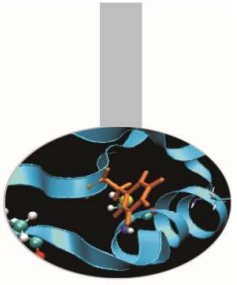
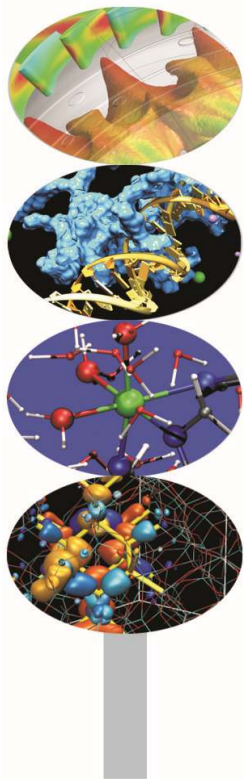
CINECA SCAI
SuperComputing Applications and Innovation

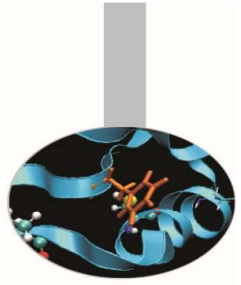


Matplotlib

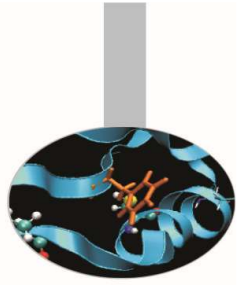


CINECA

Indice



Matplotlib: Modulo Pylab



"Matplotlib tries to make easy things easy and hard things possible"

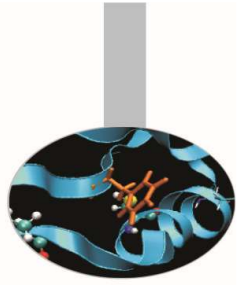
John Hunting

Uno strumento per la grafica bidimensionale è fornito dalla libreria Matplotlib.

La libreria Matplotlib nasce per emulare in ambiente Python i comandi grafici di Matlab.

È sviluppata interamente in Python e utilizza il modulo Numpy per la rappresentazione di grandi array.

Matplotlib: Modulo Pylab



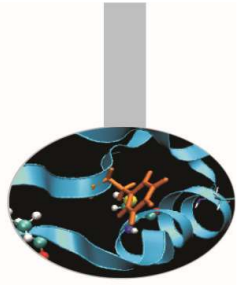
Matplotlib è divisa in tre parti:

- Pylab interface: set di funzioni fornite dal modulo Pylab.
- Matplotlib API
- Backend: grafici per l'output su file e visuali per l'output su interfacce grafiche.

Contiene diverse funzioni per il calcolo scientifico.

È possibile utilizzare la sintassi LaTex per aggiungere formule nei grafici.

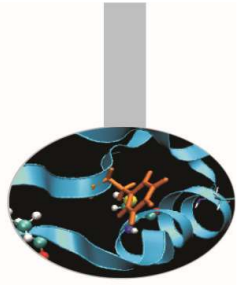
Matplolib: Modulo Pylab



Quali sono i vantaggi nell'utilizzare matplotlib?

- Usa Python: MATLAB manca di molte proprietà necessarie a renderlo un linguaggio general purpose
- E' open source
- E' cross-platform: Linux, Windows, Mac OS e Sun Solaris
- E' customizzabile ed estendibile
- Ha un'ottima resa grafica
- Possibilità di generare postscript per includere i grafici in documenti TeX
- Embeddable in una GUI per lo sviluppo di applicazioni
- Ha una sintassi semplice e leggibile

Introduzione a Pylab

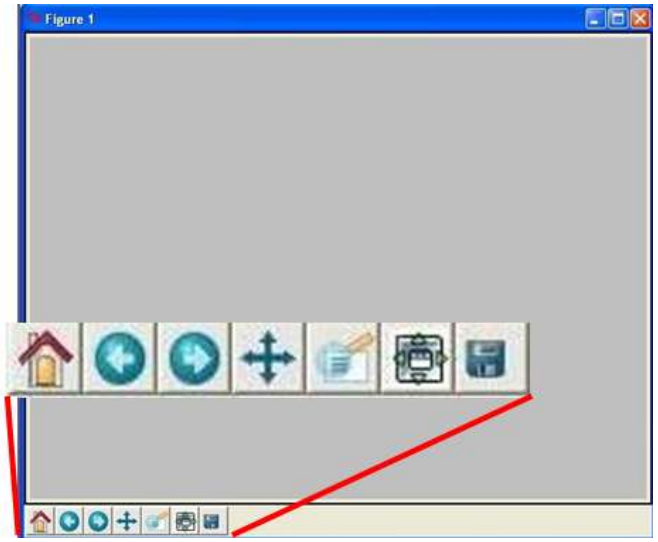


L'interfaccia Pylab costituisce il modo più semplice per lavorare con Matplotlib.

Le funzioni sono molto simili all'ambiente Matlab.

Esempio

```
>>>from pylab import *
>>>figure()
>>>show()
```

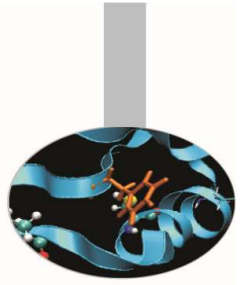


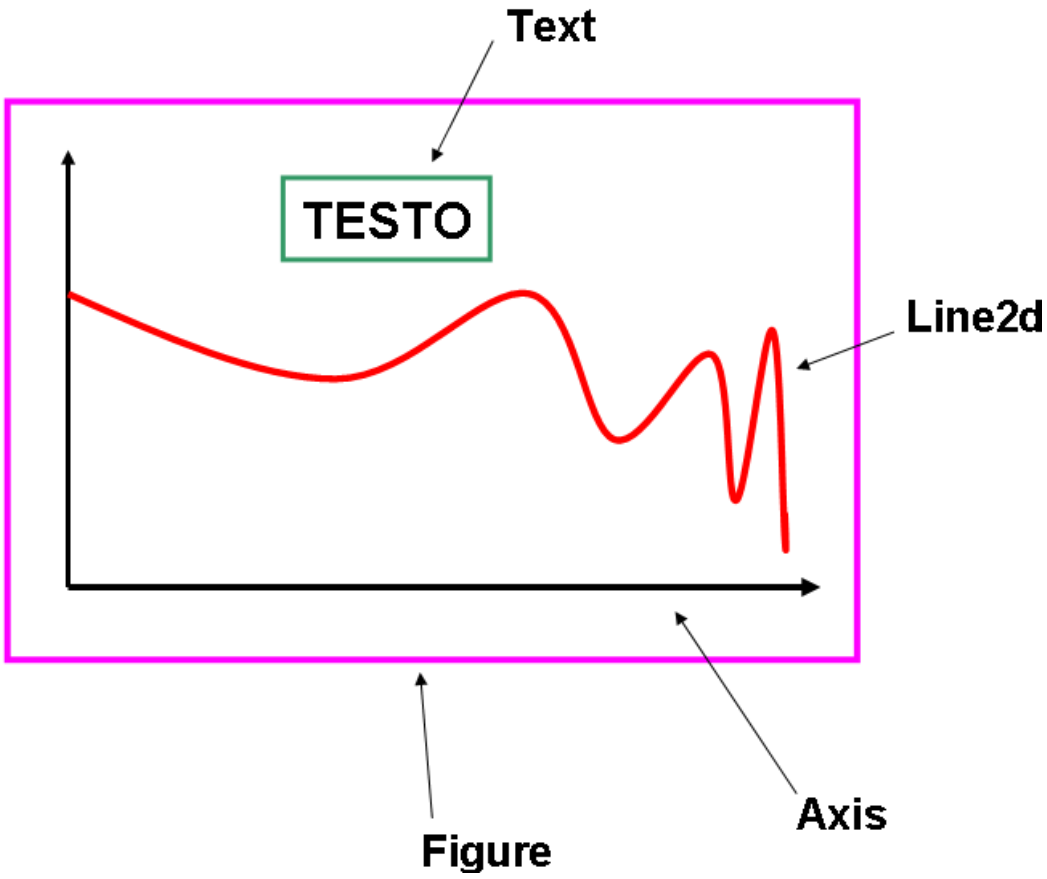
La funzione *figure()* istanzia un oggetto figura.

