1 1;1 3 2;2 1 2];
b=[180 300 240]';
x=a\b;
x=inv(a)*b
x =
36.0000
48.0000
60.0000

In its final form, the solution to the given system of equations can be easily read off as x=36, y=48,z=60.

Work to be done:

1. Create a sub matrix
2. Addition and Subtraction of Matrices
3. Division of Matrices
4. Scalar Operations of Matrices
5. Transpose of a Matrix
6. Concatenating Matrices
7. Matrix Multiplication
8. Determinant of a Matrix
9. Inverse of a Matrix