Lecture Two:

✓ Matrix generation

- Entering a vector
- Entering a matrix
- Matrix indexing
- Colon operator
- Linear spacing
- Colon operator in a matrix
- Creating a sub-matrix
- Matrix generators

Introduction

Matrices are the basic elements of the MATLAB environment. A matrix is a two-dimensional array consisting of m rows and n columns. Special cases are column vectors (n = 1) and row vectors (m = 1).

In this section we will illustrate how to apply different operations on matrices. MATLAB supports two types of operations, known as matrix operations and array operations.

Matrix generation

Matrices are fundamental to MATLAB. Therefore, we need to become familiar with matrix generation and manipulation. Matrices can be generated in several ways.

- Entering a vector

A vector is a special case of a matrix. The purpose of this section is to show how to create vectors and matrices in MATLAB. As discussed earlier, an array of dimension 1 x n is called a **row vector**, whereas an array of dimension m x 1 is called a **column vector**. The elements of vectors in MATLAB are enclosed by square brackets and are separated by spaces or by commas. For example, to enter a row vector, v, type

```
>> v = [1 4 7 10 13]
          v = 1    4    7    10    13
```

Column vectors are created in a similar way, however, semicolon (;) must separate the components of a column vector,

```
>> w = [1;4;7;10;13]
         w =
                  1
                  4
                  7
                  10
                  13
```

On the other hand, a row vector is converted to a column vector using the transpose operator. The transpose operation is denoted by an apostrophe or a single quote (').

```
>> w = v'
      w =
            1
            4
            7
            10
            13
```

Thus, v(1) is the first element of vector v, v(2) its second element, and so forth. Furthermore, to access blocks of elements, we use MATLAB's colon notation (:). For example, to access the first three elements of v, we write,

```
>> v(1:3)
    ans = 1  4  7
```

Or, all elements from the third through the last elements,

```
>> v(3,end)

 ans = 7    10    13
```

where end signifies the last element in the vector. If v is a vector, writing

```
>> v(:)
```

produces a column vector, whereas writing

```
>> v(1:end)
```

produces a row vector.

- Entering a matrix

A matrix is an array of numbers. To type a matrix into MATLAB you must

- ❖ begin with a square bracket, [
- ❖ separate elements in a row with spaces or commas (,)
- ❖ use a semicolon (;) to separate rows
- ❖ end the matrix with another square bracket,].

Here is a typical example. To enter a matrix A, such as,

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

type,

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

MATLAB then displays the 3 × 3 matrix as follows,

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

Note ! that the use of semicolons (;) here is different from their use mentioned earlier to suppress output or to write multiple commands in a single line. Once we have entered the matrix, it is automatically stored and remembered in the Workspace. We can refer to it simply as matrix A. We can then view a particular element in a matrix by specifying its location. We write,

```
>> A(2,1)

    ans =

        4
```

A(2,1) is an element located in the second row and first column. Its value is 4.

- **Matrix indexing**

We select elements in a matrix just as we did for vectors, but now we need two indices. The element of row i and column j of the matrix A is denoted by A(i,j). Thus, A(i,j) in MATLAB refers to the element Aij of matrix A. The first index is the row number and the second index is the column number. For example, A(1,3) is an element of first row and hird column. Here, A(1,3)=3.

Correcting any entry is easy through indexing. Here we substitute A(3,3)=9 by A(3,3)=0. The result is

```
>> A(3,3) = 0
A =
      1     2     3
```

```
     4     5     6
     7     8     0
```

Single elements of a matrix are accessed as A(i,j), where i ≥ 1 and j ≥ 1. Zero or negative subscripts are not supported in MATLAB.

- Colon operator

The colon operator will prove very useful and understanding how it works is the key to efficient and convenient usage of MATLAB. It occurs in several different forms. Often we must deal with matrices or vectors that are too large to enter one element at a time. For example, suppose we want to enter a vector x consisting of points

(0, 0.1, 0.2, 0.3, · · · , 5). We can use the command

```
>> x = 0:0.1:5;
```

The row vector has 51 elements.

- Linear spacing

On the other hand, there is a command to generate linearly spaced vectors: linspace. It is similar to the colon operator (:), but gives direct control over the number of points. For example,

```
y = linspace(a,b)
```

generates a row vector y of 100 points linearly spaced between and including a and b.

```
y = linspace(a,b,n)
```

generates a row vector y of n points linearly spaced between and including a and b. This is useful when we want to divide an interval into a number of subintervals of the same length. For example,

```
>> theta = linspace(0,2*pi,101)
```

divides the interval [0, 2π] into 100 equal subintervals, then creating a vector of 101 elements.