La funzione *close(n)* chiude la finestra *n*

La funzione *show()* visualizza tutte le figure create

Introduzione a Pylab





Le principali entità su cui lavorare sono:

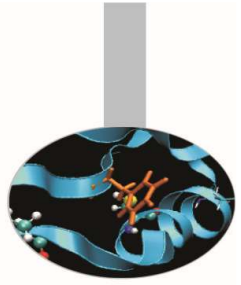
Figure l'oggetto figure ha attributi propri (risoluzione,dimensioni,).

Line2d le linee2d possiedono diversi marcatori propri, etc.

Text è possibile modificare e gestire testo (plain o math)

Axis per la gestione degli assi

Matplotlib



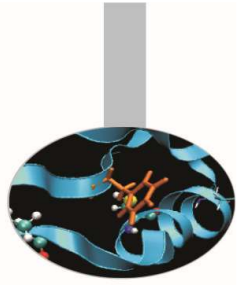
Matplotlib è disegnata per la programmazione object oriented: si possono definire oggetti per colours, lines, axes, etc.

Si può adottare anche un approccio funzionale: i plot possono essere generati usando funzioni, in una interfaccia Matlab-like.

L'approccio object-oriented è generalmente preferito per plot non-interattivi (i.e., scripting).

La pylab interface è utile per lavorare interattivamente e disegnare.

Matplotlib

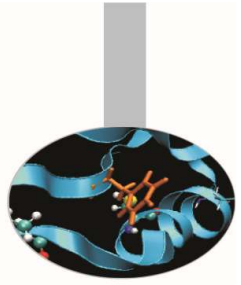


2 modi per usare Matplotlib:

- Object-oriented way: Il modo *Pythonico* di lavorare con Matplotlib. Il modulo pyplot fornisce un'interfaccia alla libreria matplotlib.

- pylab: Un modulo che unisce Matplotlib e NumPy in un ambiente simile a MATLAB. Assi e figure sono create automaticamente dalla funzione di disegno.

Matplolib API



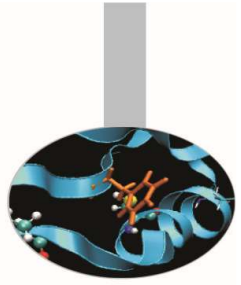
L'approccio OO rende tutto più esplicito e consente la customizzazione dei grafici

Esempio primo.py

```
import numpy as np
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_subplot(111)
t=np.arange(0,5,0.05)
f=2*np.pi*np.sin(2*np.pi*t)
ax.plot(t,f)
ax.set_title('Primo grafico')
ax.grid(True)
ax.set_xlabel('x')
ax.set_ylabel('y')
fig.show()
```

Matplotlib API: necessario per embedding in GUI

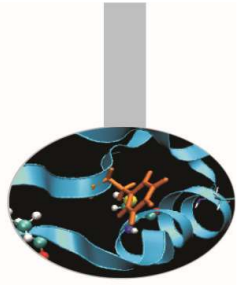
Matplotlib API



Esempio plots.py

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> x = np.arange(0, 10, 0.1)
>>> y = np.random.randn(len(x))
>>> fig = plt.figure()      # instance of the fig obj
>>> ax = fig.add_subplot(111) # instance of the axes   obj
>>> l, m = ax.plot(x, y, x, y**2) # returns a tuple of obj
>>> l.set_color('blue')
>>> m.set_color('red')
>>> t = ax.set_title('random numbers')
>>> plt.show()
```

pyplot vs pylab

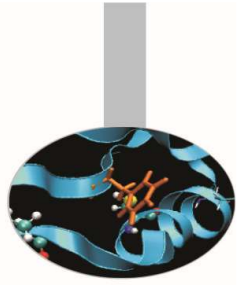


pylab

```
>> from pylab import *
>> t=arange(0,5,0.05)
>> f=2*pi*sin(2*pi*t)
>> plot(t,f)
>> grid()
>> xlabel('x')
>> ylabel('y')
>> title('Primo grafico')
>> show()
```

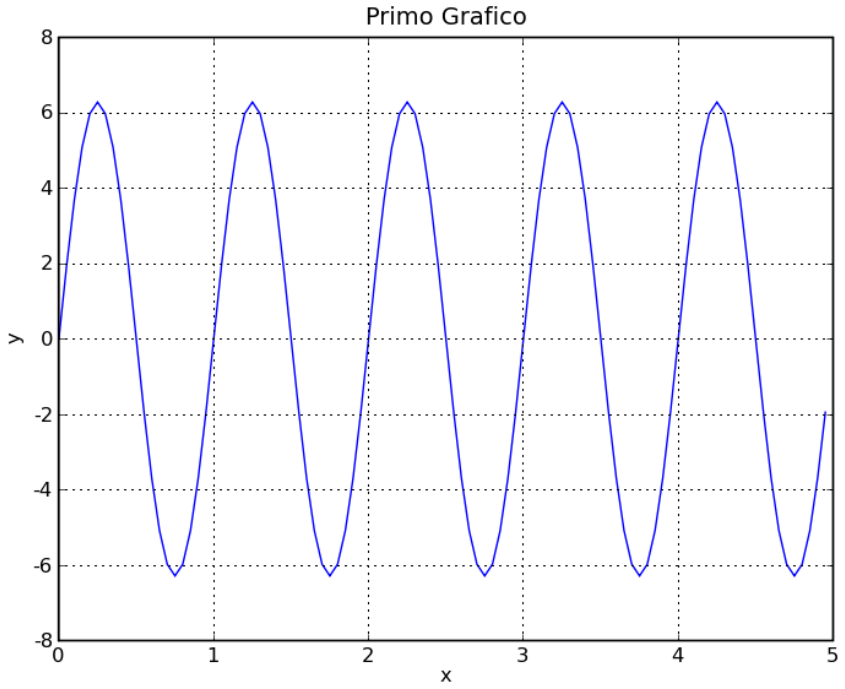
pylab mode: preferibile per interactive plotting

Comandi di base di pylab



Esempio onda.py

```
>>>from numpy import *
>>>from pylab import *
>>>t=arange(0,5,0.05)
>>>f=2*pi*sin(2*pi*t)
>>>plot(t,f)
>>>grid()
>>>xlabel('x')
>>>ylabel('y')
>>>title('Primo grafico')
>>>show()
```

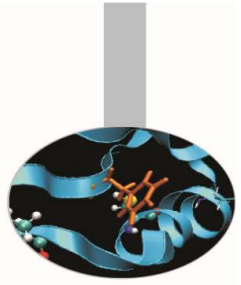


Il grafico viene visualizzato solo alla chiamata della funzione show().

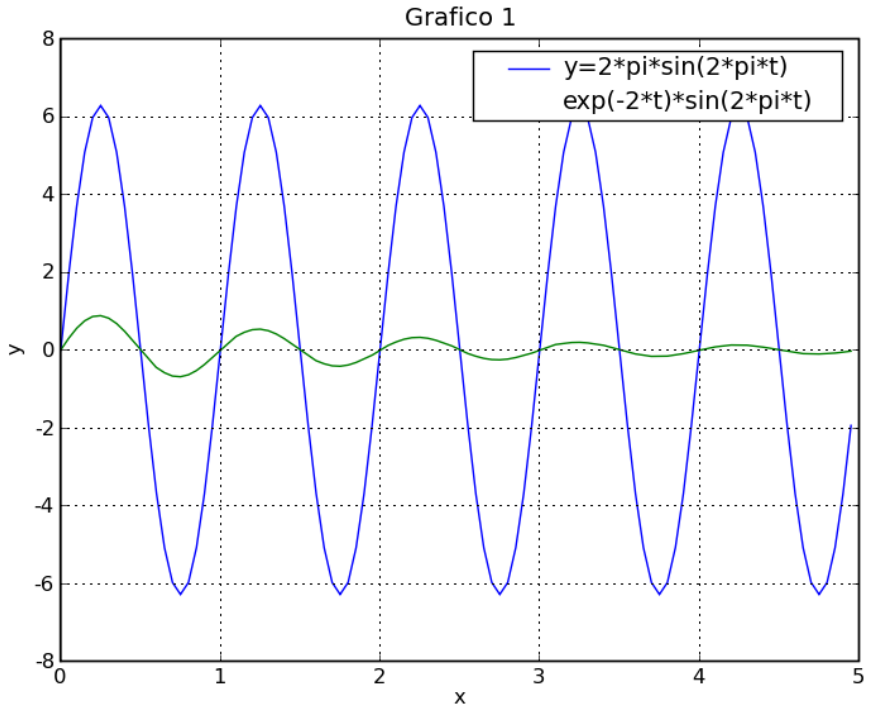
Per lavorare interattivamente è necessario impostare:

- mode interactive *matplotlib.rcParams['interactive']=True*
- il tipo di backend *matplotlib.rcParams['backend']='TkAgg'*

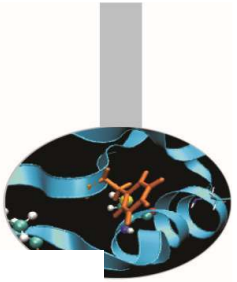
Comandi di base



```
>>>hold(True)
>>>f2=sin(2*pi*t)*exp(-2*t)
>>>plot(t,f2)
>>>legend((‘y=2*pi*sin(2*pi*x)’,‘sin(2*pi*x)*exp(-2*x)’))
```



Comandi di base

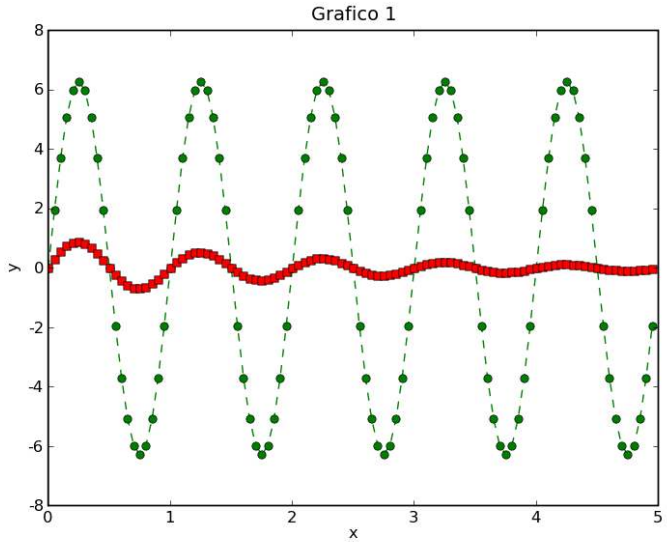


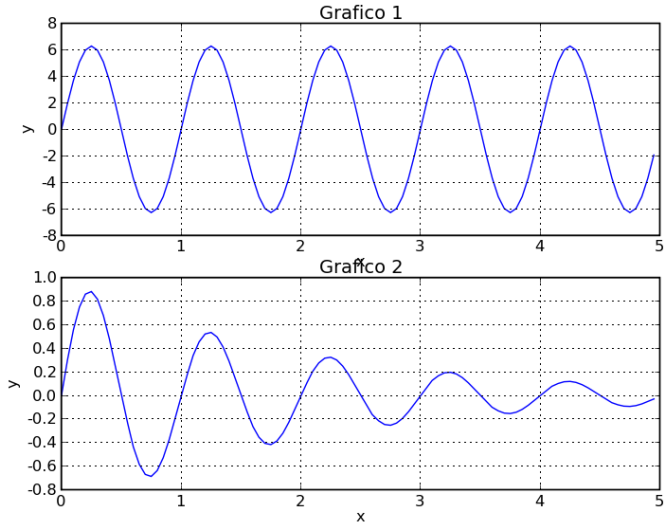
In alternativa :

```
>>>clf()
>>>plot(t,f,'g--o',t,f2,'r:s')
>>>xlabel('x')
>>>ylabel('y')
>>>title('Grafico 1')
```

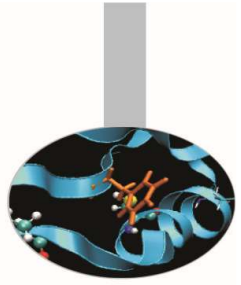
SUBPLOT

```
>>>subplot(211)
>>>plot(t,f)
>>>xlabel('x');ylabel('y') ; title('Grafico 1')
>>>subplot(212)
>>>plot(t,f2)
>>>xlabel('x');ylabel('y') ; title('Grafico 2')
```





Figure



E’ possibile gestire e creare un numero arbitrario di figure tramite il comando *figure().*

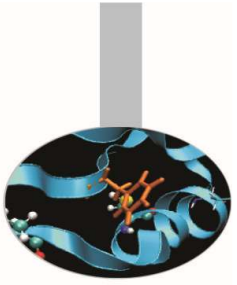
E’ possibile gestire i seguenti attributi della figura:

- *figsize:* dimensione in inches
- *facecolor:* colore di riempimento
- *edgecolor:* colore del bordo
- *dpi:* risoluzione
- *frameon:* per mantenere il background grigio alla figura.

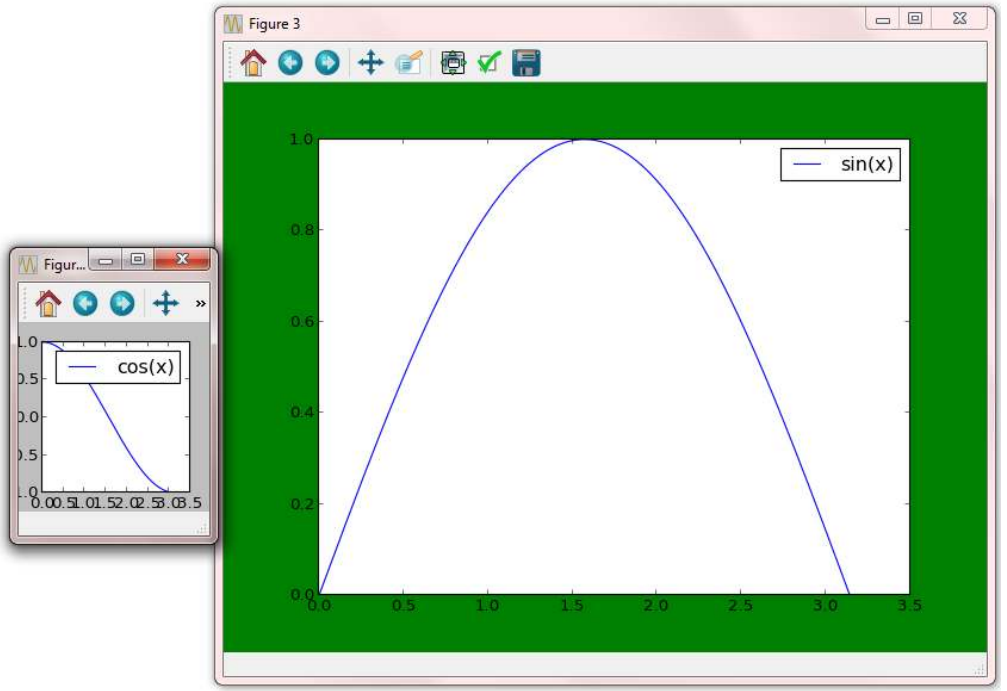
Per chiudere la figura si possono usare i comandi:

- *close(num)*
- *close(istance)*
- *close(‘all’)*

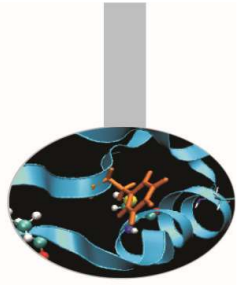
Figure



```
x=arange(0,pi,0.01)
y=sin(x)
y2=cos(x)
figure(facecolor='g')
plot(x,y,label='sin(x)')
legend()
figure(figsize=[3,3])
plot(x,y2,label='cos(x)')
legend()
close(1)
close('all')
```



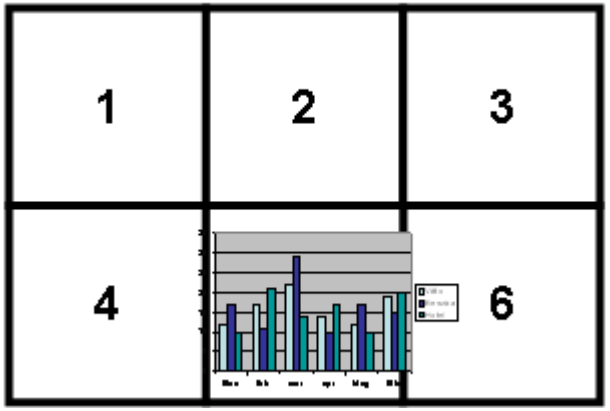
Plot e Subplot



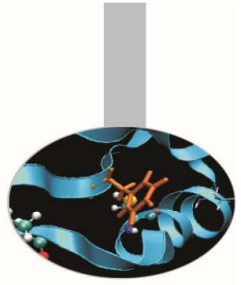
Il comando *plot(line2d , [properties line2d])* è un comando versatile che consente di creare grafici multilinea specificando lo stile.

Il comando *subplot(nrows,ncol,index)* permette di creare grafici multipli su una griglia con un numero specifico di righe e di colonne.

subplot(2,3,5)



Plot e Subplot

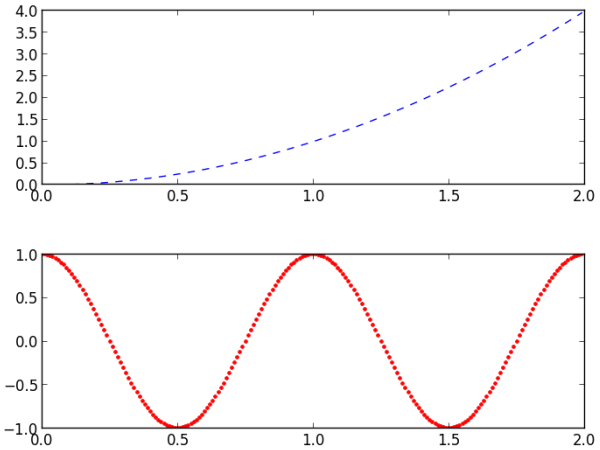


Creating subplot - pylab

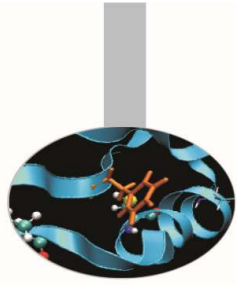
```
from pylab import *
x = arange (0, 2.0, 0.01)

subplot(2, 1, 1)
plot(x, x ** 2, 'b--')
subplot(2, 1, 2)
plot(x, cos(2*pi*x), 'r.')

subplots_adjust(hspace = 0.5)
show()
```

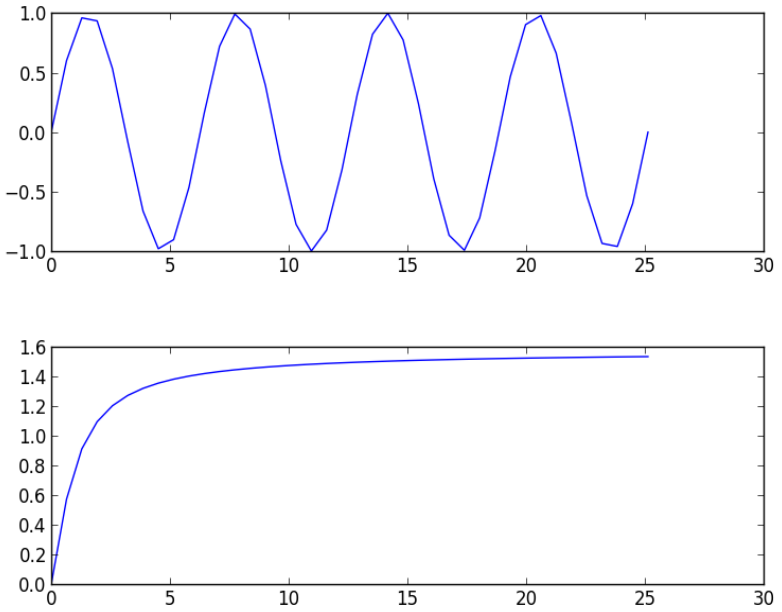


Plot e Subplot

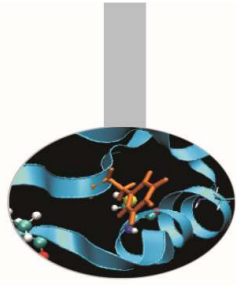


Creating subplot - OO

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 8*np.pi,
  num=40)
f=plt.figure()
ax=f.add_subplot(2,1,1)
ax.plot(x, np.sin(x))
ax2=f.add_subplot(2,1,2)
ax2.plot(x, np.arctan(x))
f.subplots_adjust(
  left=0.13, right=0.97,
  top=0.97, bottom=0.10,
  wspace=0.2, hspace=0.4)
plt.show()
```



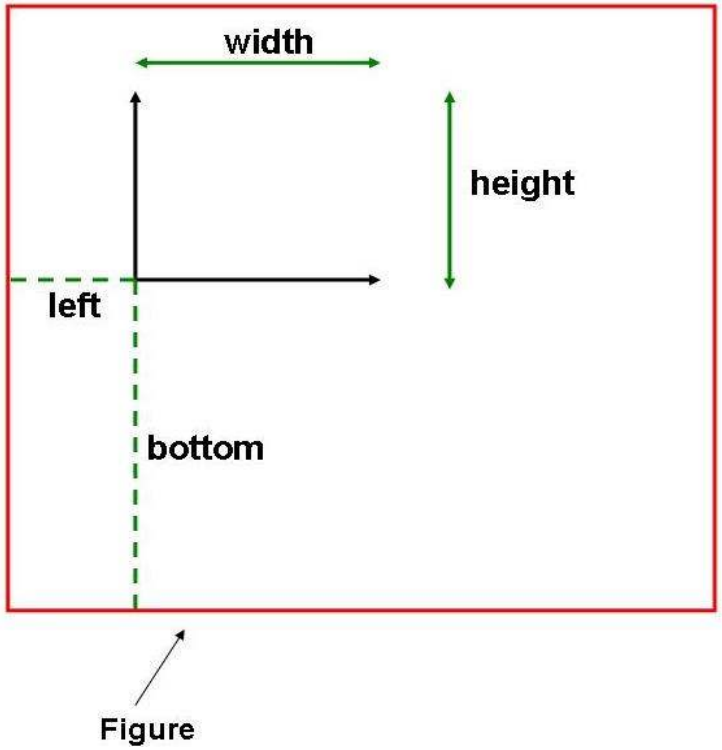
Axes



L'oggetto *axes()* permette la gestione degli assi e si comporta in maniera simile a subplot.

axes() equivale a subplot(111)

axes([left,bottom, width, height]) posiziona e dimensiona il grafico secondo la lista di parametri passati come argomento.



Alcuni metodi

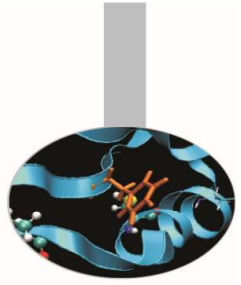
axis([*xmin,xmax,ymin,ymax*])

grid()

xticks(*location,label*)

legend([*list_lines*],[*list_label*], *loc,* [*text_prop*])

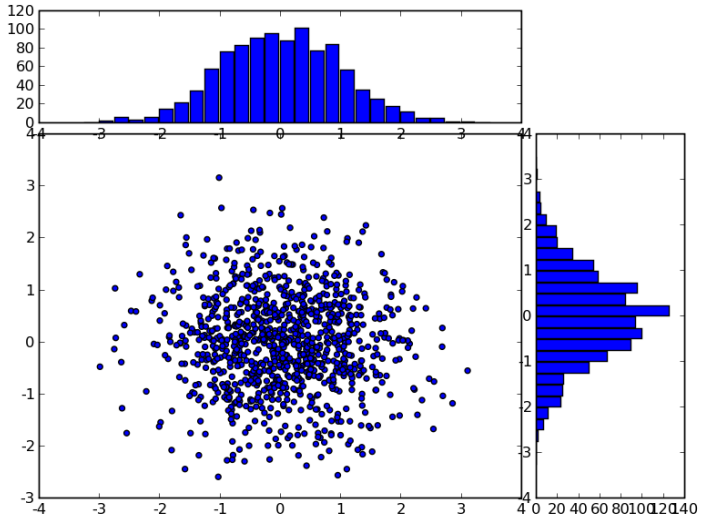
Axes



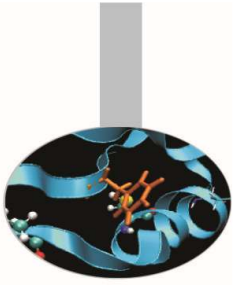
Esempio histoaxis.py

```
import numpy as np
import matplotlib.pyplot as plt
x = np.random.randn(1000)
y = np.random.randn(1000)
axscatter = plt.axes([0.1,0.1,0.65,0.65])
axhistx = plt.axes([0.1,0.77,0.65,0.2])
axhisty = plt.axes([0.77,0.1,0.2,0.65])

axscatter.scatter(x, y)
plt.draw()
binwidth = 0.25
xymax = max( [max(np.fabs(x)),
    max(np.fabs(y))] )
lim = ( int(xymax/binwidth) + 1) * binwidth
bins = np.arange(-lim, lim + binwidth, binwidth)
axhistx.hist(x, bins=bins)
plt.draw()
axhisty.hist(y, bins=bins, orientation='horizontal')
plt.draw()
plt.show()
```



Axes

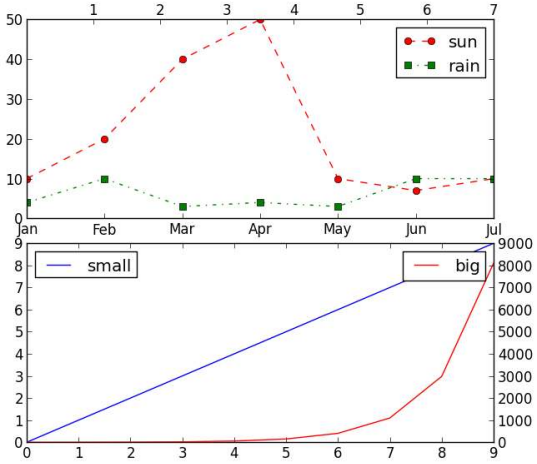


Esempio doppio.py

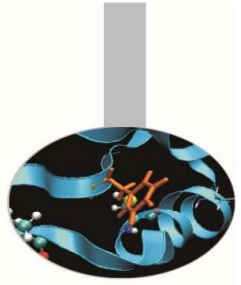
```
import numpy as np
from matplotlib import pyplot as plt
x=[1,2,3,4,5,6,7]
y=[10,20,40,50,10,7,10]
y2=[4,10,3,4,3,10,10]

f=plt.figure()
ax=f.add_axes([0.1,0.55,0.7,0.4])
l1,=ax.plot(x,y,'r--',marker='o')
l2,=ax.plot(x,y2,marker='s',color='green',linestyle='-.')
ax.set_xticks(x)
ax.set_xticklabels(['Jan','Feb','Mar','Apr','May','Jun',
'Jul'])
ax.legend([l1,l2],['sun','rain'])
bx=ax.twiny()
bx.set_xticks(x)

ax2=f.add_axes([0.1,0.1,0.7,0.4])
ax2.plot(np.arange(10),np.arange(10),label='small')
ax2.legend(loc=2)
by=ax2.twinx()
by.plot(np.arange(10),np.exp(np.arange(10)),'r',label='big')
by.legend()
plt.show()
```



Line2D Properties



L'oggetto linea ha diversi attributi: è possibile modificare le dimensioni, lo stile, il colore etc. La funzione:

*setp(*args, **kwargs)*

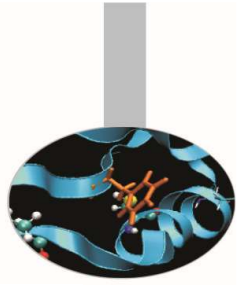
permette di cambiare tali attributi.

In alternativa è possibile modificare gli attributi tramite i metodi dell'oggetto line2D.

Tra gli attributi ricordiamo:

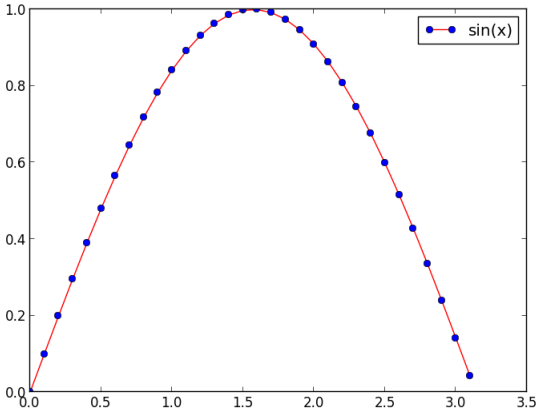
- *color* 'b', 'r', 'g', 'y', ' k', 'w' , 'c', 'm'
- *linewidth* float
- *linestyle* '', '-', '- -', ':', '.-'
- *label* stringa
- *marker* '.', 'o', 'D', '^', 's', '*', '+', 'h'
- *markersize float*
- *markerfacecolor* color

Line2D Properties

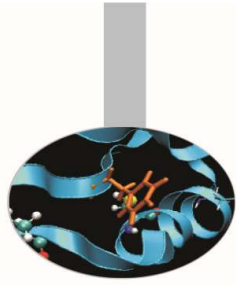


Creating subplot-- pylab

```
x=np.arange(0,np.pi,0.1)
plot(x,sin(x),marker='o',color='r',
   markerfacecolor='b',label='sin(x)')
legend()
```



