

- I packages di tipo scientifico non fanno parte del nucleo del linguaggio Python e devono quindi essere installati singolarmente
- I metodi di installazione sono vari e dipendono dal Sistema Operativo ospite. Ecco alcuni suggerimenti:
 - `ipython`, `numpy`, `scipy`, `matplotlib`
 - **Windows:** Installazione dal repository PyPI
 - **MacOS:** Usare *Homebrew* o installazione dal repository PyPI
 - **Linux:** Tutte le principali distribuzioni includono i packages, altrimenti si può installare dal repository PyPI
 - `astropy`:
 - **Windows:** Installazione dal repository PyPI
 - **MacOS:** Usare *Homebrew* o installazione dal repository PyPI
 - **Linux:** Tutte le principali distribuzioni includono il package, altrimenti si può installare dal repository PyPI
 - `astroquery`:
 - **Windows:** Installazione dal repository PyPI
 - **MacOS:** Usare *Homebrew* o installazione dal repository PyPI
 - **Linux:** Alcune distribuzioni includono il package, altrimenti si può installare dal repository PyPI

- PyPY è la fonte primaria di packages per Python:
<http://pypi.python.org>
- Normalmente non si usa direttamente, ma attraverso la procedura **pip**
- Esempi:
 - **pip install astropy**
 - **pip list**
 - **pip uninstall astropy**
- L'installazione può richiedere la presenza del compilatore C
- Suggerimenti:
 - Se il package esiste come package standard del S.O. (es.: msi [Windows], pkg [MacOS], rpm/deb [Linux]) è conveniente utilizzarlo.
 - Altrimenti utilizzare **pip**.
 - Se si vuole usare la versione più recente del package: usare **pip**

ipython è un ambiente per l'uso di python¹ che integra ed estende le caratteristiche interattive.

Esempio:

```
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
Type "copyright", "credits" or "license" for more information.

IPython 2.4.1 -- An enhanced Interactive Python.
?          -> Introduction and overview of IPython's features.
%quickref  -> Quick reference.
help       -> Python's own help system.
object?    -> Details about 'object', use 'object??' for extra details.

In [1]: import fibo
...
```

- L'ambiente Ipython:
 - completamento linee ed "history"
 - interazione con il S.O.
 - strumenti di introspezione migliorati
 - visualizzazione di grafici semplificata
 - comandi "magici"
- `ipython --pylab`
 - include numpy con il nome **np**
 - include in forma semplificata **matplotlib**

1) L'uso di ipython con altri linguaggi di programmazione e per applicazioni di calcolo parallelo non sarà approfondito in questa sede.

Il modulo `numpy.py` (solitamente importato con il nome **np**) fornisce:

- 1 Un oggetto **ndarray**: collezione di elementi omogenei, ad N dimensioni
- 2 Operazioni veloci su ndarray
- 3 Funzioni per algebra lineare
- 4 Trasformate di Fourier
- 5 Generatori di numeri casuali

Creazione di array:

```
>>> import numpy as np

>>> ar1 = np.array([1, 2, 3, 4, 5, 6, 7, 8])
>>> ar2 = np.array([[1, 2, 3, 4],[5, 6, 7, 8]])

>>> ar1.size, ar2.size
(8, 8)
>>> ar1.shape, ar2.shape
((8,), (2, 4))
>>> ar1.dtype, ar2.dtype
(dtype('int64'), dtype('int64'))

>>> ar5 = np.zeros((2,4))
>>> ar6 = np.ones((2,4,3))

>>> ar7 = np.identity(10)

>>> ar8 = np.linspace(0, np.pi, 5)

>>> # ar9 = np.loadtxt("datafile.txt")
>>> # ar9 = np.fromfile("datafile.txt")
```

array e “views”:

```
>>> ar3 = np.arange(1, 9)
>>> ar4 = ar3.reshape(2,4)
>>> ar3
array([1, 2, 3, 4, 5, 6, 7, 8])
>>> ar4
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
>>> ar3[3] = 0
>>> ar3
array([1, 2, 3, 0, 5, 6, 7, 8])
>>> ar4
array([[1, 2, 3, 0],
       [5, 6, 7, 8]])
```

