Introduzione a Matlab per Ingegneria dell'Energia [1]

A. Sommariva[2]

Abstract

Ambienti software dedicati al calcolo numerico. Interfaccia grafica di Matlab (la command window). Le variabili Matlab. Matlab: operazioni e funzioni elementari predefinite (funzioni elementari). Assegnazioni. I vettori in Matlab (definire i vettori in Matlab, lunghezza e dimensione di un vettore, vettori equispaziati, accesso alle componenti di un vettore, operazioni e funzioni vettoriali). Definizione di funzioni matematiche. La grafica in Matlab (la scala semilogaritmica, i comandi `legend` e `title`). Le stringhe di testo. I comandi `format`, `disp`, `fprintf`. Operazioni con le matrici (sulle matrici e vettori). Definizione di una matrice. Operatori di relazione e logici. Le istruzioni condizionali. Ciclo for. Ciclo while (legame tra ciclo for e ciclo while). Gestione dei files dei dati (come caricare dati da files, salvare dati su file). Altri comandi.

Ultima revisione: 23 dicembre 2018

1. Ambienti software dedicati al calcolo numerico

Il proposito di questa nota è su come utilizzare Matlab sotto Linux, Windows o MacOS.

L'ambiente MATLAB (sigla per Matrix Laboratory) ha avuto origine nel 1983 ed ha ricevuto un certo successo per la semplicità dei suoi comandi (cf. [4])

Per l'utilizzo da casa, l'università di Padova dispone di una utenza CAMPUS, che prevede il download gratuito di tale programma, consentendo ad ogni studente di utilizzarlo nel proprio computer. In tal senso, si consideri [7].

Per chi non volesse utilizzare Matlab, che è un linguaggio commerciale, si osservi che

- GNU Octave presenta così tante similitudini da essere utilizzato con codici MATLAB con alte probabilitá di non presentare alcun errore di sintassi (cf. [8]);
- SCILAB, nonostante abbia alcune similitudini con Matlab presenta profonde differenze con lo stesso.

Esistono altri linguaggi di programmazione, come ad esempio C, Fortran, Phyton, Java, spesso utilizzati in ambiente *numerico*, ma la loro struttura risulta molto diversa da Matlab.

Per quanto riguarda il cosidetto *calcolo simbolico*, tra i programmi più comuni citiamo Mathematica, Maxima e Maple, che però non sono adatti ai propositi del corso.

2. Interfaccia grafica di Matlab

Al fine di accedere agli ambienti Matlab tipicamente

- si clicca su un'icona di Matlab oppure
- dalla shell di Linux, si digita il comando `matlab` seguito dal tasto di invio.

Subito dopo si apre l'interfaccia grafica che, a meno di preferenze diverse, è composta di 4 ambienti

- *Workspace*: una sottofinestra che mostra il nome e il contenuto delle variabili immagazzinate (ad esempio numeri, strutture più complesse come vettori, o matrici),
- *Current directory*; una sottofinestra che contiene informazioni sulla cartella in cui si sta lavorando, ad esempio i files presenti nella cartella stessa,
- *Command history*: una sottofinestra che contiene una lista di tutti i comandi digitati;
- *Command window*: una sottofinestra nella quale vengono inseriti i comandi o dalla quale viene *lanciata* l'esecuzione dei programmi.

2.1. La command window

La *command window* permette di interagire con l'ambiente di calcolo di Matlab e inizialmente si presenta come una linea di comando >> detta *prompt*.

Ci sono due usi tipici della stessa,

- si scrivono una serie di istruzioni al fine di raggiungere un qualche risultato numerico;
- si lanciano programmi Matlab, salvati su un file di testo con estensione `.m`, e ci si attende nuovamente una qualche risposta numerica.

Tipicamente, per familiarizzare con l'ambiente Matlab, è bene dapprima cominciare a scrivere una breve sequenza di comandi sulla command window per poi passare a implementare programmi veri e propri.

I programmi li commenteremo adeguatamente mediante il carattere "`%`" seguito da un adeguato testo illustrativo.

3. Le variabili Matlab

In questa sezione descriviamo il concetto di *variabile*, discutendone i casi più comuni.

Definizione. Una variabile, in informatica, è un contenitore di dati situato in una porzione di memoria (una o pi ù locazioni di memoria) destinata a contenere valori, suscettibili di modifica nel corso dell'esecuzione di un programma. [13]

Una variabile è caratterizzata da un nome (inteso solitamente come una sequenza *ammissibile* di caratteri e cifre, ovvero che non cominci con un numero, sia senza spazi, e non sia utilizzata dall'ambiente per altri propositi).

I *valori* di uso più comune in Matlab sono

- numeri (talvolta detti *scalari* e si dice aventi *dimensione* 1×1, ovvero 1 riga di 1 elemento) come ad esempio 3.141592653589793, 0, -1;
- vettori [14], ovvero una lista di n-numeri (ognuno dei quali si dice *elemento* o *componente*), che può essere scritta
  1. in orizzontale ed in tal caso è detta *vettore riga* (si dice avente *dimensione* $1 \times n$, ovvero 1 riga di n elementi), come ad esempio la *coppia*
  $$(5, 7),$$
  avente dimensione 1×2 o la *tripla*
  $$(3.1567, -234.343546, 0.4536),$$
  avente dimensione 1×3;
  2. in verticale ed in tal caso è detta *vettore colonna* (e si dice avente *dimensione* $n \times 1$ ovvero n righe di 1 elementi), come ad esempio la *coppia*
  $$\begin{pmatrix} 5 \\ 8 \end{pmatrix}$$
  avente dimensione 2×1, o la *tripla*
  $$\begin{pmatrix} 3.1415 \\ -1 \\ 2.7182 \end{pmatrix}.$$
  avente dimensione 3×1;
- una matrice una tabella ordinata di elementi [10], consistente di m vettori riga di dimensione $1 \times n$, ad esempio la matrice *rettangolare* in cui m può essere diverso da n (e si dice avente *dimensione* $m \times n$, ovvero m righe di n elementi), ovvero
$$\begin{pmatrix} 3.1415 & 24.2 \\ -1 & 16.2 \\ 0 & 2.7182 \end{pmatrix}$$
oppure la matrice *quadrata*, in cui $m = n$ (e si dice avente *dimensione* $n \times n$), ovvero
$$\begin{pmatrix} 0.215 & 4.22 \\ -0155 & 6.82 \end{pmatrix}.$$
- una *stringa* [11], ovvero una sequenza di caratteri alfanumerici con un ordine prestabilito.

4. Matlab: operazioni e funzioni elementari predefinite

In questa sezione, mostreremo alcuni comandi di MATLAB che risulteranno utili per implementare gli algoritmi descritti in seguito.

Inizialmente descriveremo le operazioni tra scalari per poi ripensarle successivamente una sezione successiva in termini vettoriali e più in generale matriciali.

Le comuni operazioni aritmetiche sono indicate con

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

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

```
>> 2+3
ans =
     5
>> 2-3
ans =
    -1
>> 2*3
ans =
     6
>> 2/3
ans =
   6.6667e-01
>> 2^3
ans =
     8
>>
```

Nota. 4.1. *Matlab, oltre ai numeri macchina, include quantità speciali come*

- `-Inf`*: ovvero meno infinito, frutto ad esempio di calcoli del tipo* `-5/0`*;*
- `+Inf`*: ovvero più infinito, frutto ad esempio di calcoli del tipo* `5/0`*;*
- `NaN`*: not a number, usualmente frutto di operazioni che danno luogo a indeterminatezza come* `0/0`*;*

```
>> -5/0
ans =
  -Inf
>> +5/0
ans =
   Inf
>> 0/0
ans =
   NaN
>>
```

Altre costanti di interesse sono

- `eps`*: è la precisione di macchina, ovvero la distanza tra 1 e il primo numerico macchina successivo, in doppia precisione e vale circa* $2.220446049250313e-16$*;*
- `pi`*: ovvero* $\pi = 3.14159265358979\ldots$*;*
- `realmax`*: è circa* $1.797693134862316e{+}308$ *ed è il più grande numero macchina normalizzato ed in precisione doppia;*
- `realmin`*: è circa* $2.225073858507201e-308$ *ed è il più piccolo numero macchina positivo, normalizzato ed in precisione doppia (si noti che è un numero inferiore di* `eps`*).*

Nota. 4.2. *Matlab ha per gli scalari una notazione esponenziale. Per comprenderla bene forniamo alcuni esempi direttamente dalla command window:*

```
>> 0.0001 % di seguito si prema il tasto di INVIO.
ans =
     1.000000000000000e-04
>> exp(1)
ans =
     2.718281828459046e+00
>>
```

In altri termini, spesso il numero viene descritto come un numero $m \in [1, 10)$ *seguito da* `e-k` *o* `e+k` *, e con ciò si intende rispettivamente* $m \cdot 10^{-k}$ *o* $m \cdot 10^{+k}$*.*

La notazione esponenziale è molto comoda per rappresentare numeri piccoli in modulo, come ad esempio errori, o numeri molto grandi in modulo.

