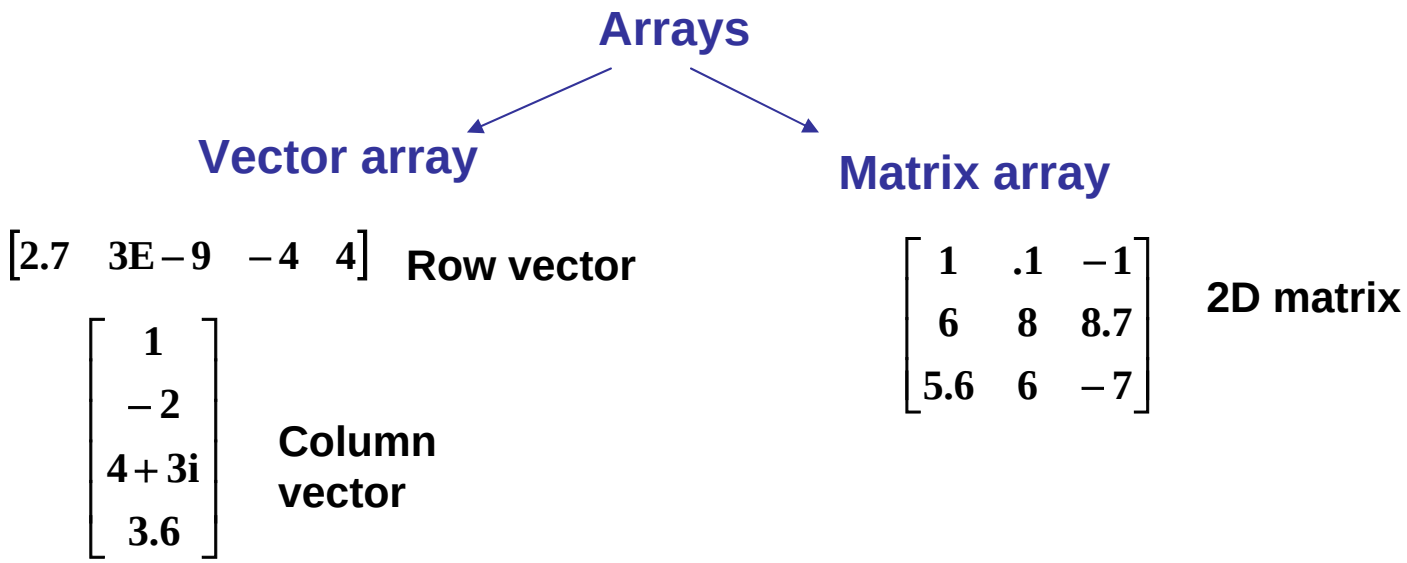
Scalars: Variables that represent single numbers, as considered to this point. Note that complex numbers are scalars, even though they have two components, the real part and the imaginary part.

Arrays: Variables that represent more than one number. Each number is called an **element** of the array. Rather than than performing the same operation on one number at a time, array operations allow operating on multiple numbers at once.

Row and Column Arrays: A row of numbers (called a row vector) or a column of numbers (called a column vector).

Two-Dimensional Arrays: A two-dimensional table of numbers, called a matrix.



Vettori

Consider computing $y = \sin(x)$ for $0 \leq x \leq \pi$. It is impossible to compute y values for all values of x, since there are an infinite number of values, so we will choose a finite number of x values. Consider computing

$$y = \sin(x), \quad \text{where } x = 0, 0.1\pi, 0.2\pi, \ldots, \pi$$

You can form a list, or **array** of the values of x, and then using a calculator you can compute the corresponding values of y, forming a second array y. These can be written down as:

x	0	0.1 π	0.2 π	0.3 π	0.4 π	0.5 π	0.6 π	0.7 π	0.8 π	0.9 π	π
y	0	.31	.59	.81	.95	1.0	.95	.81	.59	.31	0

Elements can be denoted by subscripts, e.g. x_1 is the first element in x y_5 is the fifth element in y. The subscript is the index, address, or location of the element in the array.

Creazione di vettori: lista esplicita

```
>> x=[0 .1*pi .2*pi .3*pi .4*pi .5*pi .6*pi .7*pi .8*pi .9*pi pi]
x =
  Columns 1 through 7
         0    0.3142    0.6283    0.9425    1.2566    1.5708    1.8850
  Columns 8 through 11
    2.1991    2.5133    2.8274    3.1416
```

MATLAB functions can be applied to vectors to compute a resulting vector:

```
>> y=sin(x)
y =
  Columns 1 through 7
         0    0.3090    0.5878    0.8090    0.9511    1.0000    0.9511
  Columns 8 through 11
    0.8090    0.5878    0.3090    0.0000
```

Indirizzamento di un elemento di un vettore:

A vector element is addressed in MATLAB with an integer index (also called a *subscript*) enclosed in parentheses. For example, to access the third element of x and the fifth element of y:

```
>> x(3)
ans =
    0.6283
```

```
>> y(5)
ans =
    0.9511
```

N.B.: differentemente dal linguaggio C, gli indici in Matlab cominciano da 1!