Line2D Properties

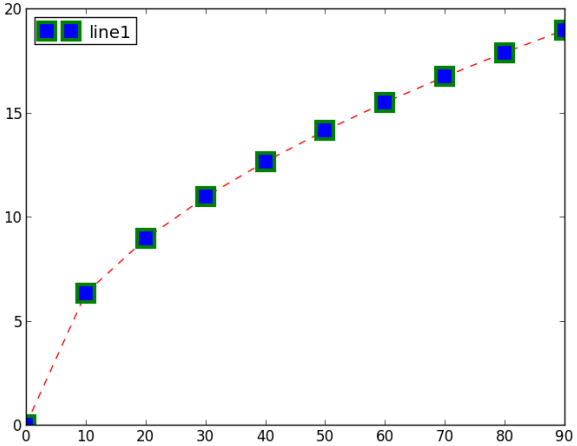


Creating subplot-- OO

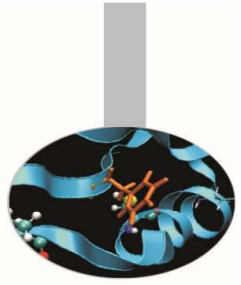
Esempio sale.py

```
import numpy as np
from matplotlib import pyplot as plt

x=np.arange(0,100,10)
y=2.0*np.sqrt(x)
f=plt.figure()
ax=f.add_subplot(111)
line,=ax.plot(x,y)
line.set_color('r')
line.set_linestyle('--')
line.set_marker('s')
plt.setp(line,markeredgecolor='green',
  markerfacecolor='b',markeredgewidth=3)
line.set_markersize(15)
plt.show()
```



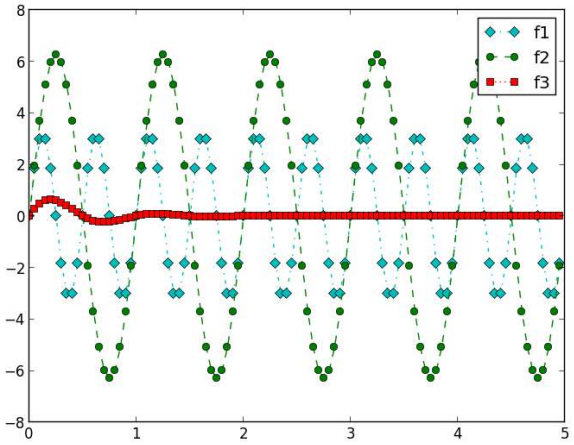
Line2D Properties



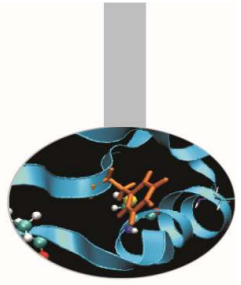
Creating Multi-line plot

Creating subplot-- pylab

```
t=np.arange(0,5,0.05)
f=2*np.pi*np.sin(2*np.pi*t)
f2=np.sin(2*np.pi*t)*np.exp(-2*t)
plt.plot(t,f,'g--o',t,f2,'r:s‘)
hold(True)
f3=2*np.pi*np.sin(2*pi*t)*np.cos(2*pi*t)
plt.plot(t,f3,'c-.D',label='f3')
plt.legend(('f1','f2‘,’f3’))
plt.show()
```



Line2D Properties

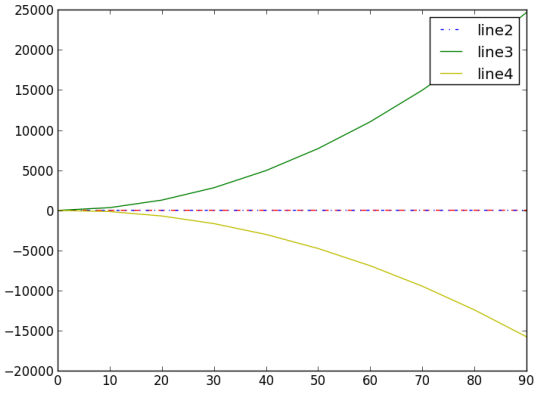


Creating Multi-line plot

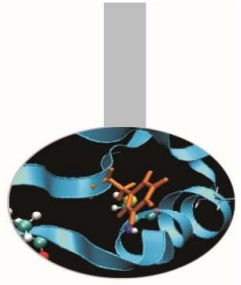
Creating subplot-- OO

Esempio crescedecresce.py

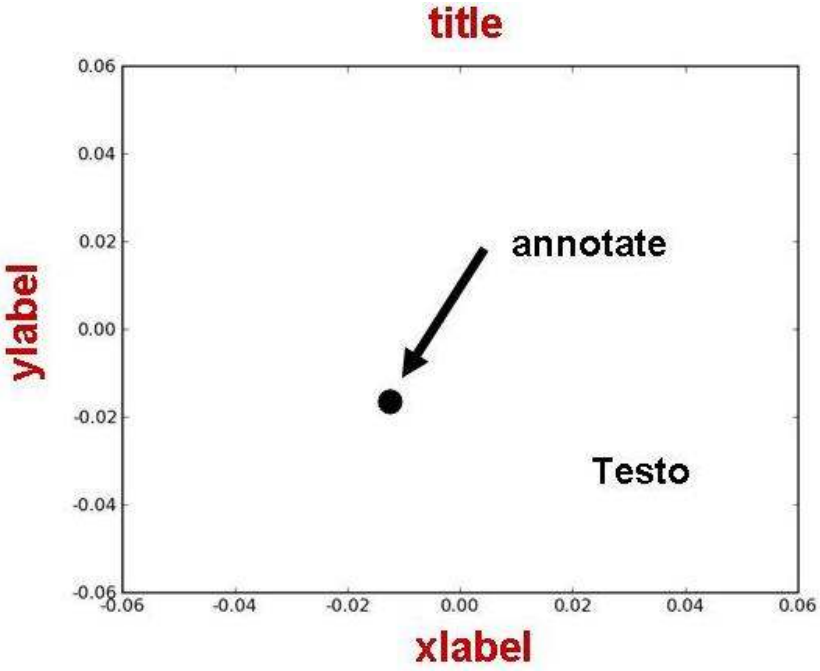
```
import numpy as np
from matplotlib import pyplot as plt
x=np.arange(0,100,10)
y1=2.0*np.sqrt(x);
y2=3.0*x**(1.0/3.0)
y3=4.0*x+3.0*x**2
y4=5.0*x-2.0*x**2
f=plt.figure()
ax=f.add_subplot(111)
line1,=ax.plot(x,y1,'r--')
line2,=ax.plot(x,y2,'b-.')
line3,line4=ax.plot(x,y3,x,y4)
line3.set_color('g')
line4.set_color('y')
ax.legend([line2,line3,line4],['line2','line3','line4'])
plt.show()
```



Gestione del testo



Pylab permette di gestire stringhe di testo all'interno di grafici.



xlabel (s, *args, **kwargs)

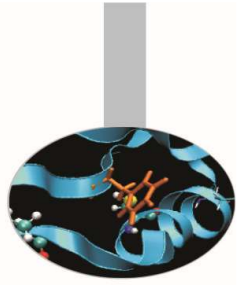
ylabel (s, *args, **kwargs)

title (s, *args, **kwargs)

annotate (s, xy, xytext=None, xycoords='data', textcoords='data', arrowprops=None,**props)

text (x, y, s, fontdict=None, **kwargs)

Gestione del testo

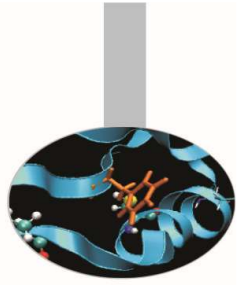


Inoltre Pylab è in grado di inglobare espressioni matematiche in espressioni di testo utilizzando la sintassi LaTex.

