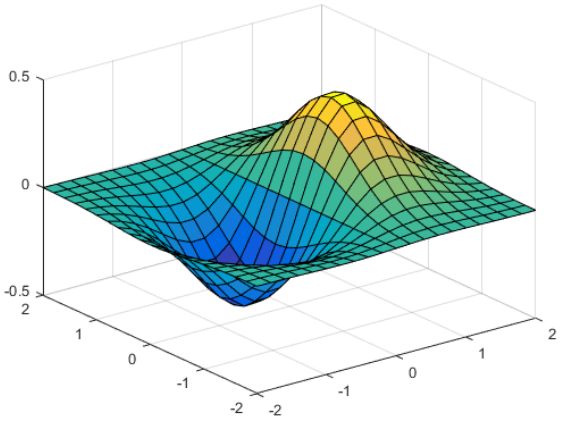
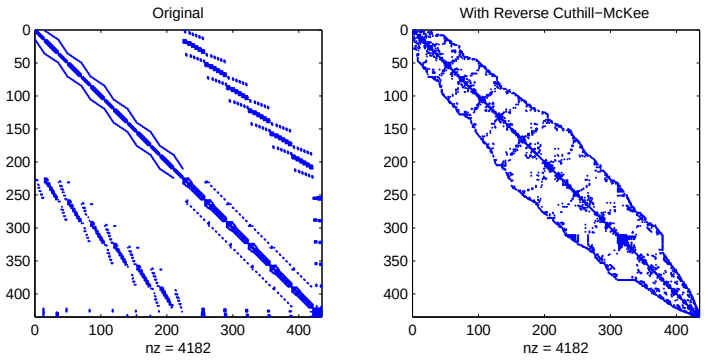
Ángeles Martínez Calomardo

ANALISI NUMERICA CON MATLAB





Università degli Studi di Trieste

1

Introduzione a MATLAB/Octave

In questo capitolo si descriverà l'ambiente MATLAB, che, molto più di un linguaggio di programmazione, è un ambiente integrato per la programmazione e per la visualizzazione. In alternativa è possibile utilizzare GNU Octave (Octave d'ora in poi), un software numerico molto simile a MATLAB distribuito gratuitamente.

MATLAB e Octave sono ambienti integrati per il Calcolo Scientifico e la visualizzazione grafica, a loro volta scritti in linguaggio C e C++.

MATLAB

MATLAB è distribuito da The MathWorks (si veda il sito `www.mathworks.com`). La sua principale caratteristica è la manipolazione di matrici, come viene sottolineato dall'acronimo MATLAB che deriva da MATrix LABoratory, vale a dire "laboratorio matriciale". MATLAB può inoltre esser visto come una calcolatrice scientifica evoluta e come un linguaggio di programmazione ad alto livello.

Per avviare MATLAB in ambiente Unix basta digitare il comando `matlab` seguito dal tasto di invio.

Octave

Anche Octave è un ambiente integrato per il calcolo scientifico e la visualizzazione grafica come MATLAB. A differenza di MATLAB , Octave è distribuito gratuitamente dalla GNU (si veda il sito `www.octave.org`). MATLAB e Octave presentano delle differenze ma sono sufficientemente compatibili da permettere alla maggior parte di programmi MATLAB di essere eseguiti senza modifiche in ambiente Octave e viceversa. Per avviare Octave in ambiente Unix basta digitare il comando `octave` seguito dal tasto di invio. È possibile utilizzare il programma Octave attraverso una interfaccia grafica simile a quella di MATLAB, per esempio attraverso il software `qtoctave`.
L'interfaccia grafica di MATLAB è costituita da 4 ambienti:

- **Workspace.** Una finestra che mostra il contenuto del workspace (variabili memorizzate e loro valore).
- **Current directory.** Una finestra sulla cartella in cui si sta lavorando, che mostra i files presenti nella cartella stessa.

- **Command history.** Contiene una lista di tutti i comandi digitati.
- **Command window.** Finestra nella quale vengono inseriti i comandi.

Command window / Octave Terminal

La `Command window/Octave Terminal` permette di interagire con l'ambiente di calcolo di MATLAB/Octave che si presenta come una linea di comando detta *prompt*. Permette di eseguire programmi (script) presenti in MATLAB/Octave, ma anche programmi costruiti dall'utente usando il linguaggio MATLAB/Octave e salvati su un file di testo con estensione .m (m-file). MATLAB ha il suo proprio editore di testo incorporato. Per invocarlo basta scrivere `edit` seguito del nome del file che si vuole editare. Per eseguire un programma costruito dall'utente occorre scrivere dopo il prompt il nome del file **senza l'estensione .m**. Un programma MATLAB/Octave non deve essere compilato: premuto il tasto enter le istruzioni vengono interpretate.

Per essere eseguiti i programmi devono essere presenti nella cartella di lavoro (current directory). Per capire qual è la cartella attuale esiste il comando `pwd`. Il comando `what` lista i file .m presenti nella cartella di lavoro, mentre il comando `ls` lista tutti i files presenti nella stessa.

Variabili e assegnazione

In MATLAB/Octave non occorre dichiarare le variabili: l'assegnazione coincide con la dichiarazione. Il comando `=` serve per assegnare ad una data variabile un valore numerico o una stringa di caratteri. Ad esempio,

```
>> stringa='Questa e'' una stringa di caratteri'

stringa=

Questa e' una stringa di caratteri
```

```
>> a = 2/3
a =
     0.66667
>> b = 3/2
b =
     1.5000
>> a*b
ans =
     1
```

MATLAB/Octave crea le variabili a e b nel momento in cui viene loro assegnato un valore. Se il risultato di un'espressione non viene assegnato a nessuna variabile definita dall'utente, viene assegnato alla variabile di default `ans`.

Una caratteristica di MATLAB è quella di visualizzare a video il risultato di ogni istruzione. Se si desidera sopprimere l'echo a video del risultato di una istruzione occorre

apporre alla fine della stessa il simbolo `;`. I nomi delle variabili possono essere lunghi al massimo 19 caratteri e devono iniziare con un carattere alfabetico (si faccia attenzione poichè MATLAB distingue tra maiuscole e minuscole, ovvero MATLAB è *w case sensitive*).

