

Chapter 3

Arrays

Matrices

Example

(Tabular Display of Data) The following matrix gives the airline distance; between the indicated cities (in miles):

	London	Madrid	NewYork	Tokyo	
London	0	785	3469	5959	
Madrid	785	0	3593	6706	
New York	3469	3593	0	6757	
Tokyo	5959	6706	6757	0	

■ Example

- The windchill table that follows is a matrix .A combination of air temperature and wind speed makes a body feel cooler than the actual temperature.For example when temperature is 10°F and the wind is 15 miles per hour ,this causes body heat loss equal to that when the temperature is -18°F with no wind.

	$^{\circ}\text{F}$	$^{\circ}\text{F}$	$^{\circ}\text{F}$	$^{\circ}\text{F}$	$^{\circ}\text{F}$	$^{\circ}\text{F}$
	15	10	5	0	-5	-10
mph						
5	12	7	0	-5	-10	-15
10	-3	-9	-15	-22	-27	-34
15	-11	-18	-25	-31	-38	-45
20	-17	-24	-31	-39	-46	-53

Two Dimensional Arrays:

Matrices

A matrix has multiple rows and columns

Example: The matrix M

$$\mathbf{M} = \begin{bmatrix} 2 & 4 & 10 \\ 16 & 3 & 7 \\ 8 & 4 & 9 \\ 3 & 12 & 15 \end{bmatrix}$$

has four rows and three columns.

Creating Matrices

If the matrix is small you can type it row by row, separating the *elements* in a given row with *spaces* or *commas* and separating the *rows* with semicolons. For example, typing

Example

```
>>A = [2,4,10;16,3,7];
```

creates the following matrix:

$$A = \begin{bmatrix} 2 & 4 & 10 \\ 16 & 3 & 7 \end{bmatrix}$$

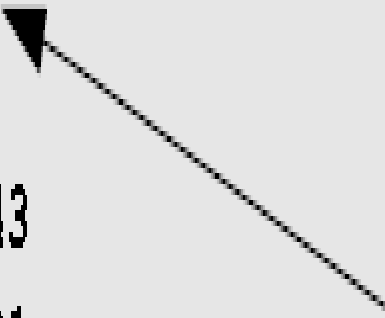
Remember, spaces or commas separate elements in different *columns*, whereas semicolons separate elements in different *rows*.

Example

```
>> a=[5 35 43; 4 76 81; 21 32 40]
```

```
a =
```

5	35	43
4	76	81
21	32	40



A semicolon is typed before
a new line is entered.

Creating Matrices from Vectors

Suppose

$a = [1, 3, 5]$

And

$b = [7, 9, 11]$

Note the difference between the results given
by

$[a \ b]$

and

$[a; b]$

in the following session:

```
>>c = [a b];  
c =  
     1     3     5     7     9    11  
>>D = [a;b]  
D =  
     1     3     5  
     7     9    11
```

You need not use symbols to create a new array. For example, you can type

```
>> D = [[1,3,5];[7,9,11]];
```


Matrices and transpose Operation

- Transpose operation interchanges rows and columns. Transpose operation is carried out by ‘

- `>> A=[-2 6;-3 5]`

- `A =`

- `-2 6`

- `-3 5`

- `>> A'`

- `ans =`


- `-2 -3`

- `6 5`

Array Addressing

- $A(:,3)$ denotes all the elements in the third column of the matrix A .
- $A(:,2:5)$ denotes all the elements in the second through fifth columns of A .
- $A(2:3,1:3)$ denotes all the elements in the second and third rows that are also in the first through third columns.
- $v = A(:)$ creates a vector v consisting of all the columns of A stacked from first to last.
- $A(\text{end},:)$ denotes the last row in A , and $A(:,\text{end})$ denotes the last column.

```
>> A=[1 3 5 7 9 11; 2 4 6 8 10 12; 3 6 9 12 15 18; 4 8 12 16  
20 24; 5 10 15 20 25 30]
```




Define a matrix A with
5 rows and 6 columns.

A =

1	3	5	7	9	11
2	4	6	8	10	12
3	6	9	12	15	18
4	8	12	16	20	24
5	10	15	20	25	30

```
>> B=A(:,3)
```



Define a column
vector B from the
elements in all of the
rows of column 3 in
matrix A.

Continued from example

B =

5
6
9
12
15

>> C=A(2,:)

C =

2 4 6 8 10 12

>> E=A(2:4,:)

E =

2 4 6 8 10 12
3 6 9 12 15 18
4 8 12 16 20 24

>> F=A(1:3,2:4)

F =

3 5 7
4 6 8
6 9 12

>>

Define a row vector C from the elements in all of the columns of row 2 in matrix A.

