Chapter 3. Vectors and Matrices

1. Operations on vectors and matrices

1.1. Vector and Matrix in Scilab:

A vector can be a row vector or a column vector.

In Scilab, a row vector contains elements in a single row. The elements are separated by commas "**,**", **spaces**, or both, and enclosed in square brackets **[ ]**. For example:

```
--> v = [1,2,3];    // Using commas
--> v = [1,2,3];    // Using spaces
--> v = [1,2 3];    // both comma and space
```

A column vector contains elements in a single column. The elements are separated by semicolons "**;**" and enclosed in square brackets **[ ]**. For example:

```
--> v = [1;2;3];    // Using semicolons
```

Specific vectors can be generated using the **":"** (colon) operator, as shown below:

```
--> x=1:5

 x  =

   1.   2.   3.   4.   5.
```

This assigns the integers from **1** to **5** to vector **x**. You can specify a different increment, for example:

```
--> y = 0:%pi/4:%pi

 y  =

   0.    0.7853982    1.5707963    2.3561945    3.1415927
```

This command always produces a row vector. Negative increments are also possible:

```
--> y = 6:-1: 1

 y  =

   6.    5.    4.    3.    2.    1.
```

You can create tables using these commands, which can later be used for graphical representations. For example:

```
--> x = [0: 0.2:1];
--> y = exp(-x).*sin(x);
--> [x' y']
 ans  =
   0.    0.
   0.2   0.1626567
   0.4   0.2610349
   0.6   0.3098824
   0.8   0.3223289
   1.    0.3095599
```

To specify only the minimum, maximum values, and the number of desired values, use the **linspace** command:

```
k = linspace(-%pi ,%pi ,5)
 k  =

  -3.1415927  -1.5707963   0.   1.5707963   3.1415927
```

A matrix in Scilab is defined as a set of row vectors (resp. column vectors) separated by a semicolons (resp. commas) and enclosed in brackets **[ ]**.

The matrix $\mathbf{A}=\begin{pmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 0\end{pmatrix}$ can be written in Scilab using one of the following syntaxes :

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

The number of elements in each row (column) must be the same across all rows (columns) of the matrix.

An element of a matrix is referenced by its row and column numbers. **A(i,j)** denotes the element in the *i*-th row and *j*-th column of matrix **A**. For example:

```
--> A(2,3)
 ans  =
   6.
--> A(3,3)=A(1,3)+A(3,1)
 A  =
   1.    2.    3.
   4.    5.    6.
   7.    8.    10.
```

1.2. Submatrix Manipulation:

A(i,:) : Extracts row **i**.

A(:,j): Extracts column **j**.

A(:): Gives all elements of **A** as a column vector.

A(i:k,j:l): Extracts the submatrix between rows **i** and **k** and columns **j** and **l**.

A([i k],[j l]): Extracts the submatrix containing rows **i** and **k**, and columns **j** and **l**. For example:

```
--> A(2,:)
 ans  =
   4.   5.   6.
--> A(:,3)
 ans  =
   3.
   6.
   10.
--> A(:)
 ans  =
   1.
   4.
   7.
   2.
   5.
   8.
   3.
   6.
   10.
--> A(1:2,1:3)
 ans  =
   1.   2.   3.
   4.   5.   6.
--> A([1 3],[2 1])
 ans  =
   2.   1.
   8.   7.
```

1.3. Deleting rows or columns:

Deleting rows or columns is equivalent to inserting an empty matrix **[ ]**. For example:

```
--> A([1 2],:)=[]
 A  =

   7.   8.   10.
```

This instruction is equivalent to:

```
--> A=A([3:$],:)
 A  =
   7.    8.    10.
```

The symbol **$** designates the last element. The symbol **$** avoids the need to compute matrix dimensions. Example: Swap the first and last rows of matrix **A,**

```
--> A([1 $],:)=A([$ 1],:)
 A  =

   7.    8.    10.
   4.    5.    6.
   1.    2.    3.
```

1.3. Insert rows/columns:

This instruction inserts a row in the last position:

```
--> A($+1,:)=[11 12 13]
 A  =

   1.    2.    3.
   4.    5.    6.
   7.    8.    10.
   11.   12.   13.
```

And this instruction insert a column at the third position:

```
--> A=[A(:,1:2),[11;12;13],A(:,3)]
 A  =

   1.    2.    11.   3.
   4.    5.    12.   6.
   7.    8.    13.   10.
```

1.4. Basic Operations on Matrices:

Transposition

The special character "’" (prime or apostrophe) indicates the transposition operation. For example:

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

   1.   2.   3.
   4.   5.   6.
   7.   8.   0.

