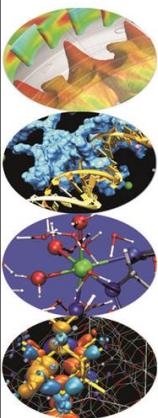


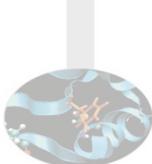
Introduzione a **matplotlib**: il modulo *pyplot*



Mario Rosati
CINECA – Roma
m.rosati@cineca.it



Cos'è *Matplotlib*



- *Matplotlib* è un modulo per la generazione di grafici 2D (in piccola parte anche 3D):
 - è completamente sviluppata in Python,
 - utilizza il modulo *Numpy* per la rappresentazione di grandi array
 - le sue funzionalità la rendono particolarmente adatta ad applicazioni di calcolo scientifico (tra le altre cose, è possibile utilizzare la sintassi *LaTex* per aggiungere formule sui grafici)
- *Matplotlib* è costituita da tre componenti:
 - La *matplotlib API* (interfaccia a oggetti), utilizzata direttamente per inserire funzionalità per la creazione di grafici nei propri script Python
 - Il modulo **pyplot** è un'interfaccia procedurale alla Matplotlib API ed è progettato per *emulare*, in ambiente Python, i più comuni comandi grafici di Matlab; dunque è pensato prevalentemente per un utilizzo interattivo
 - *Output back-end*: per l'output dei grafici su varie tipologie di GUI e su diversi formati di file

Un semplice grafico interattivo

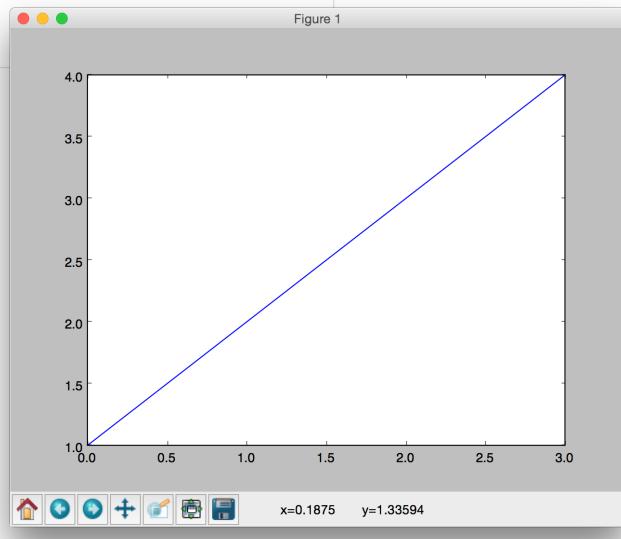
In [1]: `import matplotlib.pyplot as plt`

In [2]: `plt.plot([1,2,3,4])`

Out[2]: [`<matplotlib.lines.Line2D at 0x108b9f850>`]

In [3]: `plt.show()`

- Se alla funzione `plt.plot()` viene passata solo una lista di numeri o un array, *matplotlib* assume che si tratti della sequenza di valori y del grafico e li associa alla sequenza naturale di valori x: 0,1,2,3,..
- La funzione `plt.show()` visualizza la figura e blocca l'interprete Python fino a quando la *plotting window* sarà chiusa



3

La toolbar della *plotting window*



Mostra la *plotting area* originale



Undo/Redo della visualizzazione



Navigazione all'interno del plot



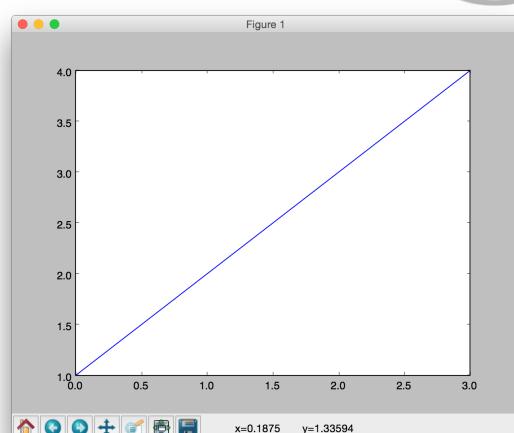
Zoom di una porzione rettangolare della *plotting area*



Personalizzazione dei *subplot*

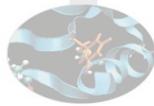


Salvataggio/Esportazione della figura



4

Il modulo *pyplot*: generalità



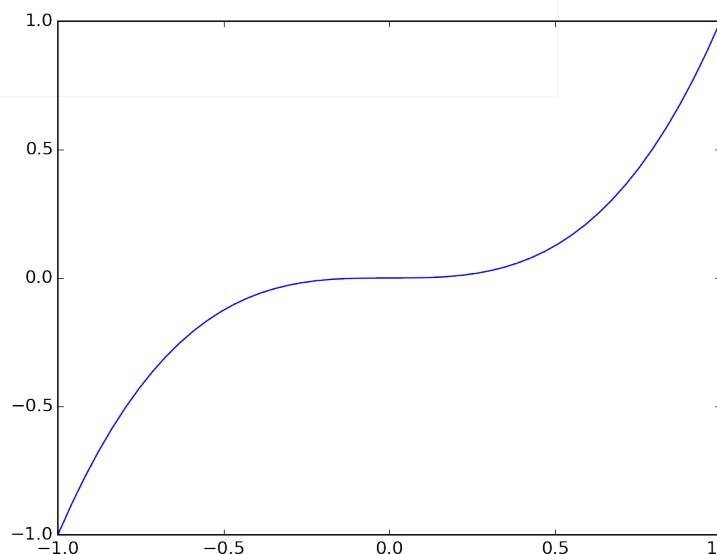
- Il modulo *pyplot* è una raccolta di funzioni *command style*, che consentono di utilizzare le funzionalità di *matplotlib* in interattivo, con una modalità simile a quanto possibile all'interno dell'ambiente MATLAB.
- Ogni funzione di *pyplot* agisce su una singola finestra di plot, detta **figura**; ad esempio:
 - crea una figura
 - crea una *plotting area* all'interno di una figura
 - disegna dei grafici in una *plotting area*
 - decora un plot con etichette o con altri elementi grafici
- *pyplot* è **stateful**, ovvero tiene traccia dello stato della figura corrente e della relativa *plotting area* e le sue funzioni di *plotting* agiscono sulla figura corrente

5

Il nostro primo grafico

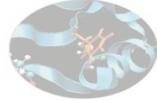


```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> x = np.linspace(-1.0,1.0,50,endpoint=True)
>>> y = x**3
>>> plt.plot(x,y)
>>> plt.show()
```



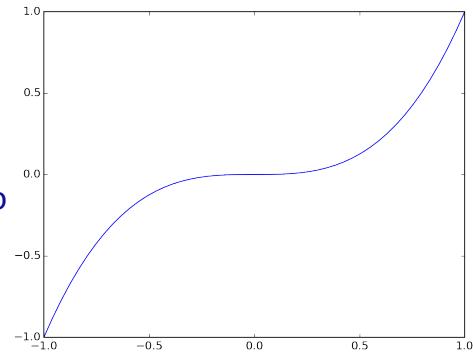
6

Il nostro primo grafico (II)



Osserviamo alcuni aspetti del comportamento di default di `plt.plot()`, che, in caso di necessità, possono essere modificati:

- la dimensione degli assi combacia perfettamente con il *range* dei dati del grafico
- i *tick mark* degli assi sono posti ogni 0.5 unità
- non c'è un titolo né ci sono etichette sugli assi
- non c'è una legenda
- il colore della linea del grafico è blu



7

Un po' di cosmesi: impostare la *dimensione degli assi*



La funzione `pyplot.axis([Xmin, Xmax, Ymin, Ymax])` consente di modificare gli estremi degli assi del grafico :

Dimensione degli assi del grafico

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> x = np.linspace(-1.0,1.0,50,endpoint=True)
>>> y = x**3
>>> plt.plot(x,y)

