Python - Analisi dati e grafici

“giuliof"

GOLEM

2 ottobre 2018

Cosa abbiamo fatto la volta scorsa?

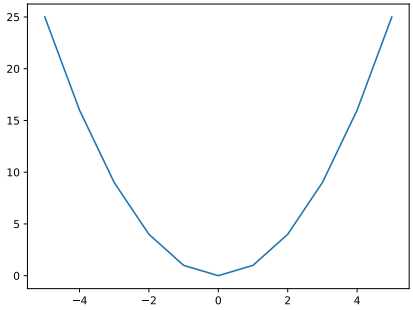
- Liste, array e dizionari
- Lettura e scrittura su file
- Librerie (cenni alla libreria matematica)

Link alla lezione precedente:
`https://golem.linux.it/wiki/Ore_del_GOLEM#3_Luglio:_Python_303`

Primo grafico

```
import matplotlib.pyplot as plt

## Crea una serie di coordinate y corrispondenti alle x
x = [-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5]
y = []
for punto in x:
  y += [punto ** 2]

## Disegna il grafico
plt.plot(x,y)
plt.show()
### ATTENZIONE: il programma si blocca qui finche non si chiude il grafico
```

Primo grafico



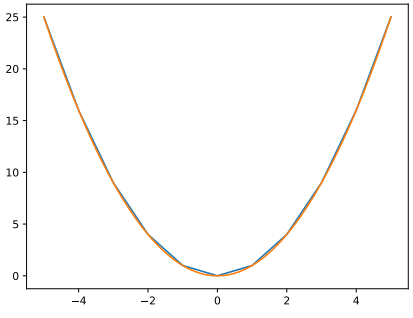
- La libreria matplotlib...
- Le liste non sono compatibili con le operazioni matematiche. Quindi non posso fare `x * 2` sperando di ottenere ogni elemento moltiplicato per due. Qui entra in gioco la libreria **NumPy**.

Incontriamo NumPy

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

## Crea una serie di coordinate y corrispondenti alle x
x1 = np.array([-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5])
x2 = np.linspace(-5, 5, 100)
y1 = x1 ** 2
y2 = x2 ** 2

## Disegna i grafici
plt.plot(x1,y1)
plt.plot(x2,y2)
plt.show()
```

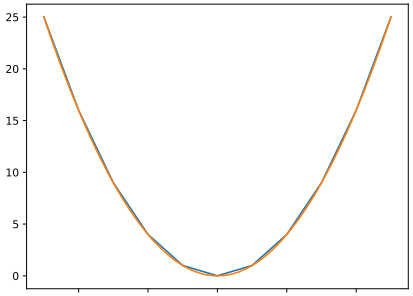
Incontriamo NumPy

- `array([...])` consente di creare una lista ottimizzata per le operazioni numeriche, o addirittura una matrice. I singoli elementi sono accessibili come una lista. È iterabile nel ciclo for.
- `linspace(min, max, points)` crea un array NumPy con *points* elementi compresi fra *min* e *max*, massimo escluso!
- Sugli array NumPy possono essere eseguite le operazioni matematiche. La libreria numpy mette a disposizione funzioni speciali che operano sugli array (`np.sqrt()`, `np.cos()`, ...)



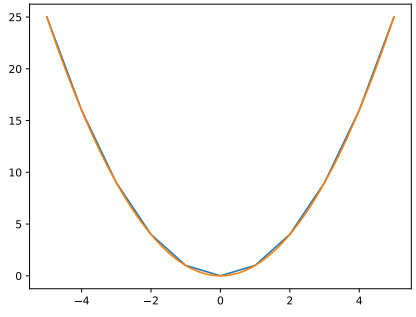
Incontriamo NumPy

- `array([...])` consente di creare una lista ottimizzata per le operazioni numeriche, o addirittura una matrice. I singoli elementi sono accessibili come una lista. È iterabile nel ciclo for.
- `linspace(min, max, points)` crea un array NumPy con *points* elementi compresi fra *min* e *max*, massimo escluso!
- Sugli array NumPy possono essere eseguite le operazioni matematiche. La libreria numpy mette a disposizione funzioni speciali che operano sugli array (`np.sqrt()`, `np.cos()`, ...)



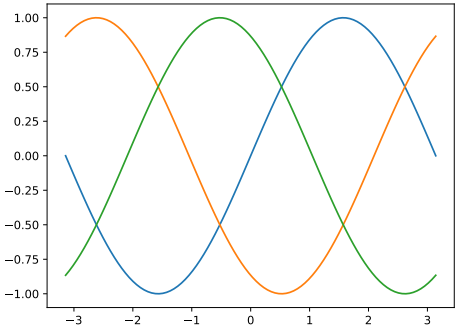
Incontriamo NumPy

- `array([...])` consente di creare una lista ottimizzata per le operazioni numeriche, o addirittura una matrice. I singoli elementi sono accessibili come una lista. È iterabile nel ciclo for.
- `linspace(min, max, points)` crea un array NumPy con *points* elementi compresi fra *min* e *max*, massimo escluso!
- Sugli array NumPy possono essere eseguite le operazioni matematiche. La libreria numpy mette a disposizione funzioni speciali che operano sugli array (`np.sqrt()`, `np.cos()`, ...)

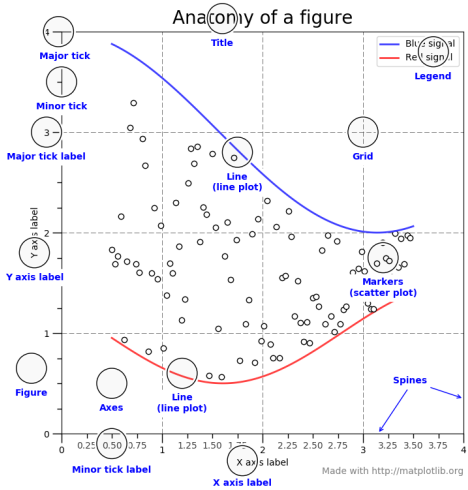


Esportazione immagini

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

x = np.linspace(-np.pi,np.pi,100)
y1 = np.sin(x)
y2 = np.sin(x - 2*np.pi/3)
y3 = np.sin(x + 2*np.pi/3)

plt.plot(x,y1)
plt.plot(x,y2)
plt.plot(x,y3)
plt.savefig('pippo.png')
# ma anche jpeg, pdf, svg...
```