Slice e selezione di porzioni di array:

```
>>> a=np.arange(150).reshape(10,5,3)
```

```
>>> a
```

```
array([[[ 0,  1,  2],
        [ 3,  4,  5],
        [ 6,  7,  8],
        [ 9, 10, 11],
        [12, 13, 14]],
```

```
       [[15, 16, 17],
        [18, 19, 20],
        [21, 22, 23],
        [24, 25, 26],
        [27, 28, 29]],
```

```
       ...
```

```
>>> a[1,:,:]
```

```
array([[15, 16, 17],
       [18, 19, 20],
       [21, 22, 23],
       [24, 25, 26],
       [27, 28, 29]])
```

```
>>> a[2:4,:,:]
```

```
array([[[30, 31, 32],
        [33, 34, 35],
        [36, 37, 38],
        [39, 40, 41],
        [42, 43, 44]],
```

```
       [[45, 46, 47],
        [48, 49, 50],
        [51, 52, 53],
        [54, 55, 56],
        [57, 58, 59]])
```

```
>>> a[:,1,1]
```

```
array([ 4, 19, 34, 49, 64, 79, 94, 109, 124, 139])
```

Semplici operazioni su array

package: numpy.py - 7

Prodotto array \times scalare:

```
>>> a = np.arange(0., 10, 0.3)
>>> a
array([ 0. , 0.3, 0.6, 0.9, 1.2, 1.5, 1.8, 2.1, 2.4, 2.7, 3. ,
        3.3, 3.6, 3.9, 4.2, 4.5, 4.8, 5.1, 5.4, 5.7, 6. , 6.3,
        6.6, 6.9, 7.2, 7.5, 7.8, 8.1, 8.4, 8.7, 9. , 9.3, 9.6,
        9.9])
>>> a*3
array([ 0. , 0.9, 1.8, 2.7, 3.6, 4.5, 5.4, 6.3, 7.2,
        8.1, 9. , 9.9, 10.8, 11.7, 12.6, 13.5, 14.4, 15.3,
        16.2, 17.1, 18. , 18.9, 19.8, 20.7, 21.6, 22.5, 23.4,
        24.3, 25.2, 26.1, 27. , 27.9, 28.8, 29.7])
```

Somma elemento \times elemento:

```
>>> b = np.arange(10., 0, -0.3)
>>> b
array([ 10. , 9.7, 9.4, 9.1, 8.8, 8.5, 8.2, 7.9, 7.6,
        7.3, 7. , 6.7, 6.4, 6.1, 5.8, 5.5, 5.2, 4.9,
        4.6, 4.3, 4. , 3.7, 3.4, 3.1, 2.8, 2.5, 2.2,
        1.9, 1.6, 1.3, 1. , 0.7, 0.4, 0.1])
>>> a+b
array([ 10., 10., 10., 10., 10., 10., 10., 10., 10., 10., 10.,
        10., 10., 10., 10., 10., 10., 10., 10., 10., 10.,
        10., 10., 10., 10., 10., 10., 10., 10., 10., 10.,
        10.])
```

Prodotto scalare ($\sum a_i b_i$):

```
>>> np.dot(a,b)
555.38999999999726
```

Prodotto vettoriale:

```
>>> prod = np.outer(a,b)
>>> prod.shape
(34, 34)
```

Tutte le funzioni di numpy operano normalmente elemento per elemento

- Funzioni trigonometriche
- Funzioni iperboliche
- Arrotondamento (round(), floor(), ceil(), ...)
- Somme, prodotti, differenze
- Esponenziali, logaritmi, bessel ...
- Funzioni su elementi floating point
- Funzioni aritmetiche
- Funzioni su numeri complessi

Attenzione ai nomi di funzione comuni:

```
>>> import numpy as np
>>> a=np.array([[1,2,3],[4,5,6]])
>>> max(a)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: The truth value of an array with more than one element is ambiguous. U
>>> np.max(a)
6
>>>
```

Per esplorare il contenuto dei sotto moduli facciamo riferimento al manuale:

- **numpy.ma**: manipolazione di array con “maschere”
- **numpy.linalg**: algoritmi di algebra lineare
- **numpy.matlib**: operazioni su matrici
- **numpy.random**: numeri e distribuzioni casuali
- **numpy.fft**: algoritmi per trasformate di Fourier discrete

Problema: verificare se esistano correlazioni significative fra le nascite e la fasi della luna

Dati di ingresso: numero di nati registrati per ogni giorno dal 1/6/1978 al 15/11/1986 (Anagrafe del comune di Firenze)

file: `dati_nascite.dat`

```
791124 0 0 0
791125 0 1 1
791126 0 0 0
.....
```

File di 3090 linee contenenti: data, numero maschi registrati, numero femmine registrate, numero totale registrati.

