MATLAB

Chapter II : Vectors and matrices

Matlab was originally designed to allow mathematicians, scientists and engineers to easily use the mechanisms of linear algebra. Therefore, the use of vectors and matrices is very intuitive and convenient in Matlab.

1. The vectors:

A vector is an ordered list of elements. If the elements are arranged horizontally we say that the vector is a vector row, on the other hand if the elements are arranged vertically we say that it is a vector column.

To create a vector linejust write the list of its components in square brackets **[**and**]** and separated by spaces or commas as follows:

```
>> V = [5, 2, 13,-6]                % Create a vector line V
      V =
         5     2    13    -6

>> U = [4 -2 1]                     % % Create a vector line U
      U =
         4    -2     1
```

To create a colonneil vector is possible to use one of the following methods :

1. write the components of the vector in square brackets [and] and semicolons separated (**;**)as follows:

```
>> U = [4;-2;1]                     % Creating a column vector U
      U =
             4
            -2
             1
```

2. write vector vertically :

```
>> U = [
4
-2
1
]
      U =

           4
           -2
           1
```

3. calculate the transpose of a line vector:

```
>> U = [4-21]'                          % Creating a column vector U
      U =
             4
            -2
             1
```

If the components of an **X**sont vector ordered with consecutive values, we can note it with the following notation :

X = first_element : last element *(Square brackets are optional in this case)*

for exemple :

```
>> X=1:8                                % you can also write colon (1,8)
      X =
            1     2     3     4     5     6     7     8
>> X = [1:8]
      X =
            1     2     3     4     5     6     7     8
```

If the components of an **X**sont vector ordered with consecutive values but with a pitch (increment/decrement) different from 1, we can specify the pitch with the notation :

X = first_element : step: last element *(Brackets are optional)*

for exemple :

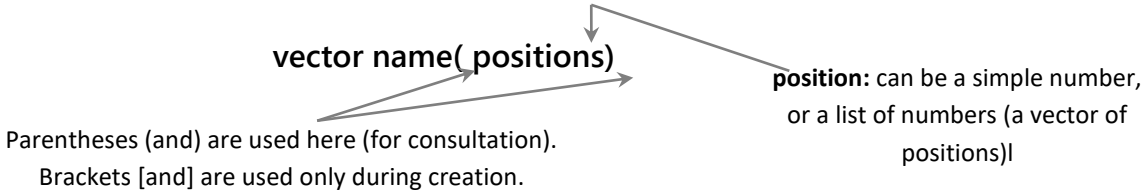
```
>> X = [0:2:10]                 % vector X contains even numbers < 12
      X =
            0     2     4     6     8    10
>> X= [-4:2:6]                  % on can also write colon (-4,2,6)
      X =
           -4    -2     0     2     4     6
>> X=0:0.2:1              % you can also write colon (0,0.2,1)
      X =
      0    0.2000    0.4000    0.6000    0.8000    1.0000
```

You can write more complex expressions like :

```
>> V = [1:2:5,-2:2:1]
      V =
            1     3     5    -2     0
>> A = [1 2 3]
      A =
            1     2     3
>> B = [A, 4, 5, 6]
      B =
            1     2     3     4     5     6
```

1.1 Referencing and access to vector elements :

The elements of a vector are accessed using the following general syntax:



Exemples :

```
>> V=[5, -1, 13,-6, 7]          % creation of vector V which contains 5 elements
V =
5    -1    13    -6     7

>> V(3)                         % the 3rd position
       ans =
              13

>>V(2:4)                        % from second to fourth position
       ans =
              -1    13    -6

>>V(4:-2:1)                     % from the 4th pos to the 1st with the pitch = -2
       ans =
              -6    -1

>>V(3:end)                      % from the 3rd position to the last
       ans =
              13    -6     7

>>V([1,3,4])                    % 1st, 3rd and 4th position only
       ans =
              5    13    -6

>> V(1)=8                       % set the first element to 8
V =
              8    -1    13    -6     7

>> V(6)=-3                      % add a sixth element with the value -3
       V =
              8    -1    13    -6     7    -3

>> V(9)=5                       % add a ninth element with value 5
       V =
              8    -1    13    -6     7    -3     0     0     5


>> V(2)=[]                      % Remove second item
```

```
      V =
            8    13    -6     7    -3     0     0     5
```

```
>>V(3:5)=[]                % Delete from 3rd to 5th element
      V =
            8    13     0     0     5
```