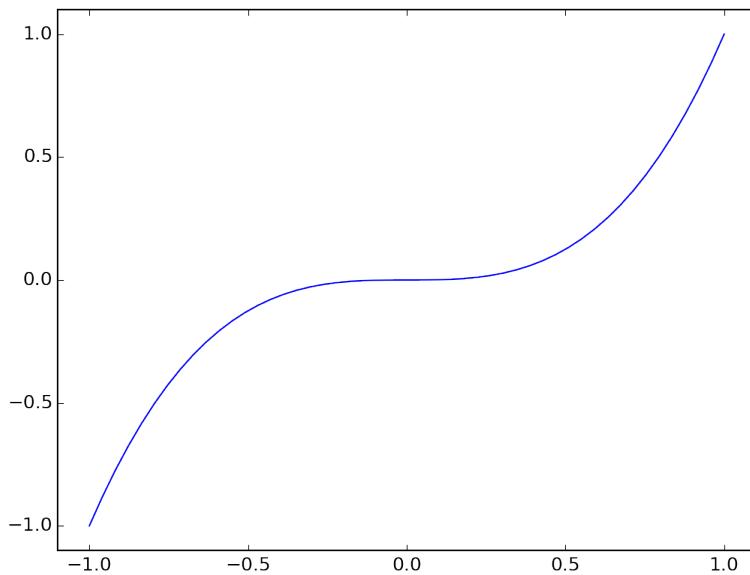
>>> plt.axis([-1.1, 1.1, -1.1, 1.1])

>>> plt.show()
```

8

Un po' di cosmesi: impostare la *dimensione degli assi* (II)

```
>>> plt.axis([-1.1, 1.1, -1.1, 1.1])
```



9

Un po' di cosmesi: impostare i *tick mark* degli assi

- La funzione `pyplot.xticks() / yticks()` permette di modificare il comportamento di default di `pyplot` per i *tick mark* dell'asse x/y
- Tali funzioni consentono d'impostare sia la posizione sia l'etichetta dei *tick mark* del relativo asse

Tick mark dell'asse x ogni 0.25 unità

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> x = np.linspace(-1.0,1.0,50,endpoint=True)
>>> y = x**3
>>> plt.plot(x,y)

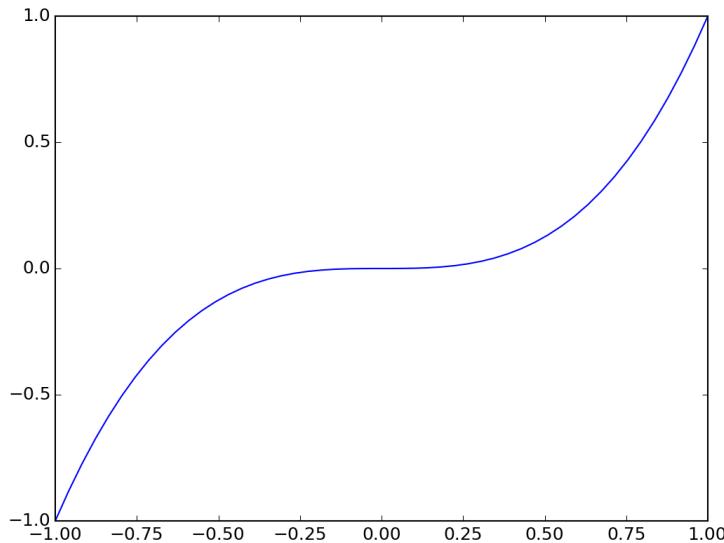
>>> # using Python's list comprehension syntax
>>> plt.xticks([0.25*k for k in range(-4,5)]) 

>>> plt.show()
```

10

Un po' di cosmesi: impostare i *tick mark* degli assi (II)

```
>>> plt.xticks([0.25*k for k in range(-4,5)])
```



11

Ancora sui *tick mark* degli assi

- Il primo argomento di `pyplot.xticks() / yticks()` può anche essere una lista non equi-spaziata
- A queste funzioni è possibile passare anche una di etichette da associare ai singoli *tick* del relativo asse

Tick mark dell'asse y non equi-spaziati e con etichetta *user defined*

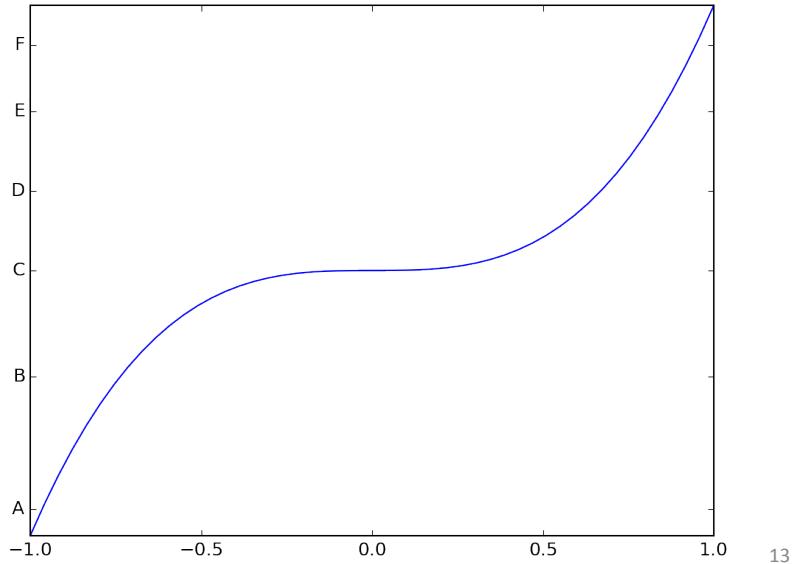
```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> x = np.linspace(-1.0,1.0,50,endpoint=True)
>>> y = x**3
>>> plt.plot(x,y)

>>> plt.yticks([-0.9,-0.4, 0.0, 0.3, 0.6, 0.85],
...             ['A','B','C','D','E','F'])
...
>>> plt.show()
```

12

Ancora sui *tick mark* degli assi (II)

```
>>> plt.yticks([-0.9, -0.4, 0.0, 0.3, 0.6, 0.85],  
...             ['A','B','C','D','E','F'])
```

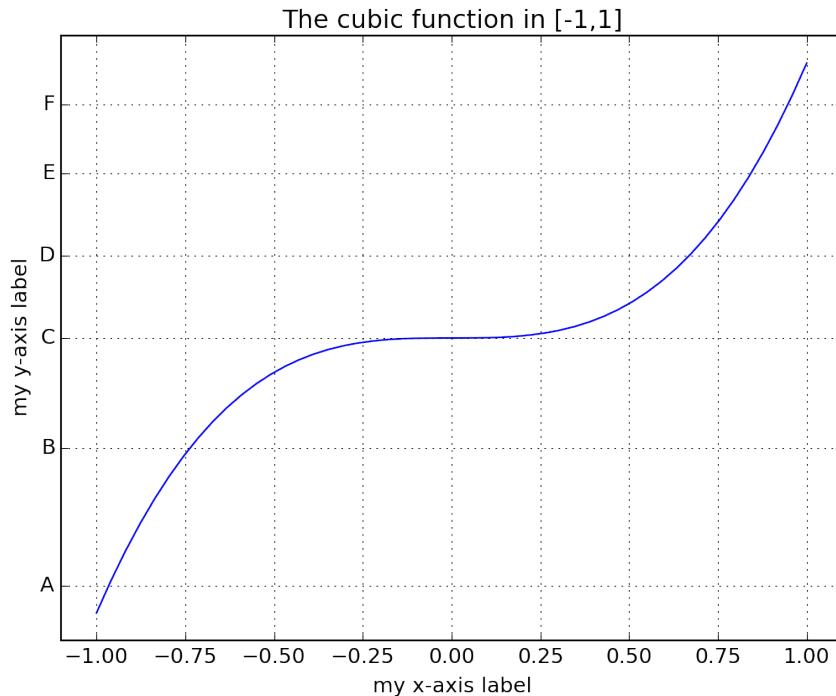
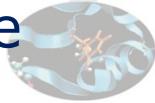


Un po' di cosmesi: titolo, etichette degli assi e griglia

Titolo ed etichette degli assi

```
>>> import numpy as np  
>>> import matplotlib.pyplot as plt  
>>> x = np.linspace(-1.0,1.0,50,endpoint=True)  
>>> y = x**3  
>>> plt.plot(x,y)  
  
>>> plt.title('The cubic function in [-1,1]')  
  
>>> plt.xlabel('my x-axis label')  
  
>>> plt.ylabel('my y-axis label')  
  
>>> plt.grid()  
  
>>> plt.show()
```

