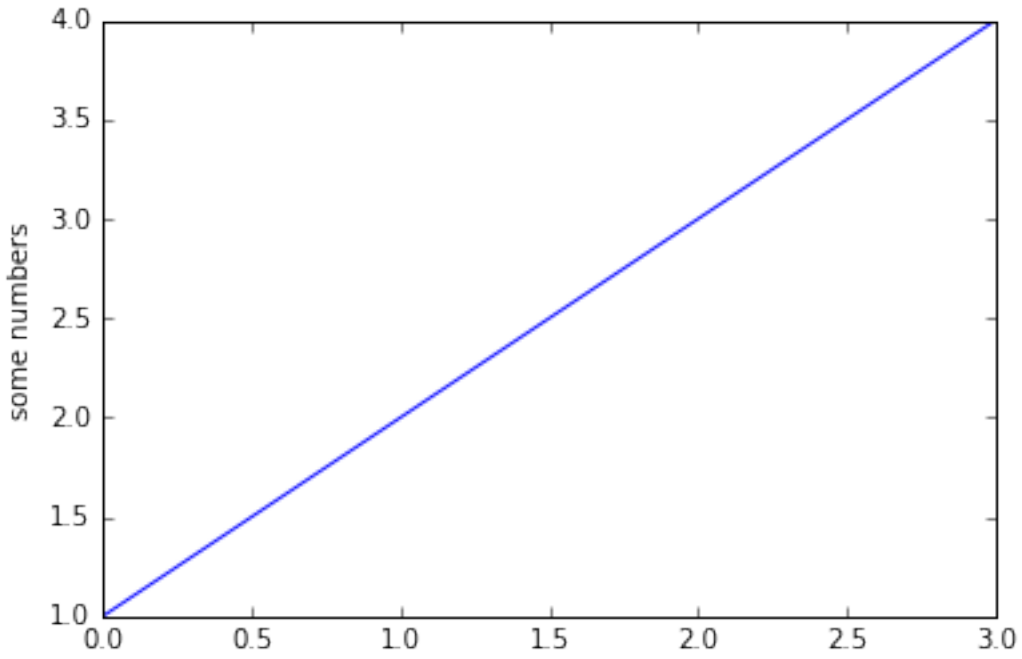
nb2-tutorial_pyplot

December 5, 2017

Breve compendio per l'uso dei plot in Python

matplotlib e' un contenitore di programmi realizzato per facilitare il plotting attraverso un suo sottoinsieme chiamato **matplotlib.pyplot**. Ciascuna funzione di **pyplot** produce un effetto sulla figura: p.es.: crea una figura, ritaglia un'area di plot all'interno di una figura, disegna linee nell'area del plot, sovrappone etichette ed indicazioni sul grafico, ecc. In **matplotlib.pyplot** vari parametri che definiscono un plot sono preservati nella sessione come p.es. la finestra di plot corrente cosi' che le funzioni di plot sono dirette agli assi attuali.

```
In [1]: import matplotlib.pyplot as plt
        plt.plot([1,2,3,4])
        plt.ylabel('some numbers')
        plt.show()
```

