UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA INDUSTRIALE

DIPARTIMENTO MATEMATICA

DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"

Laboratorio di Calcolo Numerico LAB 9

Matrici in Matlab

ripasso e alcuni comandi utili per i sistemi lineari

Docenti: E. Bachini, L. Bruni

Email: elena.bachini@unipd.it Email: bruni@math.unipd.it

8 maggio 2024

Outline

1. Ripasso sulle matrici (si vedano anche slides Lab.4)
2. Norme, condizionamento, inversa, determinante
3. Risoluzione di sistemi lineari
4. Lettura e scrittura facile di file di testo

Ripasso sulle matrici (si vedano anche slides Lab.4)

Comandi per la creazione e modifica di matrici I

- `zeros(M,N)` crea matrice nulla con dimensione MxN
- `zeros(M)` crea matrice nulla con dimensione MxM
- `ones(M,N)` crea matrice di dimensione MxN con componenti 1
- `ones(M)` crea matrice di dimensione MxM con componenti 1
- `eye(M,N)` crea matrice di dimensione MxN con componenti al di fuori della diagonale principale tutte nulle, e componenti uguali ad 1 sulla diagonale principale.
- `eye(M)` crea la matrice identica o unità di dimensione MxM.
- `diag(v,k)` dove `v` è un vettore (riga [1,N] o colonna [N,1]) crea una matrice quadrata diagonale con `k`-esima diagonale uguale a `v`. La dimensione sarà (N+|k|) x (N+|k|) (si ricordi che la diagonale principale corrisponde a `k`=0.

Comandi per la creazione e modifica di matrici II

- `diag(A,k)` dove `A` è una matrice, restituisce un vettore colonna estraendo la diagonale indicata.
- `triu(A,k)` (`k`$\in Z$ opzionale, default `k`=0) crea matrice triangolare superiore estraendo da `A` la parte dalla k-esima diagonale "in su". Attenzione alla dimensione del risultato.
- `tril(A,k)` (`k`$\in Z$ opzionale, default `k`=0) crea matrice triangolare inferiore estraendo da `A` la parte dalla k-esima diagonale "in giù"). Attenzione alla dimensione del risultato.
- `fliplr(A)` riordina le colonne di `A` prendendole da destra a sinistra.
- `fliplr(v)` se `v` è un vettore riga restituisce un vettore riga con le componenti ordinate dall'ultima alla prima; se `v` è un vettore colonna restituisce `v` stesso.
- `flipud(A)` riordina le righe di `A` prendendole dal basso in alto.

Comandi per la creazione e modifica di matrici III

- `flipud(v)` se `v` è un vettore riga restituisce `v` stesso; se `v` è un vettore colonna restituisce un vettore colonna con le componenti ordinate dall'ultima alla prima.
- `reshape(v,M,N)` crea una matrice di dimensione MxN utilizzando le componenti del vettore `v` (riga o colonna) ed inserendole per colonne. Attenzione che in numero di componenti di `v` deve essere M x N.
- `repmat(A,M,N)` crea matrice ottenuta replicando M volte in riga ed N volte in colonna il blocco A.
- `magic(k)` crea matrice quadrata di dimensione k$\times$k con righe e colonne intere, a somma costante.
- `rand(M,N)` matrice MxN di numeri pseudocasuali con distribuzione uniforme in $[0, 1]$

Alcuni esempi I

```
>> A = eye(3) + diag([2  3],1) % somma due matrici 3 x 3
A =
     1     2     0
     0     1     3
     0     0     1

>> A = A - tril(A)
A =
     0     2     0
     0     0     3
     0     0     0

>> flipud(A)
ans =
     0     0     0
     0     0     3
     0     2     0

>> fliplr(A)
ans =
     0     2     0
     3     0     0
     0     0     0
```

Alcuni esempi II

```
>> v = A(:)'  % v vettore colonna che contiene i 9 elementi di A
v =
     0     0     0     2     0     0     0     3     0

>> B = reshape(v,3,3)  % Costruiamo una matrice con il vettore v
B =
     0     2     0
     0     0     3
     0     0     0
```

```
>> u = [1,2]; B = diag(u)
B =
     1     0
     0     2

>> A = repmat(B,2,3)
A =
     1     0     1     0     1     0
     0     2     0     2     0     2
     1     0     1     0     1     0
     0     2     0     2     0     2
```

