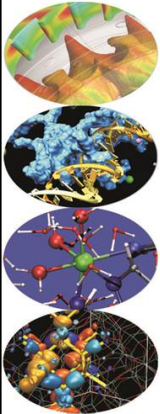
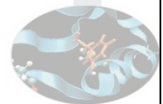


Introduzione a *SciPy*

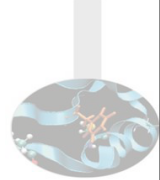


Mario Rosati
CINECA – Roma
m.rosati@cineca.it



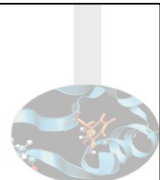
Introduzione ...

- Il pacchetto SciPy contiene diversi “toolbox” dedicati ai problemi più comuni del calcolo scientifico
- I suoi diversi sotto-moduli corrispondono a diverse applicazioni, come interpolazione, integrazione, ottimizzazione, elaborazione di immagini, statistica, funzioni speciali, algebra lineare, ...
- SciPy può essere paragonato ad altre librerie standard largamente utilizzate nel calcolo scientifico come la GSL (GNU Scientific Library per C e C++) o i toolbox di Matlab



... Introduzione

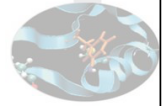
- SciPy è il pacchetto di base per il calcolo scientifico in Python ed è disegnato per operare in modo efficiente su array NumPy
- Prima di implementare una nuova routine, vale la pena verificare se l'algoritmo desiderato non sia già implementato in SciPy
- Non di rado, i programmatori non professionisti tendono a “reinventare la ruota”, una prassi che spesso porta alla produzione di codice *buggy*, non ottimizzato, difficile da condividere ed quasi impossibile da mantenere!
- Al contrario le routine di SciPy sono largamente testate e ben ottimizzate e, quindi, quando possibile, dovrebbero essere utilizzate



Warning

- Questo modulo del corso “*Python for Computational Science*” non vuole essere una introduzione al calcolo numerico, cosa che è al di là dei nostri compiti istituzionali.
- Una enumerazione dei diversi sottomoduli e delle funzioni di SciPy risulterebbe piuttosto lunga e noiosa, quindi ci *concentreremo su pochi esempi* per fornire un’idea generale su come si usa SciPy per il calcolo tecnico scientifico
- Per usare efficacemente le funzionalità di SciPy, è consigliabile fare largo uso delle funzionalità di help dei singoli moduli accessibili, ad esempio, dall’interno di iPython

La struttura di SciPy ...

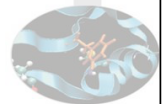


- SciPy è composto da sotto-moduli specifici per tipologia di applicazione; ecco una lista dei principali sotto-moduli disponibili:

Sub-module	Task
<code>scipy.cluster</code>	Clustering package
<code>scipy.constants</code>	Physical and mathematical constants
<code>scipy.fftpack</code>	Discrete Fourier transform
<code>scipy.integrate</code>	Integration e ODEs routines
<code>scipy.interpolate</code>	Interpolation
<code>scipy.io</code>	Data input and output
<code>scipy.linalg</code>	Linear algebra routines
<code>scipy.ndimage</code>	n-dimensional image package

5

... La struttura di SciPy



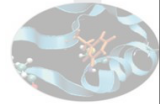
Sub-module	Task
<code>scipy.odr</code>	Orthogonal distance regression
<code>scipy.optimize</code>	Optimization and root finding
<code>scipy.signal</code>	Signal processing
<code>scipy.sparse</code>	Sparse matrices
<code>scipy.spatial</code>	Spatial data structures and algorithms
<code>scipy.special</code>	special mathematical functions
<code>scipy.stats</code>	Statistics

- Tutti i moduli dipendono da NumPy, ma sono praticamente indipendenti uno dall'altro; dunque il modo standard di utilizzarli è:

```
>>> import numpy as np
>>> from scipy import stats #stat sub-mod.
```

6

File I/O: modulo `scipy.io`

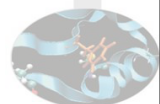


Leggere e scrivere file Matlab e leggere immagini

```
>>> import numpy as np
>>> from scipy import io
>>> a = np.ones((3,3))
>>> io.savemat('file.mat', {'a': a}) # 2nd arg is a dict
>>> data = io.loadmat('file.mat', struct_as_record=True)
>>> data['a']
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
>>> from scipy import misc
>>> misc.imread('myImage.png') # e' anche in matplotlib
```

7

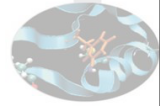
Funzioni speciali: il modulo `scipy.special`



- Il modulo contiene una vasta scelta di funzioni speciali
- Il suo help è molto bel fatto ed contiene un elenco completo delle funzioni disponibili, raggruppate per tipologia: è un ottimo riferimento per vedere cosa è presente nel modulo e come utilizzarlo
- Alcune delle principali (elenco breve e non esaustivo!) funzioni presenti:
 - Funzioni di Bessel (`scipy.special.jn, ...`)
 - Funzioni ellittiche e funzioni integrali (`scipy.special.ellipj, ...`)
 - Funzioni Gamma (`scipy.special.gamma, scipy.special.gammaln`)
 - Error Function (`scipy.special.erf`)
 - Diverse classi di polinomi ortonormali (Legendre, Laguerre, Hermite, ...)