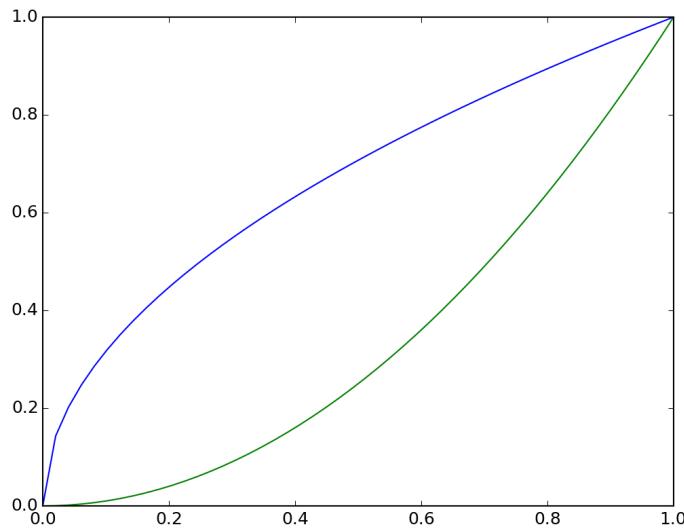
Un po' di cosmesi: il risultato finale



15

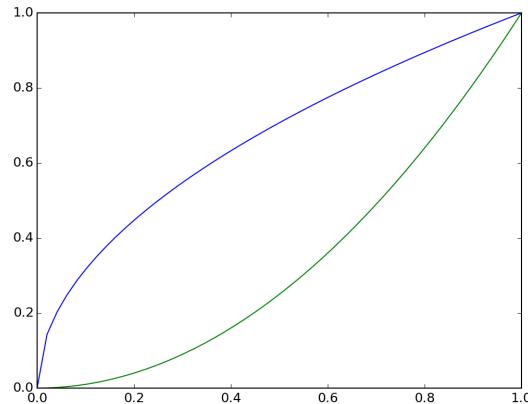
Più grafici nello stesso plot

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> x = np.linspace(0,1.0,50,endpoint=True)
>>> y1= x**0.5
>>> y2= x**2
>>> plt.plot(x,y1)
>>> plt.plot(x,y2)
>>> plt.show()
```



16

Più grafici nello stesso plot (2)



Osservazioni:

- Per default la linea del secondo grafico è verde
- Potrebbe essere utile aggiungere una legenda e anche dei marcatori in corrispondenza dei nostri dati

17

Colori di default e prime personalizzazioni del grafico



- Nella figura è riportata la sequenza dei colori di default per i grafici nello stesso sistema di assi
- Di default, l'eventuale ottavo grafico nello stesso sistema di assi torna ad essere disegnato in blu e così via
- Oltre alle coordinate del grafico, al comando `pyplot.plot()` possono essere passati ulteriori argomenti per alterare il comportamento di default nel disegno del relativo grafico, non solo per l'aspetto del colore.
- Ad esempio la funzione:

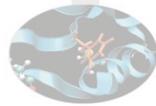
—	1 blue
—	2 green
—	3 red
—	4 cyan
—	5 magenta
—	6 bilious yellow
—	7 grey

```
pyplot.plot(x,y, color='red', linestyle='--')
```

disegna il grafico con una linea tratteggiata in colore rosso

18

Ancora su *plot linestyle*



- Possibili argomenti (da mettere tra apici) per la direttiva *linestyle*:
 - *solid line style* (default)
 - *dashed line style*
 - . *dash-dot line style*
 - : *dotted line style*
- Ovviamente è possibile aggiungere al grafico anche i *marker* dei dati; ad esempio la funzione:

```
pyplot.plot(x,y, marker='o', color='red')
```

disegna il grafico con una linea rossa con *marker* a forma di cerchio

- E' possibile utilizzare anche una sintassi "meno verbosa"; ad esempio la funzione:

```
pyplot.plot(x,y,'r--o')
```

disegna il grafico con una *dashed line* rossa con *marker* circolare per i dati

19

Più grafici nello stesso plot



```
>>> import numpy as np
>>> import matplotlib.pyplot as plt

>>> x = np.linspace(0,1.0,50,endpoint=True)
>>> y1= x**0.5
>>> y2= x**2

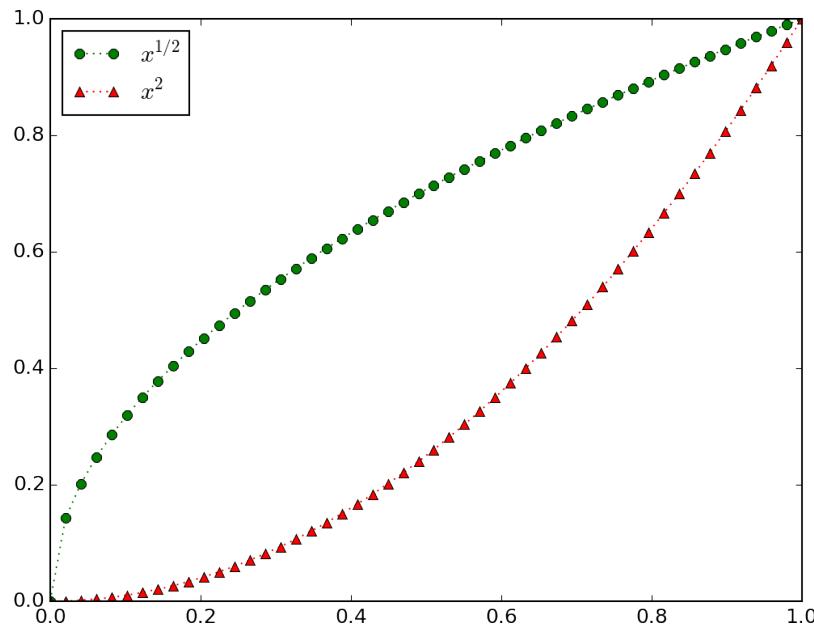
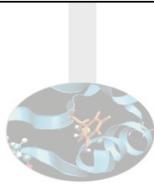
>>> plt.plot(x,y1,'g:o',label='$x^{1/2}$')
>>> plt.plot(x,y2,'r:^',label='$x^2$')
>>> plt.legend(loc='upper left')

>>> plt.grid(); plt.show()
```

- La proprietà *label* nelle due istruzioni di *plotting* è per necessaria per inserire la descrizione del grafico che sarà visualizzata in legenda
- Il comando `plt.legend()` consente di inserire una legenda nel grafico e la proprietà *loc*, di indicarne la posizione nel grafico stesso

20

Più grafici nello stesso plot: il risultato finale



NB: per i posizionamenti alternativi della legenda e per il controllo delle sue ulteriori proprietà consultare `help(pyplot.legend)`

21

Ancora sui marker



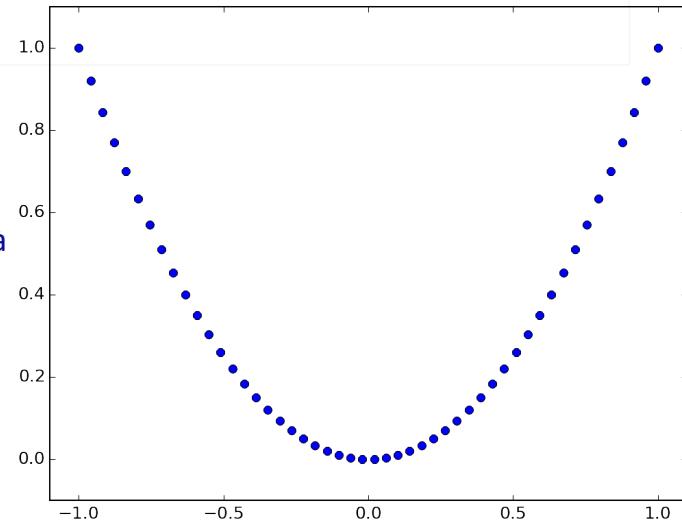
- Nella figura i possibili marker predefiniti
- Il comando `pyplot.plot` prevede proprietà per la gestione dell'aspetto dei marker:
 - `markersize=float`: consente di personalizzare la dimensione del marker
 - `markeredgewidth=float`: consente di personalizzare la dimensione del bordo del marker
 - `markerfacecolor='color string'`: consente di scegliere il colore dell'interno del marker
 - `markeredgecolor='color string'`: consente di scegliere il colore del bordo del marker

●	○	◆	D	×	x
•	·	◆	d	+	+
·	,	●	h		
▼	v	●	H	1	1
◀	<	★	*	2	2
▶	>	●		3	3
▲	^			4	4

22

Come ottenere uno scatter plot

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> x = np.linspace(-1,1,50,endpoint=True)
>>> y = x**2
>>> plt.plot(x,y, linestyle='', marker='o')
>>> plt.axis([-1.1,1.1,-0.1,1.1])
>>> plt.show()
```



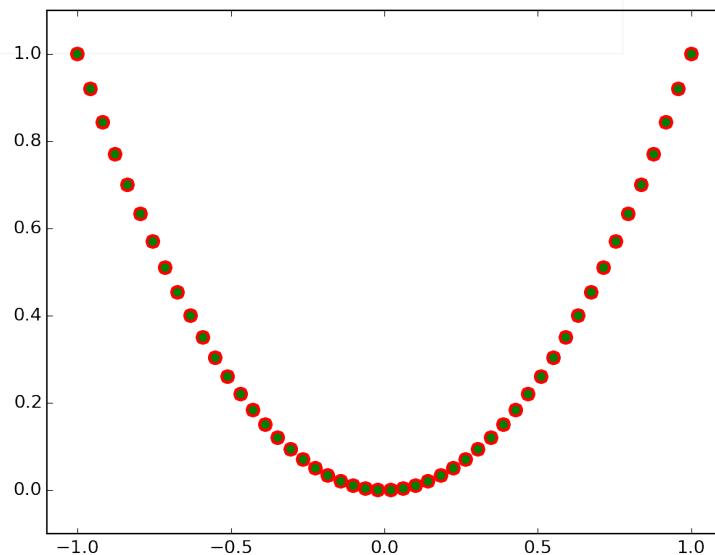
23

- Con il comando `pyplot.plot` è possibile disegnare uno *scatter plot*, utilizzando opportunamente le proprietà `linestyle` e `marker`
- NB: nel modulo `pyplot` è presente anche la funzione specifica `pyplot.scatter`

scatter plot e fancy marker

```
...
>>> plt.plot(x,y, linestyle='', marker='o',
            markersize=8.0, markeredgewidth=2.0,
            markerfacecolor='green',
            markeredgecolor='red')
...

