Alcune differenze importanti tra octave/matlab e il linguaggio C:

	octave	C
caratteri	‘z’	‘z’
stringhe	‘pippo’	“pippo”
negazione	c ~= 5	c != 5
fine riga	; o nulla	;

nelle istruzioni di controllo	Non si usano le parentesi per delimitare la condizione; si va necessariamente a capo o si usa la virgola; si chiude il blocco con “end”:
	
	if x == 5
	  x = x^2
	end
	
	if x == 5
	  x = x^2;
	  v = z + x;
	end
		Si usano le parentesi tonde, non si va necessariamente a capo, i blocchi sono aperti e chiusi da parentesi graffe:
		
		if (x == 5) z = x^2;
		
		if (x == 5) {
		  z = x^2;
		  v = z + x;
		}

	Le funzioni possono restituire più risultati	Le funzioni restituiscono al più un risultato
… e molte altre		

Octave, oltre ad essere un **linguaggio di programmazione,** è uno strumento di calcolo molto potente che permette di lavorare in modo **interattivo**. In octave possiamo definire variabili e costanti ed effettuare operazioni con istruzioni di una sola riga.

Entrati in ambiente octave nella "Command Window" compare:

>>

il che significa che il calcolatore é pronto a ricevere istruzioni e ad eseguirle.

Se vogliamo eseguire la somma 3+2 basta scrivere:

>> 3+2

e premere il tasto invio. Otteniamo la risposta:

ans = 5

e il calcolatore é di nuovo pronto a ricevere un nuovo comando. Il risultato dell'operazione eseguita e` memorizzato nella *variabile* di parcheggio **ans**. In **ans** è memorizzato il valore dell'ultima espressione calcolata (se non è stato assegnato ad un'altra variabile).

Possiamo assegnare alla variabile **x** il valore 3:

```
>> x=3
x =  3
>> x^2
ans =  9
```

Per evitare l'effetto eco sul video si digita un "**;**" alla fine dell'istruzione.

Se vogliamo modificare il valore di **x**:

```
>> x=5;
```

Nota: Utilizzando i tasti |↑| e |↓| si possono recuperare gli ultimi comandi eseguiti.

Octave è case sensitive!

Tutte le **funzioni matematiche** elementari: trigonometriche, esponenziali e di arrotondamento sono funzioni predefinite di octave

alcuni esempi:

```
>> sqrt(2)
ans = 1.4142
```

```
>>
```

```
>> cos(0)
ans =
1
```

<u>Alcuni comandi:</u>

clc

clear

who

whos

Octave lavora essenzialmente su matrici. Le Matrici possono essere introdotte esplicitamente per mezzo di istruzioni di assegnazione.

Per esempio:

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

assegna ad a la matrice

1 2 3

4 5 6

7 8 9

In generale per costruire una matrice m x n si puo` utilizzare l'istruzione:

>>nome = [riga 1;riga 2;.....;riga m]
Ogni riga composta da n elementi separati da uno spazio o da una virgola.

INDICIZZAZIONE DI UNA MATRICE! a(1,3)

Funzioni predefinite per costruire matrici

Tra le funzioni predefinite in Octave ce ne sono alcune che permettono di costruire matrici particolari.Vediamo le piu` conosciute:

>>rand(m,n)
crea una matrice m x n di elementi casuali compresi tra 0 e 1.

>>zeros(m,n)
crea una matrice m x n di elementi tutti nulli

>>ones(m,n)
crea una matrice m x n di elementi tutti uguali a 1

Alcune funzioni predefinite di octave permettono di assegnare a variabili le dimensioni di una matrice (**size**) e la lunghezza di un vettore (**length**).

>>[m,n]=size(a)

>>m = size(a,1)

>>n=length(v)

Vettori

Possiamo creare vettori riga o colonna, con le stesse regole adottate per le **matrici.** Per i **Vettori riga** basta ad esempio scrivere tra parentesi quadre gli elementi del vettore in ordine, separati da uno spazio come ad esempio:

>>a=[1 2 3 4]

Per i **Vettori colonna** possiamo scrivere tra parentesi quadre gli elementi del vettore in ordine, separati da ";" come ad esempio:

>>v=[5;6;7]

Ribadiamo: gli spazi possono essere sostituiti da virgole,i punti e virgola dal ritorno a capo.

octave ci permette di costruire i **vettori** anche come **progressione aritmetica** di passo **p**.
La sintassi in generale è la seguente:

nome=[a:p:b]

dove **a** e **b** sono rispettivamente il primo e l'ultimo elemento della progressione (estremi dell'intervallo), **p** la distanza (passo) tra due valori successivi della progressione (**a**,**b** e **p** sono numeri reali).

