MATLAB *

A. SOMMARIVA [†] E M.VENTURIN [‡]

Mostriamo ora come utilizzare Matlab/Octave sotto Linux, osservando che comunque sussistono molte analogie con quanto si debba fare sotto Windows o MacOS. Al fine di implementare gli algoritmi e verificarne il funzionamento, necessita evidentemente un ambiente di programmazione. Tra quelli che hanno avuto maggior successo in Analisi Numerica citiamo il Fortran, il C++, Python, Matlab (o le sue alternative freeware GNU Octave o Scilab).

In generale, accanto a uno qualsiasi di questi linguaggi troviamo librerie di calcolo scientifico, che in qualche senso da definire sono *codici* che un utente può utilizzare per risolvere problemi dell'Analisi Numerica più velocemente ed efficacemente. I programmi di queste librerie sono stati implementati con cura da specialisti, cosicchè l'utente finale deve solo richiamarli, qualora lo ritenga opportuno.

In questo corso utilizzeremo MATLAB (sigla per Matrix Laboratory). Tale programma di tipo commerciale ha avuto origine nel 1983 ed ha avuto successo per la semplicità dei suoi comandi. All'url

http://www.mathworks.com

si possono trovare maggiori precisazioni sul suo utilizzo.

Come anticipato, esistono dei linguaggi la cui struttura è molto compatibile a MATLAB: GNU Octave o Scilab. Tra questi ultimi, GNU Octave presenta così tante similitudini da essere utilizzato con codici MATLAB con alte probabilitá di non presentare alcun errore di sintassi. Per dettagli tecnici si consulti

http://octave.sourceforge.net/

GNU Octave può trovarsi gratuitamente in molte distribuzioni di Linux, come ad esempio Quantian. Per dettagli si veda

http://dirk.eddelbuettel.com/quantian.html/

L'installazione su PC con Windows o MacOs risulta più complicata. Si considerino le distribuzioni per PC

1. http://www.math.mcgill.ca/loisel/octave-workshop/
2. http://octave.sourceforge.net/

e per MAC

http://wiki.octave.org/wiki.pl?OctaveForMac

1.1. Il Workspace di Matlab. I programmi scritti in linguaggio Matlab (o GNU Octave), vanno necessariamente salvati in un file di testo avente estensione *.m* (detto comunemente *m-file*).

Per familiarizzare con tali ambienti, si apra una finestra di tipo *terminale* e si digiti da tale *shell* il comando `matlab` (o in alternativa `octave`).

---

*Ultima revisione: 10 novembre 2011

[†] DIPARTIMENTO DI MATEMATICA PURA ED APPLICATA, UNIVERSITÀ DEGLI STUDI DI PADOVA, VIA TRIESTE 63, 35121 PADOVA, ITALIA (ALVISE@MATH.UNIPD.IT)

[‡] DIPARTIMENTO DI INFORMATICA, UNIVERSITÀ DEGLI STUDI DI VERONA, STRADA LE GRAZIE 15, 37134 VERONA, ITALIA (MANOLO.VENTURIN@GMAIL.COM)

L'ambiente di calcolo, che si presenta con una linea di comando del tipo >> permette di eseguire programmi che appartengono all'ambiente e programmi costruiti dall'utente, semplicemente digitando il nome dell'm-file che li contiene, senza estensione, sulla riga di comando e premendo il tasto `ENTER`. Così se il programma è stato salvato nel file `mioprogramma.m` e vogliamo eseguirlo con Matlab/Octave, digiteremo dall'*ambiente di lavoro* solamente `mioprogramma`.

Per essere eseguiti i programmi devono essere *visibili* dall'ambiente di lavoro di Matlab/Octave (il cosidetto *workspace*). Più correttamente devono essere nel path di lavoro di Matlab/Octave.

Per capire questo punto si digiti sulla *finestra* di Matlab/Octave il comando `ls`. Immediatamente vengono listati i files nella cartella corrente. Per cambiare cartella basta digitare sulla shell di Matlab/Octave un comando del tipo `cd altracartella` dove `altracartella` è una directory in cui vogliamo muoverci. Il comando `cd ..` muove Matlab/Octave in una cartella contentente la directory attuale. Matlab/Octave vede tutti i file che può listare con `ls`.

L'ambiente mantiene disponibili in memoria tutte le variabili che sono state inizializzate dal momento dell'ingresso nell'ambiente in poi, sia tramite singoli comandi che tramite istruzioni contenute nei programmi eseguiti, entrambi impartiti dalla linea di comando dell'ambiente. Questa è una sostanziale differenza rispetto alla programmazione in C o in FORTRAN, dove le variabili vengono allocate all'inizio dell'esecuzione del programma e poi de-allocate (cioè cancellate) al termine dell'esecuzione stessa. Una conseguenza immediata di questo fatto è che, ad esempio, in un ambiente come il Matlab non è necessario salvare esplicitamente su file i valori delle variabili che si vogliono osservare (es. graficare) dopo l'esecuzione del programma.

In termini grossolani, ciò significa che se il programma che abbiamo appena fatto girare conteneva la variabile `n`, una volta terminata l'esecuzione il valore di `n` è memorizzato e richiamabile dalla shell di Matlab/Octave. Per capire questo si digiti

```
a=atan(0.2);
```

e si prema `ENTER`. Quindi `a` e si prema nuovamente `ENTER`. Se tutto è stato eseguito correttamente viene scritto su monitor

```
>> a=atan(0.2);
>> a

a =

    0.1974

>>
```

segno che la variabile `a` dopo essere stata assegnata è mantenuta in memoria. Osserviamo inoltre che il ";" in $a = \arctan(0.2)$; ha l'effetto di non stampare il valore della variabile a.

Un'altra caratteristica molto importante di questo ambiente è 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 da shell

```
help nomeprogramma
```

per produrre la stampa a video delle prime righe commentate del file *programma.m*.

Per esempio digitando `help sin` e premendo `ENTER` otteniamo da Matlab 6

```
>>help sin

SIN    Sine.
    SIN(X) is the sine of the elements of X.

 Overloaded methods
    help sym/sin.m
```

In qualche versione Matlab/Octave in Linux può accadere che non sia ovvio come uscire dall'help. Viene infatti visualizzato il messaggio relativo all'help terminante con `END`. Per uscire dall'help, senza chiudere Octave/Matlab, si digiti `q`.

1.2. L'esecuzione dei programmi. Matlab e Octave effettuano la dichiarazione automatica delle strutture dati: quando viene usata una variabile per la prima volta, essi creano automaticamente lo spazio in memoria per contenerla. Questo meccanismo è molto comodo per il programmatore e non è comune nei linguaggi di programmazione. Ciò significa che non bisogna dire ad esempio che la variabile `n` è un numero reale, poichè Matlab/Octave non lo richiedono. In linguaggio tecnico questa si chiama *tipizzazione dinamica*.

A verificare la correttezza sintattica dei programmi ci pensa l'interprete, ma per verificare che lo svolgimento dei calcoli sia quello desiderato, è necessario confrontare un'esecuzione su un problema di piccole dimensioni con i calcoli a mano.