```



24

Scatter plot di grosse moli di dati



Nel caso di uno *scatter plot* di un data set di grandi dimensioni si ottiene un risultato soddisfacente utilizzando il *pixel come singolo marker*

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt

>>> x = np.linspace(-1.0,1.0,10000,endpoint=True)
>>> z = np.random.rand(10000)

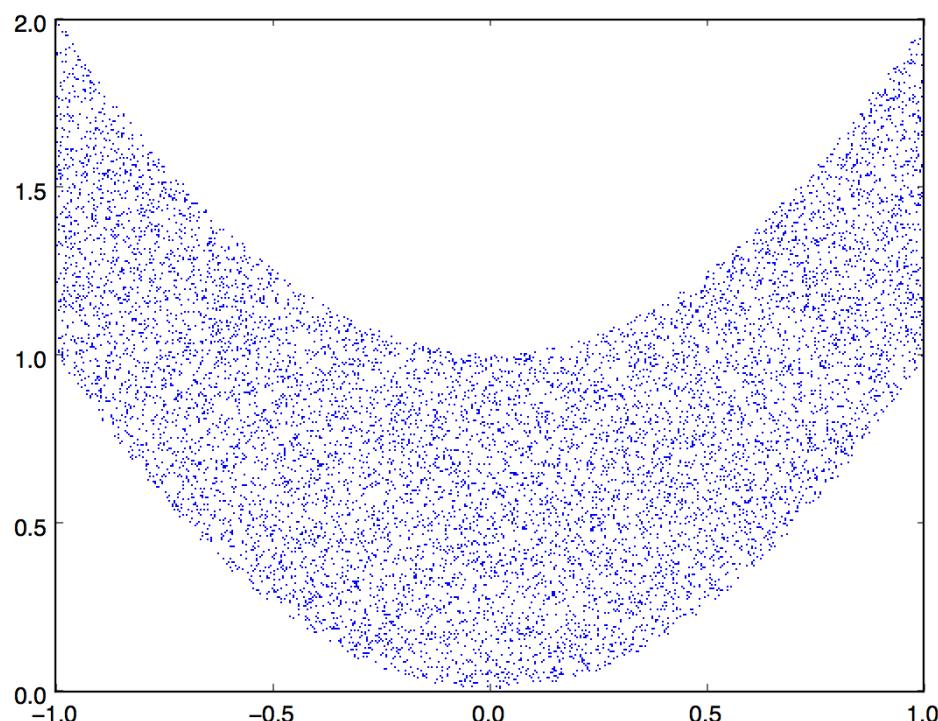
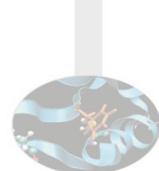
>>> y = x**2+z

>>> plt.plot(x,y, linestyle='', marker=',',
            markerfacecolor='blue')

>>> plt.show()
```

25

Scatter plot di grosse moli di dati (2)



26

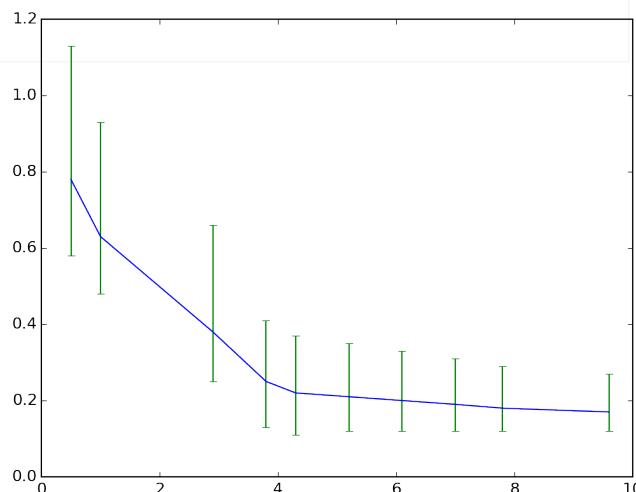
Plot con error bar

```
>>> import matplotlib.pyplot as plt
>>> x = [0.5,1.0,2.9,3.8,4.3,5.2,6.1,7.0,7.8,9.6]
>>> y = [0.78,0.63,0.38,0.25,0.22,0.21,0.20,0.19,0.18,0.17]

>>> ym = [0.2,0.15,0.13,0.12,0.11,0.09,0.08,0.07,0.06,0.05]
>>> yM = [0.35,0.3,0.28,0.16,0.15,0.14,0.13,0.12,0.11,0.1]

>>> plt.errorbar(x,y, yerr=(ym,yM), ecolor='green')
>>> plt.show()
```

- La dimensione delle liste **ym** ed **yM** deve essere la stessa di **y**
- La funzione supporta anche la proprietà **xerr**, per impostare eventuali “error array” per l’asse x

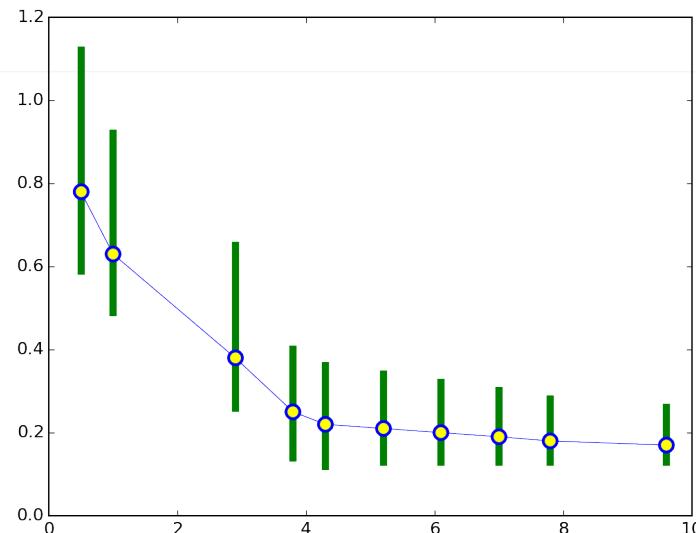


27

Plot con error bar: cosmesi

```
...
>>> plt.errorbar(x,y, yerr=(ym,yM), linewidth=0.5,
                  marker='o', markeredgecolor='blue',
                  markeredgewidth=2,markerfacecolor='yellow',
                  ecolor='green', markersize=10, elinewidth=5.0,
                  capsized=0)
...

```



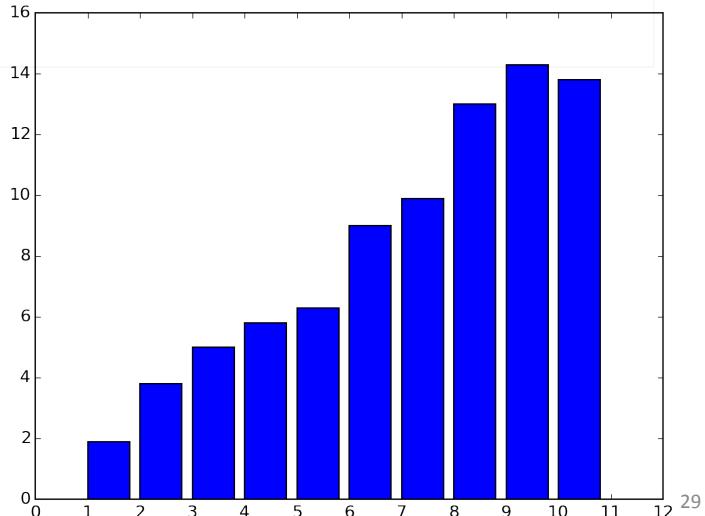
28

Bar plot

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> y = [1.9,3.8,5.0,5.8,6.3,9.0,9.9,13.0,14.3,13.8]
>>> x = np.linspace(1,10,10,endpoint=True)

>>> plt.bar(x,y)
>>> plt.xticks(np.linspace(0,12,13,endpoint=True))
>>> plt.show()
```