Ad esempio, non è immediato capire quanto sia un errore che valga 0.00000001 *mentre lo è nella forma* $1.0e-08$*, ovvero* $1 \cdot 10^{-8}$*.*

4.1. Funzioni elementari

Funzioni elementari comunemente usate sono

- le funzioni

`abs`	valore assoluto	`sqrt`	radice quadrata
`sign`	segno	`rem`	resto della divisione

- le funzioni trigonometriche e le loro inverse

`sin`	seno	`cos`	coseno
`tan`	tangente	`cot`	cotangente
`asin`	arco seno	`acos`	arco coseno
`atan`	arco tangente	`acot`	arco cotangente

- le funzioni esponenziali e le loro inverse

`exp`	esponenziale
`log 2`	logaritmo base 2
`log10`	logaritmo base 10
`log`	logaritmo naturale

- le funzioni iperboliche

`sinh`	seno iperbolico
`cosh`	coseno iperbolico
`tanh`	tangente iperbolico
`asin`	arco seno iperbolico
`acosh`	arco coseno iperbolico
`atanh`	arco tangente iperbolica

- le funzioni di parte intera e arrotondamento

`fix`	arrotondamento verso 0
`round`	arrotondamento verso l'intero più vicino
`floor`	arrotondamento verso $-\infty$
`ceil`	arrotondamento verso $+\infty$

Vediamo alcuni esempi:

- funzioni trigonometriche e le loro inverse

```
>> sin(pi)
ans =
      1.22464679914735e-16
>> cos(pi)
ans =
      -1
>> tan(pi/4)
ans =
      1
>> atan(1)
ans =
      0.785398163397448
>>
```

- funzioni esponenziali e le loro inverse

```
>> exp(0)
ans =
      1
>> log(1)
ans =
      0
>>log10(10)
ans =
      1
>>
```

- funzioni iperboliche

```
>> sinh(0)
ans =
      0
>> cosh(0)
ans =
      1
>> acosh(1)
ans =
      0
>>
```

- le funzioni di parte intera e arrotondamento

```
>> fix(pi) % arrotondamento verso 0
ans =
     3
>> round(pi) % arrotondamento verso l'intero piu' ...
     vicino
ans =
     3
>> ceil(pi) % arrotondamento verso + infinito
ans =
     4
>> floor(pi) % arrotondamento verso - infinito
ans =
     3
>> fix(-pi) % arrotondamento verso 0
ans =
    -3
>> round(-pi) % arrotondamento verso l'intero piu'...
      vicino
ans =
    -3
>> ceil(-pi) % arrotondamento verso + infinito
ans =
    -3
>> floor(-pi) % arrotondamento verso - infinito
ans =
    -4
>>
```

Nota. 4.3. *Per tutte le funzioni viste, e tutte le altre che fanno parte dell'ambiente Matlab, la chiamata*

`help <nome funzione>`,

permette di avere un aiuto sul contenuto delle stesse.

Se per esempio avessimo dei dubbi su `fix`*:*

```
>> help fix
 fix    Round towards zero.
    fix(X) rounds the elements of X to the nearest ...
     integers towards zero.

    See also floor, round, ceil.

    Reference page for fix
    Other functions named fix
>>
```

5. Assegnazioni

In precedenza, abbiamo detto che

- una variabile è un contenitore di dati situato in una porzione di memoria (una o più locazioni di memoria) destinata a contenere valori, suscettibili di modifica nel corso dell'esecuzione di un programma,
- una variabile è caratterizzata da un nome (inteso solitamente come una sequenza di caratteri e cifre).

In questa sezione intendiamo vedere come *dare* a uno di questo *contenitori* un valore. Tale processo si chiama *assegnazione*.

In Matlab l'assegnazione avviene come segue

`<nome variabile>=<valore variabile>`

Si consideri l'esempio

```
>> a=3
a =
     3
>> b=pi;
>> a
a =
     3
>>
```

In questo processo, abbiamo

- assegnato alla variabile denotata con "a" il valore 3, e visualizzato il risultato (nel formato corrente);
- assegnato alla variabile denotata con "b" il valore π, e mettendo il "`;`" non abbiamo visualizzato il risultato;
- osservato che il valore di "a" è ancora in memoria.

Quindi per assegnare un valore ad una variabile basta scrivere una stringa di testo (nel nostro caso "a" e "b"),

- senza spazi vuoti,
- che sia di natura alfanumerica,
- che cominci per lettera,
- non sia uguale a qualche stringa predefinita in Matlab, come il nome di una istruzione (ad esempio non si può usare `for` che viene utilizzato come vedremo per i cicli di iterazione),

e quindi dopo il segno di "=" dargli un valore (nel nostro caso rispettivamente gli scalari 3 e π).

Finora abbiamo definito due variabili "a" e "b". Il comando

- `who` lista le variabili definite nella command window,
- `whos` ne descrive anche la struttura.

Vediamo tutto ciò direttamente:

```
>> % Era "a=3" e  "b=pi"
>> who
Your variables are:
a  b
>> whos
  Name      Size            Bytes  Class     Attributes
  a         1x1      8  double
  b         1x1      8  double
>>
```

In particolare `whos` dice che

- abbiamo due variabili a, b,
- che sono vettori 1×1 ovvero scalari,
- occupano 8 bytes,
- sono `double` ossia *numeri in precisione doppia*.

6. I vettori in Matlab

In questa sezione definiamo i vettori riga e colonna, alcune funzioni Matlab dedicate e di seguito il significato delle operazioni e funzioni elementari già applicate per scalari.

6.1. *Definire i vettori in Matlab*

In generale i vettori, che abbiamo detto essere *liste di n numeri*, possono essere definiti *componente per componente*.

Per quanto concerne il vettore *riga*

$$(3.1567, -234.343546, 0.4536).$$

basta scrivere nella command window

```
>> [3.1567,-234.343546,0.4536]
ans =
    3.1567 -234.3435    0.4536
>>
```

Diversamente, nel caso del vettore colonna

$$\begin{pmatrix} 3.1415 \\ -1 \\ 2.7182 \end{pmatrix}.$$

digitiamo

```
>> [3.1415; -1; 2.7182]
ans =
    3.1415
   -1.0000
    2.7182
>>
```

La differenza consiste nel fatto che per un vettore riga abbiamo distinto un numero dal successivo mediante una *virgola* (ma basta anche uno spazio vuoto), mentre nel vettore colonna mediante un *punto e virgola*.

Se vogliamo trasformare un vettore riga in un corrispettivo vettore colonna (o viceversa), basta utilizzare il simbolo di *trasposizione* "'" (ovvero l'accento verticale).

```
>>a=[1;2;3] % La variabile "a" ha quale valore quello ...
     del vettore colonna (1; 2; 3).
a =
     1
     2
     3
>> b=a' % La variabile "b" ha quale valore quello del ...
     vettore colonna (1; 2; 3), ma trasposto e quindi ...
     sara' un vettore riga con le stesse componenti.
b =
     1     2     3
>> c=b' % La variabile "c" ha quale valore quello del ...
     vettore riga b ma trasposto e quindi sara' un ...
     vettore colonna con le stesse componenti.
c =
     1
     2
     3
>>
```

Importante. Per definire il vettore vuoto, ovvero privo di componenti, basta eseguire il comando del tipo `v=[]`. Questo può tornare comodo per *inizializzare* un vettore.

6.2. *Lunghezza e dimensione di un vettore*

Per determinare il numero di componenti di un vettore

$$v = (v_1, \ldots, v_n)$$

si usano

- il comando `length` che riporta quante componenti ha un vettore,
- il comando `size` che determina la dimensione di un vettore e, a differenza di `length`, chiarisce se è di tipo riga o colonna.

```
>> vettore_colonna=[2;5;1] % vettore con 3 componenti e ...
     dimensione 3 x 1.
vettore_colonna =
     2
     5
     1
>> length(vettore_colonna)
ans =
     3
>> size(vettore_colonna)
ans =
     3     1
>> vettore_riga=[2,5,1]  % vettore con 3 componenti e ...
     dimensione 1 x 3
vettore_riga =
     2     5     1
>> length(vettore_riga)
ans =
     3
>> size(vettore_riga)
ans =
     1     3
>> vettore_vuoto=[]
vettore_vuoto =
     []
>> length(vettore_vuoto)
ans =
     0
>> size(vettore_vuoto)
ans =
     0     0
>>
```

In Matlab ci sono altri vettori di facile definizione, quelli che

- hanno tutte componenti nulle, generabili con `zeros`,
- quelli in cui queste sono uguali a 1, generabili con `ones`.

