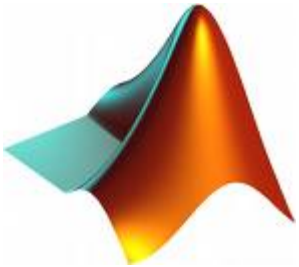


Simulazione dei Sistemi dinamici con Matlab-Simulink

A.A. 2023-24

Creazione e manipolazione di array

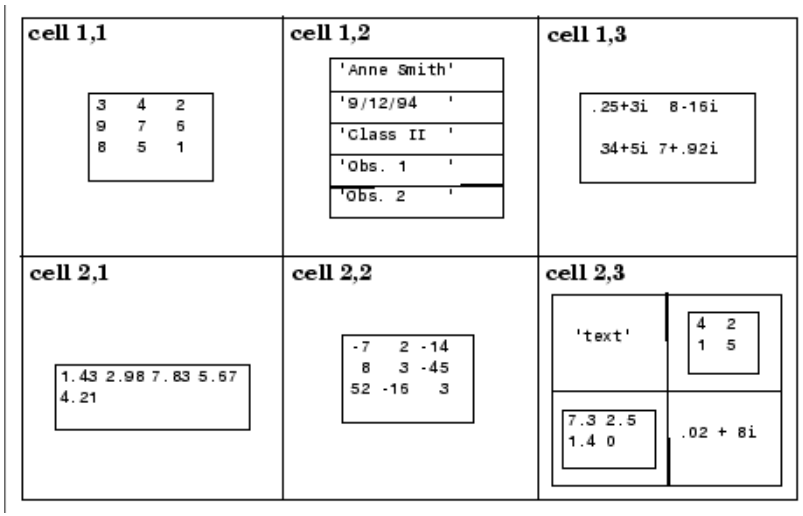
Prof. Alessandro Pisano
apisano@unica.it

Tipi di variabili

Il tipo di dato più importante in Matlab è il tipo **Array**.
Il tipo di Array più utilizzato è in particolare **l'Array bidimensionale** (n righe ed m colonne). Esso include gli scalari (1x1), i vettori riga e colonna (1xn , nx1), e le matrici quadrate e rettangolari.

In moltissimi problemi di simulazione e calcolo scientifico il tipo di dato Array bidimensionale per tutte le variabili è sufficiente a coprire le esigenze del problema. Da qui a breve studieremo nel dettaglio la creazione e la manipolazione di Array bidimensionali.

Sono disponibili anche altri tipi di dati, ad esempio gli array multidimensionali, il tipo `struct`, analogo al tipo di dato struct del linguaggio C (una struttura ripetitiva con dei sottocampi eterogenei), o il tipo `cell array` (v. figura)



Creazione di vettori e matrici

La sintassi di base per la creazione di un vettore riga prevede che gli elementi siano inseriti fra parentesi quadre, e separati da uno spazio o, indifferentemente, da una virgola.

```
v1=[1 2 3 4 5];
v1=[1, 2, 3, 4, 5];
```

Istruzioni equivalenti

L'inserimento o meno del punto e virgola alla fine di un istruzione, come già visto, abilita o meno la stampa nella Command Window della variabile creata.

Anche per la creazione di un vettore colonna ci sono varie strade.

```
v2=[1; 2; 3; 4; 5];
```

Il punto e virgola implica il passaggio alla riga successiva del vettore

In alternativa si può usare l'apice, che corrisponde all'operazione di trasposizione (di un vettore, ma, come vedremo a breve, anche di una matrice)

```
v2=[1 2 3 4 5]';
```

Creazione di vettori e matrici

La sintassi di base per la creazione di una matrice (array bidimensionale) prende spunto dalla sintassi per la creazione di un vettore, e prevede che il passaggio alla riga successiva della matrice sia individuato mediante un punto e virgola

```
M1=[1 3 5; 2 4 6; 7 8 10]
```

```
M1 =

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

Output nella Command Window

Creazione di una matrice rettangolare

```
M2=[1 3 5 7; 2 4 6 8]
```

```
M2 =

     1     3     5     7
     2     4     6     8
```

Creazione di vettori e matrici

Matrici o vettori possono ovviamente essere creati definendone gli elementi per mezzo di variabili precedentemente definite

```
a=1;
M3=[a 3 5; 2 a+1 6; 7 8 a+2]
```

```
M3 =

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

La seguente sintassi mostra come le variabili in ambiente Matlab siano case sensitive.

```
a=2
A=3
M4=[a A; 2*A a^2]
```

```
a =

     2


A =

     3


M4 =

     2     3
     6     4
```

`a` ed `A` sono due variabili distinte che non hanno nulla a che fare l'una con l'altra!

Creazione di vettori e matrici

Le precedenti sintassi sono evidentemente inefficaci per la creazione di vettori o matrici di grosse dimensioni.

Vediamo come creare in maniera semplice vettori e matrici di dimensione potenzialmente elevata che abbiamo una predefinita «struttura».

Vettori ordinati in cui gli elementi adiacenti hanno incremento unitario

Sintassi generale

```
v4=elemento_iniziale:elemento_finale;
```

Esempio

```
v4=1:200;

v5=1.5:5.5;
```

Creazione di vettori e matrici

Vettori ordinati con incremento/decremento arbitrario

Sintassi generale

```
v5=elemento_iniziale:incremento:elemento_finale;
```

Esempio: la seguente sintassi crea un vettore in cui il primo elemento è nullo, l'ultimo elemento è unitario, e l'incremento fra elementi adiacenti è pari a 0.1

```
v5=0:0.1:1
```

Il vettore cosi generato ha evidentemente 11 elementi, e la seguente struttura:

```
v5 =

  Columns 1 through 9

         0    0.1000    0.2000    0.3000    0.4000    0.5000    0.6000    0.7000    0.8000

  Columns 10 through 11

    0.9000    1.0000
```

Creazione di vettori e matrici

L'incremento può anche essere negativo

Esempio: `v6=2:-0.2:0`

```
v6 =

  Columns 1 through 9

    2.0000    1.8000    1.6000    1.4000    1.2000    1.0000    0.8000    0.6000    0.4000

  Columns 10 through 11

    0.2000         0
```

