9

Matrices

Matrices are the most important and most frequently used objects in Scilab. This type is used to portray tables with one or two dimensions in a unified way.

9.1. Creating and modifying

There are several ways of creating tables in Scilab. The most simple is based on the concatenation operator `[]`. To create a matrix, enter a list of values, row by row, in between brackets while obeying the following syntax rules:

- Values on the same row are separated by spaces or a comma (`,`).
- Rows are separated by semicolons (`;`).

The console commands can be entered over several lines (as long as you don't forget to close the matrix correctly).

Caution *› Contrary to other languages, the cell/row/column numbering in Scilab starts at 1 (rather than at 0).*

The command `size` returns the size of a matrix (number of rows and columns). The result of this command is a matrix with one row and two columns.

```
-->A=[1 2 3; 4 5 6; 7 8 9] // matrix 3 lines and 3 columns
 A  =
    1.    2.    3.
    4.    5.    6.
    7.    8.    9.

-->typeof(A) // same type as real numbers
 ans  =
 constant

-->size(A) // size of A
 ans  =
    3.    3.

-->B=[10,11,12;15 14 13] // matrix with 2 rows and 3 columns
 B  =
    10.    11.    12.
```

```
    15.    14.    13.

-->size(B) // size of B
 ans  =
    2.    3.

-->// enter over multiple lines

-->[1 2 3;
-->3 4 5]
 ans  =
    1.    2.    3.
    3.    4.    5.
```

Tip *› The [ ] command produces an empty matrix, with zero rows and zero columns. A number (real, complex, integer) is considered a matrix with one line and one column.*

Matrices can later be concatenated to create larger ones, by using `[ ]`, as long as they have the right dimensions.

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

-->B=[10,11,12;15 14 13];

-->C=[A;B]      // A above B
 C  =
    1.     2.     3.
    4.     5.     6.
    7.     8.     9.
    10.    11.    12.
    15.    14.    13.

-->D=[B;A]      // B above A
 D  =
    10.    11.    12.
    15.    14.    13.
    1.     2.     3.
    4.     5.     6.
    7.     8.     9.

-->E=[C,D]      // C left of D
 E  =
    1.     2.     3.     10.    11.    12.
    4.     5.     6.     15.    14.    13.
    7.     8.     9.     1.     2.     3.
    10.    11.    12.    4.     5.     6.
    15.    14.    13.    7.     8.     9.

-->F=[D,C]      // D left of C
 F  =
    10.    11.    12.    1.     2.     3.
    15.    14.    13.    4.     5.     6.
    1.     2.     3.     7.     8.     9.
    4.     5.     6.     10.    11.    12.
```

```
    7.     8.     9.     15.    14.    13.

-->G=[A,B]      // A left of B -> error

    !--error 5

 Inconsistent column/row dimensions.
```

You can also concatenate matrices with the `cat` command.

```
-->A=[1 2 3; 4 5 6; 7 8 9]    // 3x3 matrix (3 rows, 3 columns)
 A  =
    1.    2.    3.
    4.    5.    6.
    7.    8.    9.

-->B=[10,11,12;15 14 13]    // 2x3 matrix (2 rows, 3 columns)
 B  =
    10.    11.    12.
    15.    14.    13.

-->// concatenating matrices

-->C=[A;B]    // A above B
 C  =
    1.     2.     3.
    4.     5.     6.
    7.     8.     9.
    10.    11.    12.
    15.    14.    13.

-->cat(1,A,B)    // =C
 ans  =
    1.     2.     3.
    4.     5.     6.
    7.     8.     9.
    10.    11.    12.
    15.    14.    13.

-->D=[B;A]    // B above A
 D  =
    10.    11.    12.
    15.    14.    13.
    1.     2.     3.
    4.     5.     6.
    7.     8.     9.

-->cat(1,B,A)    // =D
 ans  =
    10.    11.    12.
    15.    14.    13.
    1.     2.     3.
    4.     5.     6.
    7.     8.     9.
```

```
-->E=[C,D]    // C left of D
 E  =
    1.     2.     3.     10.    11.    12.
    4.     5.     6.     15.    14.    13.
    7.     8.     9.     1.     2.     3.
    10.    11.    12.    4.     5.     6.
    15.    14.    13.    7.     8.     9.

-->cat(2,C,D)    // =E
 ans  =
    1.     2.     3.     10.    11.    12.
    4.     5.     6.     15.    14.    13.
    7.     8.     9.     1.     2.     3.
    10.    11.    12.    4.     5.     6.
    15.    14.    13.    7.     8.     9.
```

Caution *› The matrices you wish to concatenate need to have compatible dimension, otherwise you will get an error message (5 or 6) stating* Inconsistent row/column dimensions.

You can generate certain special matrices automatically by providing their size as input to the following functions:

- `zeros` produces a zero matrix and `ones` yields a matrix full of 1s.
- `eye` creates the identity matrix (with 1s on the main diagonal and 0s elsewhere).
- `rand` fills a matrix with pseudorandom numbers uniformly distributed in the interval [0;1[(also see `grand` which is used to generate random numbers according to different distributions as described online `help grand`).