Ad esempio,

```
>> zeros(5,1) % vettore avente 5 righe ognuna di ...
     lunghezza 1 (vettore colonna)
ans =
     0
     0
     0
     0
     0
>> zeros(1,5) % vettore avente 1 riga di lunghezza 5 (...
     vettore riga)
ans =
     0     0     0     0     0
>> ones(1,6)
ans =
     1     1     1     1     1     1
>>
```

6.3. *Vettori equispaziati*

I vettori riga $v = (v_1, v_2, \ldots, v_n)$ con componenti equispaziate ovvero tali che $v_{k+1} - v_k = c$, per $k = 1, \ldots, n-1$, sono particolarmente facile da descrivere.

Supponiamo di voler definire il vettore riga

$$v = (3, 5, 7, 9, 11).$$

Notiamo che $v_{k+1} - v_k = 2$, per $k = 1, \ldots, 4$ e quindi il vettore riga v, di dimensione 1×5, ha componenti equispaziate, in cui la prima vale 3 e l'ultima vale 11.

Definiamo v in due modi alternativi.

Il primo comando è

`u=a:h:b`

e genera il vettore $u = (u_1, \ldots, u_m)$ tale che

$$u_k = a + k \cdot h,$$

con $u_k \leq b$.

Nel nostro caso quindi $a = 3$, $b = 11$ e la spaziatura è $h = 2$, da cui

```
>> v=3:2:11 % vettore con primo valore 3, ultimo valore ...
     11, con spaziatura 2
v =
     3     5     7     9    11
>>
```

Il comando

`u=linspace(a,b,m)`

genera un vettore $u = (u_1, \ldots, u_m)$ con m componenti equispaziate, che comincia da a e finisce con b, ovvero

$$u_k = a + (k-1) \cdot \frac{b-a}{m-1}, \quad k = 1, \ldots, m.$$

Nel nostro caso quindi $a = 3$, $b = 11$, $n = 5$ e ricaviamo

```
>> v=linspace(3,11,5) % vettore con primo valore 3, ...
     ultimo valore 11, con 5 componenti equispaziate.
v =
     3     5     7     9    11
>>
```

Nota. 6.1. *Mentre il comando* `linspace(a,b,n)` *produce un vettore in cui il primo componente è* a *e l'ultimo è* b*, il comando* `a:h:b` *non ha questa proprietà se* $b - a$ *non è multiplo di* h*.*

Così ,

```
>> v=3:4:17 % 17-3=14 e 14 non e' multiplo di 4.
>> v=3:4:17
v =
     3     7    11    15
>>  % b=17 non e' l'ultima componente del vettore.
```

Nota. 6.2. *Il comando* $u = a : h : b$*, non necessita che* `h` *sia un numero naturale.*

```
>> v=2:0.5:4
v =
    2.0000    2.5000    3.0000    3.5000    4.0000
>>
```

Inoltre il comando ha senso anche se $a > b$ *e* $h < 0$*.*

```
>> v=5:-1:1
v =
     5     4     3     2     1
>>
```

6.4. *Accesso alle componenti di un vettore*

Sia $v = (v_1, \ldots, v_n)$. Per accedere alle singole componenti di `v` in Matlab, si utilizzano comandi del tipo `v(i)` dove `i` è un numero intero positivo minore di n o più in generale un vettore di interi con componenti in $1, 2, \ldots, n$.

Vediamo un esempio

```
>> v=[3.1 -1.2 5.7 7.1 2.3]
v =
    3.1000   -1.2000    5.7000    7.1000    2.3000
>> v(2) % seleziona la seconda componente del vettore
ans =
   -1.2000
>> v([3 2 4]) % seleziona nell'ordine la terza, la ...
     seconda e la quarta componente.
ans =
    5.7000   -1.2000    7.1000
>> u=[2;5;1] % vettore colonna
u =
     2
     5
     1
     6
>> u(2:4) % seleziona dalla seconda e alla quarta ...
     componente
ans =
     5
     1
     6
>>
```

A volte dati due vettori riga (o colonna)

$$u = (u_1, \ldots, u_m), \quad v = (v_1, \ldots, v_n)$$

può tornare utile un comando con cui ottenere il vettore ottenuto concatenandoli, ovvero

$$w = (u_1, \ldots, u_m, v_1, \ldots, v_n).$$

Per far questo, a seconda siano entrambi vettori riga o colonna, bastano rispettivamente i comandi `w=[u v]` e `w=[u; v]`.

Facciamo come al solito un esempio.

```
>> u=[1 2];
>> v=[3 4 5];
>> w=[u v]
```

```
w =
     1     2     3     4     5
>> u=u'; % vettore colonna.
>> v=v'; % vettore colonna.
>> w=[u; v]
w =
     1
     2
     3
     4
     5
>>
```

Si osservi che per selezionare l'ultima colonna di un vettore `v` è sufficiente scrivere `v(end)`.

```
>> v=[3 4 5];
>> v(end)
ans =
     5
>>
```

6.5. *Operazioni e funzioni vettoriali*

Supponiamo $u = (u_1, \dots, u_n)$ e $v = (v_1, \dots, v_n)$ siano vettori della stessa dimensione ed s uno scalare.

Le istruzioni,

- `c=s*u`, assegnano alla variabile `c` il prodotto dello scalare s con il vettore u, ovvero $c = (c_1, \dots, c_n)$ con

$$c_1 = s \cdot u_1, c_2 = s \cdot u_2, \dots, c_n = s \cdot u_n;$$

- `c=u'`, assegnano alla variabile `c` la trasposizione del vettore u,

- `c=u+v`, assegnano alla variabile `c` la somma del vettore u col vettore v, ovvero $c = (c_1, \dots, c_n)$ con

$$c_1 = u_1 + v_1,\ c_2 = u_2 + v_2, \dots,\ c_n = u_n + v_n;$$

- `c=u-v`, assegnano alla variabile `c` la sottrazione del vettore u col vettore v, ovvero $c = (c_1, \dots, c_n)$ con

$$c_1 = u_1 - v_1,\ c_2 = u_2 - v_2, \dots,\ c_n = u_n - v_n;$$

- `c=u.*v`, assegnano alla variabile `c` il prodotto *puntuale* del vettore u col vettore v, ovvero $c = (c_1, \dots, c_n)$ con

$$c_1 = u_1 \cdot v_1,\ c_2 = u_2 \cdot v_2, \dots,\ c_n = u_n \cdot v_n;$$

- `c=u./v`, assegnano alla variabile `c` la divisione *puntuale* del vettore u col vettore v, ovvero $c = (c_1, \dots, c_n)$ con

$$c_1 = \frac{u_1}{v_1},\ c_2 = \frac{u_2}{v_2}, \dots,\ c_n = \frac{u_n}{v_n}.$$

- `c=u.^k`, assegnano alla variabile `c` la potenza k-sima *puntuale* del vettore u, ovvero $c = (c_1, \dots, c_n)$ con

$$c_1 = u_1^k,\ c_2 = u_2^k, \dots,\ c_n = u_n^k.$$

Nota. 6.3. *Se u e v sono due vettori colonna la scrittura* `c=u'*v` *calcola l'usuale prodotto scalare u e v. Ricordiamo che se*

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

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`.

```
>> u=[1; 2]
u =
     1
     2
>> v=[3; 4]
v =
     3
     4
>> u'*v
ans =
    11
>> v'*u
ans =
    11
```

Vediamo qualche esempio

```
>> s=10;
>> u=[1; 2]
u =
     1
     2
>> v=[3; 4]
v =
     3
     4
>> s*u % s=10, allora moltiplica ogni componente di "u" ...
     per 10.
ans =
     10
     20
>> u' % trasposto del vettore colonna "u"
ans =
     1     2
>> u+v % somma dei vettori "u" e "v" risulta (u(1)+v(1),...
     u(2)+v(2)) =(1+3,2+4)
ans =
     4
     6
>> u-v % sottrazione dei vettori "u" e "v" risulta (u(1)...
     -v(1),u(2)-v(2)) =(1-3,2-4)
ans =
    -2
    -2
>> u.*v % prodotto puntuale dei vettori "u" e "v" ...
     risulta (u(1)*v(1),u(2)*v(2))=(1*3,2*4)
ans =
     3
     8
>> u./v % divisione puntuale dei vettori "u" e "v" ...
     risulta (u(1)/v(1),u(2)/v(2)) =(1/3,2/4)
ans =
    0.3333
    0.5000
 >> u.^3 % cubo puntuale del vettore "u" e risulta ( (u...
     (1))^3, (u(2))^3 )=(1^3,2^3).
ans =
     1
     8
>>
```

E' importante osservare che se $f : \Omega \subseteq \mathbb{R} \to \mathbb{R}$ è una funzione elementare e $u = (u_1, u_2, \ldots, u_n)$ un vettore, allora $f(u) = (f(u_1), f(u_2), \ldots, f(u_n))$, ovvero il vettore ottenuto mediante la valutazione componente per componente della funzione.