Comandi per matrici speciali

- matrice di Hilbert `hilb(n)` (matrice quadrata `n` $\times$ `n`)
- matrice di Pascal `pascal(n)` (matrice quadrata `n` $\times$ `n`)
- matrice di Vandermonde `vander(v)`, matrice quadrata di dimensione `n` $\times$ `n`, con `n` numero delle componenti del vettore (riga o colonna) `v`, avente come colonne, a partire dall'ultima alla prima, le potenze componente per componente (operatore puntuale) del vettore `v`, ovvero `v.^k`, per `k` $= 0, \ldots,$ `n-1`.
- matrice di Toeplitz `toeplitz(c,r)`, avente come prima colonna il vettore `c` e prima riga il vettore `r`
- matrice di Hankel `hankel(c,r)`, avente come prima colonna il vettore `c` e ultima riga il vettore `r`

Esempi di matrici speciali

```
>> hilb(3)
ans =

    1.0000    0.5000    0.3333
    0.5000    0.3333    0.2500
    0.3333    0.2500    0.2000

>> pascal(3)
ans =
     1     1     1
     1     2     3
     1     3     6

>> vander([1  2  3])
ans =
     1     1     1
     4     2     1
     9     3     1

>> toeplitz([1  2],[1  20  30]) %prima componente = prima componente
ans =
     1    20    30
     2     1    20

>> hankel([1  2],[2  20  30]) % ultima componente = prima componente
ans =
     1     2    20
     2    20    30
```

Dimensioni

- `size(A)` con `A` matrice, restituisce un vettore le cui componenti sono le dimensioni di `A`. E' possibile ottenere solo il valore della k-esima dimensione con il comando `size(A,k)`.
- `size(v)` con `v` vettore, restituisce un vettore di due componenti, una delle quali è sempre 1.
- `length(A)` con `A` matrice, non restituisce il numero totale di elementi di `A`, ma restituisce il valore della dimensione massima.
- `length(v)` con `v` vettore (riga o colonna), restituisce il numero di componenti del vettore.

Esempi di dimensioni

```
>> A = [1 2 3; 4 5 6]
A =
     1     2     3
     4     5     6

>> v = size(A)
v =
     2     3

>> size(v)
ans =
     1     2

>> length(v)
ans =
     2

>> length(A)
ans =
     3

>> length(A(:))
ans =
     6
```

Alcune funzioni su array I

- `sum(A)` Se `A` è una matrice, anche `logical`, restituisce un vettore riga con le somme degli elementi di ogni colonna.
  Se `A` è un vettore (riga o colonna), restituisce lo scalare corrispondente alla somma delle componenti.

```
>> A=magic(3)
A =
     8     1     6
     3     5     7
     4     9     2
>> sum(A)              % somma per colonne
ans =
    15    15    15
>> sum(sum(A))
ans =
    45
```

- `prod(A)` Opera come `sum(A)`.

```
>> prod(A)
ans =
    96    45    84
```

Alcune funzioni su array II

- `any(A)` Funzione logica. Se `A` è una matrice, restituisce un vettore riga `logical` le cui componenti sono la costante logica vero (=1) se almeno un elemento della colonna corrispondente di `A` è non nullo. Se è un vettore, restituisce il valore vero (=1) se almeno una componente è non nulla.

```
>> A= [ -1 0 3 0; 0 0 1 2; 0 0 2 1]
A =
    -1     0     3     0
     0     0     1     2
     0     0     2     1
>> any(A)
ans =
  1x4 logical array
   1   0   1   1

>> v = zeros(3,1);
>> any(v)
ans =
  logical
   0
```

Alcune funzioni su array III

- `all(A)` Funzione logica. Se `A` è una matrice, restituisce un vettore riga `logical` le cui componenti sono la costante logica vero (=1) se tutti gli elementi della colonna corrispondente di `A` sono non nulli. Se è un vettore, restituisce il valore vero (=1) se tutte le componenti sono non nulle.

```
>> A= [ -1 0 3 0; 0 0 1 2; 0 0 2 1]
A =
    -1     0     3     0
     0     0     1     2
     0     0     2     1
>> all(A)
ans =
  1x4 logical array
   0   0   1   0

>> v = ones(3,1);
>> all(v)
ans =
  logical
   1
```