Per chi volesse saperne di più , si consultino ad esempio

1. http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/learnmatlab.pdf
2. http://www.mathworks.com/moler/

2. Matlab: operazioni con matrici e vettori. In questa sezione, mostreremo alcuni comandi di MATLAB che risulteranno utili per implementare gli algoritmi descritti in seguito.

2. Matlab: operazioni con matrici e vettori.

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

Mostriamo un primo esempio sull'utilizzo di tali operazioni in Matlab.

```
>>format short
>>(2+3*pi)/2
  ans =
  5.7124
>>format long
>>(2+3*pi)/2
  ans =
  5.71238898038469
>>format long e
>>(2+3*pi)/2
  ans =
  5.712388980384690e+000
```

Si vede che il valore di $(2+3\cdot\pi)/2$ viene approssimato fornendo poche o molte cifre decimali, anche in notazione esponenziale (qui e+000 sta per 10^0).
Altre funzioni elementari comunemente usate sono

`abs`	valore assoluto
`sin`	seno
`cos`	coseno
`tan`	tangente
`cot`	cotangente
`asin`	arco seno
`acos`	arco coseno
`atan`	arco tangente
`sinh`	seno iperbolico
`cosh`	coseno iperbolico
`tanh`	tangente iperbolica
`asinh`	arco seno iperbolico
`acosh`	arco coseno iperbolico
`atanh`	arco tangente iperbolica
`sqrt`	radice quadrata
`exp`	esponenziale
`log 2`	logaritmo base 2
`log10`	logaritmo base 10
`log`	logaritmo naturale
`fix`	arrotondamento verso 0
`round`	arrotondamento verso l'intero più vicino
`floor`	arrotondamento verso $-\infty$
`ceil`	arrotondamento verso $+\infty$
`sign`	segno
`rem`	resto della divisione

Si consideri l'esempio

```
>>a=3-floor(2.3)
  ans =
  1
>>a=3-ceil(2.3)
  ans =
  0
>>b=sin(2*pi);
```

Si noti che il "`;`" ha come effetto che non mette in display il valore della variabile b.

Se digitiamo il comando senza "`;`" il valore della variabile b verrà visualizzato come segue

```
>>a=sin(2*pi)
  a =
-2.4493e-016
```

Si osservi che il risultato non è 0 come ci si aspetterebbe, ma comunque un valore molto piccolo in modulo.

In Matlab l'utente può definire una funzione scrivendo un M-file, cioè un file con l'estensione *.m*. Per scrivere una funzione si digiti nella shell di Linux `pico fun.m`. Per salvare il file premere contemporaneamente il tasto `CTRL` e `X`. Se si accettano le modifiche digitare `y`. Mostriamo di seguito un esempio di funzione.

```
function y=fun(x)
y=5+sin(x);
```

Di conseguenza

```
y=fun(pi);
```

assegna alla variabile di input x il valore π e alla variabile di output y il valore $5 + \sin(\pi)$.

Ovviamente Matlab segnala errore se alla variabile di output y non è assegnato alcun valore. Per convincersene si scriva la funzione

```
function y=fun(x)
z=5+sin(x);
```

e da shell si esegua il comando

```
y=fun(pi);
```

come risultato Matlab avvisa su shell

```
Warning: One or more output arguments not assigned during call to 'fun'.
```

Alcune osservazioni:

- Ricordiamo che è fondamentale salvare il file in una directory appropriata e che se la funzione è chiamata da un programma al di fuori di questa directory una stringa di errore verrà visualizzata nella shell

  ```
  ???  Undefined function or variable 'fun'.
  ```

  Le funzioni predefinite da Matlab sono visibili da qualsiasi directory di lavoro. Quindi se il file *fattoriale.m* creato dall'utente è nella cartella *PROGRAMMI* e viene chiamato dalla funzione *binomiale.m* che fa parte di una cartella esterna *ALTRO* (ma non di *PROGRAMMI*), Matlab segnala l'errore compiuto. Se invece *binomiale.m* chiama la funzione Matlab predefinita *prod.m*, la funzione *binomiale.m* viene eseguita perfettamente.
- L'uso delle variabili è locale alla funzione. In altre parole se scriviamo

  ```
  s=fun(pi);
  ```

  durante l'esecuzione della funzione di *fun* viene assegnata alle variabili x, y una allocazione di memoria *locale* che viene rilasciata quando si esce da `fun`. Uno degli effetti è che il programma

  ```
  >>y=(2*pi);
  >>x=fun(y)
  ```

  viene eseguito correttamente nonostante ci sia un'apparente contrasto tra le x ed y della parte nella workspace di Matlab con le variabili x ed y della funzione *fun*, che peraltro hanno un significato diverso (alla x del programma viene assegnata in *fun* la variabile locale y!).

- Spesso risulta necessario avere più variabili di input o di output in una funzione. Per capirne la sintassi si consideri l'esempio

  ```
  function [s,t] = fun2(x,y)
    s=(x+y);
    t=(x-y);
  ```

- Per ulteriori dubbi sulla programmazione di una funzione si esegua da shell il comando

  ```
  >>help function
  ```

  Osservazione. Spesso nell'help di Matlab le funzioni sono in maiuscolo, ma quando debbono essere chiamate si usi il minuscolo. Per esempio, si digiti sulla workspace di Matlab/Octave

  ```
  >>help sum
  ```

  Quale risposta abbiamo

  ```
  SUM(X,DIM) sums along the dimension DIM.
  ```

  mentre

  ```
  >>a=[1 2];
  >>SUM(a);
  ??? Capitalized internal function SUM; Caps Lock may be on.
  >>sum(a)
  ans =
     3
  ```

  Conseguentemente il comando (vettoriale) `sum` che somma tutte le componenti di un vettore non può essere scritto in maiuscolo.

Esistono vari modi per definire una matrice A. Se ad esempio

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

un primo metodo è via l'assegnazione diretta

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

un secondo ciclo *for* (come vedremo successivamente)

```
for s=1:3
    for t=1:3
        A(s,t)=3*(s-1)+t;
    end
end

a
```

Per implementare questi due cicli for nidificati si scriva il programma su un file `ciclofor.m` e lo si lanci dalla shell di Matlab/Octave con il comando `ciclo``. Viene eseguito sequenzialmente quanto segue:

1. Pone $s = 1$.
2. Pone $t = 1$ e valuta la componente $(s, t) = (1, 1)$ della matrice A. Pone $t = 2$ e valuta la componente $(s, t) = (1, 2)$ della matrice A. Pone $t = 3$ e valuta la componente $(s, t) = (1, 3)$ della matrice A.
3. Pone $s = 2$.
4. Pone $t = 1$ e valuta la componente $(s, t) = (2, 1)$ della matrice A. Pone $t = 2$ e valuta la componente $(s, t) = (2, 2)$ della matrice A. Pone $t = 3$ e valuta la componente $(s, t) = (2, 3)$ della matrice A.
5. Pone $s = 3$.
6. Pone $t = 1$ e valuta la componente $(s, t) = (3, 1)$ della matrice A. Pone $t = 2$ e valuta la componente $(s, t) = (3, 2)$ della matrice A. Pone $t = 3$ e valuta la componente $(s, t) = (3, 3)$ della matrice A.