--> B=A'
 B  =

   1.   4.   7.
   2.   5.   8.
   3.   6.   0.

--> X=[-1 0 2]'
 X  =

  -1.
   0.
   2.
```

The transposition operation transposes matrices in the classical sense. If **Z** is a complex matrix, then **Z'** gives the conjugate transpose, which both transposes the matrix and takes the complex conjugate of each element. While the command **Z.'** gives the transpose of the matrix **Z** without changing the sign of the imaginary parts. For example:

```
--> Z = [1 + 2*%i,3 - %i; -%i,4 + 3*%i]
 Z  =
   1. + 2.i   3. - i
  -i          4. + 3.i

--> Z_conjugate_transpose =Z'
 Z_conjugate_transpose  =
                           1. - 2.i   i
                           3. + i     4. - 3.i
--> Z_transpose = Z.'
 Z_transpose  =
                  1. + 2.i  -i
                  3. - i     4. + 3.i
```

Addition and Subtraction:

The operators "**+**" and "-" act on matrices and vectors. These operations are valid as long as the dimensions of the matrices or vectors are the same. For example, with the matrices from the previous example, the addition:

```
--> A=[1 2 3; 4 5 6; 7 8 0]
 A  =
   1.   2.   3.
   4.   5.   6.
   7.   8.   0.
--> X=[-1 0 2]'
 X  =
  -1.
   0.
   2.
--> A+X
Inconsistent row/column dimensions.
```

produces an "Inconsistent row/column dimensions" error because **A** is **3×3** and **X** is **3×1**. However, the following operation is valid:

```
--> A
 A  =
   1.   2.   3.
   4.   5.   6.
   7.   8.   0.
```

```
--> B
 B  =
   1.   4.   7.
   2.   5.   8.
   3.   6.   0.
--> C=A+B
 C  =
   2.    6.    10.
   6.    10.   14.
   10.   14.   0.
```

Addition and subtraction are also defined if one of the operands is a scalar, that is, a **1×1** matrix. In this case, the scalar is added to or subtracted from all elements of the other operand. For example:

```
--> z=X-1
 z  =
  -2.
  -1.
   1.
```

Multiplication:

The symbol "*" represents the matrix multiplication operator. This operation is valid as long as the dimensions of the operands are compatible, meaning that the number of columns of the left operand must equal the number of rows of the right operand. For example, the following operation is not valid:

```
--> X*z
Inconsistent row/column dimensions.
```

produces an "Inconsistent row/column dimensions" error. However, the following command:

```
--> X'*z
 ans  =
   4.
```

gives the dot product of **X** and **z**. Another valid command is:

```
--> b=A*X
 b  =
   5.
   8.
  -7.
```

Multiplying a matrix by a scalar is, of course, always valid:

```
--> A*2
 ans  =
   2.    4.    6.
   8.    10.   12.
   14.   16.   0.
```

Matrix inversion and division:

The inverse of a square matrix can be easily obtained with the **inv** command. For example:

```
--> B=inv(A)
 B  =
  -1.7777778   0.8888889  -0.1111111
   1.5555556  -0.7777778   0.2222222
  -0.1111111   0.2222222  -0.1111111
```

```
--> C=B*A
 C  =
   1.           4.441D-16    0.
  -2.220D-16    1.           0.
   0.           0.           1.
```

Note the minor errors due to finite calculation precision (matrix **C** should ideally be the identity matrix).

Matrix *division* is implemented in Scilab with the following meaning: the expression **A/B** yields the result of the operation $\mathbf{AB}^{-1}$. This is equivalent to the command **A*inv(B)**. For left division, the expression **A\B** yields the result of the operation $\mathbf{A}^{-1}\mathbf{B}$. This is equivalent to the command **inv(A)*B**. Dimension compatibility between the two matrices must be respected for this division to be meaningful.

Left division is commonly used when solving a linear system. For example, to solve the linear system **Ay=X**, with the equations:

$$\begin{cases} \mathbf{y}_1 + 2\mathbf{y}_2 + 3\mathbf{y}_3 = -1 \\ 4\mathbf{y}_1 + 5\mathbf{y}_2 + 6\mathbf{y}_3 = 0 \\ 7\mathbf{y}_1 + 8\mathbf{y}_2 = 2 \end{cases}$$

in Scilab, we write:

```
--> y=A\X
 y  =
   1.5555556
  -1.1111111
  -0.1111111
```

When Scilab interprets this expression, it does not invert matrix A before multiplying it by **X** on the right. Instead, it effectively solves the system of equations using Gaussian elimination, which is approximately three times faster than if we had written:

```
--> y=inv(A)*X

 y  =

   1.5555556

  -1.1111111

  -0.1111111
```

because in this case, we first need to calculate the inverse of the matrix (solving 3 linear systems) and then perform a matrix-vector multiplication. The accuracy of the results can be verified as follows:

```
--> A*y-X

 ans  =

   0.