Per esempio la sintassi: xlabel(r'$y_i=2\pi \sin(2\pi x)$')
equivale a $y_i = 2\pi \sin(2\pi x)$

E' necessario inoltre imporre: rcParams(text.usetex)=True

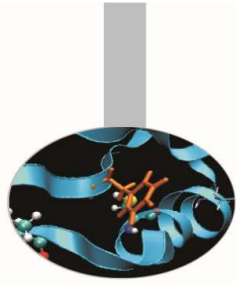
Text Properties



L'oggetto testo possiede le seguenti proprietà:

- *Fontsize:* xx-small, x-small, small, medium, large, x-large, xx-large
- *Fontstyle:* *normal, italic, oblique*
- *Color*
- *Rotation:* *degree , 'vertical', 'horizontal'*
- *Verticalalignment:* *'top', 'center', 'bottom'*
- *Horizontalalagnment:* *'left', 'center', right'*

Text Properties

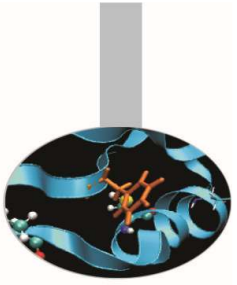


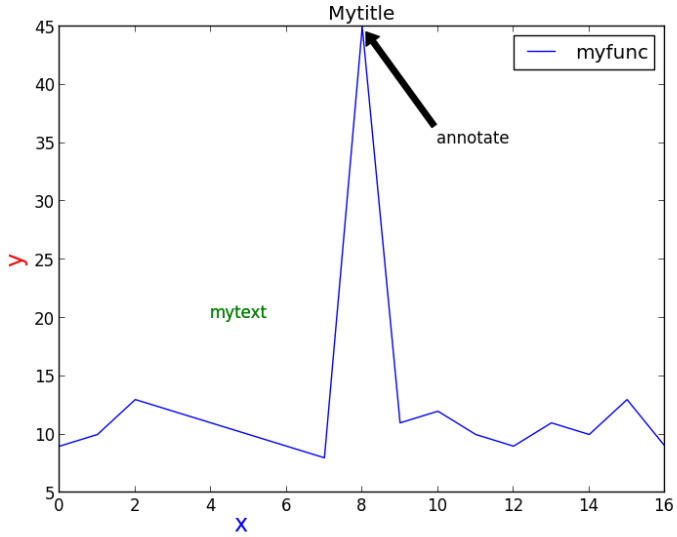
Gli attributi possono essere modificati in tre modi:

- Tramite *keyword arguments,* tramite la funzione *setp,* tramite i metodi dell'oggetto testo:

```
>>>plt.xlabel('ciao', color ='r', fontsize='large')      #keyword arguments
>>>l=plt.ylabel('asse y')
>>>plt.setp(l,rotation=45)                               #setp()
>>>l.set_color('r')                                      #object method
```

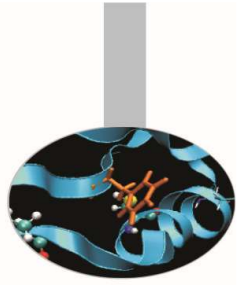
Text






```
x=[9,10,13,12,11,10,9,8,45,11,12,10,9,
11,10,13,9]
plt.plot(x,label='myfunc')
plt.legend()
plt.title('Mytitle')
plt.ylabel('y',fontsize='medium',color='r')
plt.xlabel('x',fontsize='x-large',color='b',position=(0.3,1))
plt.text(4,20,'mytext', color='g',fontsize='medium')
plt.annotate('annotate',xy=(8,45),xytext=(10,
35),arrowprops=dict(facecolor='black',shrink=0.05))
```

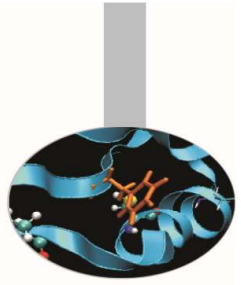
Images File



Ci sono diversi modi per usare matplotlib:

- Lavoro interattivo tramite shell python (meglio IPython).
- Attraverso degli script di processamento e generazione di file di immagini.
- Embedding in una graphical user interface, per consentire all'utente di interagire con i dati visualizzati.

Images File

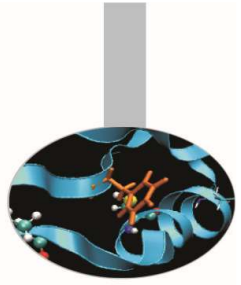


La visualizzazione del plot è time-consuming, specialmente per plot multipli e complessi.

I plot possono essere salvati senza essere visualizzati tramite la funzione **savefig()** :

```
x = np.arange(0,10,0.1)
plt.plot(x, x ** 2)
plt.savefig('C:/myplot.png')
```

Diagrammi a barre

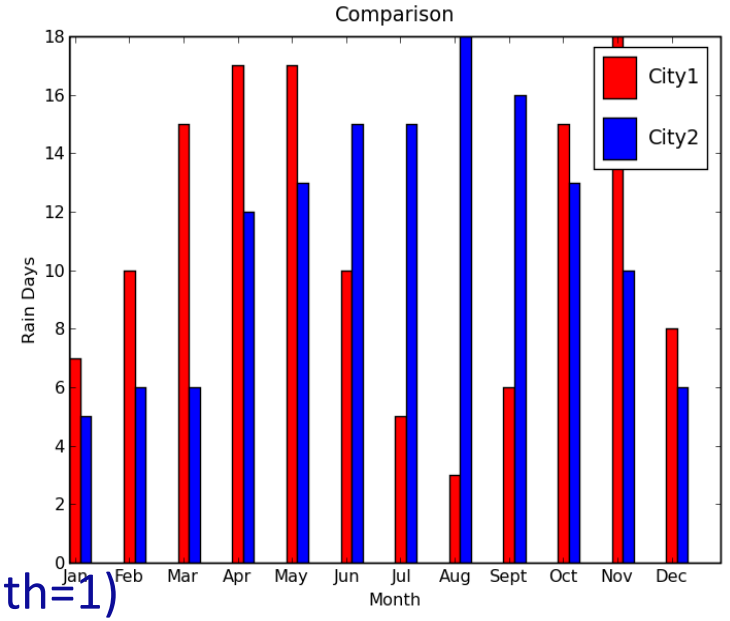


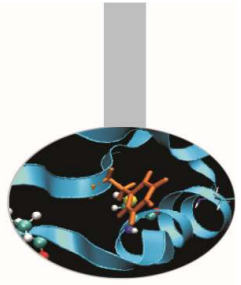
Come creare un diagramma a barre:

bar(left, height)

Esempio barre.py

```
n_day1=[7,10,15,17,17,10,5,3,6,15,18,8]
n_day2=[5,6,6,12,13,15,15,18,16,13,10,6]
m=['Jan','Feb','Mar','Apr','May','Jun',
   'Jul','Aug','Sept','Oct','Nov','Dec']
width=0.2;   i=np.arange(len(n_day1))
r1=plt.bar(i, n_day1,width, color='r',linewidth=1)
r2=plt.bar(i+width,n_day2,width,color='b',linewidth=1)
plt.xticks(i+width/2,m)
plt.xlabel('Month'); ylabel('Rain Days'); title('Comparison')
plt.legend((r1[0],r2[0]),('City1','City2'),loc=0,labelspacing=0.06)
plt.show()
```





