



# GNU Octave

MATLAB open source

---

Alessandro Zunino

18 Maggio 2018

Laboratorio di Calcolo e Multimedia (LCM)

# Introduzione

---

# Octave: cosa è?

Inizialmente ideato da John W. Eaton nel 1988 come strumento per l'ingegneria chimica, **GNU Octave** è diventato un linguaggio di alto livello per il calcolo numerico. Fa parte del progetto **Progetto GNU** ed è perciò un software libero nei termini della GNU General Public License. Si tratta di un linguaggio interpretato, non compilato, in cui le matrici sono il tipo di dato fondamentale ed è strutturato per avere un formalismo di tipo matematico.

È disponibile per **Windows, macOS e Linux**, oltre che per **Android**.

Vi sono alcune **differenze** fra Octave e Matlab. Le principali sono:

- I commenti possono essere preceduti sia dal carattere `#` che `%`.
- Sono supportati operatori ispirati dal C++ (`++`, `--`, `+=`, `*=`, `/=`).
- Elementi possono essere indicizzati senza definire nuove variabili (ad esempio `[1:10](3)`).
- Le stringhe possono essere comprese sia fra `"` che fra `'`.
- I blocchi possono essere terminati con specifiche istruzioni di controllo (per esempio `endif`, `endfor`, `endwhile`, ecc.).
- Le funzioni possono essere definite all'interno di script e dalla linea di comando.
- Esiste anche il ciclo `do-until` (simile al `do-while` in C).

Octave può essere usato senza e con GUI (Graphic User Interface) dalla versione 4.0.0. Per aprirlo senza interfaccia grafica digitare da terminale:

```
$ octave --no-gui
```

Per aprirlo con interfaccia grafica:

```
$ octave --force-gui
```

I comandi possono essere inseriti direttamente dalla linea di comando. Per muoversi fra le cartelle si usano gli stessi comandi di *bash* (*ls*, *cd*, *pwd*, ecc.).

## Workspace e pacchetti

Tutte le variabili vengono salvate in memoria, all'interno del cosiddetto *workspace*. Esso è visibile dalla GUI e contiene anche un breve descrizione dei dati salvati. Per pulirlo si usa il comando `clear all`.

Oltre alle librerie standard, si possono usare estensioni chiamate *packages*. Si possono installare digitando:

```
>> pkg install -forge <package>
```

Per vedere l'elenco completo dei pacchetti installati digitare:

```
>> pkg list
```

## Sintassi di base

---

L'operatore uguale = permette di definire una variabile, senza specificarne il tipo. In presenza del punto e virgola ; al termine del comando l'uscita a video è soppressa.

```
>> a=2
```

```
a = 2
```

```
>> a=2;
```

Il tipo di dato è implicito (viene automaticamente imposto dall'interprete). Il tipo fondamentale è la matrice, oltre a questo i principali sono scalari, intervalli e stringhe.

```
>> a=2;  
>> typeinfo(a)  
ans = scalar
```

L'elenco completo è fornito da `typeinfo()`.

# Vettori

Gli elementi di un vettore sono racchiusi fra parentesi quadre.  
Separando gli elementi con uno spazio o una virgola si ottiene una riga, con un punto e virgola una colonna.

```
>> v=[1 3 4 11]
```

```
v =
```

```
    1    3    4   11
```

```
>> w=[1; 3; 4; 11]
```

```
w =
```

```
    1  
    3  
    4  
   11
```

# Matrici

Una matrice è definita analogamente.

```
>> m=[1 4 5 1; 1 7 0 4; 8 3 3 9]
```

```
m =
```

```
1 4 5 1
1 7 0 4
8 3 3 9
```

```
>> size(m)
```

```
ans =
```

```
3 4
```

# Trasposizione

Si può trasporre una matrice utilizzando l'operatore `'`. Con esso si può passare da un vettore riga ad uno colonna e viceversa.

```
>> n=m'
```

```
n =
```

```
1    1    8
4    7    3
5    0    3
1    4    9
```

```
>> size(n)
```

```
ans =
```

```
4    3
```

# Intervalli

Sequenze ordinate i cui elementi sono separati da un passo costante, possono essere definite dall'operatore `:`, dove il passo è 1 a meno che non specificato diversamente inserendolo fra due operatori `:`.

```
>> 1:10
```

```
ans =
```

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

```
>> 1.35:0.1:1.8
```

```
ans =
```

```
    1.3500    1.4500    1.5500    1.6500    1.7500
```

Gli intervalli sono considerabili a tutti gli effetti dei vettori.

