Chapter Six
Array Operations and Linear Equations

1. Array operations

MATLAB has two different types of arithmetic operations:

- matrix arithmetic operations
- array arithmetic operations.

A- Matrix arithmetic operations

As we mentioned earlier, MATLAB allows arithmetic operations: **+**, **-**, *****, **/** and **^** to be carried out on matrices. Thus:

A+B or B+A	**is valid if A and B are of the same size**
A*B	**Is valid if number of column of matrix A equals to number of rows of matrix B**
A^2	**Is valid if A is square matrix and equals A*A**
N *A or A* N	**Multiplies each element of A by number (N)**

B- Array arithmetic operations

the character pairs **(.+)** and **(.-)** are not used.

. *	**Element-by-element multiplication**
. /	**Element-by-element division**
. ^	**Element-by-element exponentiation**

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

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \qquad B = \begin{bmatrix} 10 & 20 & 30 \\ 40 & 50 & 60 \\ 70 & 80 & 90 \end{bmatrix}$$

```
>> C = A. * B

  C =
        10   40   90
       160  250  360
       490  640  810
```

Also we can write this code as bellow:

```
[M,N] = size(A);      % = size(B), as well!
for i = 1:M
      for j = 1:N
            C(i,j) = A(i,j)*B(i,j)
      end
end
```

```
>> A.^2
ans=
     1    4    9
    16   25   36
    49   64   81
```

The relations below summarize the above operations. To simplify, let's consider two vectors **U** and **V** with elements **U =[ui]** and **V = [vj].**

U. * V	**Produces [u_1*v_1 u_2*v_2 ...... u_n*v_n]**
U. / V	**Produces [u_1/v_1 u_2/v_2 ...... u_n/v_n]**
U. ^ V	**Produces [u_1^{v1} u_2^{v2} ...... u_n^{vn}]**

Table 14 : Summary of matrix and array operations

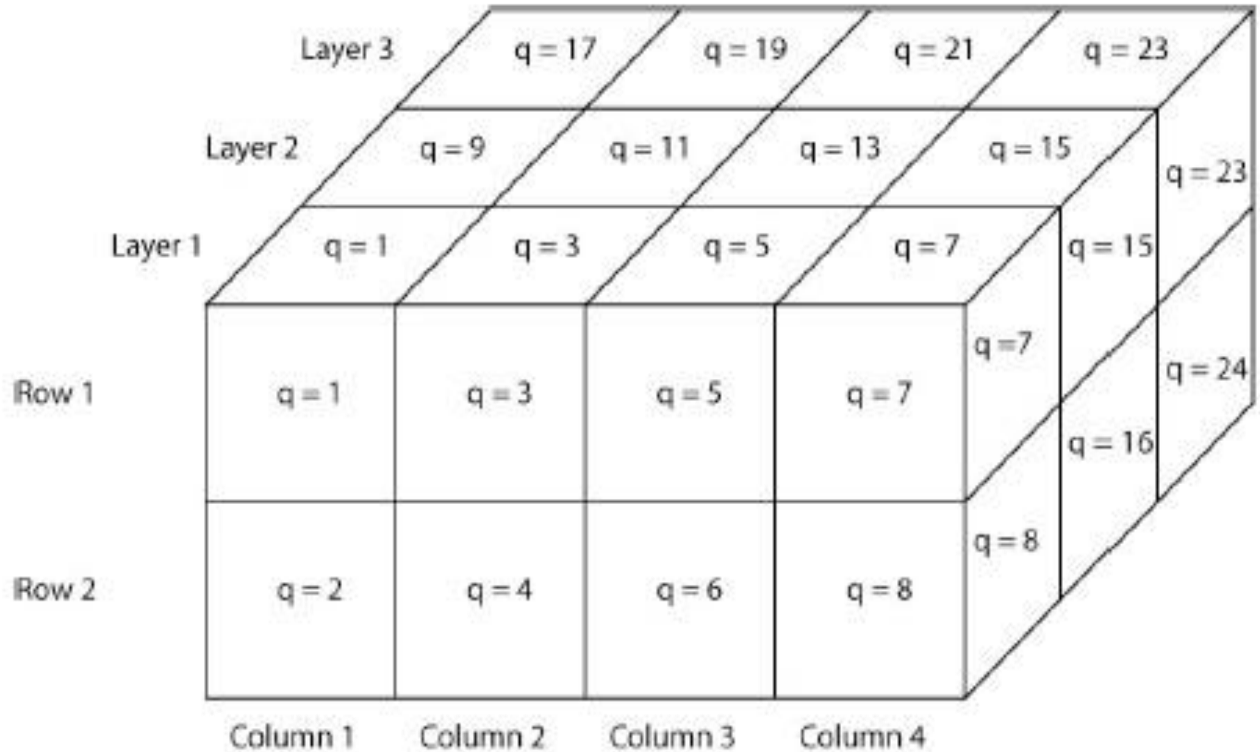
Operation	Matrix	Array
Addition	$+$	$+$
Subtraction	$-$	$-$
Multiplication	$*$	$.*$
Division	/	./
Left division	\	.\
Exponentiation	^	.^