Attenzione: se l'incremento non è opportunamente dimensionato in relazione agli elementi iniziale e finale, può accadere che l'ultimo elemento del vettore non coincida con quanto specificato nella sintassi nel parametro `elemento_finale`

Esempio: `v7=0:0.15:0.7`

Il vettore creato è il seguente:

```
v7 =

         0    0.1500    0.3000    0.4500    0.6000
```

Creazione di vettori e matrici

Una sintassi alternativa per la creazione di vettori equispaziati prevede l'impiego della funzione `linspace`

Sintassi generale

```
V=linspace(elemento_iniziale, elemento_finale,dimensione)
```

Esempio

```
v8=linspace(0,pi,50)
```

Il vettore creato ha elemento iniziale nullo, elemento finale pari a pigreco, e 50 elementi equispaziati. La spaziatura fra gli elementi adiacenti viene pertanto determinata «automaticamente»

La funzione linspace può anche generare vettori aventi elementi ordinati in senso decrescente

```
v9=linspace(1,0,25)
```

```
v9 =

  Columns 1 through 9

    1.0000    0.9583    0.9167    0.8750    0.8333    0.7917    0.7500    0.7083    0.6667

  Columns 10 through 18

    0.6250    0.5833    0.5417    0.5000    0.4583    0.4167    0.3750    0.3333    0.2917

  Columns 19 through 25

    0.2500    0.2083    0.1667    0.1250    0.0833    0.0417         0
```

Creazione di vettori e matrici

La funzione `logspace` consente di crear e vettori i cui elementi hanno una spaziatura non costante (come la funzione linspace) ma logaritmica.

Sintassi generale

```
V=logspace(a,b,n)
```

Viene generato un vettore di dimensione n avente come primo elemento 10^a e come ultimo elemento 10^b. Se si omette il parametro n il suo valore di default è 50.

Esempio

```
v10=logspace(-1,1,8)
```

```
v10 =

    0.1000    0.1931    0.3728    0.7197    1.3895    2.6827    5.1795   10.0000
```

Altre istruzioni per la creazione di matrici strutturate

eye(n)	crea una matrice identità di dimensione ***n x n***
ones(n)	crea una matrice quadrata di dimensione ***n*** i cui elementi sono unitari
ones(m,n)	crea una matrice ***m x n*** i cui elementi sono unitari
zeros(n)	crea una matrice quadrata di dimensione ***n*** i cui elementi sono nulli
zeros(m,n)	crea una matrice ***m x n*** i cui elementi sono nulli
rand(n)	crea una matrice quadrata di dimensione ***n*** di elementi casuali compresi nell'intervallo [0,1]
rand(m,n)	crea una matrice ***m x n*** di elementi casuali compresi nell'intervallo [0,1]

Altre istruzioni per la creazione di matrici strutturate

Esempi

```
A1=eye(3)
```

```
A1 =

     1     0     0
     0     1     0
     0     0     1
```

```
A2=ones(5,1)
```

```
A2 =

     1
     1
     1
     1
     1
```

```
A3=ones(4)
```

```
A3 =

     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1
```

Esempi

```
A4=zeros(3,4)
```

```
A4 =

     0     0     0     0
     0     0     0     0
     0     0     0     0
```

```
A5=rand(6)
```

```
A5 =

    0.8147    0.2785    0.9572    0.7922    0.6787    0.7060
    0.9058    0.5469    0.4854    0.9595    0.7577    0.0318
    0.1270    0.9575    0.8003    0.6557    0.7431    0.2769
    0.9134    0.9649    0.1419    0.0357    0.3922    0.0462
    0.6324    0.1576    0.4218    0.8491    0.6555    0.0971
    0.0975    0.9706    0.9157    0.9340    0.1712    0.8235
```

Concatenazione di matrici

La *concatenazione* è il processo di «accorpamento» fra array, per crearne di più grandi.

La *concatenazione* può essere orizzontale o verticale

Esempio di c*oncatenazione* orizzontale

```
M1=eye(3)
M2=zeros(3,4)
```

```
M1 =

     1     0     0
     0     1     0
     0     0     1

M2 =

     0     0     0     0
     0     0     0     0
     0     0     0     0
```

```
M3=[M1 M2]
```

```
M3 =

     1     0     0     0     0     0     0
     0     1     0     0     0     0     0
     0     0     1     0     0     0     0
```

Le matrici devono ovviamente avere il medesimo numero di righe

Concatenazione di matrici

La c*oncatenazione* può anche coinvolgere più di 2 matrici

```
M1=eye(3)
M2=zeros(3,4)
M3=rand(3)

M4=[M1 M2 M3]
```

```
M1 =

     1     0     0
     0     1     0
     0     0     1


M2 =

     0     0     0     0
     0     0     0     0
     0     0     0     0


M3 =

    0.1190    0.3404    0.7513
    0.4984    0.5853    0.2551
    0.9597    0.2238    0.5060


M4 =

    1.0000         0         0         0         0         0         0    0.1190    0.3404    0.7513
         0    1.0000         0         0         0         0         0    0.4984    0.5853    0.2551
         0         0    1.0000         0         0         0         0    0.9597    0.2238    0.5060
```

Esempio di *concatenazione* verticale

```
M1=eye(3)
M2=zeros(4,3)
```

```
M1 =

     1     0     0
     0     1     0
     0     0     1


M2 =

     0     0     0
     0     0     0
     0     0     0
     0     0     0
```

```
M3=[M1;M2]
```

Le matrici devono ovviamente avere il medesimo numero di colonne

```
M3 =

     1     0     0
     0     1     0
     0     0     1
     0     0     0
     0     0     0
     0     0     0
     0     0     0
```

Esercizio: Costruire la seguente matrice 5 x 8

```
M =

     1     0     0     0     0     0     0     0
     0     1     0     0     0     0     0     0
     0     0     1     0     0     0     0     0
     1     1     1     5     5     5     5     5
     1     1     1     5     5     5     5     5
```

Suggerimento:
la sintassi `A=k*ones(n,m)` (in cui k è uno scalare) genera una matrice n x m in cui ciascun elemento è uguale a k