Concatenazione semplice di blocchi

Concatenazione "a livello singolo"

Possiamo concatenare in riga, es: `A=[A1,A2]` o in colonna, es: `A=[A1;A2]`, o in modo multiplo (i.e., concatenare in colonna righe definite per concatenazione di blocchi, es, `A=[A11,A12; A21,A22,A23]`), solo se le dimensioni sono consistenti, ovvero:

- Se concateniamo in riga `A=[A1,A2]` dobbiamo avere `size(A1,1)==size(A2,1)`
- Se concateniamo in colonna `A=[A1;A2]` dobbiamo avere `size(A1,2)==size(A2,2)`
- Se concateniamo in modo multiplo, ciascuna riga deve essere definita rispettando la compatibilità e poi concatenata in colonna rispettando la compatibilità in colonna.

Esempio e controesempio

Le linee sono state aggiunte per chiarire il risultato dell'esempio.

```
A = [zeros(2,2), ones(2,3); ones(2,3), zeros(2,2)]


     0     0  |  1     1     1
     0     0  |  1     1     1
A =  ---------------------------
     1     1     1  |  0     0
     1     1     1  |  0     0
```

Errato!

```
A = [zeros(2), ones(3); ones(3), zeros(2)]

Error using horzcat
Dimensions of arrays being concatenated are not consistent.
```

Norme, condizionamento, inversa, determinante

Norme di vettori I

Sia $X = \mathbb{R}^n$, ed $\mathbf{x}$ un vettore costituito da n componenti, ossia $\mathbf{x} = (x_1, \ldots, x_n)^T$. Le norme di vettore più usate sono le **norme di Hölder** o **norme** p, con $1 \leq p \leq \infty$, così definite:

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}}$$

- `norm(x)` oppure `norm(x,2)` calcola la norma 2 o euclidea di x (default)

$$\|\mathbf{x}\|_2 = \sqrt{|x_1|^2 + \cdots + |x_n|^2} = \left(\sum_{i=1}^{n} |x_i|^2 \right)^{\frac{1}{2}}$$

Norme di vettori II

Si noti che $\|\mathbf{x}\|_2^2 = (\mathbf{x}, \mathbf{x})$, dove $(\cdot, \cdot)$ indica il prodotto scalare tra due vettori

- `norm(x,1)` calcola la norma 1 (o della convergenza in media):

$$\|\mathbf{x}\|_1 = |x_1| + \cdots + |x_n| = \sum_{i=1}^{n} |x_i|$$

- `norm(x,Inf)` calcola la norma infinito (o norma del massimo o della convergenza assoluta):

$$\|\mathbf{x}\|_\infty = \max_{1 \le i \le n} |x_i|$$

- `norm(x,-Inf)` come `Inf`, ma calcola il minimo.

Norme di matrice I

Assegnata una norma di vettore $\|\cdot\|_p$ si definisce norma naturale di una matrice $A \in \mathbb{R}^{m\times n}$ (o norma indotta dalla norma di vettore)

$$\|A\|_p := \max_{\mathbf{x}\neq 0} \frac{\|A\mathbf{x}\|_p}{\|\mathbf{x}\|_p}.$$

- `norm(A)` o `norm(A,2)` calcola la norma 2 (o spettrale) di A:

$$\|A\|_2 = \sqrt{\rho(A^T A)},$$

dove $\rho(A^T A)$ rappresenta l'autovalore massimo in modulo della matrice quadrata $A^T A \in \mathbb{R}^{n\times n}$ (default).

Norme di matrice II

- `norm(A,1)` calcola la norma 1 (o del massimo tra le somme per colonne):

$$\|A\|_1 = \max_{1 \le j \le n} \sum_{i=1}^{m} |a_{ij}|$$

- `norm(A,Inf)` calcola la norma infinito (o del massimo tra le somme per righe):

$$\|A\|_\infty = \max_{1 \le i \le m} \sum_{j=1}^{n} |a_{ij}|$$

Numero di condizionamento I

Il numero di condizionamento $\kappa_p(A)$ di una matrice quadrata invertibile A (rispetto ad una norma data $\|\cdot\|_p$) misura il malcondizionamento del sistema lineare $Ax = b$ ed è definito come

$$\kappa_p(A) := \|A\|_p \, \|A^{-1}\|_p$$

Numeri di condizionamento *elevati* indicano che la matrice è quasi singolare.

