# Linguaggi di programmazione nella fusione

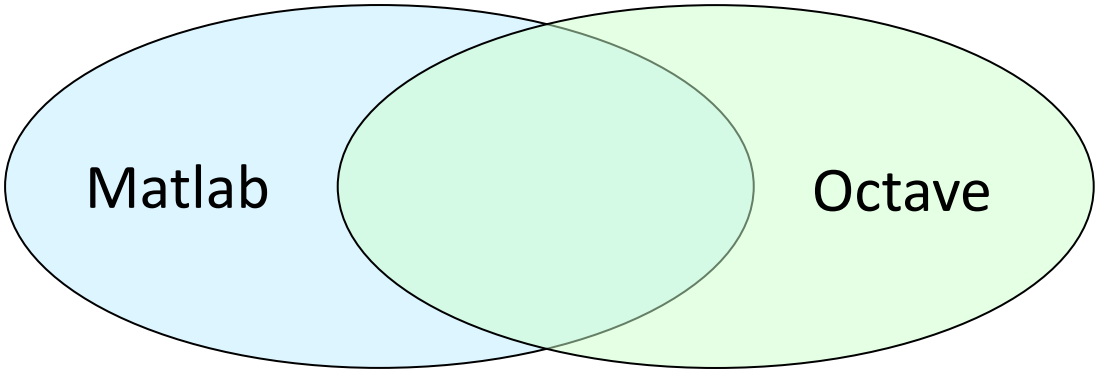
Edmondo Giovannozzi

Introduzione a Octave.

edmondo.giovannozzi@enea.it

# Introduzione

- Octave è un linguaggio ed un interprete.
- Il linguaggio è simile a Matlab® con minori differenze, a cui comunque bisogna fare attenzione.
- Gli sviluppatori cercano di mantenersi il più compatibili possibile.
- Non tutti i toolbox e tutte le funzioni di Matlab® sono disponibili in Octave.
- Package di Octave: http://octave.sourceforge.net/



# Gestione dei Package di Octave

Lista dei package installati

**>> pkg** list

...

installare un package da sourceforge

**>> pkg** install -forge *odepkg*

...

caricare un package installato ed avere accesso alle sue procedure (*odepkg* come esempio).

**>> pkg** load *odepkg*

...

# Per iniziare

- Per iniziare:

  **Lanciare l'applicazione**
- Per uscire:

  **>> exit**
- Help

  **>> help <*nome comando*>**

  Esempio:

  **>> help plot**
- Informazioni sulle variabili presenti in memoria:

  **>> whos**
- Le variabili di base sono le matrici di numeri reali, ma ci sono anche le matrici logiche, le stringhe (array di caratteri), gli array di celle e le strutture.

Attenzione!

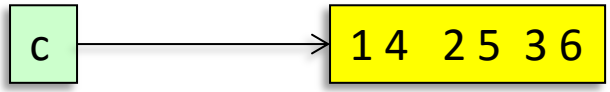
In Matlab e Octave, come in Unix le maiuscole e le minuscole sono distinte.

Tutti i comandi e le funzioni standard sono in minuscolo.

# Matrici Reali

```
>> a = [1, 2, 3, 4];          % vettore  orizzontale di numeri reali
>> b = [1; 2; 3; 4];          % vettore verticale di numeri reali
>> c = [1,2,3; 4,5,6]         % matrice 2x3 di numeri reali
c =
    1    2    3
    4    5    6
```

Attenzione come in FORTRAN le matrici sono memorizzate per colonne:




```
>> whos
        a          1x4                32  double
        b          4x1                32  double
        c          2x3                48  double
>> repmat([1,2,3], [2,1])
ans =   1   2   3
        1   2   3
>> clear          % per cancellare le variabili create
```

per creare una matrice a partire da un'altra ripetendola lungo le diverse dimensioni

# Creazione Matrici Reali

```
>> a = ones(2,3);  % matrice 2x3 piena di 1
>> b = zeros(3,2); % matrice 3x2 piena di 0
>> c = 0:0.4:2     % vettore di reali tra 0 e 2 a passi di 0.4
c =
         0    0.4000    0.8000    1.2000    1.6000    2.0000

>> e = linspace(0, 10, 3)  % vettore di 3 reali tra 0 e 10
e =
         0    5.0  10.0

>> d = c' ;        % vettore colonna trasposto di c
>> whos
          a          2x3                48  double
          b          3x2                48  double
          c          1x6                24  double
          d          6x1                48  double
          e          1x3                24  double
```

# Indici

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

Selezione del 5 elemento in memoria
>> **a(5)**
ans = 3

Selezione di un singolo elemento
>> **a(2,3)**
ans = 6

Selezione di una colonna
>> **a(:,2)**
ans =
2
5

Tutti gli elementi
>> **a(:)**
ans =
1
4
2
5
3
6

Indici:

| | |
|---|---|
| ***k*** | l'elemento **k**-esimo. |
| ***k*:*h*** | dall'elemento **k** a quello **h**. |
| ***k*:*d*:*h*** | come sopra a passi di **d**. |
| ***k*:end** | dall'elemento **k** alla fine. |
| ***k*:end-*1*** | dall'elemento **k** al penultimo |
| **:** | tutti gli elementi di quella dimensione. |
| **a** | con **a** vettore di indici, gli elementi corrispondente all'indice. |
| **b** | con **b** vettore logico, gli elementi per cui **b** è 1. |