2. Reshaping arrays

1- Create 3D array

Assume **X** is an **i-by-m-by-n** matrix. Where, i represents row, m is represents columns and n represents layers.

E.g.: X is a 2×4×3 matrix



```
>> X= zeros(2,4,3)

X(:,:,1) =

     0     0     0     0
     0     0     0     0
```

Layer 1

```
X(:,:,2) =

     0     0     0     0
     0     0     0     0
```

Layer 2

```
X(:,:,3) =

     0     0     0     0
     0     0     0     0
```

Layer 3

Size of matrix X

```
>> [i,m,n] = size(X)

i =
       2
m =
       4
n =
       3
```

2 rows

4 columns

3 layers

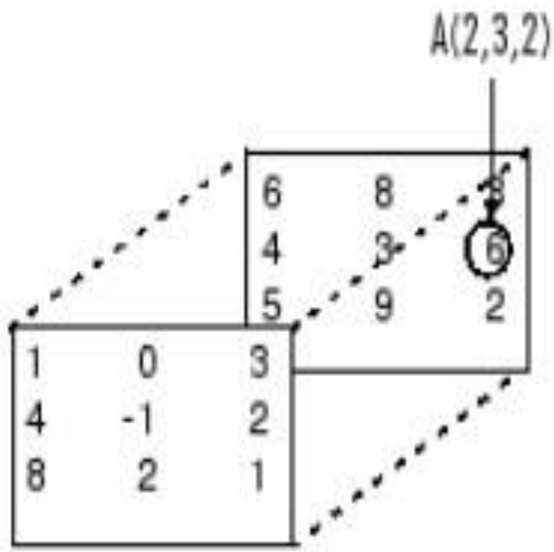
2- Building Multidimensional Arrays with the cat Function

B = cat(dim, A1, A2...)

where, **A1 & A2** and so on are the arrays to concatenate, and dim is the dimension along which to concatenate the arrays.

For example, to create a new array with cat:

```
>> A = cat(3, [1 0 3; 4 -1 2; 8 2 1], [6 8 3; 4 3 6; 5 9 2])
```




```
A(:,:,1) =

     1     0     3
     4    -1     2
     8     2     1


A(:,:,2) =

     6     8     3
     4     3     6
     5     9     2
```

3- Reshaping

B = reshape(A,[s1 s2 s3 ...])