Elementi di matrici possono essere sia a sx che a ds di un assegnamento:

```
>> A = x(1)
>> y(11) = 0
```

Indirizzamento di un gruppo di elementi di un vettore:

Colon notation: Addresses a block of elements `(start:increment:end)`

- **Start: primo indice**
- **Increment: numero da aggiungere per avere l'indice successivo**
- **End: ultimo indice**

Note:

- ✓ ***start*, *increment* e *end* devono essere interi**
- ✓ ***Increment* può essere negativo**
- ✓ **gli indici devono essere positivi**
- ✓ **Se increment =1, si può omettere ed utilizzare (start:end)**

- `2:2:7` means "start with 2, count up by 2, and stop at 7."

```
>> x(2:2:7)
ans =
    0.3142    0.9425    1.5708
```

- `1:5` means "start with 1 and count up to 5."

```
>> x(1:5)
ans =
         0    0.3142    0.6283    0.9425    1.2566
```

- `3:-1:1` means "start with 3 and count down to 1."

```
>> y(3:-1:1)
ans =
    0.5878    0.3090         0
```

- `7:end` means "start with 7 and count up to the end of the vector."

```
>> x(7:end)
ans =
    1.8850    2.1991    2.5133    2.8274    3.1416
```

Creazione di vettori: altri metodi

1. Explicit list
2. Combining

- **Combining**: A vector can also be defined using another vector that has already been defined. For example:

```
>> B = [1.5, 3.1];
>> S = [3.0 B]
S =
    3.0000    1.5000    3.1000
```

3. Changing

- **Changing**: Values can be changed by referencing a specific address. For example,

```
>> S(2) = -1.0;
>> S
S =
    3.0000   -1.0000    3.1000
```

changes the second value in the vector `S` from 1.5 to −1.0.

4. Extending

- **Extending**: Additional values can be added using a reference to a specific address. For example, the following command:

```
>> S(4) = 5.5;
>> S
S =
    3.0000   -1.0000    3.1000    5.5000
```

extends the length of vector `S` from 3 to 4.

Applying the following command

```
>> S(7) = 8.5;
>> S
S =
    3.0000   -1.0000    3.1000    5.5000         0         0    8.5000
```

extends the length of `S` to 7, and the values of `S(5)` and `S(6)` are automatically set to zero because no values were given for them.

5. colon notation

```
(start:increment:end)
```

N.B:

- ✓ ***start*, *increment* e *end* possono non essere interi e possono avere sia valori positivi che negativi**

```
>> x=(0:0.1:1)*pi
x =
  Columns 1 through 7
         0    0.3142    0.6283    0.9425    1.2566    1.5708    1.8850
  Columns 8 through 11
    2.1991    2.5133    2.8274    3.1416
```

>> z = (9.9 : -1.1 : -9.9) oppure: >> z = [9.9 : -1.1 : -9.9]

6. linspace function

- `linspace`: This function generates a vector of uniformly incremented values, but instead of specifying the increment, the number of values desired is specified. This function has the form:

  ```
  linspace(start,end,number)
  ```

  The increment is computed internally, having the value:

$$\texttt{increment} = \frac{\texttt{end} - \texttt{start}}{\texttt{number} - 1}$$

```
>> x=linspace(0,pi,11)
x =
  Columns 1 through 7
         0    0.3142    0.6283    0.9425    1.2566    1.5708    1.8850
  Columns 8 through 11
    2.1991    2.5133    2.8274    3.1416
```

In this example:

$$\texttt{increment} = \frac{\pi}{11 - 1} = 0.1\pi$$

Vettori colonna:

1. lista esplicita

A column vector, having one column and multiple rows, can be created by specifying it element by element, separating element values with semicolons:

```
>> c = [1;2;3;4;5]
c =
     1
     2
     3
     4
     5
```

oppure:

```
>> C1 = [1
2
3
4
5]
```

2. Trasposta di vettore riga

The **transpose** operator (') is used to transpose a row vector into a column vector

```
>> a = 1:5
a =
     1     2     3     4     5
>> b = a'
b =
     1
     2
     3
     4
     5
```

Nota: **l'operatore trasposta in un vettore di numeri complessi, genera un vettore trasposto, con i valori coniugati:**

```
>> a = [1+j 2-3j -3+4j]
a =
   1.0000+ 1.0000i   2.0000- 3.0000i  -3.0000+ 4.0000i
>> b = a'
b =
   1.0000- 1.0000i
   2.0000+ 3.0000i
  -3.0000- 4.0000i
```

per non eseguire la coniugazione, occorre anteporre . prima dell'operatore ':

```
>> c = a.'
c =
   1.0000+ 1.0000i
   2.0000- 3.0000i
  -3.0000+ 4.0000i
```

Matrici

- Begin with [, end with]
- Spaces or commas are used to separate elements in a row.
- A semicolon or Enter is used to separate rows.

