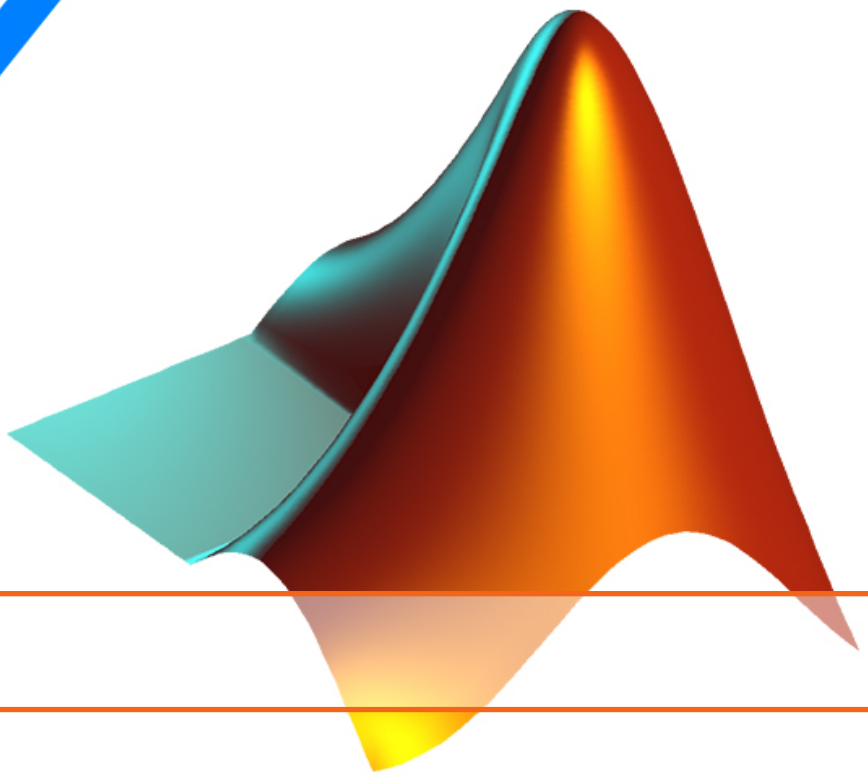
II Vectors and Matrices

CYDOMEDIA

WHAT IS
MATLAB?
MATLAB PLOT AND
MATLAB FUNCTIONS



4. The vectors

4.1 Creation:

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 line just 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 column vector, you can 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 a vector X are ordered with consecutive values, it can be represented using the following notation:

$$X = \texttt{first\_element} : \texttt{last element}$$

This generates a vector X that starts from **first_element** and increments by 1 until it reaches **last_element**. For example:
X=1:5
This will result in the vector:
X=[1 , 2 , 3 , 4 , 5]

■ Example 4.1

```
>> X=1:8 % you can also write colomn (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 a vector X are ordered with consecutive values but with a pitch (increment/decrement) different from 1, the pitch can be specified using the following notation:

$$X = \texttt{first\_element} : \texttt{step} : \texttt{last element}$$

Or: **X=start : pitch : end**
For example, for a vector with values starting at 1, incrementing by 2, and ending at 9, it would be written as:

```
>> X = 1:2:9
```

The notation $X = 1:2:9$ in MATLAB defines a vector X that starts at 1, increments by 2, and ends at 9. The resulting vector will be:

$X = [1,3,5,7,9]$

This syntax is a compact way to generate vectors with a specified start value, step size (pitch), and end value.

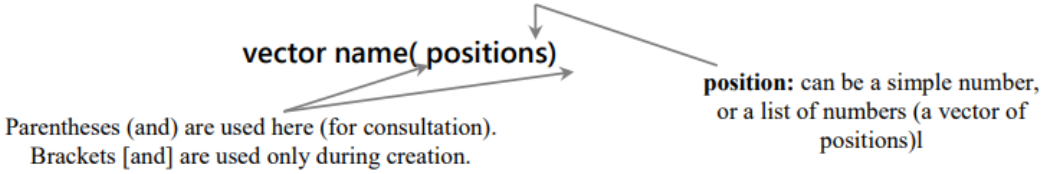
■ **Example 4.2**
```
>> X = [0:2:10] % vector X contains even numbers < 12
X =
0   2   4   6   8   10
>> X= [-4:2:6] % on can also write colon (-4,2,6)
X =
-4   -2   0   2   4   6
>> X= 0:0.2:1 % you can also write colon (0,0.2,1)
X =
0   0.2000   0.4000   0.6000   0.8000   1.0000
```
You can write 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
```