Di conseguenza, la dimensione di `u` e `f(u)` è la stessa.

Vediamo un esempio.

```
>> linspace(0,2*pi,5) % vettore riga.
u =
         0    1.5708    3.1416    4.7124    6.2832
>> v=sin(u) % v=(sin(u(1)),sin(u(2)),...,sin(u(5))).
v =
         0    1.0000    0.0000   -1.0000   -0.0000
>> x=[0 1]' % vettore colonna
x =
     0
     1
>> y=exp(x) % y=(exp(0),exp(1)), y vettore colonna ...
     perche' lo e' x.
y =
    1.0000
    2.7183
>> z=exp(log(x)) % ricordare che x=exp(log(x)), z ...
     vettore colonna perche' lo e' x.
z =
     0
     1
>> w=x.*exp(x) % prodotto di due funzioni, componente ...
     per componente.
w =
         0
    2.7183
>> % si osservi che w(1)=x(1)*exp(x(1))=0*1=0, w(2)=x(2)...
     *exp(x(2))=1* 2.7183= 2.7183.
>> 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
>> 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 % cosa succede se non usiamo bene il prodotto ...
     puntuale.
??? Error using ==> mpower
Matrix must be square.
>>
```

Importante. L'uso delle operazioni puntuali è una importante caratteristica di Matlab, che risulterà utile nel definire nuove funzioni vettoriali.

Importante. Eccetto il caso in cui uno dei due vettori u, v sia uno scalare, si sottolinea che u e v devono avere la stessa dimensione.

```
>> u=[1 2 3];
>> v=[4 5];
>> u+v % u ha dim. 1 x 3, mentre v ha dim 1 x 2
Matrix dimensions must agree.
>> u.*v % u ha dim. 1 x 3, mentre v ha dim 1 x 2
Matrix dimensions must agree.
>> u+1 % il risultato e' (u(1)+1,u(2)+1,u(3)+1)...
     =(1+1,2+1,3+1).
ans =
     2     3     4
>>
```

7. Definizione di funzioni matematiche

In precedenza abbiamo definito operazioni e funzioni elementari di natura vettoriale.

In questa sezione mostriamo come definirne di nuove, associarle ad una variabile e infine valutarle.

Per definire una funzione matematica si usano i comandi `inline` e `@`. Attualmente il primo viene ritenuto obsoleto e viene suggerito l'uso del secondo.

Tipicamente se una tal funzione, diciamo f, deve essere valutata nel vettore x si usa `f(x)` ma in Matlab suggerisce il comando `feval(f,x)`.

Esempio. Definiamo $f(x) = x \cdot \sin(x)$, applicabile a vettori $x = (x_1, \ldots, x_n)$, cosicchè $f(x) = (f(x_1), \ldots, f(x_n))$. La valutiamo nel vettore di 1000 punti equispaziati in $[0, 1]$, in cui il primo è a e l'ultimo è b e ne facciamo il grafico.

Scriviamo nella command window

```
>> f=@(x) x.*sin(x); % definizione di funzione ...
     vettoriale
>> % f=inline('x.*sin(x)'); %% stessa funzione con il ...
     comando 'inline'
>> x=linspace(0,1,1000); % vettore di ascissa.
>> y=feval(f,x); % valutazione di funzione
>> plot(x,y); % grafico di funzione
>>
```

ottenendo il grafico di f in $[0, 1]$.

8. La grafica di Matlab

In questa sezione mostriamo come partendo da dati sia possibile disegnare grafici, utilizzando i comandi vettoriali di Matlab.

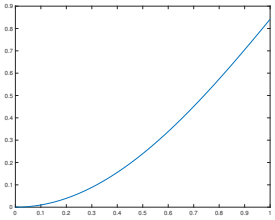

Figura 1: Grafico della funzione $f(x) = x \cdot \sin(x)$, nell'intervallo $[0, 1]$.

A tal proposito consideriamo la funzione

$$f(x) = \exp(x) \cdot \sinh(x) \cdot x^2 \cdot \tan(x) \cdot \log(x + 0.001)$$

nell'intervallo $[0, 1]$, e disegniamo un suo grafico.

```
>> x=linspace(0,1,1000);
>> y=exp(x).*sinh(x).*x.^2.*tan(x).*log(x+0.001);
>> plot(x,y,'r-')
```

- Viene eseguito il plot della funzione f campionandola nei punti $x_k = 0 + \frac{k-1}{999} \in [0, 1]$, con $k = 1, \ldots, 1000$ e pone il risultato in y_k;
- osserviamo che essendo x un vettore, pure $\exp(x)$, $\sinh(x)$, $x.^2$, $\tan(x)$, $\log(x + 0.001)$ sono vettori e quindi viene utilizzato il prodotto puntuale `.*`, e non $*$, per ottenere il risultato finale;
- osserviamo che "r" sta per *rosso*, e con "-" si esegue l'interpolazione lineare a tratti tra i valori assunti dalla funzione (ovvero si uniscono le coppie (x_k, y_k), (x_{k+1}, y_{k+1}), $k = 1, \ldots, 999$, con un segmento); per ulteriori delucidazioni sul comando di plot si digiti nella shell di Matlab `help plot`.

In altri termini, se $x = (x_1, \ldots, x_n)$, $y = (y_1, \ldots, y_n)$, il comando `plot(x,y)` disegna il grafico ottenuto unendo tutte le coppie (x_k, y_k), (x_{k+1}, y_{k+1}), $k = 1, \ldots, n-1$, mediante un segmento (tecnicamente questo processo si chiama `interpolazione lineare a tratti`).

Vediamo un ulteriore esempio.

Dallo sviluppo di Mac Laurin, ovvero di Taylor centrato in $x_0 = 0$, sappiamo che per $x \approx 0$ abbiamo

$$\exp(x) \approx p(x) := 1 + x + \frac{x^2}{2} + \frac{x^3}{6}.$$

Vogliamo vedere la qualità dell'approssimazione nell'intervallo $[0, 5]$. Viene naturale fare il grafico delle due funzioni f, p e poi dell'errore assoluto compiuto $|f(x) - p(x)|$.

Digitiamo dapprima

```
>> x=linspace(0,5,1000);
>> y=exp(x);
>> z=1+x+(x.^2)/2+(x.^3)/6;
>> plot(x,y,x,z);
```

Abbiamo

- definito un vettore x avente 1000 componenti equispaziate, in cui la $x_k = 0 + \frac{5(k-1)}{999}$, per $k = 1, \ldots, 1000$.
- definito un vettore y avente 1000 componenti in cui $y_k = f(x_k)$;
- definito un vettore z avente 1000 componenti in cui $z_k = p(x_k)$ (facendo attenzione alle operazioni puntuali tra vettori);
- illustrato il grafico (approssimato), mediante le coppie (x_k, y_k), (x_k, z_k), $k = 1, \ldots, 1000$, delle funzioni f e p in $[0, 5]$.

Si ottiene il grafico in figura.

Nota. 8.1. *E' interessante osservare che con due comandi* `plot` *successivi come in*

```
>> plot(x,y);
>> plot(x,z);
```

Matlab avrebbe disegnato il primo grafico, lo avrebbe cancellato e disegnato il secondo grafico. Per ovviare a questo problema si può usare il comando di `plot` *utilizzando*

`plot(x,y,x,z)`,

come nell'esempio.

Alternativamente si possono fare due grafici separati, ma occorre introdurre

- `hold on` *che permette di sovrapporre più grafici nella stessa figura,*
- `hold off` *che non permette di seguito di sovrapporre ulteriori grafici nella stessa figura.*

Il precedente codice diventa:

```
>> x=linspace(0,5,1000);
>> y=exp(x);
>> z=1+x+(x.^2)/2+(x.^3)/6;
>> plot(x,y,'b-');
>> hold on; % mantieni il grafico nella finestra.
>> plot(x,z,'r-'); % mette questo grafico nella finestra...
     precedente.
>> hold off; % non mantenere altri grafici.
```

8.1. La scala semilogaritmica

Nel descrivere graficamente gli errori, si ricorre spesso alla *scala logaritmica*, mediante il comando `semilogy`.