Possibile soluzione:

```
A=eye(3);
B=zeros(3,5);
C=ones(2,3)
D=5*ones(2,5)
M=[A B; C D]
```

Altre istruzioni per la analisi di matrici

size(A) restituisce le dimensioni della matrice ***A***

La funzione size deve ricevere come argomento di ingresso un array, e restituisce all'esterno un vettore di due elementi che sono nell'ordine il numero di righe ed il numero di colonne della matrice A

```
A1=[1 2 3;4 5 6;7 8 9;10 11 12]

dimA1=size(A1)
```

```
A1 =

     1     2     3
     4     5     6
     7     8     9
    10    11    12


dimA1 =

     4     3
```

length(b) restituisce la lunghezza del vettore riga o colonna ***b***

Comandi per gestire una sessione di lavoro con Matlab

clc	cancella il contenuto della finestra dei comandi;
clear all	elimina tutte le variabili dal workspace
clear v1 v2	elimina le variabili v1 e v2 dal workspace
save 14ott2022	salva tutte le variabili del workspace in un file binario `14ott2022.mat`
load 14ott2022	carica nel workspace tutte le variabili salvate nel un file binario `14ott2022.mat`
...	l'istruzione continua nella riga successiva.
1.2e-4	notazione esponenziale per il numero $1.2\ 10^{-4}$

Precedenze

Le parentesi tonde prevalgono su tutti gli operatori.

L'elevamento a potenza (^) ha precedenza superiore al prodotto (*) ed alla divisione (/)

Accesso agli array

Accesso ad elementi singoli

V(i) *identifica l'elemento* ***i****-esimo del vettore* ***V***

```
V=[3 1 7 4 5]
V3=V(3)
```

```
V =

     3     1     7     4     5


V3 =

     7
```

A(n,m) *identifica l'elemento che occupa la riga* ***n*** *e la colonna* ***m*** *nella matrice* ***A***

```
B=[7 2 3; 5 8 6; 8 3 9;10 9 8]
B23=B(2,3)
```

```
B =

     7     2     3
     5     8     6
     8     3     9
    10     9     8


B23 =

     6
```

Accesso agli array

Accesso ad elementi singoli

V(end) *identifica l'ultimo elemento del vettore* ***V***

A(n,end) *identifica l'ultimo elemento della riga* ***n****-esima*
A(end,m) *identifica l'ultimo elemento della riga* ***m****-esima*

```
V=[3 1 7 4 5]
Vlast=V(end)
```

```
V =

     3     1     7     4     5


Vlast =

     5
```

```
V8=V(8)
```

```
Index exceeds the number of array elements. Index must not exceed 5.
```

```
B=[7 2 3; 5 8 6; 8 3 9;10 9 8]
Blastriga1=B(1,end)
Blastcolonna2=B(end,2)
```

```
B =

     7     2     3
     5     8     6
     8     3     9
    10     9     8
```

```
Blastriga1 =

     3


Blastcolonna2 =

     9
```

Accesso agli array

Accesso a porzioni selezionate

vettori

v(n1:n2) *rappresenta tutti gli elementi del vettore* ***v*** *con indice compreso tra n1 ed n2*

v(n2:end) *rappresenta tutti gli elementi del vettore* ***v*** *con indice superiore o uguale ad n2*

Accesso agli array

Accesso a porzioni selezionate - esempi

```
v=2:3:26
```

Creazione di un vettore di test

```
v =

     2     5     8    11    14    17    20    23    26
```

```
w1=v(2:4)
```

Estrae dal vettore le componenti dalla seconda alla quarta

```
w1 =

     5     8    11
```

```
w2=v(4:end)
```

Estrae dal vettore le componenti dalla quarta fino all'ultima

```
w2 =

    11    14    17    20    23    26
```

Accesso agli array

Accesso a porzioni selezionate - esempi

```
w3=v(end-2:end)
```

Estrae dal vettore le ultime 3 componenti

```
w3 =

    20    23    26
```

Accesso agli array

Accesso a porzioni selezionate

matrici

M(i,:) *identifica tutti gli elementi della i-esima riga della matrice* ***M***
M(:,i) *identifica tutti gli elementi della i-esima colonna della matrice* ***M***

M(end,:) *identifica tutti gli elementi della ultima riga della matrice M*
M(:,end) *identifica tutti gli elementi della ultima colonna della matrice M*

M(n1:n2,:) *identifica tutti gli elementi con indice di riga compreso tra n1 ed n2*
M(:,n1:n2) *identifica tutti gli elementi con indice di colonna compreso tra n1 ed n2*

M(3:5,1:6) *identifica tutti gli elementi con indice di riga compreso tra 3 ed 5 e con indice di colonna compreso tra 1 ed 6*

Accesso agli array

Accesso a porzioni selezionate - esempi

```
M=[1 2 3 4;5 6 7 8;9 10 11 12; 13 14 15 16]
```

Creazione di una matrice di test

```
M =

     1     2     3     4
     5     6     7     8
     9    10    11    12
    13    14    15    16
```

```
A1=M(:,3)
```

Estrae dalla matrice la terza colonna

```
A1 =

     3
     7
    11
    15
```

```
A2=M(1:2,:)
```

Estrae dalla matrice le prime due righe

```
A2 =

     1     2     3     4
     5     6     7     8
```

Accesso agli array

Accesso a porzioni selezionate - esempi

```
A3=M(2:3,2:3)
```

Estrae dalla matrice la «parte centrale»

```
A3 =

     6     7
    10    11
```

Modifica di array

Avendo un certo vettore definito nel workspace, ad esempio

```
V=[3  1  7  4  5]
```

Desideriamo modificare il valore di uno o più degli elementi, ad esempio assegnare al terzo elemento il valore nullo. Ciò può essere fatto con la sintassi

```
V(3)=0
```

Il risultato è

```
V =

     3     1     0     4     5
```

Ora modifichiamo anche le prime due componenti del vettore. Ciò può essere fatto con un'unica istruzione:

```
V(1:2)=[7  8]
```

```
V =

     7     8     0     4     5
```

Modifica di array

Se si assegna un valore ad una componente di un vettore con indice più elevato della dimensione del vettore **la dimensione del vettore viene corrispondentemente aumentata e tutti i nuovi valori aggiunti, eccetto l'ultimo, sono posti pari a zero.**

