Corso Python per docenti

Lezione A08 – Numpy

Alberto Montresor

Università di Trento

2021/02/09

Acknowledgments: Stefano Teso, Numpy Documentation
http://disi.unitn.it/~teso/courses/sciprog/python_appendices.html

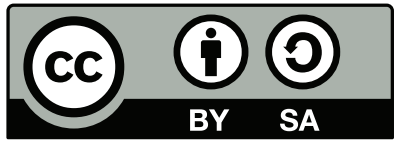


Table of contents

Cos'è Numpy?

Numpy è una libreria grutuita che permette di effettuare computazioni efficienti su scalari, vettori, matrici e più generalmente tensori.

Caratteristiche

- Indicizzazione flessibile e manipolazione di dati arbitrari multidimensionali.
- Molti routine numeriche (algebra lineare, analisi di Fourier, ottimizzazione globale, etc.) utilizzando una varietà di algoritmi;
- Interfaccia python per un gran numero di librerie sottostanti, che contengono implementazioni efficienti di algoritmi numerici.
- Supporto per generazione di numeri casuali da un gran numero di distribuzioni

Alcuni link

Website di Numpy

`http://www.numpy.org/`

Dcoumentazione ufficiale

`https://docs.scipy.org/doc/numpy/`

Source code

`https://github.com/numpy/numpy`

Importare Numpy

Il modo classico per importare Numpy è il seguente:

```
import numpy as np
```

Esistono sottomoduli specifici che vanno importati in questo modo (per esempio, il modulo per l'algebra lineare `linalg`):

```
import numpy.linalg as la
```

La classe array `ndarray`

Un `ndarray` è un array n-dimensionale (a.k.a. tensore)

- I concetti incontrati in analisi e algebra possono essere implementati come array: i vettori sono array 1D, le matrici sono array 2D.
- Dato un array, la sua dimensionalità e forma può essere ispezionata utilizzando gli attributi: `shape`, `ndim` e `dtype`:

```
import numpy as np
x = np.zeros(10)
print(x)          [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
print(x.shape)    (10,)
print(x.ndim)     1
print(x.dtype)    float64
```

Creare array di varie forme

Creare array da zero può essere utile per inizializzazioni varie:

- Creare un array contenente 0: `np.zeros()`
- Creare un array contenente 1: `np.ones()`

```
import numpy as np
print( np.zeros(2) )
```

```
[ 0.  0.]
```

```
print( np.ones( (2,2) ) )
```

```
[[ 1.  1.]
 [ 1.  1.]]
```

```
print( np.zeros( (2,2,2) ) )
```

```
[[[ 0.  0.]
  [ 0.  0.]]

 [[ 0.  0.]
  [ 0.  0.]]]
```

Creare array di varie forme

Per creare una matrice diagonale, utilizzate i metodi `diag()` e `ones()`:

```
import numpy as np
print( np.diag( np.ones(3) )
```

```
[[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
```

Creare array di varie forme

L'analogo di `range()` in Numpy è `np.arange()` (molto più potente)

```
import numpy as np
print( np.arange(10,100,10) )
print( np.arange(0.0,1.0,0.1))
print( np.linspace(0.0,1.0,11))
print( np.diag( np.arange(5) ) )
```

```
10 20 30 40 50 60 70 80 90]
[ 0.   0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  0.9]
[ 0.   0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  0.9  1. ]
[[0 0 0 0 0]
 [0 1 0 0 0]
 [0 0 2 0 0]
 [0 0 0 3 0]
 [0 0 0 0 4]]
```

Creare array random

- Un vettore di 5 elementi distribuiti uniformemente fra $[0, \ldots, 10[$
- Un vettore di 5 elementi con distribuzione normale $\mu = 1$ and $\sigma = 0.2$
- Una matrice 3x3 da una distribuzione uniforme in $\{0, 1\}$

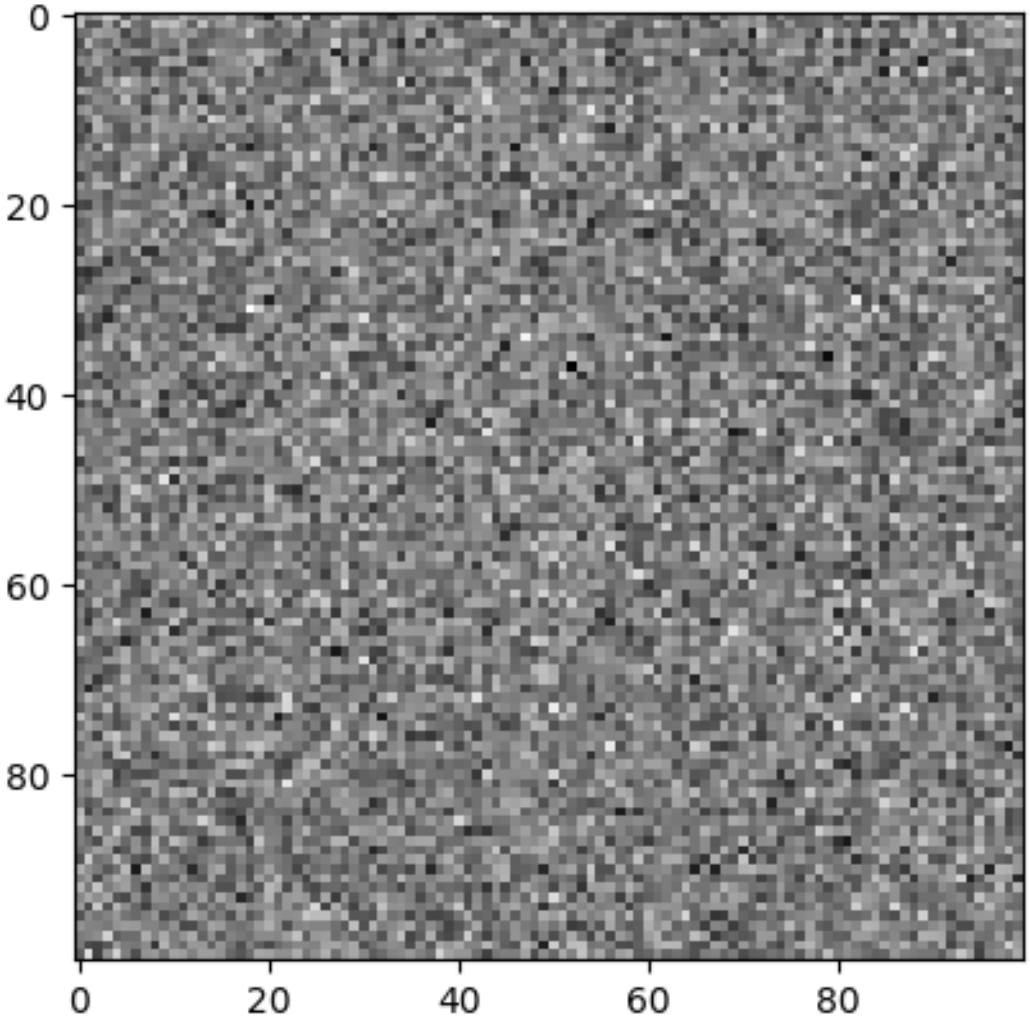
```
import numpy as np
print(np.random.uniform(0, 10, size=5))
print(np.random.normal(1, 0.2, size=5))
print(np.random.randint(0, 2, size=(3, 3)))
```

```
[ 7.91700684  7.41652128  8.1393401   0.8123227   5.50427964]

[ 1.14191823  0.89203955  1.09505607  0.8081311   0.82282836]

[[0 0 0]
 [0 1 1]
 [0 1 1]]
```

Creare array random

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

np.random.seed(0)

rm = np.random.normal(0, 1, size=(100, 100))

plt.imshow(rm, cmap="gray", interpolation="nearest")
plt.savefig("random-matrix.png")
```

Creare array random



Da pandas a numpy e ritorno

È veramente facile "convertire" una `Series` o un `DataFrame` in an array: infatti, Pandas è basato su array Numpy e l'array sottostante è accessibile con l'attributo `values`.

```
import pandas as pd
import numpy as np
iris = pd.read_csv("iris.csv")
print(type(iris.PetalWidth.values))
```

```
<type 'numpy.ndarray'>
```

Re-shaping, un-raveling

- È possibile passare da vettori a matrici/tensori con `reshape()`
- È possibile passare da matrici/tensori a vettori con `ravel()`

```
import numpy as np
x = np.arange(9)
print(x)
y = x.reshape((3, 3))
print(y)


z = y.ravel()
print(z)
```

```
[0 1 2 3 4 5 6 7 8]

[[0 1 2]
 [3 4 5]
 [6 7 8]]

[0 1 2 3 4 5 6 7 8]
```

Iterare su un vettore

È possibile iterare sugli array in molti modi:

- Iterando su tutti gli elementi di un `ndarray`

```
for element in A.flat:
    print(element)
```

- Iterando sulla prima dimensione:

```
for row in A:
    print(row)
```

Indicizzazione

- `x[i,j]` estrae gli elementi in riga `i`, colonna `j` – come nell'algebra lineare.
- Potete usare la notazione `:` per selezionare elementi specifici su un asses (e.g. una colonna, una riga, un sotto-array).

```
import numpy as np
x = np.arange(9).reshape((3, 3))
print(x)
print(x[0,:]) # First row
print(x[:,0]) # First column
print(x[1:3, :]) # Subarray
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
[0 1 2]
[0 3 6]
[[3 4 5]
 [6 7 8]]
```

Indicizzazione

La stessa sintassi si usa per vettori multidimensionali:

```
import numpy as np
x = np.arange(5**5).reshape((5,)*5)
print(x.shape)
print(x[0,0,:,:,0])
```

```
(5, 5, 5, 5, 5)
[[  0   5  10  15  20]
 [ 25  30  35  40  45]
 [ 50  55  60  65  70]
 [ 75  80  85  90  95]
 [100 105 110 115 120]]
```

Vettori multidimensionali

Può essere difficile visualizzare mentalmente operazioni su matrici n -dimensionali, quando n è maggiore di 3, a meno che non abbiano una semantica concreta.

Esempio

Supponiamo che un array 4D contenga le prestazioni di diversi algoritmi di allineamento di sequenza su più cluster di sequenze in più DB:

- Il primo asse è l'indice dell'algoritmo
- Il secondo asse è l'indice del database
- Il terzo asse è l'indice di un gruppo di sequenze
- Il quarto asse è una delle tre misure delle prestazioni: precisione, richiamo e accuratezza

Vettori multidimensionali

Il tensore prende questa forma:
`performances[alg][db][cluster][measure]`

Possiamo definire costanti simboliche per identificare gli indici di una colonna. Ad esempio,

```
NEEDLEMAN = 0 # Needleman-Wunsch
SMITH = 1     # Smith-Waterman


PRECISION = 0
RECALL = 1
ACCURACY = 2
```

Per estrarre l'accuratezza dell'algoritmo Needleman-Wunsch su tutti i database e tutte le sequenze di cluster, potete fare:

```
print(performances[NEEDLEMAN,:,:,ACCURACY])
```

Broadcasting

Le operazioni fra scalari e vettori sono mandate in broadcast, come in Pandas:

```
import numpy as np
x = np.arange(5)*10
print(x)                    [ 0 10 20 30 40]
y = -x
print(y)                    [  0 -10 -20 -30 -40]
z = x+y
print(z)                    [0 0 0 0 0]
```

Broadcasting

Le operazioni possono anche aggiornare sotto-array:

```
import numpy as np
x = np.arange(16).reshape((4, 4))
print(x)
x[1:3,1:3] += 100
print(x)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]

[[  0   1   2   3]
 [  4 105 106   7]
 [  8 109 110  11]
 [ 12  13  14  15]]
```

Dimensioni compatibili

Le operazioni fra vettori di forma differente ma di dimensione compatibile vengono mandate in broadcast facendo "matching" fra l'ultima dimensione (più a destra). E.g, la somma fra una matrice `x` e un vettore `y` fa il brodcast di `y` su tutte le righe di `x`

```
import numpy as np
x = np.arange(9).reshape((3, 3))
print(x)

y = np.arange(3)
print(y)

z = x + y
print(z)
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]

[0 1 2]

[[ 0  2  4]
 [ 3  5  7]
 [ 6  8 10]]
```

Dimensioni incompatibili

Le operazioni tra vettori di dimensioni incompatibili danno origine ad errori:

```
>>> np.arange(3) + np.arange(2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: operands could not be broadcast together with
shapes (3,) (2,)
```

Funzioni matematiche

Numpy contiene molte funzioni matematiche, che possono essere applicate a tensori (oltre che a valori scalari)

`https://docs.scipy.org/doc/numpy/reference/routines.math.html`

Trigonometric functions

`sin` (x, /[, out, where, casting, order, ...])	Trigonometric sine, element-wise.
`cos` (x, /[, out, where, casting, order, ...])	Cosine element-wise.
`tan` (x, /[, out, where, casting, order, ...])	Compute tangent element-wise.
`arcsin` (x, /[, out, where, casting, order, ...])	Inverse sine, element-wise.
`arccos` (x, /[, out, where, casting, order, ...])	Trigonometric inverse cosine, element-wise.
`arctan` (x, /[, out, where, casting, order, ...])	Trigonometric inverse tangent, element-wise.
`hypot` (x1, x2, /[, out, where, casting, ...])	Given the “legs” of a right triangle, return its hypotenuse.
`arctan2` (x1, x2, /[, out, where, casting, ...])	Element-wise arc tangent of `x1/x2` choosing the quadrant correctly.
`degrees` (x, /[, out, where, casting, order, ...])	Convert angles from radians to degrees.
`radians` (x, /[, out, where, casting, order, ...])	Convert angles from degrees to radians.
`unwrap` (p[, discont, axis])	Unwrap by changing deltas between values to 2*pi complement.
`deg2rad` (x, /[, out, where, casting, order, ...])	Convert angles from degrees to radians.
`rad2deg` (x, /[, out, where, casting, order, ...])	Convert angles from radians to degrees.

Funzioni matematiche

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

# The x values
x = 2 * np.pi * np.arange(0.0,1.0,0.01)

# The y values, uncorrupted
y = np.sin(x)
plt.plot(x, y)

# The y values, now corrupted by Gaussian noise
y += np.random.normal(0, 0.05, size=100)
plt.plot(x, y)

plt.savefig("sin.png")
```

Funzioni matematiche

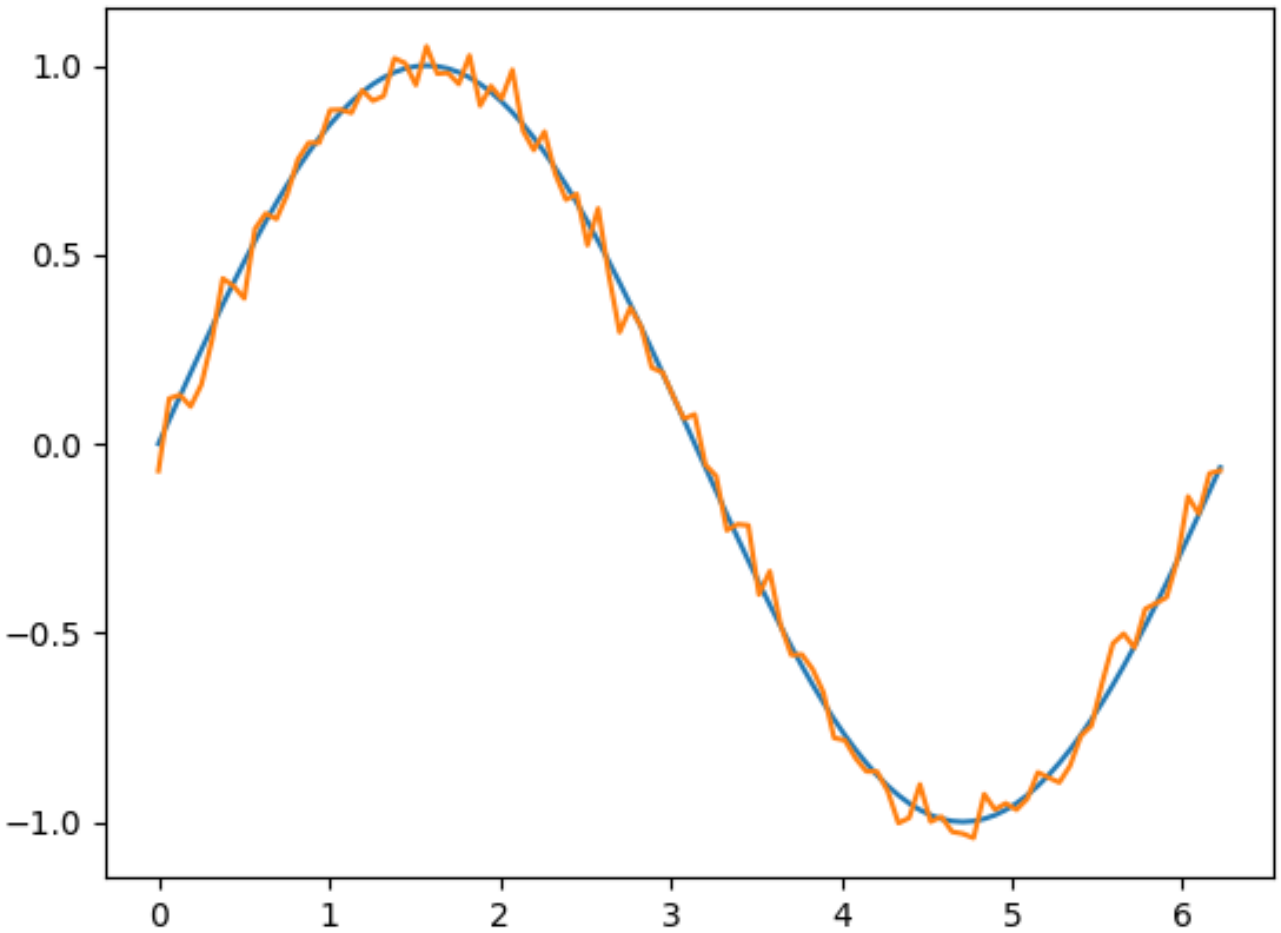


Table of contents

Cos'è MatPlotLib?

MatPlotLib è una libreria di plotting per Python e per Numpy

Due approcci:

- La libreria generale MatPlotLib fornisce il framework generale per disegnare un gran numero di figure
- Il modulo MatPlotLib `pyplot` fornisce un approccio dichiarativo per fare plot, molto simile alla sintassi di MatLab

`pyplot` è più semplice del framework generale, e permette di produrre plot molto velocemente.
Tuttavia, il vero potere è dato dal framework generale.

MatPlotLib plotting script

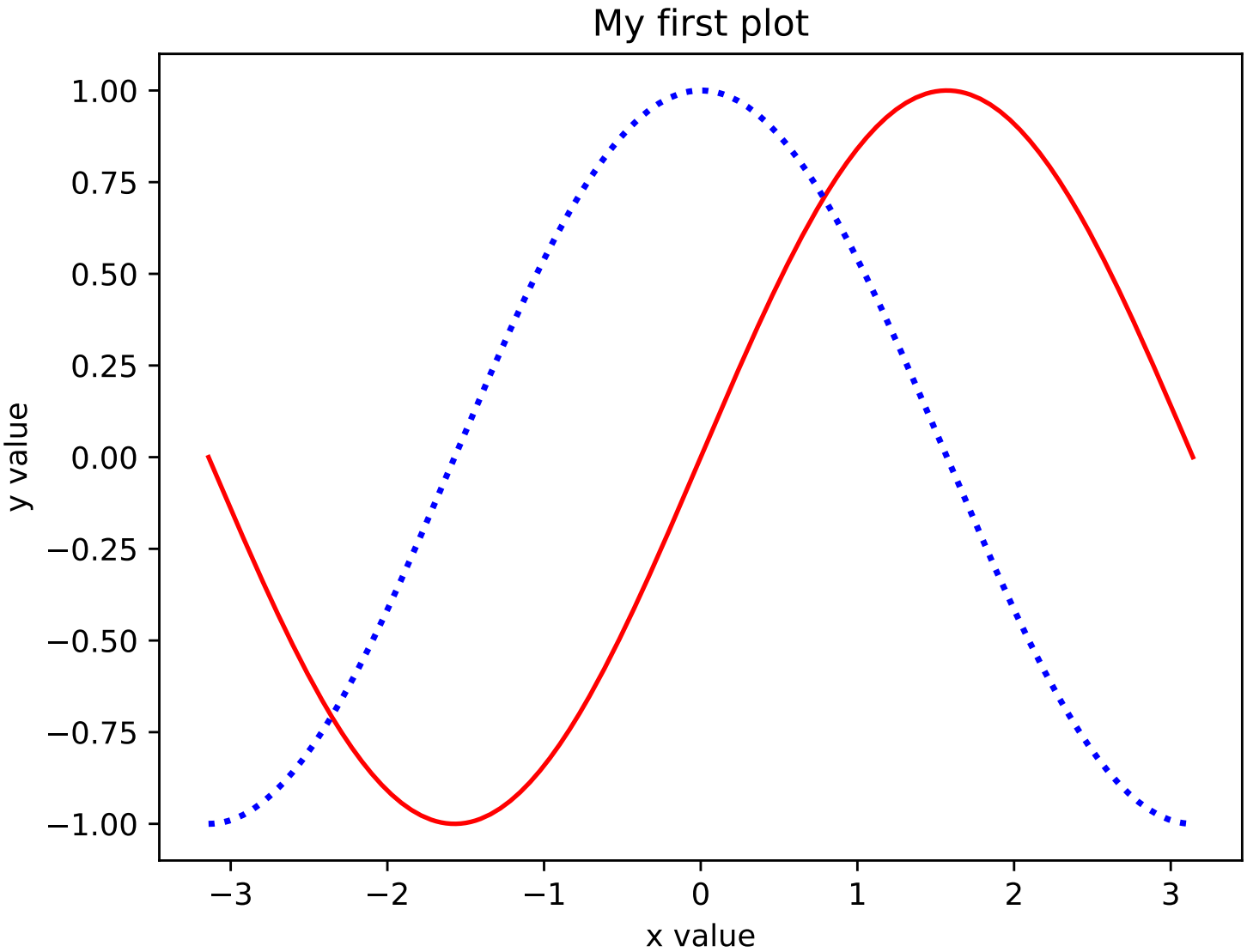
Struttura generale

- si importano i moduli richiesti (`numpy as np` e `matplotlib.pyplot as plt`),
- si carica o si genera qualche dato,
- (opzionale) si customizza l'aspetto del plot
- si genera il plot (`plot()`, `bar()`, `pie()` etc),
- lo si mostra o lo si salva in un file (formati: PNG, PDF, SVG etc)

Esempio 1

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(-np.pi, np.pi, 200)
y1 = np.sin(x)
y2 = np.cos(x)
plt.title("My first plot")
plt.plot(x, y1, "r-")
plt.plot(x, y2, color="blue", linewidth=2.0, linestyle=":")
plt.ylabel('y value')
plt.xlabel('x value')
plt.savefig("figure1.pdf", bbox_inches='tight')
```

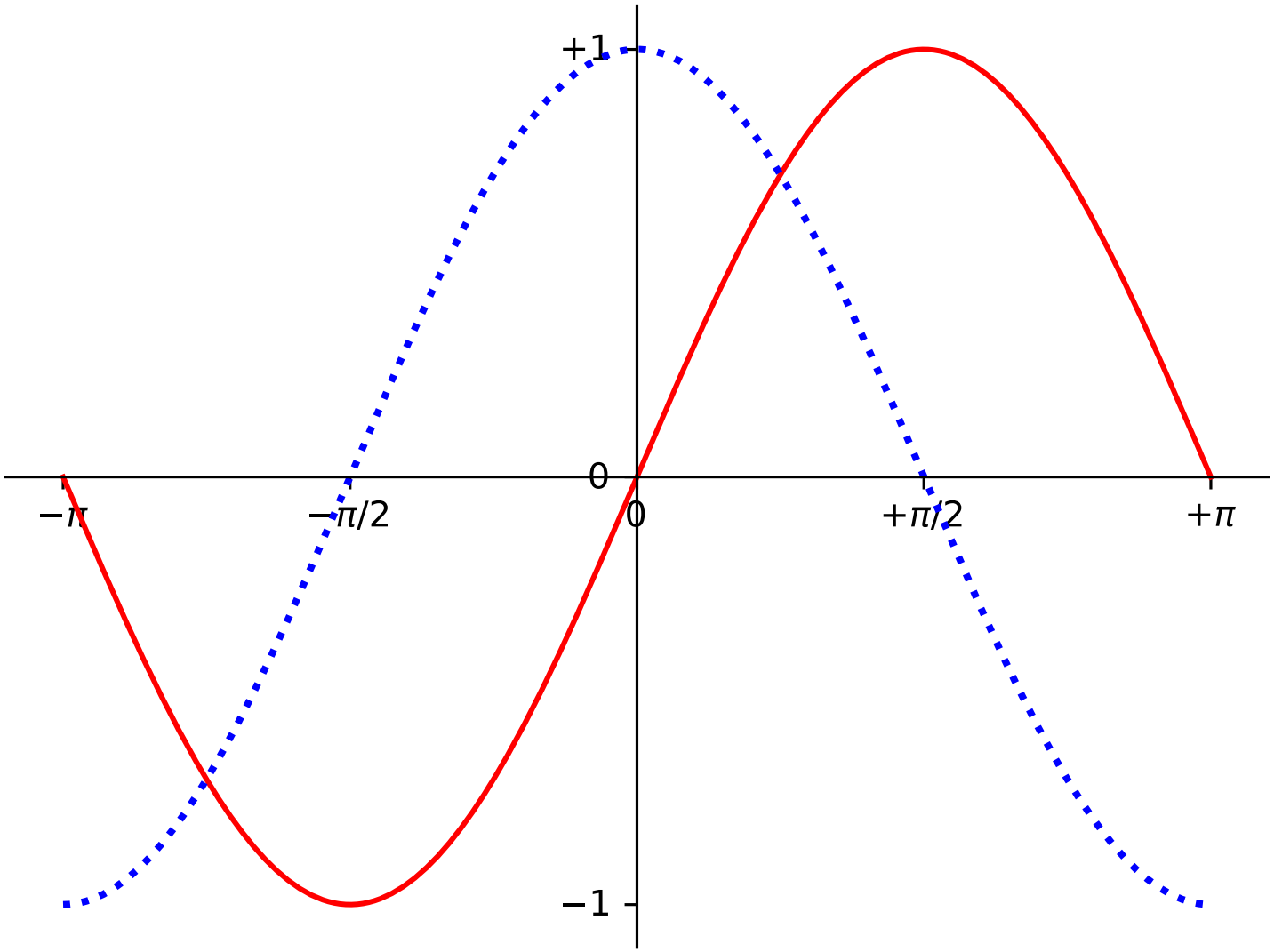
Esempio 1


My first plot
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
y value
−3
−2
−1
0
1
2
3
x value

Esempio 2

```
plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],
  [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$'])
plt.yticks([-1, 0, +1],
  [r'$-1$', r'$0$', r'$+1$'])
plt.plot(x, y1, "r-")
plt.plot(x, y2, color="blue", linewidth=2.0, linestyle=":")
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data',0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data',0))
```

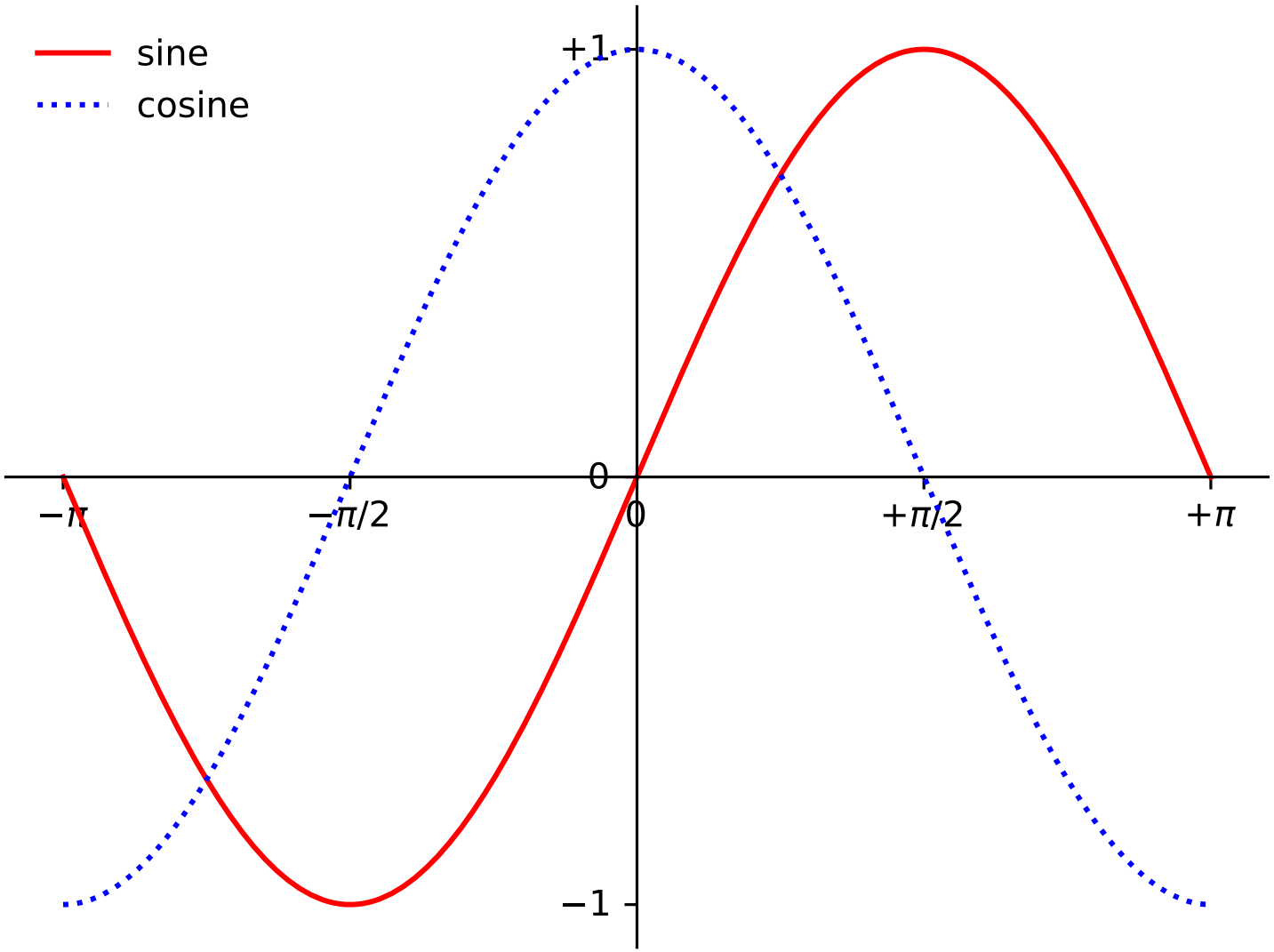
Esempio 2



Esempio 3

```
plt.plot(x, y1, "r-", label="sine")
plt.plot(x, y2, "b:", label="cosine")
plt.legend(loc='upper left', frameon=False)
```

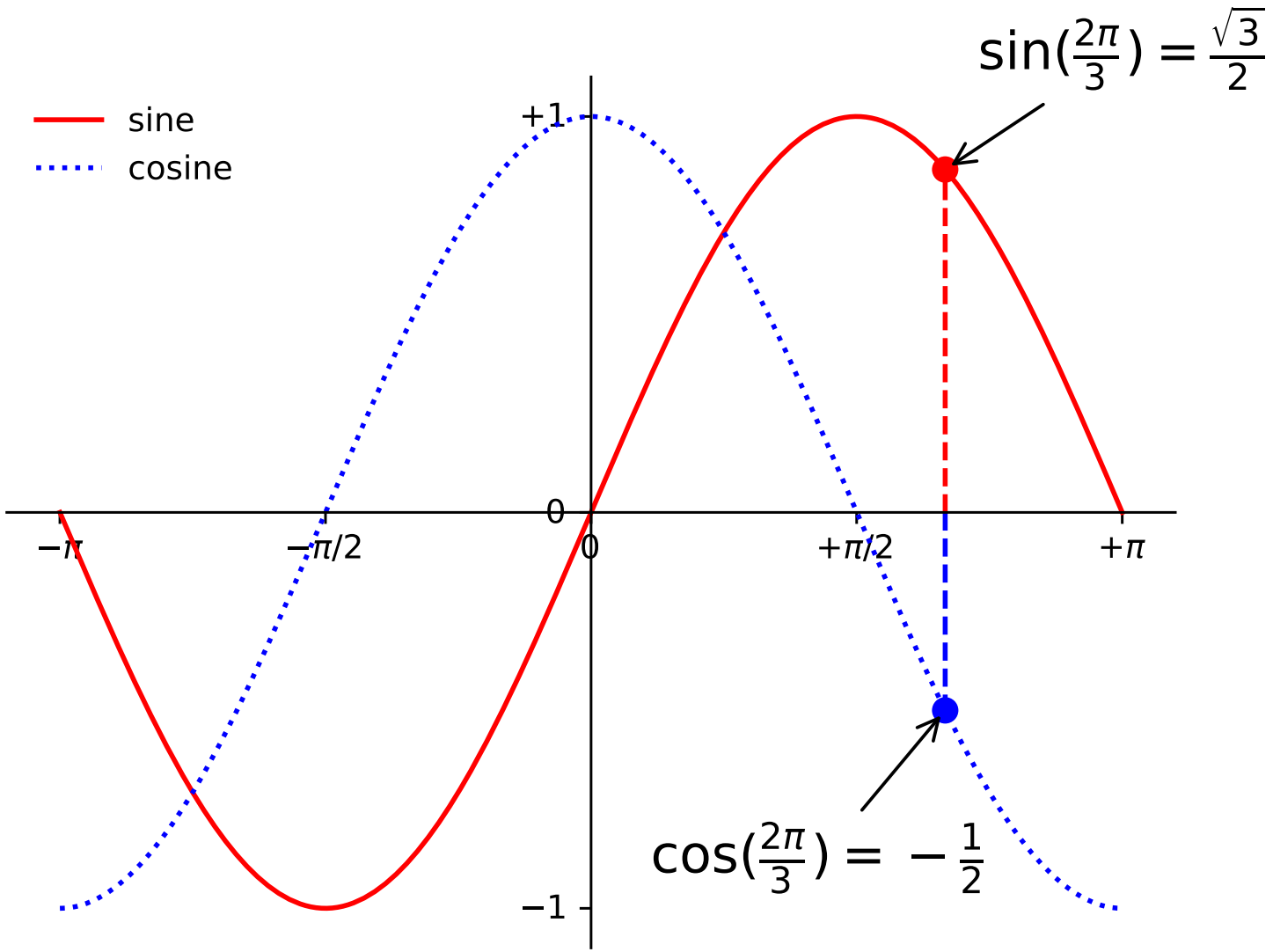
Esempio 3



Esempio 4

```
t = 2*np.pi/3
plt.plot([t,t],[0,np.cos(t)], "b--")
plt.scatter([t,],[np.cos(t),], 50, color ='blue')

plt.annotate(r'$\sin(\frac{2\pi}{3})=\frac{\sqrt{3}}{2}$',
  xy=(t, np.sin(t)), xycoords='data',
  xytext=(+10, +30), textcoords='offset points',
  fontsize=16, arrowprops=dict(arrowstyle="->"))

plt.plot([t,t],[0,np.sin(t)], "r--")
plt.scatter([t,],[np.sin(t),], 50, color ='red')

plt.annotate(r'$\cos(\frac{2\pi}{3})=-\frac{1}{2}$',
  xy=(t, np.cos(t)), xycoords='data',
  xytext=(-90, -50), textcoords='offset points',
  fontsize=16, arrowprops=dict(arrowstyle="->"))
```

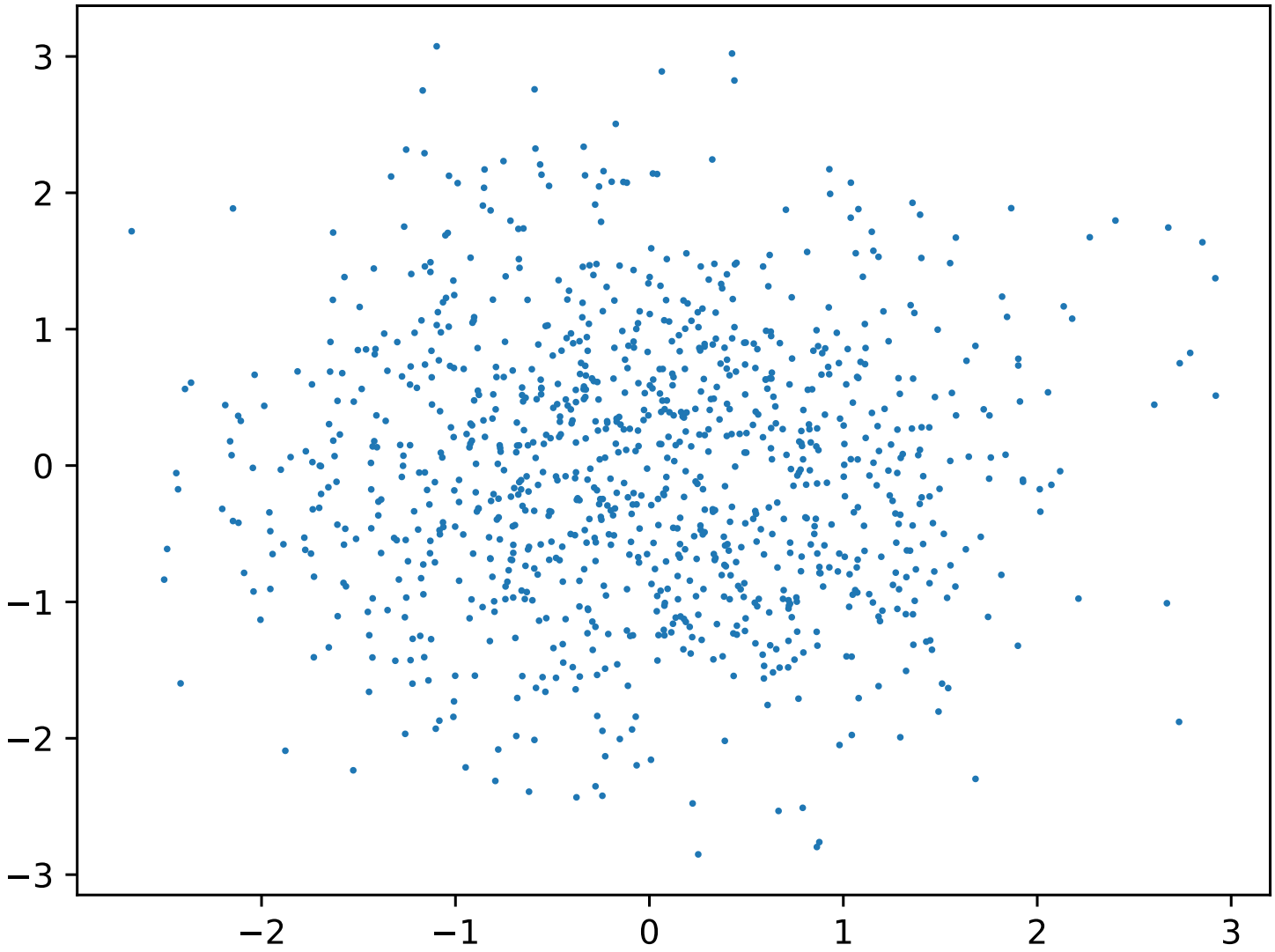
Esempio 4


sine
cosine
+1
0
−1
−π
−π/2
0
+π/2
+π
$\sin(\frac{2\pi}{3}) = \frac{\sqrt{3}}{2}$
$\cos(\frac{2\pi}{3}) = -\frac{1}{2}$

Esempio 5

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

n = 1024
X = np.random.normal(0,1,n)
Y = np.random.normal(0,1,n)

plt.scatter(X,Y,s=1)
plt.savefig("figure5.pdf")
```

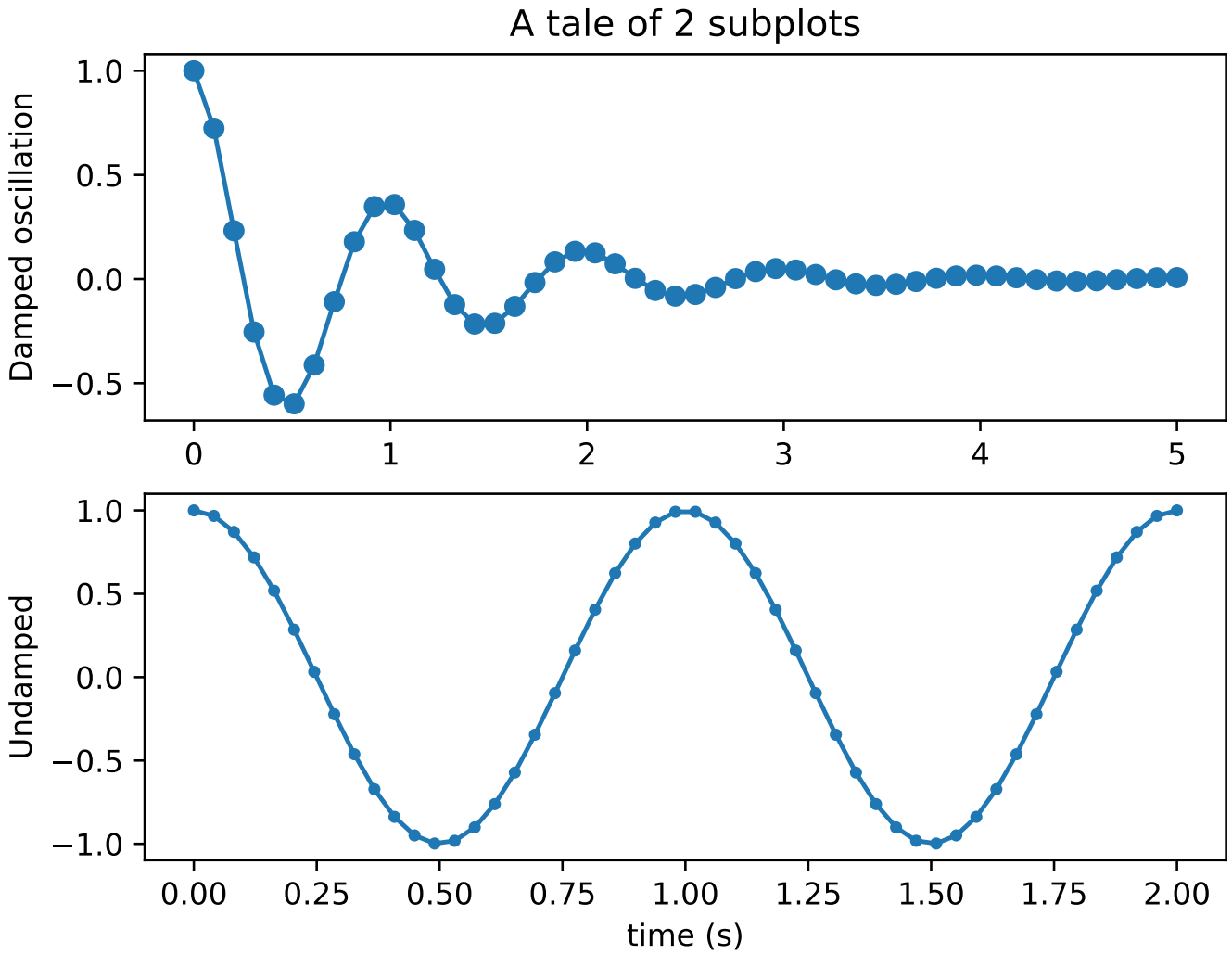
Esempio 5



Esempio 6

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

x1 = np.linspace(0.0, 5.0)
x2 = np.linspace(0.0, 2.0)
y1 = np.cos(2 * np.pi * x1) * np.exp(-x1)
y2 = np.cos(2 * np.pi * x2)

plt.subplot(2, 1, 1)
plt.plot(x1, y1, 'o-')
plt.title('A tale of 2 subplots')
plt.ylabel('Damped oscillation')

plt.subplot(2, 1, 2)
plt.plot(x2, y2, '.-')
plt.xlabel('time (s)')
plt.ylabel('Undamped')

plt.savefig("figure6.pdf")
```

Esempio 6


A tale of 2 subplots
Damped oscillation
1.0
0.5
0.0
−0.5
0
1
2
3
4
5
Undamped
1.0
0.5
0.0
−0.5
−1.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
time (s)

Tutorial

http://www.labri.fr/perso/nrougier/teaching/matplotlib/
http://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1003833