Se $x = (x_1, \ldots, x_n)$, $y = (y_1, \ldots, y_n)$, il comando

`semilogy(x,y)`

descrive il grafico ottenuto unendo tutte le coppie $(x_k, \log_{10}(y_k))$, $(x_{k+1}, \log_{10}(y_{k+1}))$, $k = 1, \ldots, n-1$, mediante un segmento.

Di seguito digitiamo

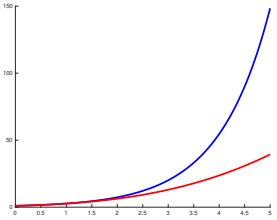

Figura 2: Grafico della funzione $f(x) = \exp(x)$ (in blu), $p(x) = 1 + x + \frac{x^2}{2} + \frac{x^3}{6}$ (in rosso), nell'intervallo $[0, 5]$.

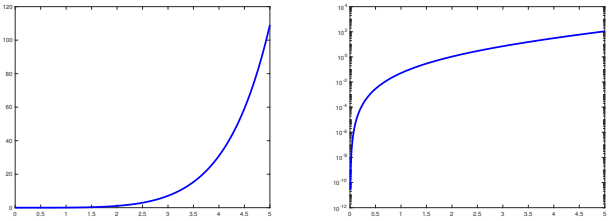

Figura 3: Differenza tra `plot` e `semilogy` nel rappresentare le coppie $(x_k, |f(x_k) - p(x_k)|)$ per $t = 1, \ldots, 1000$.

```
>> err=abs(y-z);
>> plot(x,err,'b-','Linewidth',3);
>> pause; % si guardi il grafico e si prema un qualsiasi...
     tasto.
>> semilogy(x,err,'b-','Linewidth',3);
```

Con questo codice abbiamo

- definito un vettore `err` avente 1000 componenti, in cui la k-sima vale $|f(x_k) - p(x_k)|$, per $k = 1, \ldots, 1000$;
- illustrato il grafico dell'errore in scala usuale,
- illustrato il grafico dell'errore in scala semilogaritmica.

Dai grafici (si veda anche la relativa figura), si comprende che

- per $x \geq 1$, l'approssimazione della formula di Mac Laurin in esame risulta scadente, mentre risulta di buona qualità in un intorno di $x = 0$
- dalla figura col comando `plot`, non siamo in grado di descrivere la qualità dell'approssimazione in un intorno di 0; tale comando esegue il grafico delle coppie (x_k, err_k) mediante interpolazione lineare a tratti,
- dalla figura col comando `semilogy`, siamo in grado di descrivere la qualità dell'approssimazione in un intorno di 0; tale comando esegue il grafico delle coppie $(x_k, \log_{10}(err_k))$ mediante interpolazione lineare a tratti.

Deduciamo che la scala semilogaritmica permette una miglior descrizione degli errori forniti.

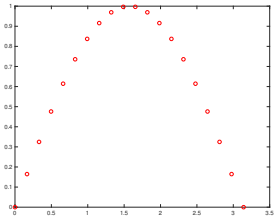

Figura 4: Grafico delle coordinate $(x_k, sin(x_k))$, per $x_k = 0 + (k-1)\pi/19$, $k = 1, \ldots, 20$.

Nota. 8.2. *Molte volte è utile disegnare esclusivamente le coppie (x_k, u_k), con $k = 1, \ldots, n$. A tal proposito si usa il comando* `plot(x,y,'ro')` *(o se si intende fare il grafico in scala semilogaritmica* `semilogy(x,y,'ro')`*). In questo caso, vengono tracciati cerchietti rossi a rappresentare ogni coppia.*

```
>> x=linspace(0,pi,20);
>> y=sin(x);
>> plot(x,y,'ro');
>>
```

8.1.1. Altri comandi per grafici

Matlab ha altre modalità di plot, quali

- `semilogx`, che esegue il grafico, mediante interpolazione lineare a tratti, delle coppie $(\log_{10}(x_k), y_k)$,
- `loglog`, che esegue il grafico, mediante interpolazione lineare a tratti, delle coppie $(\log_{10} \log_{10}(x_k), \log_{10}(y_k))$, Entrambe sono tipicamente di uso meno comune rispetto a `semilogy`.

Un comando utile è quello di `clf`, ossia *clear figure*, con cui viene cancellata qualsiasi cosa sia stata disegnata nella finestra.

A volte tra un grafico e l'altro, può tornare comodo il comando di `pause`, con cui si blocca il codice finchè un qualsiasi tasto non venga digitato.

In alternativa, il comando `pause(3)` interrompe l'esecuzione per 3 secondi e poi la riprende. Ovviamente si può utilizzare il numero di secondi desiderato, diversamente da 3.

Esercizio 1. Dall'help di Matlab:

```
>> help eps
 eps  Spacing of floating point numbers.
    D = eps(X), is the positive distance from ABS(X) to ...
     the next larger in
    magnitude floating point number of the same ...
     precision as X.
    X may be either double precision or single precision...
     .
    For all X, eps(X) is equal to eps(ABS(X)).

    eps, with no arguments, is the distance from 1.0 to ...
     the next larger double
    precision number, that is eps with no arguments ...
     returns 2^(-52).
    ...
```

```
    See also realmax, realmin.

    Reference page for eps
    Other functions named eps
>>
```

Si

- definisca il vettore u avente componenti $-15, -14, \ldots, -1$;
- utilizzando u, si definisca il vettore x avente componenti $10^{-15}, 10^{-14}, \ldots, 10^{-1}$;
- valuti il vettore y la cui k-sima componente vale $\text{eps}(x_k)$;
- effettui il grafico in scala semilogaritmica delle coppie (u_k, y_k), unite con un segmento in colore magenta (ci si aiuti con l'help del comando `plot`, ma si effettui il grafico con *semilogy*).
- sovrapponga al grafico precedente, il grafico in scala semilogaritmica delle coppie (u_k, y_k) con cerchietti in blu.

Esercizio 2. Si considerino le funzioni

1. $f_1(x) = 1 - x - \exp(-2x)$ per $x \in [-1, 1]$;
2. $f_2(x) = 2x \cdot \exp(x) - 1$ per $x \in [0, 1]$;
3. $f_3(x) = x^2 - 2x - \exp(-x + 1)$ per $x \in [-2, 2]$;
4. $f_4(x) = \sqrt{x + 2} + x \cdot \sin(x)$ per $x \in [0, 1]$.

Si

- definisca il vettore x avente 50 componenti equispaziate nell'intervallo di riferimento;
- si valuti il vettore y la cui k-sima componente vale

$$(x_k, f_i(x_k)), \ i = 1, \ldots, 5, \ \ k = 1, \ldots, 50;$$

- si effettui il grafico delle coppie (u_k, y_k), unite con un segmento in colore rosso.
- si sovrapponga al grafico precedente il grafico delle coppie (u_k, y_k) con cerchietti in blue.

8.2. *I comandi* `legend` *e* `title`

Per arricchire le figure si può

- dare un titolo alla figura mediante il comando `title`;
- creare una legenda che descriva quanto disegnato, mediante il comando `legend`;

Vediamo un esempio:

```
>> x=0:0.001:2*pi;
>> y=sin(x);
>> z=cos(x);
>> plot(x,y,'k-'); hold on; plot(x,z,'r-');
>> title('Funzioni trigonometriche');
>> legend('seno','coseno');
>> hold off;
```

Il risultato è la seguente figura, in cui abbiamo spostato leggermente la legenda mediante il mouse.

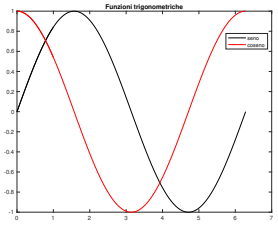

Figura 5: Grafico di $\sin(x)$, $\cos(x)$ in $[0, 2\pi]$, con titolo e legenda.

9. Le stringhe di testo

In questa sezione introduciamo il tipo di dati stringa, consistente di una sequenza di caratteri alfanumerici.

Ad esempio:

```
>> s='pippo_pluto_e_paperino'
s =
    'pippo_pluto_e_paperino'
>>
```

Nota. 9.1. *Un classico problema è quello di rappresentare stringhe contenenti proprio "' ". Lo si supera digitando tali apostrofi due volte.*

```
>> disp('L''accento viene rappresentato correttamente')
L'accento viene rappresentato correttamente
>>
```

Se la stringa è immagazzinata in una variabile, è comunque possibile rappresentare il contenuto della variabile mediante `disp`.

```
>> s='Frankestein Junior';
>> disp(s)
Frankestein Junior
>>
```

Un comando simile a `disp` è `fprintf`.

```
>> fprintf('Frankestein Junior')
Frankestein Junior>>
```

Notiamo che non ha mandato a capo dopo *Junior*. Per farlo, basta aggiungere il descrittore di formato `\n`.

```
>> fprintf('Frankestein Junior \n')
Frankestein Junior
>>
```

Se lo avessimo usato pure all'inizio, avremmo ottenuto

```
>> fprintf('\n Frankestein Junior \n')

 Frankestein Junior
>>
```

saltando così pure una riga.