# Indici (2)

| 2 : 2 : end | | | | | |
|---|---|---|---|---|---|
| | 2 | | 4 | | 6 |
| F | T | F | T | F | T |

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

1 : 2 : end

```
>> a(1:2:end, 2:2:end)
>> a(1:2:end, [false, true, false, true, false, true])
>> a(1:2:end, [2,4,6])
```

→

| | | |
|---|---|---|
| 2 | 4 | 6 |
| 5 | 3 | 1 |

```
>>  [amn, imn] = min(a)
amn =  1   2   2   1   2   1
imn =  1   1   2   2   3   3
```

minimo lungo il primo asse

indice nel primo asse dove si trova il minimo

```
>> id = sub2ind(size(a), imn, 1:6)
>> a(id)
ans =  1   2   2   1   2   1
```

id: indice lineare corrispondente

| | | | | | | |
|---|---|---|---|---|---|---|
| imn | → | 1 | 1 | 2 | 2 | 3 | 3 |
| 1:6 | → | 1 | 2 | 3 | 4 | 5 | 6 |
| id | → | 1 | 4 | 8 | 11 | 15 | 18 |

# Operatori (1)

- Molti operatori lavorano elemento per elemento:

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

- Così anche molte funzioni elementari:

```
>> y = sin(0:0.4:2)
y =
         0    0.3894    0.7174    0.9320    0.9996    0.9093
```

# Operatori (2)

- Gli operatori matematici: “*****” , “**/**” , “**^**” , “****” sono matriciali, i corrispondenti operatori che operano elemento per elemento sono: “**.***”, “**./**”, “**.^**” , “**.**”.
- L’operatore “****” risolve il sistema di equazioni matriciale:

  **A*X** = **B** , **X** = **A\B**.

Attenzione:

```
>> a = [1 ; 2 ; 3];
>> 1 / a
ans =
    [ 0.07    0.14    0.21]
```

L’operazione matematica è definita, anche se molto spesso non è ciò che volevamo. Per avere l’inverso del vettore, elemento per elemento dovevamo scrivere:
**>> 1 ./ a**

# Esempi operatori matriciali

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

>> b = [10;
        20]

>> A * b
ans =   50
       110
       170
```

```
>> c  = [10;
         20;
        -10]

>> p = A \ c
p =  -26.667
      21.667
```

Soluzione sistema:

$$A\,p = c$$

Ai minimi quadrati.

Moltiplicazione matriciale, ovviamente il numero di colonne di **A** deve essere uguale al numero di righe di **b**

# Numeri complessi

I numeri complessi sono gestiti automaticamente.

```
>> a = log(-2)
a =      0.6931 + 3.1416i
>> a + (3 + 2i) + 3j
ans =  3.6931 + 8.1416i
>> b = complex(2)
b     =  2
>> isreal(b)
ans = 0
>> imag(b)
ans = 0
>> exp(a)
ans  = -2.0000 + 0.0000i
>> imag(exp(a))
ans  = 2.4493e-16
```

**Funzioni**:

| | |
|---|---|
| **conj** | complesso coniugato. |
| **real** | parte reale. |
| **imag** | parte immaginaria. |
| **abs** | magnitudine. |
| **angle** | angolo di fase in radianti. |
| **isreal** | test se è un vettore reale o complesso |

```
>> a = [1,2i;3,4i]
a =    1          0 + 2i
       3          0 + 4i
>> a'
ans = 1          3
      0 – 2i     0 – 4i
>> a.'
ans = 1          3
      0 + 2i     0 + 4i
```

Attenzione, l'operatore " **'** " calcola la matrice trasposta coniugata, usare l'operatore " **.'** " per la semplice trasposizione

# Array e operatori logici

```
>> a = [10, 20, 30, 40, 50, 60];
>> b = a > 30
b =
     0     0     0     1     1     1
>> whos
 Name Size  Bytes  Class
  a       1x6   48     double
  b       1x6    6     logical
>> a(b)
ans =
    40    50    60
>> id = find(b)
id =
     4     5     6
>> a(id)
ans =
    40    50    60
```

Operatori di comparazione:

| | |
|---|---|
| == | uguale a |
| ~= | diverso da |
| >, < | maggiore, minore |
| >=, <= | maggiore uguale, minore uguale |

Operatori e funzioni logiche:

| | |
|---|---|
| **&** | and logico (elemento per elemento) |
| **&&** | and logico (short circuit, scalare) |
| **\|** | or logico(elemento per elemento) |
| **\|\|** | or logico(short circuit, scalare) |
| **~** | not logico (elemento per elemento) |
| **xor** | or esclusivo (funzione) |
| **any** | **vero** se qualche elemento di un vettore è diverso da zero o è **vero** (funzione). |
| **all** | **vero** se tutti gli elementi di un vettore sono diverso da zero o sono **veri** (funzione). |
| **true** | **vero** |
| **false** | **falso** |

# Stringhe

Le stringhe sono vettori riga di caratteri:

```
>> sa = [‘pluto’ ; ‘pippo’]
sa =
pluto
pippo
>> whos
 Name      Size            Bytes  Class
  sa        2x5                10  char array
```

Attenzione in memoria sono memorizzate per colonne

```
>> sa(:)’
ans =
ppliuptpoo
```

Principali funzioni:

| | |
|---|---|
| **ischar** | test se un vettore è una stringa. |
| **strcmp** | compara due stringhe. |
| **findstr** | trova una stringa all’interno di un’altra. |
| **strmatch** | trova una stringa in un array di stringhe od un cell array di stringhe (vedi dopo). |
| **strtok** | separa una stringa in token (uno per volta). |
| **regexp** | trova un espressione regolare in una stringa. |
| **num2str** | trasforma un numero in una stringa. |
| **str2double** | trasforma una stringa in un numero. |
| **sprintf** | Output su una stringa. (vedi I/O) |
| **sscanf** | Input da una stringa. (vedi I/O) |

# Array di celle

```
>> a = {1, 'Ciao', [1,2,3] }
a =
    [1]    'Ciao'    [1x3 double]
```

array di celle: può contenere oggetti di diversa natura. Si creano con le parentesi graffe.

```
>> b = a(2);
```

è ancora un array di celle composto di una sola cella.

```
>> c = a{2};
```

è l'elemento nella cella

```
>> whos
 Name   Size      Bytes  Class
  a      1x3         36    cell
  b      1x1          4    cell
  c      1x4          4    char
```

Per inserire un elemento in un array di celle preesistente:

```
>> a(2) = {'FTU'}
>> a{2} = 'FTU'
>> a(2:3) = {'FTU',123}
>> a{2:3} = ??      errato
```

```
>> st = {'pippo','pluto','topolino'};
```

specialmente utile con stringhe di diversa lunghezza

# Strutture

```
>> a.x = 1;
>> a.y = [1,2,3];
>> a.s = 'Ciao'
a =
    x: 1
    y: [1 2 3]
    s: 'Ciao'
>> a(2).x = 'pluto'
a =
1x2 struct array with fields:
    x
    y
    s
>> a(1).y
ans =
     1     2     3
>> c = 'y'
>> a(1).(c)
ans =
     1     2     3
```

Un'unica variabile **a** è composta di diversi campi (**x**, **y**, **s** nell'esempio), che possono essere variabili di qualsiasi tipo.

Si può avere un array di strutture.

Stessa sintassi per leggere il valore di un campo

Nome del campo passato tramite una variabile

# Funzioni matematiche

```
>> pi
ans = 3.1416
>> atan2(3,4)
ans = 0.6435
>> round(3.6)
ans = 4
>> floor(3.6)
ans = 3
>> mod(13,5)
ans = 3
>> 1/0
ans = Inf
>> eps
ans = 2.2204e-16
```

Tutte le principali funzioni trigonometriche ed esponenziali:

**acos**, **acosh**, **acot**, **acoth**, **acsc**, **acsch**, **asec**, **asech**, **asin**, **asinh**, **atan**, **atanh**, **atan2**, **cos**, **cosh**, **cot**, **coth**, **csc**, **csch**, **exp**, **log**, **log2**, **log10**, **sec**, **sech**, **sin**, **sinh**, **tan**, **tanh**, ...

E quelle più specialistiche:

**airy**, **besselh**, **besseli**, **besselj**, **besselk**, **bessely**, **beta**, **betainc**, **betaln**, **ellipj**, **ellipke**, **erf**, **erfc**, **erfcinv**, **erfcx**, **erfinv**, **expint**, **gamma**, **gammainc**, **gammaln**, **legendre**, etc. ...

Costanti principali:

| | |
|---|---|
| **pi** | pi greco |
| **i,j** | unità immaginaria |
| **eps** | Accuratezza relativa di un numero reale |
| **Inf** | Infinito |
| **NaN** | Not-a-Number |

# Funzioni che lavorano sugli Array

```
>> a = [1,2,4];
>> mean(a)
ans =  2.3333
>> a = [1,2; 3,4; 6,8]
a  =  1   2
      3   4
      6   8
>> mean(a)
ans  =  3.3333  4.6667
>> mean(a,2)
ans  =   1.5
         3.5
         7.0
>> diff(a)
ans =  2    2
       3    4
>> cumtrapz([10,20,30],a)
ans = 0     0
      20    30
      65    90
```

Principali funzioni di analisi:

| | |
|---|---|
| **max, min** | massimo, minimo di una matrice |
| **sum, prod** | somma, prodotto degli elementi |
| **mean, std** | media, deviazione standard |
| **median** | mediana |
| **trapz** | integrazione numerica trapezoidale |

Se applicati ad una matrice lavorano sulla prima dimensione, che diventa unitaria.

Funzioni cumulative

| | |
|---|---|
| **cumsum** | somma cumulativa |
| **cumtrapz** | integrale trapezoidale cumulativo |

Altre:

| | |
|---|---|
| **sort** | ordina un vettore |
| **diff** | differenza numerica |
| **gradient** | gradiente |
| **fft** | FFT |
| **filter** | filtro numerico (convoluzione) |

# Input Output