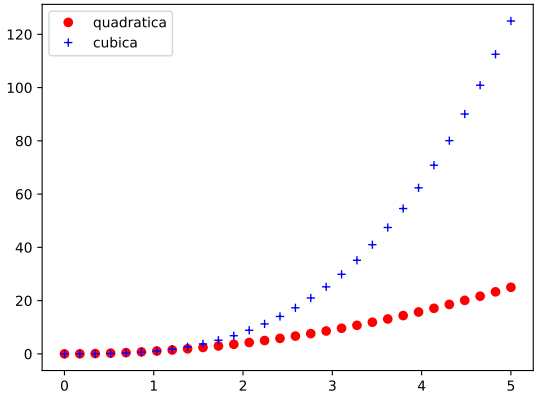
I termini di matplotlib



Decorazioni

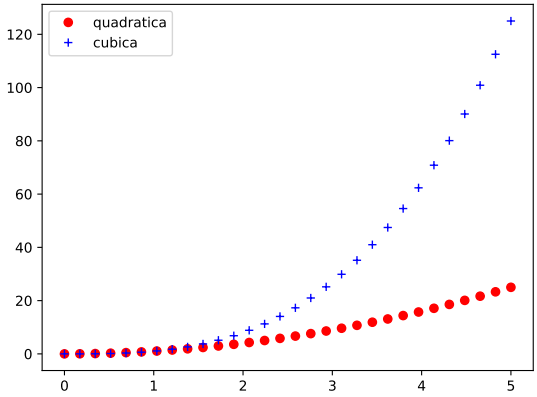
```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0,5,30)
y1 = x**2
y2 = x**3

## Disegn(i)amo il grafico
plt.plot(x, y1, 'ro', label='quadratica')
plt.plot(x, y2, 'b+', label='cubica')
plt.legend(loc='best')
plt.show()
```

Decorazioni



- Argomenti opzionali di `plot()`:
  - `label` per popolare la legenda;
  - `fmt` (*format*) per colore, stile di linea e marker;
- `legend()` costruisce il box di legenda. La posizione può essere forzata con l'opzione `loc` (*best*, *upper right*, *lower left*, *center*...).

Decorazioni



- Argomenti opzionali di `plot()`:
  - `label` per popolare la legenda;
  - `fmt` (*format*) per colore, stile di linea e marker;
- `legend()` costruisce il box di legenda. La posizione può essere forzata con l'opzione `loc` (*best*, *upper right*, *lower left*, *center*...).