Il descrittore `\t` permette invece di fare un *tab*, ovvero spostare la stringa verso destra.

```
>> fprintf('\t Frankestein Junior \n')
           Frankestein Junior
>>
```

10. I comandi `format`, `disp`, `fprintf`

In questa sezione introduciamo i formati numerici e quindi i comandi `disp` ed `fprintf` relativamente a numeri macchina.

I formati più comuni sono

- `format short`: notazione decimale con 4 cifre dopo la virgola;
- `format short e`: notazione esponenziale con 4 cifre dopo la virgola;
- `format short g`: la migliore delle precedenti;
- `format long`: notazione decimale con 15 cifre dopo la virgola in doppia precisione, e 7 cifre dopo la virgola in singola precisione.
- `format long e`: notazione esponenziale con 15 cifre dopo la virgola in doppia precisione, e 7 cifre dopo la virgola in singola precisione.
- `format long g`: la migliore delle precedenti.

Vediamo un esempio:

```
>> format short; pi
ans =
   3.1416
>> format short e; pi
ans =
   3.1416e+00
>> format short g; pi
ans =
   3.1416
>> format long; pi
ans =
   3.141592653589793
>> format long e; pi
ans =
   3.141592653589793e+00
>> format long g; pi
ans =
   3.14159265358979
>>
```

Per il display di variabili o di altre quantitá valutabili, si utilizzano i comandi `disp`, `fprintf`.
Per capire come si usino, vediamo i seguenti esempi.

```
>> s=pi/10
s =
    0.3142
>> disp(s)
    0.3142
>> % "s" in formato decimale, 1 cifra prima della ...
     virgola, 6 dopo la virgola.
>> % "\n" manda a capo.
>> fprintf('%1.6f \n',s)
0.314159
>> % s in formato esponenziale, 1 cifra prima e 6 dopo ...
     la virgola.
>> fprintf('%1.6e \n',s)
3.141593e-01
>> % "\t" esegue un "tab" ovvero sposta leggermente a ...
     destra quanto scritto.
>> fprintf('\t %1.6e \n',s)
   3.141593e-01
>> fprintf('\t La variabile s vale: %1.15e \n',s) % E' ...
     possibile aggiungere testo descrittivo.
   La variabile s vale: 3.141592653589793e-01
>> a=pi; b=exp(1);
>> fprintf('\t s: %1.15e t: %1.15e \n',a,b) % Piu' ...
     variabili.
   s: 3.141592653589793e+00 t: 2.718281828459046e+00
>>
```

Nell'esempio, abbiamo

- utilizzato `disp` per visualizzare il valore della variabile `s`;
- utilizzato `fprintf` per visualizzare il valore della variabile `s`, con una cifra prima della virgola e 6 dopo la virgola, in notazione decimale ed esponenziale, per poi andare a capo con `\n`;
- utilizzato `fprintf` per visualizzare il valore della variabile `s`, immettendo con `\t` alcuni caratteri vuoti, e a seguire descrivere la quantità con una cifra prima della virgola e 6 dopo la virgola, per poi andare a capo con `\n`;
- nella penultima abbiamo usato un test che descrivesse qualcosa della variabile;
- nell'ultima abbiamo usato più di una variabile.

Nota. 10.1. *Per descrivere il significato delle ascisse e delle ordinate, si usano i comandi* `xlabel`, `ylabel`, *che aggiungono un testo scelto dal programmatore vicino ai corrispettivi assi.*

11. Operazioni con le matrici

Esistono vari modi per definire una matrice A.

Se ad esempio

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

il più comune è via l'assegnazione diretta

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

In altri termini si scrivono più vettori riga, e il ";" indica che si passa a descrivere la riga successiva.

Con il comando `A(i,j)` è possibile selezionare la componente (i, j) della matrice A.

Inoltre col il comando `A(:,j)` si seleziona la j-sima colonna di A, mentre con `A(i,:)` si fa altrettanto con la i-sima riga.

Ad esempio:

```
>> A=[1 2 3; 4 5 6; 7 8 9]
A =
     1     2     3
     4     5     6
     7     8     9
>> A(2,3) % terzo elemento della seconda riga.
ans =
     6
>> A(:,3)  % terza colonna.
ans =
     3
     6
     9
>> A(2,:) % seconda riga.
ans =
     4     5     6
>>
```

Supponiamo

$$A = (a_{i,j})_{i\in[1,m],j\in[1,n]},\ \ B = (b_{i,j})_{i\in[1,m],j\in[1,n]}$$

siano matrici della stessa dimensione $m \times n$ ed s uno scalare.

L'istruzione

- `c=s*A` assegna a `c` il prodotto dello scalare s con la matrice A, ovvero $c = (c_{i,j})_{i\in[1,m],j\in[1,n]}$ con

$$c_{i,j} = s \cdot a_{i,j},\ \ i = 1,\dots,m, j = 1,\dots,n;$$

- `c=A'` assegna a `c` la trasposizione della matrice A, , ovvero $c = (c_{i,j})_{i\in[1,n],j\in[1,m]}$ con

$$c_{i,j} = a_{j,i},\ \ i = 1,\dots,n, j = 1,\dots,m;$$

- `c=A+B` assegna a `c` la somma della matrice A col la matrice B, ovvero $c = (c_{i,j})_{i\in[1,m],j\in[1,n]}$ con

$$c_{i,j} = a_{i,j} + b_{i,j},\ \ i = 1,\dots,m, j = 1,\dots,n;$$

- `c=A-B` assegna a `c` la sottrazione della matrice A col la matrice B, ovvero $c = (c_{i,j})_{i\in[1,m],j\in[1,n]}$ con

$$c_{i,j} = a_{i,j} - b_{i,j},\ \ i = 1,\dots,m, j = 1,\dots,n;$$

- `c=A.*B` assegna a `c` il prodotto *puntuale* della matrice A col la matrice B, ovvero $c = (c_{i,j})_{i\in[1,m],j\in[1,n]}$ con

$$c_{i,j} = a_{i,j} \cdot b_{i,j},\ \ i = 1,\dots,m, j = 1,\dots,n;$$

- `c=A./B` assegna a `c` la divisione *puntuale* della matrice A col la matrice B, ovvero $c = (c_{i,j})_{i\in[1,m],j\in[1,n]}$ con

$$c_{i,j} = \frac{a_{i,j}}{b_{i,j}},\ \ i = 1,\dots,m, j = 1,\dots,n;$$

- `c=A.^k` assegna a `c` la potenza k-sima *puntuale* della matrice A, ovvero $c = (c_{i,j})_{i\in[1,m],j\in[1,n]}$ con

$$c_{i,j} = a_{i,j}^k,\ \ i = 1,\dots,m, j = 1,\dots,n;$$

Vediamo alcuni esempi.

```
>> A=[1 2; 3 4]
A =
     1     2
     3     4
>> B=[7 8; 9 10]
B =
     7     8
     9    10
>> A+B
ans =
     8    10
    12    14
>> A-B
ans =
    -6    -6
    -6    -6
>> A.*B
ans =
     7    16
    27    40
>> A./B
ans =
    0.1429    0.2500
    0.3333    0.4000
>> A.^2
ans =
     1     4
     9    16
>>
```

Nota. 11.1. *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

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

Con riferimento all'esempio precedente:

```
>> A=[1 2; 3 4]
A =
     1     2
     3     4
>> B=[7 8; 9 10]
B =
     7     8
     9    10
>> A*B
ans =
    25    28
    57    64
>> A.*B
ans =
     7    16
    27    40
>>
```

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

Così

```
>> A =[ 1,2; 3,4];
>> size(A)
ans =
     2     2
>> eye(2) % matrice identica di dimensione 2.
ans =
     1     0
     0     1
>> zeros(2) % matrice zero di dimensione 2.
ans =
     0     0
     0     0
>> diag(A) % vettore contenente a(1,1), a(2,2).
ans =
     1
     4
>>
```

11.1. Sulle matrici e vettori

Osserviamo che se A è una matrice $m \times n$, u un vettore colonna $n \times 1$ allora $A * u$ è l'usuale prodotto matrice-vettore, e il risultato è un vettore $m \times 1$.