1.2 Element-by-element operations for vectors :

With two vectors $\vec{u}$ and $\vec{v}$, it is possible to perform element-by-element calculations using the following operations :

operation	meaning	Exemple with : >> u= [-2,6,1] ; >> v= [3,-1, 4] ;
+	Addition of vectors	**>> u+2** ans = 0 8 3 **>>u+v** ans = 1 5 5
-	Vector subtraction	**>> u-2** ans = -4 4 -1 **>> u-v** ans = -5 7 -3
.*	Element-by-element multiplication	**>> u*2** ans = -4 12 2 **>> u.*2** ans = -4 12 2 **>> u.*v** ans = -6 -6 4
./	Division element by element	**>> u/2** ans = -1.0000 3.0000 0.5000 **>> u./2** ans = -1.0000 3.0000 0.5000 **>> u./v** ans = -0.6667 -6.0000 0.2500
.^	Power item by item	**>> u.^2** ans = 4 36 1 **>>u.^v** ans = -8.0000 0.1667 1.0000

Writing an expression such as: ***u^2*** generates an erreurcar this expression refers to a matrix multiplication (***u*u*** must be rewritten ***u*u'*** or ***u'*u*** to be valid).

1.3 The linspace function :

The creation of a vector whose components are ordered by intervallerégulier and with a well determined number of elements can be realized with the function:

linspace(start, end, number of elements).

The increment step is calculated automatically by Matlab according to the formula :

$$\text{step} = \frac{end - start}{number\ of\ elements - 1}$$

for exemple :

```
>> X=linspace(1,10,4)            % a vector of four elements from 1 to 10
      X =
            1     4     7    10

>> Y = linspace(13,40,4)          % a vector of four elements of 13 to 40
      Y =
            13    22    31    40
```

The size of a vector (the number of its components) can be obtained with the lengthfunction as follows :

```
>>length(X)          % the size of vector X
ans =
            4
```

2. The matrices:

A matrix is a rectangular array of (two-dimensional) elements. Vectors are matrices with a single row or column (monodimensional).

To insert a matrix, follow these rules :

- Items should be bracketed **[** and**]**
- Spaces or commas are used to separate items in the same row
- Comma (or enter) is used to separate lines

To illustrate this, considering the following matrix :

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$$

This matrix can be written in Matlab with the following parameters :

```
>> A=[1,2,3,4;5,6,7,8;9,10,11,12] ;

>> A=[1 2 3 4;5 6 7 8;9 10 11 12] ;

>> A = [1,2,3,4
5,6,7,8
9,10,11,12] ;
>> A=[[1;5;9] , [2;6;10] , [3;7;11] , [4;8;12]] ;
```

The number of elements in each row (number of columns) must be the same in all rows in the matrix, otherwise an error will be reported by Matlab. For example :

```
>> X=[1 2 ; 4 5 6]
       Errorusingvertcat
       CAT arguments dimensions are not consistent.
```

A matrix can be generated by vectors as shown in the following examples:

```
>>x = 1:4                                % vector creation x
       x =
             1     2     3     4
>> y = 5:5:20                            % vector creation y
       y =
             5    10    15    20
>> z = 4:4:16                            % vector creation z
       z =
             4     8    12    16

>> A = [x ; y ; z]                        % A is formed by line vectors x, y and z
       A =
             1     2     3     4
             5    10    15    20
             4     8    12    16

>> B = [x' y' z']                         % Best formed by column vectors x, y and z
       B =
             1     5     4
             2    10     8
             3    15    12
             4    20    16

>>C = [x ; x]                            % It is formed by the same vector x2 times
       C =
             1     2     3     4
             1     2     3     4
```

2.1 Referencing and access to matrix elements :

The elements of a matrix are accessed using the following general syntax :

nom_matrice(positions_lignes , positions_colonnes)

Parentheses (and) are used here (for consultation).
Brackets [and] are used only during creation.

positions: can be a simple number, or a list of numbers (a vector of positions)

It is worth noting the following possibilities :

- Access to an element in row **i** and column **j** is done by : ***A(i,j)***
- Access to the whole number **i** line is via: ***A(i,:)***
- Access to the entire column number **j** is by: ***A(:,j)***

Exemples :

```
>> A = [1,2,3,4 ; 5,6,7,8 ; 9,10,11,12]      % creation of matrix A
      A =
            1     2     3     4
            5     6     7     8
            9    10    11    12

>> A(2,3)                          % element on the 2nd row in the 3rd column
      ans =
            7

>>A(1,:)                           % all 1st line itemsans =
            1     2     3     4

>>A(:,2)                           % all items in 2nd column
      ans =
            2
            6
            10

>>A(2:3,:)                         % all elements of the 2nd and 3rd line
      ans =
            5     6     7     8
            9    10    11    12

>>A(1:2,3:4)              % The upper right sub matrix of size 2x2
ans =
            3     4
            7     8

>>A([1,3],[2,4])                   % la sub-matrix: rows(1,3) and columns (2,4)
      ans =
            2     4
            10    12

>>A(:,3)=[]                % Delete the third column
      A =
            1     2     4
            5     6     8
            9    10    12

>>A(2,:)=[]                % Delete the second line
      A =
            1     2     4
            9    10    12
```

```
>> A=[A,[0;0]]                    % add a new column {or A(:,4)=[0;0]}
      A =
            1     2     4     0
            9    10    12     0

>> A=[A;[1,1,1,1]]          % Add a new line {or A(3,:)=[1,1,1,1]}
      A =
            1     2     4     0
            9    10    12     0
            1     1     1     1
```

