MAT 1206 – Introduction to MATLAB

CHAPTER 02: Fundamental Operators and Commands

Lesson 3: Vectors, Matrices and arrays

Content

- Vectors
- Matrices
- Arrays

Vectors

- A vector is a one-dimensional array of numbers. MATLAB allows creating two types of vectors:
  - Row vectors
  - Column vectors

Cont..

- Row Vectors

Row vectors are created by enclosing the set of elements in square brackets, using space or comma to delimit the elements.

Code	Output
r = [7 8 9 10 11] Or r = [7, 8, 9, 10, 11]	r = 7 8 9 10 11

Cont..

- Column Vectors

  Column vectors are created by enclosing the set of elements in square brackets, using semicolon to delimit the elements.

Code	Output
c = [7; 8; 9; 10; 11]	c = 7 8 9 10 11

Referencing the Elements of a Vector

You can reference one or more of the elements of a vector in several ways. The ith component of a vector v is referred as v(i). For example:

Code	Output
v = [1; 2; 3; 4; 5; 6]; % creating a column vector of 6 elements v(3)	ans = 3

Cont..

When you reference a vector with a colon, such as v(:), all the components of the vector are listed.

Code	Output
v = [1; 2; 3; 4; 5; 6]; % creating a column vector of 6 elements v(:)	ans = 1 2 3 4 5 6

Cont..

MATLAB allows you to select a range of elements from a vector.

For example, let us create a row vector rv of 9 elements, then we will reference the elements 3 to 7 by writing rv(3:7) and create a new vector named sub_rv.

Code	Output
rv = [1 2 3 4 5 6 7 8 9]; sub_rv = rv(3:7)	sub_rv = 3 4 5 6 7

Vector Operations

In this section, the following vector operations are discussed:

- Addition and Subtraction of Vectors
- Scalar Multiplication of Vectors
- Transpose of a Vector
- Appending Vectors
- Vector Dot Product
- Vectors with Uniformly Spaced Elements

Addition and Subtraction of Vectors

You can add or subtract two vectors. Both the vectors must be of same type and have same number of elements.

Example

Create a script file with the following code:

Code	Output

A = [7, 11, 15, 23, 9];
B = [2, 5, 13, 16, 20];
C = A + B;
D = A - B;
disp(C);
disp(D);
	9 16 28 39 29
	5 6 2 7 -11

Scalar Multiplication of Vectors

When you multiply a vector by a number, this is called the scalar multiplication. Scalar multiplication produces a new vector of same type with each element of the original vector multiplied by the number.

Example

Create a script file with the following code:

Code	Output

`v = [ 12 34 10 8];`
 	
`m = 5 * v`
	`m =`
	`60   170    50    40`

Transpose of a Vector

The transpose operation changes a column vector into a row vector and vice versa. The transpose operation is represented by a single quote (').

Example

Create a script file with the following code:

Code	Output

r = [1 2 3 4];
tr = r';
v = [1;2;3;4];
tv = v';
disp(tr); disp(tv);
	1
	2
	3
	4
	
	1 2 3 4

Appending Vectors

MATLAB allows you to append vectors together to create new vectors.

If you have two row vectors r1 and r2 with n and m number of elements, to create a row vector r of n plus m elements by appending these vectors, you write:

Code
r = [r1, r2]
r = [r1; r2]

However, to do the second operation, both the vectors should have same number of elements.

Cont.

Similarly, you can append two column vectors c1 and c2 with n and m number of elements. To create a column vector c of n plus m elements, by appending these vectors, you write:

Code
c = [c1; c2]
c = [c1, c2]

However, to do the first, both the vectors should have same number of elements.

Cont.

Example

Create a script file with the following code:

Code	Output	

r1 = [1 2 3 4];
r2 = [5 6 7 8];
r = [r1,r2]
rMat = [r1;r2]
c1 = [1; 2; 3; 4];
c2 = [5; 6; 7; 8];
c = [c1; c2]
cMat = [c1,c2]
	r =
	1 2 3 4 5 6 7 8
	rMat =
	1 2 3 4
	5 6 7 8
		c =
		1
		2
		3
		4
		5
		6
		7
		8
		cMat =
		1 5
		2 6
		3 7
		4 8

Vector Dot Product

Dot product of two vectors a = (a1, a2, …, an) and b = (b1, b2, …, bn) is given by:
a.b = ∑(ai.bi)

Dot product of two vectors a and b is calculated using the **dot** function.