Definiamo un vettore `V` di test con 4 elementi e poi attribuiamo un valore alla decima componente del vettore.

```
V=[1 2 3 4]
V(10)=7
```

```
V =

     1     2     3     4


V =

     1     2     3     4     0     0     0     0     0     7
```

Modifica di array

Vediamo come modificare parti di matrici

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

```
A =

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

Modifichiamone l'elemento centrale

```
A(2,2)=1
```

```
A =

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

Ora modifichiamo («sovrascriviamo») anche la prima riga

```
A(1,:)=[10 10 10]
```

```
A =

    10    10    10
     4     1     6
     7     8     9
```

Rimozione di elementi da un array

E' possibile rimuovere un elemento da un vettore attribuendogli il valore «[]» (che significa «vettore vuoto») tale elemento viene rimosso dal vettore, del quale quindi viene ridotta di 1 la dimensione.

Definiamo un vettore `V` di test con 5 elementi e Rimuoviamo dal vettore **l'ultima componente**

Ciò può essere fatto mediante la seguente sintassi

```
V=[3 1 7 4 5]
V(end)=[]
```

```
V =

     3     1     7     4     5

V =

     3     1     7     4
```

Rimozione simultanea della **seconda e terza componente**

```
w=[1 3 5 7 9 11]
w([2 3])=[]
```

```
w =

     1     3     5     7     9    11

w =

     1     7     9    11
```

Funzioni speciali per matrici

det(A)	Calcola il determinante della matrice quadrata A
inv(A)	Calcola l'inversa della matrice quadrata A
rank(A)	Calcola il rango della matrice A
pinv(A)	Calcola la pseudoinversa della matrice A
eig(A)	Calcola (e restituisce in un vettore) gli autovalori della matrice quadrata A

```
A=[1 2 3;4 5 6;7 8 10];
B=[2 4 6;1 1 3];
```

```
A =

     1     2     3
     4     5     6
     7     8    10

B =

     2     4     6
     1     1     3
```

```
detA=det(A)
detB=det(B)
```

```
detA =

   -3.0000

Error using det
Matrix must be square.

Error in untitled4 (line 8)
detB=det(B)
```

Funzioni speciali per matrici

det(A)	Calcola il determinante della matrice quadrata A
inv(A)	Calcola l'inversa della matrice quadrata A
rank(A)	Calcola il rango della matrice A
pinv(A)	Calcola la pseudoinversa della matrice A
eig(A)	Calcola (e restituisce in un vettore) gli autovalori della matrice A

```
invA=inv(A)
```

```
invA =

   -0.6667   -1.3333    1.0000
   -0.6667    3.6667   -2.0000
    1.0000   -2.0000    1.0000
```

```
autovA=eig(A);

lambda1=autovA(1)
lambda2=autovA(2)
lambda3=autovA(3)
```

```
lambda1 =

   16.7075


lambda2 =

   -0.9057


lambda3 =

    0.1982
```

Operazioni su matrici e vettori

Prodotto scalare e prodotto vettoriale

```
x1=[1 2 3];
x2=[4 5 6];

xscalprod=x1*x2'
```

Prodotto scalare da realizzarsi mediante prodotto standard fra un vettore riga ed un vettore colonna

```
xscalprod =

    32
```

```
xvecprod=cross(x1,x2)
```

Prodotto vettoriale: funzione cross

```
xvecprod =

    -3     6    -3
```

Prodotto fra matrici

```
A1=[1 2 3;4 5 6;7 8 9];
x1=[1 2 3];
```

Il prodotto fra matrici (codificato dall'asterisco, simbolo associato alla moltiplicazione standard) deve rispettare i vincoli dimensionali sul numero di righe e di colonne degli operandi

```
prodotto=A1*x1
```

```
Error using  *
Incorrect dimensions for matrix multiplication. Check that the number of columns in the first matrix matches
the number of rows in the second matrix. To operate on each element of the matrix individually, use TIMES
(.*) for elementwise multiplication.

Error in untitled2 (line 4)
prodotto=A1*x1
```

```
prodotto= A1*x1'
```

```
prodotto =

    14
    32
    50
```

Operazioni su matrici e vettori

MATLAB consente di elaborare simultaneamente tutti i valori di una matrice utilizzando un unico operatore o funzione aritmetica

Sommare il medesimo valore scalare a tutti gli elementi di una matrice

```
A=[1 2 3;4 5 6;7 8 10]
```

```
B=A+2
```

```
A =

     1     2     3
     4     5     6
     7     8    10


B =

     3     4     5
     6     7     8
     9    10    12
```

La medesima operazione può essere ovviamente applicata anche ad un vettore riga o colonna

Funzioni di vettori

Anche tutte le funzioni matematiche scalari (es. sin, sqrt, abs, sign) sono tali che se ricevono in ingresso un vettore (o matrice) restituiscono un vettore (o matrice) di pari dimensione che contiene il risultato della operazione applicata via via a ciascun elemento:

```
x1=[1 2 -3 4 5];
abs(x1)
```

```
x1 =

     1     2    -3     4     5


ans =

     1     2     3     4     5
```

```
A=[2 4;9 16]
sqrt(A)
```

```
A =

     2     4
     9    16


ans =

    1.4142    2.0000
    3.0000    4.0000
```

Funzioni di vettori

Alcuni operatori non si prestano ad essere applicati a vettori.
Ad esempio l'elevamento al quadrato.

```
x1=[1  2  -3  4  5]
x1^2
```

```
Error using  ^
Incorrect dimensions for raising a matrix to a power. Check that the matrix is square and the power is a
scalar. To operate on each element of the matrix individually, use POWER (.^) for elementwise power.

Error in untitled6 (line 2)
x1^2
```

Per applicare a tutti gli elementi x_i di un vettore, ad esempio, l'operazione $sqrt(x_i)/x_i$ incontriamo simili problemi.
Anche il rapporto tra vettori non è una operazione consentita (non viene interpretata in Matlab come il rapporto tra le relative componenti, ma origina un messaggio di errore)

Per realizzare tali funzionalità si utilizzano gli **operatori elemento per elemento**