Esercizio 1. *Assegnare alla variabile A il valore 1, e scrivere a dopo il prompt. Si osservi che A ed a sono due variabili distinte.*

Il comando `who` permette di sapere quali sono le variabili dell'utente attualmente in memoria. Il comando `whos` ne mostra anche la dimensione e l'occupazione di memoria (numero di bytes).

```
whos
Variables in the current scope:

  Attr Name        Size                     Bytes  Class
  ==== ====        ====                     =====  =====
       A           1x1                          8  double
       a           1x1                          8  double
       ans         1x1                          8  double
       b           1x1                          8  double

Total is 4 elements using 32 bytes
```

Le variabili possono essere cancellate utilizzando il comando `clear`. `clear` seguito del nome di una variabile cancella la variabile, mentre `clear` seguito da enter cancella tutte le variabili del workspace. Ci sono variabili predefinite come l'unità immaginaria i o il numero π, che nonostante la grande flessibilità di MATLAB lo permetta, sarebbe bene non sovrascrivere. Il comando `clc` pulisce la finestra dei comandi (command window) e riporta il cursore in alto a sinistra.

L'istruzione `format`

In MATLAB/Octave tutti i calcoli vengono effettuati utilizzando i numeri in virgola mobile in doppia precisione, secondo lo standard IEEE-754r. L'istruzione `format` permette di modificare il formato di visualizzazione dei risultati ma **non modifica la precisione con cui i calcoli vengono eseguiti**.
I principali formati di visualizzazione dei risultati si ottengono digitando `help format`. Dato il numero $1/7$, alcuni formati comunemente usati sono

`format short`	produce	0.1429
`format short e`	produce	1.4286e-01
`format short g`	produce	0.14286
`format long`	produce	0.142857142857143
`format long e`	produce	1.428571428571428e-01
`format long g`	produce	0.142857142857143

Gli stessi formati sono disponibili in Octave e forniscono risultati con lievi discrepanze.

$+$	addizione
$-$	sottrazione
$*$	prodotto
$/$	divisione
$\wedge$	elevamento a potenza

sin	`sin`
cos	`cos`
tan	`tan`
arcsin	`asin`
arccos	`acos`
arctan	`atan`
exp	`exp`
ln	`log`
$\log_2$	`log2`
$\log_{10}$	`log10`
$\|.\|$	`abs`
$\sqrt{\cdot}$	`sqrt`

Tabella 1.1: *Operazioni aritmetiche e principali funzioni matematiche*

Operatori e funzioni matematiche predefinite

Le operazioni aritmetiche e funzioni matematiche predefinite sono quelle elencate nella tabella 1.1.

Matrici e vettori

Ricordiamo che una matrice di numeri con m **righe** ed n **colonne** altro non è che una tabella di numeri della forma

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ & & \cdots & \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

dove $a_{ij} \in \mathbb{R}$ $(i = 1, \ldots, m;\ j = 1, \ldots, n)$, se i numeri sono reali.
In MATLAB/Octave le variabili hanno una struttura di tipo **matriciale**:

- Gli scalari sono considerati matrici 1×1.
- I vettori riga sono matrici $1 \times n$.
- I vettori colonna sono matrici $n \times 1$.

Per definire una **matrice** se ne possono innanzitutto assegnare direttamente gli elementi riga a riga. Ad esempio digitando

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

si produce

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

Notiamo che i punto e virgola separano righe diverse.
L'elemento in riga i e colonna j di A si accede con $A(i, j)$. Per la matrice A dell'esempio precedente

```
>> A(2,3)
ans =  6
```

Esercizio 2. *Costruire una matrice* 2×3 *con i primi sei numeri interi come coefficienti. Azzerare gli elementi* $A(1,1)$ *e* $A(2,2)$.

Soluzione:

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

>> A(1,1) = 0;
>> A(2,2) = 0;
>> A

A =
   0   2   3
   4   0   6
```

Vettori

MATLAB/Octave tratta i vettori come casi particolari di matrici.

Per memorizzare il vettore riga $x = [1, 2, 3, 4, 5]$ occorre digitare

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

mentre

```
y = [-2; 4 ; 12];
```

produce il vettore colonna

```
y =
  -2
   4
  12
```

`zeros(1,n)` crea un vettore riga di dimensione n con tutti gli elementi nulli; `zeros(n,1)` idem per vettori colonna.

Per creare un vettore riga (colonna) con elementi uguali ad 1 usiamo `ones(1,n)` (`ones(n,1)`).

La componente i-esima di un vettore si identifica con `x(i)`. Per esempio, la terza componente del vettore y dell'esempio precedente si accede con:

```
y(3)
ans =
     12
```

Un vettore colonna si trasforma nel corrispondente vettore riga mediante trasposizione:

```
x = [-2; 4; 12]'
x =
   -2    4    12
```

Si può creare un vettore vuoto (cioè con zero componenti) con il comando `x = []`. La norma euclidea di un vettore x si definisce come $\|x\| = \sqrt{x^T x}$. In MATLAB/Octave si implementa con il comando `norm(x)`.

Operazioni tra vettori

Essendo z, u, v vettori (riga o colonna) ed s uno scalare, si definiscono le operazioni:

`z = s*u`	prodotto di un vettore per uno scalare.
`z = u+v`	somma di due vettori di dimensione n.
`z = u-v`	sottrazione di due vettori n.
`z = u.*v`	$z_i = u_i \cdot v_i$ (prodotto tra due vettori componente a componente).
`z = u./v`	$z_i = u_i / v_i$ (divisione tra due vettori componente a componente).

Il prodotto scalare tra due vettori colonna x e y di dimensione n: $s = x^T y = \sum_{i=1}^{n} x_i y_i$ si esegue in MATLAB/Octave come `s = x'*y`

Esercizio 3. *Si definisca il vettore colonna u di componenti $(1, 2)$ e il vettore colonna v di componenti $(3, 4)$; si calcoli $u + v$, $u - v$, il prodotto scalare di u per v e il prodotto componente a componente.*

Soluzione:

```
>> u=[1;2];
>> v=[3;4];
>> u+v
ans =
   4
   6
>> u-v
ans =
  -2
  -2
>> u'*v
ans =  11
>> u.*v
ans =
   3
   8
```

Uso vettoriale delle funzioni elementari

Le funzioni matematiche predefinite in MATLAB/Octave sono vettoriali. Questo significa che possono essere applicate simultaneamente a tutte le componenti di un vettore. Esempi:

```
>> u = [1:1:6]
u =
   1   2   3   4   5   6
>> log(u)
ans =
   0.00000   0.69315   1.09861   1.38629   1.60944   1.79176
>> exp(u)
ans =
    2.7183     7.3891    20.0855    54.5982   148.4132   403.4288
```

Per le operazioni moltiplicative (prodotto, divisione, elevamento a potenza) bisogna anteporre il punto all'operatore corrispondente per lavorare vettorialmente.