8

Linear algebra: calcolo del determinante

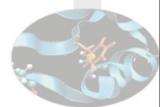


Funzione per il calcolo del determinante di una matrice quadrata

```
>>> from scipy import linalg
>>> arr = np.array([[1, 2],
...                 [3, 4]])
>>> linalg.det(arr)
-2.0
>>> arr = np.array([[3, 2],
...                 [6, 4]])
>>> linalg.det(arr)
0.0
>>> linalg.det(np.ones((3, 4)))
Traceback (most recent call last):
...
ValueError: expected square matrix
```

9

Linear algebra: calcolo dell'inversa ...



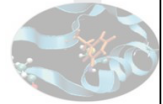
Funzione per il calcolo dell'inversa di una matrice quadrata

```
>>> from scipy import linalg
>>> arr = np.array([[1, 2],
...                 [3, 4]])
>>> iarr = linalg.inv(arr)
>>> iarr
array([[ -2. ,  1. ],
       [ 1.5, -0.5]])
>>> np.allclose(np.dot(arr, iarr), np.eye(2))
True
```

NB: `np.allclose` ritorna il valore True se i due array sono uguali entro una certa tolleranza, che, eventualmente, può essere passata come argomento. Per i dettagli: `np.allclose?`

10

... Linear algebra: calcolo dell'inversa



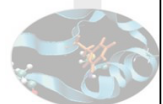
Calcolo dell'inversa: che succede con una matrice singolare?

```
>>> from scipy import linalg
>>> arr = np.array([[3, 2],
...                [6, 4]])

>>> iarr = linalg.inv(arr)
Traceback (most recent call last):
...
LinAlgError: singular matrix
```

11

Linear algebra: singular-value decomposition



La *singular-value decomposition* di una matrice A consiste nel calcolo di 2 matrici unitarie U e V e di un array s di valori singolari (reali e non negativi), tali che $A = USV$, in cui S è la matrice diagonale di elementi diagonali s

Funzione per la *singular-value decomposition*

```
>>> from scipy import linalg
>>> a = np.arange(9).reshape((3,3))
>>> u, spec, v = linalg.svd(arr)

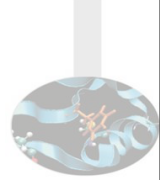
>>> s = np.diag(spec)
>>> svd = u.dot(s).dot(v)
>>> np.allclose(svd,a)
True
```

NB: se il calcolo non converge,
è generato un `LinAlgError`

Sono disponibili anche altre decomposizioni standard: QR, LU, Cholesky, Schur

12

Linear algebra: soluzione di un sistema lineare



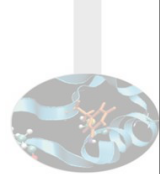
Data una matrice A (n,m) ed un vettore b (n), calcolare il vettore x (n) tale che $Ax=b$

Soluzione di un sistema lineare

```
>>> from scipy import linalg
>>> A = np.array([[1,2],[3,4]])
>>> b = np.array([[5],[6]])
>>> x = linalg.solve(A,b)
>>> x
array([[ -4. ],
       [  4.5]])
>>> np.allclose(A.dot(x)-b,np.zeros((2,2)))
True
```

13

Ottimizzazione e fit: scipy.optimize

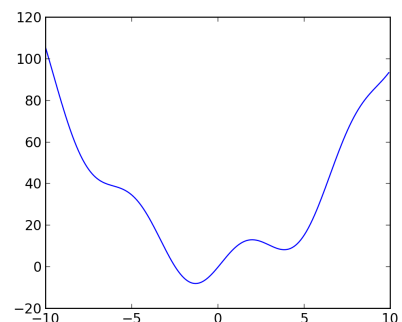


In generale, l'ottimizzazione consiste nella ricerca di una soluzione numerica a un problema di minimizzazione o nel calcolo degli zeri di una funzione

Ricerca del minimo di una funzione: definizione e grafico

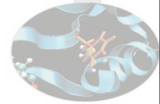
```
>>> def f(x):
...     return x**2 + 10*np.sin(x)
>>> import matplotlib.pyplot as plt
>>> x = np.arange(-10, 10, 0.1)
>>> plt.plot(x, f(x))
>>> plt.show()
```

La funzione ha un minimo globale intorno a -1 ed un minimo locale intorno a 4!



14

Ricerca del minimo assoluto ...



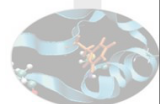
Un metodo generale ed efficiente per la ricerca del minimo della funzione scelta consiste nel condurre una discesa lungo il gradiente, partendo da un punto iniziale. L'algoritmo BFGS è una valida soluzione del problema!

Ricerca del minimo con l'algoritmo BFGS (*starting point* x=0)

```
>>> from scipy import optimize
>>> optimize.fmin_bfgs(f, 0)
Optimization terminated successfully.
      Current function value: -7.945823
      Iterations: 5
      Function evaluations: 24
      Gradient evaluations: 8
array([-1.30644003])
```

15

... Ricerca del minimo assoluto ...