Operazioni elemento per elemento

+	*somma fra array A e scalare b*	***A+b***
-	*sottrazione array A e scalare b*	***A-b***
.^	*elevamento a potenza tra array A e scalare b*	***A.^b***

```
x1=[1 2 -3 4 5]
X1.^2
```

```
x1 =

     1     2    -3     4     5


ans =

     1     4     9    16    25
```

```
A=[1 2;3 4]
A.^3
```

```
A =

     1     2
     3     4


ans =

     1     8
    27    64
```

Operazioni elemento per elemento

.*	*moltiplicazione tra array (dimensioni omologhe)*	***A.*B***
./	*divisione tra array (dimensioni omologhe)*	***A./B***
.^	*elevamento a potenza tra array (dimensioni omologhe)*	***A.^B***

*Le operazioni **elemento per elemento** fra array vengono svolte tra gli elementi che occupano posizioni omologhe.*

```
A=[1 2;3 4]
B=[3 2;2 1]
```

```
P1=A.*B
```
Prodotto elemento per elemento

```
P2=A*B
```
Prodotto standard

```
A =

     1     2
     3     4

B =

     3     2
     2     1
```

```
P1 =

     3     4
     6     4

P2 =

     7     4
    17    10
```

Attenzione alla differenza fra prodotto elemento per elemento e prodotto standard

Uso combinato fra funzione di vettori e operatori elemento per elemento

Es. a partire da un vettore generico x=[x1, x2, …, xn] determinare il vettore z la cui generica componente z_i ha la seguente espressione

$$z_i = \frac{\sqrt{|x_i|}}{x_i}$$

```
x=1:10;
z=sqrt(abs(x))./x
```

```
z =

  Columns 1 through 6

    1.0000    0.7071    0.5774    0.5000    0.4472    0.4082

  Columns 7 through 10

    0.3780    0.3536    0.3333    0.3162
```

NUMERI COMPLESSI

abs(x)	*calcola il valore assoluto di x se x è reale, ed il suo modulo se x è un numero complesso*
angle(x)	*calcola la fase di un numero complesso*
conj(x)	*calcola il numero complesso coniugato di x*
imag(x)	*restituisce la parte immaginaria di un numero complesso x*
real(x)	*restituisce la parte reale di un numero complesso x*

Esempi

```
z1=1+2i
```

Definizione di un numero complesso

```
z1 =

   1.0000 + 2.0000i
```

```
modulo=abs(z1)
```

Calcolo del modulo

```
modulo =

    2.2361
```

```
fase=angle(z1)
```

Calcolo della fase (in radianti)

```
fase =

    1.1071
```

NUMERI COMPLESSI

abs(x)	*calcola il valore assoluto di x se x è reale, ed il suo modulo se x è un numero complesso*
angle(x)	*calcola la fase di un numero complesso*
conj(x)	*calcola il numero complesso coniugato di x*
imag(x)	*restituisce la parte immaginaria di un numero complesso x*
real(x)	*restituisce la parte reale di un numero complesso x*

Esempi

```
z1cc=  conj(z1)
```

Calcolo del complesso coniugato

```
z1cc =

   1.0000 - 2.0000i
```

```
RE=real(z1)
IM=imag(z1)
```

Parti reale e immaginarie

```
RE =

     1


IM =

     2
```

Prodotto e divisione fra numeri complessi si calcolano con gli operatori standard di prodotto e divisione

APPROSSIMAZIONI

ceil(x) approssima x al numero intero più vicino verso infinito
fix(x) approssima x al numero intero più vicino verso lo zero
floor(x) approssima x al numero intero più vicino verso - infinito
round(x) approssima x al numero intero più vicino
sign(x) calcola il segno di x e restituendo 0 se x = 0, 1 se x > 0, -1 se x < 0

Esempi

```
x=1.23
CEIL=ceil(x)
FIX=fix(x)
FLOOR=floor(x)
ROUND=round(x)
SEGNO=sign(x)
```

```
x =

    1.2300


CEIL =

     2


FIX =

     1


FLOOR =

     1


ROUND =

     1
```

Istruzioni utili per gli script

`return` Interrompe l'esecuzione dello script.
Spesso inserito in costrutti condizionali IF

`pause` Interrompe l'esecuzione dello script finché non viene premuto un tasto qualunque

CTLR + C Digitato durante l'esecuzione di uno script, ne interrompe l'esecuzione.
Utile quando ad esempio si compie un errore di programmazione che instaura una condizione di stallo o di loop infinito per il programma.

Polinomi

Rappresentazione di polinomi per mezzo di **vettori**

Un polinomio di grado n è rappresentato da un vettore di dimensione n+1 che contiene tutti i coefficienti del polinomio (anche quelli eventualmente nulli) da quello del termine di grado piu elevato fino a giungere al termine noto (ultima componente del vettore)

$$p1(x) = 2x^4 + 7x^3 + x^2 - 4x + 2 \qquad vp1 = \begin{bmatrix} 2 & 7 & 1 & -4 & 2 \end{bmatrix}$$

$$p2(x) = 3x^3 + 1 \qquad vp2 = \begin{bmatrix} 3 & 0 & 0 & 1 \end{bmatrix}$$

Radici di un polinomio

Funzione "**roots**"

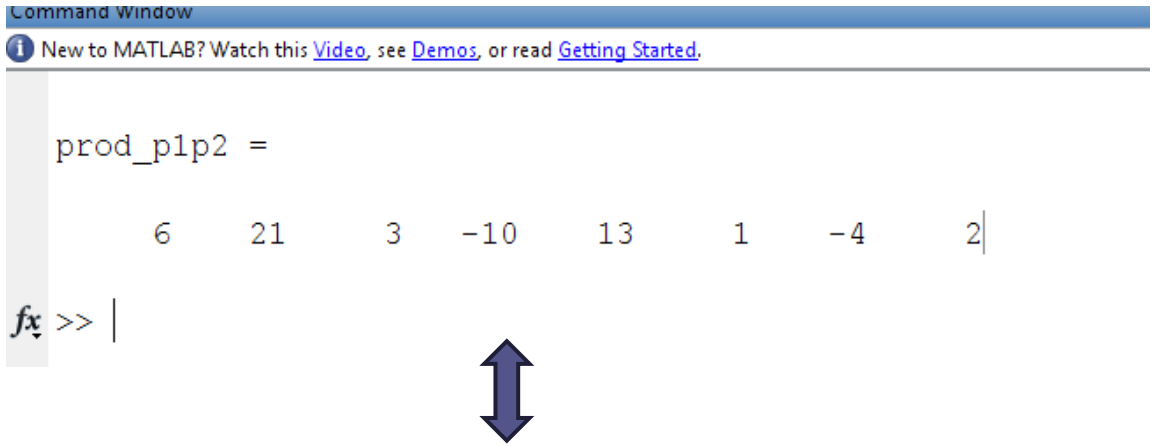
```
clear all
clc
p1=[2 7 1 -4 2];
p2=[3 0 0 1];
rad_p1=roots(p1)
```

```
rad_p1 =

  -3.0962
  -1.2285
   0.4124 + 0.3047i
   0.4124 - 0.3047i
```