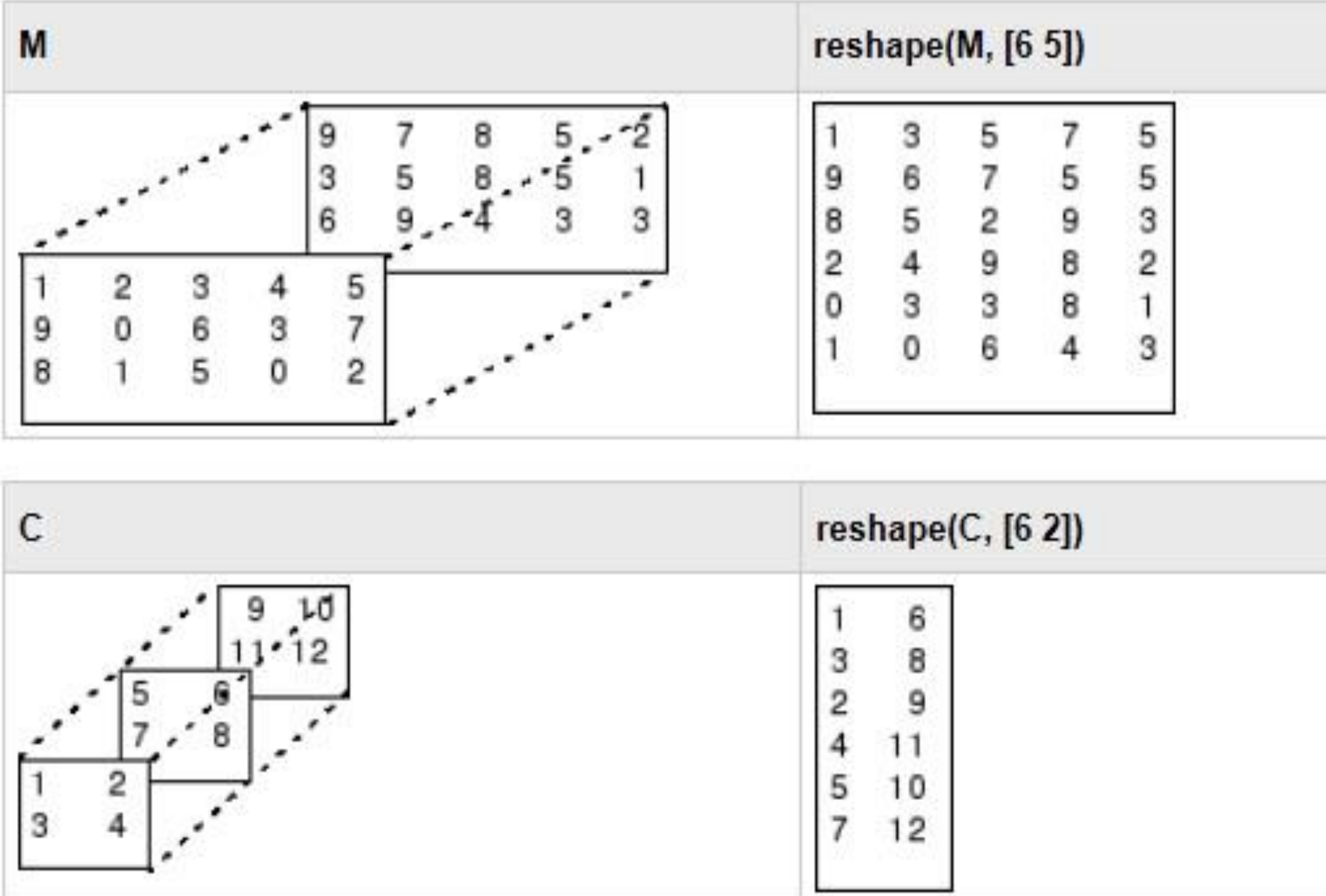
s1, s2, and so on represent the desired size for each dimension of the reshaped matrix.