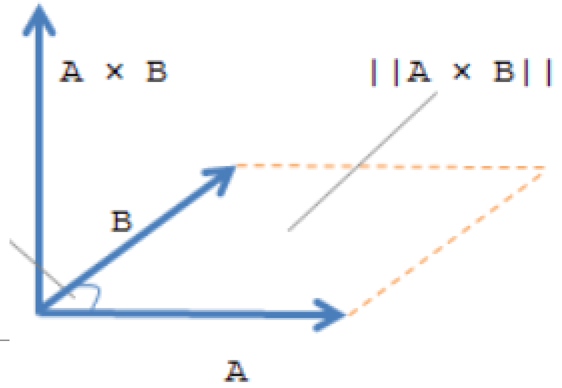
Example

Create a script file with the following code:

Code	Output

v1 = [2 3 4];
v2 = [1 2 3];
dp = dot(v1, v2);
disp('Dot Product:');
disp(dp);
	Dot Product:
	20

Vector Cross Product



The cross product between two 3-D vectors produces a new vector that is perpendicular to both.

Consider the two vectors

$$A = a_1\hat{i} + a_2\hat{j} + a_3\hat{k} \ ,$$
$$B = b_1\hat{i} + b_2\hat{j} + b_3\hat{k} \ .$$

In terms of a matrix determinant involving the basis vectors $\hat{i}$, $\hat{j}$, and $\hat{k}$, the cross product of A and B is

$$C = A \times B = \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix}$$

$$= (a_2b_3 - a_3b_2)\hat{i} + (a_3b_1 - a_1b_3)\hat{j} + (a_1b_2 - a_2b_1)\hat{k} \ .$$

Code	Output
A = [4 -2 1]; B = [1 -1 3]; C = cross(A,B)	C = -5 -11 -2

Vectors with Uniformly Spaced Elements

MATLAB allows you to create a vector with uniformly spaced elements.

To create a vector v with the first element f, last element l, and the difference between elements is any real number n, we write: v = [f : n : l]

Example

Create a script file with the following code:

Code	Output

v = [1: 2: 20];
sqv = v.^2;
disp(v);disp(sqv);
	1 3 5 7 9 11 13 15 17 19
	1 9 25 49 81 121 169 225 289 361

Matrix

A matrix is a two-dimensional array of numbers.

In MATLAB, you create a matrix by entering elements in each row as comma or space delimited numbers and using semicolons to mark the end of each row.

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

Code	Output
a = [1 2 3 4 5; 2 3 4 5 6; 3 4 5 6 7; 4 5 6 7 8]; disp(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:

Code	Output
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

Cont.

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

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

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

Cont.

Let us create a matrix v1, from the elements of the 2nd to 4th rows of the matrix a:

Code	Output
a = [1 2 3 4 5; 2 3 4 5 6; 3 4 5 6 7; 4 5 6 7 8]; v1=a(2:4,:)	v1 = 2 3 4 5 6 3 4 5 6 7 4 5 6 7 8

Cont.

Let us create a matrix v2, from the elements of the 2nd to 4th columns of the matrix a:

Code	Output
a = [1 2 3 4 5; 2 3 4 5 6; 3 4 5 6 7; 4 5 6 7 8]; v2=a(:,2:4)	v2 = 2 3 4 3 4 5 4 5 6 5 6 7

Cont.

Let us create a matrix v3, from the elements of the1st and 4th rows of the matrix a:

Code	Output
a = [1 2 3 4 5; 2 3 4 5 6; 3 4 5 6 7; 4 5 6 7 8]; v3=a([1,4],:)	v3 = 1 2 3 4 4 5 6 7

Cont.

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

Code	Output

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

Cont.

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:

Code	Output

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

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:

Code	Output

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

Cont.

Next, let us delete the fifth column of a:

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

Matrix Operations

In this section, the following basic and commonly used matrix operations are discussed:

- Addition and Subtraction 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 matrices must have the same number of rows and columns.

<table>
<tr><th>Code</th><th>Output</th></tr>
<tr><td><pre>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</pre></td><td><pre>c =
     8   7   9
     6   5   14
    12   15   10

d =
    -6   -3   -3
     2   5   -2
     2   1   8</pre></td></tr>
</table>

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.

Cont.

Code	Output

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(').

Code	Output
a = [10 12 23 ; 14 8 6; 27 8 9] b = a'	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.

Cont.

<table>
<tr><th>Code</th><th>Output</th></tr>
<tr><td>

```
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]
```

</td><td>

```
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
```

</td></tr>
</table>

Matrix Multiplication

Code	Output
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).