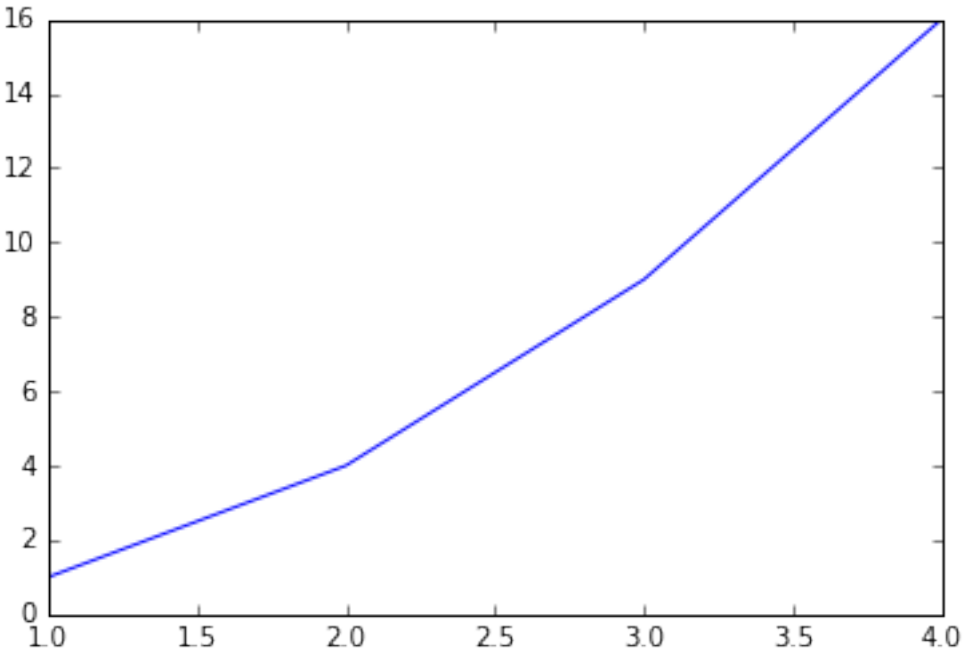


Dal plot precedente si vede che gli assi si sono automaticamente organizzati in modo da contenere la dinamica dei valori di X ed Y: 0<X<3 ed 0<Y<4. Un input formato da una sola lista di numeri viene quindi interpretato dal comando di plot() come una sequenza di valori di Y che riporta su valori della X crescenti in modo automatico. Poiche' l'uso degli indici in python inizia con 0 (zero) allora il vettore X ha la stessa lunghezza di Y ma iniozia dall 0. Quindi i valori riportati slla X sono [0,1,2,3].

Il comando plot() e' comunque versatile e accetta un numero di argomenti arbitrario. Per esempio avendo sia valori di X che di Y si puo' dare il comando:

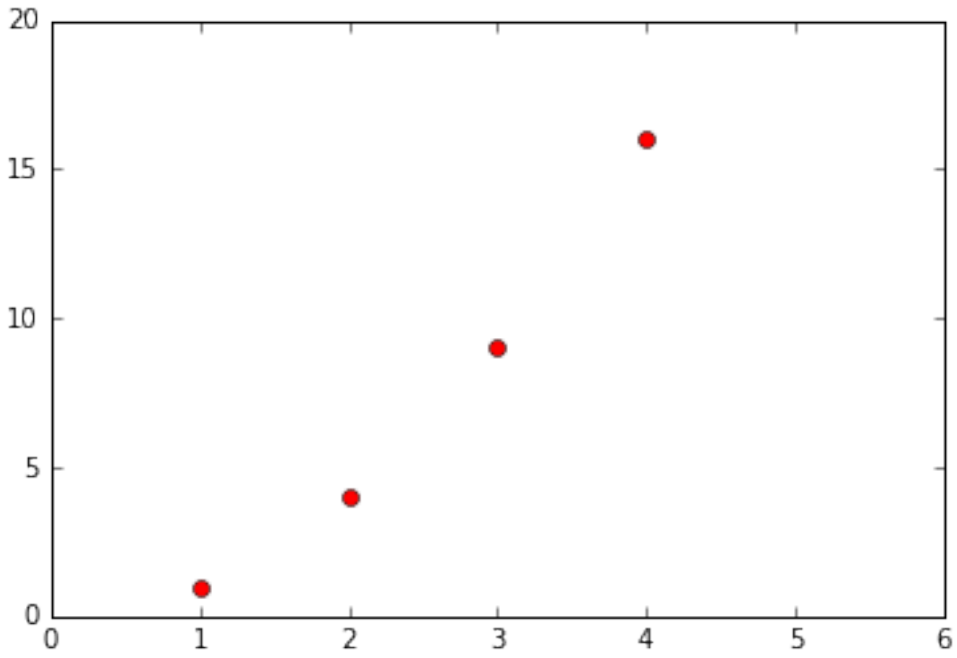
```
In [3]: plt.plot([1, 2, 3, 4], [1, 4, 9, 16])

Out[3]: [<matplotlib.lines.Line2D at 0x7fbb80385c50>]
```



Inoltre, per ogni coppia X, Y si puo' specificare un terzo argomento, in forma di stringa, che rappresenta colore e tipo di linea da usare nel plot. The letters and symbols of the format string are from MATLAB, and you concatenate a color string with a line style string. The default format string is 'b-', which is a solid blue line. Per esempio, per graficare gli stessi valori in forma di cerchi rossi si usera':

```
In [5]: import matplotlib.pyplot as plt
        plt.plot([1,2,3,4], [1,4,9,16], 'ro')
        plt.axis([0, 6, 0, 20])
        plt.show()
```