- `cond(A)` o `cond(A,2)` calcola il numero di condizionamento rispetto alla norma 2 (default).
- `cond(A,p)`, con $p = 1$ o $p =$ `Inf` , calcola il numero di condizionamento rispetto alla norma p

Numero di condizionamento II

- `rcond(A)` calcola una stima del reciproco del numero di condizionamento rispetto alla norma 1. Se A è ben condizionata, è vicino ad 1; se A è malcondizionata, è vicino ad eps (stabile ed efficiente).
- `condest(A)` calcola una stima del numero di condizionamento rispetto alla norma 1. Calcola una stima di $\|A^{-1}\|_1$ (senza calcolare l'inversa). (stabile ed efficiente).

```
>> A = hilb(20);
>> cond(A,1)
ans =
   7.6351e+18
>> condest(A)
ans =
   7.3999e+18
>> rcond(A)      % eps é circa 2.2204e-16
ans =
   1.3514e-19
>> B = rand(20); cond(B,1)
ans =
  456.6529
```

Matrice inversa e determinante I

`inv(A)` calcola l'inversa della matrice A, se A è una matrice quadrata invertibile.
Tale comando deve essere utilizzato con molta cautela. Infatti esso calcola l'inversa effettuando prima una fattorizzazione LU (oppure una fattorizzazione LDL^T se la matrice è simmetrice) e poi risolve i sistemi lineari per trovare la soluzione che corrisponde all'inversa. Pertanto NON può essere utilizzato per matrici di grandi dimensioni!

`det(A)` calcola il determinante della matrice A, se A è una matrice quadrata, effettuando una fattorizzazione LU con pivoting.
Valgono le stesse cautele del comando `inv`: NON può essere utilizzato per matrici di grandi dimensioni! Il suo calcolo può essere affetto da instabilità numerica. Inoltre per matrici singolari, può accadere che dia un risultato grande anche se in realtà dovrebbe valere zero.

Risoluzione di sistemi lineari

Backslash

Dato un sistema $A\mathbf{x} = \mathbf{b}$, Matlab prevede un solutore "general purposes" detto `backslash` che si può utilizzare equivalentemente con le seguenti sintassi:

```
x = A\b;
x = mldivide(A,b);
```

Attenzione:

Il comando backslash è di delicatissimo utilizzo. Se da un lato ha una elevata flessibilità (si usa anche per risolvere problemi che sono generalizzazioni dei sistemi lineari), si deve tener presente che è una sorta di black box su cui si ha scarsissimo controllo su cosa si stia in realtà facendo. Infatti gli algoritmi che utilizza vengono scelti da Matlab in base alle caratteristiche di A (si usi il comando `doc mldivide` e si veda la sezione **Algorithms**).

Fattorizzazione LU di Gauss I

Il Matlab non prevede alcun comando per la fattorizzazione LU senza pivoting! Effettua sempre, per ragioni di stabilità numerica, la fattorizzazione con pivoting parziale.

- `[L,U,P] = lu(A)` effettua una fattorizzazione con pivoting parziale e restituisce le matrici L, U e P tali che
$$PA = LU, \text{ ovvero } A = P^T LU$$
  - L è una triangolare inferiore a diagonale unitaria;
  - U è triangolare superiore;
  - P è la matrice di permutazione.
- `[L1,U] = lu(A)` effettua una fattorizzazione con pivoting parziale ma restituisce solo le matrici L_1 e U tali che
$$A = L_1 U$$
  - L_1 non è una triangolare inferiore, bensì una permutazione di una matrice triangolare inferiore a diagonale unitaria (è uguale a $P^T \times L$ con L e P ottenute con il comando precedente);

Fattorizzazione LU di Gauss II

- U è triangolare superiore (la stessa del comando precedente).