Un CASO PARTICOLARE è **nome=[a:b]** che produrrà il vettore con primo elemento **a**, ultimo elemento **b** ed elementi intermedi a distanza di **1**.

ES.:

>>v = [2:10] oppure v = 2:10

v = 2 3 4 5 6 7 8 9 10

>>w = [2.5:0.3:4.2] ;

w = 2.5000 2.8000 3.1000 3.4000 3.7000 4.0000

>>z = [-5:-0.3:-6]

z = -5.0000 -5.3000 -5.6000 -5.9000

>>t = [9:-0.9:3]

t = 9.0000 8.1000 7.2000 6.3000 5.4000 4.5000 3.6000

Se abbiamo la necessità di ottenere un vettore con un numero fissato n di elementi equispaziati tra due estremi a e b possiamo usare l'istruzione linspace:

nome=linspace(a,b,n)

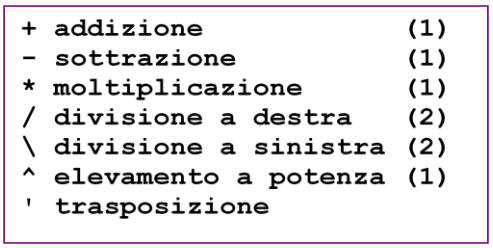
genera un vettore di n elementi tra a e b; ad esempio:

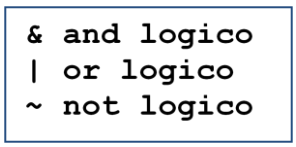
>> a = linspace (2,1,4)

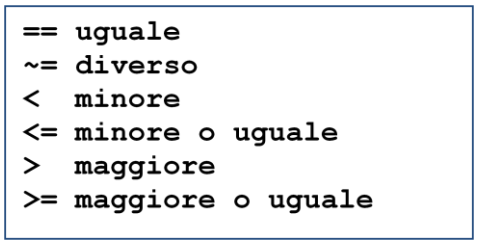
produce il vettore a di quattro elementi equispaziati dal valore 2 a 1:
a =
2.0000 1.6667 1.3333 1.0000

Un CASO PARTICOLARE è : nome=linspace(a,b) genera un vettore riga di 100 punti equispaziati tra a e b.

OPERAZIONI ARITMETICO-LOGICHE







IMPORTANTE: Se i caratteri `* , ^ , / , \ , '` sono preceduti da un punto operano elemento per elemento. Perchè abbia senso l'operazione, i due operandi devono avere dimensioni compatibili (oppure uno dei due operandi deve essere uno scalare):

```
C=A.*B equivale a C(i,j)=A(i,j)*B(i,j) con i,j indicanti riga e colonna

C=A./B equivale a C(i,j)=A(i,j)/B(i,j)

C=A.^B equivale a C(i,j)=A(i,j)^B(i,j)
```

STANDARD INPUT/OUTPUT

INPUT

L'istruzione di **INPUT** da tastiera in Octave ha la forma: **var = input('messaggio')**.

Eseguita l'istruzione, nella finestra di Command compaiono il messaggio e il cursore; il computer rimane in attesa di uno o più dati dalla tastiera..
Una volta inseriti i dati questi vengono assegnati alla variabile **var**.

Esempio:
a = input('a= ');

Sul video comparirà:
a= |

Dopo aver digitato per esempio [5,3] in a sarà memorizzato il vettore riga [5 3].

OUTPUT

Un'istruzione di output in Octave è **disp** che ha due forme:

disp('messaggio')

oppure

disp(var).

Nel primo caso produce il messaggio racchiuso tra apici, nel secondo caso produce il contenuto della variabile var al momento dell'istruzione.