Vediamone un esempio:

```
>> A=[1 2; 3 4] % matrice 2 x 2
A =
     1     2
     3     4
>> u=[5 6] % vettore 1 x 2 (non va bene per prodotto ...
     matrice vettore con A)
u =
     5     6
>> A*u
??? Error using ==> *
Inner matrix dimensions must agree.

>> u=u' % vettore 2 x 1 (va bene per prodotto matrice ...
     vettore con A)
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 % non e' la barra giusta!
??? Error using ==> mrdivide
Matrix dimensions must agree.
>> x=A\b % e' la barra giusta!
x =
    5.0000
    6.0000
>>
```

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

Un altro comodo comando Matlab 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
>>
```

12. Definizione di una funzione

In Matlab l'utente può definire una funzione scrivendo un m-file, cioè un file con l'estensione *.m*.

Per scrivere una funzione si può utilizzare l'editor di Matlab o un editor alternativo.

Nel primo caso,nella versione di Matlab 2018, basta

- selezionare sulla barra di Matlab, la icona col +, ovvero la terza da sinistra,
- un doppio click su `Script` (invece di `function` che non è immediatamente fruibile da un principiante).

Mostriamo di seguito un esempio di funzione, che possiamo scrivere mediante l'editor:

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

Alla fine salviamo il file nella cartella corrente come `fun.m`, mediante un singolo click su Save (terzo elemento da sinistra della barra di Matlab) e poi su *save* dal susseguente menu a tendina che viene proposto. I più espert osservinoche questo può essere facilmente effettuato tramite una tipica combinazione di tasti.

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 e in tal caso la struttura ha la forma

  ```
  function [y1,...,ym] =fun(x1,...,xn)
  ```

  dove al posto di `fun` si può scrivere un generico nome di funzione, come ad esempio `fun2`.

  Per capirlo meglio si consideri il caso

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

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

13. Operatori di relazione e logici

In questa sezione prima mostriamo i principali operatori di relazione e logici in Matlab e poi passiamo a vedere come scrivere istruzioni condizionali.

I principali operatori di relazione sono

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

Vediamo alcuni esempi di test che coinvolgono alcuni operatori di relazione, tenendo conto che alla risposta, Matlab con 1 intende *vero* mentre 0 intende *falso*.

```
>> 1 == 1 % ci domandiamo se 1 e' uguale a 1. La ...
     risposta 1 e' per il si, 0 per il no.
ans =
     1
>> 0 == 1
ans =
     0
>> 1 ≥ 0
ans =
     1
>>
```

I principali operatori logici sono (cf. [12])

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

```
>> (3 == 3) & (2+2 ≥ 4) % (SI & SI)=SI
ans =
  logical
   1
>> ( 3 == 3 ) & ( pi == 3 ) % (SI & NO)=NO
ans =
  logical
   0
>> (3 == 4) | (2+2 ≥ 4) % (NO o SI)=SI
ans =
  logical
   1
>> (3 == 4) | ¬(2+2 == 4) % (NO o non SI)=(NO o NO)= NO
ans =
  logical
   0
>>  ((3 == 4) | ¬(2+2 == 4)) | ( 2 == 2) % (NO o non SI)...
     o SI= NO o SI = NO.
ans =
  logical
   1
>>  % L'ultimo mostra che si possono scrivere test ...
    logici "compositi".
```

Problema. Spiegare perchè

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

14. Le istruzioni condizionali

L'*istruzione condizionale semplice* esegue sequenzialmente alcune operazioni, se certi test vengono soddisfatti, secondo

```
if (espressione logica)
 < processo 1 >
else
 < processo 2 >
end
```

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

```
if (espressione logica)
 < processo 1 >
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 calcola il segno di a, nel nostro caso $a = 50$.

La *struttura condizionale multipla*, sfrutta il fatto che nella struttura condizionale alternativa, si possano utilizzare nuovamente istruzioni condizionali (semplici o multiple), come ad esempio

```
if < espressione logica 1 e' verificata >
    < processo 1 >
else
    if < espressione logica 2 e' verificata >
        < processo 2 >
    else
        < processo 3 >
    end
end
```

A volte torna comodo il comando Matlab `switch` che a seconda del valore di una variabile esegue una porzione di programma.

```
switch (espressione switch )
case < valore 1 >
  < processo 1 >
case < valore 2 >
  < processo 2 >
...
otherwise
  < processo otherwise >
end
```

La parte `otherwise`, può non essere citata, e quindi se non si rientra nei processi dovuti a qualche `case`, il codice non effettua alcun processo.

Vediamone un esempio, ricordando che la funzione `sign(x)` vale

- 1 se $x > 0$,
- -1 se $x < 0$,
- 0 altrimenti (cioè se $x = 0$).

```
>> a=1; s=sign(a);
>> % s = 1 se a > 0, s = -1 se a < 0, s = 0 se a = 0.
>> switch s
case 1
   stringa='a > 0';
case -1
   stringa='a < 0';
otherwise
   stringa='a = 0';
end
>> stringa
stringa =
a > 0
>>
```

Esercizio facile. 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.

15. 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)
  < processo >
end
```

Vediamone un esempio.

```
>> s=0;
for j=1:10
    % assegna alla variabile "s" il valore corrente cui ...
     si somma "j".
    s=s+j;
end
>>
```

Passo passo, la variabile j assume

- il valore 1 ed $s = s + j = 0 + 1 = 1$;
- 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.

16. Ciclo while

Simile al *ciclo for* è il *ciclo while* che itera il processo ogni volta che una certa condizione è verificata, terminandolo la prima volta in cui tale condizione è falsa.

In Matlab

```
while (espressione logica)
  < processo >
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 $45 = 9 \cdot 10/2$.

16.1. *Legame tra ciclo while e ciclo for*

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 100 iterazioni.

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.

17. Gestione dei files dei dati

In molti esperimenti scientifici i dati vengono passati mediante files o registrati sugli stessi. In questa sezione discutiamo come effettuare tutto ciò .

17.1. *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 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 [6].

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 *vede* questo file come una matrice le cui componenti sono quelle della *variabile* `PDXprecip`. Il comando `load` carica questa variabile nel workspace di Matlab. Di conseguenza:

```
>> load PDXprecip.dat;
>> mese=PDXprecip(:,1)
mese =
     1
     2
     3
     4
     5
     6
     7
     8
     9
    10
    11
    12
```

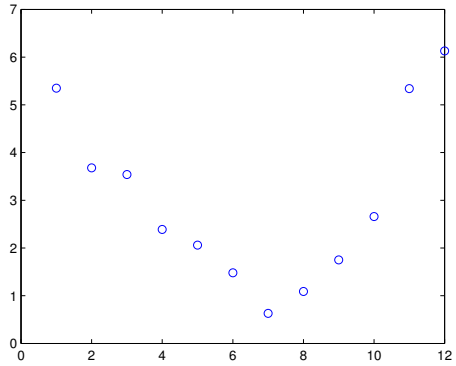

Figura 6: Grafico di alcuni dati immagazzinati su file.

```
>> 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).

17.2. Salvare dati su file

Di seguito, mostriamo come salvare dei dati su file. Questo puó essere importante in varie situazioni.

I comandi rilevanti sono

- `fopen`;
- `fprintf`;
- `fclose`.

Vediamone i dettagli, aiutandoci con l'help.

```
>> help fopen
 fopen  Open file.
    FID = fopen(FILENAME) opens the file FILENAME for ...
     read access. FILENAME is the name of the file to be...
      opened.

    FID is a scalar MATLAB integer valued double, called...
      a file identifier. You use
    FID as the first argument to other file input/output...
      routines, such as FREAD and
    FCLOSE. If fopen cannot open the file, it returns ...
     -1.
    ....
    See also fclose, ferror, fgetl, fgets, fprintf, ...
     fread, fscanf, fseek,
            ftell, fwrite.
    ....
>>
```

```
>> help fclose
 fclose Close file.
    ST = fclose(FID) closes the file associated with ...
     file identifier FID,
    which is an integer value obtained from an earlier ...
     call to FOPEN.
    fclose returns 0 if successful or -1 if not.  If FID...
      does not represent
    an open file, or if it is equal to 0 (standard input...
     ), 1 (standard
    output), or 2 (standard error), fclose throws an ...
     error.

    ST = fclose('all') closes all open files, except 0, ...
     1 and 2.

    See also fopen, ferror, fprintf, fread, frewind, ...
     fscanf, ftell, fwrite.