Code	Output
a = [1 2 3; 2 3 4; 1 2 5] det(a)	a = 1 2 3 2 3 4 1 2 5 ans = -2

If $\mathbf{A}$ is an $m \times n$ matrix and $\mathbf{B}$ is an $n \times p$ matrix,

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{pmatrix}$$

$$\begin{pmatrix} a_{11}b_{11} + \cdots + a_{1n}b_{n1} & a_{11}b_{12} + \cdots + a_{1n}b_{n2} & \cdots & a_{11}b_{1p} + \cdots + a_{1n}b_{np} \\ a_{21}b_{11} + \cdots + a_{2n}b_{n1} & a_{21}b_{12} + \cdots + a_{2n}b_{n2} & \cdots & a_{21}b_{1p} + \cdots + a_{2n}b_{np} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}b_{11} + \cdots + a_{mn}b_{n1} & a_{m1}b_{12} + \cdots + a_{mn}b_{n2} & \cdots & a_{m1}b_{1p} + \cdots + a_{mn}b_{np} \end{pmatrix}$$

Inverse of a Matrix

The inverse of a matrix A is denoted by A^{-1}such that the following relationship holds:

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

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).

Code	Output
a = [1 2 3; 2 3 4; 1 2 5]; inv(a)	ans = -3.5000 2.0000 0.5000 3.0000 -1.0000 -1.0000 -0.5000 0 0.5000

Arrays

All variables of all data types in MATLAB are multidimensional arrays. A vector is a one-dimensional array and a matrix is a two-dimensional array.

We have already discussed vectors and matrices. Now, we will discuss multidimensional arrays. However, before that, let us discuss some special types of arrays:

- zeros()
- ones()
- eye()
- rand()

zeros()

The zeros() function creates an array of all zeros:

Code	Output
zeros(2) zeros(2,3)	ans = 0 0 0 0 ans = 0 0 0 0 0 0

ones()

The ones() function creates an array of all ones:

Code	Output
ones(2) ones(2,3)	ans = 1 1 1 1 ans = 1 1 1 1 1 1

eye()

The eye() function creates an identity matrix.

Code	Output
eye(3) eye(3,2)	ans = 1 0 0 0 1 0 0 0 1 ans = 1 0 0 1 0 0

rand()

The rand() function creates an array of uniformly distributed random numbers on (0,1):

Code	Output
rand(3)	ans = 0.2373 0.5468 0.4889 0.4588 0.5211 0.6241 0.9631 0.2316 0.6791
rand(2,3)	ans = 0.3955 0.9880 0.8852 0.3674 0.0377 0.9133

Multidimensional Arrays

An array having more than two dimensions is called a multidimensional array in MATLAB.

Multidimensional arrays in MATLAB are an extension of the normal two dimensional matrix.

Generally to generate a multidimensional array, we first create a two-dimensional array and extend it.

For example, let's create a two-dimensional array a.

Code	Output
a = [7 9 5; 6 1 9; 4 3 2]	a = 7 9 5 6 1 9 4 3 2

Cont.

The array a is a 3-by-3 array; we can add a third dimension to a, by providing the values like:

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

Cont.

We can also create multidimensional arrays using the ones(), zeros() or the rand() functions.

For example,

Code	Output
b = rand(4,3,2)	b(:,:,1) = 0.8147 0.6324 0.9575 0.9058 0.0975 0.9649 0.1270 0.2785 0.1576 0.9134 0.5469 0.9706 b(:,:,2) = 0.9572 0.4218 0.6557 0.4854 0.9157 0.0357 0.8003 0.7922 0.8491 0.1419 0.9595 0.9340

Cont.

We can also use the **cat()** function to build multidimensional arrays. It concatenates a list of arrays along a specified dimension:

Syntax for the cat() function is: B = **cat(dim, A1, A2...)**

Example:

Code	Output

a = [9 8 7; 6 5 4; 3 2 1];
b = [1 2 3; 4 5 6; 7 8 9];
c = cat(3, a, b, [2 3 1; 4 7 8; 3 9 0])
	c(:,:,1) =
	9 8 7
	6 5 4
	3 2 1
	c(:,:,2) =
	1 2 3
	4 5 6
	7 8 9
	c(:,:,3) =
	2 3 1
	4 7 8
	3 9 0

Array Functions

Function	Purpose
length	Length of vector
size	Array dimensions
ndims	Number of array dimensions
isempty	Determines whether array is empty
sort	Sorts array elements in ascending or descending order
max	Return the largest elements of each column
min	Rreturn the smallest elements of each column
sum	Rreturn the sums of each column
ismember	Array elements that are members of set array
find	Can be used to find the index of an element in an array.