Esempio:
```
>> a=5;
>> b=3:
>> c=a+b;
>> disp('La somma di a e b e`'), disp(c);
```

produce:

La somma di a e b e`

8

Load, save

Istruzioni grafiche

plot

Esiste in octave una funzione predefinita che ci permette di costruire grafici: plot

plot (x,y)

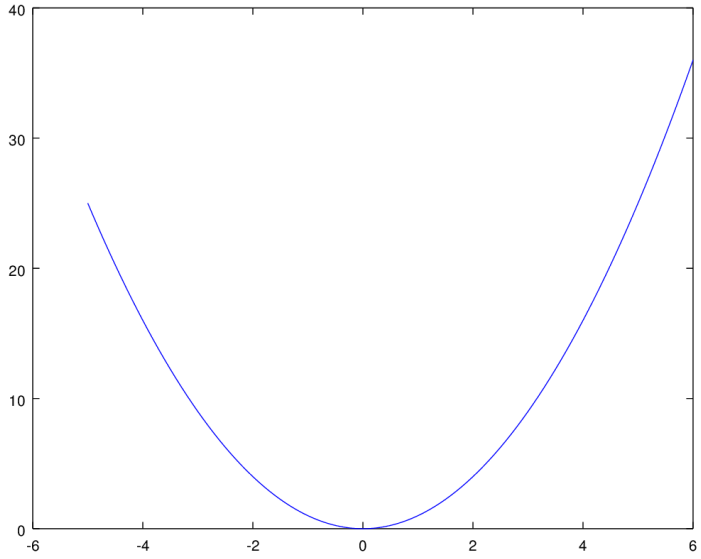
dove **x** è il vettore di ascisse comprese nell'intervallo e **y** il vettore dei corrispondenti valori di f(x).

Esempio:
```
x=[-pi:0.1:pi]; % vettore con primo elemento -π , ultimo elemento π e passo 0.1
y=sin(x) ;

plot(x,y)

x=[-5:0.1:6]; % vettore da -5 a 6, passo 0.1
y=x.^2 ;

plot(x,y)
```



Se si vuole modificare il range x e y visualizzato, immediatamente dopo l'istruzione plot si deve usare l'istruzione:

```
axis ([xmin xmax ymin ymax]),
```

dove xmin e xmax sono gli estremi dell'intervallo orizzontale e ymin e ymax gli estremi dell'intervallo verticale.

Un'opzione della funzione plot permette di differenziare i tipi di linee e il colore dei grafici. La sintassi e`:
`plot(x,y,'s')` dove x e y sono come nel caso precedente e s è una stringa di caratteri della lista seguente:

```
b blue              . point             - solid
g green             o circle            : dotted
r red               x x-mark            -. dashdot
c cyan              + plus              -- dashed
m magenta           * star
y yellow            s square
k black             d diamond
                    v triangle (down)
                    ^ triangle (up)
                    < triangle (left)
                    > triangle (right)
                    p pentagram
                    h hexagram
```

```
x=[-pi:0.1:pi]; y=sin(x); plot(x,y,'b*')
```

Più grafici possono essere plottati in una stessa finestra scrivendo tra gli argomenti di plot più coppie di vettori, ad es.:

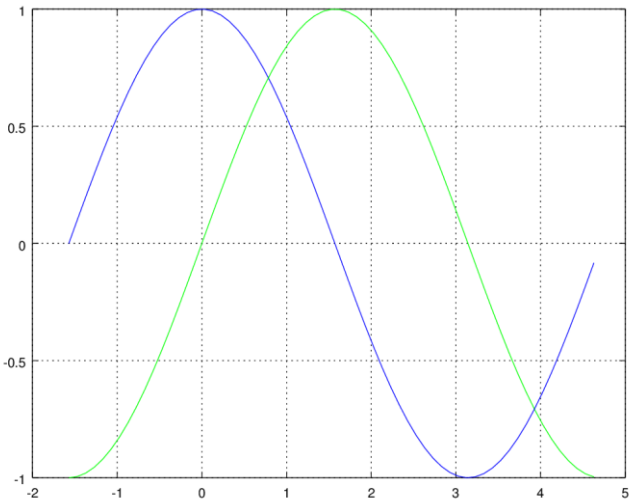
plot (x,y,x,z,h,z)

Octave sceglie i valori min e max tra tutti i vettori (ascisse e ordinate) e plotta i grafici.