    Reference page for fclose
    Other functions named fclose
>>
```

17.2.1. Esercizio I

Scriviamo nella command-window

```
>> x=1:0.1:2;
>> fileID=fopen('file.dat','w');
>> fprintf(fileID,'%1.10g \n',x);
>> fclose(fileID);
>>
```

Nel precedente codice

- si definisce un vettore di punti equispaziati v in cui la k-sima componente è $v_k = 1 + (k-1) \cdot 0.1$, con $k = 1, \ldots, 11$;
- si crea un file `file.dat` su cui, in virtù di `'w'` si può scrivere; ci si riferisce a tale file quando si scrive la variabile `fileID`;
- si scrivono su `fileID`, ovvero su `file.dat`, i contenuti del vettore `file.dat`, con una cifra prima e 10 dopo la virgola, per poi andare a capo dopo ogni numero;
- si chiude il file.

Il risultato è che viene creato il file `file.dat` che ha per testo

```
1
1.1
1.2
1.3
1.4
1.5
1.6
1.7
1.8
1.9
2
```

17.2.2. Esercizio II

Vediamo cosa fare quando i dati da registrare su file sono più complessi, come le coppie (x_k, y_k), $x_k = 1 + (k-1)/10$, $y_k = \exp(x_k)$, $k = 1, \ldots, 11$.

```
function esempio2

x=1:0.1:2;
A=[x; exp(x)]; % matrice 2 x 11.
fileID=fopen('test.txt','w');
% scrivi intestazione.
fprintf(fileID,'%6s %12s\n','x','exp(x)');
% scrivi dati su files, andando a capo di volta in volta...
    .
fprintf(fileID,'%6.2f %12.8f\n',A);
% chiudi file.
fclose(fileID);
% vedi file
edit test.txt
```

Nel precedente codice

- si definisce un vettore di punti equispaziati

$$v = (1, 1.1, \ldots, 2);$$

- si definisce la matrice A di dimensione 2×11, la cui prima riga è il vettore riga `x` e la seconda il vettore riga `exp(x)`;
- si crea un file `file.txt` su cui, in virtù di `'w'` si può scrivere; ci si riferisce a tale file quando si scrive la variabile `fileID`;
- si scrivono su `fileID`, ovvero su `file.txt`, una intestazione con scritte le stringhe `x` e `exp(x)` separate da un certo numero di spazi;
- si scrivono su `fileID`, ovvero su `file.txt`, i contenuti della matrice A, ovvero della sua k-sima colonna, al variare di $k = 1, \ldots, 11$, rispettivamente con 6 cifre prima e 2 dopo la virgola (in formato decimale), 12 cifre prima e 8 dopo la virgola (in formato decimale) per poi andare a capo dopo ogni numero;
- si chiude il file;
- si fa il display del testo.

Si ottiene

```
     x       exp(x)
  1.00   2.71828183
  1.10   3.00416602
  1.20   3.32011692
  1.30   3.66929667
  1.40   4.05519997
  1.50   4.48168907
  1.60   4.95303242
  1.70   5.47394739
  1.80   6.04964746
  1.90   6.68589444
  2.00   7.38905610
```

18. Altri comandi

Esistono vari comandi Matlab di uso comune. Ne citiamo per semplicità alcuni.

1. Per conoscere ulteriori toolboxes predefinite in Matlab, basta digitare `help`.

```
>> help
HELP topics:
Documents/MATLAB    - (No table of contents file)
...
matlab/elfun - Elementary math functions.
matlab/elmat - Elementary matrices and matrix ...
     manipulation.
matlab/funfun - Function functions and ODE solvers...
     .
matlab/general - General purpose commands.
...
matlab/matfun - Matrix functions    - numerical ...
     linear algebra
...
matlab/polyfun - Interpolation and polynomials.
...
```

2. il comando `cputime` permette come segue di sapere il tempo impiegato da un processo. Si consideri a tal proposito la funzione `test_cputime.m`

```
function test_cputime

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
```

Alternativamente potevamo eseguire (da scriversi come `test_tictoc.m`)

```
function test_tictoc

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

3. il comando `find` che determina le occorrenze di uno o più elementi in un vettore

```
>> a
a =
     5     3     6
>> find(a == 6)
ans =
     3
>> % Il valore di "a" che vale "6" e' il terzo.
```

4. il comando `rand` determina numeri casuali in $[0, 1]$,

```
>> rand(1,3) % vettore 1 x 3 di numeri casuali,
ans =
    0.8235    0.6948    0.3171
>> rand(2,2) % matrice 2 x 2 di numeri casuali.
ans =
    0.9502    0.4387
    0.0344    0.3816
>>
```

5. il comando `sort` che ordina un vettore

```
>> a=rand(1,5) % un vettore riga con 5 elementi
a =
    0.7060    0.0318    0.2769    0.0462    0.0971
>> b=sort(a) % il vettore "a" e' ordinato (...
     crescente)
b =
    0.0318    0.0462    0.0971    0.2769    0.7060
>>
```

6. il comando `input` che permette all'utente di inserire alcune variabili richieste, durante l'esecuzione del codice,

```
>> format long e;
>> s=input('inserisci un numero: ');
inserisci un numero: 3.1415
>> % dopo la richiesta di input, il codice scrive ...
     "inserisci un numero: " e aspetta l'...
     immissione di dati.
>> s
s =
     3.141500000000000e+00
>>
```

7. il comando `clear` che cancella le variabili e funzioni (quelle su command window, non quelle scritte su file!) dalla memoria.

18.1. Il diary

Uno dei comandi più interessanti di Matlab è il `diary` che scrive su file quanto visualizzato nel workspace di Matlab.

Vediamone un esempio dal workspace di Matlab:

```
>> diary on
>> s=2;
>> t=5;
>> u=s+t
u =
     7
>> diary off
```

Nella directory attuale (vista cioè da Matlab) troviamo un file di testo *diary* (senza estensione). Lo apriamo con un editor.

Il file contiene quanto apparso sulla shell di Matlab ad eccezione del prompt $>>$. Osserviamo che può essere utile per vedere a casa quanto fatto a lezione sul workspace di Matlab.

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

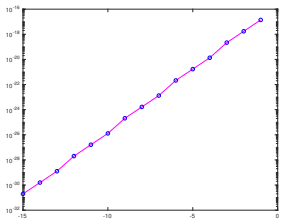

Figura 7: Grafico della funzione $eps(x)$ per $x \in [10^{-15}, 10^{-1}]$, mostrando i valori assunti in $10^{-15}, 10^{-14}, \ldots, 10^{-1}$.

```
>> 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;
```

19. Correzione degli esercizi

19.1. Esercizio 1

Scriviamo la function

```
function esercizio1

u=-15:1:-1;
x=10.^u;
y=eps(x);
semilogy(u,y)
```

Dopo aver digitato nella command window `esercizio1`, otteniamo il grafico in figura.

19.2. Esercizio 2

Scriviamo la function

```
function esercizio2

sottocaso=1;
switch sottocaso
    case 1
        f=@(x) 1-x-exp(-2*x); a=-1; b=1;
    case 2
        f=@(x) 2*x.*exp(x)-1; a=0; b=1;
    case 3
        f=@(x) x.^2-2*x-exp(-x+1); a=-2; b=2;
    otherwise
        f=@(x) sqrt(x+2)+x.*sin(x); a=0; b=1;
end

x=linspace(a,b,50);
y=feval(f,x);

plot(x,y,'r-');
hold on;
```

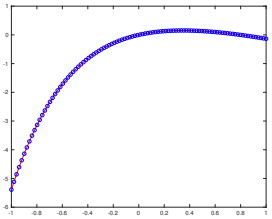

Figura 8: Grafico della funzione $f(x) = 1 - x - \exp(-2x)$ per $x \in [-1, 1]$ (in rosso), con cerchietti blu in $(x_k, f(x_k))$, $x_k = -1 + 2(k-1)/49$, $k = 1, \ldots, 50$.

```
plot(x,y,'bo');
hold off;
```

Dopo aver digitato nella command window `esercizio2`, otteniamo il grafico in figura.

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

Bibliografia

[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., http://www.mathworks.com/.

[5] The MathWorks Inc., *Numerical Computing with Matlab*, http://www.mathworks.com/moler.

[6] G. Recktenwald, *Loading Data into MATLAB for Plotting*, http://web.cecs.pdx.edu/gerry/MATLAB/plotting/loadingPlotData.html.

[7] Università degli Studi di Padova, Servizi per Utenti Istituzionali Contratti Software e Licenze MATLAB, https://www.ict.unipd.it/servizi/servizi-utenti-istituzionali/contratti-software-e-licenze/matlab

[8] Octave, // http://octave.sourceforge.net/

[9] Wikipedia, Algoritmo, http://it.wikipedia.org/wiki/Algoritmo.

[10] Wikipedia, Matrice, https://it.wikipedia.org/wiki/Matrice

[11] Wikipedia, Stringa_(informatica), https://it.wikipedia.org/wiki/Stringa_(informatica)

[12] Wikipedia, Tabella della verità, https://it.wikipedia.org/wiki/Tabella_della_verità

[13] Wikipedia, Variabile (informatica), https://it.wikipedia.org/wiki/Variabile_(informatica)

[14] Wikipedia, Vettore (matematica), https://it.wikipedia.org/wiki/Vettore_(matematica)