```
>> a = [1,2,3];
>> b = 'Ciao';
>> save prova
>> clear
>> load prova
>> c = load('prova')
c =     a: [1 2 3]
        b: 'Ciao'
>> whos
 Name  Size Bytes  Class
  a       1x3     24  double array
  b       1x4      8  char array
  c       1x1    176  struct array

>> a = [1,2,3; 4,5,6];
>> save pluto a –ascii
>> type pluto
 1.000e+00   2.000e+00   3.000e+00
 4.000e+00   5.000e+00   6.000e+00
```

Per salvare dei dati e recuperarli usare le funzioni **save** e **load**. i dati vengono scritti in binario in un file **.mat** (Matlab) od in un file di testo (Octave). Con l'opzione '**-7**' si può forzare Octave a scrivere un file binario nel formato di Matlab.

**>> save** *nomefile variabile1 variabile2*

**>> load** *nomefile*

**>>** *var* **= load(**‘*nomefile*’**)**

Nell’ultimo caso le variabili vengono memorizzate come membri in una struttura, con l’opzione **–ascii** si scrivono e si leggono in formato ascii (usare solo con singole matrici).

```
>> fid = fopen('pippo','w')
>> fprintf(fid,'Ciao %f',3.5)
>> fclose(fid)
>> type pippo
Ciao 3.500000
>> fprintf('Ciao Ciao')
Ciao Ciao
>> a = sprintf('Ciao')
a = Ciao
```

Senza l’indicazione del file scrive sullo schermo

# Lettura file

```
>> type pluto
 1.0000e+00   2.0000e+00   3.0000e+00
 4.0000e+00   5.0000e+00   6.0000e+00
>> a = load('pluto')
a = 1   2   3
    4   5   6
```

Altri comandi di input:

**fscanf** Lettura formattata da un file
**fgetl** Lettura di una linea di testo da un file
**input** Lettura di un input dall'utente

Esempio:

```
>>  b = input('Dimmi : ')
Dimmi : 34
b = 34
```

```
>> pkg load io
>> a = xlsread('OctaveTest.xlsx',1,[],'OCT')
```

Lettura di file Excel
La sintassi di Matlab è lievemente differente

```
>> a
a =
   0.100000   0.099833   0.995004
   0.300000   0.295520   0.955336
   0.500000   0.479426   0.877583
```

| | A | B | C | D |
|---|---|---|---|---|
| 1 | tempo | a | b | |
| 2 | 0.1 | 0.099833 | 0.995004 | |
| 3 | 0.3 | 0.29552 | 0.955336 | |
| 4 | 0.5 | 0.479426 | 0.877583 | |
| 5 | 0.7 | 0.644218 | 0.764842 | |

# Input/output formati

| Formato | tipo |
|---|---|
| **%*n*.*m*f** | Virgola fissa |
| **%*n*.*m*d** | Intero |
| **%*n*.*m*e** | Esponenziale |
| **%*n*.*m*g** | Esponenziale/Virgola fissa |
| **%*n*.*m*x** | Esadecimale |
| **%*n*s** | Stringa |

La larghezza del campo è ***n***. Se il numero risulta più lungo, il campo si allunga di.
Per il formato **f** *m* è il numero di cifre dopo la virgola, per i formati **e**,**g** è la precisione. Per il formato intero (**d**,**x**) *m* è il numero di cifre che deve essere scritto, eventualmente mettendo degli zeri sulla sinistra del numero. Si può ottenere un risultato simile riempendo il campo con zeri anche con il seguente formato: '**%0*n*d**'.
Se *n* è negativo i numeri si allineano a sinistra.

Esempi:

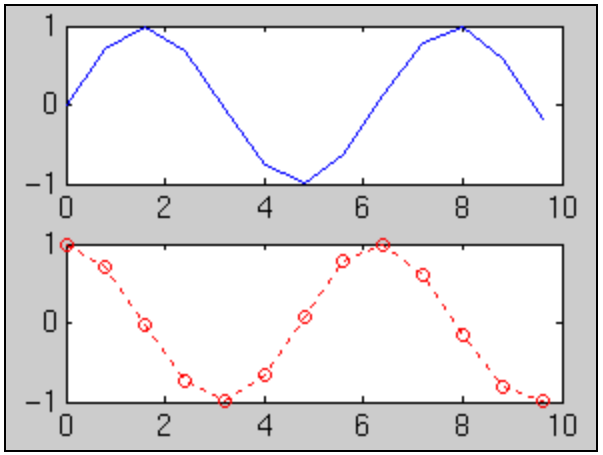
```
>> fprintf('a=%6.2f',3.5)
a=  3.50
>> fprintf('b=%-6.3dA',11)
b=011   A
>> fprintf('b=%06d',11)
b=000011
>> fprintf('Ciao \nCiao',11)
Ciao
Ciao
```

Caratteri speciali

| | |
|---|---|
| **\n** | New line |
| **\r** | Carriage return |
| **\t** | Horizontal tab |
| **** | Backslash |
| **''** | Single quotation mark |
| **%%** | Percent character |

# Grafica