```
>> u = -1:0.5:1
u =
   -1.0000   -0.5000    0.0000    0.5000    1.0000
>> u.^2
ans =
   1.00000   0.25000   0.00000   0.25000   1.00000
>> 1./u
ans =
   -1   -2  Inf    2    1
```

Prodotto matrice per vettore

Se A è una matrice $n \times m$ e u un vettore colonna $m \times 1$, allora `A * u` è l'usuale prodotto matrice per vettore.

Esercizio 4. *Sia* $A = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \\ -1 & 3 & 0 \end{pmatrix}$, $u = \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix}$. *Calcolare* $w = A \cdot u$.

Soluzione:

```
>> A = [1 3 5; 2 4 6; -1 3 0];
>> u = [1 1 2]
u =
   1   1   2

>> w = A*u'
w =
   14
   18
    2
```

Se avessimo scritto `A * u`, avremmo ottenuto un messaggio di errore perché il vettore u è stato creato come vettore riga.

Notazione ":"

In MATLAB/Octave si possono creare vettori riga con elementi equispaziati con la notazione ":" la cui sintassi è

```
vettore = [inizio:incremento:fine]
```

dove `inizio` è il primo elemento del vettore, e `incremento` è un parametro opzionale che indica la spaziatura tra gli elementi (se omesso `incremento`$= 1$).
La semantica di questo comando si può schematizzare così:

```
Crea gli elementi
vettore(i) = inizio + (i-1)*incremento
      fino a quando
      vettore(i) <= fine
```

Si osservi che `fine` non è necessariamente l'ultima componente del vettore. Esempi:

```
>> u = [1:2:10]
u =
   1   3   5   7   9
>> v = [5:-1:1]
v =
   5   4   3   2   1
```

```
>> w = [0:0.2:1]
w =
   0.00000   0.20000   0.40000   0.60000   0.80000   1.00000
```

Comando `linspace`

Un'alternativa ai due punti è il comando `linspace`(inizio,fine,numero di punti)

Il comando `linspace` genera un vettore riga con un numero prefissato di punti equispaziati compresi tra `inizio` (primo elemento) e `fine` (ultimo elemento del vettore). Se il `numero di punti` è omesso se ne creano 100. Esempi:

```
>> u = linspace(0,8,5)
u =
   0   2   4   6   8
>> v = linspace(-5,5,6)
v =
  -5  -3  -1   1   3   5
>> v = linspace(-5,5,5)
v =
  -5.00000  -2.50000   0.00000   2.50000   5.00000
```

Vettori con elementi equispaziati

Esercizio 5. *Creare il vettore u di componenti $u_i = -3 + 0.5 \cdot i,\ i = 0, \ldots, 6$, utilizzando sia la notazione* `:` *che il comando* `linspace`.

Soluzione:

```
>> u = [-3:0.5:0]
u =

  -3.00000  -2.50000  -2.00000  -1.50000  -1.00000  -0.50000   0.00000

>> u = linspace(-3,0,7)
u =

  -3.00000  -2.50000  -2.00000  -1.50000  -1.00000  -0.50000   0.00000
```

Un'istruzione MATLAB/Octave di uso molto comune è `length` che calcola il numero di elementi di un vettore.

```
>> length(u)
ans =  7
```

Comandi predefiniti che operano su matrici

Comandi che generano matrici:

`rand(m,n)`	matrice random di dimensione $m \times n$ con coefficienti distribuiti uniformemente in $[0, 1]$.
`randn(m,n)`	matrice random di dimensione $m \times n$ con distribuzione normale dei coefficienti in $[-\infty, \infty]$ di media 0 e varianza 1.
`eye(n)`	matrice identità di ordine n.
`ones(n)`	matrice di ordine n con coefficienti tutti uguali ad 1.

Altri comandi importanti che operano con matrici:

`inv(A)`	calcola l'inversa della matrice A;
`[m,n] = size(A)`	restituisce un vettore con il numero di righe (m) e quello di colonne (n) di A;
`det(A)`	calcola il determinante di A.

Esercizio 6. *Creare una matrice quadrata A di ordine 4 con tutti gli elementi uguali a 1 e calcolarne il determinante.*
Che cosa succede se proviamo a calcolare l'inversa di A?

La soluzione dell'esercizio è la seguente:

```
>> A = ones(4)
A =

   1   1   1   1
   1   1   1   1
   1   1   1   1
   1   1   1   1

>> det(A)
ans = 0
>> inv(A)
warning: inverse: matrix singular to machine precision, rcond = 0
ans =

   Inf   Inf   Inf   Inf
   Inf   Inf   Inf   Inf
   Inf   Inf   Inf   Inf
   Inf   Inf   Inf   Inf
```

Operazioni tra matrici

Essendo A, B, C matrici con coefficienti reali ed s uno scalare, si definiscono le operazioni elencate nella tabella 1.2.

`C = s*A`	prodotto di una matrice per uno scalare.
`C = A'`	trasposizione di una matrice.
`C = A+B`	somma di due matrici di dimensione $m \times n$.
`C = A-B`	sottrazione di due matrici $m \times n$.
`C = A*B`	prodotto di A (m righe e n colonne) per B (n righe e p colonne).
`C = A.*B`	$c_{ij} = a_{ij} \cdot b_{ij}$ (prodotto di due matrici componente a componente).
`C = A^p`	elevamento a potenza: $C = A^p$.
`C = A.^p`	elevamento a potenza componente per componente.

Tabella 1.2: *Operazioni tra matrici*

Riportiamo un esempio di somma, prodotto, e prodotto componente a componente di due matrici.