Il primo va bene per matrici di piccole dimensioni essendo poche le componenti da scrivere *manualmente*. Il secondo è più adatto a matrici strutturate come la matrice di Hilbert che ha componenti (rispetto alle righe e alle colonne) $a_{i,j} = (i + j - 1)^{-1}$.

Ricordiamo che in Matlab/Octave gli indici dei vettori e delle matrici partono rispettivamente da 1 e da $(1, 1)$, non da 0 come talvolta descritto in formulazioni matematiche.

Osserviamo che il comando `A(i,j)` permette di selezionare la componente (i, j) della matrice A. Ad esempio:

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

Tra le più comuni operazioni tra matrici ricordiamo

```
C=s*A
C=A'
C=A+B
C=A-B
C=A*B
C=A.*B
```

che assegnano alla variabile C rispettivamente il prodotto tra lo scalare s e la matrice A, la trasposta della matrice A, la somma, la sottrazione, il prodotto e il prodotto puntuale, cioè componente per componente di due matrici A, B (non necessariamente quadrate). Così se ad esempio $s = 10$,

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

$$B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$$

e `C=A.*B` allora C è la matrice

$$C = \begin{pmatrix} 5 & 12 \\ 21 & 32 \end{pmatrix}$$

Osserviamo che quello citato non corrisponde all'usuale prodotto di matrici. Infatti, se

1. A ha m righe ed n colonne,
2. B ha n righe ed p colonne,

allora $C = A * B$ è una matrice con m righe e p colonne tale che $C = (c_{i,j})$ con

$$c_{i,j} = \sum_{k=1}^{n} a_{i,k} \cdot b_{k,j}, \; i = 1, \ldots, m, \, j = 1, \ldots, p.$$

Vediamo il nostro caso particolare. Se $D = A * B$ abbiamo

$$D = \begin{pmatrix} (1 \cdot 5 + 2 \cdot 7) & (1 \cdot 6 + 2 \cdot 8) \\ (3 \cdot 5 + 4 \cdot 7) & (3 \cdot 6 + 4 \cdot 8) \end{pmatrix} = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$$

Eseguiamo dal workspace di Matlab/Octave

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

Si ottiene

$$D = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$$

Ricordando che in generale l'usuale prodotto (indicato con $*$) tra due matrici non è commutativo ci aspettiamo che $D = A * B$ non coincida con $F = B * A$. E infatti da `F=B*A` otteniamo

$$F = \begin{pmatrix} 23 & 34 \\ 31 & 46 \end{pmatrix}$$

Osserviamo che si possono scrivere matrici rettangolari (e non necessariamente solo matrici quadrate). Infatti

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

Altri comandi di comune utilizzo sono

`rand(m,n)`	matrice di numeri random di ordine m per n
`det(A)`	determinante della matrice A
`size(A)`	numero di righe e colonne di A
`hilb(n)`	matrice di Hilbert di ordine n
`eye(n)`	matrice identica di ordine n
`zeros(n)`	matrice nulla di ordine n
`ones(n)`	matrice con componenti 1 di ordine n
`diag(A)`	vettore diagonale della matrice A
`inv(A)`	inversa di A
`norm(A)`	norma di A (anche vettori!)
`cond(A)`	condizionamento di A
`eig(A)`	autovalori di A

Pensando i vettori riga (o colonna) come particolari matrici, ci si rende conto che le operazioni appena introdotte possono essere usate pure nel caso vettoriale. In altri termini, se u e v sono vettori ed s uno scalare,

```
c=s*u
c=u'
c=u+v
c=u-v
c=u.*v
```

assegnano alla variabile c rispettivamente il prodotto dello scalare s con il vettore u, la trasposta del vettore u, la somma, la sottrazione e il prodotto puntuale (cioè componente per componente) di due vettori u, v. Se u e v sono due vettori colonna la scrittura `c=u'*v` calcola l'usuale *prodotto scalare* u e v. Ricordiamo che se

$$u = (u_i)_{i=1,\ldots,m},\ v = (v_i)_{i=1,\ldots,m}$$

allora

$$u * v = \sum_{i=1}^{m} u_i \cdot v_i.$$

Osserviamo subito che in Matlab invece di $u * v$ scriviamo `c=u'*v`.
Vediamo qualche esempio

```
>> s=10;
>> u=[1; 2]
u =
     1
     2

>> v=[3; 4]
v =
     3
     4
>> s*u
ans =
     10
     20
```

```
>> u'
ans =
     1     2

>> u+v
ans =
     4
     6

>> u-v
ans =
    -2
    -2

>> u.*v
ans =
     3
     8


>> u'*v
ans =
    11

>> v'*u
ans =
    11

>> u./v
ans =
    0.3333
    0.5000
>>
```

Osserviamo che se A è una matrice $n \times n$, u un vettore colonna $n \times 1$ allora $A * u$ è l'usuale prodotto matrice-vettore. Vediamone un esempio:

```
>> A=[1 2; 3 4]
A =
     1     2
     3     4
>> u=[5 6]
u =
     5     6
>> A*u
??? Error using ==> *
Inner matrix dimensions must agree.

>> u=u'
u =
     5
     6
>> A*u
ans =
```

```
    17
    39
>>
```

Dati una matrice quadrata non singolare A di ordine n e un vettore colonna $b \in R^n$, il comando $x = A\backslash b$ calcola la soluzione del sistema lineare $Ax = b$. Così , se vogliamo risolvere il sistema

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 17 \\ 39 \end{pmatrix}$$

la cui soluzione è il vettore

$$\begin{pmatrix} 5 \\ 6 \end{pmatrix}$$

scriveremo

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

A =

     1     2
     3     4

>> b=[17; 39]

b =

    17
    39

>> x=A/b
??? Error using ==> mrdivide
Matrix dimensions must agree.

>> x=A\b

x =

    5.0000
    6.0000

>>
```

Nell'esempio esposto si è sottolineato che bisogna fare attenzione a quale *barra* utilizzare.

In molti casi risulta utile generare vettori del tipo

$$x_k = x_0 + k \cdot h, \; k = 0, \ldots, N.$$

Vediamone alcuni esempi e i relativi comandi Matlab:

- il vettore riga $u = [0;\ 1;\ 2;\ 3;\ 4]$ può essere definito dal comando Matlab

  `u=0:4;`

  In questo caso $h = 1$.
- il vettore riga $v = [0;\ 2;\ 4]$ può essere definito dal comando Matlab

  `v=0:2:4;`

  In questo caso $h = 2$.
- il vettore riga $v = [0;\ 0.2;\ 0.4;\ 0.6]$ puo' essere definito dal comando Matlab

  `v=0:0.2:0.6.`

  In questo caso $h = 0.2$.

Un modo alternativo utilizza il comando `linspace` che dati due estremi, diciamo a, b, e un intero positivo k, genera un vettore con k componenti con spaziatura costante.
Vediamo come riottenere i vettori appena introdotti:

```
>> % -------------------- ESEMPIO 1 --------------------
>> u=0:4
u =
     0     1     2     3     4
>> % SI OSSERVI CHE IL VETTORE "u" HA 5 COMPONENTI.
>> v=linspace(0,4,5)
v =
     0     1     2     3     4
>> % SI OSSERVI CHE IL VETTORE "v" HA 5 COMPONENTI.
>>
>> % -------------------- ESEMPIO 2 --------------------
>>
>> u1=0:2:4
u1 =
     0     2     4
>> % SI OSSERVI CHE IL VETTORE "u1" HA 3 COMPONENTI.
>> v1=linspace(0,4,3)
v1 =
     0     2     4
>> % SI OSSERVI CHE IL VETTORE "v1" HA 3 COMPONENTI.
>>
>> % -------------------- ESEMPIO 3 --------------------
>>
>> u2=0:0.2:0.6
u2 =
         0    0.2000    0.4000    0.6000
>> % SI OSSERVI CHE IL VETTORE "u2" HA 4 COMPONENTI.
>> v2=linspace(0,0.6,4)
v2 =
         0    0.2000    0.4000    0.6000
>> % SI OSSERVI CHE IL VETTORE "v2" HA 4 COMPONENTI.
```

Per quanto riguarda la selezione di una componente del vettore il comando è molto simile a quello visto per le matrici:

```
>> u=0:0.2:1
u =
         0    0.2000    0.4000    0.6000    0.8000    1.0000
>> u(3)
ans =
    0.4000
>>
```

Una delle istruzioni Matlab di uso più comune è `length` che calcola il numero di elementi di un vettore. Conseguentemente

```
>> x=[0; 1; 2];
>>length(x)
ans =
      3
```

Qualora risulti necessario si interroghi l'help di Matlab, ed in particolare i toolbox `elmat`, `matfun`. Inoltre si confronti con quanto descritto più estesamente in [1], p. 1015 e seguenti.

NOTA 2.1. *In Matlab/Octave si possono eseguire speciali operazioni con le matrici. Una prima operazione consiste nel selezionare delle righe (o colonne) di una matrice. Vediamo un esempio.*

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

A =

     1     2     3
     4     5     6
     7     8     9

>> % SELEZIONE DELLA PRIMA E DELLA TERZA RIGA.
>> A([1 3],:)

ans =

     1     2     3
     7     8     9

>> % SELEZIONE DELLA SECONDA COLONNA.
>> A(:,2)

ans =

     2
     5
     8

>>
```

Se A è una matrice $m \times n$ e v è un vettore di numeri interi positivi inferiori o uguali a m allora il comando

```
>> A(v,:)
```

seleziona le righe di A indicate da v. Inoltre se u è un vettore di numeri interi positivi inferiori o uguali a n allora il comando

```
>> A(:,u)
```

seleziona le colonne di A indicate da u.

Un altro comodo comando Matlab/Octave permette di impilare *vettori o matrici. Vediamo qualche esempio.*

```
>> % AGGIUNGERE RIGHE AD UNA MATRICE.
>> A=[1 2 3; 4 5 6; 7 8 9]

A =

     1     2     3
     4     5     6
     7     8     9

>> B=[10 11 12; 13 14 15]

B =

    10    11    12
    13    14    15

>> C=[A; B]

C =

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

>>  % AGGIUNGERE COLONNE AD UNA MATRICE.

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

A =

     1     2     3
     4     5     6
     7     8     9

>>
>> B=[3.5; 4.5; 5.5]

B =
```

```
    3.5000
    4.5000
    5.5000

>> C=[A B]

C =

    1.0000    2.0000    3.0000    3.5000
    4.0000    5.0000    6.0000    4.5000
    7.0000    8.0000    9.0000    5.5000

>>
```

$==$	uguale
$\backsim=$	non uguale
$<$	minore
$>$	maggiore
$<=$	minore uguale
$>=$	maggiore uguale

&&	and
||	or
$\backsim$	not
&	and (componente per componente)
|	or (componente per componente)

Vediamo alcuni esempi di test che coinvolgono le operazioni logiche sopra citate:

```
>> 1 == 1
ans =
     1
>> 0 == 1
ans =
     0
>> 1 ~= 0
ans =
     1
>>
```

Si evince che nel verificare un test, 1 sta per vero mentre 0 sta per falso.

Problema. Spiegare perchè

```
>> 0.4*3 == 1.2
ans =
     0
>>
```

Passiamo ora a vedere alcune istruzioni fondamentali in ambiente Matlab/Octave.

2.7. Ciclo for. Il *ciclo for* è un istruzione che permette di iterare una porzione di codice, al variare di certi indici. Essa viene espressa come

```
for (variabile = vettore)
...
end
```

Vediamone un esempio.

```
>> s=0;
for j=1:10
    s=s+j;
end
```

Passo passo, la variabile j assume il valore 1 ed $s = s + j = 0 + 1 = 1$. Successivamente j assume il valore 2 ed s che precedentemente valeva 1, ora essendo $s = s + j = 1 + 2$ vale 3. Si itera il processo fino a che $j = 10$ (incluso) e alla fine $s = 55$. In effetti, la somma dei primi n numeri interi positivi vale $n \cdot (n + 1)/2$ che nel nostro caso è proprio 55.

2.8. Ciclo while. Simile al *ciclo for* è il *ciclo while* che itera il processo finchè una certa condizione è verificato. In Matlab/Octave

```
while (espressione logica)
...
end
```

Vediamo un esempio.

```
>> s=0;
>> j=1;
>> while j < 10
      s=s+j;
      j=j+1;
end
>> s

s =

    45

>>
```

Qui si itera finchè j è strettamente minore di 10, dovendo essere il test $j < 10$ verificato. Quindi l'ultimo j sommato a s è 9 ed è per questo che la somma vale $45 = 9 \cdot 10/2$.
La differenza con tra ciclo for e ciclo for consiste nel fatto che `for` è utilizzato quando è noto il numero di volte in cui compiere il ciclo mentre `while` quando questa conoscenza non è nota. Così

```
>> iter=0;
>> err=100;
```

```
>> while (err > 1e-8 && iter <= 100)
          iter=iter+1;
          err=err*rand(1);
      end
>>
```

L'utente esperto noterà che quanto appena scritto è comunque equivalente a

```
>> err=100;
>> for iter=1:100
          err=err*rand(1);
          if err <= 1e-8
             return;
          end
    end
>>
```

Il `return` consiste in un'uscita immediata dal ciclo for nonostante sia *iter* < 100.

Osserviamo però che

```
>> iter=0;
>> err=100;
>> while err > 1e-8
         iter=iter+1;
         err=err*rand(1);
   end
>>
```

non è equivalente a

```
>> err=100;
>> for iter=1:100
          err=err*rand(1);
          if err <= 1e-8
             return;
          end
   end
>>
```

in quanto il ciclo while potrebbe concludersi dopo oltre 100 iterazioni.

NOTA 2.2. *All'interno di cicli while o for il comando di* `return` *può essere sostituito dal comando* `break`.

```
>> err=100;
for iter=1:100
          err=err*rand(1);
          if err <= 1e-8
             break;
          end
    end
>>
```

Si sottolinea che, come si evince dall'help di Matlab, le due istruzioni `break` *e* `return` *non sono in generale equivalenti. A tal proposito si confronti*

```
A=[3 1; 4 5];
if isempty(A)
   d = 1;
   return
else
   d=det(A);
end
d=d+1
```

con

```
A=[3 1; 4 5];
if isempty(A)
   d = 1;
   break
else
   d=det(A);
end
d=d+1
```

2.9. Istruzione condizionale. L'istruzione condizionale esegue sequenzialmente alcune operazioni, se certi test vengono soddisfatti.

```
if (espressione logica)
...
else
...
end
```

Il ramo `else` talvolta non è necessario e possiamo quindi scrivere un'istruzione del tipo

```
if (espressione logica)
...
end
```

Vediamo un esempio.

```
a = 50;
if a > 0
     s=1;
else
     if a < 0
        s=-1;
     else
        s=0;
     end
end
fprintf('a: %5.5f s: %1.0f',a,s);
```

E facile vedere che questo codice calcolo il segno di a (supposto $\text{sign}(0) = 0$).

Esempio da provare. Dire cosa calcola il seguente codice:

```
s=1; j=1;
while j < 10
      s=s*j;
      j=j+1;
end
```

Esercizio facile. Tenendo a mente l'esempio 3, scrivere una funzione che dato un numero a fornisce come output la variabile s avente quale valore sign(a). Si ricordi che la funzione non si può chiamare `sign`, in quanto tale funzione è già presente in Matlab/Octave.

2.10. La grafica di Matlab. Non è sufficiente produrre dei (buoni) risultati numerici, bensì è anche necessario poterli osservare e valutare in modo adeguato. Solitamente questa operazione richiede la produzione di qualche tipo di grafico, per cui l'esistenza di capacità grafiche è un elemento fondamentale di un ambiente di calcolo. Matlab possiede capacità grafiche evolute ed Octave si appoggia a Gnuplot (applicativo anch'esso open-source). A tal proposito consideriamo il seguente esempio (studio della funzione $\sin(1/x)$ nell'intervallo $[\text{eps}, 1)$, dove $\text{eps} = 2.2204e-016$)

```
x=eps:0.01:1;
y=sin(1./x);
plot(x,y,'r-');
```

Viene eseguito il plot della funzione $\sin(1/x)$ campionandola nei punti $x = \text{eps} + k \cdot 0.01$ tra 0 e 1. Osserviamo che "r" sta per *rosso*, e "-" esegue l'interpolazione lineare a tratti tra i valori assunti dalla funzione; per ulteriori delucidazioni sul comando di plot si digiti nella shell di Matlab/Octave

```
help plot
```

Per scrivere tale programma, si digiti su shell Unix/Linux

```
pico studiofunzione.m
```

e quindi si copi il codice utilizzando tale editor. Quindi si digiti contemporaneamente `CTRL` ed `X` e si prema il tasto `ENTER`. Quindi dalla shell di Matlab/Octave si scriva `studiofunzione`. Se tutto è stato eseguito correttamente viene visualizzato il grafico della funzione.
Attenzione: usare l'invio per passare da una riga alla successiva. Inoltre, nelle prime versioni di Octave la funzione `plot` può avere delle stringhe relative al terzo componente che possono risultare diverse rispetto al corrispettivo in Matlab.
Nel plottare gli errori, si ricorre spesso alla *scala logaritmica*, mediante il comando `semilogy`. Vediamone un esempio. Digitiamo sia

```
>> x=0:15;
>> y=10.^(-x);
>> % GRAFICO DELLE COPPIE (x,y) RAPPRESENTATE CON "o"
>> % E UNITE DA SEGMENTI.
>> plot(x,y,ro-');
>>
```

che

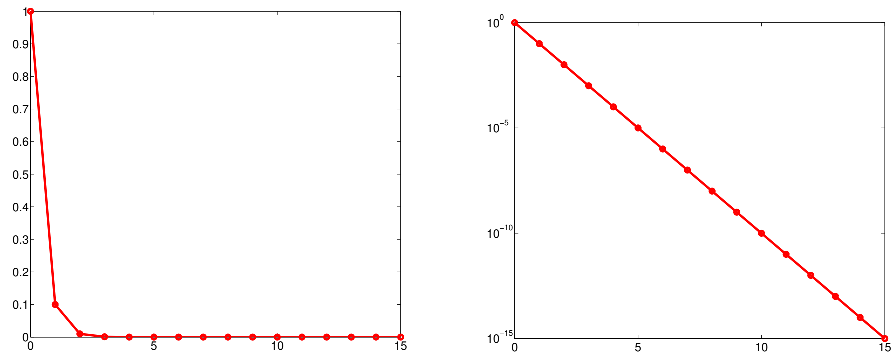

FIGURA 2.1. *Differenza tra* `plot` *e* `semilogy` *nel rappresentare le coppie* $(t, 10^{-t})$ *per* $t = 1, \ldots, 15$.

```
>> x=0:15;
>> y=10.^(-x);
>> % GRAFICO IN SCALA SEMILOGARITMICA DELLE COPPIE (x,y)
>> % RAPPRESENTATE CON "o" E UNITE DA SEGMENTI.
>> semilogy(x,y,'ro-');
>>
```

In questo test, si plottano le coppie $(t, 10^{-t})$ per $t = 0, \ldots, 15$. Si nota come il comando `plot` non mostri adeguatamente la differenza tra $10^0 = 1$ e 10^{-15}, cosa che è invece palese nel grafico in scala logaritmica ottenuto con `semilogy`. L'uso di quest'ultima scala sarà importante nello studio dell'errore dei metodi numerici implementati in seguito.

2.11. Display. Per il display di un risultato si utilizzano `disp`, `fprintf`. Per maggiori dettagli si provi ad esempio

```
>>help disp
```

Per capire come si usi il comando `fprintf` si consideri il programma

```
% ESEMPIO SUL COMANDO fprintf.
x = 0:.1:1; y = exp(x);
for index=1:length(x)
     fprintf('\t [x]: %4.2f [y]: %9.2e', x(index), y(index));
end
```

Commentiamo il codice e conseguentemente l'utilizzo di `fprintf`.

- `% ESEMPIO SUL COMANDO fprintf.`: il `%` stabilisce che è un commento al programma.
- `x = 0:0.1:1; y = exp(x);`: si crea il vettore riga
$$x = [0,\, 0.1,\, \ldots,\, 0.9,\, 1]$$
e in seguito si valuta puntualmente la funzione esponenziale in x. In altri termini
$$y = [\exp(0),\, \exp(0.1),\, \ldots,\, \exp(0.9),\, \exp(1)].$$

Le funzioni elementari precedentemente introdotte come `abs`, `sin`, ..., `cosh` godono pure di questa proprietà e rendono Matlab un linguaggio di tipo *vettoriale*. Dal punto di vista pratico il vantaggio è che molte operazioni possono essere eseguite senza utilizzare il ciclo `for`. Per capire questa particolarità si esegua una programma Matlab che calcoli il vettore

$$y = [\exp(0), \exp(0.1), \ldots, \exp(0.9), \exp(1)]$$

con un ciclo `for`.

- `for index=1:length(x)`: istruzione del ciclo `for`. Si noti che si itera per *index* che va da 1 fino alla lunghezza del vettore x, cioè 11.
- `fprintf('\n \t [x]:%4.2f [y]:%9.2e',x(index),y(index));` : in questa parte si effettua componente per componente il display dei valori nei vettori x e y. Il significato di `%4.2f` è che della componente di indice *index* del vettore x vengono messe in display *al massimo 4 cifre prima della virgola e due cifre dopo la virgola* in notazione decimale. Il significato di `%9.2e` è che della componente di indice *index* del vettore y vengono messe in display *al massimo 9 cifre prima della virgola e due cifre dopo la virgola* in notazione esponenziale. Per capire cosa sia quest'ultima rappresentazione (quella decimale è comunemente utilizzata fin dalle scuole elementari), si digiti su workspace (usare l'invio per passare da uno >> al successivo). Per quanto concerne `\n` e `\t`, il primo manda a capo prima di stampare qualcosa su schermo, il secondo crea uno spazietto.

  ```
  >>format short e
  >>100*pi
  ans=
  3.1416e+002
  ```

  Il numero $100\,\pi = 314.159265358979\ldots$ viene rappresentato come $3.1416e \cdot 10^2$. Questa notazione è particolarmente utile per rappresentazioni di errori. Digitiamo su shell (senza contare gli 0!)

  ```
  >> fprintf('[VALORE]: %2.18f', c)
  [VALORE]: 0.000000000001300000
  >>
  >> fprintf('[VALORE]: %20.18f', c)
  [VALORE]: 0.000000000001300000
  >>
  ```

  Non risulta molto chiaro a prima vista capire quanti 0 ci siano, mentre ciò risulta ovvio scrivendo

  ```
  >>fprintf('[VALORE]: %2.2e', c)
  [VALORE]: 1.30e-012
  ```

- `end`: fine del ciclo `for`.

2.12. Sulle operazioni puntuali. Una delle proprietà di Matlab/Octave è di rappresentare operazioni vettoriali. Osserviamo che le funzioni elementari precedentemente citate quali `abs`, ..., `cosh` sono implicitamente vettoriali. Per convincersi sperimentiamo sul workspace di Matlab/Octave

```
>> x=-1:0.5:1
x =
   -1.0000   -0.5000         0    0.5000    1.0000
>> abs(x)
ans =
    1.0000    0.5000         0    0.5000    1.0000
>>
```

Per le operazioni moltiplicative bisogna usare il **.** per lavorare vettorialmente. Di conseguenza:

```
>> x=-1:0.5:1
x =
   -1.0000   -0.5000         0    0.5000    1.0000
>> x.^2
ans =
    1.0000    0.2500         0    0.2500    1.0000
>> sin(x.^3)
ans =
   -0.8415   -0.1247         0    0.1247    0.8415
>> 1./x
Warning: Divide by zero.
ans =
    -1    -2   Inf     2     1
>>
```

Vediamo degli altri esempi in cui si usano vettorialmente le funzioni elementari.

```
>> x=0:pi/2:2*pi
x =
         0    1.5708    3.1416    4.7124    6.2832

>> sin(x)
ans =
         0    1.0000    0.0000   -1.0000   -0.0000

>> cos(x)
ans =
    1.0000    0.0000   -1.0000   -0.0000    1.0000

>> sin(x+2*pi)
ans =
   -0.0000    1.0000    0.0000   -1.0000   -0.0000

>> cos(x+2*pi)

ans =

    1.0000    0.0000   -1.0000   -0.0000    1.0000

>> z=[1 4 9 16]
z =
     1     4     9    16
```

```
>> sqrt(z)
ans =
     1     2     3     4

>> zz=sqrt(z)
zz =
     1     2     3     4

>> zz.^2
ans =
     1     4     9    16

>> zz^2
??? Error using ==> mpower
Matrix must be square.

>>
```

2.13. Altri comandi. Esistono vari comandi Matlab/Octave di uso comune. Ne citiamo per semplicità alcuni.

1. Per l'esecuzione di un programma `mioprogramma.m` creato dall'utente si digiti da workspace di Matlab (con Matlab avente quale directory attuale quella contenente il file `mioprogramma.m`)

   ```
   >>mioprogramma
   ```

2. Per ulteriori toolboxes predefinite in Matlab si digiti

   ```
   >>help
   ```

3. il comando `cputime` permette come segue di sapere il tempo impiegato da un processo. Si consideri a tal proposito la porzione di codice

   ```
   puntoiniziale=cputime;
   s=0; for i=1:100 s=s+i; end
   puntofinale=cputime;
   tempoimpiegato=puntofinale-puntoiniziale;
   ```

   Il valore della variabile `tempoimpiegato` consiste del tempo impiegato per svolgere le istruzioni

   ```
   s=0; for i=1:100 s=s+i; end
   ```

2.14. Il diary. Uno dei comandi più interessanti di Matlab è il `diary` che scrive su file quanto visualizzato nel workspace di Matlab/Octave. Vediamone un esempio dal workspace di Matlab/Octave:

```
>> diary on
>> s=2;
>> t=5;
>> u=s+t
```

```
u =
     7
>> diary off
```

Nella directory attuale (vista cioè da Matlab/Octave) troviamo un file di testo *diary* (senza estensione). Lo apriamo con un editor (ad esempio digitando sulla shell di Linux `pico diary`). Il file contiene quanto apparso sulla shell di Matlab/Octave ad eccezione del prompt $>>$. Osserviamo che può essere utile per vedere a casa quanto fatto a lezione sul workspace di Matlab/Octave.

Per un uso che scriva files di testo ben scritti, si suggerisce un uso del tipo

```
>> diary filename.txt
>> x=0:0.01:1;
>> y=x+1;
>> diary off;
```

in cui nel file di testo `filename.txt`, aperto ad esempio con WordPad in Windows XP si trova

```
x=0:0.01:1;
y=x+1;
```

NOTA 2.3. *Si sottilinea che l'uso di diary per registrare quanto scritto nel workspace, si limita a questo scopo. I programmi e le funzioni, specie se i relativi listati sono composti di molte righe di codice, sono usualmente scritti su files con estensione* `.m`. *Di conseguenza è molto rudimentale scrivere i programmi esclusivamente su workspace e poi salvare tutto come diary, con l'intento di usare successivamente quanto fatto.*

Nelle ultime versioni di Matlab, come ad esempio la vecchia release 6.1, si possono scrivere programmi e funzioni direttamente da un editor interno (noto come `meditor`*). Per aprire files/functions o salvare gli stessi basta utilizzare le relative e classiche icone nella barra dei menu di Matlab. In Octave invece bisogna usare un editor esterno (come ad esempio* `pico` *o* `wordpad`*).*

2.15. Facoltativo: come caricare dati da files. In molti casi, i dati sono scritti su un file e si desidera *caricarli* nel workspace o all'interno di un programma per poter eseguire un esperimento numerico. Per tale scopo, in Matlab/Octave esiste la function `load`. L'help di Matlab è molto tecnico e dice in molto molto criptico come dev'essere scritto il file. Si capisce che si deve scrivere qualcosa del tipo

```
load nomefile variabili
```

ma non molto di come deve essere scritto il file. Vediamo quindi un esempio che possa spiegare meglio l'utilizzo di `load`, magari aiutandosi con [3] oppure [5]. Supponiamo di aver registrato il file `PDXprecip.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 evidentemente le ascisse e le ordinate di alcune osservazioni (dal titolo si capisce che sono precipitazioni in alcuni giorni dell'anno). E' chiaro che il contenuto è scritto come una matrice con 12 righe e 2 colonne. Matlab/Octave vede questo file come una matrice le cui componenti sono quelle della *variabile* `PDXprecip`. Il comando `load` carica questa *variabile* nel workspace di Matlab/Octave. Di conseguenza:

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

mese =

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

>> precip=PDXprecip(:,2)

precip =

    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,precip,'o')
```

prima immagazzina le colonne di `PDXprecip.dat` rispettivamente nelle variabili `mese` e `precip` per poi eseguirne il grafico (si veda la figura).

3. Facoltativo: il comando `who`. A volte può essere utile sapere quali variabili sono presenti nel workspace. A tale scopo è utile il comando `who`. Vediamo un esempio

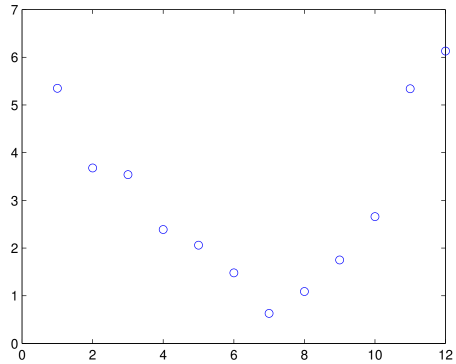

FIGURA 2.2. *Grafico di alcuni dati immagazzinati su file.*

```
>> % CANCELLIAMO TUTTE LE VARIABILI DAL WORKSPACE
>> % CON IL COMANDO clear all.
>> clear all
>> who
>> % DOPO L'ENTER WHO NON DICE NULLA ...
>> x=5;
>> y=6;
>>% SUL WORKSPACE ADESSO CI SONO LE VARIABILI x, y.
>> who

Your variables are:

x  y

>>
```

Dopo aver *cancellato* tutte le variabili del workspace col comando `clear all`, è evidente che `who` non produce effetto. Dopo aver definito x e y, il comando `who` ricorda che sono presenti le sole variabili x e y.

4. Esercizi obbligatori.

1. E' noto che per $|x| < 1$ si ha

$$\frac{1}{1-x} \approx 1 + x + x^2 + x^3 + \ldots.$$

   - Fissato $h = 0.01$ e i punti $x = -1+h, -1+2h, \ldots, 1-2h, 1-h$ si valutino in tali punti le funzioni

$$\frac{1}{1-x}$$

     e

$$1 + x + x^2 + x^3$$

     e si plottino i corrispettivi grafici in una stessa finestra.

Suggerimento: usare i comandi Matlab/Octave `hold on` e `hold off` (aiutarsi con l'help di Matlab/Octave).

```
>> help hold
 HOLD   Hold current graph
    HOLD ON holds the current plot and all axis properties
    so that subsequent graphing commands add to the
    existing graph.

    HOLD OFF returns to the default mode whereby PLOT
    commands erase the previous plots and reset all axis
    properties before drawing new plots.

    HOLD, by itself, toggles the hold state.
    HOLD does not affect axis autoranging properties.

    HOLD ALL holds the plot and the current color and
    linestyle so that subsequent plotting commands will
    not reset the color and linestyle.

    HOLD(AX,...) applies the command to the Axes object
    AX.

    Algorithm note:
    HOLD ON sets the NextPlot property of the current
    figure and axes to "add".
    HOLD OFF sets the NextPlot property of the current
    axes to "replace".

    See also ishold, newplot, figure, axes.

    Reference page in Help browser
       doc hold

>>
```

- Si plotti in scala semilogaritmica il grafico dell'errore assoluto $\left|\frac{1}{1-x} - (1 + x + x^2 + x^3)\right|$. Perchè quest'ultima quantità è particolarmente piccola per $x = 0$?

  Suggerimento: riguardo all'implementazione in Matlab/Octave, si faccia attenzione all'utilizzo delle operazioni puntuali di Matlab.

```
>> x=0:0.5:1;
>> y=1+x+x^2
??? Error using ==> mpower
Matrix must be square.

>> y=1+x+x.^2

y =

    1.0000    1.7500    3.0000
```

```
>>
```

Indicare in cosa consista l'errore di sintassi compiuto da Matlab/Octave.

2. Eulero [2, p.47] ha dimostrato che tanto

$$\sum_{k=1}^{\infty} \frac{1}{k^2}$$

quanto

$$(\ln 2)^2 + \sum_{k=1}^{\infty} \frac{1}{k^2 \cdot 2^{k-1}}$$

convergono a $\frac{\pi^2}{6}$. Calcolare per $N = 1000$ le somme

$$\sum_{k=1}^{N} \frac{1}{k^2}$$

e

$$(\ln 2)^2 + \sum_{k=1}^{N} \frac{1}{k^2 \cdot 2^{k-1}}.$$

Valutare

$$\left| \sum_{k=1}^{N} \frac{1}{k^2} - \frac{\pi^2}{6} \right|$$

e

$$\left| (\ln 2)^2 + \sum_{k=1}^{N} \frac{1}{k^2 \cdot 2^{k-1}} - \frac{\pi^2}{6} \right|.$$

Per effettuare la somma usare il comando `sum` di cui vediamo un esempio.

```
>> v=[1 3 56 23];
>> somma=0;
>> for ii=1:length(v)
somma=somma+v(ii);
end
>> somma

somma =

    83

>> sum(v)

ans =

    83

>>
```

Si osservi inoltre che

```
>> somma=0;
>> for k=1:1000
somma=somma+(1/k);
end
>> v=sum(1./(1:1000));
>> somma

somma =

    7.4855

>> v

v =

    7.4855

>>
```

Adattare questi ultimi calcoli alla somma richiesta.

5. Esercizi facoltativi.

1. **Esercizio di media difficoltà** Il calcolo del determinante [6] di una generica matrice quadrata di ordine n con:
   - la nota regola di Laplace necessita di circa $n!$ operazioni moltiplicative.
   - l'algoritmo di Gauss necessita circa $n^3/3$ operazioni moltiplicative.

   Negli anni la velocità dei microprocessori è ampiamente aumentata, come si può notare in [8]. In merito, la lista dei computers più veloci degli ultimi decenni relativamente a quanto affermato da [8], è

   - 1940: Z2, 1 operazione al secondo;
   - 1950: ENIAC, 5 mila di operazioni al secondo;
   - 1960: UNIVAC LARC, 500 mila di operazioni al secondo;
   - 1970: CDC 7600, 36 megaflops;
   - 1980: CRAY 1, 250 megaflops;
   - 1990: NEC SC-3\44R, 23,2 gigaflops;
   - 2000: IBM ASCI White, 7,226 teraflops;
   - 2008: IBM Roadrunner, 1,6 petaflops;

   Per chiarirsi le idee con i termini usati, ricordiamo che (cf. [7]):

   - megaflop: 10^6 operazioni al secondo;
   - gigaflop: 10^9 operazioni al secondo;
   - teraflop: 10^{12} operazioni al secondo;
   - petaflop: 10^{15} operazioni al secondo.

   Si supponga ora di dover calcolare il determinante di una matrice di ordine 100, rispettivamente con il metodo di Laplace e di Gauss, supponendo di dover effettuare solo operazioni moltiplicative. Quanti anni si impiegano al variare del computer? Suggerimento: per calcolare il fattoriale di un numero si usi la funzione `gamma`

```
>> gamma(1)
ans =
     1
>> gamma(2)
ans =
     1
>> gamma(3)
ans =
     2
>> gamma(4)
ans =
     6
>> gamma(5)
ans =
    24
>>
```

2. **Esercizio facoltativo**. Si implementi una funzione `fatt` che calcola il fattoriale *fatt* di un numero naturale n, ricordando che

$$\text{fatt}(n) := n! := 1 \cdot \ldots \cdot n.$$

La si utilizzi per calcolare mediante un ciclo `for` i fattoriali dei numeri naturali n tra 1 e 20. Si visualizzino i risultati ottenuti. Per gli stessi numeri naturali si usi la funzione di Matlab `gamma` per valutare $\Gamma(n)$. Che relazione sussiste tra $\Gamma(n)$ e $\text{fatt}(n)$?
Facoltativo: si utilizzi la funzione fattoriale per calcolare il binomiale

$$A = \left(\begin{array}{c} n \\ k \end{array} \right) = \frac{n!}{k!(n-k)!}.$$

3. **Esercizio facile**. La posizione di un punto P nello spazio tridimensionale può essere rappresentata da una terna del tipo (x, y, z), dove x, y e z sono le componenti lungo i tre assi cartesiani. Se due punti P_1 e P_2 sono rappresentati dai valori (x_1, y_1, z_1) e (x_2, y_2, z_2), eseguire un programma Matlab che calcoli la distanza euclidea tra questi due punti, cioè

$$d := \sqrt{((x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2)}.$$

4. **Esercizio facile**. Si vuole trovare la media aritmetica di $a = 1.7 10^{308}$ e $b = 1.6 10^{308}$. Calcolare $(a + b)/2$ e $b + (a - b)/2$. Sono i risultati gli stessi e perchè ?

5. **Esercizio medio**. Si vuole calcolare la norma Euclidea del vettore

$$[v_1, v_2] = [3 10^{307}, 4 10^{307}].$$

Effettuare $\sqrt{v_1^2 + v_2}$ e $v_2 \cdot \sqrt{(v_1/v_2)^2 + 1}$. Dire se si ottengono gli stessi risultati e giustificare la risposta.

6. **Esercizio facoltativo**. Si considerino le matrici di Hilbert A_j per $j = 1, 2, \ldots, 20$. Si osservi cha la loro costruzione risulta facilitata dal comando `hilb` (si chiami

l'help di Matlab per ulteriori informazioni). Utilizzando un ciclo `for`, si calcoli al variare di j il determinante $det(A_j)$ (via il comando `det`). Immagazzinare $det(A_j)$ in un vettore e visualizzare al variare di j tale risultato sul monitor (via il comando `fprintf`). Cosa succede se al posto di `det(A)` utilizzo `prod(eig(A))`? Ricordare che `eig(A)` fornisce il vettore degli n autovalori $\lambda_1, \ldots, \lambda_n$ della matrice quadrata A (avente n righe e n colonne) e che

$$det(A) = \lambda_1 \cdot \ldots \cdot \lambda_n.$$

7. **Esercizio facoltativo**. Dato un triangolo rettangolo di angolo $\theta > 0$ e ipotenusa $r > 0$, calcolare la lunghezza dei suoi cateti.

8. **Esercizio facoltativo, non banale**. Trascurarando l'attrito dell'aria, calcolare la gittata di un proiettile sparato con una velocità iniziale $v_0 > 0$ ed un angolo $\theta > 0$ rispetto al suolo. Nelle ipotesi citate, se la velocità iniziale e' pari a 20 m/s e l'angolo θ varia da 1 a 90 gradi con incrementi di 1 grado, determinare l'angolo θ in corrispondenza del quale la gittata è massima.

6. Online. Si suggerisce consultare, qualora necessario, i seguenti links:

1. http://it.wikipedia.org/wiki/GNU_Octave
2. http://en.wikipedia.org/wiki/MATLAB
3. http://it.wikipedia.org/wiki/MATLAB
4. http://www.gnu.org/software/octave/doc/interpreter/
5. http://kgptech.blogspot.com/2005/07/matlab.html

mentre quale interesse generale, si citano

- http://www.mathworks.com/company/newsletters/news_notes/clevescorner/dec04.html
- http://www.math.ucla.edu/~getreuer/matopt.pdf in cui si descrive come velocizzare i codici Matlab. Si sottolinea che richiede quale prerequisito una buona conoscenza di Matlab.

RIFERIMENTI BIBLIOGRAFICI

[1] V. Comincioli, *Analisi Numerica, metodi modelli applicazioni*, Mc Graw-Hill, 1990.
[2] , W. Dunham, *Euler, The Master of Us All*, The Mathematical Association of America, Dolciani Mathematical Expositions No 22, 1999.
[3] The MathWorks Inc., *Matlab, Load*, http://www.mathworks.com/access/helpdesk/help/techdoc/ref/load.html.
[4] The MathWorks Inc., *Numerical Computing with Matlab*, http://www.mathworks.com/moler.
[5] G. Recktenwald, *Loading Data into MATLAB for Plotting*, http://web.cecs.pdx.edu/~gerry/MATLAB/plotting/loadingPlotData.html.
[6] Wikipedia, Algoritmo, http://it.wikipedia.org/wiki/Algoritmo.
[7] Wikipedia, FLOPS, http://en.wikipedia.org/wiki/FLOPS.
[8] Wikipedia, Supercomputer, http://it.wikipedia.org/wiki/Supercomputer.