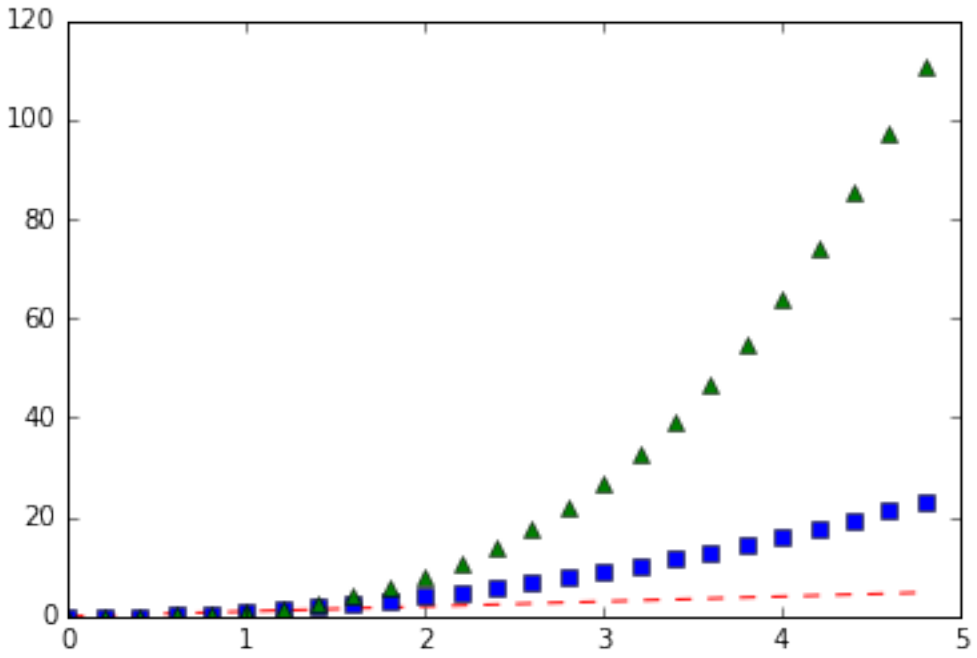


dove il comando plt.axis([0, 6, 0, 20]) e' servito per forzare gli estremi degli assi e vedere meglio i punti. Con **matplotlib** possiamo riportare nello stesso grafico piu' serie di punti. L'esempio sotto illustra come plottare con un solo comando piu' serie di punti con stili differenti:

```
In [6]: import numpy as np
        import matplotlib.pyplot as plt

        # simulo una variabile che va da 0 a 5, campionata ad intervalli di 0.2
        t = np.arange(0., 5., 0.2)

        # uso tratteggio rosso('r--'), quadrati blu('bs'), triangoli verdi('g^')
        plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
        # la prima curva e' una retta, la seconda rappresenta x^2 e la terza e' x^
        plt.show()
```