Decorazioni

carattere	colore
b	blue
g	green
r	red
c	cyan
m	magenta
y	yellow
k	black
w	white

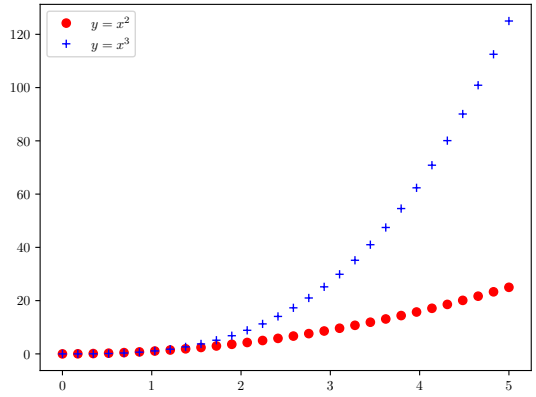
carattere	marker
o	cerchio
.	punto
,	pixel
x	×
+	+
*	★
s	■
^	△
D	◇

carattere	stile linea
–	continua (*solid*)
--	tratteggiata (*dashed*)
-.	punto-tratto (*dash-dot*)
:	tratteggiata breve (*dotted*)

Decorazioni – testo in LaTeX

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

plt.rc('text', usetex=True)

x = np.linspace(0,5,30)
y1 = x**2
y2 = x**3

plt.plot(x, y1, 'ro', label=r'$y=x^2$')
plt.plot(x, y2, 'b+', label=r'$y=x^3$')
plt.legend(loc='best')
plt.show()
```

Decorazioni – testo in LaTeX



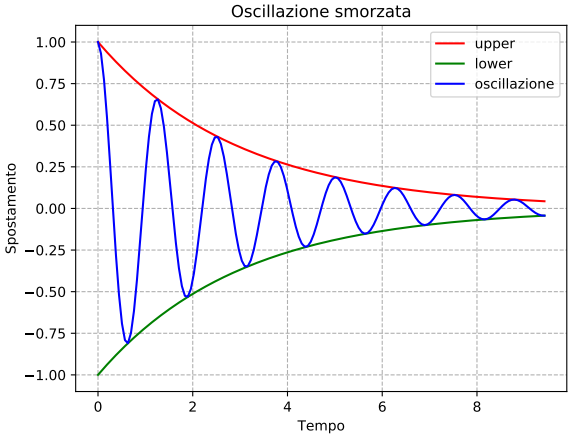
- `rc()` imposta i parametri globali di matplotlib (spessore linee, font, ...). Qui è usata per abilitare il rendering in LaTeX.
- Le formule matematiche vanno racchiuse fra `$...$`.
- Più informazioni su `https://en.wikibooks.org/wiki/LaTeX/Mathematics`.

Abbellimenti agli assi

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0,3*np.pi,150)
y1 = np.exp(-x/3)
y2 = -y1
y3 = np.cos(5*x) * y1

plt.title('Oscillazione smorzata')
plt.xlabel('Tempo')
plt.ylabel('Spostamento')

plt.plot(x, y1, 'r', label='upper')
plt.plot(x, y2, 'g', label='lower')
plt.plot(x, y3, 'b', label='oscillazione')
plt.legend(loc='best')
plt.grid(b=True, which='major', axis='both', linestyle='--')
plt.show()
```

Decorazioni – testo in LaTeX

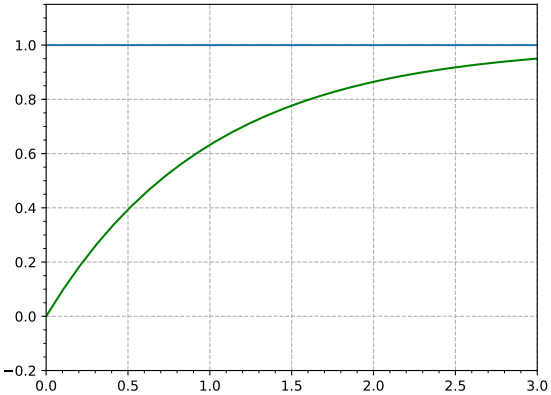
- `title()`, `xlabel()`, `ylabel()` impostano rispettivamente il testo del titolo e degli assi x ed y;
- `grid()` aggiunge la griglia verticale ed orizzontale, argomenti opzionali:
  - `which` - *'major'*, *'minor'*, *'both'*
  - `axis` - *'x'*, *'y'*, *'both'*
  - `linestyle` - analogo alla funzione `plot`