Per sovrapporre più grafici usando plot in momenti successivi occorre il comando hold:

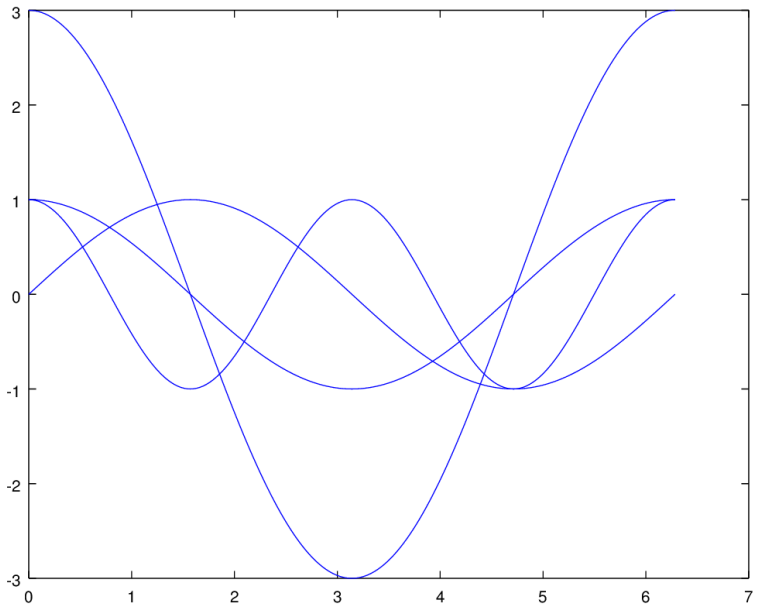
x=-pi/2:.1:3/2*pi;

```
plot(x,cos(x))
hold on  % (o semplicemente hold)
plot(x,sin(x),'g'), grid on
hold off
```



Ancora:

```
X = 0.0:pi/100:2*pi;
Y1 = cos(X);
Y2 = 3*cos(X);
Y3 = cos(2*X);
Y4 = sin(X);
plot(X,Y1), hold on
plot(X,Y2)
plot(X,Y3)
plot(X,Y4), hold off
```



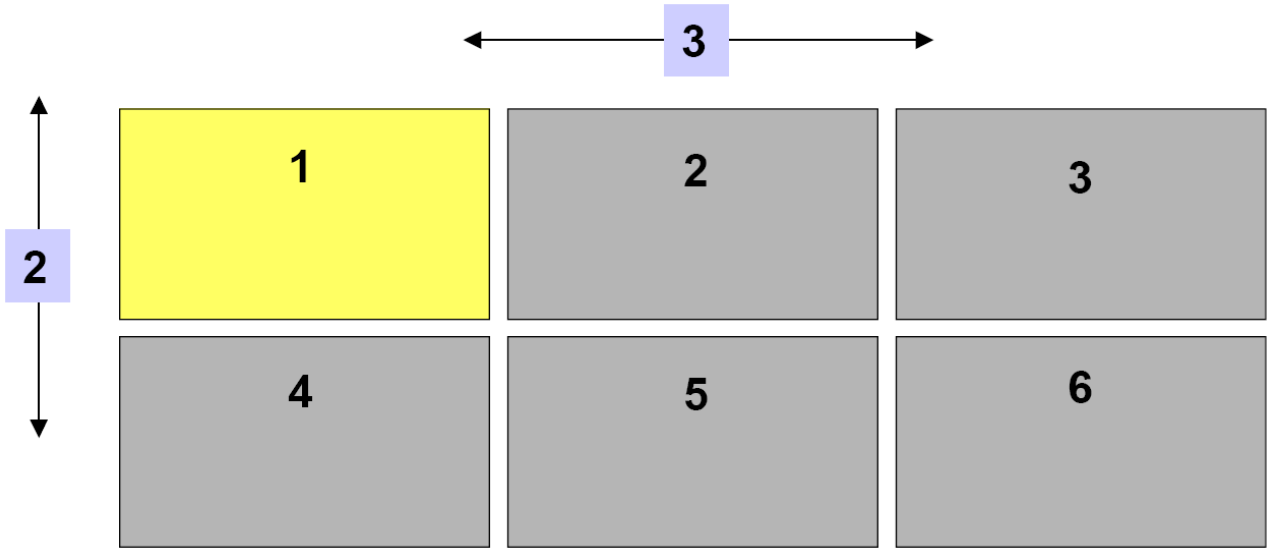
L'istruzione **subplot** permette di plottare più grafici contemporaneamente nella stessa *figure*.