  -2.220D-16

  -1.776D-15
```

As in the previous example, note the presence of errors on the order of machine precision.

Element-wise Operations:

The usual matrix operations can be performed element by element. For addition and subtraction, both viewpoints are the same since these two operations already act element by element on matrices.

The symbol "**.***" represents element-wise multiplication. If **A** and **B** have the same dimensions, then **A.*B** represents the array whose elements are simply the products of the individual elements of **A** and **B**. For example, if we define **x** and **y** as follows:

```
--> x=[1 2 3]
 x  =
   1.   2.   3.
--> y=[4 5 6]
 y  =
   4.   5.   6.
```

then the command:

```
--> z=x.*y
```

produces the result:

```
 z  =

   4.   10.   18.
```

Division works in the same way with "**./**":

```
--> z=x./y
 z  =
   0.25   0.4   0.5
```

The symbol "**.^**" represents element-wise exponentiation:

```
--> x.^y
 ans  =
   1.   32.   729.
--> x.^2
 ans  =
   1.   4.   9.
--> 2.^x
 ans  =
   2.   4.   8.
```

Note that for "**.^**", one of the operands can be a scalar.

Relational Operators:

Six relational operators are available to compare two matrices of equal dimensions:

< less than

<= less than or equal to

> greater than

>= greater than or equal to

== equal to

<> not equal

Scilab compares corresponding pairs of elements; the result is a matrix of boolean constants, with **F** (false) representing **false** and **T** (true) representing **true**. For example:

```
--> 2+2<>4
 ans  =
  F
```

Logical operators allow you to view the layout of elements in a matrix that satisfy certain conditions. For example, take the matrix:

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

The command:

```
--> P=(A<0)

P  =

 F  T  F

 T  T  F

 F  F  T
```

returns a matrix **P** with **T** indicating the negative elements of **A**.

Logical Operators:

The operators "**&**", "**|**", and "**~**" represent the logical operators **and**, **or**, and **not**, respectively. They are used to create logical expressions. For example, with the matrix from the previous example, the command:

```
--> P = (A < 0) & (modulo(A,2) == 0)
 P  =
  F  F  F
  T  T  F
  F  F  F
```

identifies the elements in **A** that are both negative and multiples of **2**.

2. Basic mathematical functions for matrix processing:

Function	Meaning
`ones(n), ones(n,m)`	Generates an n×n or n×m matrix with all elements equal to 1.
`zeros(n), zeros(n,m)`	Generates an n×n or n×m matrix with all elements equal to 0.
`eye(n)`	Generates an n×n identity matrix.

size	Calculates the number of rows and columns of a matrix.
Det	Calculates the determinant of a matrix.
inv	Calculates the inverse of a matrix.
rank	Calculates the rank of a matrix**.**
trace	Calculates the trace of a matrix**.**
spec	Calculates the eigenvalues**.**
norm	Calculates the norm**.**
diag(V)	Creates a matrix with vector V as the diagonal and 0 elsewhere.

Examples:

```
--> A=zeros(3,2)
 A  =

   0.   0.
   0.   0.
   0.   0.

--> B=ones(3,2)
 B  =

   1.   1.
   1.   1.
   1.   1.

--> I = eye(3,3)
 I  =

   1.   0.   0.
   0.   1.   0.
   0.   0.   1.

--> A=[1 2 3; 4 5 6; 7 8 9]; size(A)
 ans  =

   3.   3.
```

```
--> v=[3 5 1 2 4]; [n,m]=size(v)
 m  =

   5.

 n  =

   1.

--> det(A)
 ans  =

   6.661D-16

--> inv(A)
 ans  =

  -4.504D+15   9.007D+15  -4.504D+15
   9.007D+15  -1.801D+16   9.007D+15
  -4.504D+15   9.007D+15  -4.504D+15

--> rank(A)
 ans  =

   2.

--> trace(A)
 ans  =

   15.

--> spec(A)
 ans  =

   16.116844
  -1.116844
  -1.304D-15

--> norm(A)
 ans  =

   16.848103

--> diag(A)
 ans  =

   1.
   5.
   9.
```