■

4.2 Referencing and access to vector elements:

The elements of a vector can be accessed using the following syntax:



Or **"vector(index)".**
Here, vector refers to the name of the vector, and index specifies the position of the element you want to retrieve. MATLAB uses 1-based indexing, meaning the first element is accessed with index=1. For example, for the vector:

$X = [10,20,30,40]$

To access the second element, you would use: $X(2)$
which returns 20.

■ **Example 4.3**

```
>> 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
```

■

4.3 Element-by-element operations for vectors:

With two vectors u and 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`

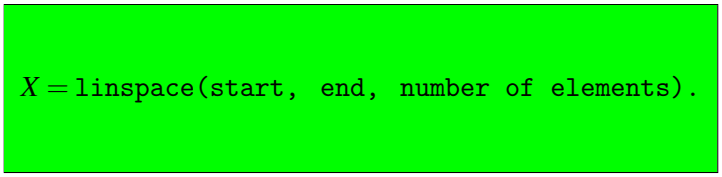
./	Division element by element	`>> u/2`
		`ans =`
		`-1.0000 3.0000 0.5000`
		`>> u./2`
		`ans =`
		`-1.0000 3.0000 0.5000`
		`>> u./v`
		`ans =`
		`-0.6667 -6.0000 0.2500`

.∧	Power item by item	`>> u.^2`
		`ans =`
		`4 36 1`
		`>> u.^v`
		`ans =`
		`-8.0000 0.1667 1.0000`

Writing an expression such as: $u\hat{}2$ generates an error car this expression refers to a matrix multiplication ($u*u$ must be rewritten $u*u'$ or $u'*u$ to be valid).

4.4 The linspace function:

The creation of a vector with components ordered by a regular interval and a specified number of elements can be achieved using the linspace function in **MATLAB**. The syntax is as follows:




```
X = linspace(start_value, end_value, number_of_elements)
```

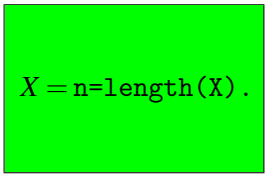
Here, **start_value** is the first element of the vector, **end_value** is the last element, and **number_of_elements** specifies how many elements should be evenly spaced between the two values. Matlab calculates the increment step automatically according to the formula:

$$step = \frac{end\_value - start\_value}{num\_elements - 1},$$

For example:

```
>> 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 (i.e., the number of its components) can be obtained using the length function in MATLAB, as follows:

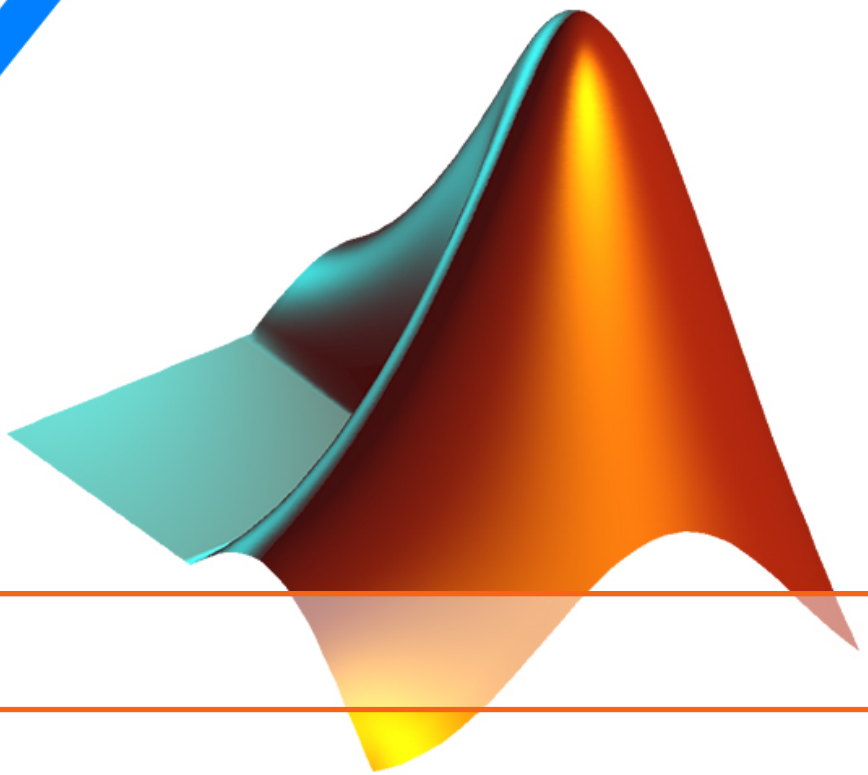


Here, X is the vector for which you want to find the size, and n will hold the number of elements in the vector.
For example:

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

CYDOMEDIA

WHAT IS
MATLAB?
MATLAB PLOT AND
MATLAB FUNCTIONS



5. The matrices:

5.1 Creation:

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]
Error using vertcat
CAT arguments dimensions are not consistent.
```

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

■ **Example 5.1**
```
>> 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
```
■

■ **Example 5.2**
```
>> t1 = [1 2 ; 2 3]
 t1 =
 1   2
 2   3
>> t1 = [3 4 ; 6 7]
 t2 =
 3   4
 6   7
>> tL = [t1 , t2] % ou [t1 t2]
 tL =
1   2   3   4
2   3   6   7
>> tc = [ t1 ; t2 ]
 tc =
1   2
2   3
3   4
6   7
```
■

Exercise 5.1 MATLAB is particularly well-suited for numerical applications involving matrices. Let's look at some methods for manipulating them.

- Define a matrix $M = [1\,2\,;\,3\,4]$ and then try the following operations in the interpreter:

```
>> 2 * M + 3
>> M + M
>> sqrt(M)
>> M * M
>> M .* M
>> ones(4)
>> ones(3, 5)
```

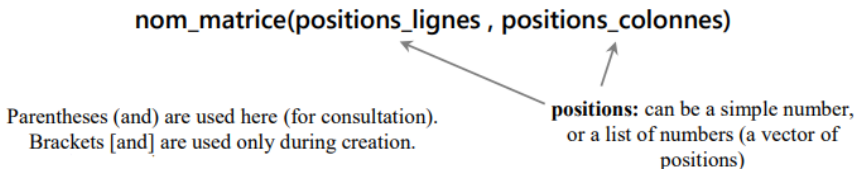
- What is the difference between the $*$ and $.*$ operators? What does the ones function do?
- How can we easily create a 54×42 matrix containing only 7s?

Response 5.1

1. The operator `*` denotes the product of two matrices (as you have seen in mathematics), whereas `.*` denotes the result of the element-by-element product of two matrices of the same dimensions: the entry (i, j) of the result is the product of the entries (i, j) of each of the two original matrices. The function `ones` creates a matrix with the specified dimensions containing only 1s.
2. `>> M = 7 * ones(54, 42);`

5.2 Referencing and access to matrix elements:

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



Or: **matrix(row,column)**.
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: $A(:, j)$.

■ Example 5.3

```
>> 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 obtained using the **size** function. When applied to a matrix A of dimension $m \times n$, this function returns a vector with two components: the first component represents the number of rows m, and the second component represents the number of columns n. The syntax is as follows:

```
dims = size(A).
```

For example:

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

Here, the variable `d` contains the dimensions of the matrix `A` as a vector. To obtain the separate dimensions, you can use the following 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
```

5.3 Automatic generation of matrices:

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

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

■ **Example 5.4** `>> ones(2)`

```
ans =
1   1
1   1
>> ones(3,6)
ans =
1   1   1   1   1   1
1   1   1   1   1   1
1   1   1   1   1   1
>> zeros(3)
ans =
0   0   0
0   0   0
0   0   0
>> zeros(2,7)
ans =
0   0   0   0   0   0   0
0   0   0   0   0   0   0
>> eye(4)
ans =
1   0   0   0
0   1   0   0
0   0   1   0
```

■

In **MATLAB**, the `magic(n)` function generates an $n \times n$ magic square, which is a square matrix where the sum of every row, column, and diagonal is the same. This number is known as the *magic constant*.

`M = magic(n)`

Input:

- `n`: The size of the magic square. It must be a positive integer.

Output:

- `M`: An $n \times n$ matrix where the sum of every row, column, and diagonal is equal.

■ **Example 5.5**

```
>> K = magic(4)
K =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
>> n = 3;
>> M = magic(n);
>> disp(M);
ans =
8   1   6
3   5   7
4   9   2
```

■

Magic Squares In a 3×3 magic square, the sum of each row, column, and diagonal is 15, which is the magic constant for a 3×3 square.

Magic Constant Formula The magic constant C for an $n \times n$ magic square is given by:

$$C = \frac{n(n^2+1)}{2}$$

For example, for a 3×3 square:

$$C = \frac{3(9+1)}{2} = 15$$

Properties of Magic Squares

- The elements of a magic square are distinct integers from 1 to n^2.
- The sum of each row, column, and diagonal is equal to the same magic constant C.
- The magic square is symmetric with respect to its center.

In MATLAB, the `rand(m, n)` function generates an $m \times n$ matrix of random numbers that are uniformly distributed between 0 and 1.

The MATLAB command `A = rand(m, n)` generates a matrix of random numbers.

Input

- **m:** The number of rows in the output matrix.
- **n:** The number of columns in the output matrix.

Output

- **A:** An $m \times n$ matrix where each element is a random number between 0 and 1, drawn from a uniform distribution.

■ **Example 5.6** Generate a 3×2 matrix of random numbers:

```
>> A = rand(3, 2);
>> disp(A);
d1 =
0.2761   0.5439
0.8059   0.9021
0.4923   0.6242
```

Generate a 4×4 matrix of random numbers:

```
>> B = rand(4, 4);
>> disp(B);
d1 =
0.9397   0.1261   0.0142   0.6077
0.6175   0.5161   0.7350   0.8002
0.9601   0.3401   0.4602   0.0503
0.3833   0.9069   0.5592   0.7212
```

■

5.4 Basic operations on the matrices:

The element-by-element operations on matrices are the same as those for vectors. The only requirement for performing such operations is that both matrices must have the same dimensions. However, the multiplication or division of matrices requires certain constraints to be satisfied (see a course on matrix algebra for more details).

operation	Meaning
+	Addition
-	Subtraction
.∗	The multiplication element by element (Element-wise multiplication)
./	The division element by element (Element-wise division)
.\	The division inverse element by element (Element-wise inverse division)
.∧	The power element by element
∗	Matrix multiplication (La multiplication matricielle)
/	Matrix division $(A/B) = (A * B^{-1})$ (La division matricielle)

■ **Example 5.7**

```
>> 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
```

■

5.5 Useful functions for matrix processing:

Here are some of the most used functions regarding matrices:

function	usefulness	Example 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`

function	usefulness	Example of use

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];`
		`>> 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`

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`

Other functions

- `poly(A)`: Returns the coefficients of the characteristic polynomial associated with matrix `A`

```
>> A = [1, 2; 3, 4];
>> poly(A)
ans =
    1.0000   -5.0000   -2.0000
```

- sort(A): Sorts the columns of `A` in ascending order. To sort in descending order, use `sort(A, 'descend')`

```
>> A = [3, 9, 2; 6, 5, 4;2 16 2];
>> sort(A)
ans =
     2     5     2
     3     9     2
     6    16     4
```

- reshape(A, n, m): Resizes `A` into a matrix of dimension `n × m` (number of elements must match)

```
A =
     3     9     2     6     5     4     2    16     2
>> reshape(A, 3, 3)
ans =
     3     6     2
     9     5    16
     2     4     2
```