Nelle prossime pagine vedremo prima come preparare alcuni strumenti utili e poi come procedere nella soluzione del problema

Esempio di uso di NumPy - segue

hands on 1 - 11

file: luna.py

```
import math

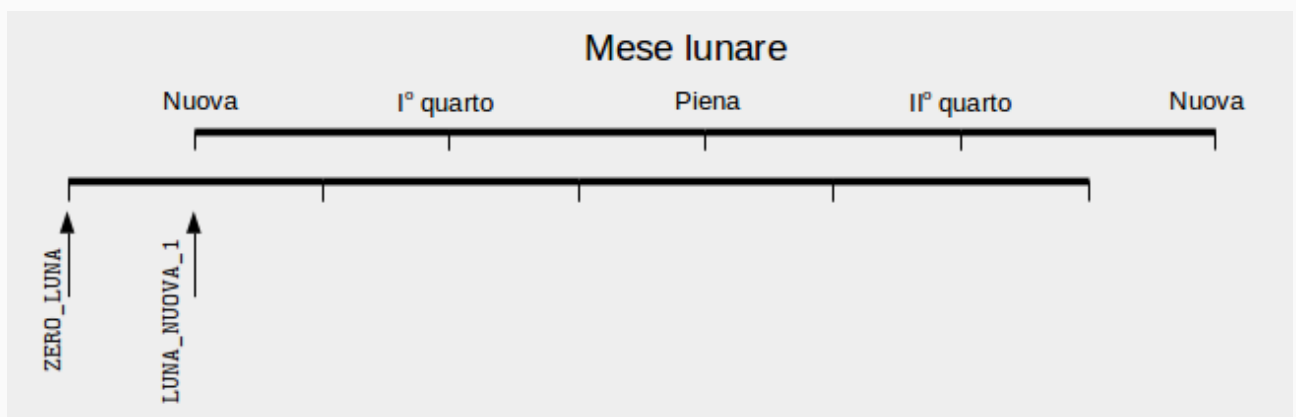
# Durata media mese lunare: 29 giorni, 12 ore, 43 minuti, 11 secondi
MESE_LUNARE = 29*86400 + 12*3600 + 43*60 + 11 # in secondi
QUARTO_LUNARE = MESE_LUNARE*0.25 # Durata di ogni quarto

# Prima luna nuova dopo 1/1/1970
LUNA_NUOVA_1 = 660262.0 # in secondi
ZERO_LUNA = LUNA_NUOVA_1 - QUARTO_LUNARE*0.5

def fase_luna(nsec): # fase lunare per l'istante assegnato.
    # 0: nuova, 1:1° quarto, 2:piena, 3:ultimo quarto.
    fase_sec = math.fmod(nsec-ZERO_LUNA, MESE_LUNARE)
    fase = int(fase_sec/QUARTO_LUNARE)
    return fase
```

Note:

- 1 Per i dati relativi al tempo si utilizzano le funzioni del modulo `time` che hanno come base dei tempi il numero di secondi dal 1/1/1970 (`sec1970`).
- 2 Il valore di `LUNA_NUOVA_1` è stato determinato usando una tavola per trovare data e ora di una luna nuova “conveniente” e convertendo il valore in `sec1970`.
- 3 Il valore `ZERO_LUNA` è anticipato di un quarto di fase in modo da considerare appartenenti al primo quarto le date in un intorno centrato sull'istante della luna nuova (vedi figura)



Esempio di uso di NumPy - segue

hands on 1 - 12

file: `tt.py`

(vedi: `time.mktime`)

```
import time

def toseconds(aammgg): # Converte data nel formato "aammgg" in
                        # secondi da 1/1/1970 (rif. mezzogiorno)
    anno = int(aammgg[:2])+1900
    mese = int(aammgg[2:4])
    giorno = int(aammgg[4:6])
    tt = (anno, mese, giorno, 12, 0, 0, 0, 0, 0)
    return time.mktime(tt)
```