- Un possibile problema di questo approccio è che, se la funzione presenta minimi locali, l'algoritmo può giungere ad uno di questi ultimi, invece che al minimo assoluto, per certe scelte dello *starting point* della discesa

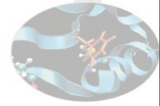
Ricerca del minimo con l'algoritmo BFGS (*starting point* x=3)

```
>>> optimize.fmin_bfgs(f, 3, disp=0)
array([ 3.83746663])
```

- Se non si conosce l'intorno in cui cade il minimo assoluto, è possibile ricorrere a più costosi metodi di ottimizzazione globale
- Per essere certi di aver trovato il minimo assoluto, il metodo più semplice consiste nell'algoritmo "*brute force*": valuta la funzione in tutti i punti di una data griglia e determina dove la funzione è minima

16

... Ricerca del minimo assoluto



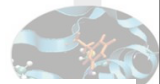
Ricerca del minimo con l'algoritmo *brute force*

```
>>> grid = (-10, 10, 0.1)
>>> optimize.brute (f, (grid, ), disp=True)
Optimization terminated successfully.
      Current function value: -7.945823
      Iterations: 11
      Function evaluations: 22
array([-1.30641113])
```

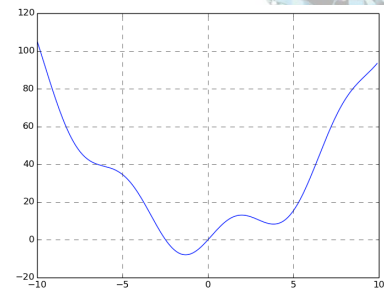
- In caso di griglia con molti punti, questo metodo può diventare costoso
- La funzione `optimize.anneal` è una soluzione alternativa al problema, che usa il metodo del *simulated annealing*
- Esistono diversi algoritmi, non compresi in *SciPy*, che risultano più efficienti per differenti classi di problemi di ottimizzazione globale; in caso di necessità, segnaliamo *OpenOpt*, *IPOPT*, *PyGMO* e *PyEvolve*

17

Ricerca degli zeri di una funzione



- La funzione `scipy.optimize.fsolve` consente di trovare lo zero di una funzione, ovvero una soluzione per l'equazione $f(x) = 0$
- Per semplicità utilizziamo la funzione vista per la ricerca del minimo assoluto

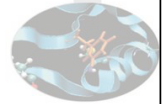


Ricerca dello zero della funzione

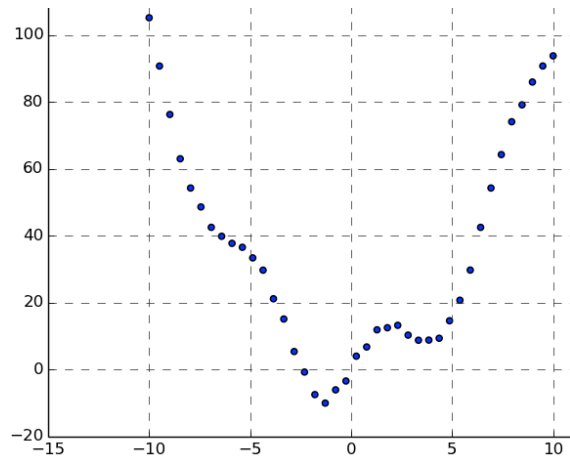
```
>>> from scipy import optimize
>>> x_1 = optimize.fsolve(f, 1) #starting point 1
>>> x_2 = optimize.fsolve(f, -3) #starting point -3
>>> x_1
array([0.])
>>> x_2
array([-2.47948183])
```

18

Fitting di una curva ...



Per procurarci il set di dati di cui trovare il *best-fit*, partiamo dalla funzione f degli esempi precedenti, ed aggiungiamo “un po’ di rumore gaussiano”

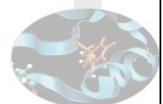


Inizializzazione del set di dati da fittare

```
>>> xd = np.linspace(-10,10,40)
>>> yd = f(xd) + np.random.randn(xd.size)
```

19

... Fitting di una curva ...



- La forma funzionale con cui fittare il campione evidentemente è $ax^2 + b\sin(x)$
- Usiamo il *non linear least square fit* per determinare il valore ottimale dei parametri a e b

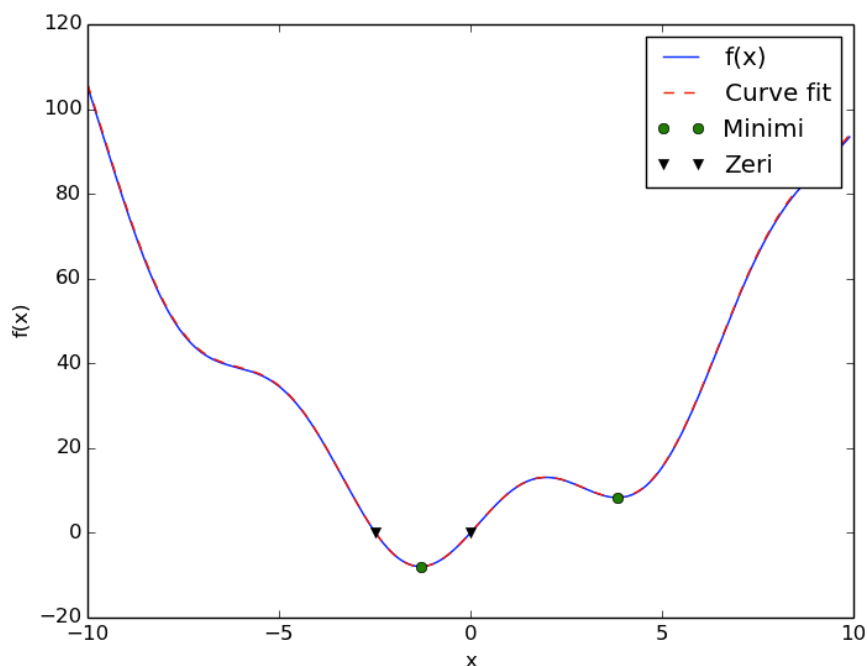
Least square fit

```
>>> from scipy import optimize
>>> def f2(x, a, b):
...     return a*x**2 + b*np.sin(x)
>>> guess = [2,2]
>>> par, par_cov = optimize.curve_fit(f2,xd,yd,guess)
>>> par
array([ 1.00340129, 10.02551458])
>>> par_cov
array([[ 1.01523991e-05, -2.76706261e-16],
       [-2.76706261e-16,  4.72408931e-02]])
```

20

... Fitting di una curva

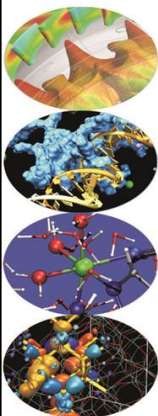
Visualizziamo tutto quanto abbiamo trovato sulla funzione $f(x)$ in un unico plot



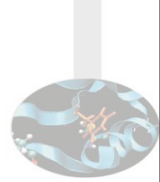
21

Introduzione a **SciPy**: laboratorio

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



SciPy: laboratorio 1

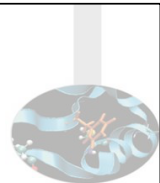


1. Generare un array A – $N \times N$ - di numeri casuali uniformemente distribuiti nell'intervallo $[0,10)$ e il vettore $b = \text{np.linspace}(0, 10, N)$.
Per i valori di $N = \text{range}(100, 1000, 100)$ ed $N = \text{range}(1000, 6000, 1000)$:
 - risolvere il sistema lineare $Ax=b$, misurando il tempo *elapsed* per ottenere la soluzione

Produrre un grafico delle misure ottenute in funzione della dimensione della Matrice A

Hint: per le misure utilizzare il *magic command* `%timeit` o la funzione `clock()` del modulo `time`

SciPy: Laboratorio 1 (2)

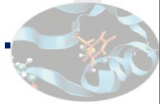


2. Nella successiva tabella sono riportate le temperature massime e minime mensili in Alaska.

Massime	17, 19, 21, 28, 33, 38, 37, 37, 31, 23, 19, 18
Minime	-62, -59, -56, -46, -32, -18, -9, -13, -25, -46, -52, -58

- Disegnare per punti la curva delle minime e delle massime nello stesso grafico
- Definire una funzione (parametrica) che descriva le temperature minime ed una che descriva le massime
(Hint 1: le funzioni dovrebbero avere una periodicità annuale. Hint 2: includere un offset temporale)
- Eseguire il fit dei dati utilizzando le funzioni definite e inserire le curve ottenute nel grafico precedente

scipy.stats: istogramma e PDF...



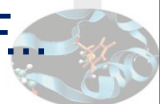
- *scipy.stats* contiene un gran numero di strumenti per descrivere campioni statistici, per lavorare con distribuzioni di probabilità - continue e discrete - e per eseguire diverse tipologie di test statistici
- Dato un campione di un processo random, il suo istogramma è un 'estimatore' della funzione densità di probabilità del processo.

Calcolo dell'istogramma di un campione con NumPy

```
>>> a = np.random.normal(size=10000) # our sample
>>> eb = np.arange(-4,4.25,0.25)      # binning edges
>>> eb.size
33
>>> h= np.histogram(a,bins=eb, normed=True)
>>> h[0].size
32
```

25

... *scipy.stats*: istogramma e PDF...



Costruiamo l'array dei punti medi di ogni bin; poi, usando *scipy.stats*, calcoliamo la PDF della normale sugli stessi punti ed infine confrontiamola, su un grafico, con l'istogramma prodotto

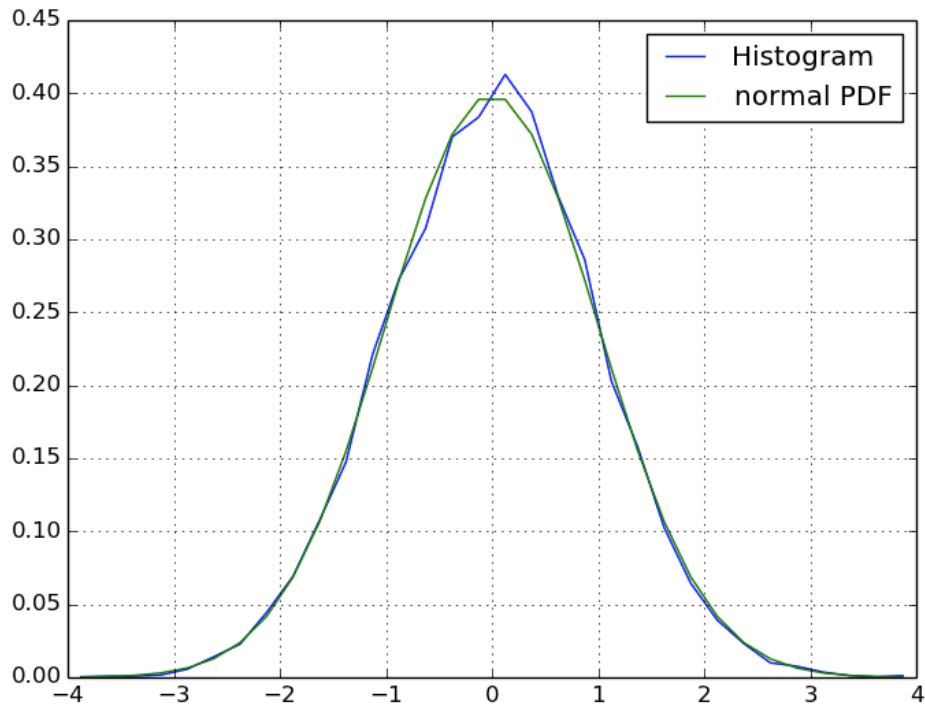
Calcolo della normal PDF sui punti medi dei bin e plot delle curve

```
>>> b = 0.5 * (eb[1:] + eb[:-1])
>>> b.size
32
>>> from scipy import stats
>>> my_pdf = stats.norm.pdf(b)

>>> import matplotlib.pyplot as plt
>>> plt.plot(b, h[0], label='Histogram')
>>> plt.plot(b, my_pdf, label='normal PDF')
```

26

... `scipy.stats`: istogramma e PDF



27

Maximum-likelihood fit

Se il campione proviene da un processo random appartenente ad una data famiglia (i processi gaussiani, nel nostro caso), possiamo usare il *maximum-likelihood fit* del nostro campione per stimare i parametri della distribuzione sottostante

Fit dei parametri della gaussiana che meglio approssima il campione

```
>>> from scipy import stats
>>> a = np.random.normal(size=10000)
>>> stats.norm.fit(a)
(0.0012754030661755386, 0.99565211852482849)
>>> b = np.random.normal(loc=1, scale=3, size=10000)
>>> stats.norm.fit(b)
(0.96615627674018123, 2.9725933382047445)
```

28

Percentili



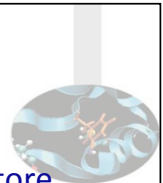
- La mediana di un campione è il numero m tale che il 50% degli elementi del campione sono più piccoli di m ed il restante 50% più grandi di m
- La mediana di un campione è chiamato anche percentile 50 perché il 50% degli elementi del campione ha un valore inferiore
- *Il percentile è un estimatore della densità di probabilità cumulativa (CDF)*

scipy.stats: calcolo del percentile

```
>>> from scipy import stats
>>> a = np.random.normal(size=1000)
>>> np.median(a)
0.038398647271696326
>>> stats.scoreatpercentile(a,50) # percentile 50
0.038398647271696326
>>> stats.scoreatpercentile(a,95) # percentile 95
1.72982274247371
```

29

Il T-test



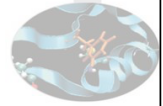
- Il T-test è un test statistico tipicamente è utile per calcolare un indicatore da associare alla bontà di un'ipotesi su uno o più campioni di dati
- Dati 2 campioni gaussiani, vogliamo verificare se sono significativamente differenti: usiamo il T-test

scipy.stats: T-test per l'indipendenza di 2 campioni

```
>>> a = np.random.normal(0,1,size=100)
>>> b = np.random.normal(1,1,size=80) # diverso da a
>>> stats.ttest_ind(a,b)
(array(-7.676065328644207), 1.0441890418905031e-12)
>>> c = np.random.normal(0.1,1,size=80) # simile ad a
>>> stats.ttest_ind(a,c)
(array(-0.7067713387921485), 0.48063276502750174)
```

30

Interpolazione: *scipy.interpolate*



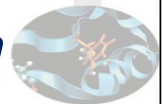
- Il modulo contiene funzionalità per il *fitting* di una funzione (1D e 2D) a partire da dati sperimentali e per la successiva valutazione della funzione in punti in cui il dato sperimentale è mancante
- Generiamo un “dato sperimentale” prossimo alla funzione seno

Generazione degli array di tempi e relative misure

```
>>> t = np.linspace(0,1,10)
>>> noise = 0.1 * (2 * np.random.random(10) -1)
>>> measures = np.sin(2* np.pi * t) + noise
```

31

Interpolazione 1D lineare e cubica



Eseguiamo un'interpolazione sia lineare che cubica dei dati sperimentali, per poi confrontarle con i dati misurati in un plot

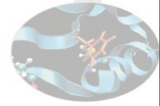
interpolazione 1D di dati sperimentali

```
>>> from scipy.interpolate import interp1d
>>> l_interp = interp1d(t, measures)
>>> c_interp = interp1d(t, measures, kind='cubic')
>>> computed_t = np.linspace(0,1,50)
>>> linear_res = l_interp(computed_t)
>>> cubic_res = c_interp(computed_t)
```

interp1d ritorna una funzione che usa l'interpolazione per trovare il valore in nuovi punti

32

Interpolazione 1D: plot per confronto con i dati ...



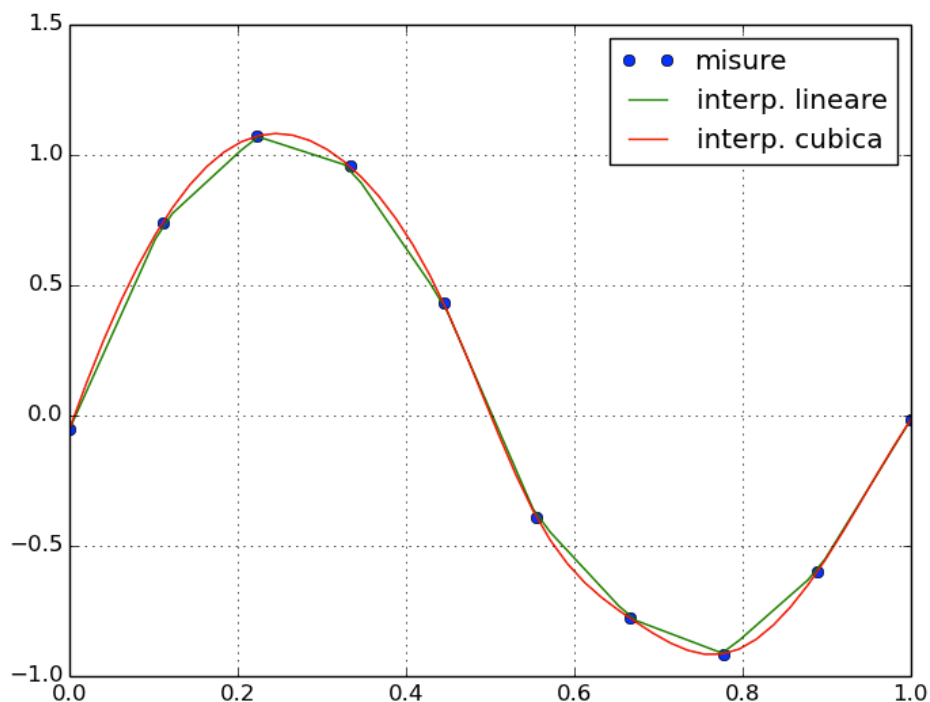
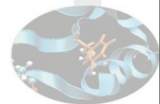
Costruiamo il plot per il confronto tra le curve calcolate e i dati sperimentali di partenza

Costruzione del plot che sovrappone le interpolazioni calcolate con i dati

```
>>> import matplotlib.pyplot as plt
>>> plt.plot(t, measures, 'o', markersize=6, \
            label='misure')
>>> plt.plot(computed_t, linear_res, \
            label='interp. lineare')
>>> plt.plot(computed_t, cubic_res, \
            label='interp. cubica')
>>> plt.grid()
>>> plt.legend()
>>> plt.show()
```

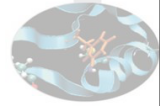
33

... Interpolazione 1D: plot per confronto con i dati



34

scipy.integrate: *integrazione numerica*



`scipy.integrate.quad()` è la più generica funzione per l'integrazione numerica 1D presente in SciPy

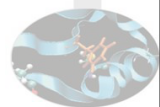
Esempio d'uso di `scipy.integrate.quad`

```
>>> from scipy.integrate import quad
>>> result, err = quad(np.sin, 0, 0.5*np.pi)
>>> np.allclose(result, 1) # check del risultato
True
>>> np.allclose(err, 1 - result) # check dell'errore
True
```

- Oltre al semplice integrale definito, `quad` sa fare molte altre cose (*cfr. help*)
- `dblquad` e `tplquad` sono usate nel caso 2D e 3D rispettivamente
- In `scipy.integrate` sono disponibili anche altri schemi d'integrazione numerica (*cfr. `fixed_quad`, `quadrature`, `romberg`*)

35

scipy.integrate: ODE ...

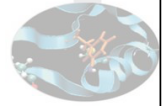


- Il sotto-modulo `scipy.integrate` contiene anche routine per l'integrazione di equazioni differenziali ordinarie (ODE)
- In particolare `scipy.integrate.odeint()` è un integratore general-purpose di ODE che usa LSODA - *Livermore Solver for Ordinary Differential equations with Automatic method switching for stiff and non-stiff problems* - (*cfr. documentazione di ODEPACK per dettagli*)
- `odeint` risolve ODE del prim'ordine della forma

$$y' = f(y, t) \quad \text{con} \quad y = [y_1(t), y_2(t), \dots, y_n(t)]$$

36

... *scipy.integrate*: ODE ...



- Come primo esempio, risolviamo l'equazione

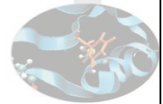
$$\frac{dy}{dt} = -2y \quad \text{con } t \in [0,4] \text{ e } y(0) = 1$$

- Iniziamo con lo scrivere in Python la funzione che calcola la derivata rispetto al tempo della funzione y

```
>>> def calc_derivative(ypos, time, e_counter):
    e_counter += 1
    return -2 * ypos
```

NB: Abbiamo aggiunto l'ulteriore argomento `e_counter`, per illustrare come la funzione sia chiamata più volte ad ogni passo d'integrazione, fino a che il solutore raggiunge la convergenza

... *scipy.integrate*: ODE ...

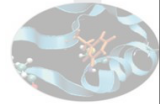


Soluzione numerica dell'equazione differenziale

```
>>> counter = np.zeros((1,), dtype=np.uint16)
>>> time_vec = np.linspace(0,4,40)
>>> from scipy.integrate import odeint
>>> yvec, info = odeint(calc_derivative, 1,
...                   time_vec, args=(counter,),
...                   full_output=True)

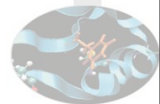
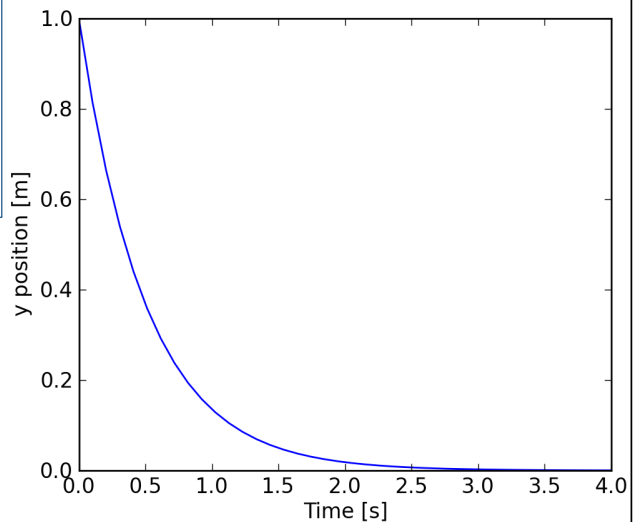
>>> counter      # numero di chiamate a calc_derivative
array([129], dtype=uint16)

# n° d'interazioni per i primi 5 time-step
>>> info['nfe'][:5]
array([31, 35, 43, 49, 53], dtype=int32)
```



Il Plot della soluzione trovata

```
>>> import matplotlib.pyplot as plt
>>> plt.plot(time_vec, yvec)
>>> plt.xlabel('Time [s]')
>>> plt.ylabel('y position [m]')
>>> plt.show()
```



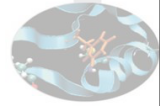
- Come secondo esempio, usiamo `odeint()` per risolvere un ODE di secondo grado, l'equazione di un oscillatore armonico *damped*:

$$x'' + 2\varepsilon\omega_0 x' + \omega_0^2 x = 0$$

in cui $\omega_0^2 = \frac{k}{m}$ e $\varepsilon = \frac{c}{2m\omega_0}$

con k costante elastica della molla, m massa del oscillatore e c coefficiente di damping

... *scipy.integrate*: ODE ...



Per poter usare `odeint()` trasformiamo la ODE di secondo grado in un sistema di ODE di primo grado:

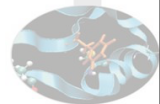
$$\begin{cases} \frac{dx}{dt} = p \\ \frac{dp}{dt} = -2\varepsilon\omega_0 p - \omega_0^2 x \end{cases}$$

Calcolo della derivata del vettore $\mathbf{y} = (x, p)$

```
>>> def dy(y, t, eps, w0):
    x, p = y[0], y[1]
    dx = p
    dp = -2*eps*w0*p - w0**2*x
    return [dx, dp]
```

41

... *scipy.integrate*: ODE ...



Soluzione numerica dell'equazione differenziale per diversi valori del damping ratio

```
# condizione iniziale
>>> y0 = [1.0, 0.0]

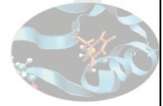
# coordinate temporali in cui trovare la soluzione
>>> t = np.linspace(0,10,1000)

>>> w0 = 2*np.pi

>>> from scipy.integrate import odeint

# risolviamo ls ODE per 4 valori diversi di eps
>>> y1 = odeint(dy,y0,t,args=(0.0,w0)) # undamped
>>> y2 = odeint(dy,y0,t,args=(0.2,w0)) # under damped
>>> y3 = odeint(dy,y0,t,args=(1.0,w0)) # critial damping
>>> y4 = odeint(dy,y0,t,args=(5.0,w0)) # over damped
```

42

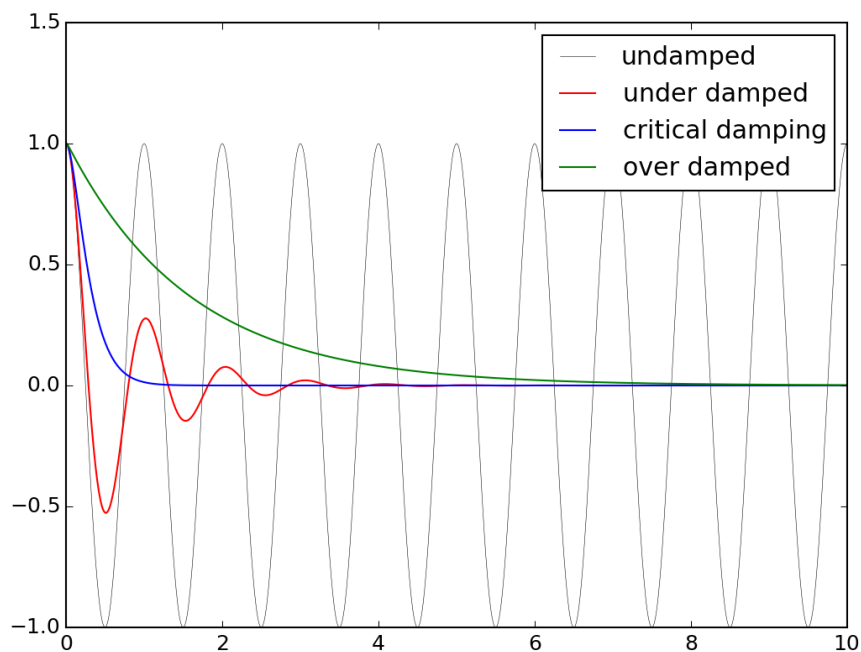
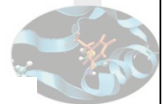


Plot delle 4 soluzioni

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

>>> plt.plot(t, y1[:,0], 'k', label="undamped",
...          linewidth=0.25)

>>> plt.plot(t, y2[:,0], 'r', label="under damped")
>>> plt.plot(t, y3[:,0], 'b', label="critical damping")
>>> plt.plot(t, y4[:,0], 'g', label="over damped")
>>> plt.legend()
>>> plt.show()
```



Fast Fourier transform: `scipy.fftpack`

- `scipy.fftpack` fornisce essenzialmente le funzioni per utilizzare in Python la libreria FFTPACK -disponibile su *NetLib*-, che è un'efficiente e ben testata libreria per FFT scritta in Fortran.
- Come esempio d'uso delle funzioni FFT di SciPy, calcoliamo la Trasformata di Fourier di una delle soluzioni del *damped oscillator*, appena calcolate

scipy.fftpack at work

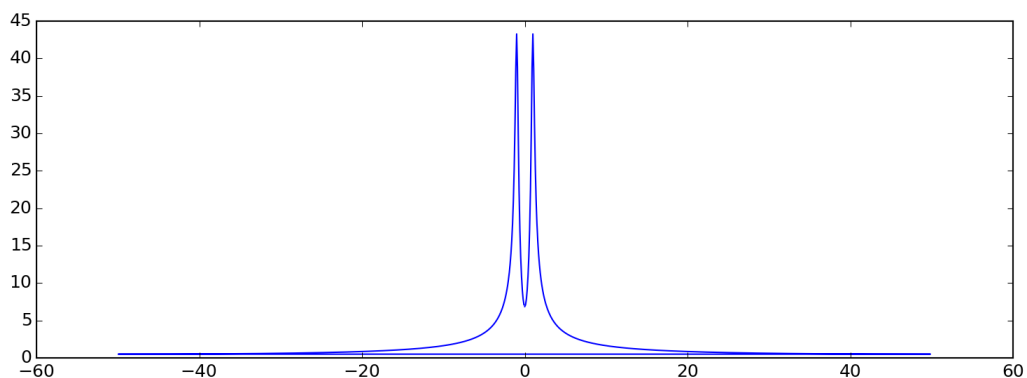
```
>>> N = len(t); dt = t[1]-t[0]
>>> import scipy.fftpack as fftpack
>>> F = fftpack.fft(y2[:,0])
>>> w = fftpack.fftfreq(N,dt)
```

45

Fast Fourier transform: `scipy.fftpack`

Il Plot della Trasformata

```
>>> plt.figure(figsize=(12,4))
>>> plt.plot(w, abs(F));
```



46

Fast Fourier transform: `scipy.fftpack`

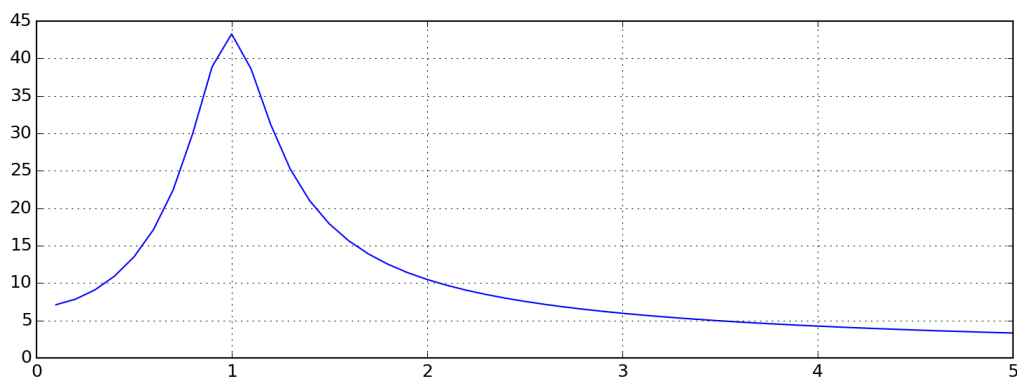
- Poiché il segnale è reale, lo spettro è simmetrico; dunque possiamo disegnare il grafico della Trasformata di Fourier calcolata solo in corrispondenza delle frequenze positive

Il Plot della Trasformata per le sole frequenze positive

```
>>> indices = np.where(w>0)
>>> w_pos = w[indices]
>>> F_pos = F[indices]
>>> plt.figure(figsize=(12,4))
>>> plt.plot(w_pos,abs(F_pos))
>>> plt.xlim(0,5); plt.grid()
>>> plt.show()
```

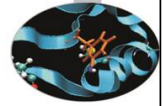
47

Fast Fourier transform: `scipy.fftpack`

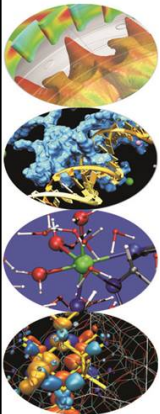


Come c'era da aspettarsi, lo spettro presenta un solo picco centrato intorno ad 1, il valore corrispondente alla frequenza dell'oscillatore del nostro esempio

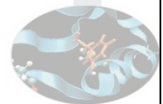
48



Introduzione a **SciPy**: laboratorio 2



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



SciPy: laboratorio 2

1. Leggere il file di dati `populations.txt` e, attraverso le funzionalità del modulo `scipy.fftpack`, verificare se le singole popolazioni presentano periodicità.
2. La funzione:

$$f(x, y) = \left(4 - 2.1x^2 + \frac{x^4}{3}\right)x^2 + xy + (4y^2 - 4)y^2$$

nel dominio $(-2,2) \times (-1,1)$ presenta più di un minimo; trovarne il minimo assoluto.

Hint: il *contour plot* a destra può aiutare a selezionare un buon punto di partenza per un “minimizzatore” multidimensionale tipo, ad esempio, il `bfgs`