>> A = [1 2 3; 4 5 6 ; 7 8 9] oppure: >> A = [1,2,3; 4,5,6; 7,8,9]

oppure: >> A = [1 2 3
4 5 6
7 8 9]

Indirizzamento elementi di matrice:

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

```
>> c1 = C(2,3)
```

La variabile c1 ha il valore dell'elemento riga 2, colonna 3 di C

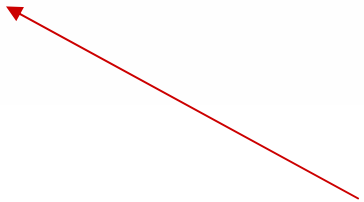
Creazione sotto_matrici

```
>>  D= [1 2 3; 4 5 6; 7 8 9; 10 11 12]
>> D_1 = D(:,2);     % vettore colonna
>> D_2 = D(:,2:3);  % matrice 4x2
>> D_3 = D(1:2,2:end)  % matrice 2x3
```

Operazioni su matrici elemento-per-elemento:

When two arrays have the same dimensions, addition, subtraction, multiplication, and division apply on an element-by-element basis. The notation for some operations is somewhat unconventional.

Operation	Algebraic Form	MATLAB
Addition	$a + b$	`a + b`
Subtraction	$a - b$	`a - b`
Multiplication	$a \times b$	`a.*b`
Division	$a \div b$	`a./b`
Exponentiation	a^b	`a.^b`



Per ottenere prodotto, divisione, elevamento di potenza, elemento-per-elemento di vettori e/o matrici, è necessario aggiungere il . prima dell'operatore (*,/,^)

Funzioni Matlab per ottenere le dimensioni di matrici

Command	Description
`s = size(A)`	For an `m` × `n` matrix `A`, returns the two-element row vector `s = [m, n]` containing the number of rows and columns in the matrix.
`[r,c] = size(A)`	Returns two scalars `r` and `c` containing the number of rows and columns in `A`, respectively.
`r = size(A,1)`	Returns the number of rows in `A` in the variable `r`.
`c = size(A,2)`	Returns the number of columns in `A` in the variable `c`.
`n=length(A)`	Returns the larger of the number of rows or columns in `A` in the variable `n`.

Esempio:

```
% esempio di utilizzo della funzione "length"
V(1:10) = 1.1
dim_V = length(V)


% esempi di utilizzo della funzione "size"
A = [1:3;4:6;7:9;10:12]   % creata la matrice A di 4x3 elementi


s = size(A)           % numero di righe e numero di colonne di A
[r s] = size(A)


n_righe = size(A,1)    % solo numero di righe di A
n_colonne = size(A,2)  % solo numero di colonne di A


max_dim_di_A = length(A)    % solo la dimensione massima di A
```

Funzioni Matlab per creare matrici con 1 o 0

Two special functions in MATLAB can be used to generate new matrices containing all zeros or all ones.

`zeros(n)`	Returns an $n \times n$ array of zeros.
`zeros(m,n)`	Returns an $m \times n$ array of zeros.
`zeros(size(A))`	Returns an array the same size as `A` containing all zeros.
`ones(n)`	Returns an $n \times n$ array of ones.
`ones(m,n)`	Returns an $m \times n$ array of ones.
`ones(size(A)`	Returns an array the same size as `A` containing all ones.

Esempio:

```
%  Esempi di creazione di matrici con 1 o 0

Zq = zeros(3)    %matrice quadrata 3x3 di zeri
Zr = zeros(2,3)  % matrice rettangolare 2x3 di zeri

A = [-1:1;3:-1:1];  % matrice 2x3
Za = zeros(size(A)) % matrice di zeri, delle stesse dimensioni di A



Uq = ones(4)    %matrice quadrata 4x4 di 1
Ur = ones(2,4)  %matrice rettangolare 2x4 di 1

B = [2.2:2:6.2;6.2:-2:2.2];  %matrice 2x3
Ub = ones(size(B))
```

Funzioni Matlab per creare matrici speciali:

Matrice identità:

eye(n) crea una matrice identità nxn

Matrici triangolari:

tril(A) crea una matrice triangolare inferiore, dalla matrice A

triu(A) crea una matrice triangolare superiore, dalla matrice A

Matrice diagonale o vettore da matrice diagonale:

D = diag(T)

- ✓ se T è vettore: D è una matrice con nella diagonale gli elementi di T
- ✓ se T è matrice: D è un vettore i cui elementi sono gli elementi nella diagonale di T

Esempio:

```
% esempi di matrici speciali

%matrice identità
ID = eye(4)   %matrice identità 4x4

A(1:3,1:3) = 3.3;   % creo una matrice 3x3 con valori 3.3

%matrici triangolari
Tl = tril(A)      % matrice triangolare inferiore da A
Tu = triu(A)      % matrice triangolare superiore da A

% vettore dalla diagonale di una matrice
V = diag(A)

% matrice diagonale da un vettore
DA = diag(V)
```

Creare una matrice da un vettore:

reshape(V,n,m) crea una matrice nxm, dal vettore V

n+m = length(V)

```
% utilizzo della funzione "reshape" per creare matrici da vettori
R = [1:9]
T_from_R = reshape(R,3,3)  %creo una matrice 3x3 dal vettore R
```

```
R = 1   2   3   4   5   6   7   8   9

T_from_R =

     1     4     7
     2     5     8
     3     6     9
```

Funzioni Matlab di algebra lineare su matrici:

Inversa di una matrice:

I = inv(A) **Se la matrice quadrata e invertibile: restituisce la sua inversa**

Determinante di una matrice:

D = det(A)

Rango di una matrice:

R = rank(A)

Soluzione di sistemi lineari:

$$\begin{cases} x_1 + 2x_2 + 3x_3 = 1 \\ x_2 + 2x_3 = 2 \\ x_3 = 3 \end{cases} \Longleftrightarrow A = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 2 \\ 0 & 0 & 3 \end{bmatrix} \quad x = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \quad b = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

$$Ax = b \qquad x = A^{-1}b$$

In Matlab:

x = b\A

Back-slash

Dà la soluzione anche se la matrice A non è quadrata e/o è singolare. Utilizza metodi di approssimazione per trovare la/le soluzioni

Se A è non singolare si può anche utilizzare il metodo "classico":

x = inv(A)*b

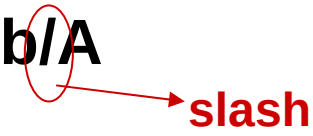
Soluzione di sistemi lineari:

Alternativa (*meno usata*):

$xA = b \qquad x = bA^{-1}$

In Matlab:

X = b/A



Proprietà degli operatori ‘/’ e ‘\’:

(b/A)’ = (A’\b’)

Le stringhe in Matlab

Una stringa di caratteri è delimitata tra apici (‘):

>> S1 = ‘ciao mondo’

>> S2 = ‘questa e’’ una stringa’

Una stringa di caratteri è un vettore → **tutte le operazioni viste per i vettori si possono applicare anche alle stringhe:**

```
>> S1_primo  = S1(1:4)        % sotto-vettore
      S1_primo  =
      ciao
>> S1_secondo = S1(5:end)
       s1_secondo =
       mondo
>> S3 = [S1_primo S1_secondo]   % concatenazione di vettori
```

Conversione di stringhe:

`char(x)`	Converts the array `x` that contains positive integers representing character codes into a character array (the first 127 codes are ASCII). The result for any elements of `x` outside the range from 0 to 65535 is not defined.
`int2str(x)`	Rounds the elements of the matrix `x` to integers and converts the result into a string matrix.
`num2str(x)`	Converts the matrix `x` into a string representation with about 4 digits and an exponent if required. This is useful for labeling plots with the `title`, `xlabel`, `ylabel`, and `text` commands.
`str2num(s)`	Converts the string `s`, which should be an ASCII character representation of a numeric value, to numeric representation. The string may contain digits, a decimal point, a leading + or - sign, an 'e' preceeding a power of 10 scale factor, and an 'i' for a complex unit.

Esempi:

```
>> X = [120:125]     % X = [120   121   122   123   124   125]
>> C = char(X)       % C = 'xyz{|}'

>>  S = num2str(X)  % S =   '120  121  122  123  124  125'

>>str_num = str2num('-2.3e2')   % str2num= -230
```

La funzione "*num2str*" viene spesso utilizzata

- nelle didascalie dei grafici (lo vedremo nelle prossime lezioni)
- nella funzione "*disp*"

La funzione "*str2num*" viene utilizzata

- quando si fa l'import di dati memorizzati in modalità testo

Esempio di utilizzo di *num2str*:

```
altezza = 2.3e-2;
base = 3.2e-2;
area = base*altezza;

% utilizzo “disp” per mostrare nel monitor i valori delle varie grandezze
disp(['base= ' num2str(base) 'm. '...
    'altezza= ' num2str(altezza) 'm. '...
    'area= ' num2str(area) 'mq.'])
```