```
>> A = [4 -1 0; -1 4 -1; 0 -1 4];
>> B = [1 2 3; -4 -5 -6; 0 1 2];
```

Si noti che, come menzionato all'inizio della dispensa, il carattere `;` usato alla fine di qualunque istruzione sopprime l'output a video.

```
>> C = A+B
C =
    5    1    3
   -5   -1   -7
    0    0    6

>> C = A*B
C =
    8    13    18
  -17   -23   -29
    4     9    14

>> C = A.*B
C =
    4    -2     0
    4   -20     6
    0    -1     8
```

Trasposizione

```
>> A = [2  -1; 3  4; -2  7]
A =
    2   -1
    3    4
   -2    7

>> B = A'
B =
    2    3   -2
   -1    4    7
```

Elevamento a potenza

```
>> A=[1  1  1  ;  2  2  2  ;  3  3  3]
A =
    1    1    1
    2    2    2
    3    3    3

>> A*A
ans =
    6    6    6
   12   12   12
   18   18   18

>> A^2
ans =
    6    6    6
   12   12   12
   18   18   18
```

Elevamento a potenza componente per componente

```
>> A.^2
ans =

   1   1   1
   4   4   4
   9   9   9
```

Estrazione di sottomatrici

Consideriamo adesso la matrice seguente

```
>> A=[1 2 3 4; 2 3 5 7;3 3 4 8; 0 -1 2 3]

A =

    1    2    3    4
    2    3    5    7
    3    3    4    8
    0   -1    2    3
```

e vediamo mediante alcuni esempi com'è possibile accedere a sottomatrici di A. Innanzitutto, l'elemento in riga i e colonna j di A si accede con $A(i,j)$. Per la matrice A dell'esempio

```
>> A(2,3)
ans =  5
```

Si possono estrarre da A sottomatrici come segue:

- Una sola riga

  ```
  >> A(3,:)
  ans =
     3   3   4   8
  ```

- Un sottoinsieme degli elementi di una riga

  ```
  >> A (1,1:4)
  ans =
     1   2   3   4

  >> A (3,2:3)
  ans =
     3   4
  ```

- Una sottomatrice

  ```
  >> A(2:4,2:4)
  ans =
     3   5   7
  ```

```
   3    4    8
  -1    2    3

>> A(4,[2,1,3,4])
ans =
  -1    0    2    3
```

- Un elemento

```
>> A(1,end)
ans = 4
```

  Dato un vettore v l'istruzione `v(end)` ne visualizza l'ultima componente.

Estrazione da A di una sua diagonale

La diagonale principale di una matrice A si accede mediante il comando `diag(A)` che restituisce la diagonale in un vettore colonna. Per la matrice dell'esempio precedente si ha:

```
>> diag(A)
ans =

   1
   3
   4
   3
```

È anche possibile accedere ad una diagonale diversa dalla principale specificando come secondo parametro del comando `diag` la posizione che occupa contando a partire dalla diagonale principale (indice 0): per le diagonali sopra la diagonale principale occorre specificare un indice positivo, mentre per quelle sotto la diagonale principale un indice negativo. Seguono due esempi:
Vettore contenente la prima sottodiagonale di A.

```
>> diag(A,-1)
ans =

   2
   3
   2
```

Vettore contenente la seconda sopradiagonale di A.

```
>> diag(A,2)
ans =

   3
   7
```

Estrazione di una sottomatrice triangolare

Data una matrice A è possibile estrarre rispettivamente la sua parte triangolare inferiore e triangolare superiore con i comandi `tril, triu` come negli esempi che seguono:

```
>> tril(A)
ans =
   1   0   0   0
   2   3   0   0
   3   3   4   0
   0  -1   2   3

>> tril(A,-1)
ans =
   0   0   0   0
   2   0   0   0
   3   3   0   0
   0  -1   2   0

>> triu(A)
ans =
   1   2   3   4
   0   3   5   7
   0   0   4   8
   0   0   0   3
```

Si noti come le matrici restituite dai comandi `tril` e `triu` hanno sempre le stesse dimensioni di A.

Costruzione di una matrice diagonale

Un altro uso del comando `diag` è quello di costruire matrici diagonali. Se a `diag` viene passato come parametro un vettore, anzichè una matrice, allora restituisce una matrice diagonale con gli elementi del vettore come elementi diagonali.

```
>> v=[ -1 -2 5 9]
v =

  -1  -2   5   9

>> D=diag(v)
D =

Diagonal Matrix

  -1   0   0   0
   0  -2   0   0
   0   0   5   0
   0   0   0   9
```

Anche in questo caso si può scegliere su quale diagonale posizionare gli elementi del vettore che viene passato come parametro al comando `diag`:

```
>> w=[-1 -2 -3]
w =

  -1   -2   -3

>> D =diag(w,-1)
D =

    0    0    0    0
   -1    0    0    0
    0   -2    0    0
    0    0   -3    0
```

si noti però che la matrice risultante non è più una matrice diagonale.

Se si vuole creare una matrice diagonale che abbia gli stessi elementi diagonali di una matrice data occorre scrivere:

```
>> A =
    1    2    3    4
    2    3    5    7
    3    3    4    8
    0   -1    2    3

>> diag(diag(A))
ans =

Diagonal Matrix

    1    0    0    0
    0    3    0    0
    0    0    4    0
    0    0    0    3
```

Inizializzazione di matrici

Una matrice quadrata con elementi tutti uguali a 1 si ottiene mediante il comando `ones(n)` dove n è la dimensione della matrice. Se si vuole creare una matrice rettangolare $m \times n$ di tutti 1, si deve scrivere `ones(m,n)`.

```
>> ones(2,4)
ans =

    1    1    1    1
    1    1    1    1
```

Analogamente, una matrice quadrata con elementi tutti uguali a 0 è data da `zeros(n)` dove n è la dimensione della matrice. Una matrice $m \times n$ con tutti gli elementi uguali a zero è data da `zeros(m,n)`.

```
>> zeros(3,2)
ans =

   0   0
   0   0
   0   0
```

La matrice identità viene generata tramite l'istruzione `eye(n)` o `eye(m,n)` a seconda che si voglia una matrice quadrata $n \times n$ o una rettangolare $m \times n$.

Se si vuole generare una matrice con elementi casuali (random), si usa l'istruzione `rand(m,n)` (oppure `randn(m,n)`).