```
>> x = 0:0.8:10;
>> y1 = sin(x);
>> y2 = cos(x);
>> figure
>> subplot(2,1,1)
>> plot(x,y1)
>> subplot(2,1,2)
>> plot(x,y2,'or--')
```




**figure(n)** apre una finestra grafica con il numero **n**.

**subplot(nr,nc,k)**

| | |
|---|---|
| (3,2,1) | (3,2,2) |
| (3,2,3) | (3,2,4) |
| (3,2,5) | (3,2,6) |

crea o seleziona il k-esimo asse, assumendo che la figura sia divisa in **nr** righe ed **nc** colonne.

**plot(x,y,stile,…)** plot del vettore **x** vs **y**. stile è una stringa che definisce il tipo di marker, lo stile della linea, ed il colore.

Vedere l'help della funzione **plot** per i dettagli.
Nell'esempio considerato: '**o**' marker circolare, '**r**' linea e marker di colore rosso, '**- -**' linea tratteggiata.

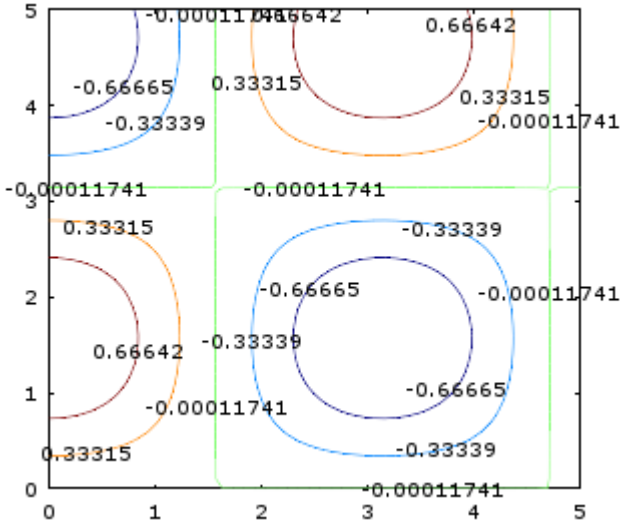
# Grafica (2)

```
>> x = linspace(0,10, 50);
>> y = linspace(0,10, 51);
>> [xx, yy] = meshgrid(x, y);
>> z = cos(xx) .* sin(yy);
```

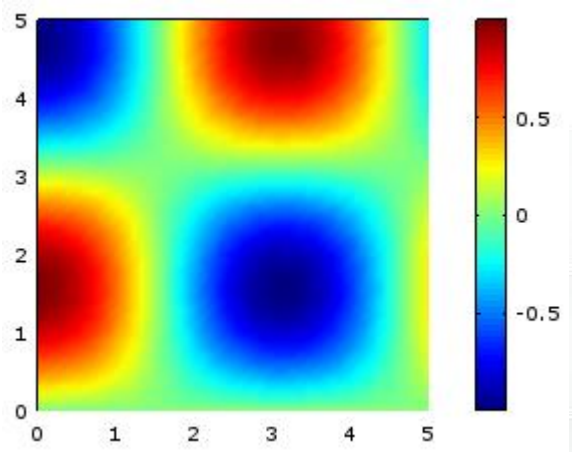
| | |
|---|---|
| x | 1x50 |
| xx | 51x50 |
| y | 1x51 |
| yy | 51x50 |
| z | 51x50 |

```
xx =
0.00   0.102   0.204 ...
0.00   0.102   0.204 ...
0.00   0.102   0.204 ...
...
yy =
0.00   0.00   0.00 ...
0.10   0.10   0.10 ...
0.20   0.20   0.20 ...
```

```
>> [c,h] = contour(x,y,z, 5);
>> clabel(c,h)
```




```
>> pcolor(x, y, z)
>> shading interp
>> colorbar
>> caxis
ans =  -0.999   0.999
```



# Strutture di programmazione

```
if <condizione>
        …
elseif <condizione>
        …
else
        …
end
```

```
switch <variabile>
case <valore>
        …
case {<valore1>, <valore2>}
        …
otherwise
        …
end
```

*<variabile>* e *<valore>* possono essere un'espressione il cui risultato è un numero scalare o una stringa.