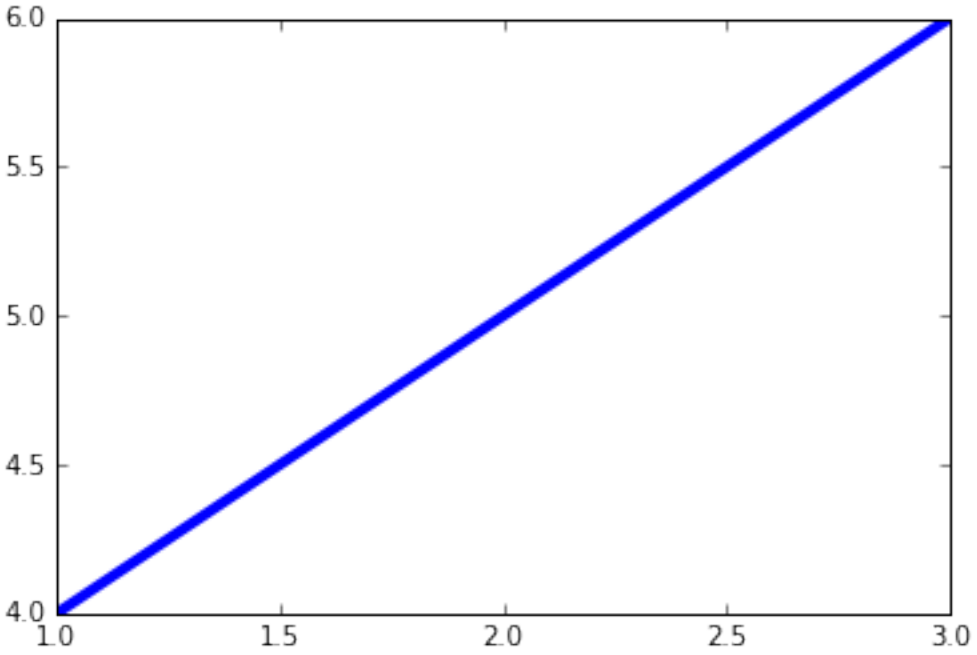


Proprieta' delle linee

- Le linee hanno vari attributi (parametri) che possono essere specificati: spessore, tratteggio, antialiasing, ecc. Le proprieta' delle linee possono essere definite in vari modi:

  P.es usando la keyword linewhidth=:

```
In [11]: plt.plot([1,2,3], [4,5,6], linewidth=4.0)
```



Tutti i comandi di plot sono indirizzati agli assi definiti al momento del plot. La funzione plt.gca() (da **g**et **c**urrent **a**xis) restituisce gli assi correnti, mentre plt.gcf() (da **g**et **c**urrent **f**igure) restituisce info sulla figura corrente. Normalmente non dovremo occuparci di questo perche' il tutto viene gestito in modo trasparente. P.es. nel seguito creiamo due plot insieme:

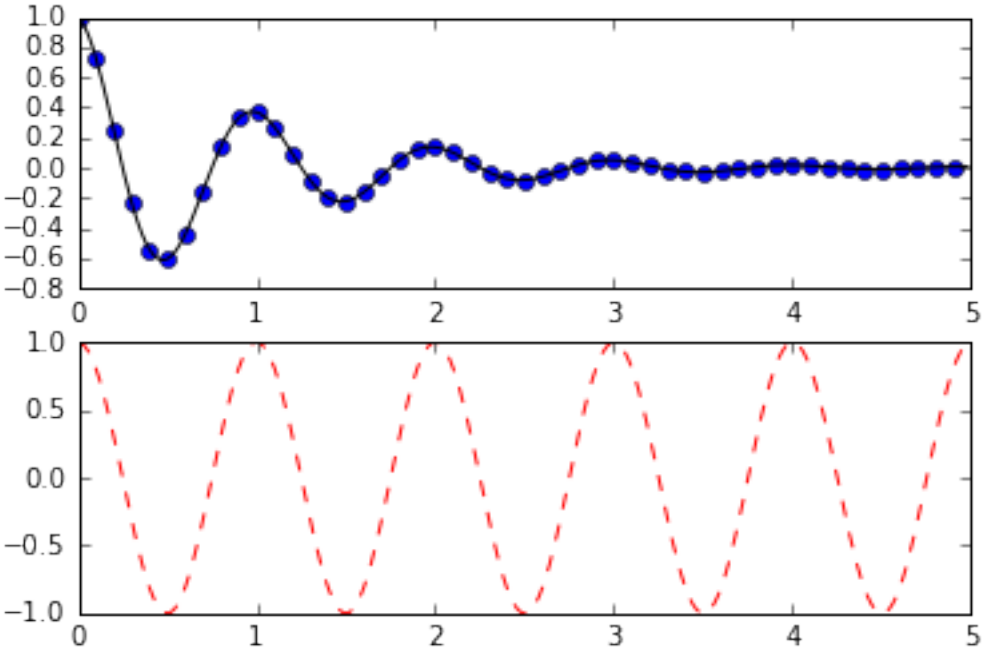
```
In [1]: import numpy as np
        import matplotlib.pyplot as plt

        def f(t):
            # definizione per calcolare un esponenziale decrescente per un coseno
            return np.exp(-t) * np.cos(2*np.pi*t)

        t1 = np.arange(0.0, 5.0, 0.1)     # punti in cui calcoliamo il valore della
        t2 = np.arange(0.0, 5.0, 0.02)    # punti piu' fitti in cui calcoliamo il va

        plt.figure(1)
        plt.subplot(211)      # (211) i tre numeri corrispondono ad una suddivisione
                              #       della finestra di plot in righe, colonne, n.de
        plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')

        plt.subplot(212)
        plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
        plt.show()
```