The dimensions of a matrix can be acquired using the **size** function. However, with a matrix **A** of dimension m nle result of this function is a vector of two components, one for puts the other for **n**.

```
>> d = size(A)
      d =
            3     4
```

Here, the variable **d** contains the dimensions of the matrix **A** as a vector. To obtain the separechin dimensions can use the syntax :

```
>> d1=size (A, 1)                % d1 contains the number of rows (m)
      d1 =
            3

>> d2=size (A, 2)                % d2 contient le nombre de colonne (n)
d2 =
            4
```

2.2 Automatic generation of matrices

In Matlab, there are functions that automatically generate particular matrices. In the following table we have the most used :

La fonction	Signification
zeros(n)	Generates an n× n matrix with all elements = 0
zeros(m,n)	Generates a matrix m ×n with all elements = 0
ones(n)	Generates an n ×n matrix with all elements = 1
ones(m,n)	Generates a matrix m× n with all elements = 1
eye(n)	Generates a dimension identity matrix $n \times n$
magic(n)	Generates a dimension magic matrix $n \times n$
rand(m,n)	Generates a matrix of dimension m× n de random values

2.3 Basic operations on the matrices:

L'opération	Signification
+	Addition
-	Subtraction
.*	The multiplication element by element
./	La division élément par élément
.	La division inverse élément par élément
.^	La puissance élément par élément
*****	La multiplication matricielle
/	La division matricielle (A/B) = ($A*B^{-1}$)

The element-by-element operations on matrices are the same as those for vectors (the only condition necessary to make an element-by-element operation is that both matrices have the same dimensions). However, the multiplication or division of matrices requires some constraints (see a course on matrix algebra for more details).

Exemple :

```
>> A=ones(2,3)
      A =
            1     1     1
            1     1     1
>> B=zeros(3,2)
      B =
            0     0
            0     0
            0     0
>> B=B+3
B =
     3     3
     3     3
     3     3
>> A*B
      ans =
            9     9
            9     9
>> B=[B , [3 3 3]']          % ou bien B(:,3)=[3 3 3]'
      B =
            3     3     3
            3     3     3
            3     3     3
>> B=B(1:2,:)                % ou bien B(3,:)=[]
      B =
            3     3     3
            3     3     3
>> A=A*2
      A =
            2     2     2
            2     2     2
>> A.*B
```

```
      ans =
            6     6     6
            6     6     6
>> A*eye(3)
      ans =
            2     2     2
            2     2     2
```

2.4 Useful functions for matrix processing

Here are some of the most used functions regarding matrices:

fonction	usefulness	Exemple of use
det	Determinant calculation of a matrix	**>> A=[1,2;3,4] ;** **>>det(A)** **ans =** **-2**
inv	Calculates the inverse of a matrix	**>>inv(A)** **ans =** **-2.0000 1.0000** **1.5000 -0.5000**
rank	Calculates the rank of a matrix	**>>rank(A)** **ans =** **2**
trace	Calculates the trace of a matrix	**>> trace(A)** **ans =** **5**
eig	Calculates the eigenvalues	**>>eig(A)** **ans =** **-0.3723** **5.3723**
dot	Calculates the scalar product of 2 vectors	**>> v=[-1,5,3];** **>> u=[2,-2,1];** **>>dot(u,v)** **ans =** **-9**
norm	Calculates the standard of a vector	**>>norm(u)** **ans =** **3**
cross	Calculates the vector product of 2 vectors	**>>cross(u,v)** **ans =** **-11 -7 8**
diag	Returns the diagonal of a matrix	**>>diag(A)** **ans =** **1** **4**
diag(V)	Creates a matrix with vector V in the diagonal and 0 elsewhere.	**>> V=[-5,1,3]** **>>diag(V)** **ans =** **-5 0 0** **0 1 0** **0 0 3**
tril	Returns the lower triangular part	**>> B=[1,2,3;4,5,6;7,8,9]**

		B = **1 2 3** **4 5 6** **7 8 9** **>>tril(B)** **ans =** **1 0 0** **4 5 0** **7 8 9** **>>tril(B,-1)** **ans =** **0 0 0** **4 0 0** **7 8 0** **>>tril(B,-2)** **ans =** **0 0 0** **0 0 0** **7 0 0**
triu	Returns the upper triangular part	**>>triu(B)** **ans =** **1 2 3** **0 5 6** **0 0 9** **>>triu(B,-1)** **ans =** **1 2 3** **4 5 6** **0 8 9** **>>triu(B,1)** **ans =** **0 2 3** **0 0 6** **0 0 0**