Note: that a reshaped array must have the same number of elements as the original array (that is, the product of the dimension sizes is constant).

```
>> B = reshape(A,[3 6])

B =

    1    0    3    6    8    3
    4   -1    2    4    3    6
    8    2    1    5    9    2
```

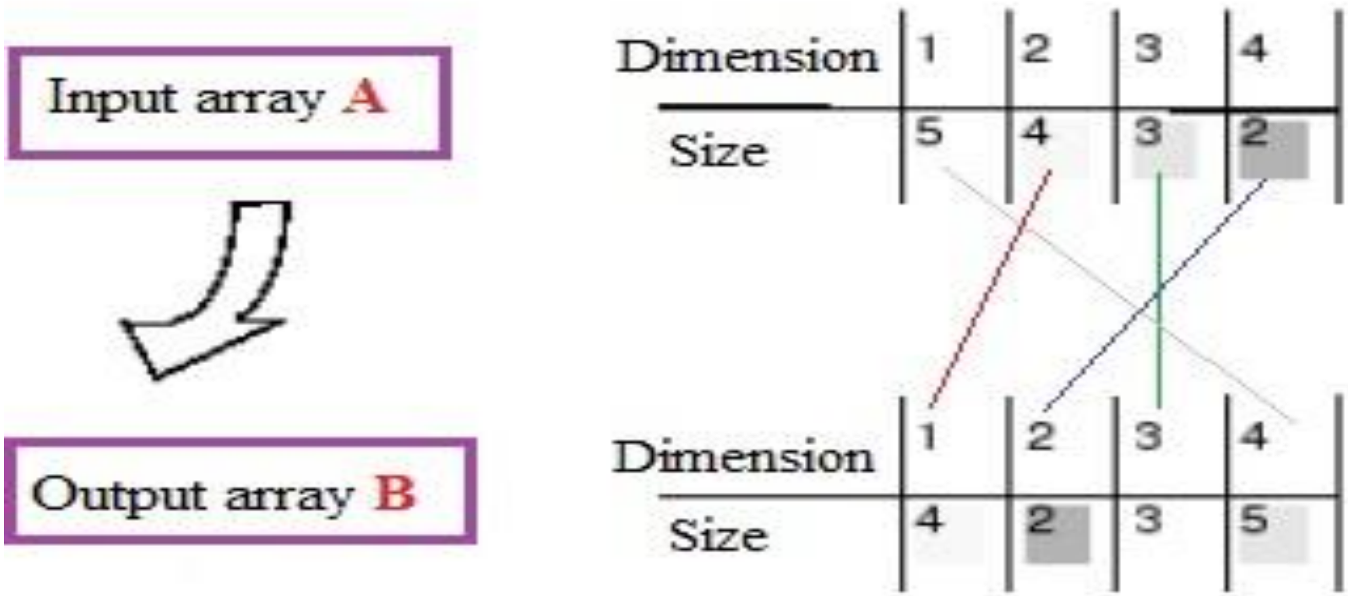


The reshape function operates in a column-wise manner. It creates the reshaped matrix by taking consecutive elements down each column of the original data construct

4- Permuting Array Dimensions

B = permute(A, dims);

```
B = permute(A,[2 4 3 1])
```



A	>> B = permute(A, [2 1 3])	>> B = permute(A, [3 2 1])
A(:,:,1) = 1 0 3 4 -1 2 8 2 1 A(:,:,2) = 6 8 3 4 3 6 5 9 2	B(:,:,1) = 1 4 8 0 -1 2 3 2 1 B(:,:,2) = 6 4 5 8 3 9 3 6 2	B(:,:,1) = 1 0 3 6 8 3 B(:,:,2) = 4 -1 2 4 3 6 B(:,:,3) = 8 2 1 5 9 2