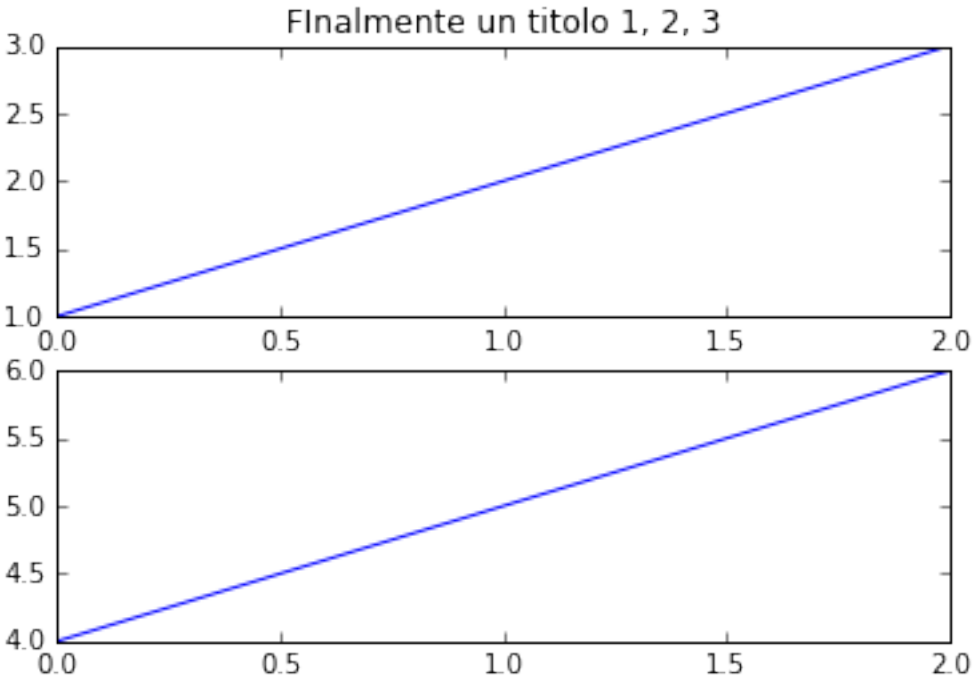
Aumentiamo la complessita' disegnando tre grafici nella stessa pagina:

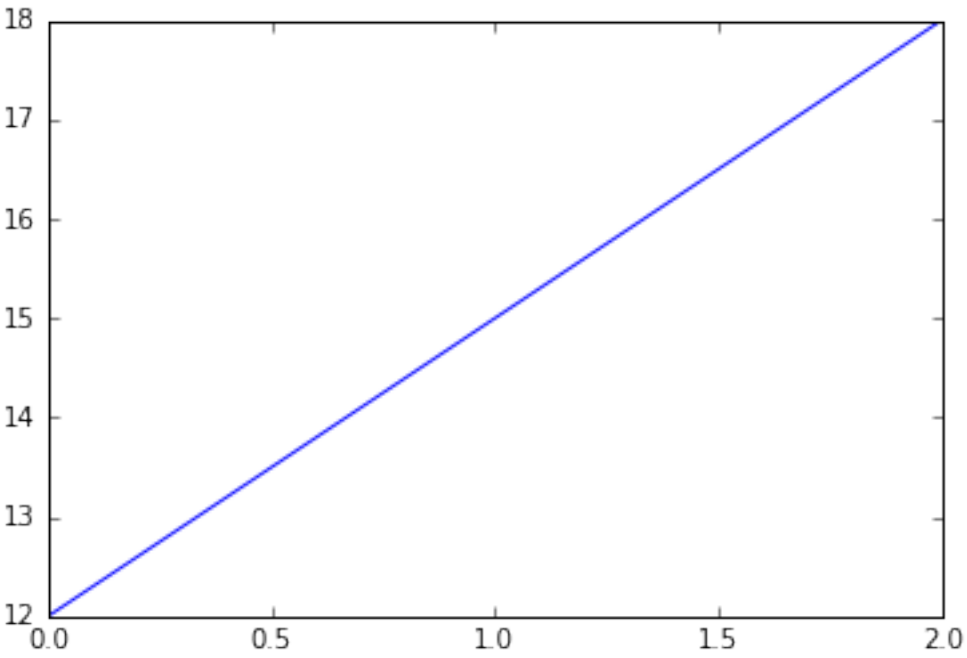
In [23]:
```
import matplotlib.pyplot as plt
plt.figure(1)                # la prima figura
plt.subplot(211)             # definisce il primo subplot nella prima figu
plt.plot([1, 2, 3])          # una sola sequenza di numeri viene associata
plt.subplot(212)             # definisce il secondo subplot nella prima fi
plt.plot([4, 5, 6])


plt.figure(2)                # apre una seconda figura
plt.plot([12, 15, 18])       # genera un unico plot se non si definisce su

# a questo punto per aggiungere un titolo in testa alle figure torno alla
plt.figure(1)                # rendo la figura 1 "current"; subplot(212) e
plt.subplot(211)             # rendo "corrente il subplot(211) nella figur
plt.title('FInalmente un titolo: 1, 2, 3') # aggiungo un titolo al subplot
```