```
>> R=rand(3,3)
R =
    0.9501    0.4860    0.4565
    0.2311    0.8913    0.0185
    0.6068    0.7621    0.8214

>> b=rand(3,1)
b =
    0.4447
    0.6154
    0.7919
```

Altre importanti funzioni che operano su matrici sono riportate in Tabella 1.3.

Soluzione di un sistema lineare

Un sistema lineare $Ax = b$ si risolve in MATLAB/Octave con il comando `A\b`. Se A è una matrice quadrata invertibile generale, l'operatore `\` restituisce la soluzione $x = A^{-1}b$ calcolata con il metodo di eliminazione di Gauss con pivoting parziale.

```
>> A=[10 -7 0; -3 2 6; 5 -1 5]
A =
    10    -7     0
    -3     2     6
     5    -1     5
>> b=[7; 4; 6]
b =
     7
     4
     6
>> x=A\b
x =
```

`norm(A,1)`	norma 1 di A
`norm(A,2)`	norma 2 di A
`norm(A,inf)`	norma infinito di A
`norm(v)`	norma euclidea del vettore v
`norm(v,1)`	norma 1 del vettore v
`norm(v,inf)`	norma infinito del vettore v
`[n,m] = size(A)`	restituisce il numero di righe e di colonne di A;
`det(A)`	determinante della matrice quadrata A
`cond(A)`	numero di condizionamento di A (in norma 2)
`inv(A)`	inversa della matrice quadrata A
`trace(A)`	traccia di A (somma degli elementi della diagonale principale)
`[L,U,P]=lu(A)`	effettua la fattorizzazione LU di A, con P matrice di permutazione tale che $PA = LU$
`R=chol(A)`	calcola la fattorizzazione di Cholesky di A simmetrica definita positiva; restituisce il fattore triangolare superiore.
`[V,D]= eig(A)`	restituisce una matrice diagonale D di autovalori di A e una matrice V le cui colonne sono i corrispondenti autovettori tale che $AV = VD$
`[Q,R] = qr(A)`	fattorizza la matrice A nel prodotto di una matrice ortogonale Q e una matrice R con elementi nulli sotto la diagonale principale

Tabella 1.3: *Funzioni che operano su matrici*

```
     0
    -1
     1
```

Nel caso di termine noto con più colonne, la scrittura $A\backslash B$, con $B = [b_1 b_2 \dots b_m]$ risolve $AX = B$, ovvero gli m sistemi lineari $Ax_i = b_i$, con $X = [x_1 x_2 \dots x_m]$.

```
>> A = [1  2  3  4; 2  3  5  7; 3   3  4  8; 0  -1  2  3]
A =

    1    2    3    4
    2    3    5    7
    3    3    4    8
    0   -1    2    3

>> B = [-2  3; 2  1; 4  5; 2  -1]
B =

   -2    3
    2    1
    4    5
    2   -1
```

```
>> X = A\B
X =

   15.6000  -13.8000
   -6.8000    6.4000
    7.2000   -9.6000
   -6.4000    8.2000
```

Nel caso di sistemi sovradeterminati (più equazioni che incognite), il comando \ calcola automaticamente la soluzione ai minimi quadrati (soluzione del sistema delle equazioni normali).

```
>> C = A(1:4,1:3)
C =

   1   2   3
   2   3   5
   3   3   4
   0  -1   2

>> d = [-1; 2; 5; 4]
d =

  -1
   2
   5
   4

>> C\d
ans =

   4.06931
  -3.04950
   0.51485
```

Inversa di una matrice

Essendo A la matrice dell'esempio precedente $A = \begin{pmatrix} 10 & -7 & 0 \\ -3 & 2 & 6 \\ 5 & -1 & 5 \end{pmatrix}$, il comando `inv(A)` restituisce l'inversa di A.

```
>> B=inv(A)
B =

  -0.1032258  -0.2258065   0.2709677
  -0.2903226  -0.3225806   0.3870968
   0.0451613   0.1612903   0.0064516
```

MATLAB come linguaggio di programmazione

MATLAB/Octave può essere considerato un linguaggio di programmazione alla stregua di Fortran, di C, ecc. Non viene compilato ma interpretato, il che lo rende poco efficiente per calcoli intensivi. Un programma MATLAB/Octave deve essere salvato in un m-file (file avente estensione .m). I programmi MATLAB/Octave possono essere di due tipi:

- `script`. Si veda la sezione 1.44 per dettagli.
- `function`. Si veda la sezione 1.45 per dettagli.

Le strutture di programmazione basilari in MATLAB/Octave sono:

- Istruzione condizionale (`if else end`)
- Cicli (`for` e `while`)

Gli operatori logici e di relazione in MATLAB/Octave sono quelli riportati nella Tabella 1.4.

Operatori logici		Operatori di relazione	
& &	short circuit AND	==	uguale
||	short circuit OR	∼=	diverso
&	AND (element wise)	<	minore
|	OR (element wise)	>	maggiore
∼	NOT	<=	minore o uguale
		>=	maggiore o uguale

Tabella 1.4: *Operatori logici e di relazione.*

Il valore restituito dagli operatori può essere vero o falso e MATLAB utilizza il numero 1 per indicare il valore vero e 0 per il valore falso.
Se, ad esempio, poniamo `x=5;` e `y=1` e scriviamo la proposizione `x < y`, MATLAB risponde con `ans = 0` indicando che il confronto esprime una condizione falsa.
Come mostrato nella tabella ci sono due tipi di operatori logici: quelli detti short circuit e quelli che operano anche su matrici e vettori componente a componente (element wise). Gli operatori short circuit AND e OR si applicano invece a predicati scalari, e sono molto più efficienti perché permettono di risparmiare tempo nella valutazione del predicato composto. Infatti, nel caso di due predicati congiunti mediante un short circuit AND, se il primo predicato è falso, il secondo predicato non viene valutato. Analogamente, nel caso di due predicati disgiunti da un short circuit OR, se il primo di loro è vero, il secondo non verrà valutato.

Il costrutto `if-else-end`

La struttura alternativa permette di eseguire un diverso insieme di istruzioni a seconda che sia vera o falsa una certa condizione logica posta dopo la parola chiave `if`.

La sintassi del costrutto `if` è la seguente:

```
if espressione logica
        istruzioni
end
```

Esempio

```
if a > b
   maxval = a
end
```

```
if espressione logica
        istruzioni
else
        istruzioni
end
```

Esempio

```
if x > 0
   a = sqrt(x)
else
   a = 0
end
```

```
if espressione logica 1
        istruzioni
elseif espressione logica 2
        istruzioni
...
else
        istruzioni
end
```

Il costrutto `for`

La sintassi del costrutto `for` è la seguente:

```
for k = vettore
        istruzioni
end
```

I comandi che si trovano tra `for` e `end` sono eseguiti per tutti i valori di `k` che sono nel `vettore`.
Se volessimo creare la matrice di Hilbert di ordine 4, basterebbe scrivere:

```
for i=1:4
    for j=1:4
        A(i,j) = 1/(i+j-1);
    end;
end
```

che produce in output

```
>> A
A =
```

```
   1.00000   0.50000   0.33333   0.25000
   0.50000   0.33333   0.25000   0.20000
   0.33333   0.25000   0.20000   0.16667
   0.25000   0.20000   0.16667   0.14286
```

che si verifica essere equivalente al comando `hilb(4)`.
L'esempio mostra come si possano usare due o più cicli `for` annidati.

Il costrutto `while`

Per il ciclo `while` la sintassi è data da:

```
while espressione logica
      istruzioni
end
```

Questo ciclo è usato quando le istruzioni devono essere ripetute fintantoché rimane vera l'espressione logica.
Ad esempio, per trovare la soluzione dell'equazione $x = \cos x$, mediante una iterazione di punto fisso, scriviamo le istruzioni:

```
x = 1; dif = 100;
while dif > 1e-8
      xnew = cos(x);
      dif = abs(xnew-x);
      x = xnew;
end
xnew
```

che producono il risultato

```
xnew =  0.739085136646572
```

Programmi in MATLAB/Octave: script

Uno script è un file di testo che contiene una semplice raccolta di istruzioni o comandi MATLAB/Octave senza interfaccia di input/output.
Ad esempio, l'insieme di istruzioni

```
a=1; b=-3; c = 2;
delta = b^2-4*a*c;
if delta < 0
   disp('radici complesse')
else
   x1 = (-b-sqrt(delta))/(2*a)
   x2 = (-b+sqrt(delta))/(2*a)
end
```

una volta salvato in un m-file, di nome `eq2grado.m` diventa uno script. Per eseguirlo, è sufficiente scrivere dopo il prompt il nome senza estensione:

```
>> eq2grado
x1 =  1
x2 =  2
```

si ottengono così le due soluzioni dell'equazione di secondo grado $x^2 - 3x + 2 = 0$.

Programmi in MATLAB/Octave: function

Come lo script, anche una function si definisce in un m-file. Il nome del file che contiene la function deve coincidere con il nome della function stessa. La definizione di ogni function MATLAB inizia con la parola chiave function, e ha la sintassi seguente:

```
function[out1, ..., outn] = nomefun(in1, ..., inm)
```

dove

- out1, $\cdots$, outn sono i parametri di output (opzionali);
- in1, $\cdots$, inm sono i parametri di input.
- Le variabili all'interno della function sono locali, il loro valore viene perduto al termine dell'esecuzione.
- Una funzione può esser invocata o da command window o da uno script.
- La function termina o all'ultima sua istruzione oppure quando si incontra per la prima volta il comando return.

Esempio

file radici.m

```
function[x1,x2,err] = radici(a,b,c)
err = 0;
delta = b^2-4*a*c;
if delta < 0
   err = 1; x1=0; x2=0;
   return
else
   x1 = (-b-sqrt(delta))/(2*a);
   x2 = (-b+sqrt(delta))/(2*a);
end
```

file scriptradici.m

```
a=1; b=-3; c = 2;
[x1,x2,err] = radici(a,b,c)
```

```
>> scriptradici
x1 =  1
x2 =  2
err = 0
```

Poiché le variabili all'interno delle function sono locali, una volta eseguito lo script, da Command Window non è possibile conoscere il valore di delta. Questo spiega il seguente messaggio di errore:

```
>> delta
error: 'delta' undefined near line 99 column 1
```

Esercizi

Si elencano alcuni esercizi per mettere in pratica quanto appreso nelle pagine precedenti sulla costruzione di semplici programmi MATLAB.

Esercizio 12. *Costruire le function elencate di seguito il cui costrutto fondamentale è un ciclo* `for`*:*

1. `s = somma(n)`, *in cui assegnato* n*, calcoli la somma* s *dei primi* n *numeri naturali.*
2. `n_fatt = fatt(n)`, *in cui assegnato* n*, calcoli il suo fattoriale* $n!$ *(si ricordi che* $n! = 1 \cdot 2 \cdot 3 \cdot 4 \cdots (n-1) \cdot n$*, e che* $0! = 1! = 1$*).*
3. `x = fibonacci(n)`, *in cui assegnato* $n > 0$*, restituisca in* `x` *i primi* n *termini della successione di fibonacci definita dalla relazione* $x_k = x_{k-1} + x_{k-2}$*,* $k \geq 3$ *con partenza da* $x_1 = x_2 = 1$*.*

Esercizio 13. *Costruire la function* `n = min_exp(a)`, *che calcoli il più piccolo intero* n *tale che* $2^n \geq a$*.*

Esercizio 14. *Costruire le function elencate di seguito:*

1. `s = somme_parz(a)`, *in cui assegnato un vettore* a*, restituisca in output il vettore* s *delle somme parziali,*
$$s_k = \sum_{j=1}^{k} a_j$$
2. `n = conta_gta(A,a)`, *in cui assegnata una matrice* A *e uno scalare* a*, produca in output il numero di elementi di* A *che sono maggiori di* a*.*
3. `w = rev_ord(v)`, *in cui assegnato un vettore* v *inverta l'ordine dei suoi elementi (l'ultimo deve diventare il primo, il penultimo deve diventare il secondo,...).*
4. `K = krylov(A,v,n)`, *in cui assegnata una matrice* $A \in R^{N \times N}$*, un vettore* $v \in R^N$ *e un intero* $n > 0$*, produca in output una matrice* K *le cui colonne sono i vettori di Krylov* $v, Av, A^2v, ..., A^{n-1}v$*.*

Input/Output in MATLAB/Octave

La funzione `input` sospende l'esecuzione del programma e richiede all'utente di inserire dei dati in ingresso.
Esempio:

```
>> z = input('Inserisci un valore')
```

mostra la stringa

```
Inserisci un valore
```

nella finestra di comando. Se l'utente inserisce ad esempio il valore 5, tale valore viene assegnato alla variabile z. Funziona anche per inserire matrici e vettori, ma in questo caso l'utente deve inserire le parentesi opportune così come le virgole o punti e virgola necessari. Per esempio:

```
A = input('Inserisci i valori della matrice A')
```

richiede all'utente di inserire una matrice. Se introduciamo ad esempio:

```
[1 2 3 ; 4 5 6]
```

viene visualizzato

```
A =
   1 2 3
   4 5 6
```

Il comando `disp` serve per visualizzare una stringa di caratteri (testo racchiuso tra apici), o una variabile senza che ne venga visualizzato il nome.

```
>> x=1:2:19;
>> disp(x)
    1    3    5    7    9   11   13   15   17   19

>> disp('Questa e'' una stringa');
Questa e' una stringa
```

Si possono visualizzare più dati in un unico comando `disp`:

- Stringhe e variabili numeriche insieme

  ```
  >> disp(['Convergenza in ',num2str(iter),' iterazioni']);
  Convergenza in 23 iterazioni
  ```

  Il comando `num2str` converte un numero in una stringa.

- Più variabili numeriche

  ```
  >> disp([val,err,iter])
     2.1099e+00    1.0000e-10    2.3000e+01
  ```

L'output del comando `disp` finisce sempre con un avanzamento di linea.

Output con formato: comando `fprintf`

Per visualizzare un insieme di dati di output con un certo formato si usa il comando `fprintf` con degli opportuni descrittori di formato:

Descrittore	Significato
%f	formato decimale (virgola fissa)
%e	notazione esponenziale
%i o %d	notazione per interi con segno
%g	la notazione piú compatta tra %f ed %e
%s	stringa di caratteri
\n	avanzamento di linea
\t	tabulazione
\b	backspace

Tra % e il tipo di formattazione è possibile precisare il numero minimo di caratteri da stampare e il numero di cifre decimali dopo il punto. Alcuni esempi sono raccolti nella seguente tabella:

Valore	%6.3f	%6.0f	%6.3e	%6.3g	%6.3d	%d
2	2.000	2	2.000e+000	2	002	2
0.02	0.020	2	2.000e−002	0.02	000	0
200	200.000	200	2.000e+002	200	200	200
`sqrt(2)`	1.414	1	1.414e+000	1.41	001	1

Ad esempio, il formato %6.3f significa che si rappresenta il numero in virgola fissa (f) mettendogli a disposizione almeno 6 spazi e visualizzando 3 cifre dopo il punto decimale.

Come ulteriore esempio consideriamo le istruzioni

```
>> tensione = 3.5;
>> fprintf('La tensione e' \%8.2f millivolts \n', tensione),
```

che producono:

```
>> La tensione e'     3.50 millivolts
```

Si noti lo spazio vuoto prima del numero 3.50 che si deve all'aver destinato complessivamente 8 spazi per il numero.
Con lo stesso comando `fprintf` si può realizzare la scrittura di dati su file. Per stampare su file una lista di variabili, secondo un certo formato specificato, la sintassi del comando `fprintf` è:

fprintf(fid, formato, var1,...,varn)

Per esempio, supponiamo di aver in memoria la variabile x il cui valore è 0.001. Apriamo il file `prova.txt` in modo scrittura (si noti la stringa 'w'):

```
>> fid=fopen('prova.txt','w');
```

e ci scriviamo dentro il valore di x in formato esponenziale:

```
>> fprintf(fid,'\n Ho scritto il numero x= %5E  \n',x);

>> fclose(fid);
```

una volta chiuso il file, mediante l'istruzione `fclose` nella cartella di lavoro è presente il file `prova.txt` che ha come contenuto:

```
 Ho scritto il numero x= 1.000000E-03
```

La lista di variabili da stampare su file può essere formata da array (vettori e/o matrici).

```
>> v=[1 2 3];

>>fprintf('\n Componenti del vettore v: \t  %2d  %2d %2d \n',v)

 Componenti del vettore v:     1   2  3
>>
```

Nel caso in cui si voglia stampare una matrice occorre prestare attenzione al fatto che il comando `fprintf` stampa i dati di ogni variabile matriciale per colonne:

```
>>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

>>fprintf('\n   %2d  %2d %2d \n ',A)

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

Per stampare correttamente la matrice A occorre passare come parametro al comando `fprintf` la sua trasposta:

```
>>fprintf('\n   %2d  %2d %2d %2d\n ',A')

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

Comandi save e load

Il comando `save` permette di salvare variabili memorizzate nel workspace su un file binario chiamato `matlab.mat`. Se si vogliono salvare solo alcune variabili e si vuole specificare un nome diverso per il file si usa `save nomefile lista-variabili`. Il comando `load` ripristina nella sessione corrente tutte le variabili memorizzate nel file binario `matlab.mat`. Analogamente, `load nomefile lista-variabili` permette di caricare da file solo le variabili elencate nella lista. Il comando `load` può essere usato

anche per caricare dati da file, ad esempio vettori o matrici, scrivendo `load nomefile variabili`.
Ad esempio, supponiamo di aver registrato il file `precip.dat`

```
1       5.35
2       3.68
3       3.54
4       2.39
5       2.06
6       1.48
7       0.63
8       1.09
9       1.75
10      2.66
11      5.34
12      6.13
```

Il file contiene le ascisse e le ordinate di alcune osservazioni (dal titolo si capisce che sono precipitazioni in alcuni giorni dell'anno), ed è quindi scritto come una matrice con 12 righe e 2 colonne. Il comando `load` carica questa variabile nel workspace di MATLAB/Octave . Vediamo come:

```
>> load precip.dat;
>> mese=precip(:,1)

mese =

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

>> prec=precip(:,2)

prec =

    5.3500
    3.6800
    3.5400
    2.3900
    2.0600
    1.4800
```

```
        0.6300
        1.0900
        1.7500
        2.6600
        5.3400
        6.1300