- Colon operator in a matrix

The colon operator can also be used to pick out a certain row or column. For example, the statement A(m:n,k:l specifies rows m to n and column k to l. Subscript expressions refer to portions of a matrix. For example,

```
>> A(2,:)
ans =
      4    5     6
```

is the second row elements of A. The colon operator can also be used to extract a sub-matrix from a matrix A.

```
>> A(:,2:3)
ans =
        2        3
        5        6
        8         0
```

A(:,2:3) is a sub-matrix with the last two columns of A. A row or a column of a matrix can be deleted by setting it to a null vector, [ ].

```
>> A(:,2)=[]

 ans =

        1        3

        4        6

        7        0
```

- Creating a sub-matrix

To extract a submatrix B consisting of rows 2 and 3 and columns 1 and 2 of the matrix A, do the following

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

B =

    4        5

    7        8
```

To interchange rows 1 and 2 of A, use the vector of row indices together with the colon operator.

```
>> C = A([2 1 3],:)
C =
        4    5    6
        1    2    3
        7    8    0
```

It is important to note that the colon operator (:) stands for all columns or all rows. To create a vector version of matrix A, do the following

```
>> A(:)
ans =
        1
        2
        3
        4
        5
        6
        7
        8
        0
```

The submatrix comprising the intersection of rows p to q and columns r to s is denoted by

A(p:q,r:s).

As a special case, a colon (:) as the row or column specifier covers all entries in that row or column; thus

- A(:,j) is the jth column of A, while
- A(i,:) is the ith row, and
- A(end,:) picks out the last row of A.

The keyword end, used in A(end,:), denotes the last index in the specified dimension. Here are some examples.

```
>> A
A =
        1     2     3
        4     5     6
        7     8     9

>> A(2:3,2:3)
ans =
        5     6
        8     9
>> A(end:-1:1,end)
ans =
        9
        6
        3
>> A([1 3],[2 3])
ans =
        2      3
        8      9
```

- Deleting row or column

To delete a row or column of a matrix, use the empty vector operator, [ ].

```
>> A(3,:) = []
A =
        1    2    3
        4    5    6
```

Third row of matrix A is now deleted. To restore the third row, we use a technique for

creating a matrix

```
>> A = [A(1,:);A(2,:);[7 8 0]]
A =
        1    2    3
        4    5    6
        7    8    0
```

Matrix A is now restored to its original form.

- Dimension

To determine the dimensions of a matrix or vector, use the command size. For example,

```
>> size(A)
ans =
       3    3
```

means 3 rows and 3 columns.

Or more explicitly with,

```
>> [m,n]=size(A)
```

- Transposing a matrix

The transpose operation is denoted by an apostrophe or a single quote ('). It flips a matrix about its main diagonal and it turns a row vector into a column vector. Thus,

```
>> A'
ans =
        1     4     7
        2     5     8
        3     6     0
```

By using linear algebra notation, the transpose of m x n real matrix A is the n x m matrix that results from interchanging the rows and columns of A. The transpose matrix is denoted A^T.

- Matrix generators

MATLAB provides functions that generates elementary matrices. The matrix of zeros, the matrix of ones, and the identity matrix are returned by the functions zeros, ones, and eye, respectively.

Table 1: Elementary matrices

`eye(m,n)`	Returns an m-by-n matrix with 1 on the main diagonal
`eye(n)`	Returns an n-by-n square identity matrix
`zeros(m,n)`	Returns an m-by-n matrix of zeros
`ones(m,n)`	Returns an m-by-n matrix of ones
`diag(A)`	Extracts the diagonal of matrix A
`rand(m,n)`	Returns an m-by-n matrix of random numbers

```
>> b=ones(3,1)
b =
        1
        1
        1
```

Equivalently, we can define b as >> b = [1;1;1]

```
>> eye(3)
ans =
        1     0     0
        0     1     0
        0     0     1
>> c = zeros(2,3)
c =
        0     0     0
        0     0     0
```

In addition, matrices can be constructed in a block form. With C defined by C = [1 2; 3 4], we may create a

matrix D as follows

```
>> D = [C zeros(2); ones(2) eye(2)]
D =
        1     2     0     0
        3     4     0     0
        1     1     1     0
        1     1     0     1
```