Python

Laboratorio di Metodi Computazionale e Statistici (2023/2024)

Roberta Cardinale, Fabrizio Parodi

Dipartimento di Fisica

Numpy: modifica delle dimensioni, i/o su file

- `numpy.shape(arr)`
  controlla la forma dell'array
- `numpy.reshape(arr, shape)`
  modifica la forma dell'array (senza cambiarne il contenuto) la "shape" è espressa come $(nrow, ncol)$
- `numpy.flatten()`
  converte in una dimensione
- `numpy.loadtxt(filename)`
  legge un file organizzato a righe e colonne e ritorna la matrice (2 array)
- `numpy.savetxt(filename,arr)`
  scrive su file l'array `arr`
- `numpy.flags`
  restituisce informazioni sull'organizzazione dei dati dell'array in memoria

Moduli: ROOT

Interfaccia a ROOT (PyROOT)

- Stesse classe stessi metodi di ROOT/C++
- Accesso ai metodi solo con .
- L'attivazione dell'editor grafico avviene tramite la chiamata a `gApplication` (oggetto python pre-esistente, equivalente al puntatore gApplication in C++)
- Valida alternativa alle macro ROOT (non al ROOT compilato).

Esempio:

```
import math
from ROOT import *
fun1 = TF1('fun1','sin(x)/x', -5*math.pi,
                               5*math.pi )
fun1.Draw()
gApplication.Run()
```

Esercizio

- Leggere il file pendolo.dat sia passo-passo che con la funzione loadtxt.
- Fare il grafico di y in funzione di x.

Lettura passo-passo

```
import numpy as np
from ROOT import *
x  = np.array([])
y  = np.array([])
ex = np.array([])
ey = np.array([])
file = open('pendolo.dat','r')
for line in file:
    str = line.split()
    if len(str)>0:
        x = np.append(x,float(str[0]))
        y = np.append(y,float(str[1]))
        ex = np.append(ex,float(str[2]))
        ey = np.append(ey,float(str[3]))
gr = TGraphErrors(len(x),x,y,ex,ey)
gr.Draw("AP")
gApplication.Run(True)
```

Lettura con loadtxt

```
from ROOT import *
import numpy as np
mat = np.loadtxt('pendolo.dat')
x  = mat[:,0].flatten()
y  = mat[:,1].flatten()
ex = mat[:,2].flatten()
ey = mat[:,3].flatten()
gr = TGraphErrors(len(x),x,y,ex,ey)
gr.Draw("AP")
gApplication.Run(True)
```

Moduli: Matplotlib

Matplotlib è uno dei moduli di Python più utilizzati, permette di realizzare e visualizzare grafici di ogni tipo.

Esempio: grafico 2D

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-5*np.pi,5*np.pi,100)
y = np.sin(x)/x
plt.plot(x,y)
plt.show()
```

Info, esempi e guide su `https://matplotlib.org/users/index.html`

Moduli: Matplotlib

La sintassi generale di plot è la seguente:

```
plot([x], y, [fmt])
```

dove `[x]` e `[fmt]` indicano parametri che possono essere omessi.
I possibili valori (concatenabili !) per la stringa `[fmt]` sono i seguenti:

Line Styles

character	description
'-'	solid line style
'--'	dashed line style
'-.'	dash-dot line style
':'	dotted line style

Colors

The supported color abbreviations are the single letter codes

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

Markers

character	description
'.'	point marker
','	pixel marker
'o'	circle marker
'v'	triangle_down marker
'^'	triangle_up marker
'<'	triangle_left marker
'>'	triangle_right marker
's'	square marker
'p'	pentagon marker
'*'	star marker
'h'	hexagon1 marker
'H'	hexagon2 marker
'+'	plus marker
'x'	x marker
'D'	diamond marker
'd'	thin_diamond marker
'|'	vline marker
'_'	hline marker

Moduli: Matplotlib

- `figure` crea una pagina vuota
- `subplots` crea sottopagine
- `legend` crea la legenda
- `title`, `xlabel` , `ylabel` titoli e nomi degli assi

Notebook

- Python (così come altri linguaggi) può essere runnato in particolari interfacce web Jupyter (Julia/Python/R) Notebook
  - Il notebook può basarsi su risorse locali (useremo questo)
  - O utilizzare software su server remoti
- Utilizzeremo il notebook di ROOT (che supporta C++, Python e ROOT)

  ```
  root --notebook
  ```
- Il notebook è un tool di discussione/sviluppo/didattica. Permette di presentare testo e codice insieme.
- Estremamente utile per "abbozzare" un progetto, meno per il suo sviluppo.

Esercizio

Sia in pyroot che in matplotlib (con notebook):

- disegnare $sin(x)/x$ tra $[-10\pi, 10\pi]$ e $G(0,1)$ tra $[-3,3]$ in due sottodivisioni della finestra grafica.

Notebook ROOT/C++