Modificare gli intervalli

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

x = np.linspace(0,3,30)
y = 1-np.exp(-x)

plt.plot(x, y, 'g-')
plt.axhline(y=1)
plt.grid(True, which='major', axis='both', linestyle='--')
plt.xlim(x.min(), x.max())
plt.ylim(y.min()-0.2, y.max()+0.2)
plt.minorticks_on()
plt.show()
```

Modificare gli intervalli



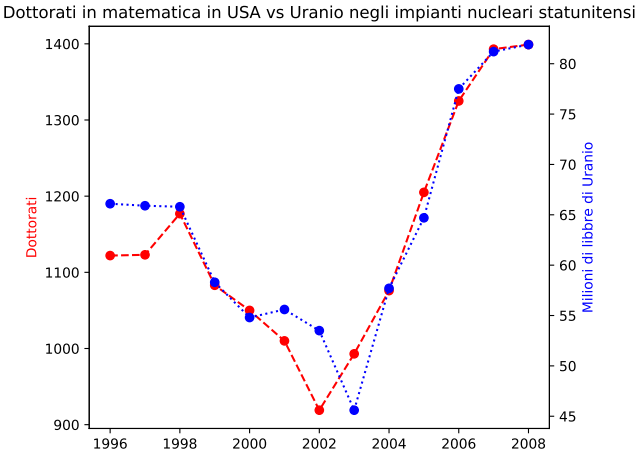
- `xlim()` e `ylim()` modificano gli estremi del grafico, anche singolarmente:
  - su xlim con `right=` o `left=`;
  - su ylim con `top=` o `bottom=`.
- `minorticks_on()` abilita la scala inferiore su entrambi gli assi;
- Inoltre: `axhline()` e `axvline()` disegnano linee orizzontali o verticali.

Twin axes, doppio asse y

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

# Dati da http://www.tylervigen.com/spurious-correlations
x = np.array([1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007,
    ↪ 2008])
doctor = np.array([1122, 1123, 1177, 1083, 1050, 1010, 919, 993, 1076, 1205, 1325, 1393,
    ↪ 1399])
uranium = np.array([66.1, 65.9, 65.8, 58.3, 54.8, 55.6, 53.5, 45.6, 57.7, 64.7, 77.5,
    ↪ 81.2, 81.9])

plt.title('Dottorati in matematica in USA vs Uranio negli impianti nucleari statunitensi')

plt.plot(x, doctor, 'ro--')
plt.ylabel('Dottorati', color='r')
plt.twinx(ax=None)
plt.ylabel('Milioni di libbre di Uranio', color='b')
plt.plot(x, uranium, 'bo:')
plt.tight_layout()

plt.show()
```

Twin axes, doppio asse y

- `twinx()` crea un secondo asse y indipendente su cui verranno riferiti i *plot* successivi;
- **Achtung!** twinx rende indipendenti anche le legende.



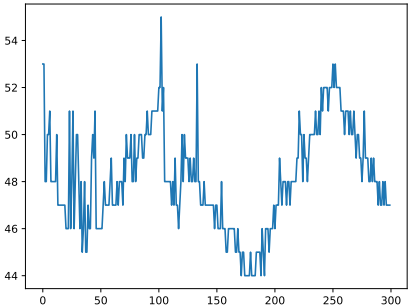
Lettura di CSV

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

temperature = np.loadtxt('data.csv', delimiter=',', usecols=(0,))
# Per default, conta da 0 e incrementa di 1
x = np.arange(len(temperature))