... e adesso procediamo lanciando python:

(vedi: `np.loadtxt`)

```
>>> import numpy as np
>>> import tt, luna
>>> filename = "dati_nascite.dat"
>>> dati = np.loadtxt(filename, dtype=np.int_, # Leggi dati
...                  converters={0: tt.toseconds}, unpack=True)
```

(vedi: `np.vectorize`)

```
>>> cvt_luna = np.vectorize(luna.fase_luna)
>>> faseluna = cvt_luna(dati[0]) # Determina fase della luna
```

```
>>> m0 = np.array([1 if x==0 else 0 for x in faseluna])
>>> m1 = np.array([1 if x==1 else 0 for x in faseluna])
>>> m2 = np.array([1 if x==2 else 0 for x in faseluna])
>>> m3 = np.array([1 if x==3 else 0 for x in faseluna])
```

```
>>> nati0 = np.dot(m0,dati[3])
>>> nati1 = np.dot(m1,dati[3])
>>> nati2 = np.dot(m2,dati[3])
>>> nati3 = np.dot(m3,dati[3])
```

```
>>> totnati = nati0+nati1+nati2+nati3
>>> nati_gg = float(totnati)/len(faseluna)
```

```
f_osserv = [nati0, nati1, nati2, nati3]
f_attesa = [np.sum(m0)*nati_gg, np.sum(m1)*nati_gg,
            np.sum(m2)*nati_gg, np.sum(m3)*nati_gg]
```

Esempio di uso di NumPy - segue

hands on 1 - 13

Ho raccolto in un file le istruzioni mostrate interattivamente

file: `nati.py`

```
# Calcolo frequenza di nascita nelle fasi lunari

import numpy as np
import tt
import luna

filename = "dati_nascite.dat"
dati = np.loadtxt(filename, dtype=np.int_, # Leggi dati
                  converters={0: tt.toseconds}, unpack=True)

cvt_luna = np.vectorize(luna.fase_luna)
faseluna = cvt_luna(dati[0]) # Determina fase della luna

m0 = np.array([1 if x==0 else 0 for x in faseluna])
m1 = np.array([1 if x==1 else 0 for x in faseluna])
m2 = np.array([1 if x==2 else 0 for x in faseluna])
m3 = np.array([1 if x==3 else 0 for x in faseluna])

nati0 = np.dot(m0, dati[3])
nati1 = np.dot(m1, dati[3])
nati2 = np.dot(m2, dati[3])
nati3 = np.dot(m3, dati[3])

totnati = nati0+nati1+nati2+nati3

nati_gg = float(totnati)/len(faseluna)

f_osserv = [nati0, nati1, nati2, nati3]
f_attesa = [np.sum(m0)*nati_gg,
            np.sum(m1)*nati_gg,
            np.sum(m2)*nati_gg,
            np.sum(m3)*nati_gg]

chi_sq = sum([(x-y)**2/y for (x,y) in zip(f_osserv, f_attesa)])

p0 = float(nati0)/totnati
p1 = float(nati1)/totnati
p2 = float(nati2)/totnati
p3 = float(nati3)/totnati
```

Esempio di uso di NumPy - segue

hands on 1 - 14

Per utilizzarle con ipython:

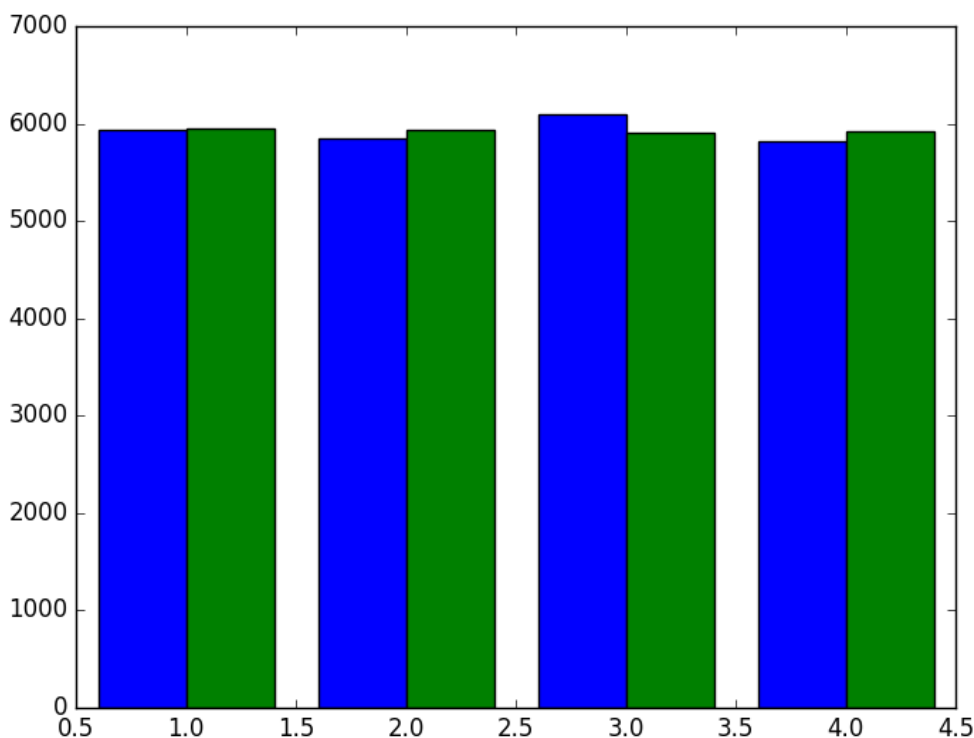
```
$ ipython --pylab
```

```
.....
```

```
In [1]: %run nati.py
```

```
In [2]: plt.bar(np.arange(0.6, 4.5, 1), f_osserv, width=0.4, color="blue")
```

```
In [3]: plt.bar(np.arange(1, 4.5, 1), f_attesa, width=0.4, color="green")
```



Le differenze sono significative?

```
In [5]: sum([(x-y)**2/y for (x,y) in zip(f_osserv, f_attesa)])
```

```
Out[5]: 9.1029261874500467
```

Tavola distribuzione CHI-QUADRATO

Gradi di libertà	Livello di Probabilità a									
	1.00	0.99	0.95	0.90	0.25	0.10	0.05	0.025	0.01	0.005
1				0.02	1.32	2.71	3.84	5.02	6.64	7.88
2	0.01	0.02	0.10	0.21	2.77	4.61	5.99	7.38	9.21	10.60
3	0.07	0.12	0.35	0.58	4.11	6.25	7.82	9.35	11.35	12.84
4	0.21	0.30	0.71	1.06	5.39	7.78	9.49	11.14	13.28	14.86

matplotlib è una libreria grafica 2D usabile in programmi python e in particolare nell'ambiente ipython.

- Produce grafici “publication quality”
- Uso interattivo
- Modellata sui comandi grafici di *matlab*
- Configurazione di default utilizzabile con il minimo di dettagli
- Tutti i dettagli possono essere controllati
- Interfaccia OO per l'uso avanzato
- Toolkit per grafici 3D

matplotlib si presta all'uso interattivo con ipython:

```
$ ipython --pylab
```

```
....
```

```
In [1]: %run nati.py
```

```
In [2]: plt.plot(dati[3])
```