• **subplot(m,n,k)**

m=righe, n=colonne

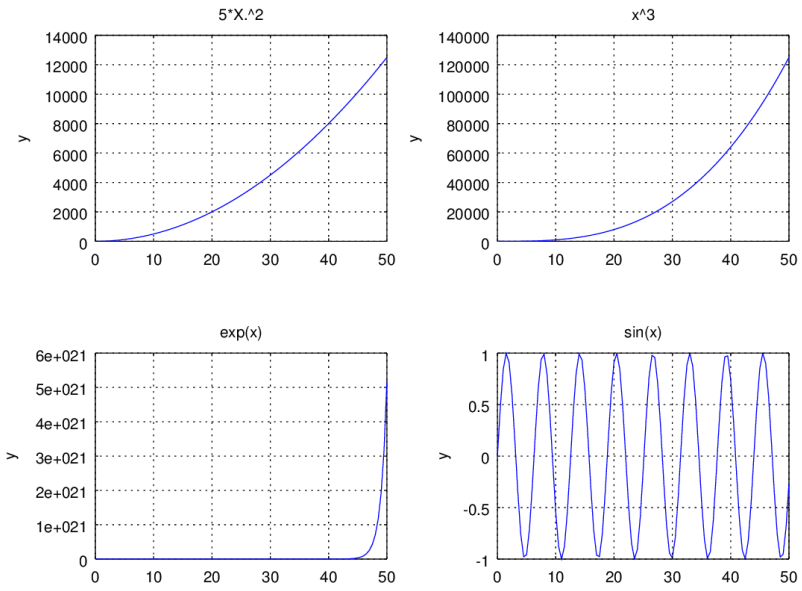
k=posizione corrente (per righe)

Esempio: **subplot(2,3,1)** invia le seguenti istruzioni grafiche (plot etc) nel riquadro in giallo.



X=0:0.5:50;
Y1=5*X.^2;Y2=X.^3; Y3=exp(X);Y4=sin(X);
subplot(2,2,1), plot(X,Y1), title('5*X.^2'), ...
ylabel('y'), grid

subplot(2,2,2), plot(X,Y2), title('x^3'), ...
ylabel('y'), grid
subplot(2,2,3), plot(X,Y3), title('exp(x)'), ...
ylabel('y'), grid
subplot(2,2,4), plot(X,Y4), title('sin(x)'), ...
ylabel('y'), grid



fplot('function',limits)

esempi:

fplot('sin(x)', [0 2*pi])
figure
fplot('x^3-1',[-1,1],'r--')

La funzione figure crea una nuova finestra grafica che diviene figura attiva.

(copia-incolla per portare i grafici in un file .doc)

STRUTTURE DI CONTROLLO

FOR

In octave la ripetizione di blocchi di istruzioni per un numero di volte specificato e in modo incondizionato viene eseguita tramite l'istruzione di ciclo FOR ...END la cui sintassi e`:

for indice = espressione
blocco di istruzioni

end

Dove **indice** è una quantità che assume diversi valori a seconda di espressione, e **end** segna la fine del blocco di istruzioni da ripetere.
Spesso **espressione** ha la forma **x1:x2** oppure **x1:step:x2**. **indice** viene utilizzato come contatore, **x1** è il valore iniziale assunto dal contatore, **step** l'incremento dato al contatore ad ogni ciclo (in mancanza di **step** l'incremento è 1), **x2** il valore finale che controlla il ciclo.
Il blocco di istruzioni verrà ripetuto tante volte quanti sono i valori che assume il contatore.

Esempio:

```
for i = 1 : 10
    disp('Siamo al ciclo n.')
    disp(i)
end
```

for i = [2, 3, 5, 7, 11, 13, 17, 19]
se vogliamo scandire i numeri primi.