Define a matrix E from the elements in rows 2 through 4 and all the columns in matrix A.

Create a matrix F from the elements in rows 1 through 3 and columns 2 through 4 in matrix A.

Example

```
>> MAT=[3 11 6 5; 4 7 10 2; 13 9 0 8]
```

Create a 3×4 matrix.

```
MAT =
```

3	11	6	5
4	7	10	2
13	9	0	8

```
>> MAT(3,1)=20
```

Assign a new value to the (3,1) element.

```
MAT =
```

3	11	6	5
4	7	10	2
20	9	0	8

```
>> MAT(2,4)-MAT(1,2)
```

Use elements in a mathematical expression.

```
ans =
```

```
-9
```

You can use array indices to extract a smaller array from another array. For example, if you first create the array **B**

$$\mathbf{B} = \begin{bmatrix} 2 & 4 & 10 & 13 \\ 16 & 3 & 7 & 18 \\ 8 & 4 & 9 & 25 \\ 3 & 12 & 15 & 17 \end{bmatrix}$$

then type `C = B(2:3, 1:3)`, you can produce the following array:

$$\mathbf{C} = \begin{bmatrix} 16 & 3 & 7 \\ 8 & 4 & 9 \end{bmatrix}$$

Additional Array Functions

`[u,v,w] = find(A)` Computes the arrays u and v , containing the row and column indices of the nonzero elements of the matrix A , and the array w , containing the values of the nonzero elements. The array w may be omitted.

`length(A)` Computes the largest value of m or n if A is an $m \times n$ matrix.

continued

`max (A)` Returns a row vector containing the largest elements in each column if **A** is a matrix.

If any of the elements are complex, `max (A)` returns the elements that have the largest magnitudes.

continued

`[x,k] = max(A)` Similar to `max(A)` but stores the maximum values in the row vector `x` and their indices in the row vector `k`.

`min(A)` Same as `max(A)` but returns the minimum values

`[x,k] = min(A)` same as `[x,k]=max(A)` but returns the minimum values

continued

`size(A)`

Returns a row vector $[m \ n]$ containing the sizes of the $m \times n$ array A .

`sort(A)`

Sorts each column of the array A in ascending order and returns an array the same size as A .

`sum(A)`

Sums the elements in each column of the array A and returns a row vector containing the sums.

Example

```
>> A=[6 2;-10 -5;3 0]
```

```
A =
```

```
     6     2  
    -10    -5  
     3     0
```

```
>> max(A)
```

```
ans =
```

```
     6     2
```

```
>> min(A)
```

```
ans =
```

```
    -10    -5
```

```
>> size(A)
```

```
ans =
```

```
     3     2
```

```
>> length(A)
```

```
ans =
```

```
3
```

```
>> sort(A)
```

```
ans =
```

```
-10  -5
```

```
3    0
```

```
6    2
```

```
>> sum(A)
```

```
ans =
```

```
-1  -3
```

Example

For example, if

$$\mathbf{A} = \begin{bmatrix} 6 & 0 & 3 \\ 0 & 4 & 0 \\ 2 & 7 & 0 \end{bmatrix}$$

then the session

```
>>A = [6, 0, 3; 0, 4, 0; 2, 7, 0];  
>>[u, v, w] = nd(A)
```

returns the vectors

$$\mathbf{u} = \begin{bmatrix} 1 \\ 3 \\ 2 \\ 3 \\ 1 \end{bmatrix}$$

$$\mathbf{v} = \begin{bmatrix} 1 \\ 1 \\ 2 \\ 2 \\ 3 \end{bmatrix}$$

$$\mathbf{w} = \begin{bmatrix} 6 \\ 2 \\ 4 \\ 7 \\ 3 \end{bmatrix}$$

The vectors \mathbf{u} and \mathbf{v} give the (row, column) indices of the nonzero values, which are listed in \mathbf{w} . For example, the second entries in \mathbf{u} and \mathbf{v} give the indices (3, 1), which specifies the element in row 3, column 1 of \mathbf{A} , whose value is 2.