Out[23]: <matplotlib.text.Text at 0x7f628d01f208>





Andiamo ora a scrivere del testo all'interno del grafico: questo puo' essere riportato sia in punti predefiniti del grafico (p.es. per specificare, assi e titolo con xlabel(), ylabel(), title()) o anche in punti generici sceti da noi (con text()).

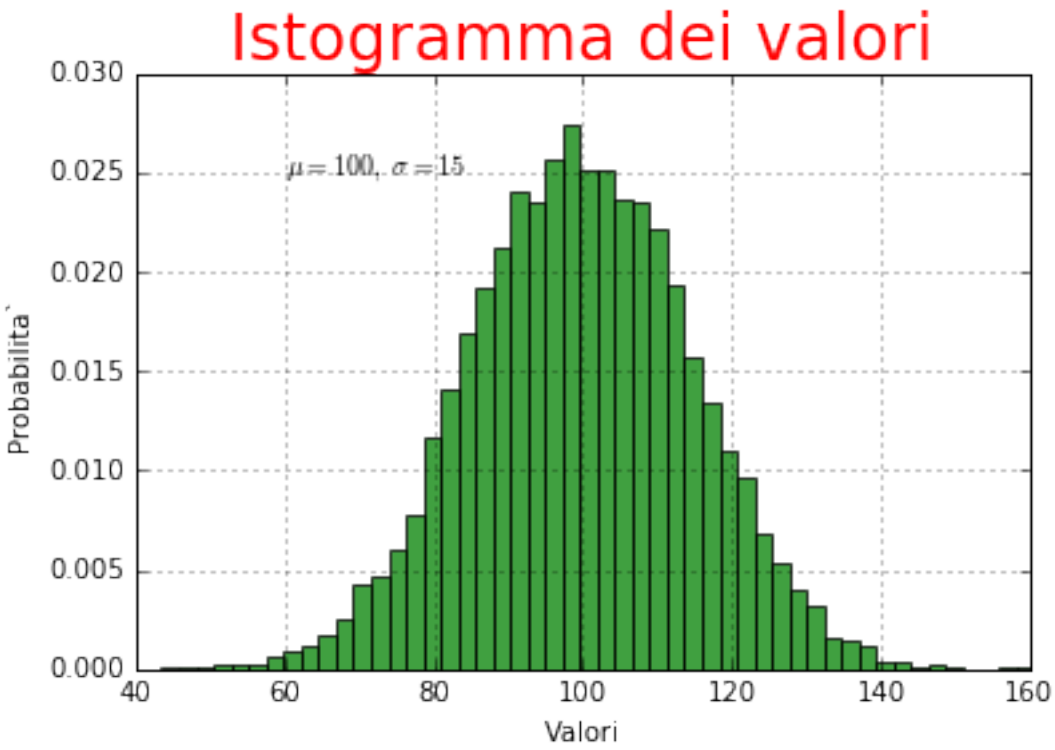
```
In [52]: import numpy as np
         import matplotlib.pyplot as plt

         # stabilisco il punto di partenza (il "seed") dei numeri
         # casuali, per rendere riproducibile i grafico
         np.random.seed(19680801)

         # ora genero i dati simulati
         mu, sigma = 100, 15
         x = mu + sigma * np.random.randn(10000) # estrae da una gaussiana standard
                                                 # cioe' a media zero e sigma unita

         # genero l'istogramma dei dati simulati
         plt.hist(x, 50, normed=1, facecolor='g', alpha=0.75)    # 'g' sta per green

         plt.xlabel('Valori')
         plt.ylabel('Probabilita`')
         plt.title('Istogramma dei valori',fontsize=24, color='red') # cambia il fo
         plt.text(60, .025, r'$\mu=100,\ \sigma=15$')
         plt.axis([40, 160, 0, 0.03])
         plt.grid(True)
         plt.show()
```



