Chapter 4

Vectors and matrices in MATLAB

1. Introduction

The main element in MATLAB is the matrix, where everything is written to work with matrices. In this chapter, we introduce the use of vectors and matrices in MATLAB including their creation, functions and algebraic operation that can be implemented on vectors and matrices.

2. Vectors in MATLAB

Vectors serve as fundamental data structures for a wide range of mathematical applications. In the context of MATLAB, a vector represents a one-dimensional array of variables. The orientation of these variables within the vector distinguishes between two key types: the row vector and the column vector. When the elements are arranged in a horizontal form, it is termed a row vector. Conversely, if the elements are arranged vertically, it is called as a column vector.

2.1. Create vector in MATLAB

For creating a row vector in MATLAB, simply enclose the elements or variables within square brackets [ ] and separate them using either commas or spaces, as follows:

```
>> V= [1, 5, 8, 7, 10] % or V=[1 5 8 7 10] Creating a row vector V
V=
     1    5    8    7    10
```

For creating a column vector, we write the elements of the vector in square brackets and separate them using semi-colons (`;`).

```
>> V= [1; 5; 8; 7; 10]          % Creating a column vector V
V=
      1
      5
      8
      7
      10
```

When the elements are sequentially arranged, we can efficiently generate the vector using the following expression:

`V = [the first element : the last element]`

```
>> V = [1:10]     % or V = 1:10
V =
     1     2     3     4     5     6     7     8     9    10
```

If the variables of a vector `V` are arranged with consecutive values but with a step that differs from 1, we can generate it using the following expression:

`V = [the first element : the step : the last element]`

```
>> V = [1:2:10] % or V = 1:2:10
V =
     1     3     5     7     9
```

To generate a sequence of values evenly spaced between two defined boundaries, we can utilize the `linspace()` function, following this fundamental format:

`linspace (S,E,n)` where

S: The starting value of the vector;

E: The ending value of the vector;

n: The number of elements to be specified in the vector.

```
>> V = linspace (2,5,6)

V =

    2.0000    2.6000    3.2000    3.8000    4.4000    5.0000
```

We can also create a vector based on existing vectors or variables as indicated in these lines:

```
>> V1 = [1 2 3];

>> V2 = [4 5 6];

>> V12 = [V1 V2]


V12 =

     1     2     3     4     5     6
```

In addition to creating vectors, we have the flexibility to perform a range of operations during their construction. This includes the ability to modify, add, or remove elements within the vectors. Therefore, these operations are executed through using the following commands:

`V(x)` **:** returns the values located at position `'x'` within the vector. `x` can be integer, `end` index or vector to indicate respectively position, last element or set of elements located at the indicated vector position `x`.

```
>> V = [1, 5, 8, 7, 10];
>> V (3)

ans =

     8

>> V (end)

ans =

     10

>> V ([1 3 5])

ans =

     1     8     10
```

`V(x) = []` **:** This command removes the elements located in the position x

```
>> V = [1, 5, 8, 7, 10]
V =
     1     5     8     7    10

>> V(3) = []

V =

     1     5     7    10
```

`V(x)=a`: This command replaces the element located in the position `x` with the element a

```
>> V = [1, 5, 8, 7, 10]

V =

     1     5     8     7    10

>> V(3) = 6
```

```
V =

     1     5     6     7    10
```

`V(X) = A` : This command replaces the elements located in the vector of positions `X` (Index vector) with the elements in the vector `A`

```
>> V = [1, 5, 8, 7, 10]
V =

     1     5     8     7    10


>> V ([1 3 5]) = [2 4 6]

V =

     2     5     4     7     6
```

`V = [V, e]` : This command adds the element e to the vector `V`

```
>> V = [1, 5, 8, 7, 10]

V =

     1     5     8     7    10

>> V = [V, 2]

V =

     1     5     8     7    10     2
```

2.2. Important functions for vectors

MATLAB offers a wide range of functions for working with vectors. Here are some commonly used functions that are particularly useful for vector operations:

`length (V)` : Returns the vector size

```
>> V = [1, 5, 7, 8, 9];
length (V)

ans =

     5
```

`sum (v)` : Calculates the sum of the elements within the vector

```
>> V = [1, 5, 7, 8, 9];
>> sum (V)

ans =

    30
```

prod (v) : Calculates the product of the elements within the vector

```
>> V = [1, 5, 7, 8, 9];
>> prod (V)

ans =

        2520
```

max (v) : Returns the highest value of the vector

```
>> V = [1, 5, 7, 8, 9];
>> max (V)

ans =

     9
```

min (v) : Returns the minimum value of the vector

```
>> V = [1, 5, 7, 8, 9];
>> min (V)

ans =

     1
```

mean (v) : The average value of the elements within the vector

```
>> V = [1, 5, 7, 8, 9];
>> mean (V)

ans =

     6
```

sort (v) : Arranges the elements of the array in ascending order

```
>> V = [3 2 8 0 5];
>> sort (V)

ans =

0     2     3     5     8
```