plt.plot(x,temperature)
plt.show()
```

data.csv

```
53,44,43,42,41,42,42,37
53,43,43,41,41,41,42,37
48,42,47,43,40,41,42,36
48,42,47,43,40,41,42,36
50,45,43,41,40,41,42,36
..,..,..,..,..,..,..,..
```

Lettura di CSV



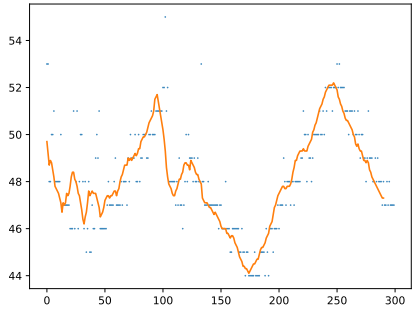
- `loadtxt()` restituisce un array NumPy raggruppato per righe;
- Per discriminare una singola colonna usare l'opzione `usecols`, indicando i numeri di colonna da estrapolare;
- Per ottenere i dati raggruppati per colonne, usare l'opzione `unpack=True`.

Lettura di CSV – condizionamento dei dati

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

def moving_average(data, window_size):
  window = np.ones(window_size)/float(window_size)
  return np.convolve(data, window, 'valid')

temperature = np.loadtxt('data.csv', delimiter=',', usecols=(0,))
# Per default, conta da 0 e incrementa di 1
x = np.arange(len(temperature))

media   = moving_average(temperature, 10)

plt.plot(x,temperature, '.', markersize=1)
plt.plot(x[:-9], media)
plt.show()
```

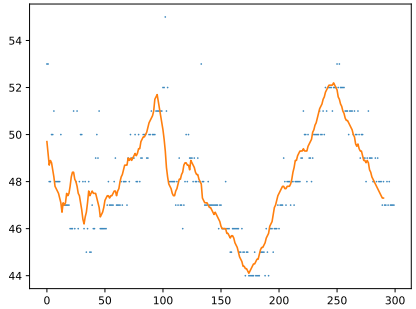
Lettura di CSV – condizionamento dei dati

- Un modo semplice per “ripulire" misure molto rumorose è effettuare una *media mobile*;
- Maggiore è la finestra, minore è la sensibilità a variazioni rapide;
- **Achtung!** l'operazione di media mobile riduce il numero di punti.



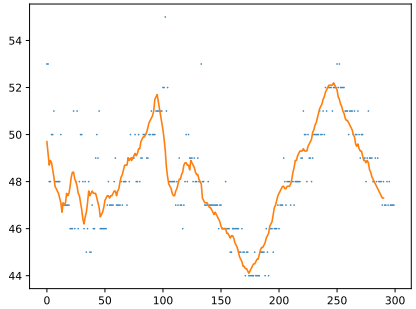
Lettura di CSV – condizionamento dei dati

- Un modo semplice per “ripulire" misure molto rumorose è effettuare una *media mobile*;
- Maggiore è la finestra, minore è la sensibilità a variazioni rapide;
- **Achtung!** l'operazione di media mobile riduce il numero di punti.

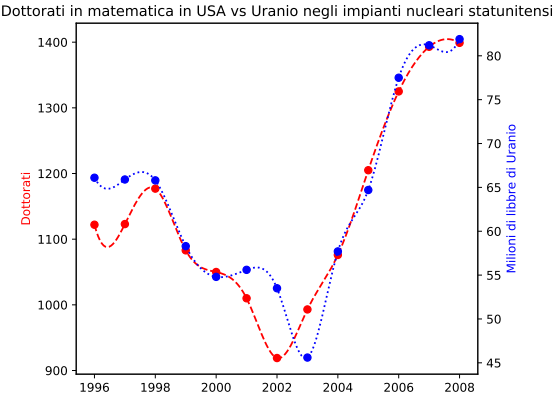


Lettura di CSV – condizionamento dei dati

- Un modo semplice per “ripulire" misure molto rumorose è effettuare una *media mobile*;
- Maggiore è la finestra, minore è la sensibilità a variazioni rapide;
- **Achtung!** l’operazione di media mobile riduce il numero di punti.

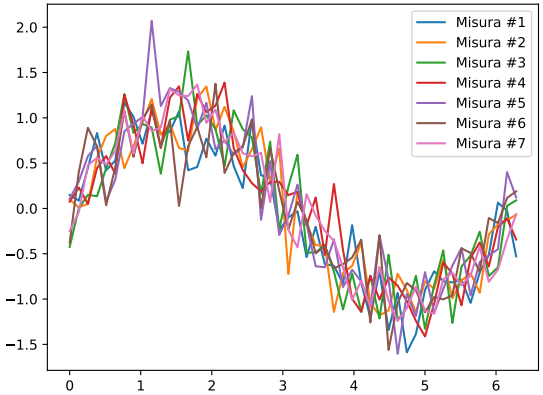


Un appunto alla presentazione dei dati – *interpolazione*



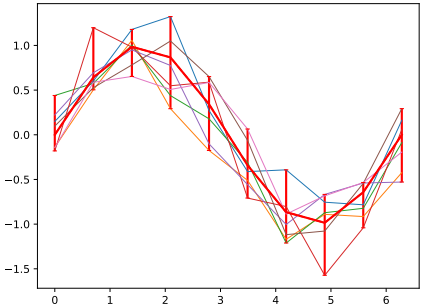
Libreria `scipy.interpolate`, funzione `spline(x,y,x_sp)`