Le matrici di permutazione sono matrici ortogonali ($P^{-1} = P^T$).

```
>> A = magic(3);
>> [L,U,P] = lu(A)
L =
    1.0000         0         0
    0.5000    1.0000         0
    0.3750    0.5441    1.0000
U =
    8.0000    1.0000    6.0000
         0    8.5000   -1.0000
         0         0    5.2941
P =
     1     0     0
     0     0     1
     0     1     0
>> [L1,U1] = lu(A)
L1 =
    1.0000         0         0
    0.3750    0.5441    1.0000
    0.5000    1.0000         0
U1 =
    8.0000    1.0000    6.0000
         0    8.5000   -1.0000
```

Fattorizzazione LU di Gauss III

```
         0         0    5.2941
```

Fattorizzazione LL^T di Cholesky I

N.B. I comandi NON controllano che la matrice A sia simmetrica!

- `[L,flag] = chol(A,'lower')` se `flag` $= 0$, L rappresenta la matrice triangolare inferiore tale che $A = LL^T$ e la matrice A è definita positiva; se `flag` $\neq 0$ la matrice NON è definita positiva e `flag` indica l'indice di colonna in cui si è dovuto fermare. La fattorizzazione viene effettuata considerando solo la parte inferiore di A.
- `[R,flag] = chol(A)`
  oppure `[R,flag] = chol(A,'upper')` se `flag` $= 0$, R rappresenta la matrice triangolare superiore tale che $A = R^T R$ e la matrice A è definita positiva; se `flag` $\neq 0$ la matrice NON è definita positiva e `flag` indica l'indice di riga in cui si è dovuto fermare. La fattorizzazione viene effettuata considerando solo la parte superiore di A.

Fattorizzazione LL^T di Cholesky II

```
>> A=pascal(3) % pascal matrice simmetrica, definita positiva
A =
     1     1     1
     1     2     3
     1     3     6
>> [L,flag] = chol(A,'lower');
>> flag
flag =
     0

>> A(3,3) = 5  % modifico un elemento diagonale
A =
     1     1     1
     1     2     3
     1     3     5
>> [L,flag] = chol(A,'lower');
>> flag        % la matrice NON é definita positiva
flag =
     3
>> L           % L non é stata completata
L =
     1     0
     1     1
```

Fattorizzazione LL^T di Cholesky III

```
>> A = magic(3); A = A'*A      % A simmetrica, definita positiva
A =
    89    59    77
    59   107    59
    77    59    89

>> A(1,3) = 0                  % la rendo NON simmetrica
A =
    89    59     0
    59   107    59
    77    59    89
>> [L,flag] = chol(A,'lower') % fattorizzata con successo!
L =
    9.4340         0         0
    6.2540    8.2394         0
    8.1620    0.9655    4.6314
flag =
     0
```

Soluzione di sistemi lineari con fattorizzazione di Gauss

Risolvere un sistema $Ax = b$ con A quadrata invertibile.

Gauss con Pivoting

- calcolo della fattorizzazione $PA = LU$ (Gauss con pivoting);
- Sistema 1: soluzione di $Ly = Pb$ tramite sostituzioni in avanti;
- Sistema 2: soluzione di $Ux = y$ tramite sostituzioni all'indietro.

```
>> A = magic(3);
>> [L U P] = lu(A);
>> b = A*[1;1;1]; % calcolo b in modo che x sia (1,1,1)^T
>> y = L\(P*b);   % Sistema 1 triang. inferiore
>> x = U\y        % Sistema 2 triang. superiore
x = 1.0000
    1.0000
    1.0000
>> det(A)
ans =
  -360
>> det(P)
ans =
    -1
>> det(U)
ans =
   360
```

Soluzione di sistemi lineari con fattorizzazione di Cholesky

Cholesky

- A deve essere simmetrica e con elementi diagonali positivi;
- calcolo della possibile fattorizzazione $A = LL^T$. Se esiste:
  - Sistema 1: soluzione di $Ly = b$ tramite sostituzioni in avanti
  - Sistema 2: soluzione di $L^T x = y$ tramite sostituzioni all'indietro

```
>> A=pascal(3) % pascal matrice simmetrica, definita positiva
A =
     1     1     1
     1     2     3
     1     3     6
>> [L,flag] = chol(A,'lower');
>> flag
flag =
     0
>> b = A*[1;1;1]; % calcolo b in modo che x sia (1,1,1)^T
>> y =L\b;        % Sistema 1 triang. inferiore
>> x = L'\y       % Sistema 2 triang. superiore
x =
     1
     1
     1
```