2.3. Algebraic operations

MATLAB is a powerful programming environment for numerical computing that offers robust support for vector operations. We can perform several algebraic computations on vectors in MATLAB. The following window provides some of the common algebraic operations on vectors in MATLAB:

```
>> V = [1, 5, 7, 8, 9];
>> U = [6, 8, 4, 9, 7];

% addition

>> V + U

ans =

     7    13    11    17    16

% subtraction

>> V - U

ans =

    -5    -3     3    -1     2

% Multiplication element by element

>> V .* U

ans =

     6    40    28    72    63

% division element by element

>> V ./ U

ans =

    0.1667    0.6250    1.7500    0.8889    1.2857

% Transpose of a vector

>> V'

ans =

     1
     5
     7
     8
     9
```

Note

It is important to note that algebraic operations should be performed on vectors of the same size to ensure consistent results.

2.4. Practical work 4 (Vectors)

1. Give the result of the following expressions and validate them using MATLAB command window:

```
V1=[6 2 4];              V2=[2, 7, 9];       V3=[1:9];
V4=[V2 V1];              V5=(1:2:15);        V6=[2;8;3;7;5];
V7=linspace(2,20,6);   V8=[2:9,V7];
```

2. Given two vectors U1 = [3 1 2] and U2 = [5 2 6], follow the execution of these commands

```
>> U1(2)
>> U2(-3)
>> U3 = U1 - U2
>> U4 = U1 -5
>> U5 = U2(U1)
>> U6 = sort(U2);  U6(3) = 0
>> length(U1)
```

3. Using the command window:
   a) Create a row vector (V1) composed of 10 elements linearly spaced between 5 and 19.
   b) Create an additional row vector, V2, of identical size to V1, comprising odd integers greater than or equal to 3.
   c) Create a new vector (prodvec) as the multiplication element by element of the two vectors V1, V2.
   d) Create a new vector (divvec) as the division element by element of the two vectors V1, V2.
   e) Generate the transposed vectors of V1 and V3

4. Using MATLAB command window, calculate the contents of the following vectors.

$$A = [3\ 5\ 6\ 2];\ B = [4\ 1\ 2\ 8];\quad C = [10\ 7\ 11\ 3];$$

$$V1 = A + 2B;\quad V2 = A - \frac{B}{2};\quad V3 = B * C;$$

$$V4 = \frac{A}{5 - B}$$

$$V5 = \frac{-B + \sqrt{B^2 + 4AC}}{2a}$$

$$V6 = (V1 * e^{-3AB+2})/5$$

5. Given a vector V, write MATLAB commands to:
    a) Calculate the number of elements in V;
    b) Delete the last element of V;
    c) Chang the last three elements by e1, e2 and e3;
    d) Create a vector U containing the square of the first ten elements in the odd position of the vector V.

3. Matrix in MATLAB

Matrix in MATLAB is used to store same type of data. A matrix can be defined as a table of values with r rows and c columns where $r \times c$ represents the dimension of the matrix.

3.1. Create a matrix in MATLAB

For creating a matrix in MATLAB, we use square brackets `[]` to define the matrix's elements, separating rows with semicolons and columns with spaces or commas. The number of elements in each row must be always the same. The following example describes how to create a simple 3×3 matrix.

```
% Create a 3x3 matrix
>> M = [7, 2, 3; 9, 5, 1; 7, 4, 9]% or M=[7 2 3; 9 5 1; 7 4 9]

M =

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

In addition, we can use the enter key instead of a semicolon to separate rows.

```
% Create a 3x3 matrix using enter key
>> M = [7 2 3
9 5 1
7 4 9]

M =

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

If we need arranged raw elements, we use iterators as indicated in this example.

```
>> M = [1:3; 4:6; 7:2:11] % arranged raw elements

M =

     1     2     3
     4     5     6
     7     9    11
```

To create a zero matrix (a matrix containing only zeros) of specific dimensions in MATLAB, we can use the **`zeros`** function, as follows

```
>> null_matrix = zeros(3)

null_matrix =
     0     0     0
     0     0     0
     0     0     0

>> null_matrix1 = zeros(2,3)

null_matrix1 =

     0     0     0
     0     0     0
```

To generate a matrix filled with ones in MATLAB, we can employ the **ones** function.

```
>> ones_matrix = ones(3)

ones_matrix =

     1     1     1
     1     1     1
     1     1     1

>> ones_matrix1 = ones(2,3)

ones_matrix1 =

     1     1     1
     1     1     1
```

An identity matrix is a square matrix where the principal diagonal contains ones, and all other elements are set to zero. To generate an identity matrix, we use **`eye`** function.

```
>> identity_matrix = eye(3)

identity_matrix =

     1     0     0
     0     1     0
     0     0     1
```

In MATLAB, we can create a matrix with random numbers using **`rand()`** function.