Se non si vuole specificare il passo, ma la lunghezza del vettore che si vuole avere, è possibile usare il comando `linspace`.

```
>> x=linspace(1,20,5)
```

```
x =
```

```
    1.0000    5.7500   10.5000   15.2500   20.0000
```

```
>> length(x)
```

```
ans = 5
```

# Indicizzazione

Una volta creata una matrice (un vettore è una matrice avente una dimensione uguale a 1) è possibile fare riferimento ad un suo elemento tramite l'operatore ( )

```
m =
```

```
1  4  5  1
1  7  0  4
8  3  3  9
```

```
>> m(1,2)
ans = 4
```

**NOTA BENE:** il conteggio inizia da uno, non da zero.

# Indicizzazione

Una qualunque sottomatrice può essere ottenuta tramite l'indicizzazione. Il primo e ultimo elemento lungo una dimensione sono separati dall'operatore `:`. Se si vogliono ottenere tutti gli elementi lungo una dimensione si inserisce `:` senza estremo superiore ed inferiore.

```
>> m(2:3,1:3)
```

```
ans =
```

```
1    7    0
```

```
8    3    3
```

```
>> m(1,:)
```

```
ans =
```

```
1    4    5    1
```

Due o più matrici possono essere unite fino a formare un'unica matrice.

```
>> a=[1 2];  
>> b=[4 5];  
>> c=[3;6];  
>> [[a;b] c]  
ans =
```

```
1 2 3  
4 5 6
```

Le stringhe vengono inserite fra due apici ' e si comportano come un vettore di caratteri.

```
>> s='hello'  
s = hello  
>> typeinfo(s)  
ans = sq_string  
>> s(2)  
ans = e  
>> [s ' world']  
ans = hello world
```

# Operazioni di base

---

## Operazioni di base

Gli operatori  $+$ ,  $-$ ,  $*$ ,  $/$  eseguono le quattro operazioni fra scalari.  
L'operatore  $^$  esegue l'elevamento a potenza.

```
>> a=2;  
>> b=3;  
>> a+b  
ans = 5  
>> a*b  
ans = 6  
>> a^2  
ans = 4
```

## Operazioni di base

Gli operatori  $+$ ,  $-$ ,  $*$ ,  $/$ ,  $^$  eseguono le note operazioni di algebra matriciale nel caso in cui gli operandi siano vettori o matrici. In particolare  $m*n$  rappresenta il prodotto riga per colonna fra  $m$  e  $n$  e  $m/n$  rappresenta il prodotto riga per colonna fra  $m$  e l'inversa di  $n$ .

```
>> m=[1 2; 7 2; 3 5];  
>> n=[1 0 1 0; 0 3 7 5];  
>> m*n  
ans =
```

```
1     6    15    10  
7     6    21    10  
3    15    38    25
```

Gli operatori `.*`, `./`, `.^` eseguono invece le operazioni elemento per elemento (le dimensioni delle due matrici devono quindi essere identiche).

```
>> m=[1 2; 7 2];
```

```
>> n=[1 3; 5 7];
```

```
>> m.*n
```

```
ans =
```

```
    1    6  
   35   14
```

## Operazioni di base

Il supporto ai numeri complessi è nativo, sia come scalari che come elementi di una matrice.

```
>> z = 2 + 3i;
```

```
>> conj(z)
```

```
ans = 2 - 3i
```

```
>> real(z)
```

```
ans = 2
```

```
>> imag(z)
```

```
ans = 3
```

```
>> abs(z)
```

```
ans = 3.6056
```

```
>> arg(z)
```

```
ans = 0.98279
```