- The complete syntax of the sort function is `sort(A, dim, mode)`, where `dim` selects a dimension along which to sort and `mode` selects the direction of the sort, 'ascend' for ascending order and 'descend' for descending order.
- The `min`, `max`, `sort` function can be made to act on rows instead of columns by transposing the array

Example

So, for example, `sort(A,2, 'descend')` would sort the elements in each row of **A** in descending order

```
>> A=[6 0 3;0 4 0;2 7 0]
```

```
A =
```

```
    6    0    3
```

```
    0    4    0
```

```
    2    7    0
```

```
>> sort(A,2,'descend')
```

```
ans =
```

```
    6    3    0
```

```
    4    0    0
```

```
    7    2    0
```

Element by Element Operations

Multiplying a matrix by a scalar w produces a matrix whose elements are the elements of A multiplied by w

```
>> A=[-2 6; -3 5]
```

```
A =
```

```
    -2     6
```

```
    -3     5
```

```
>> 5*A
```

```
ans =
```

```
   -10    30
```

```
   -15    25
```


Array Addition and Subtraction

For example:

$$\begin{bmatrix} 6 & -2 \\ 10 & 3 \end{bmatrix} + \begin{bmatrix} 9 & 8 \\ -12 & 14 \end{bmatrix} = \begin{bmatrix} 15 & 6 \\ -2 & 17 \end{bmatrix}$$

Array subtraction is performed in a similar way.

The addition shown above is performed in MATLAB as follows:

```
>>A = [6,-2;10,3];  
>>B = [9,8;-12,14]  
>>A+B  
ans =  
    15     6  
    -2    17
```

Element by Element Multiplication:

```
>> A=[11 5;-9 4]
```

```
A =  
    11     5  
    -9     4
```

```
>> B=[-7 8;6 2]
```

```
B =  
    -7     8  
     6     2
```

```
>> C=A.*B
```

```
C =
```

```
-77    40
```

```
-54     8
```

Element by Element division

```
>> C=A./B
```

```
C =
```

```
-1.5714    0.6250
```

```
-1.5000    2.0000
```

Element by Element Exponentiation

■ `>> A=[11 5;-9 4]`

■ `A =`

■ `11 5`

■ `-9 4`

■ `>> B=A.^3`

■ `B =`

■ `1331 125`

■ `-729 64`

MATRIX OPERATIONS

Vector Matrix Multiplication

```
>> A=[2 7;6 -5]
```

```
A =
```

```
2    7
```

```
6   -5
```

```
>> b=[3;9]
```

```
b =
```

```
3
```

```
9
```

```
>> A*b
```

```
ans =
```

```
69
```

```
-27
```

Matrix-Matrix Multiplication

In the product of two matrices **AB**, the number of *columns* in **A** must equal the number of *rows* in **B**.

The product **AB** has the same number of *rows* as **A** and the same number of *columns* as **B**.

Use the operator `*` to perform matrix multiplication in MATLAB. The following MATLAB session shows how to perform the matrix multiplication shown in.

Example

```
>> A=[6 -2;10 3;4 7]
```

```
A =
```

```
     6     -2  
    10      3  
     4      7
```

```
>> B=[9 8;-5 12]
```

```
B =
```

```
     9      8  
    -5     12
```

```
>> A*B
```

```
ans =
```

```
64    24  
75   116  
1    116
```


Matrix multiplication does not have the commutative property; that is, in general, **$\mathbf{AB} \neq \mathbf{BA}$** . A simple example will demonstrate this fact:

$$\mathbf{AB} = \begin{bmatrix} 6 & -2 \\ 10 & 3 \end{bmatrix} \begin{bmatrix} 9 & 8 \\ -12 & 14 \end{bmatrix} = \begin{bmatrix} 78 & 20 \\ 54 & 122 \end{bmatrix}$$

whereas

$$\mathbf{BA} = \begin{bmatrix} 9 & 8 \\ -12 & 14 \end{bmatrix} \begin{bmatrix} 6 & -2 \\ 10 & 3 \end{bmatrix} = \begin{bmatrix} 134 & 6 \\ 68 & 66 \end{bmatrix}$$

Reversing the order of matrix multiplication is a common and easily made mistake.

The following product is defined in matrix multiplication and gives the result shown:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} [y_1 \ y_2 \ y_3] = \begin{bmatrix} x_1 y_1 & x_1 y_2 & x_1 y_3 \\ x_2 y_1 & x_2 y_2 & x_2 y_3 \\ x_3 y_1 & x_3 y_2 & x_3 y_3 \end{bmatrix}$$

The following product is also defined:

Example

Matrix multiplication of a column vector x and a row vector y produces a matrix

```
>> x=[3;4;7]
```

```
x =
```

3

4

7

```
>> y=[5 7 8]
```

```
y =
```

5

7

8

```
>> x*y
```

```
ans =
```

15 21 24

20 28 32

35 49 56

Product of a row vector x by a matrix A produces a row vector

Example

```
>> x=[5 4 2]
```

```
x =
```

```
    5    4    2
```

```
>> A=magic(3)
```

```
A =
```

```
    8    1    6
```

```
    3    5    7
```

```
    4    9    2
```

```
>> x*A
```

```
ans =
```

```
60 43
```

Example

Height versus Velocity

The maximum height h achieved by an object thrown with a speed v at an angle θ to the horizontal, neglecting drag, is

$$h = \frac{v^2 \sin^2 \theta}{2g}$$

Create a table showing the maximum height for the following values of v and θ :

$$v = 10, 12, 14, 16, 18, 20 \text{ m/s} \qquad \theta = 50^\circ, 60^\circ, 70^\circ, 80^\circ$$

The rows in the table should correspond to the speed values, and the columns should correspond to the angles.

```
>> clear
```

```
>> % g acceleration of gravity
```

```
>> g=9.81;
```

```
>> % v is the speed
```

```
>> v=[10:2:20];
```

```
>> % theta is the angle
```

```
>> theta=[50:10:80];
```

```
>> h=( (v'.^2)*(sind(theta).^2) )/(2*g)
```

```
h =
```

2.9909	3.8226	4.5006	4.9432
4.3070	5.5046	6.4809	7.1181
5.8623	7.4924	8.8212	9.6886
7.6568	9.7859	11.5216	12.6545
9.6907	12.3853	14.5820	16.0158
11.9638	15.2905	18.0025	19.7726

```
>> % T is the table
```

```
>> T=[0 theta; v' h]
```

T =

	0	50.0000	60.0000	70.0000	80.0000
10.0000	2.9909	3.8226	4.5006	4.9432	
12.0000	4.3070	5.5046	6.4809	7.1181	
14.0000	5.8623	7.4924	8.8212	9.6886	
16.0000	7.6568	9.7859	11.5216	12.6545	
18.0000	9.6907	12.3853	14.5820	16.0158	
20.0000	11.9638	15.2905	18.0025	19.7726	

Special Matrices

Two exceptions to the noncommutative property are the *null* or *zero* matrix, denoted by **0** and the *identity*, or *unity*, matrix, denoted by **I**.

The null matrix contains all zeros and is not the same as the *empty* matrix [], which has no elements.

These matrices have the following properties:

$$\mathbf{0A} = \mathbf{A0} = \mathbf{0}$$

$$\mathbf{IA} = \mathbf{AI} = \mathbf{A}$$

The **identity matrix** is a square matrix whose diagonal elements are all equal to one, with the remaining elements equal to zero.

For example, the 2×2 identity matrix is

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

The functions `eye(n)` and `eye(size(A))` create an $n \times n$ identity matrix and an identity matrix the same size as the matrix `A`.

Example

```
>> idn=eye(5)
```

```
idn =
```

1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

Sometimes we want to initialize a matrix to have all zero elements. The `zeros` command creates a matrix of all zeros.

Typing `zeros(n)` creates an $n \times n$ matrix of zeros, whereas typing `zeros(m,n)` creates an $m \times n$ matrix of zeros.

Typing `zeros(size(A))` creates a matrix of all zeros having the same dimension as the matrix **A**. This type of matrix can be useful for applications in which we do not know the required dimension ahead of time.

The syntax of the `ones` command is the same, except that it creates arrays filled with ones.

```
>> zr=zeros(3,4)
```

```
zr =
```

0	0	0	0
0	0	0	0
0	0	0	0

```
ne=ones(4,3)
```

```
ne =
```

1	1	1
1	1	1
1	1	1
1	1	1

Matrix Left Division and Linear Algebraic Equations

$$6x + 12y + 4z = 70$$

$$7x - 2y + 3z = 5$$

$$2x + 8y - 9z = 64$$

```
>>A = [6,12,4;7,-2,3;2,8,-9];
```

```
>>b = [70;5;64];
```

```
>>Solution = A\b
```

Solution =

3

5

-2

The solution is $x = 3$, $y = 5$, and $z = -2$.

Inverse of a Matrix

The matrix B is the inverse of the matrix A if, when the two matrices are multiplied, the product is the identity matrix. Both matrices must be square and the multiplication order can be BA or AB .

$$BA = AB = I$$

```
>> A=[2 1 4;4 1 8;2 -1 3]
```

```
A =
```

```
2    1    4
```

```
4    1    8
```

```
2   -1    3
```

continued

```
>> B=inv(A)
```

```
B =
```

5.5000	-3.5000	2.0000
2.0000	-1.0000	0
-3.0000	2.0000	-1.0000

```
>> A*B
```

```
ans =
```

1	0	0
0	1	0
0	0	1