`rand` returns a single uniformly distributed random number in the interval (0,1).

`rand(n)` returns an n-by-n matrix of random numbers.

`rand(v, u)` returns a v-by-u array of random numbers where v and u indicate the size of each dimension.

`randi(imax)` returns a pseudorandom scalar integer between `1` and `imax`.

`randi(imax,n)` returns an n-by-m matrix of pseudorandom integers drawn from the discrete uniform distribution on the interval `[1,imax]`.

`randi(imax,n,m)` returns an n-by-m array where `n, m` indicates the size of each dimension.

The below command window shows some examples of rand and **`randi`** functions.

```
>> rand

ans =

    0.8147

>> rand(2)% returns a 2 by 2 matrix of random numbers.

ans =

    0.9058    0.9134
    0.1270    0.6324

>> rand(2,3)  % returns an 2 by 3 matrix of random numbers.

ans =

    0.0975    0.5469    0.9649
    0.2785    0.9575    0.15761


>> randi(30)% returns a random integer between 1 and 30.

ans =

    18

>> randi(30,2)% returns a 2 by 2 array of random integers between 1
              % and 30

ans =

    15    11
     1     5

>> randi(30,2,3)% returns a 2 by 3 array of random integers between
                % 1 and 30

ans =

    24    16    19
    10     5     8
```

More specification of the `rand` function can be found in MATLAB help.

In addition to creating matrices, we have the flexibility to perform a range of operations during their construction. This includes the ability to modify, add, or remove elements within the matrices. Therefore, these operations are executed through using the following commands:

`M(i)`: This is called linear indexing which returns the value located at position `'i'` within the matrix where the i index is defined as provided below in the matrix.

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

`M(i,j)`: Return the values located at position `(i,j)` within the matrix, where `i` and `j` indicate the row and the column positions respectively.
`M(i:ii,j:jj)`: Refer to a subset of matrix with `i` to `ii` rows and columns `j` to `jj`.
`M(i,:); M(:,j)`: Return the row `i` or the column `j` respectively.

All the above operations are indicated in the following command window:

```
>> M = [7 2 3; 9 5 1; 7 4 9]

M =

     7     2     3
     9     5     1
     7     4     9

>> M(2) % the second element in the matrix

ans =

     9

>> M(2,3) % the element on the second row and the third column

ans =

     1

>> M(2,:)% Second row of the matrix

ans =

     9     5     1

>> M(:,3)% Third column of the matrix

ans =

     3
     1
     9
```

```
% Submatrix referred to the second and third row, first to third
% columns
>> M(2:3,1:3)

ans =

     9     5     1
     7     4     9
```

3.2. Important functions for matrices

`size ()`: This function determines the dimensions (number of rows and columns) of a matrix.

`numel()` : Returns the number of elements in a matrix.

```
>> M = [7 2 3; 9 5 1]

M =
     7     2     3
     9     5     1

>> size(M)

ans =

     2     3

>> [r, c] = size(M)% r: number of rows, c: number of columns

r =

     2
c =

     3
>> numel(M)

ans =

     6
```

`det ()`: This function computes and returns the determinant of a square matrix.

```
>> M = [7 2 3; 9 5 1; 7 4 9]
M =
     7     2     3
     9     5     1
     7     4     9

>> det(M)

ans =
  142.0000
```

trace (): This function computes the trace or the sum of the diagonal elements

```
>> M = [7 2 3; 9 5 1; 7 4 9]

M =
     7     2     3
     9     5     1
     7     4     9

>> trace(M)% trace of this matrix is = 7+5+9

ans =

    21
```

inv (): This function generates the inverse of a square matrix.

```
>> M = [7 2 3; 9 5 1; 7 4 9]

M =
     7     2     3
     9     5     1
     7     4     9

>> inv(M)

ans =

    0.2887   -0.0423   -0.0915
   -0.5211    0.2958    0.1408
    0.0070   -0.0986    0.1197
```

MATLAB offers other functions which can be applied to matrices such as: **reshape()**, **fliplr()**, **flipud()** and **rot90()**. The indicated functions are used to reshape, flip or rotate the matrix. The reader can check the MATLAB help to know how these functions work.

Moreover, in the MATLAB environment, we have the capability to perform additional operations on matrices such as inserting, deleting, and modifying elements within matrices.