Lettura e scrittura facile di file di testo

Salvare dati con save I

Abbiamo visto che il comando `save` viene utilizzato per creare dei file di tipo `.mat`, non leggibili, che contengano la rappresentazione interna dei contenuti di variabili (scalari, vettori, matrici) del Workspace e che possono essere recuperate successivamente con il comando `save`.
Usando opportune opzioni il comando `save` può essere usato anche per creare un file di tipo testo (con estensione consigliata `.txt` o `.dat`) che contenga i corrispondenti valori decimali di variabili (scalari, vettori, matrici).
Può essere utile per salvare rapidamente, senza preoccuparsi di utilizzare formati differenziati, dei dati o dei risultati.
I valori delle variabili indicate vengono messi uno dopo l'altro, rispettando le loro dimensioni.

```
save('nomefile.txt','-ascii','var1','var2','var3')
```

La lista delle variabili può essere più o meno numerosa. `'-ascii'` è l'opzione che serve a generare il file di tipo testo.

Salvare dati con save II

```
>> A= magic(4)                % Matrice 4 x 4
A =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
>> b = [1 2 3]                % Vettore riga 1 x 3
b =
     1     2     3
>> x = [10;20];               % Vettore colonna 2 x 1
>> a = 1.2345678912345 67; % Scalare con 16 cifre decimali
>> save('prova.txt','-ascii','A','b','x','a')
>> type prova.txt
   1.6000000e+01   2.0000000e+00   3.0000000e+00   1.3000000e+01
   5.0000000e+00   1.1000000e+01   1.0000000e+01   8.0000000e+00
   9.0000000e+00   7.0000000e+00   6.0000000e+00   1.2000000e+01
   4.0000000e+00   1.4000000e+01   1.5000000e+01   1.0000000e+00
   1.0000000e+00   2.0000000e+00   3.0000000e+00
   1.0000000e+01
   2.0000000e+01
   1.2345679e+00
```

Se si desiderano più cifre decimali (formato esponenziale con 17 cifre decimali) bisogna usare l'opzione `'-double'`:

Salvare dati con save III

```
>> save('prova1.txt','-ascii','-double','x','a')
>> type prova1.txt

   1.0000000000000000e+01
   2.0000000000000000e+01
   1.2345678912345670e+00
```

Se si vuole aggiungere valori ad un file esistente, bisogna aggiungere l'opzione '-append':

```
>> y = [10 20];          % Vettore riga 1 x 2
>> save('prova1.txt','-ascii','-double','-append','y')
>> type prova1.txt

   1.0000000000000000e+01
   2.0000000000000000e+01
   1.2345678912345670e+00
   1.0000000000000000e+01   2.0000000000000000e+01
```

Leggere dati su file con `load` I

Se abbiamo dei dati memorizzati in un file di testo (qualunque sia il formato, sempre che si usino le notazioni delle costanti Matlab) possiamo recuperarli facilmente con il comando `load`.

```
nomevar = load('nomefile.txt')
```

Purtroppo con tale comando NON riusciamo a leggere file di testo che NON contengano lo stesso numero di dati in tutte le righe. Ovvero riusciamo a leggere solo delle tabelle di valori numerici.

```
>> type prova1.txt

   1.0000000000000000e+01
   2.0000000000000000e+01
   1.2345678912345670e+00
   1.0000000000000000e+01   2.0000000000000000e+01
>> A = load('prova1.txt');
Error using load
Unable to read file 'prova1.txt'. Input must be a MAT-file or an ASCII
     file containing numeric
data with same number of columns in each row.
```

Leggere dati su file con `load` II

Se non assegnamo il comando ad una variabile, di default crea una variabile il cui nome coincide con il nome del file (senza estensione)

```
>> save('prova2.txt','-ascii','-double','x','a')
>> type prova2.txt

   1.0000000000000000e+01
   2.0000000000000000e+01
   1.2345678912345670e+00
>> load('prova2.txt')
>> prova2
prova2 =
   10.0000
   20.0000
    1.2346
>> z = load('prova2.txt')
z =
   10.0000
   20.0000
    1.2346
```

NON serve sapere quante righe e quante colonne ci sono nella tabella di dati memorizzata nel file di testo!