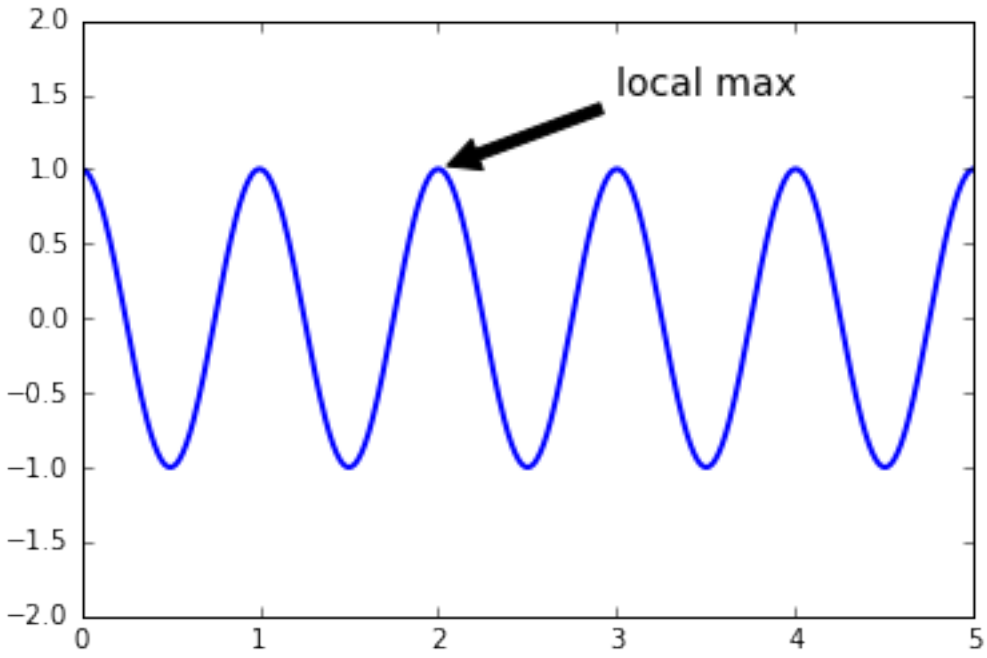
Possiamo anche annotare un grafico usando plt.annotate() che permette di controllare sia la posizione xy su cui puntare, sia la posizione xytext a cui inserire il testo:

```
In [65]: import numpy as np
         import matplotlib.pyplot as plt

         t = np.arange(0.0, 5.0, 0.01)
         s = np.cos(2*np.pi*t)
         line, = plt.plot(t, s, lw=2)

         plt.annotate('local max',fontsize=14, xy=(2, 1), xytext=(3, 1.5),
                      arrowprops=dict(facecolor='black', shrink=0.05)
                      )

         plt.ylim(-2,2)
         plt.show()
```



Quando i dati da rappresentare si estendono su intervalli di valori molto grandi sara' utile usare assi logaritmici:

```
In [66]: import numpy as np
         import matplotlib.pyplot as plt

         from matplotlib.ticker import NullFormatter  # useful for `logit` scale

         # Fixing random state for reproducibility
         np.random.seed(19680801)

         # estrae valori da una gaussiana normalizzata nell'intervallo ]0, 1[
         # loc e' il centro; scale e' la standard deviation
         y = np.random.normal(loc=0.5, scale=0.4, size=1000)
         y = y[(y > 0) & (y < 1)]
         y.sort()
         x = np.arange(len(y))

         # plot with various axes scales
         plt.figure(1)

         # linear
         plt.subplot(221)     # divido il campo in 2x2 plot, usando qui il primo (1)
         plt.plot(x, y)
         plt.yscale('linear')
         plt.title('linear')
```