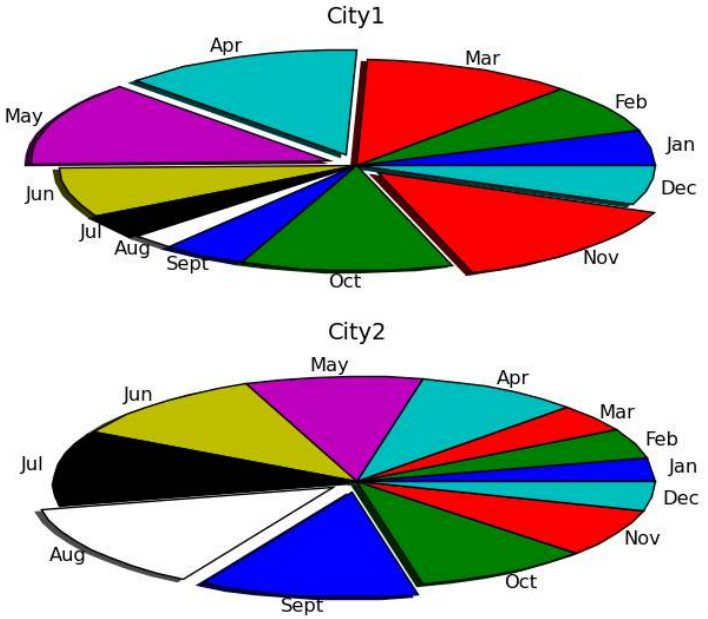
Torta

Oppure con gli stessi dati come creare una torta:

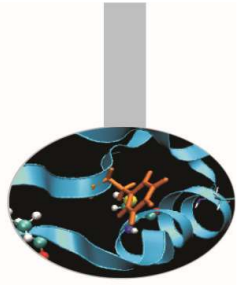
pie(x)

Esempio torta.py

```
plt.subplot(211)
plt.pie(n_day1,labels=m,
  explode=[0,0,0,0.1,0.1,0,0,0,0,0,0.1,0
  shadow=True)
plt.title('City1')
plt.subplot(212)
plt.pie(n_day2,labels=m,
  explode=[0,0,0,0,0,0,0,0.1,0.1,0,0,0],
  shadow=True)
plt.title('City2')
```



Meshgrid

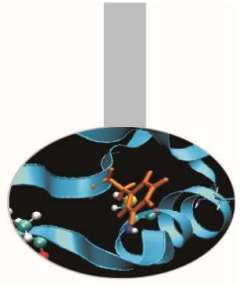


- Come costruire una griglia bidimensionale?
- Data una griglia (xi,yi) vogliamo calcolare per ciascun punto della griglia il valore della funzione f(xi,yi)

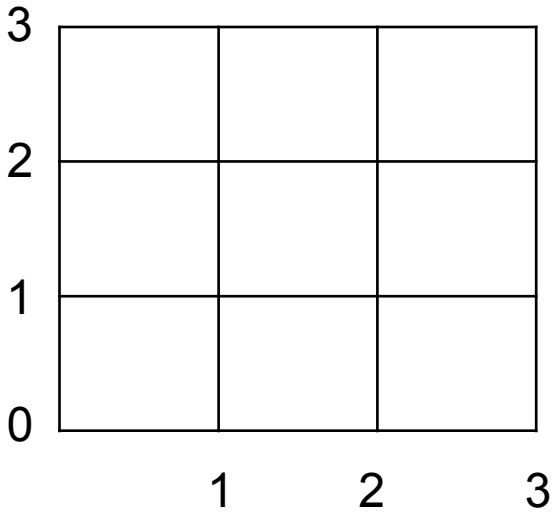
```
>>> x=np.arange(4)
>>> y=np.arange(4)
>>> def f(x,y):
        return x**2+y
>>> x
array([0, 1, 2, 3])
>>> y
array([0, 1, 2, 3])
>>> f(x,y)
array([ 0,  2,  6, 12])
```

WRONG!!

Meshgrid

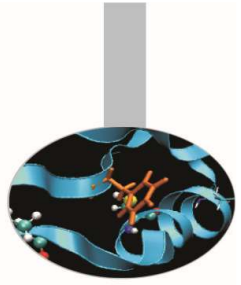


```
xx,yy=np.meshgrid(x,y)
>>> xx
array([[0, 1, 2, 3],
       [0, 1, 2, 3],
       [0, 1, 2, 3],
       [0, 1, 2, 3]])
>>> yy
array([[0, 0, 0, 0],
       [1, 1, 1, 1],
       [2, 2, 2, 2],
       [3, 3, 3, 3]])
>>> f(xx,yy)
array([[ 0,  1,  4,  9],
       [ 1,  2,  5, 10],
       [ 2,  3,  6, 11],
       [ 3,  4,  7, 12]])
```



OK!!

Contour plot



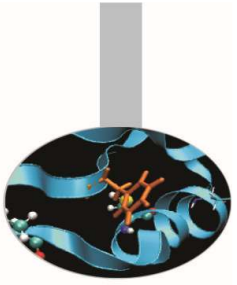
Esempio contour.py

contourf(*args, **kwargs)
contour(*args,**kwargs)
meshgrid(x,y)

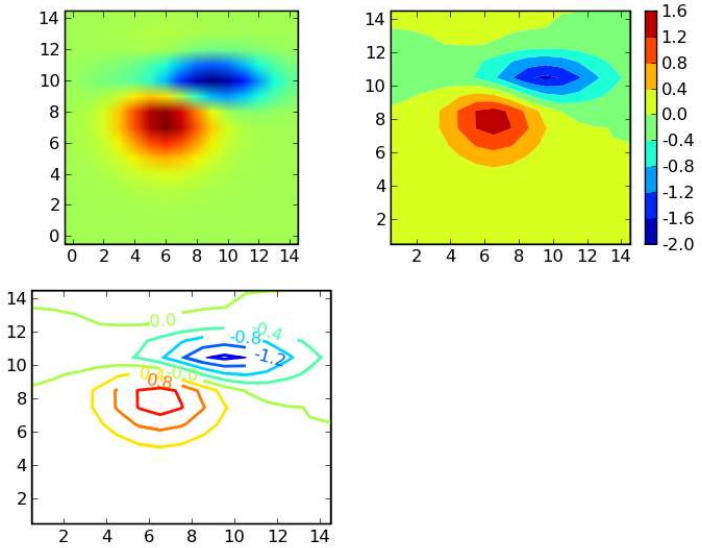
```
import numpy as np
from matplotlib import pyplot as plt
from matplotlib import mlab as ml

delta = 0.5
x = np.arange(-3.0, 4.001, delta)
y = np.arange(-4.0, 3.001, delta)
X, Y = np.meshgrid(x, y)
Z1 = ml.bivariate_normal(X, Y, 1.0, 1.0, 0.0, 0.0)
Z2 = ml.bivariate_normal(X, Y, 1.5, 0.5, 1, 1)
Z = (Z1 - Z2) * 10
levels = np.arange(-2.0, 1.601, 0.4)
```

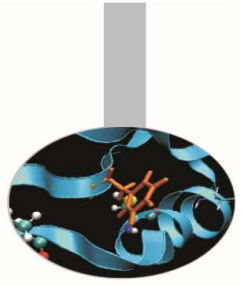
Contour plot



```
plt.figure(facecolor="w")
plt.subplot(221)
plt.imshow(Z,origin= 'lower')
plt.subplot(222,axisbg="w")
l= plt.contourf(Z,levels,origin='lower')
plt.colorbar(l)
plt.subplot(223,axisbg="w")
l= plt.contour(Z, levels,origin= 'lower',linewidths=2)
plt.clabel(l,inline=1, fmt='%1.1f',fonsize=14)
plt.show()
```



Output



Matplotlib supporta diversi backend grafici. Possiamo dividere la tipologia di backend in due categorie:

- User interface backend: per l'assemblaggio in GUI. In Python esistono diverse librerie per la costruzione di interfaccie grafiche tra cui Tkinter, PyQt, pygtk che vengono supportate da matplotlib.
- Hardcopy backend: per la stampa su file. Vengono supportati i seguenti formati *.jpg, *png, *svg, *pdf, *rgba.