Prodotto tra polinomi

Funzione "**conv**"

```
clear all
clc
p1=[2 7 1 -4 2];
p2=[3 0 0 1];
prod_p1p2=conv(p1,p2)
```

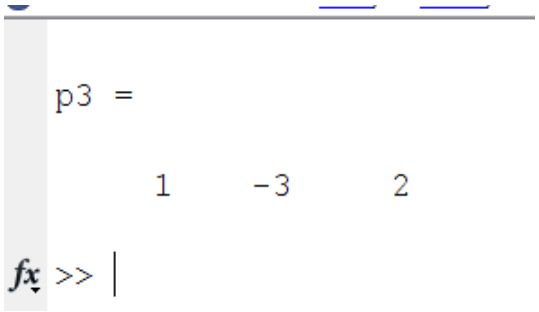


$$P(x) = 6x^7 + 21x^6 + 3x^5 - 10x^4 + 13x^3 + x^2 - 4x + 2$$

Polinomio di radici assegnate (Funzione "**Poly**")

$$p3 = poly([z1 \quad z2 \quad \ldots \quad zn]) \rightarrow p3(x) = (x - z1)(x - z2)\ldots(x - zn)$$

```
clear all
clc
p3=poly([1 2])
```

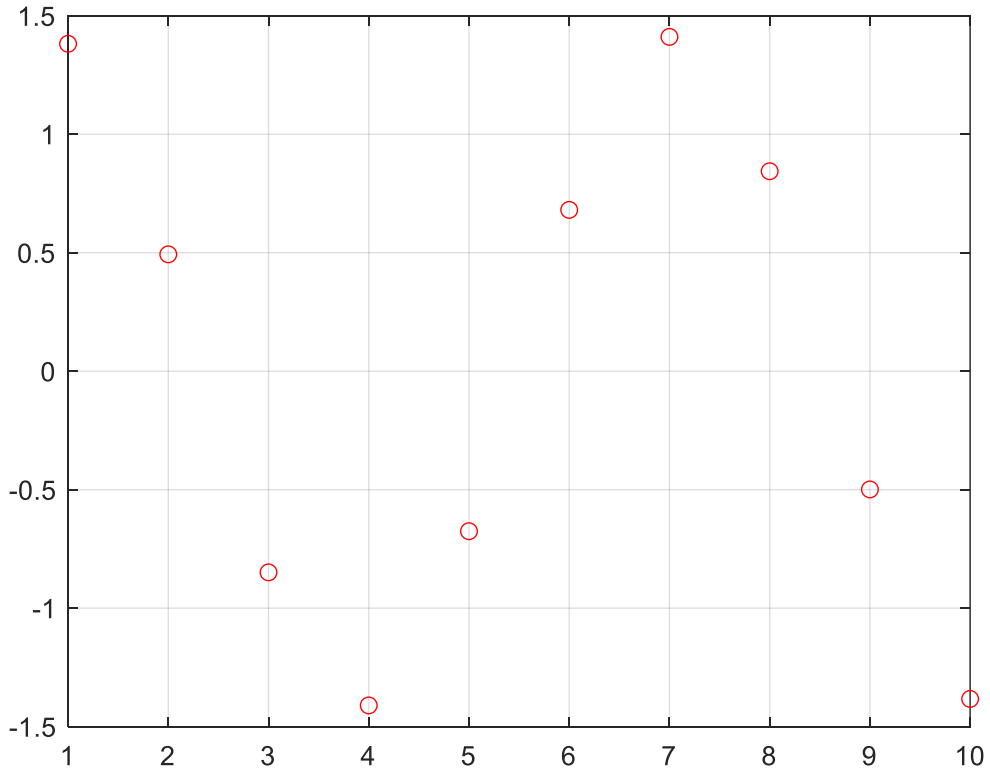
Polynomial fitting

Una esigenza frequente nell'ingegneria è quella di interpolare dei dati acquisiti sperimentalmente mediante una curva che «sposi» al meglio i dati. Vediamo come risolvere tale problema mediante curve approssimanti polinomiali.

Generiamo fittiziamente dei dati da interpolare, campionando una funzione armonica in 10 istanti differenti

```
t = linspace(1,10,10);
y = sin(t)+cos(t);
```

```
plot(t,y,'ro')
axis([0 10 -1.2 1.2])
grid
```



Si deve usare la funzione `polyfit` . I primi due parametri da passare in ingresso alla funzione sono i vettori che contengono gli istanti di acquisizione ed i dati acquisiti. Il terzo parametro è il grado del polinomio che si ricerca

```
p = polyfit(t,y,6)
```

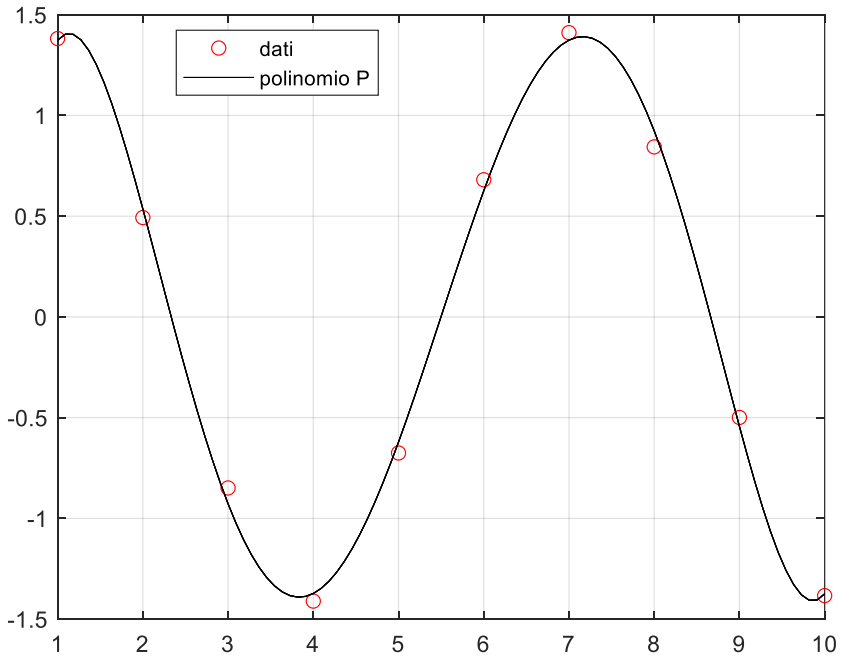
```
p =

    0.0049   -0.1345    1.3013   -5.1982    7.5145   -2.1144
```

Il polinomio interpolante restituito dalla funzione è:

$$P(t) = 0.0047t^5 - 0.1345t^4 + 1.3013t^3 - 5.1982t^2 + 7.5145t - 2.3134$$

Sovrapponiamo al grafico precedente la curva polinomiale calcolata per indagare il grado di sovrapposizione con i dati

```
t1 = linspace(1,10,100);
y1 = polyval(p,t1);
hold on
plot(t1,y1,'k')
hold off
```



Analisi dei dati.

Operazioni fondamentali

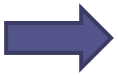
max	Componente massima.
min	Componente minima.
mean	Valor medio.
std	Deviazione standard.
sort	ordina
sum	Somma degli elementi.
prod	Prodotto degli elementi.

Vediamo alcuni esempi di impiego di tali funzioni applicate a vettori

```
v=[1 2 0 3.5 5 2];
x1=max(v)
x2=min(v)
x3=mean(v)
X4=std(v)
```

```
x1 =
     5

x2 =
     0

x3 =
    2.2500

x4 =
    1.7819
```

```
v=[1 2 0 3.5 5 2];
x5=sum(v)
x6=prod(v)
```



```
x5 =
   13.5000

x6 =
     0
```

```
v=[1 2 0 3.5 5 2];
x9=sort(v)
x10=sort(v,'descend')
```

```
x9 =

         0    1.0000    2.0000    2.0000    3.5000    5.0000


x10 =

    5.0000    3.5000    2.0000    2.0000    1.0000         0
```

Le funzioni `max` e `min` possono anche restituire, in aggiunta all'elemento massimo o minimo, la posizione all'interno del vettore di tale elemento.
Tale funzionalità risulta molto utile, e la impiegheremo in vari futuri esempi.

```
v=[1 2 0 3.5 7.5 7.5]
[massimo posiz_massimo]=max(v)
[minimo posiz_minimo]=min(v)
```

```
v =

    1.0000    2.0000         0    3.5000    7.5000    7.5000
```

```
massimo =

    7.5000


posiz_massimo =

     5
```

```
minimo =

     0


posiz_minimo =

     3
```

Quando tali funzioni operano su matrici anziché su vettori restituiscono i rispettivi valori di uscita riferiti alle singole colonne della matrice

```
v=[1 2 0 3.5 5 2;0 1 4 7 2 0.5;4 1 3 2 9 6.5]
x1=max(v)
x2=min(v)
x3=sum(v)
```

```
v =

    1.0000    2.0000         0    3.5000    5.0000    2.0000
         0    1.0000    4.0000    7.0000    2.0000    0.5000
    4.0000    1.0000    3.0000    2.0000    9.0000    6.5000


x1 =

    4.0000    2.0000    4.0000    7.0000    9.0000    6.5000


x2 =

         0    1.0000         0    2.0000    2.0000    0.5000


x3 =

    5.0000    4.0000    7.0000   12.5000   16.0000    9.0000
```

Esercizio

Scrivere uno script che richieda all'utente l'inserimento da tastiera di un numero intero n, e successivamente:

- crei una matrice M quadrata di dimensione n di numeri casuali
- ne calcoli la «parte simmetrica» $S = (M + M^T)/2$
- valuti l'elemento più grande dell'ultima riga di S
- calcoli gli autovalori di S e ne determini il piu piccolo

Input interattivo

```
n=input('Inserire un numero intero n: \n')
```

L'esito della precedente istruzione è che compare nel prompt la stringa `Inserire un numero intero n` ed il numero che viene digitato dall'utente viene salvato nel workspace nella variabile `n`

Soluzione

Scrivere uno script che richieda all'utente l'inserimento da tastiera di un numero intero n, e successivamente:
- crei una matrice M quadrata di dimensione n di numeri casuali
- ne calcoli la «parte simmetrica» $S = (M + M^T)/2$
- valuti l'elemento più grande dell'ultima riga di S
- calcoli gli autovalori di S e ne determini il più piccolo

```
n=input('Inserire un numero intero n: \n')

M=rand(n);
S=(M+M')/2

ultimarigaS=S(n,:)
%sintassi alternativa
ultimarigaS=S(end,:)

maxultimariga=max(ultimarigaS)

autovS=eig(S)

minautov=min(autovS)
%sintassi alternativa
minautov=min(eig(S))
```

Operatori relazionali

<	*minore*
<=	*minore o uguale*
>	*maggiore*
>=	*maggiore o uguale*
==	*uguale*
~=	*diverso*

~ → Alt+126

Possono essere applicati fra un array ed uno scalare, oppure fra due array delle medesime dimensioni

Nel primo caso, gli elementi dell'array vengono tutti confrontati con lo scalare.
Nel secondo caso vengono confrontati gli elementi con posizione omologa.
In entrambi i casi, si ottiene un array di valori booleani (0 ed 1).