Length, size, ndims and isempty

Code	Output
x = [7.1, 3.4, 7.2, 28/4, 3.6, 17, 9.4, 8.9]; length(x) % length of x vector	ans = 8
y = rand(3, 4, 5, 2); ndims(y) % no of dimensions in array y	ans = 4
size(y) a = [1 2 3; 4 5 6; 7 8 9]; size(a)	ans = 3 4 5 2 ans = 3 3
isempty(a)	ans = logical 0

Sort

Code	Output
v = [23 45 12 9 5 0 19 17] % horizontal vector sort(v) %sorting v m = [2 6 4; 5 3 9; 2 0 1] % two dimensional array sort(m, 1) % or sort(m), sorting m along the column sort(m, 2) % sorting m along the row	v = 23 45 12 9 5 0 19 17 ans = 0 5 9 12 17 19 23 45 m = 2 6 4 5 3 9 2 0 1 ans = 2 0 1 2 3 4 5 6 9 ans = 2 4 6 3 5 9 0 1 2

Max

If the array is one dimensional, return the largest element. Otherwise, return the largest elements of each column.

Code	Output

a1=[4 8 9 6 3 2 7];
a2=[8;9;6;2;3;5];
a3 = [9 8 7; 6 5 4; 3 2 1];
a4=cat(3,[4 5;9 8],[8 6;4 3],[8 5;6 9]);
 	
max(a1)
max(a2)
max(a3)
max(a4)
	ans =
	9
	
	ans =
	9
	
	ans =
	9 8 7
	
	ans(:,:,1) =
	9 8
	ans(:,:,2) =
	8 6
	ans(:,:,3) =
	8 9

Min

If the array is one dimensional, return the smallest element. Otherwise, return the smallest elements of each column.

<table>
<tr><th>Code</th><th>Output</th></tr>
<tr><td><pre>
a1=[4 8 9 6 3 2 7];
a2=[8;9;6;2;3;5];
a3 = [9 8 7; 6 5 4; 3 2 1];
a4=cat(3,[4 5;9 8],[8 6;4 3],[8 5;6 9]);

min(a1)
min(a2)
min(a3)
min(a4)
</pre></td><td><pre>
ans =
    2

ans =
    2

ans =
    3   2   1

ans(:,:,1) =
    4   5
ans(:,:,2) =
    4   3
ans(:,:,3) =
    6   5
</pre></td></tr>
</table>

Sum

If the array is one dimensional, return the sum of elements. Otherwise, return the sums of each column.

Code	Output

a1=[4 8 9 6 3 2 7];
a2=[8;9;6;2;3;5];
a3 = [9 8 7; 6 5 4; 3 2 1];
a4=cat(3,[4 5;9 8],[8 6;4 3],[8 5;6 9]);
 	
sum(a1)
sum(a2)
sum(a3)
sum(a4)
	ans =
	39
	
	ans =
	33
	
	ans =
	18 15 12
	
	ans(:,:,1) =
	13 13
	ans(:,:,2) =
	12 9
	ans(:,:,3) =
	14 14

ismember

ismember(A,B) returns an array containing logical 1 (true) where the data in A is found in B. Elsewhere, the array contains logical 0 (false).

Code	Output
A = [5 3 4 2]; B = [2 4 4 4 6 8]; C = [9 8 7; 6 5 4; 3 2 1]; ismember(A,B) ismember(4,A) ismember(1,B) ismember(2,C)	ans = 1×4 logical array 0 0 1 1 ans = logical 1 ans = logical 0 ans = logical 1

Find

Find the index of an element in an array.

Code	Output
A = [5 3 4 2]; B = [9 8 7; 6 5 4; 3 2 5]; find(A==4) [r,c]=find(B==4) [r,c]=find(B==5) find(A<4)	ans = 3 r = 2 c = 3 r = 2 3 c = 2 3 ans = 2 4

Cell Array

Cell arrays are arrays of indexed cells where each cell can store an array of a different data types.

Code	Output

c = cell(2, 5);
c = {'Red', 'Blue', 'Green', 'Yellow', 'White'; 1 2 3 4 5}
	c =
	  2×5 cell array
	    {'Red'} {'Blue'} {'Green'} {'Yellow'} {'White'}
	    {[1]} {[2]} {[3]} {[4]} {[5]}

c(:,2)
 	
c(1:2,1:2)
 	
% c(1:2,[1 3 5])
	ans =
	  2×1 cell array
	    {'Blue'}
	    {[2]}
	ans =
	  2×2 cell array
	    {'Red'} {'Blue'}
	    {[1]} {[2]}

Questions/queries?