```
-->O=zeros(2,3)    // zero matrix
 O  =
    0.    0.    0.
    0.    0.    0.

-->U=ones(4,3)    // matrix of ones
 U  =
    1.    1.    1.
    1.    1.    1.
    1.    1.    1.
    1.    1.    1.

-->Id=eye(3,3)    // identity matrix
 Id  =
    1.    0.    0.
    0.    1.    0.
    0.    0.    1.

-->M=rand(2,2)    // random numbers
 M  =
    0.537622980    0.225630349
    0.119992550    0.627409308
```

Tip › *When the functions* `zeros`, `ones`, `eye` *and* `rand` *take a matrix as argument, they generate a matrix of the same size as the input matrix. More specifically, if the input is a number, these functions create a 1×1 matrix (i.e. just a number!).*

```
-->A=rand(2,2)
 A  =
    0.280649802    0.778312860
    0.128005846    0.211903045

-->// creates a zero matrix of the same size as A

-->zeros(A)
 ans  =
    0.    0.
    0.    0.

-->// zeros(2) does not create a vector with two entries

-->zeros(2)
 ans  =
    0.
```

To retrieve the different values stored inside a matrix, specify the row and column of the entry of interest in between parentheses `()`. To modify a matrix value, use the `=` to assign a new value to the entry.

```
-->A=[1 2 3; 4 5 6; 7 8 9]    // 3x3 matrix (3 rows, 3 columns)
 A  =
    1.    2.    3.
    4.    5.    6.
    7.    8.    9.

-->A(2,3)    // value stored at row 2, column 3
 ans  =
    6.

-->A(2,3)=-1    // modify the value at row 2 column 3
 A  =
    1.    2.    3.
    4.    5.  - 1.
    7.    8.    9.

-->A(4,5)=10    // this assignment increases the size of A
 A  =
    1.    2.    3.    0.    0.
    4.    5.  - 1.    0.    0.
    7.    8.    9.    0.    0.
    0.    0.    0.    0.    10.

-->    // entry that doesn't exist in A

-->A(10,10)    // this call returns error 21
```

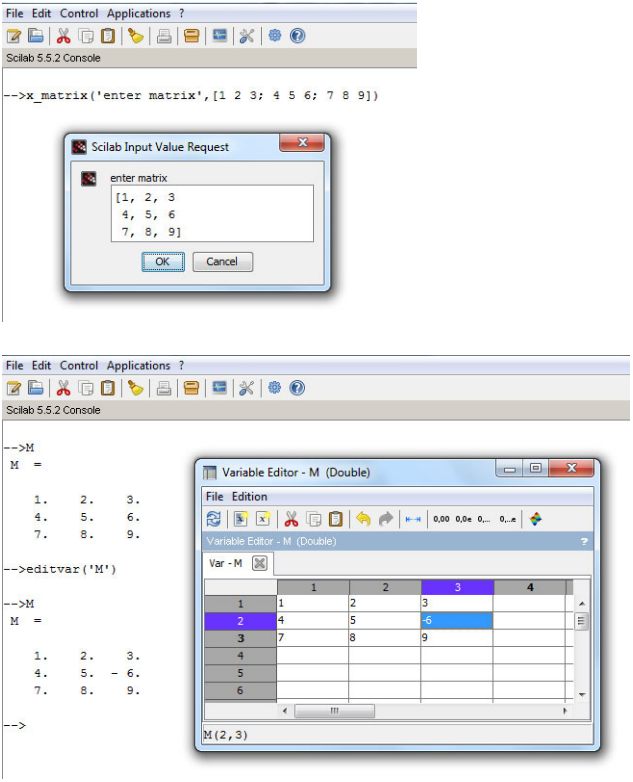
```
      !--error 21

   Invalid index.
```

Tip *› If you assign a value to a matrix entry that does not exist (the row/column number is greater than the matrix size), Scilab automatically increases the size of the matrix to assign the new value. It also incidentally fills the other entries that were created in the process with 0s.*

You can also modify matrix entries via a graphical interface with the function `x_matrix` (see Figure 9.1) or by using the variable editor with the command `editvar` (see Figure 8.1 or Figure 9.1).

Figure 9.1 : *Graphical interface to modify matrix entries*



Caution *› If you try and access a matrix value with an index that exceeds the matrix size, Scilab returns an execution error which states* Invalid index. *This is the case when calling the entry/row/column number 0 (or indexing with negative or non-integer numbers).*