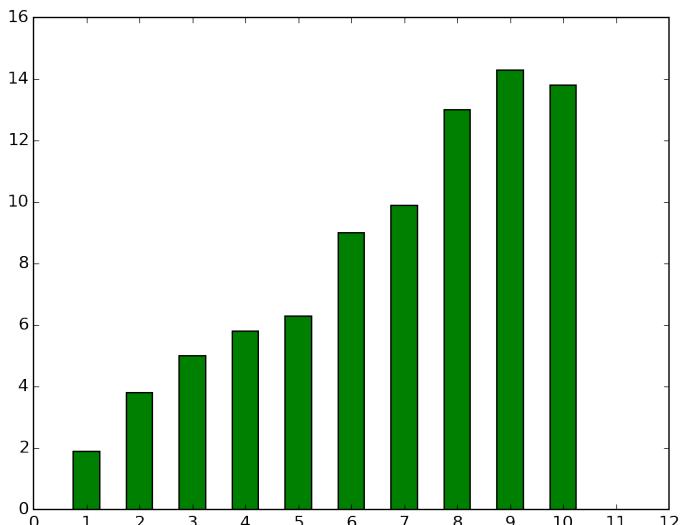
- A differenza della funzione `pyplot.plot`, in `pyplot.bar` il primo argomento (`array x`) è obbligatorio.
- Di default, la singola coordinata dell'array `x` è l'angolo in basso a sinistra della relativa barra; la proprietà `align`, consente di variare questo comportamento



Bar plot: cosmesi

```
...
>>> plt.bar(x,y, align='center', width=0.35, color='green')
...
```

- `align='center'` consente di ottenere barre centrate sulla relativa coordinata x
- L'argomento della proprietà `width`, può anche essere un array, per ottenere barre di dimensioni diverse



Bar plot: un ulteriore esempio

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt

>>> avr_men = (20,35,30,35,27); std_men = (2,3,4,1,2)
>>> avr_women = (25,32,34,20,25); std_women = (3,5,2,3,3)

>>> index = np.arange(5);
>>> eConf = {'ecolor': '0.3'}

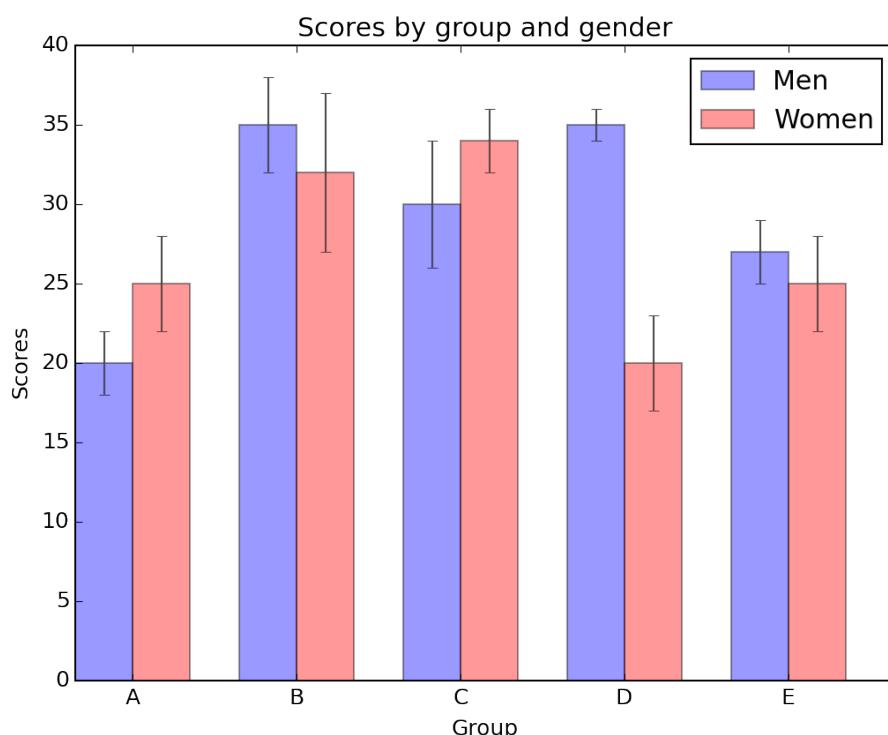
>>> plt.bar(index, avr_men, width=0.35, alpha=0.40,
           color='b', yerr=std_men, error_kw=eConf, label='Men')

>>> plt.bar(index+0.35, avr_women, width=0.35, alpha=0.40,
           color='r', yerr=std_women, error_kw=eConf, label='Women')

>>> plt.xlabel('Group'); plt.ylabel('Scores')
>>> plt.title('Scores by group and gender')
>>> plt.xticks(index + 0.35, ('A', 'B', 'C', 'D', 'E'))
>>> plt.legend(); plt.show()
```

31

Bar plot: un ulteriore esempio (2)



32

Istogrammi

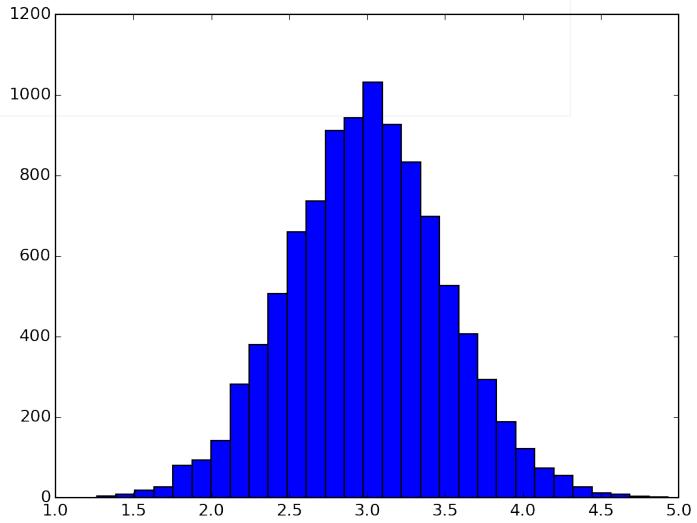
```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
```

```
>>> data = np.random.normal(loc=3, scale=0.5, size=10000)
```

```
>>> plt.hist(data, bins=30)
```

```
>>> plt.show()
```

- `np.random.normal` è una funzione del modulo di `numpy` per la generazione di numeri casuali; nel nostro caso genera 10000 numeri casuali da normale con media 3 e varianza 0.5
- La proprietà `bins` di `plt.hist` indica il numero degli intervalli per il conteggio; l'argomento di `bins` può essere un proprio array di N+1 estremi degli intervalli



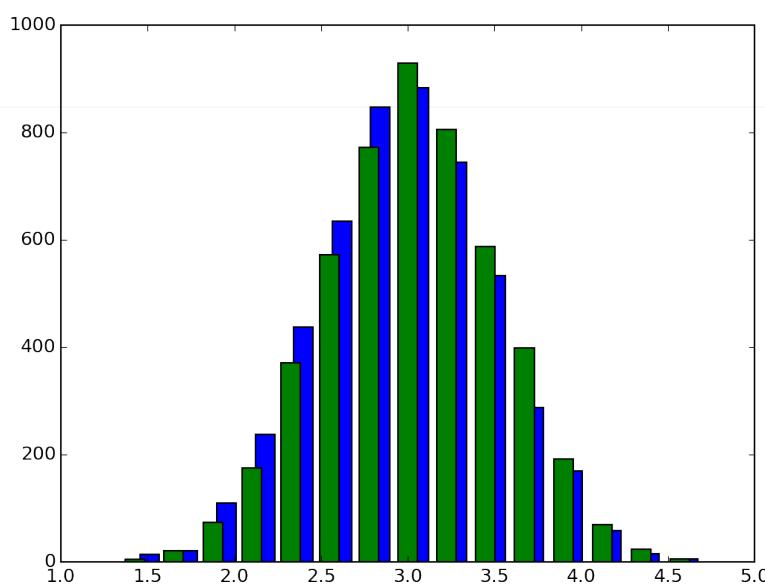
33

Istogrammi di più set di dati

...