>> plot(mese,prec,'o','Markersize',6,'MarkerFacecolor','r')
```

Nelle istruzioni precedenti, con il comando `load` il contenuto del file precip.dat viene copiato nella variabile (matrice) `precip`, poi si immagazzinano le colonne della matrice rispettivamente nelle variabili `mese` e `prec` per poi creare il grafico mostrato in Figura 1.5.

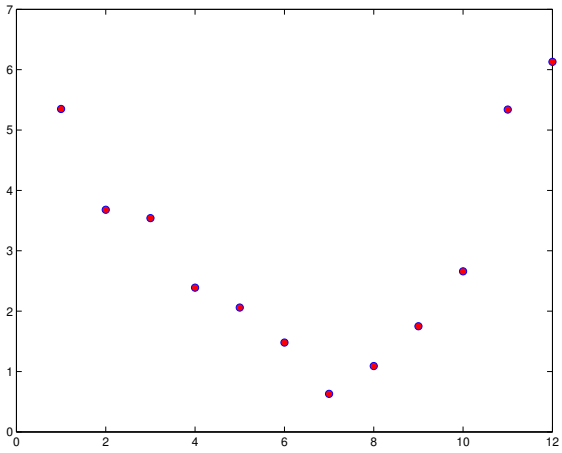


Figura 1.5: *Grafico di alcuni dati immagazzinati su file.*

Come misurare la durata di un programma

Per confrontare due programmi che risolvono lo stesso problema è utile misurare il tempo di CPU (wallclock time) impiegato per eseguirli. In MATLAB/Octave questo tempo si può misurare in secondi con il comando: `cputime`.

Consideriamo, ad esempio, le istruzioni seguenti per misurare il tempo di calcolo del determinante di una matrice quadrata di dimensione 5000:

```
>> A = rand(5000);
>> t = cputime;
>> det(A);
>> tfin=cputime;
>> cpu=tfin-t
cpu =   32.500
```

Esercizio 15. *Si crei una matrice quadrata random A di dimensione variabile da $n = 200$ a $n = 8000$ (con passo 200), e un vettore v lungo n. Misurare il tempo di esecuzione del prodotto $A * v$.*

Per risolverlo scriviamo lo script `matvet.m`

```
n=8000; step=200;
A=rand(n,n);
v=rand(n,1);
T=[ ];
sizea=[ ];
for k=200:step:n
        AA=A(1:k,1:k);
        vv=v(1:k);
        t=cputime;
        bb=AA*vv;
        tt=cputime-t;
        T=[T;tt];
        sizea=[sizea;k];
end
plot(sizea,T)
```

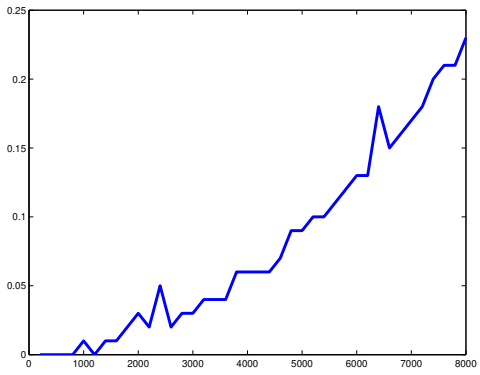


Figura 1.6: *Script* `matvet.m` *(figura a sinistra) e grafico dei tempi di esecuzione del prodotto matrice-vettore in funzione della dimensione del problema (figura a destra).*

Usando il comando

```
>> plot(sizea,T)
```

otteniamo il grafico, riportato in Figura 1.6, del tempo di esecuzione dell'algoritmo in funzione della dimensione della matrice. I tempi sono espressi in secondi.

Per misurare il tempo totale trascorso dall'inizio dell'esecuzione di un programma sono disponibili i comandi `tic` e `toc`. Il comando `tic` fa partire un cronometro interno che viene fermato all'esecuzione del comando `toc`. Il tempo misurato è espresso in secondi, L'uso di questi comandi è illustrato nell'esempio seguente dove si studia come cambia il tempo necessario per generare due matrici con elementi aleatori e moltiplicare puntualmente le loro trasposte:

```
tic
A = rand(12000, 4400);
B = rand(12000, 4400);
toc
C = A'.*B';
toc
```

Si ottiene:

```
Elapsed time is 1.304582 seconds.
Elapsed time is 1.888379 seconds.
```

dove il primo tempo misura la generazione delle due matrici e il secondo il tempo totale intercorso tra `tic` e l'ultimo `toc`.

Il comando `diary`

Un comando interessante di Matlab è il comando `diary` che scrive su file quanto visualizzato nella shell di MATLAB/Octave durante la sessione di lavoro, dal momento in cui l'utente scrive `diary on`. Quando non si desidera più salvare su file i comandi della sessione di lavoro basta scrivere `diary off`. Queste due istruzioni hanno come conseguenza la creazione del file di testo `diary` nella directory attuale di lavoro. Tale file contiene quanto apparso sulla shell di MATLAB/Octave ad eccezione del prompt >>. Si noti che può essere utile per vedere a casa quanto fatto a lezione sul workspace di MATLAB/Octave .

Help in linea

Concludiamo questa breve guida all'uso dell'ambiente MATLAB ricordandone una caratteristica molto importante, e cioè, avere un meccanismo di *help* in linea, disponibile per tutti i comandi ed i programmi che siano visibili ed eseguibili dal workspace (quindi anche quelli costruiti dall'utente). A tal proposito basta scrivere dopo il prompt

```
help nomeprogramma
```

per produrre la stampa a video delle prime righe commentate del file *nomeprogramma.m*.
Per esempio digitando `help sin` e premendo il tasto ENTER otteniamo da MATLAB

```
>> help sin

  SIN     Sine of argument in radians.
    SIN(X) is the sine of the elements of X.

    See also asin, sind.
    Overloaded methods:
       distributed/sin
       sym/sin
```

In alcune distribuzioni per uscire dall'help, senza chiudere MATLAB/Octave, occorre digitare `q`. Nelle distribuzioni più recenti oltre al comando `help` è disponibile anche il comando `doc` che mostra la stessa informazione in forma di guida html.