```
TCanvas c;
c.Divide(2,1);
c.Draw();
c.cd(1);
TF1 f1("f1","sin(x)/x",-10*TMath::Pi(),10*TMath::
f1.Draw();
c.cd(2);
TF1 f2("f2","TMath::Gaus(x,0,1)",-3,3);
f2.Draw();
```

Vedi nb_ROOTC.ipynb

Notebook pyroot

```
import ROOT
c=ROOT.TCanvas()
c.Divide(2,1)
c.Draw()
c.cd(1)
f1 = ROOT.TF1("f1","sin(x)/x",-10*ROOT.TMath.Pi()
f1.Draw()
c.cd(2)
f2 = ROOT.TF1("f2","TMath::Gaus(x,0,1)",-3,3)
f2.Draw()
```

Vedi nb_pyroot.ipynb

Notebook matplotlib (I)

```
import matplotlib.pyplot as plt
import numpy as np
import math  as m
import scipy.stats as sc
plt.subplot(1,2,1)
x  = np.linspace(-10*m.pi,10*m.pi,100)
y  = np.sin(x)/x
plt.plot(x,y)
plt.subplot(1,2,2)
x2 = np.linspace(-3,3,100)
y2 = sc.norm(0,1).pdf(x2)
plt.plot(x2,y2)
```

Vedi nb_matplotlib1.ipynb

Notebook matplotlib (II)

```
import matplotlib.pyplot as plt
import numpy as np
import math  as m
import scipy.stats as sc
fig, (ax1,ax2) = plt.subplots(1,2)
x = np.linspace(-10*m.pi,10*m.pi,100)
y = np.sin(x)/x
x2 = np.linspace(-3,3,100)
y2 = sc.norm(0,1).pdf(x2)
ax2.plot(x2,y2)
ax1.plot(x,y)
```

Vedi nb_matplotlib2.ipynb

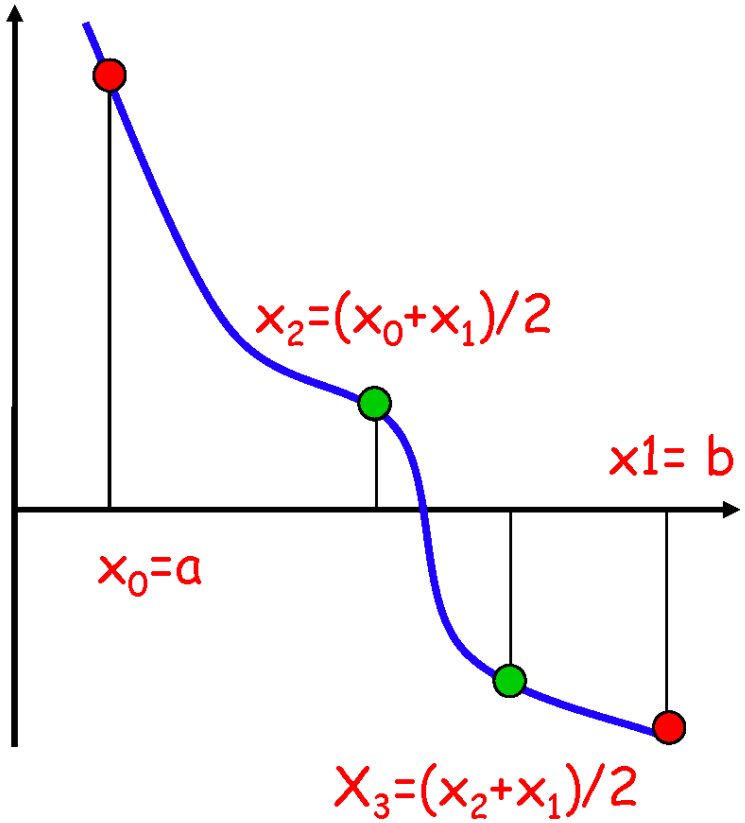
Applicazioni: algoritmo di bisezione

Il metodo di bisezione per la ricerca degli zeri consiste nel fissare un intervallo (ai cui estremi la funzione ha segno opposto), calcolare il valore della funzione nel punto medio e determinare in quale dei due sotto-intervalli si trova lo zero.
Ovvero si controlla quale coppia tra

`f(x0,x1)` e `f(x1,x2)`

sia di segno discorde.

L'intervallo che contiene lo zero diventa il nuovo intervallo di ricerca e così via. L'iterazione continua fino a che l'ampiezza dell'intervallo non scende sotto un valore prefissato.



Esercizio

Sia in pyroot (script) che in matplotlib (con notebook):

- cercare la soluzione dell'equazione trascendente $e^x = cos(x)$