```
>>> plt.hist(data[:5000], bins=15, rwidth=0.5)
>>> plt.hist(data[5000:], bins=15, rwidth=0.5)
```

...



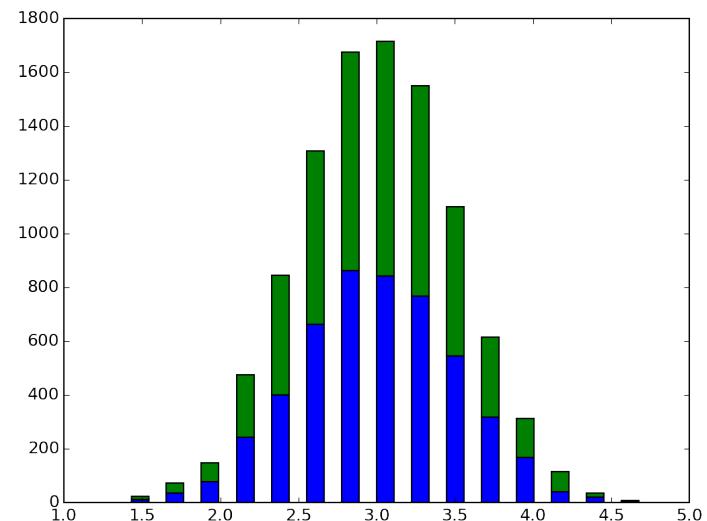
34

Istogrammi di più set di dati (2)

```
...
>>> plt.hist([data[5000],data[5000:], bins=15, histtype='barstacked')
...

```

Se le due serie di dati prevedono lo stesso “binning”, possono anche essere rappresentate in forma “*stacked*”, aggiungendo, ad una singola chiamata a *pyplot.hist*, oltre ai anche l’argomento *histtype='barstacked'*



La funzione *subplot*

La funzione **plt.subplot(nrows, ncols, index)** permette di costruire *subplot* in una singola figura. Vediamone un esempio d’uso:

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> x = np.linspace(0.0, 5.0)
>>> y1 = np.cos(2*np.pi*x)*np.exp(-x)
>>> y2 = np.cos(2*np.pi*x)

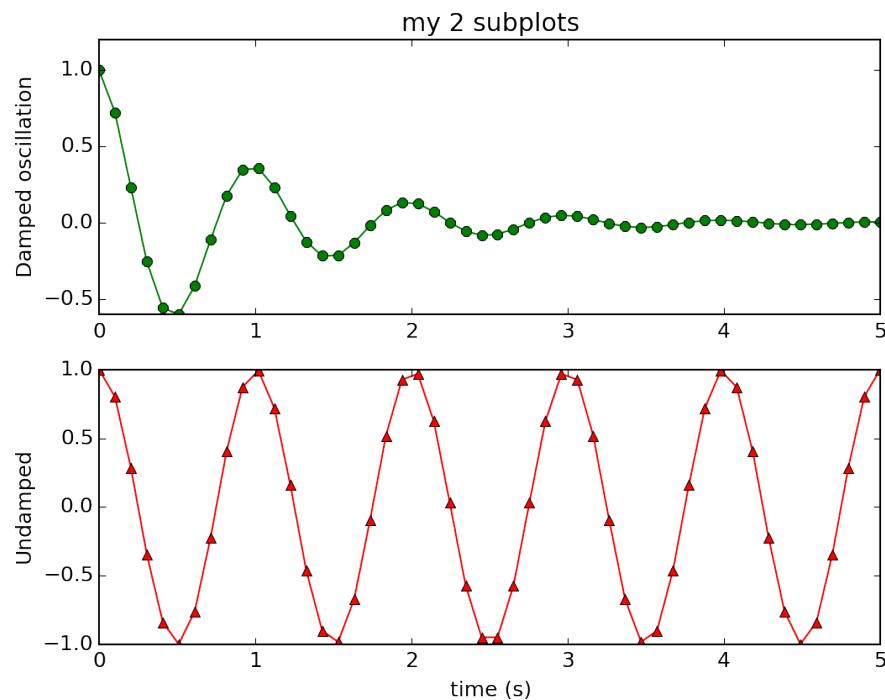
>>> plt.subplot(2, 1, 1)
>>> plt.plot(x, y1, 'go-')
>>> plt.title('my 2 subplots')
>>> plt.ylabel('Damped')

>>> plt.subplot(2, 1, 2)
>>> plt.plot(x, y2, 'r^-')
>>> plt.xlabel('time (s)')
>>> plt.ylabel('Undamped')

>>> plt.show()
```

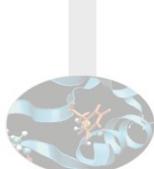
36

Il risultato del nostro subplot



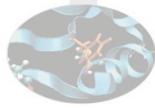
37

Altri tipi di plot

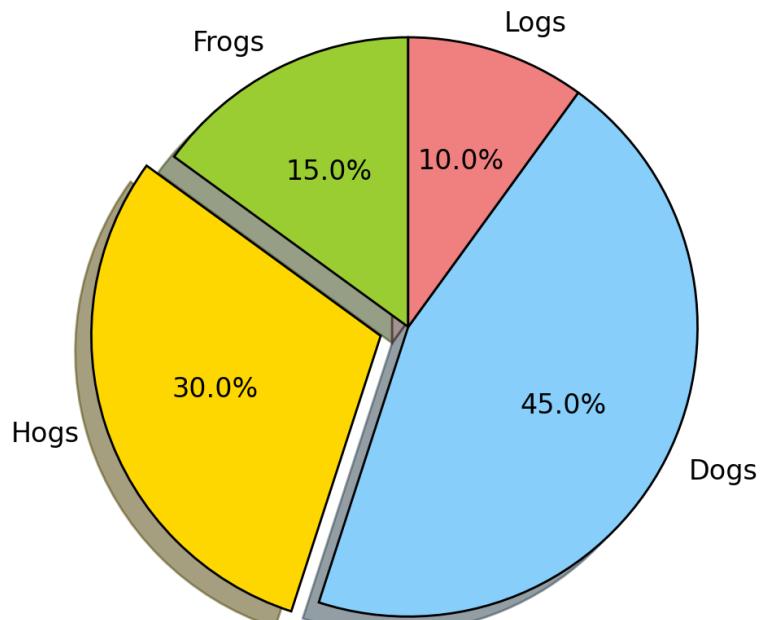


- Oltre alle funzioni finora descritte, il modulo `matplotlib.pyplot` ne mette a disposizione altre, che consentono di ottenere ulteriori tipologie di plot.
- Per una *rassegna* sulle tipologie di grafico che è possibile consultare <http://matplotlib.org/gallery.html>, in cui, per ogni plot visualizzabile, è disponibile lo script Python che lo genera
- Vediamo qualche ulteriore esempio di cosa è possibile realizzare in `matplotlib`, senza entrare nel dettaglio dei comandi (istruzioni) necessari