Therefore, removing and adding a row or a column from a matrix is performed based on the following commands:

M(i,:) = []: This command removes the row `i` from the matrix

```
>> M = [7 2 3; 9 5 1; 7 4 9]
```

```
M =

     7     2     3

     9     5     1

     7     4     9

>> M(2,:) = []

M =

     7     2     3

     7     4     9
```

`M(:,j) =[]:` This commands removes the column `j` from the matrix

```
>> M = [7 2 3; 9 5 1; 7 4 9]

M =

     7     2     3
     9     5     1
     7     4     9

M(:,2) = []

M =

     7     3
     9     1
     7     9
```

`M(i,j) = a`: This command replaces the element located in the position `(i,j)` with the element a

`M(i,:) = V`: This command replaces the row `i` with the vector `V`.

`M(:,j) = V`: This command replaces the column `j` with the vector `V`.

```
>> M = [7 2 3; 9 5 1; 7 4 9]

M =

     7     2     3
     9     5     1
     7     4     9

>> M(:,2) = [10 11 12]

M =

     7    10     3
     9    11     1
     7    12     9
```

`M=[M, v]`: Writing this expression allows adding a new column where v is a column vector.

```
>> M = [7 2 3; 9 5 1; 7 4 9]

M =

     7     2     3
     9     5     1
     7     4     9

>> M1 = [M; [10 15 12]]

M1 =

     7     2     3
     9     5     1
     7     4     9
    10    15    12
```

`M=[M; v]`: Writing this expression allows adding a new row where v is a row vector.

```
>> M = [7 2 3; 9 5 1; 7 4 9]

M =

     7     2     3
     9     5     1
     7     4     9

>> M1 = [M, [10; 15; 12]]

M1 =

     7     2     3    10
     9     5     1    15
     7     4     9    12
```

To add a new row at the beginning of the matrix we use the notation **`M = [v; M]`**, where v is a row vector. **`M = [u, M]`** is used in MATLAB to add a new column at the beginning of the matrix `M` where u is a column vector.

3.3. Algebraic operations

MATLAB provides a wide range of algebraic operations for matrices. Therefore, the next window includes some common algebraic operations that we can perform on matrices in MATLAB environment.

```
>> A = [5 2 4; 9 2 8]

A =
     5     2     4
     9     2     8

>> B = [6 1 3; 7 2 9]

B =

     6     1     3
     7     2     9

% Adds matrices A and B element-wise
>> M1 = A + B

M1 =

    11     3     7
    16     4    17

% Subtracts matrices B from A element-wise

>> M2 = A - B

    -1     1     1
     2     0    -1

% Matrix division  element by element
>> M4 = A./B

M4 =

    0.8333    2.0000    1.3333
    1.2857    1.0000    0.8889

% Matrix multiplication element by element
>> M5 = A.*B

M5 =

    30     2    12
    63     4    72

% Matrix multiplication
>> M6 = A*B
Error using  *
Inner matrix dimensions must agree.
>> C = [2 7; 1 3; 8 4]
C =
     2     7
     1     3
     8     4
```

```
>> M6 = A*C

M6 =

    44    57

    84   101
```

3.4. Practical work 4 (Matrices)

1. Using MATLAB software, validate the following expressions:

   M1 = [6 2 ; 0 4]; M2 = [2,7,9 ; 2,3,8 ; 6,3,4]; M3 = [1:5 ; 5:2:13 ; 5:-1:1];

   M4 = [M2 M3]; M5 = [zeros(3,2); ones(3,2)].

2. Given two matrices P1 and P2 , provide the execution of these commands

$$P1\begin{pmatrix}2 & 1 & 7\\ 2 & 3 & 8\\ 7 & 5 & 9\end{pmatrix}, \qquad P2\begin{pmatrix}1 & 8 & 2\\ 9 & 2 & 8\\ 1 & 6 & 5\end{pmatrix}$$

```
>> P1(2)
>> P2(-3)
>> P3 = P1 - P2
>> P4 = P1 -5
>> P5 = P2(2,:)
>> P6 = P2(1:2, 2:3);
>> size(U1)
>> nmel(U1)
```

3. Given a matrix M = [3 8 1; 2 5 4; 7 5 3; 0 9 5], write the appropriate commands for the following queries:
   - Value of the element on the second row and the third column.
   - V1 = the first row of M;
   - Sum of elements of the second and the third column;
   - Replace the last column by fives.

4. Given a Matrix M, write MATLAB commands to:
   - Calculate the number of elements in M;
   - Delete the last row of M;
   - Calculate the number of rows and columns of M;
   - Change the third column by ones;
   - Create a matrix U containing the rows on the even position of the matrix M.