```
for i = <beg>:<end>
        …
end
```

```
while <condizione>
        …
end
```

```
try
        …
catch
        …
end
```

Gestione dell'errore

Normalmente vengono eseguite le istruzioni tra **try** e **catch**, in caso di errore quelle tra **catch** ed **end**.

| | |
|---|---|
| **break** | uscita anticipata da un ciclo **for** o **while** |
| **continue** | salta le rimanenti istruzioni e va alla successiva iterazione. |

# Funzioni e Script

Il nome di una funzione o di uno script è quello del file in cui risiede.

I nomi dei file terminano in **.m**

**Script**:

sequenza di comandi come se fossero immessi da tastiera. Vedono le stesse variabili che sono visibili all'utente.

**Funzioni**:

hanno argomenti di input ed di output ben distinti. La variabili locali sono private e non sono visibili all'esterno della funzione.

Le modifiche agli argomenti in input restano locali.

Chiamata di una funzione:

**>> [out1,out2,…] = *funzione*(in1,in2,…)**

oppure

**>> *funzione*(in1,in2,…)**

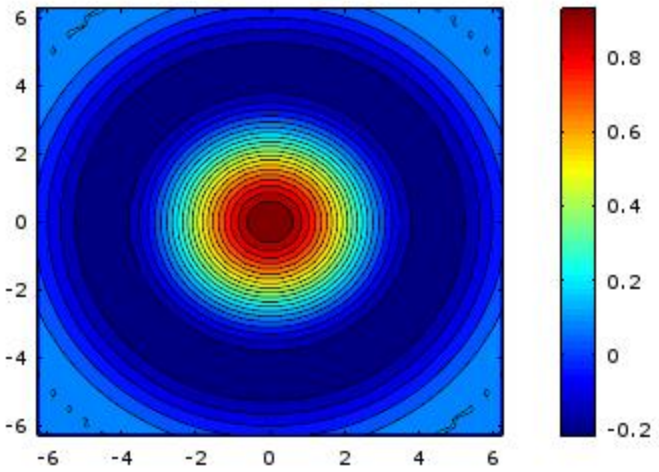
I parametri in ingresso sono in1, in2, etc. mentre out1, out2, etc. sono i parametri in uscita. Le parentesi quadre sono opzionali se è presente un solo parametro in uscita.

Definizione di una funzione in un file. Attenzione il nome della funzione è comunque quello del file in cui risiede!

```
function [o1,o2] = funzione(i1,i2)
…
o1 = …
o2 = …
```

# Funzioni

```
>> help sinc2d
SINC2D crea una mappa della sinc in
Le coordinate X ed Y vanno da -2*pi a
In in ingresso il numero di punti lungo l
>> w = sinc2d(30,50)
w =
  scalar structure containing the fields:
    x: 1x30 matrix
    y: 1x50 matrix
    z: 50x30 matrix
>> contourf(w.x, w.y, w.z, 20)
>> colorbar
```




```
function w = sinc2d(nx, ny)
%SINC2D crea una mappa della sinc in 2d.
%Le coordinate X ed Y vanno da -2*pi a 2*pi.
%In in ingresso il numero di punti

w.x = linspace(-2*pi, 2*pi, nx);
w.y = linspace(-2*pi, 2*pi, ny);

[XX, YY] = meshgrid(w.x, w.y);
r = hypot(XX, YY);

w.z = sin(r)./r;
w.z(isnan(w.z)) = 1;
```

Best Practice:

Avere una struttura come argomento di uscita di una funzione. Dare dei nomi esplicativi ai campi della struttura.

# Funzioni anonime e handle a funzioni

```
>> fh = @sin
fh  = @sin
>> fh(pi/2)
ans = 1
```

Con il carattere **@** prima di una funzione già definita si crea un **handle** a quella funzione che si può memorizzare in una variabile.

Lo handle si può usare per richiamare la funzione.

```
>> sh = @(x)x.^2+7
sh = @(x) x .^ 2 + 7
>> sh(5)
ans = 32
>> quad(@(x)x.^2, 0, 1)
ans =  0.33333
>> quad(@sin, 0, pi/2)
ans =  1.00000
```

Lo stesso carattere **@** può essere usato per creare una funzione anonima.

L'uso tipico è quando si passa una funzione ad un'altra come argomento.

**quad**(f, x1, x2) integrazione numerica tra x1 e x2 nelle versioni Matlab più recenti sostituita da integral.

# Funzioni in parallelo sui cell array

la funzione **cellfun** applica una funzione ad uno o più array di celle in parallelo.

```
>> a = {1,2,3};
>> b = {5,6,7};
>> d = cellfun(@(x,y) x+y, a, b)
d =    6    8   10
```

Tipicamente, quando i contenuti delle celle non sono uniformi come nelle stringhe.

```
>> cellfun(@length,{'Ciao','Come','stai?'})
ans =   4   4   5
```

Esiste anche la funzione **arrayfun** che opera su array. Da non usare per funzioni che normalmente operano sui vettori.