Examples: If you have a matrix A , which is consist of 4 rows, 2 columns and 1 page
A =[5 6 ; 8 2 ; 2 2 ; 1 3]

the order argument of permute function indicates dimensions, are 1 = row, 2 = column and 3 = layer dimensions

```
B = permute(A,[3,2,1]);        % [3,2,1] means [ layer,column,row]
C = permute(A,[3,1,2]);        % [3,1,2] means [layer,row,column]
D = permute(A,[1,3,2]);        % [1,3,2] means [ row, layer,column]
E = permute(A,[2,3,1]);        % [2,3,1] means [ column, layer,row]
F = permute(A,[2,1,3]);        % [2,1,3] means [ column,row, layer]
G = permute(A,[1,2,3]);        % [1,2,3] means [ row,column, layer]
```

matrix	size		
	row	column	layer
original	4	2	1
A,[3,2,1]	1	2	4
A,[3,1,2]	1	4	2
A,[1,3,2]	4	1	2
A,[2,3,1]	2	1	4
A,[2,1,3]	2	4	1
A,[1,2,3]	4	2	1

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

ans(:,:,1) =
    5     6
ans(:,:,2) =
    8     2
ans(:,:,3) =
    2     2
ans(:,:,4) =
    1     3

C = permute(A,[3,1,2])

ans(:,:,1) =
    5     8     2     1
ans(:,:,2) =
    6     2     2     3
```

D = permute(A,[1,3,2])

```
ans(:,:,1) =
    5
    8
    2
    1
ans(:,:,2) =
    6
    2
    2
    3
```

E = permute(A,[2,3,1])

```
ans(:,:,1) =
    5
    6
ans(:,:,2) =
    8
    2
ans(:,:,3) =
    2
    2
ans(:,:,4) =
    1
    3
```

F = permute(A,[2,1,3]) this is transpose and same as [2,1]

```
ans =
    5    8    2    1
    6    2    2    3
```

G = permute(A,[1,2,3]); this makes no difference

```
ans =
 5    6
 8    2
 2    2
 1    3
```

3. Rotating matrices and arrays

To rotate an **m-by-n** matrix **X** to **90°** counterclockwise one may use:

```
Y = rot90(X)
```

There are another may do it like this:

```
Y = X(:,n:-1:1)          % rotate 90 degrees counterclockwise

Y = X(m:-1:1,:)          % rotate 90 degrees clockwise

Y = X(m:-1:1,n:-1:1)     % rotate 180 degrees
```

In the above, one may replace m and n with end.

```
>> y = rot90(x)
y =
    3    5    7    9
    2    4    6    8

>> y = x(:,2:-1:1)
y =
    3    2
    5    4
    7    6
    9    8

>> y = x(4:-1:1,:)
y =
    8    9
    6    7
    4    5
    2    3

>> y = x(4:-1:1,2:-1:1)
y =
    9    8
    7    6
    5    4
    3    2
```

4. Solving linear equations

linear equations is written

$$\mathbf{Ax = b}$$

In linear algebra we learn that the solution to **Ax = b** can be written as $\mathbf{x = A^{-1}b}$, where $\mathbf{A^{-1}}$ is the inverse of **A**.

For example, consider the following system of linear equations

$$\begin{aligned} x + 2y + 3z &= 1 \\ 4x + 5y + 6z &= 1 \\ 7x + 8y \qquad &= 1 \end{aligned}$$

The coefficient matrix **A** is

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix} \quad \text{and the vector} \quad b = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

There are typically three ways to solve for x in MATLAB:

1. **The first one is to use the matrix inverse, inv.**

```
>> A = [1 2 3; 4 5 6; 7 8 0];
>> b = [1; 1; 1];
>> x = inv(A)*b
  x =
      -1.0000
       1.0000
      -0.0000
```

2. The second one is to use the backslash (\)operator.

