Array Operations and Linear Equations

3.1 Array operations

MATLAB has two different types of arithmetic operations: matrix arithmetic operations and array arithmetic operations. We have seen matrix arithmetic operations in the previous lab. Now, we are interested in array operations.

3.1.1 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 `A`'s number of column equals `B`'s number of rows
`A^2`	is valid if `A` is square and equals `A*A`
α`*A` or `A*`α	multiplies each element of `A` by α

3.1.2 Array arithmetic operations

On the other hand, array arithmetic operations or *array operations* for short, are done *element-by-element*. The period character, ., distinguishes the array operations from the matrix operations. However, since the matrix and array operations are the same for addition ($+$) and subtraction ($-$), the character pairs ($.+$) and ($.-$) are not used. The list of array operators is shown below in Table 3.2. If `A` and `B` are two matrices of the same size with elements $\mathbf{A} = [a_{ij}]$ and $\mathbf{B} = [b_{ij}]$, then the command

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

Table 3.1: Array operators

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

produces another matrix `C` of the same size with elements $c_{ij} = a_{ij}b_{ij}$. For example, using the same 3×3 matrices,

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

we have,

```
>> C = A.*B
C   =
    10     40     90
   160    250    360
   490    640    810
```

To raise a scalar to a power, we use for example the command `10^2`. If we want the operation to be applied to each element of a matrix, we use `.^2`. For example, if we want to produce a new matrix whose elements are the square of the elements of the matrix **A**, we enter

```
>> 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 $U = [u_i]$ and $V = [v_j]$.

$U. * V$	produces	$[u_1v_1 \; u_2v_2 \ldots \; u_nv_n]$
$U./V$	produces	$[u_1/v_1 \; u_2/v_2 \ldots \; u_n/v_n]$
$U.\hat{}V$	produces	$[u_1^{v_1} \; u_2^{v_2} \ldots \; u_n^{v_n}]$

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

Table 3.2: Summary of matrix and array operations

3.2 Solving linear equations

One of the problems encountered most frequently in scientific computation is the solution of systems of simultaneous linear equations. With matrix notation, a system of simultaneous linear equations is written

$$Ax = b \tag{3.1}$$

where there are as many equations as unknown. A is a given square matrix of order n, b is a given column vector of n components, and x is an unknown column vector of n components.

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

For example, consider the following system of linear equations

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

The coefficient matrix A is

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad \text{and the vector} \quad b = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

With matrix notation, a system of simultaneous linear equations is written

$$Ax = b \tag{3.2}$$

This equation can be solved for x using linear algebra. The result is $x = A^{-1}b$.

There are typically two 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. The numerical algorithm behind this operator is computationally efficient. This is a numerically reliable way of solving system of linear equations by using a well-known process of Gaussian elimination.

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

This problem is at the heart of many problems in scientific computation. Hence it is important that we know how to solve this type of problem efficiently.

Now, we know how to solve a system of linear equations. In addition to this, we will see some additional details which relate to this particular topic.

3.2.1 Matrix inverse

Let's consider the same matrix A.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$$

Calculating the inverse of A manually is probably not a pleasant work. Here the hand-calculation of A^{-1} gives as a final result:

$$A^{-1} = \frac{1}{9} \begin{bmatrix} -16 & 8 & -1 \\ 14 & -7 & 2 \\ -1 & 2 & -1 \end{bmatrix}$$

In MATLAB, however, it becomes as simple as the following commands:

```
>> A = [1 2 3; 4 5 6; 7 8 0];
>> inv(A)
ans =
    -1.7778    0.8889   -0.1111
     1.5556   -0.7778    0.2222
    -0.1111    0.2222   -0.1111
```

which is similar to:

$$A^{-1} = \frac{1}{9}\begin{bmatrix} -16 & 8 & -1 \\ 14 & -7 & 2 \\ -1 & 2 & -1 \end{bmatrix}$$

and the determinant of A is

```
>> det(A)
ans  =
     27
```

For further details on applied numerical linear algebra, see [10] and [11].

3.2.2 Matrix functions

MATLAB provides many matrix functions for various matrix/vector manipulations; see Table 3.3 for some of these functions. Use the online help of MATLAB to find how to use these functions.

`det`	Determinant
`diag`	Diagonal matrices and diagonals of a matrix
`eig`	Eigenvalues and eigenvectors
`inv`	Matrix inverse
`norm`	Matrix and vector norms
`rank`	Number of linearly independent rows or columns

Table 3.3: Matrix functions