```
>> aa = [1,2,3];
>> bb = [4,5,6];
>> dd = arrayfun(@(x,y) x+y, aa, bb)
dd =   5   7   9
```

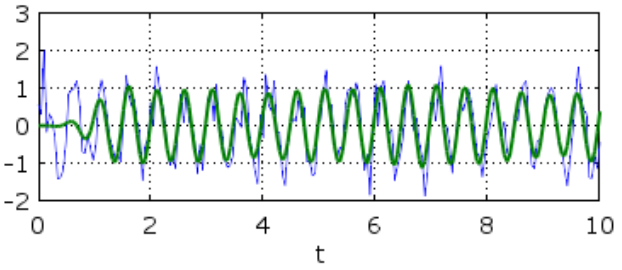
# Analisi di Fourier

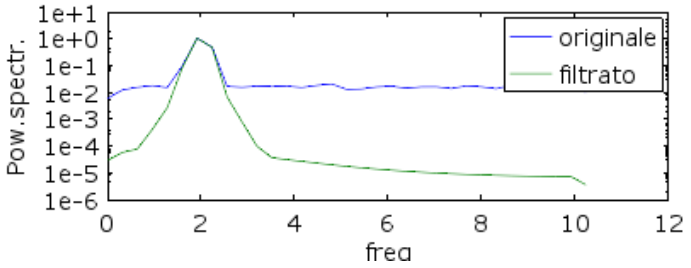
```
>> ft = fft([1,2,3,4])
ft = 10+0i   -2+2i   -2+0i   -2-2i
>> ifft(ft)
ans =   1   2   3   4
```

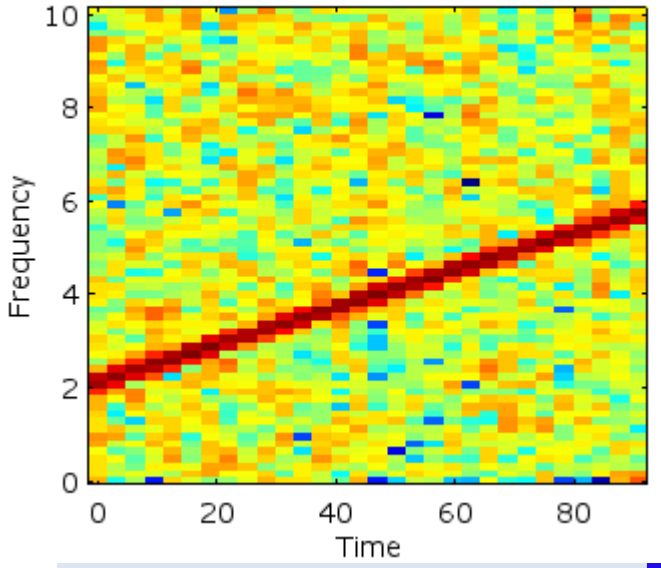
```
>> t = linspace(0,100,N);
>> dt = mean(diff(t))
>> y = sin(2*2*pi*t) + 0.4*randn(1,N);
>> b = fir1(40,[1.5,2.5]*2*dt);
>> yf = filter(b,1,y);
```

```
>> [py, fry] = pwelch(y,[],[],[],1/dt,'db');
>> [pyf, fryf] = pwelch(yf,[],[],[],1/dt,'db');
>> semilogy(fry, py, fryf, pyf)
>> legend('originale','filtrato  ')
```

```
>> ys = sin(2*pi*t.*(2 + 0.02*t)) + 0.4*randn(1,N);
>> specgram(ys, 128, 1/dt)
```



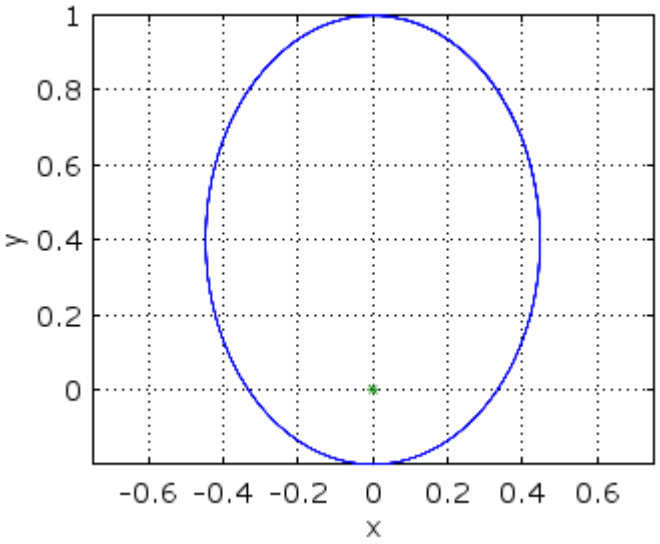



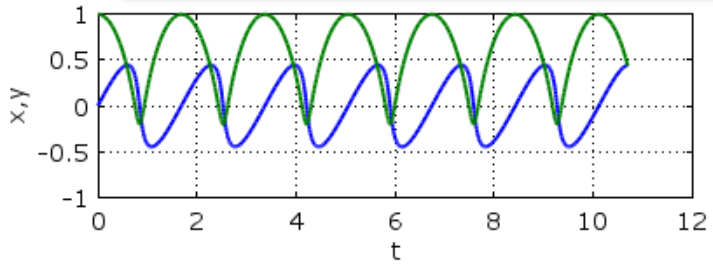



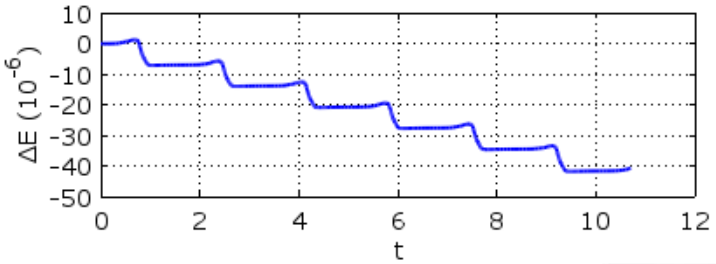
# Integrazione di un ODE