# Scripting

---

Operazioni consecutive possono essere raccolte in un unico file di testo (usualmente con estensione `.m`) attraverso l'editor integrato nella GUI di Octave. In uno script è possibile eseguire **cicli** (`for`, `while`, ecc.) e usare altre istruzioni di controllo (`if`, `switch`, ecc.)

```
% test.m
clear all; % pulisce il workspace
close all; % chiude tutte le finestre
```

```
x=[1 5 11 98 1 23 13 2];
avg=0;
for k=1:length(x)
    avg=avg+x(k);
end
avg=avg/length(x)
```

**NOTA BENE:** questo sopra riportato è un esempio di **pessima** programmazione. Bisogna sempre preferire la vettorizzazione ai cicli.

È possibile estendere le funzionalità di Octave scrivendo funzioni aggiuntive: devono essere contenute in un file avente lo stesso nome della funzione e il primo comando del file deve essere `function`. Lavorando nella cartella in cui è situato il file, si potrà usare la funzione chiamandola con il nome con cui è stata definita.

```
% average.m
function y = average(x)
if ~isvector(x)
    error('Input must be a vector')
end
y = sum(x)/length(x);
end
```

Per eseguire uno script è sufficiente richiamarla dalla linea di comando con `source` (specificando il percorso relativo) o chiamandola con il suo nome (senza estensione) se il file sorgente è situato nella stessa cartella di lavoro.

```
>> source test.m
avg = 19.250
>> test
avg = 19.250
>> x=rand(10,1);
>> average(x)
ans = 0.46091
```

# Funzioni e grafici

---

Molte **funzioni** matematiche, sia basilari (logaritmi, esponenziali, funzioni trigonometriche, ecc.) che speciali (funzioni di Airy, di Bessel, ecc.) sono nativamente supportate.

Si possono facilmente visualizzare grafici a partire da un vettore per la variabile indipendente e uno per la funzione da disegnare. Il comando `plot` permette di creare il grafico.

```
x=0:0.1:10; % vettore variabile indipendente
y=sin(x); % vettore funzione
plot(x,y)
grid on; % attiva la griglia
xlabel('x'); % testo sul grafico
ylabel('sin(x)');
title('Sine plot');
```

## Plot 3D

Per creare grafici 3D è prima necessario creare una griglia (una matrice) che descrive il piano delle variabili indipendenti tramite il comando `meshgrid`. Dopodichè è possibile visualizzare il grafico col comando `mesh` o `surf`.

```
x=-10:0.1:10; % definizione degli assi
y=-10:0.1:10;
[xx,yy]=meshgrid(x,y); % definizione del piano
rr=sqrt(xx.^2+yy.^2);
zz=sin(rr)./rr; % definizione della funzione
mesh(xx,yy,zz)
xlabel('x');
ylabel('y');
zlabel('z');
title('Sombrero plot');
```

Per generare un istogramma a partire da un vettore di numeri è sufficiente usare il comando `hist` il quale prende in ingresso il vettore stesso e il numero di bin con cui costruire l'istogramma. Se in ingresso si pone una matrice, l'istogramma avrà tante colonne quante sono quelle della matrice, ognuna di un colore distinto.

```
r=randn(100000,1); % centomila numeri casuali con
    distribuzione gaussiana
[counts,centers] = hist(r,100); % counts è il
    numero di dati per bin, centers è il centro dei
    bin
bar(centers,counts,1) % genera l'istogramma, con
    larghezza relativa 1
```

# Input/Output

---

## Scrittura su file

Per l' `input/output` esistono numerosi comandi, ognuno con molte opzioni. Per salvare una matrice in un file di testo si usa il comando `save` dove l'opzione `-ascii` fa sì che venga salvata così come è, senza intestazioni.

```
pkg load communications;
x=0:0.01:10;
y=1+3.1*x-0.02*x.^3;
yn=awgn(y,10); % add white gaussian noise, snr=10.
%yn=y+randn(1,length(y));
plot(x,y);
hold all;
plot(x,yn);
legend('smooth','with noise');
m=[x' yn'];
save -ascii 'polynomial.dat' m
```

## Salvataggio di immagini

Il salvataggio su file delle immagini può essere fatto direttamente dall'interfaccia grafica, oppure tramite comandi testuali. In particolare l'opzione `-dpdfatexstandalone` permette di generare separatamente un grafico vettoriale con tutte le porzioni di testo in un file scritto in linguaggio  $\text{\LaTeX}$ , il quale può essere direttamente incluso in un documento.

```
h=figure;
plot(x,y)
text(2, 5, ['$\displaystyle\leftarrow y=1+3.1x-0.02
x^3$']);
set(h, 'visible', 'off');
print(h, 'plot_prova.pdf', '-dpdfatexstandalone');
set(h, 'visible', 'on');
system('pdflatex plot_prova');
open('plot_prova.pdf')
```

Per la lettura da file è sufficiente usare il comando `load` specificando il percorso relativo del file a partire dalla cartella di lavoro. Il file caricato è automaticamente una matrice.

```
data=load('polynomial.dat'); % il file è caricato
    nella matrice data
x=data(:,1); % prima colonna
y=data(:,2); % seconda colonna
plot(x,y);
```

# Propagazione degli errori

---

## Propagazione degli errori

Noti i valori e le incertezze di due grandezze fisiche e la relazione matematica che le lega ad una terza, è possibile calcolare il valore e l'incertezza anche su quest'ultima utilizzando un generatore di numeri casuali con distribuzione gaussiana.

```
% fun_test.m
function y = fun_test(a,b)
    y = 2 +0.47*a.^2+4*exp(1-b);
end
% propagazione.m
a=5;
err_a=0.7;
b=1.3;
err_b=0.1;
```

## Propagazione degli errori

```
r=randn(100000,1); % numeri casuali gaussiani
da=a+r*err_a; % distribuzione di a
db=b+r*err_b; % distribuzione di b
c=fun_test(a,b) % valore di c
dc=fun_test(da,db); % distribuzione di c
err_c=std(dc) % deviazione standard di c
```

```
figure(1)
subplot(1,3,1)
hist(da,100);
subplot(1,3,2)
hist(db,100);
subplot(1,3,3)
hist(dc,100);
```

# Fit polinomiale e non-lineare

---

## Fit polinomiale

Per **fit polinomiale** si usa la funzione `polyfit`, che restituisce i parametri e della statistica sul fit. La funzione `polyval` restituisce il polinomio valutato coi coefficienti ottenuti dal fit e il vettore degli errori.

```
data=load('polynomial.dat');
x=data(:,1);
y=data(:,2);
[p,s] = polyfit(x,y,3); % si deve specificare l'
    ordine del polinomio richiesto
[f,delta] = polyval(p,x,s);
plot(x,y, '.');
hold all;
plot(x,f, 'LineWidth',3, 'r');
legend('data', 'fit');
```

Per evidenziare l'intervallo di errore si possono inserire nel grafico anche gli estremi superiori ed inferiori di confidenza.

```
y1=f-delta;  
y2=f+delta;  
plot(x,f,'LineWidth',3,'r');  
hold all  
plot(x,y1,'--','Color','r','LineWidth',2);  
plot(x,y2,'--','Color','r','LineWidth',2);  
grid on;
```

Alternativamente si può colorare l'area compresa fra le barre di errore.

```
y1=f-delta;  
y2=f+delta;  
X=[x',fliplr(x')];  
Y=[y2',fliplr(y1')];  
fill(X,Y,'c')  
hold all;  
plot(x,f,'LineWidth',3,'b');
```

## Fit non-lineare

Per il **fit non-lineare** è necessario usare il pacchetto *optim*, che contiene più funzioni con questo scopo, ad esempio `leasqr`. Questa funzione restituisce anche statistica sulla bontà del fit, fra cui il coefficiente di determinazione  $R^2$ , la matrice di covarianza dei parametri e il vettore dei residui.

```
pkg load communications;  
x=0:0.01:10;  
y=3+0.1*log(2+x);  
yn=awgn(y,50);  
plot(x,yn);  
legend('smooth','with noise');  
m=[x' yn'];  
save -ascii 'log.dat' m
```

```
pkg load optim;
data=load('log.dat');
x=data(:,1);
y=data(:,2);
f=@(x,p) p(1)+p(2)*log(p(3)+x); % function handle
pin=[0.1; 0.1; 0.1]; % valori iniziali di p
[fy, p, cvg, iter, corp, covp, covr, stdresid, Z,
    r2] = leasqr(x,y,pin,f);
plot(x,y);
hold all;
plot(x,fy);
legend('data','fit');
```

# Trasformata di Fourier

---

È possibile calcolare la **trasformata discreta di Fourier** di una sequenza discreta (cioè un vettore) tramite la funzione `fft` (Fast Fourier Transform). Il vettore risultante sarà una sequenza simmetrica rispetto alla frequenza di Nyquist (metà della frequenza di campionamento). Se si desidera averla simmetrica rispetto all'origine è necessario applicare successivamente la funzione `fftshift`, la cui operazione inversa è `ifftshift`. La trasformata inversa è ottenuta tramite il comando `ifft`.

```
dt=0.001; % intervallo di campionamento
t=dt:dt:2; % asse temporale
y=sin(2*pi*3*t)+sin(2*pi*100*t)+sin(2*pi*362*t);
fs=1/dt; % frequenza di campionamento
df=fs/length(t);
ff=(0:(length(t)-1))*df; % asse delle frequenze
fy=fft(y); % fast fourier transform
figure(1)
subplot(1,2,1)
grid on;
plot(t,y);
subplot(1,2,2)
plot(ff,abs(fy)); % fy è complesso
grid on;
```

# Suoni e immagini

---

Un'immagine monocromatica può essere rappresentata da una matrice, dove ogni elemento è un pixel (il cui valore è compreso fra 0 e 255 se la profondità è di 8 bit). Un'immagine in RGB è rappresentato da un array di 3 matrici. Il comando `imshow` permette di visualizzare l'immagine, `imagesc` visualizza invece l'immagine a colori scalati.

```
m=[0 0 0 0 0;0 1 0 1 0; 0 0 0 0 0;0 0 0 0 0; 1 0 0
    0 1; 0 1 1 1 0; 0 0 0 0 0];
c = zeros(7,5,3); % array tridimensionale
c(:,:,1)=m; % matrice R
c(:,:,2)=ones(7,5); % matrice G
imshow(c); % visualizza l'immagine
imwrite(m, 'smile.png', 'png') % salva l'immagine
```

Un suono è un'onda acustica. Essa può quindi essere rappresentata da una combinazione lineare di sinusoidi. Una singola nota può essere descritta da una senoide ad una frequenza precisa, il cui suono può essere ascoltato tramite il comando `soundsc`, nota la frequenza di campionamento.

```
dt=0.0001; % secondi
fs=1/dt; % hertz
t=dt:dt:3;
f0=261.6; % frequenza del do
note=f0*2.^((0:11)/12); % scala temperata
y=0.8*sin(2*pi*note(1)*t);
soundsc(y,fs); % riproduce il suono
audiowrite('sound.wav',y,fs); % salva il file audio
```

Note conclusive

---

# Eseguibili

Gli script di octave possono essere resi **eseguibili** sui sistemi Unix.  
Per farlo è necessario inserire la riga

```
#! octave-interpretter-name -q -f
```

all'inizio del file di testo. Ad esempio, lo script `octavesh.m`:

```
#! /usr/bin/octave -qf
```

```
x=zeros(nargin,1);  
for i = 1:nargin  
    x(i)=str2num(argv(){i});  
end  
mean(x)
```

restituisce la media dei numeri inseriti da terminale.

Dopo essere stato reso eseguibile

```
$ chmod +x octavesh.m
```

può essere eseguito da terminale col comando

```
$ ./octavesh.m 1 2 3 4  
ans = 2.5000
```

Octave è scritto interamente in C++, perciò le sue **librerie** possono essere utilizzate direttamente in un codice C++ per coloro che preferissero un linguaggio compilato di più basso livello.