38



Pie plot

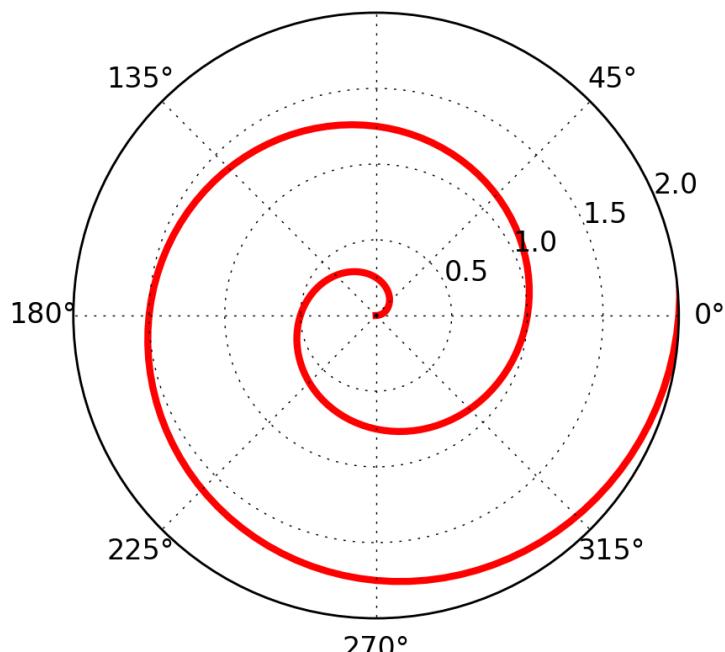


39



Polar plot

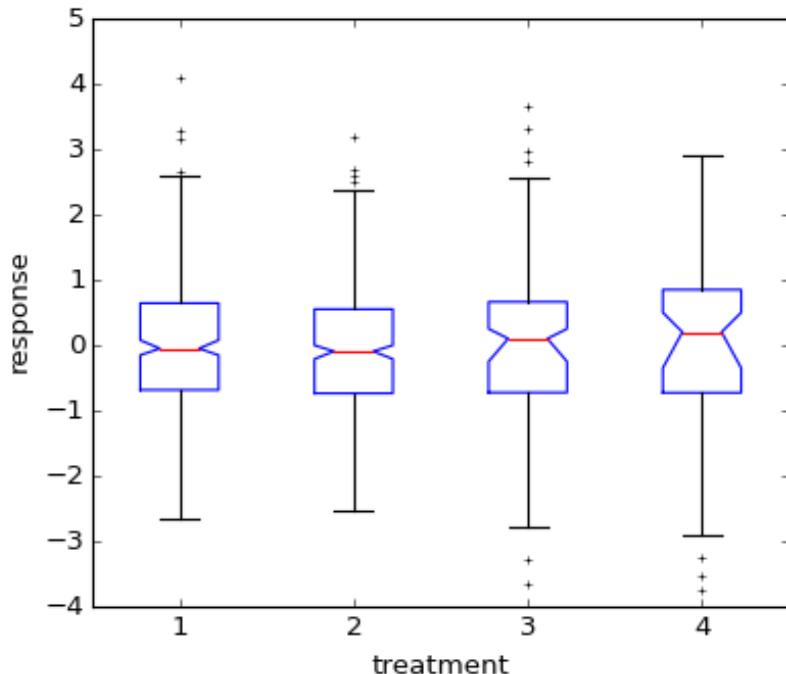
A line plot on a polar axis
 90°



40



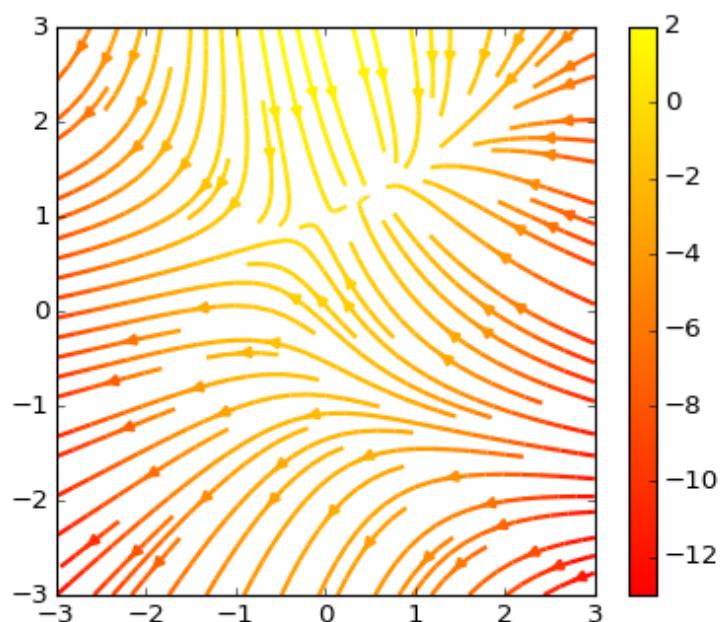
box plot



41

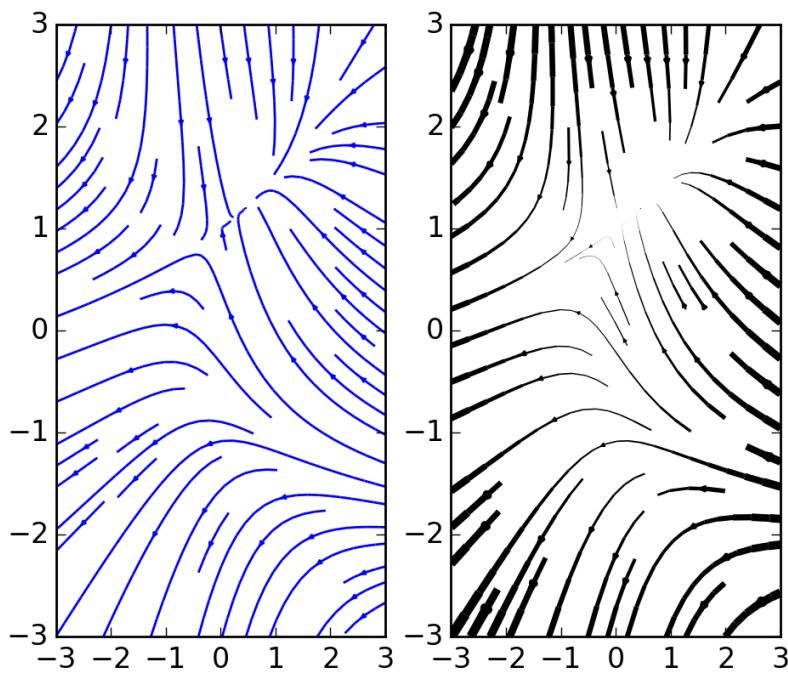
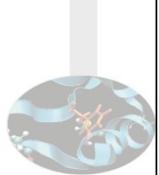


streamline plot

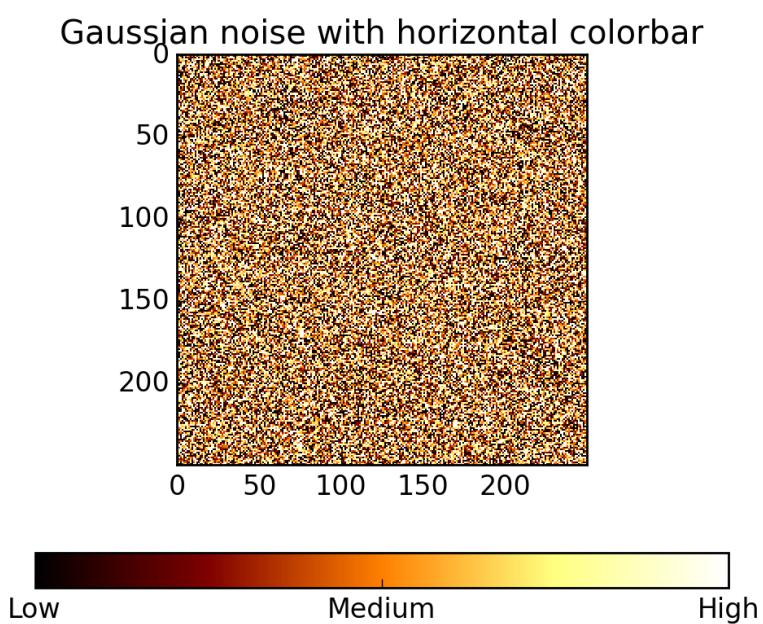


42

streamline plot (II)



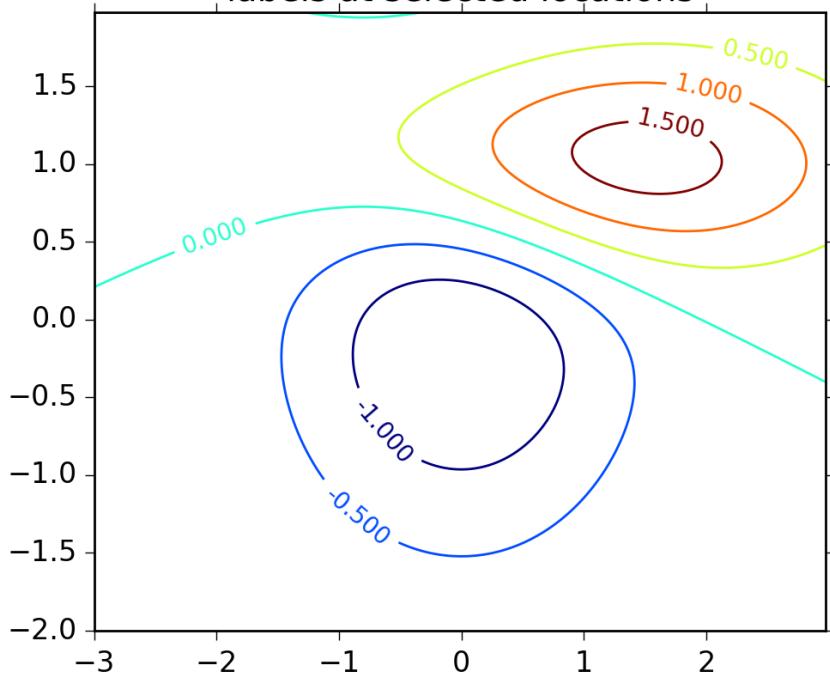
Una matrice in "grafica"