IF
Se una o più istruzioni devono essere eseguite solo sotto condizione si usa l'istruzione IF la cui sintassi nella forma più semplice è:

```
if condizione
    blocco di istruzioni
end
```

dove "**condizione"** può essere un test di confronto tra due espressioni numeriche
che utilizza i seguenti operatori:
< minore di
<= minore o uguale di
> maggiore di
>= maggiore o uguale di
== uguale a
~= diverso da
oppure una combinazione di singoli test per mezzo di operatori logici:
&& and
|| or
~ not

Esempio

Se a, b, c sono i coefficienti di un'equazione di secondo grado, per calcolare solo le radici reali possiamo costruire una `function`:

```
function [x1,x2] = eq2(a,b,c)
delta = b^2-4*a*c;
if delta >= 0
    disp('Le radici sono reali')
    rad=sqrt(delta);
    x1=(-b-rad)/(2*a);
    x2=(-b+rad)/(2*a);
end
```

La sintassi più generale dell'istruzione if è:

```
if condizione1
   blocco di istruzioni 1
elseif condizione2
   blocco di istruzioni 2
   ...
else
   blocco di istruzioni n
end
```

WHILE

Se si ha la necessità di ripetere una o più istruzioni fintanto che una condizione sarà verificata non sapendo a priori il numero di ripetizioni, è necessario usare l'istruzione **WHILE ...END** la cui sintassi è:

```
while condizione
   blocco di istruzioni
end
```

Dove *blocco di istruzioni* verrà eseguito fintanto che condizione risulta vera.

Esempio:

Dovessimo sommare tutti i voti di uno studente (ad esempio per farne la media), senza doverne conoscere a priori il numero, potremmo costruire questo script:

```
nvoti = 0; somma = 0;
voto = input('voto? (0 per finire)');
while voto ~= 0
   somma = somma + voto;
   nvoti = nvoti + 1;
   voto = input('voto? (0 per finire)');
end
media = somma/nvoti;
disp (media)
```

SWITCH

```
switch switch_expr
  case case_expr
    statement,...,statement
  case {case_expr1,case_expr2,case_expr3,...}
    statement,...,statement
...
  otherwise
    statement,...,statement
End
```

L'istruzione **switch** fa una selezione fra più possibilità. Ad esempio se si attiva uno dei nomi seguenti, viene segnalato quale tra essi è attivo.

```
% nome='ortensia';
nome='rosa';
% nome='giglio';
% nome='garofano';
switch nome
  case 'rosa'
    disp('si tratta di una rosa')
  case 'garofano'
    disp('si tratta di un garofano')
  case 'giglio'
    disp('si tratta di un giglio')
  otherwise
    disp('si tratta di un altro fiore')
end
```

BREAK

break permette di terminare immediatamente l'esecuzione di un un ciclo for o while (ad esempio in caso di errore). Quando è eseguita l'istruzione break Matlab salta automaticamente all'istruzione end che termina il ciclo.

Esempio:

```
  for i = 1 : 10
    a = input('');
    if a == 0; disp('attenzione e` un denominatore!'); break; end
    x(i) = 1/a;
  end
```

Se vi sono più cicli annidati, break termina solo quello più interno in cui si trova.

M-FILE

Matlab consente di memorizzare una sequenza di istruzioni in un *file*; questo, per essere accessibile, deve avere l'estensione ".m"e pertanto si chiama M-file.

Gli **M-file** possono essere di due tipi: ***script*** o ***function***.

M-file tipo SCRIPT (per brevità: *script*)

Contengono semplicemente una **sequenza di istruzioni** Matlab, nella forma in cui si scriverebbero dalla linea attiva della finestra di Command. Utilizzano tutte le variabili definite in precedenza e, al termine dell'esecuzione, tutte le eventuali modifiche sono visibili all'esterno. In questo senso si dice che tutte le variabili sono GLOBALI. Accertatisi che la "Current Directory" di Matlab sia la stessa in cui è memorizzato lo *script*, per eseguirlo è sufficiente digitare, nella finestra di Command, il nome del file (senza l'estensione *.m*).

>> sommeprodotto ¿

Oppure, in alternativa, soprattutto mentre si costruisce o corregge uno *script* all'interno del M-file Editor, selezionare nel menu`: **Debug** e poi **Run.**

M-file tipo FUNCTION (per brevità: *function*)

Definiscono una nuova funzione Matlab. Essi accettano dei dati in input e restituiscono dei dati di output come risultato della loro elaborazione.