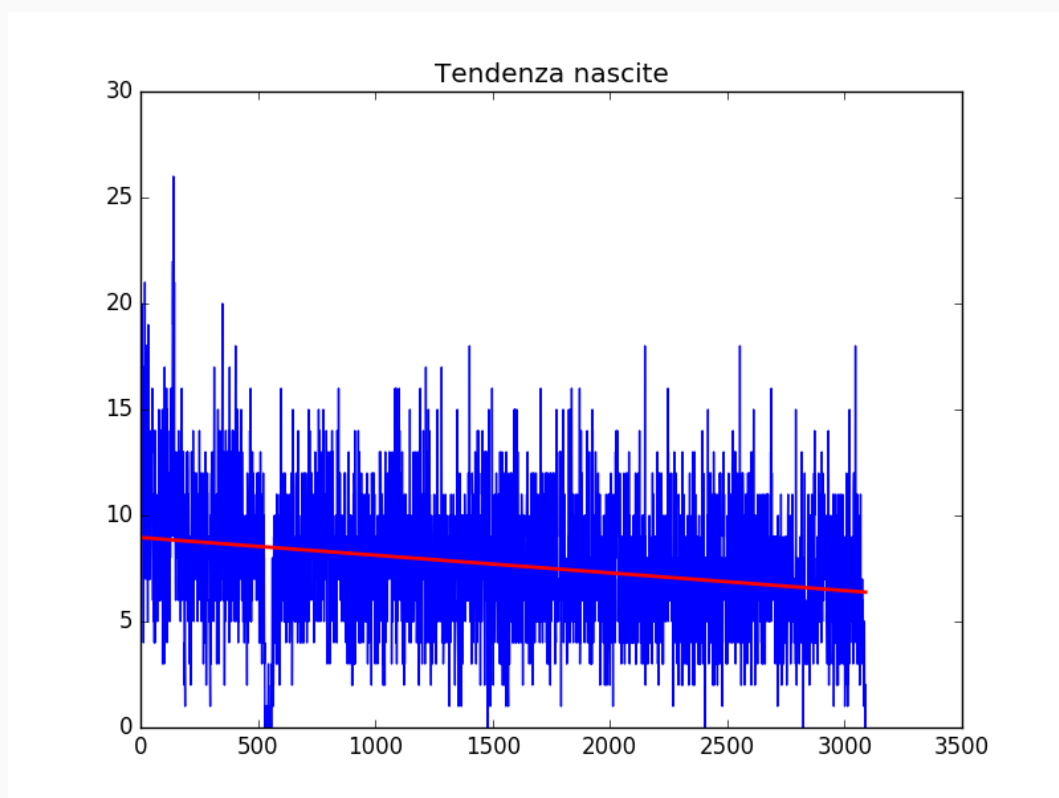
(vedi: **polyfit**)

```
In [3]: x = np.arange(len(dati[3]))
```

```
In [4]: a,b = np.polyfit(x, dati[3], 1)
```

```
In [5]: plt.plot((0, x[-1]),(b, a*x[-1]+b),linewidth=2,color="red")
```

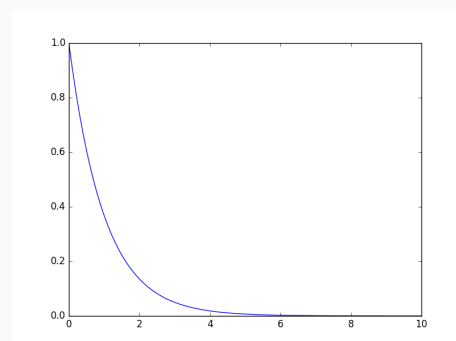
```
In [6]: plt.title("Tendenza nascite")
```



Esempi:

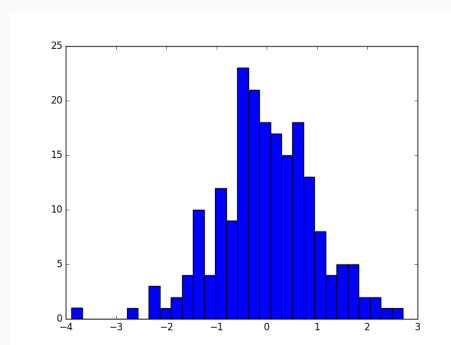
Funzione per punti

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> a = np.linspace(0,10,100)
>>> b = np.exp(-a)
>>> plt.plot(a,b)
>>> plt.show()
```



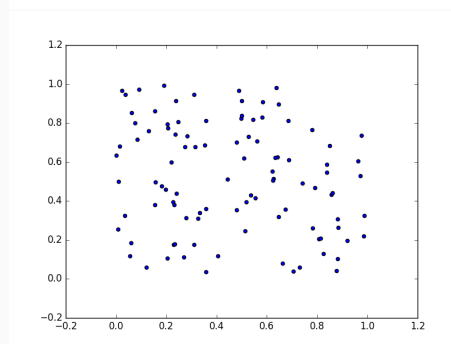
Istogramma

```
>>> from numpy.random import normal
>>> x = normal(size=200)
>>> plt.hist(x,bins=30)
>>> plt.show()
```



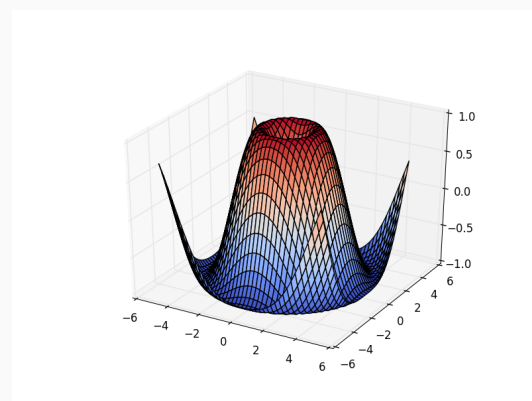
Scatter plot