```
>> A = [1 2 3; 4 5 6; 7 8 0];
>> b = [1; 1; 1];
>> x = A\b
   x =
       -1.0000
        1.0000
       -0.0000
```

3- Using "solve" command

```
syms  x y z
eq1 = 'x + 2*y + 3*z = 1';
eq2 = '4*x + 5*y + 6*z = 1';
eq3 = '7*x + 8*y = 1';
[x,y,z] = solve(eq1, eq2, eq3)
```

Consider the following system of three equations in four unknowns.

$$\begin{aligned} x + 2y + 3z + 2w &= 1 \\ 4x + 5y + 6z + w &= 1 \\ 7x + 8y - w &= 1 \end{aligned}$$

We can solve for x, y, and z in terms of w.

```
syms x y z w
eq1 = 'x + 2*y + 3*z+2*w = 1';
eq2 = '4*x + 5*y + 6*z+w = 1';
eq3 = '7*x + 8*y-w = 1';
[x,y,z] = solve(eq1, eq2, eq3, 'x,y,z')
```

5. Integration and Derivation

1- Integration

Certain functions can be symbolically integrated in MATLAB with the **int** command.

Ex: Find the integration for the equation $\boldsymbol{f = \int x^2 \, dx}$, we need to define x symbolically first.

```
>> syms x
>> int(xˆ2)
 ans =
        x^3/3
```

Ex: Evaluate the integral $f = \int_1^2 x^2\,dx$ **,** In this case, we will use the code int(fun,xmin,xmax). Which, **fun** is the numerically integrates function, from $\mathbf{x_{min}}$ to $\mathbf{x_{max}}$.

```
>> int(x^2,1,2)
 ans =
 
         7/3
```

Mathematical Operation	MATLAB® Command
$\int x^n \ dx$	int(x^n)
$\int_0^{\pi/2} \sin(2x)dx$	int(sin(2*x), 0, pi/2)
g = cos(at + b) $\int g(t)dt$	g = cos(a*t + b); int(g) or int(g, t)

2- Derivation

We can use the **diff** command to find the derivatives.

Ex: find the derivative of x^4

```
>> syms x
>> diff(x^4)
   ans =
          4*x^3
```

Now if we need the **second derivative** of x^4, we use this command:

```
>> syms x
>> diff(x^4,2)
  ans =
        12*x^2
```

Now, suppose we want to evaluate the derivative at **x = 2.1**, Enter the command:

```
>> subs( diff(x^4), x , 2.1 )
   ans =
        37.0440
```

f	diff(f)
syms x n **f = x^n;**	diff(f) ans = n*x^(n - 1)
syms a b t **f = sin(a*t + b);**	diff(f) ans = a*cos(b + a*t)
syms theta **f = exp(i*theta);**	diff(f) ans = exp(theta*i)* i

Mathematical Operator	MATLAB Command
$\frac{df}{dx}$	diff(f) or diff(f, x)
$\frac{df}{da}$	diff(f, a)
$\frac{d^2f}{db^2}$	diff(f, b, 2)

Ex:

```
>> syms s t
>> f = sin(s*t);
>> diff(f,t)
 ans =
          s*cos(s*t)

>> syms x
>> f = sin(x^2);
>> df = diff(f,x)
df =
     2*x*cos(x^2)

>> syms x t
>> diff(sin(x*t^2),t)
 ans =
     2*t*x*cos(t^2*x)

>> syms x y
>> diff(x*cos(x*y), y, 2)
ans =
-x^3*cos(x*y)
```

Mixed Derivatives

Differentiate this expression with respect to the variables **x** and **y**:

>> syms x y

>> diff(x*sin(x*y), x, y)

ans =

2*x*cos(x*y) - x^2*y*sin(x*y)

Derivative of a Matrix in Matlab•

We can use the same technique to find the derivative of a matrix. If we have a matrix **A** having the following values:

$$A = \begin{bmatrix} \cos(4x) & 3x \\ x & \sin(5x) \end{bmatrix}$$

```
>> syms x
>> A = [cos(4*x) 3*x ; x sin(5*x)];
>> diff(A)
 ans =
        [ -4*sin(4*x) ,          3]
        [      1  ,     5*cos(5*x)]
```