La prima istruzione distingue la *function* e deve avere questa forma sintattica:

function[o1,o2,.....,on] = nome-funzione (i1,i2,.....,im)

dove **o1,o2,.....on** sono le *variabili di output*, **nome-funzione** e` il nome del file (senza l'estensione .m) e **i1,i2,.....im** sono le *variabili di input*. Se la variabile di output è una sola si possono omettere le parentesi quadre. Le linee successive contengono la sequenza di istruzioni necessarie a calcolare l'output a partire dall'input.

M-file tipo FUNCTION (continuazione)

- Nel corpo di una *function* Matlab tutte le variabili sono LOCALI, cioè non sono visibili all'esterno della *function,* salvo se sono definite "*global*". Le *function* dialogano quindi con l'esterno solo attraverso le variabili di input e output. Grazie a questa loro caratteristica di "scatole nere", è buona abitudine non scrivere, dentro una *function*, istruzioni di **input/output** , ma far eseguire questi comandi ad un file *script* esterno che richiama le *function* (vedi l'esempio). Per lo stesso motivo, conviene terminare tutte le istruzioni con il "punto e virgola" in modo da evitare output indesiderati sullo schermo.
- Evitare di salvare le function con nomi di funzioni interne di Matlab poichè, al momento della chiamata, Matlab cerca prima tra le function interne, per cui quella programmata dall'utente con stesso nome non verrà mai considerata.

- Analogamente agli script anche le function programmate dall'utente sono trovate (ed eseguite) da Matlab solo se la "Current Directory" è la stessa in cui sono state memorizzate (già detto negli script) oppure se il cammino in cui si trovano è aggiunto al PATH conosciuto da Matlab.
- In testa ad uno *script* o ad una *function*, possono essere scritte linee di commento, ad esempio sul significato delle variabili in ingresso e uscita della *function* stessa, precedute (come tutti i commenti in Matlab) dal simbolo **%**. Questo è molto utile perchè in qualunque momento successivo si possono richiamare, nella finestra di Command, questi commenti semplicemente digitando il comando:

help nome-funzione.

Per eseguire una function occorre digitare una chiamata nella stessa forma in cui e` scritta la prima istruzione :

[w1,w2,.....,wn] = nome-funzione (v1,v2,.....,vm)
dove questa volta **v1,v2,.....,vm** contengono valori noti (anche sotto forma di variabili gia` definite), mentre **w1,w2,.....,wm** sono le variabili in cui vogliamo memorizzare i risultati, e non coincidono necessariamente con i nomi **o1,o2,.....,on** (inseriti nella definizione della funzione)

M-file tipo FUNCTION (esempio)

```
%    function [s,p] = sumprod(x,y)
% calcola somma e prodotto di due numeri x e y
%
function [s,p] = sumprod(x,y)
s = x+y;
p = x*y;
```

Memorizzare nel file **sumprod**.m!

```
>> help sumprod
    function [s,p] = sumprod(x,y)
calcola somma e prodotto di due numeri x e y
>> [som,prodot] = sumprod(3,4)
som = 7
prodot = 12
```
Si notino i valori espliciti passati in input e il diverso nome delle variabili in output.

M-file tipo FUNCTION (continuazione)
Possiamo ora salvare il seguente *script* nel file **sommeprodotto**.m:

```
% programma che calcola somma e prodotto di a e b dati in input,
% utilizzando la function [s,p] = sumprod(x,y)
%
a=input('1^ numero');
b=input('2^ numero');
[som,prodot]=sumprod(a,b);
disp('la somma e'''); disp(som)
disp('il prodotto e'''); disp(prodot)
```
che eseguiamo con:
```
>> sommeprodotto
1^ numero 5
2^ numero 6
la somma e'
11
il prodotto e'
30
```

M-file tipo FUNCTION (continuazione)

Tipi di function possibili in base al numero di parametri e valori in uscita:

```
function f1
disp('Eccomi, sono f1!')

function m = f2
disp('Eccomi, sono f2, e ti restituisco un valore!')
m = 5;

function f3(x, y)
disp('Eccomi, sono f3, e ti stampo x e y!')
fprintf('x = %f, y = %f\n', x, y);

function [q , c] = f4(x)
disp('Eccomi, sono f4, ti stampo x e te ne restituisco il quadrato e il cubo!')
fprintf('x = %f\n', x);
q = x^2;
c = x^3;
```

Le quattro funzioni si chiamano rispettivamente con:

```
f1
x = f2
f3(13, 23)
[a, b] = f4(5)
```