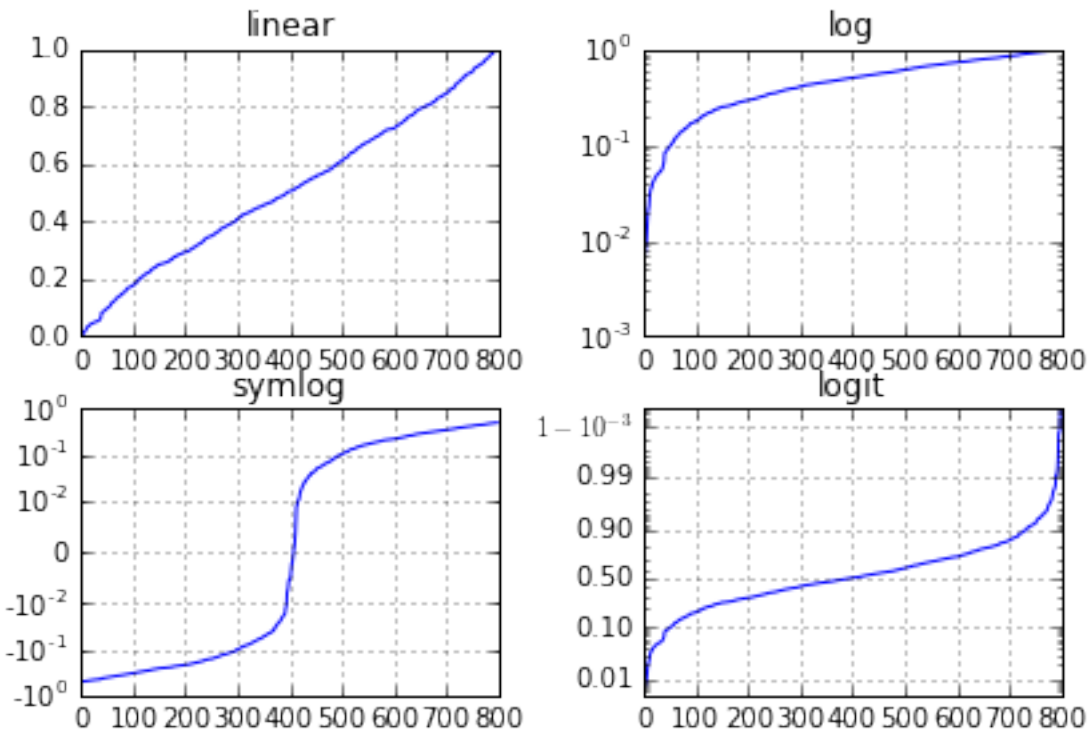
```
plt.grid(True)


# log
plt.subplot(222)    # divido il campo in 2x2 plot, usando qui il secondo
plt.plot(x, y)
plt.yscale('log')
plt.title('log')
plt.grid(True)


# symmetric log
plt.subplot(223)    # divido il campo in 2x2 plot, usando qui il terzo (3)
plt.plot(x, y - y.mean())
plt.yscale('symlog', linthreshy=0.01)
plt.title('symlog')
plt.grid(True)

# logit
plt.subplot(224)    # divido il campo in 2x2 plot, usando qui il quarto (4
plt.plot(x, y)
plt.yscale('logit')
plt.title('logit')
plt.grid(True)
# Format the minor tick labels of the y-axis into empty strings with
# `NullFormatter`, to avoid cumbering the axis with too many labels.
plt.gca().yaxis.set_minor_formatter(NullFormatter())
# Adjust the subplot layout, because the logit one may take more space
# than usual, due to y-tick labels like "1 - 10^{-3}"
plt.subplots_adjust(top=0.92, bottom=0.08, left=0.10, right=0.95, hspace=0
                    wspace=0.35)

plt.show()
```



In [ ]:

In [ ]:

In [ ]: