Introduction to matlab

The array is a fundamental form that MATLAB uses to store and manipulate data. An array is a list of numbers arranged in rows and/or columns. The simplest array (one - dimensional) is a row or a column of numbers. A more complex array (two dimensional) is a collection of numbers arranged in rows and columns. One use of arrays is to store information and data.

Reference

1. Amos Gilat, "MATLAB® An Introduction with Applications", John Wiley & Sons, Inc.2011.
2.Stormy Attaway, "Matlab:A Practical Introduction to Programming and ProblemSolving",Elsevier, Inc,2009.
3. Tobin A. Driscoll ,"Learning MATLAB",SIAM,2009.

1.1. Creating Vector and Matrix

Vectors and matrices are used to store sets of values, all of which are the same type. A vector can be either a row vector or a column vector. A matrix can be visualized as a table of values. The dimensions of a matrix are r × c, where r is the number of rows and c is the number of columns. This is pronounced “r by c.” If a vector has n elements, a row vector would have the dimensions 1 × n, and a column vector would have the dimensions n × 1. A scalar (one value) has the dimensions 1 × 1. Therefore, vectors and scalars are actually just subsets of matrices. Here are some diagrams showing, from left to right, a scalar, a column vector, a row vector, and a matrix:

5

3
7
4

5	88	3	11

9	6	3
5	7	2
4	33	8

The scalar is 1 × 1, the column vector is 3 × 1 (3 rows by 1 column), the row vector is 1 × 4 (1 row by 4 columns), and the matrix is 3 × 3. All the values stored in these matrices are stored in what are called elements.

1.1.1 Creating Row and Column Vectors

The vector is created by typing the elements (numbers) inside square brackets [ ].

Row vector: To create a row vector type the elements with a space or a comma between the elements inside the square brackets. For example:

```
>> v = [1  2  3  4]
v =
   1    2    3    4
>> v = [1,2,3,4]
v =
   1    2    3    4
```

Column vector: To create a column vector type the left square bracket [and then enter the elements with a semicolon between them, or press the Enter key after each element. Type the right square bracket] after the last element .For example:

```
>> c = [1; 2; 3; 4]
c =
   1
   2
   3
   4
```

To (**Creating a vector with constant spacing by specifying the first term, the spacing, and the last term**):
In a vector with constant spacing the difference between the elements is the same. For example, in the vector v = 2 4 6 8 10, the spacing between the elements is (2). A vector in which the first term is m, the spacing is q, and the last term is n is created by typing:

```
variable_name = [m:q:n]   or   variable_name = m:q:n
```
(The brackets are optional.)

Some examples are:

```
>> x=[1:2:13]                    First element 1, spacing 2, last element 13.
x =
     1     3     5     7     9    11    13
>> y=[1.5:0.1:2.1]       First element 1.5, spacing 0.1, last element 2.1.
y =
   1.5000   1.6000   1.7000   1.8000   1.9000   2.0000   2.1000

>> z=[-3:7]                      First element –3, last term 7.
                                 If spacing is omitted, the default is 1.
z =
    -3    -2    -1     0     1     2     3     4     5     6
7
>> xa=[21:-3:6]                  First element 21, spacing –3, last term 6.
```

To **(Creating a vector with linear or equal spacing by specifying the first and last terms, and the number of terms):**
A vector with n elements that are linearly (equally) spaced in which the first element is xi and the last element is xf can be created by typing the linspace command (MATLAB determines the correct spacing):

```
variable_name = linspace(xi,xf,n)
```
First element — Last element — Number of elements

When the number of elements is omitted, the default is 100. Some examples are:

```
>> va=linspace(0,8,6)        6 elements, first element 0, last element 8.
va =
      0    1.6000    3.2000    4.8000    6.4000    8.0000
>> vb=linspace(30,10,11)     11 elements, first element 30, last element 10.
vb =
    30    28    26    24    22    20    18    16    14    12    10
>> u=linspace(49.5,0.5)      First element 49.5, last element 0.5.
                             When the number of elements is omitted, the default is 100.
u =
  Columns 1 through 10
   49.5000   49.0051   48.5101   48.0152   47.5202   47.0253
46.5303   46.0354   45.5404   45.0455
.............                100 elements are displayed.
Columns 91 through 100
    4.9545    4.4596    3.9646    3.4697    2.9747    2.4798
1.9848    1.4899    0.9949    0.5000
>>
```

1.1.2 Creating a Two-Dimensional Array (matrix)

A two-dimensional array, also called a matrix, has numbers in rows and columns. A matrix is created by assigning the elements of the matrix to a variable.This is done by typing the elements, row by row, inside square brackets [ ]. First type the left bracket [then type the first row, separating the elements with spaces or commas. To type the next row type a semicolon or press Enter. Type the right bracket] at the end of the last row.

```
variable_name=[1st row elements; 2nd row elements; 3rd
               row elements; ... ; last row elements]
```

For example:

```
>> mat = [4 3 1; 2 5 6]
mat =
   4     3     1
   2     5     6
```

1.2 Commands for Building Arrays and Matrices

A two-dimensional array, also called a matrix, has numbers in rows and columns. Matrices can be used to store information like the arrangement in a table. Matrices play an important role in linear algebra and are used in science and engineering to describe many physical quantities. A matrix is created by assigning the elements of the matrix to a variable. This is done by typing the elements, row by row, inside square brackets [ ]. First type the left bracket [then type the first row, separating the elements with spaces or commas. To type the next row type a semicolon or press Enter. Type the right bracket] at the end of the last row.

1.2.1 The *zeros*, *ones* and, *eye* Commands

The zeros(m,n), ones(m,n), and eye(n) commands can be used to create matrices that have elements with special values. The zeros(m,n) and the ones(m,n) commands create a matrix with m rows and n columns in which all elements are the numbers 0 and 1, respectively. The eye (n) command creates a square matrix with n rows and n columns in which the diagonal elements are equal to 1 and the rest of the elements are 0. This matrix is called the identity matrix. Examples are:

```
>> zr=zeros(3,4)
zr =
     0     0     0     0
     0     0     0     0
     0     0     0     0
>> 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
```

1.2.2 Referring to and Modifying Matrix Elements

To refer to matrix elements, the row and then the column indices are given in parentheses (always the row index first and then the column). For example, this creates a matrix variable mat, and then refers to the value in the second row, third column of mat as examples:

```
>> VCT=[35 46 78 23 5 14 81 3 55]   <-- Define a vector.
VCT =
    35    46    78    23     5    14    81     3    55
>> VCT(4)                           <-- Display the fourth element.
```

```
ans =
    23
>> VCT(6)=273
VCT =
    35    46    78    23     5   273    81     3    55

>> VCT(2)+VCT(8)
ans =
    49
>> VCT(5)^VCT(8)+sqrt(VCT(7))
ans =
   134
>>
>> MAT=[3 11 6 5; 4 7 10 2; 13 9 0 8]
MAT =
     3    11     6     5
     4     7    10     2
    13     9     0     8
>> MAT(3,1)=20
MAT =
     3    11     6     5
     4     7    10     2
    20     9     0     8
>> MAT(2,4)-MAT(1,2)
ans =
    -9
```

Assign a new value to the sixth element.

The whole vector is displayed.

Use the vector elements in mathematical expressions.

Create a 3 × 4 matrix.

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

Use elements in a mathematical expression.

1.2.3 The Transpose Operator

The transpose operator, when applied to a vector, switches a row (column) vector to a column (row) vector. When applied to a matrix, it switches the rows (columns) to columns (rows). The transpose operator is applied by typing a single quote ’ following the variable to be transposed. Examples are:

```
>> aa=[3  8  1]
aa =
     3     8     1
>> bb=aa'
bb =
     3
     8
     1
>> C=[2 55 14 8; 21 5 32 11; 41 64 9 1]
C =
     2    55    14     8
    21     5    32    11
    41    64     9     1
>> D=C'
D =
     2    21    41
    55     5    64
    14    32     9
     8    11     1
>>
```

Define a row vector `aa`.

Define a column vector `bb` as the transpose of vector `aa`.

Define a matrix `C` with 3 rows and 4 columns.

Define a matrix `D` as the transpose of matrix `C`. (`D` has 4 rows and 3 columns.)

1.2.4 Using a Colon: in Addressing Arrays

A colon can be used to address a range of elements in a vector or a matrix.
For a vector:
va(:) Refers to all the elements of the vector va (either a row or a column vector).
va(m:n) Refers to elements m through n of the vector va.

For a matrix:
A(:,n) Refers to the elements in all the rows of column n of the matrix A. ***A(n,:)*** Refers to the elements in all the columns of row n of the matrix A. ***A(:,m:n)*** Refers to the elements in all the rows between columns m and n of the matrix A.
A(m:n,:) Refers to the elements in all the columns between rows m and n of the matrix A.
A(m:n,p:q) Refers to the elements in rows m through n and columns p through q of the matrix A.
The use of the colon symbol in addressing elements of matrices is demonstrated in Tutorial :

```
>> 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]

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)
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 matrix A with 5 rows and 6 columns.

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

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.

1.2.5 Deleting Elements

An element, or a range of elements, of an existing variable can be deleted by reassigning nothing to these elements. This is done by using square brackets with nothing typed in between them. By deleting elements a vector can be made shorter and a matrix can be made to have a smaller size. Examples are:

```
>> kt=[2 8 40 65 3 55 23 15 75 80]
kt =
    2    8   40   65    3   55   23   15   75   80
>> kt(6)=[]
kt =
    2    8   40   65    3   23   15   75   80
>> kt(3:6)=[]
kt =
    2    8   15   75   80
>> mtr=[5 78 4 24 9; 4 0 36 60 12; 56 13 5 89 3]
mtr =
     5    78     4    24     9
     4     0    36    60    12
    56    13     5    89     3
>> mtr(:,2:4)=[]
mtr =
     5     9
     4    12
    56     3
```

Define a vector with 10 elements.

Eliminate the 6th element.

The vector now has 9 elements.

Eliminate elements 3 through 6.

The vector now has 5 elements.

Eliminate all the rows of columns 2 through 4.

1.3 Built-in Functions for Handling Arrays

MATLAB has many built-in functions for managing and handling arrays. Some of these are listed below:

Function	Description	Example

`length(A)`	Returns the number of elements in the vector `A`.	`>> A=[5 9 2 4];`
		`>> length(A)`
		`ans =`
		`4`

`size(A)`	Returns a row vector `[m,n]`, where `m` and `n` are the size $m \times n$ of the array `A`.	`>> A=[6 1 4 0 12; 5 19 6 8 2]`
		`A =`
		`6 1 4 0 12`
		`5 19 6 8 2`
		`>> size(A)`
		`ans =`
		`2 5`

`reshape(A, m,n)`	Creates a `m` by `n` matrix from the elements of matrix `A`. The elements are taken column after column. Matrix `A` must have `m` times `n` elements.	`>> A=[5 1 6; 8 0 2]`
		`A =`
		`5 1 6`
		`8 0 2`
		`>> B = reshape(A,3,2)`
		`B =`
		`5 0`
		`8 6`
		`1 2`

`diag(v)`	When `v` is a vector, creates a square matrix with the elements of `v` in the diagonal.	`>> v=[7 4 2];`
		`>> A=diag(v)`
		`A =`
		`7 0 0`
		`0 4 0`
		`0 0 2`

`diag(A)`	When `A` is a matrix, creates a vector from the diagonal elements of `A`.	`>> A=[1 2 3; 4 5 6; 7 8 9]`
		`A =`
		`1 2 3`
		`4 5 6`
		`7 8 9`
		`>> vec=diag(A)`
		`vec =`
		`1`
		`5`
		`9`

Example:

Using the ones and zeros commands, create a matrix in which the first two rows are 0s and the next two rows are 1s.

```
>> A(1:2,:)=zeros(2,5)
A =
     0     0     0     0     0
     0     0     0     0     0
>> A(3:4,:)=ones(2,5)
A =
     0     0     0     0     0
     0     0     0     0     0
     1     1     1     1     1
     1     1     1     1     1
```

First, create a 2 × 5 matrix with 0s.

Add rows 3 and 4 with 1s.

A different solution to the problem is:

```
>> A=[zeros(2,5);ones(2,5)]
A =
     0     0     0     0     0
     0     0     0     0     0
     1     1     1     1     1
     1     1     1     1     1
```

Create a 4 × 5 matrix from two 2 × 5 matrices.

1.4 Matrix and Array Operations

The arithmetic operators +,-, *, ˆ ,/ ,\ are interpreted in a matrix sense: Examples are:

```
>> VectA=[8 5 4]; VectB=[10 2 7];
>> VectC=VectA+VectB
VectC =
    18     7    11
>> A=[5 -3 8; 9 2 10]
A =
     5    -3     8
     9     2    10
>> B=[10 7 4; -11 15 1]
B =
    10     7     4
   -11    15     1
>> A-B
ans =
     -5   -10     4
     20   -13     9
>> C=A+B
C =
    15     4    12
```

Define two vectors.

Define a vector VectC that is equal to VectA + VectB.

Define two 2 × 3 matrices A and B.

Subtracting matrix B from matrix A.

Define a matrix C that is equal to A + B.

```
>> A = [1 2 3; 4 5 6; 7 8 9];
>> D = [0 0 1; 0 0 1; 0 0 1];
>> A - 3*D
ans =
     1     2     0
     4     5     3
     7     8     6

>> Asq = A^2
Asq =
    30    36    42
    66    81    96
   102   126   150
```

The multiplication operation * is executed by MATLAB according to the rules of linear algebra. This means that if A and B are two matrices, the operation A*B can be carried out only if the number of columns in matrix A is equal to the number of rows in matrix B. The result is a matrix that has the same number of rows as A and the same number of columns as B. for example:

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \\ A_{41} & A_{42} & A_{43} \end{bmatrix} \text{ and } B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \\ B_{31} & B_{32} \end{bmatrix}$$

then the matrix that is obtained with the operation A*B has dimensions with the elements:

$$\begin{bmatrix} (A_{11}B_{11} + A_{12}B_{21} + A_{13}B_{31}) & (A_{11}B_{12} + A_{12}B_{22} + A_{13}B_{32}) \\ (A_{21}B_{11} + A_{22}B_{21} + A_{23}B_{31}) & (A_{21}B_{12} + A_{22}B_{22} + A_{23}B_{32}) \\ (A_{31}B_{11} + A_{32}B_{21} + A_{33}B_{31}) & (A_{31}B_{12} + A_{32}B_{22} + A_{33}B_{32}) \\ (A_{41}B_{11} + A_{42}B_{21} + A_{43}B_{31}) & (A_{41}B_{12} + A_{42}B_{22} + A_{43}B_{32}) \end{bmatrix}$$

A numerical example is:

$$\begin{bmatrix} 1 & 4 & 3 \\ 2 & 6 & 1 \\ 5 & 2 & 8 \end{bmatrix} \begin{bmatrix} 5 & 4 \\ 1 & 3 \\ 2 & 6 \end{bmatrix} = \begin{bmatrix} (1\cdot 5 + 4\cdot 1 + 3\cdot 2) & (1\cdot 4 + 4\cdot 3 + 3\cdot 6) \\ (2\cdot 5 + 6\cdot 1 + 1\cdot 2) & (2\cdot 4 + 6\cdot 3 + 1\cdot 6) \\ (5\cdot 5 + 2\cdot 1 + 8\cdot 2) & (5\cdot 4 + 2\cdot 3 + 8\cdot 6) \end{bmatrix} = \begin{bmatrix} 15 & 34 \\ 18 & 32 \\ 43 & 74 \end{bmatrix}$$

Multiplication of array is demonstrated in Tutorial:

```
>> A=[1 4 2; 5 7 3; 9 1 6; 4 2 8]
A =
     1     4     2
     5     7     3
     9     1     6
     4     2     8
>> B=[6 1; 2 5; 7 3]
B =
     6     1
     2     5
     7     3
>> C=A*B
C =
    28    27
    65    49
    98    32
    84    38
>> D=B*A
??? Error using ==> *
Inner matrix dimensions must agree.

>> F=[1 3; 5 7]
F =
     1     3
     5     7
>> G=[4 2; 1 6]
```

Define a 4 × 3 matrix A.

Define a 3 × 2 matrix B.

Multiply matrix A by matrix B and assign the result to variable C.

Trying to multiply B by A, B*A, gives an error since the number of columns in B is 2 and the number of rows in A is 4.

Define two 2 × 2 matrices F and G.

```
G =
     4     2
     1     6
>> F*G                                   Multiply F*G
ans =
     7    20
    27    52
>> G*F                                   Multiply G*F
ans =                                     Note that the answer for G*F is not the
    14    26                              same as the answer for F*G.
    31    45
>> AV=[2 5 1]                            Define a three-element row vector AV.
AV =
     2     5     1
>> BV=[3; 1; 4]                          Define a three-element column vector BV.
BV =
     3
     1
     4
>> AV*BV                                 Multiply AV by BV. The answer is a scalar.
ans =                                     (Dot product of two vectors.)
    15
```

When an array is multiplied by a number (actually a number is 1*1 a array), each element in the array is multiplied by the number. For example:

```
>> A=[2 5 7 0; 10 1 3 4; 6 2 11 5]      Define a 3 × 4 matrix A.
A =
     2     5     7     0
    10     1     3     4
     6     2    11     5
>> b=3                                   Assign the number 3 to the variable b.
b =
     3
```

Linear algebra rules of array multiplication provide a convenient way for writing a system of linear equations. For example, the system of three equations with three unknowns

$$A_{11}x_1 + A_{12}x_2 + A_{13}x_3 = B_1$$
$$A_{21}x_1 + A_{22}x_2 + A_{23}x_3 = B_2$$
$$A_{31}x_1 + A_{32}x_2 + A_{33}x_3 = B_3$$

Or

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \\ B_3 \end{bmatrix}$$

1.5 Built-in Functions for Analyzing Arrays

MATLAB has many built-in functions for analyzing arrays. Table below lists some of these functions.

Function	Description	Example

`mean(A)`	If `A` is a vector, returns the mean value of the elements of the vector.	`>> A=[5 9 2 4];`
		`>> mean(A)`
		`ans =`
		`5`

`C=max(A)`
 	
`[d,n]=max(A)`
	If `A` is a vector, `C` is the largest element in `A`. If `A` is a matrix, `C` is a row vector containing the largest element of each column of `A`.
	
	If `A` is a vector, `d` is the largest element in `A`, and `n` is the position of the element (the first if several have the max value).
		`>> A=[5 9 2 4 11 6 11 1];`
		`>> C=max(A)`
		`C =`
		`11`
		
		`>> [d,n]=max(A)`
		`d =`
		`11`
		`n =`
		`5`

`min(A)`
 	
`[d,n]=min(A)`
	The same as `max(A)`, but for the smallest element.
	
	The same as `[d,n]= max(A)`, but for the smallest element.
		`>> A=[5 9 2 4];`
		`>> min(A)`
		`ans =`
		`2`

`sum(A)`	If `A` is a vector, returns the sum of the elements of the vector.	`>> A=[5 9 2 4];`
		`>> sum(A)`
		`ans =`
		`20`

`sort(A)`	If `A` is a vector, arranges the elements of the vector in ascending order.	`>> A=[5 9 2 4];`
		`>> sort(A)`
		`ans =`
		`2 4 5 9`

`median(A)`	If `A` is a vector, returns the median value of the elements of the vector.	`>> A=[5 9 2 4];`
		`>> median(A)`
		`ans =`
		`4.5000`

`std(A)`	If `A` is a vector, returns the standard deviation of the elements of the vector.	`>> A=[5 9 2 4];`
		`>> std(A)`
		`ans =`
		`2.9439`

`det(A)`	Returns the determinant of a square matrix `A`.	`>> A=[2 4; 3 5];`
		`>> det(A)`
		`ans =`
		`-2`

`dot(a,b)`	Calculates the scalar (dot) product of two vectors `a` and `b`. The vectors can each be row or column vectors.	`>> a=[1 2 3];`
		`>> b=[3 4 5];`
		`>> dot(a,b)`
		`ans =`
		`26`

`cross(a,b)`	Calculates the cross product of two vectors `a` and `b`, (`axb`). The two vectors must have each three elements.	`>> a=[1 3 2];`
		`>> b=[2 4 1];`
		`>> cross(a,b)`
		`ans =`
		`-5 3 -2`

`inv(A)`	Returns the inverse of a square matrix `A`.	`>> A=[2 -2 1; 3 2 -1; 2 -3 2];`
		`>> inv(A)`
		`ans =`
		`0.2000 0.2000 0`
		`-1.6000 0.4000 1.0000`
		`-2.6000 0.4000 2.0000`