```
>>> from numpy.random import rand
>>> a = rand(100)
>>> b = rand(100)
>>> plt.scatter(a,b)
>>> plt.show()
```



Superficie 3D

```
>>> from matplotlib import cm
>>> from mpl_toolkits.mplot3d import Axes3D
>>> fig = plt.figure()
>>> ax = fig.gca(projection="3d")
>>> X = np.arange(-5, 5, 0.25)
>>> Y = np.arange(-5, 5, 0.25)
>>> X, Y = np.meshgrid(X, Y)
>>> R = np.sqrt(X**2 + Y**2)
>>> Z = np.sin(R)
>>> surf = ax.plot_surface(X, Y, Z, rstride=1,
>>>                        cstride=1, cmap=cm.coolwarm)
>>> plt.show()
```



File: `wind_2d.stat`:

mappa spaziale di dati sul vento intorno a Mt.Graham

```
### GRID SIZE X, GRID SIZE Y, START EXIT, END EXIT, EXIT NUM, MIN WIND, MAX WIND #
      100      100      4      10      7 1.4624E-02 5.78828
#  I   J   XLAT   XLON   ZS   W.MOD. W.ANG. W.X (UT) W.Y(VT)
   2   2   32.656 -109.941 1912.875 1.7029 62.383 -1.5089 -0.78941
   2   3   32.657 -109.941 1916.125 1.5125 71.936 -1.4380 -0.46900
   2   4   32.658 -109.941 1924.812 1.4617 76.891 -1.4236 -0.33153
   2   5   32.659 -109.941 1935.500 1.4756 85.017 -1.4700 -0.12814
.....
```

File: `plot_wind_2d.py`:

```
from __future__ import print_function

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

##### Dati di ingresso
DELTAX=0.1 # Risoluzione spaziale
SYMx=51-1 # Posizione centro X
SYMy=51-1 # Posizione centro Y
filename="wind_2d.stat"
#####

data = np.loadtxt(filename, skiprows=3, unpack=True) # Lettura dati

ipoints = long(np.max(data[0])-np.min(data[0])+1) # Dimensione X
jpoints = long(np.max(data[1])-np.min(data[1])+1) # Dimensione Y

ZS = data[4].reshape(ipoints, -1).transpose() # Trasforma in mappa
WM = data[5].reshape(ipoints, -1).transpose() # Trasforma in mappa
WUT = data[7].reshape(ipoints, -1).transpose() # Trasforma in mappa
WVT = data[8].reshape(ipoints, -1).transpose() # Trasforma in mappa

arrayI = np.arange(0, ipoints)*DELTAX # Scale spaziali
arrayJ = np.arange(0, jpoints)*DELTAX
maxI = np.max(arrayI)
maxJ = np.max(arrayJ)
```

File: plot_wind_2d.py - segue:

```
im = plt.imshow(WM, interpolation="bilinear", origin="lower", # Immagine
                extent=[0, maxI, 0, maxJ], cmap=cm.hot_r)

cbar = plt.colorbar(im, orientation="vertical") # Barra con scala colori
cbar.set_label("Wind speed (m/s)")

maxzs = np.max(ZS)
minzs = np.min(ZS)
levels = np.linspace(minzs, maxzs, 25) # Livelli per contorni

plt.contour(arrayI, arrayJ, ZS, levels, colors="k", origin="lower",
            linewidths=0.5)
plt.plot([arrayI[SYMX]], [arrayJ[SYMY]], "o", # Punto centrale
        color="black", ms=8)
plt.quiver(arrayI[:,5], arrayJ[:,5], WUT[:,5,:5], WVT[:,5,:5], # Frecce
          headwidth=6, headlength=6)

plt.title("Wind speed - hour 007 or hour 000 MST")
plt.xlabel("Km")
plt.ylabel("Km")
plt.tight_layout()
#plt.show()
plotfile = "wind_speed.png"
plt.savefig(plotfile, dpi=200)
print("Creato file con plot:", plotfile)
```