Frequentemente utilizzati nella condizione di attivazione di costrutti sintattici `if` o `while`, che vedremo in seguito

Vediamo esempi di applicazione delle varie casistiche e operatori.

Confronto fra scalari

```
x=1; y=3;
B1=x<=y
```

```
B1 =

  logical

   1
```

Confronto fra vettore e scalare

```
v=[1 2 3 4 5]
Boolv=v<3
```

```
v =

     1     2     3     4     5


Boolv =

  1×5 logical array

   1   1   0   0   0
```

Confronto fra vettori

```
v1=[1 2 3 6 5]
v2=[1 2 3 8 5]
Boolv2=v1~=v2
```

Istruzione di assegnazione

Operatore Relazionale di diversità

```
v1 =

     1     2     3     6     5


v2 =

     1     2     3     8     5


Boolv2 =

  1×5 logical array

   0   0   0   1   0
```

Confronto fra matrice e scalare

```
M=[1 2 3;4 5 6;7 4 4]
BoolM=M==4
```

Istruzione di assegnazione

Operatore Relazionale di uguaglianza

```
M =

     1     2     3
     4     5     6
     7     4     4


BoolM =

  3×3 logical array

   0   0   0
   1   0   0
   0   1   1
```

Confronto fra matrici

```
A1=[10 5; 1 1]
A2=[2 3;4 5]


BoolA12=A1>A2
```

```
A1 =

    10     5
     1     1


A2 =

     2     3
     4     5
```

```
BoolA12 =

  2×2 logical array

   1   1
   0   0
```

Funzioni logiche

any (x)	*1 se almeno un elemento di x è non nullo, zero altrimenti.*
all (x)	*1 se tutti gli elementi di x sono non nulli, zero altrimenti*
find (x)	*restituisce gli indici degli elementi non nulli del vettore in ingresso*

```
x=[4 1 0 2 6 0]

a1=any(x)
a2=all(x)
a3=find(x)
```

```
a1 =

  logical

   1


a2 =

  logical

   0


a3 =

     1     2     4     5
```

Con input matriciale, `any` ed `all` lavorano separatamente sulle colonne, mentre `find` restituisce le posizioni di tutti gli elementi non nulli con una notazione «particolare»

```
x=[11 12 13;4 0 5; 9 8 0]
a1=any(x)
a2=all(x)
a3=find(x)
```

```
x =

    11    12    13
     4     0     5
     9     8     0
```

```
a1 =

  1×3 logical array

   1   1   1
```

```
a2 =

  1×3 logical array

   1   0   0
```

```
a3 =

     1
     2
     3
     4
     6
     7
     8
```

Tempo [s]	Pressione [bar]
0	62.3800
0.1000	62.3900
0.2000	62.4000
0.3000	62.4000
0.4000	62.4100
0.5000	62.4200
0.6000	62.4300
0.7000	62.4300
0.8000	62.4400
0.9000	62.4500
1	62.4600
1.1000	62.4700
1.2000	62.4800
1.3000	0
1.4000	62.4900
1.5000	62.5000
1.6000	62.5100
1.7000	62.5200
1.8000	62.5300
1.9000	62.5400
2	62.5500
2.1000	62.5600
2.2000	62.5700
2.3000	62.5800
2.4000	62.5800
2.5000	0
2.6000	62.6000
2.7000	62.6100
2.8000	62.6200
2.9000	62.6300
3	62.6400

Rimozione da uno stream di dati al di sotto di un valore di soglia

Carichiamo nel workspace, con il comando

```
load stream
```

il file stream.mat

Esso provoca il salvataggio nel workspace di due vettori, time e data che contengono rispettivamente istanti di acquisizione e misure acquisite (misure di pressione acquisite ogni decimo di secondo per 3 secondi, quindi 31 acquisizioni)

Si notino due misure nulle (poste in 14esima e 26esima posizione), che denotano un errore di acquisizione.

Desideriamo rimuovere dallo stream le misure inferiori a 10 bar che ove andassi a processare i dati (ad esempio per estrarne il valore minimo) causerebbero errori.

```
clear all, clc
load stream

figure(1)
plot(time,data),grid
title('Dati grezzi')
```


```
soglia=10;
z1=data<soglia;
```

z1 è un vettore di elementi booleani, che contiene 1 in corrispondenza dei dati sottosoglia (il 14 esimo ed il 26 esimo), e zero altrove

```
z1 =

  31×1 logical array

   0
   0
   0
   0
   0
   0
   0
   0
   0
   0
   0
   0
   1
   0
   ⋮
   1
   0
   0
   0
   0
   0
```

Con la funzione find estraiamo gli indici associati alle posizioni di tali misure «sottosoglia».

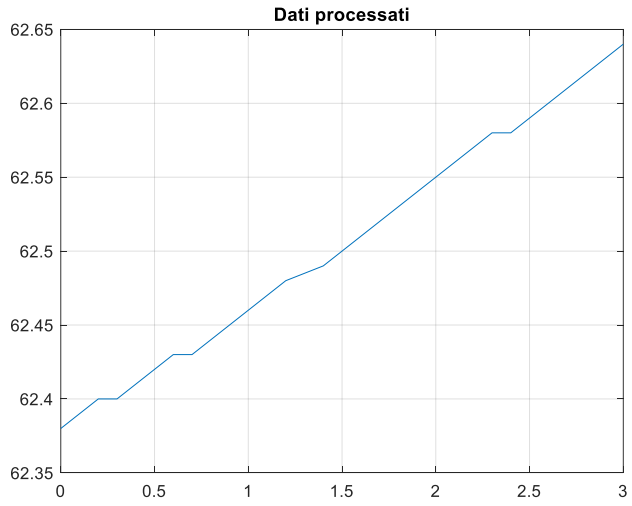
```
indici=find(z1)
```

```
indici =

    14
    26
```

```
data(indici)=[];
time(indici)=[];
```

Queste due istruzioni, secondo quanto visto in precedenza, rimuovono dai vettori `data` e `time` la 14esima e la 26esima componente. Ora i vettori hanno quindi dimensione 29, non più 31.

```
figure(2)
plot(time,data),grid
title('Dati processati')
```



Il grafico dei dati processati mostra come le misure «spurie» siano state rimosse dai dati.