Simple contour plot

labels at selected locations

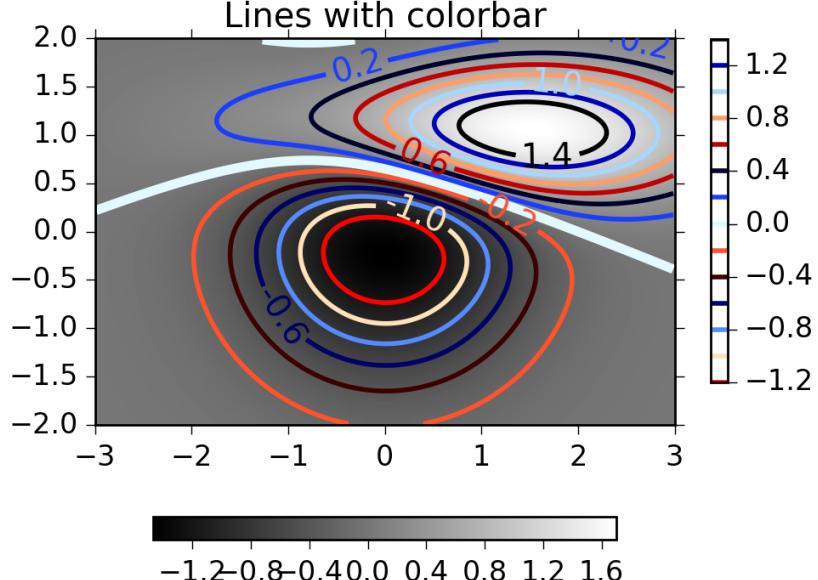


45



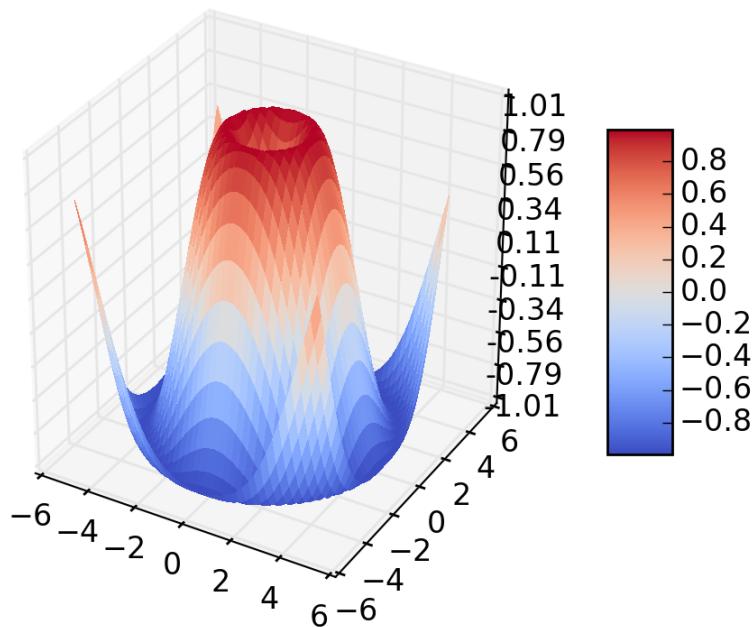
contour plot with colorbar

Lines with colorbar



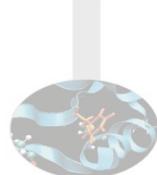
46

Plot in 3D



47

Output



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

- *User interface backend*: per l’assemblaggio dei grafici in una GUI; in Python esistono diverse librerie per la costruzione di interfacce grafiche tra cui *Tkinter*, *PyQt*, *pygtk* che vengono supportate da *matplotlib*.
- *Hardcopy backend*: per la stampa su file; i formati *.jpg, *.png, *.svg, *.pdf, *.rgba sono supportati.

Nel modulo *pyplot*, è disponibile la funzione:

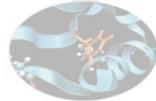
```
savefig(filename, dpi=150, transparent=False)
```

in cui il formato dell’immagine che sarà generata, in generale, dipende dall’estensione di **filename** (NB: sono visualizzate solo le principali proprietà)

49

Reference ed approfondimenti:

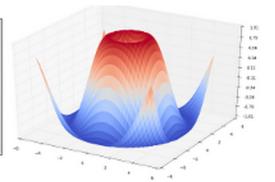
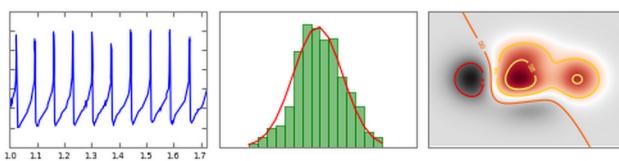
<http://www.matplotlib.org>



[home](#) | [examples](#) | [gallery](#) | [pyplot](#) | [docs](#) »

Introduction

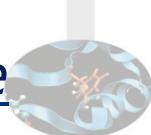
matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. matplotlib can be used in python scripts, the python and [ipython](#) shell (ala MATLAB® or Mathematica®), web application servers, and six graphical user interface toolkits.



matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc, with just a few lines of code. For a sampling, see the [screenshots](#), [thumbnail](#) gallery, and [examples](#) directory

50

[matplotlib.org: la documentazione](http://www.matplotlib.org)



[home](#) | [examples](#) | [gallery](#) | [pyplot](#) | [docs](#) »

Overview

Release: 1.4.2

Date: October 25, 2014

[Download PDF](#)

- [User's Guide](#)

[Introduction](#)

[Configuration Guide](#)

[Beginner's Guide](#)

[Advanced Guide](#)

[What's new in matplotlib](#)

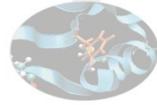
[Github stats](#)

[License](#)

[Credits](#)

51

[matplotlib.org: beginner's guide](#)



[home](#) | [examples](#) | [gallery](#) | [pyplot](#) | [docs](#) » [User's Guide](#) »

Beginner's Guide

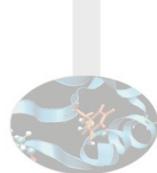
Release: 1.4.2

Date: October 25, 2014

- [Pyplot tutorial](#)
 - [Controlling line properties](#)
 - [Working with multiple figures and axes](#)
 - [Working with text](#)
- [Customizing plots with style sheets](#)
 - [Defining your own style](#)
 - [Composing styles](#)
 - [Temporary styling](#)
- [Interactive navigation](#)
 - [Navigation Keyboard Shortcuts](#)
- [Working with text](#)
 - [Text introduction](#)
 - [Basic text commands](#)
 - [Text properties and layout](#)
 - [Writing mathematical expressions](#)
 - [Typesetting With XeLaTeX/LuaLaTeX](#)

52

[matplotlib.org: advanced guide](#)



[home](#) | [examples](#) | [gallery](#) | [pyplot](#) | [docs](#) » [User's Guide](#) »

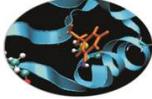
Advanced Guide

Release: 1.4.2

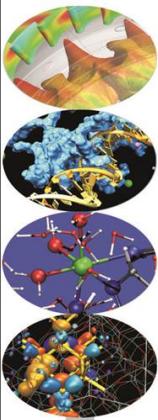
Date: October 25, 2014

- [Artist tutorial](#)
 - [Customizing your objects](#)
 - [Object containers](#)
 - [Figure container](#)
 - [Axes container](#)
 - [Axis containers](#)
 - [Tick containers](#)
- [Customizing Location of Subplot Using GridSpec](#)
 - [Basic Example of using subplot2grid](#)
 - [GridSpec and SubplotSpec](#)
 - [Adjust GridSpec layout](#)
 - [GridSpec using SubplotSpec](#)
 - [A Complex Nested GridSpec using SubplotSpec](#)
 - [GridSpec with Varying Cell Sizes](#)
- [Tight Layout guide](#)
 - [Simple Example](#)

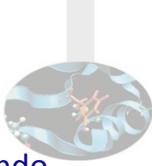
53



Il modulo matplotlib.pyplot: laboratorio



Laboratorio pyplot



1. Tracciare il grafico della funzione $\sin(x)$ nell'intervallo $[0,10]$, utilizzando una linea tratteggiata rossa.
hint: la funzione `numpy.sin(x)` ritorna un array delle stesse dimensioni di `x`, contenente il seno di `x` elemento per elemento.

2. Sulla stessa figura precedente, inserire anche titolo ed etichette degli assi in colore verde e con una dimensione del font di 16 punti
hint: consultare l'help delle funzioni che consentono di inserire titolo ed etichette degli assi

3. Alla figura dell'esercizio 2, aggiungere il grafico della funzione $\cos(x)$, sempre nell'intervallo $[0,10]$, utilizzando una linea blu e marker a forma di diamante, e, ovviamente, anche una legenda

4. Con i 2 *dataset* precedenti, costruire un bar plot, con barre gialle per il seno e barre verdi per il coseno