```
>> mass = 3.0
>> [t, y] = ode45(@(t,y) keplerfun(t,y,mass),  [0,1.7],[1;0;0;1]);
>> plot(y(:,3),y(:,4), 0,0,'o')
>> T = 0.5*sum(y(:,1:2).^2,2);
>> U =  - 3.0./hypot(y(:,3),y(:,4));
>> E = T + U;
```

```
function yp = keplerfun(t, y, massa)
  vel = y(1:2);
  pos = y(3:4);
  r = hypot(pos(1),pos(2));
  acc = - massa ./ r.^3 .* pos;
  yp = vertcat(acc, vel);
```



# Grafica con gli handle

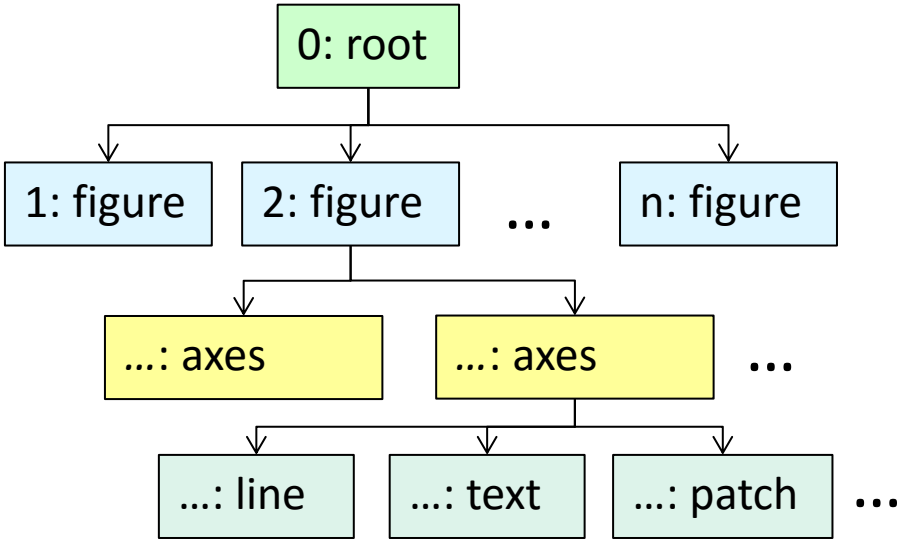
Gli handle sono dei numeri a cui sono associati degli oggetti grafici (*questo è cambiato nelle ultime versioni di Matlab® dove gli handle sono dei riferimenti a degli oggetti veri e propri*).

- **root** il suo handle è 0.
- **figure** normalmente il loro handle è un numero intero. Sono sotto **root**.
- Tutti gli altri handle sono numeri non interi.
- Ogni oggetto grafico ha delle proprietà, modificabili tramite l'handle.
- Più oggetti grafici si possono riunire in gruppi **hggroup**.

```
>> hl = plot([1,2],[3,4])
hl = -11.826
```

```
>> gcf → 1
>> gca → 16.505

>> get(hl,'xdata') → 1  2
>> get(hl,'ydata') → 3  4
```




| | |
|---|---|
| **gcf** | handle della figura corrente. |
| **gca** | handle dell'asse corrente. |
| **gco** | handle dell'oggetto corrente, l'ultimo su cui si è cliccato. |
| **gcbo** | handle dell'oggetto la cui callback si sta eseguendo. |
| **set** | imposta le proprietà dell'oggetto specificato dall'handle. |
| **get** | recupera le proprieta dell'oggetto specificato. |
| **hggroup** | crea n gruppo di oggeti grafici. |

# Classi nuovo stile, vecchio stile

Classi vecchi stile basate su directory che iniziavano per **@**.
Le classi nuovo stile definite tramite **classdef**.
La classe deve stare in un file che si chiama come la classe stessa (es. *testclass.m*).

```
>> a = testclass('Ed');
```

Chiamato il costuttore

```
>> a
a =
Sono: Ed
```

Chiamato il metodo **disp**

```
>> a.scrivi
Ciao Ed!
```

Chiamato il metodo **scrivi**

```
classdef testclass
  properties
    name;
  end

  methods
    function obj = testclass(a)
      obj.name = a;
    end

    function scrivi(obj)
      fprintf('Ciao %s!\n',obj.name);
    end

    function disp(obj)
      fprintf('Sono: %s',obj.name)
    end
  end
end
```

costruttore

chiamato automaticamente

# Esercizi per casa

- Soluzione di una ODE per un pendolo reale (forza proporzionale al sin(theta)):
  - Calcolare periodo del pendolo in funzione dell'ampiezza.
  - Mappa di Poincarè (plot della velocità vs la posizione) per diverse orbite ad energia crescente.
- Dato un segnale a + b*sin(omega*t) + noise
  - Eseguire un fit non lineare su a, b e omega.
  - Calcolare l'errore su a, b e omega in funzione dell'ampiezza del noise e confrontarlo con una serie di test numerici con ampiezza del noise crescente.