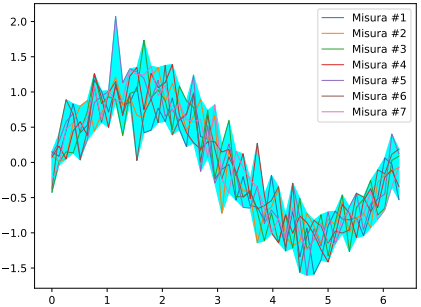
Barre di errore



Misure consecutive riportano valori diversi: sono affette da errori

Barre di errore





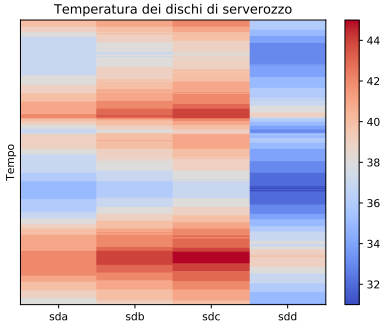

```
plt.errorbar(x,y,yerr=err,
capsize=2, color='red')
```

```
plt.fill_between(x, minimum,
maximum, color='cyan')
```

Tile plot

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

## Creiamo una serie di coordinate y corrispondenti alle x
temp = np.loadtxt('data.csv', delimiter=',', usecols=(4,5,6,7))

labels = ['sda', 'sdb', 'sdc', 'sdd']

plt.xticks(np.arange(len(labels)), labels)
plt.yticks([])
plt.ylabel('Tempo')
plt.title('Temperatura dei dischi di serverozzo')
plt.imshow(temp, aspect='auto', cmap='coolwarm')
plt.colorbar()

plt.show()
```

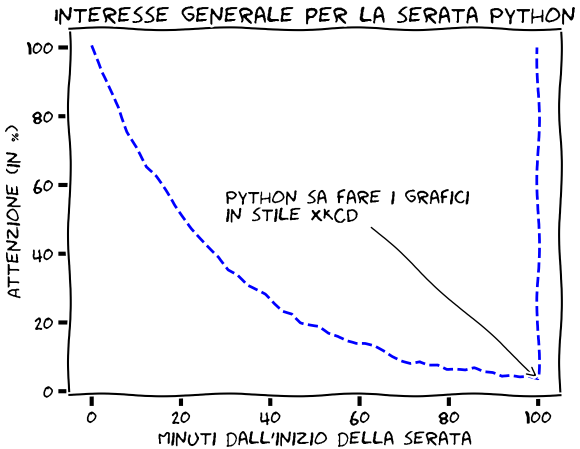
Tile plot



- `imshow()` è una funzione di plotting di immagini (*image show*), per mostrare *bitmap*.
- Associando una opportuna colormap si possono disegnare, ad esempio, mappe termiche o grafici di carico della CPU.

  `https://matplotlib.org/examples/color/colormaps_reference.html`

Per concludere



Link utili

- Quick reference python3
  https://perso.limsi.fr/pointal/_media/python:cours:mementopython3-english.pdf
- Manuale matplotlib
  https://matplotlib.org/faq/usage_faq.html#usage
- https://python4mpia.github.io/plotting/advanced.html
- https://realpython.com/python-matplotlib-guide/
- http://www.marmakoide.org/download/teaching/dm/dm-matplotlib.pdf
- https://www.slideshare.net/ewblen/introduction-to-matplotlib-for-data-analysis

Grazie per l'attenzione!

GOLEM - Gruppo Operativo Linux Empoli

GOLEM – Gruppo Operativo Linux Empoli
presso "La Vela – Margherita Hack"
via Magolo, 32 – 50053 Empoli (FI)
tutti i martedì sera dalle 21.30 alle 24.00

Crediti & licenza

Questa serata è offerta da giuliof (`giulio@glgprograms.it`)
a partire dal materiale di Fiore (FabLab Toscana) e con contributi di lucam
utilizzando un template LaTeX a cura di giomba.
Materiale rilasciato sotto GPL3 presso `golem.linux.it`