ENSAM Bordeaux – mars 2020

Calcul Scientifique avec

python™ 3

- ▷ Introduction - Prise en main
- ▷ Les bases du langage Python
- ▶ I/O fichiers ASCII, tracés de courbes
- ▷ Les modules Numpy et Scipy
- ▷ Conception/programmation OO

Jean-Luc Charles - Éric Ducasse

Arts & Métiers, I2M

Arts et Métiers Sciences et Technologies

I2M Bordeaux

Les Entrées/Sorties Fichier concernent 2 types de fichiers :

- **Fichier ASCII** (*fichier texte*) : 1 octet ⇜⇝ 1 caractère (code ASCII)
  - ▷ **ASCII délimité** données séparées par un **délimiteur** :
    `espace` `\t` `;` `,` ...
  - ▷ **ASCII tabulé** données séparées par des tabulations `\t` (export tableur) ;
  - ▷ **ASCII CSV** Comma Separated Value (export tableur) ;
  - ▷ **ASCII libre** format spécifique... mais lisible !
- **Fichier binaire** : contient des données au format binaire
  - ▷ **nombres** entiers, flottants 32 ou 64 bits (codage IEEE754) ;
  - ▷ **objets applicatifs** données binaires spécifiques à une application (images, son, traitement de texte, PDF...).

Les Entrées/Sorties Fichier ASCII pur

code ASCII : 1 octet = valeur décimale $\in [0, 255]$ $\leftrightsquigarrow$ 1 caractère (glyphe)

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

- **Caractères de contrôle** : valeur décimale $\in [0, 31] \cup \{127\}$

LF	*Line Feed*	`\n`	*new line*	saut de ligne
CR	*Carriage Return*	`\r`	*return*	retour chariot
HT	*Horiz. Tabulation*	`\t`	*tabulation*	tabulation
...				

- **ASCII 7 bits** : octets de valeur décimale $\in [32, 126]$) $\leftrightsquigarrow$ caractères US

Les Entrées/Sorties Fichier ASCII étendu

- **ASCII 8 bits**
  octets de valeur décimale $\in [127, 255]$ $\leftrightsquigarrow$ caractères accentués (nationaux).

- Les normes ISO-8859 pour l'ASCII 8 bits :

```
ISO 8859-1    West European languages (Latin-1)
ISO 8859-2    Central and East European languages (Latin-2)
ISO 8859-3    Southeast European and miscellaneous languages (Latin-3)
ISO 8859-4    Scandinavian/Baltic languages (Latin-4)
ISO 8859-5    Latin/Cyrillic
ISO 8859-6    Latin/Arabic
ISO 8859-7    Latin/Greek
ISO 8859-8    Latin/Hebrew
ISO 8859-9    Latin-1 modification for Turkish (Latin-5)
ISO 8859-10   Lappish/Nordic/Eskimo languages (Latin-6)
ISO 8859-11   Latin/Thai
ISO 8859-13   Baltic Rim languages (Latin-7)
ISO 8859-14   Celtic (Latin-8)
ISO 8859-15   West European languages (Latin-9)
ISO 8859-16   Romanian (Latin-10)
```

- Le module `codecs` définit les `codecs` Python (*encoders* et *decoders*) pour l'encodage des caractères docs.python.org/library/codecs.html#standard-encodings.

- Les temps modernes : l'encodage universel Unicode et l'encodage UTF-8.

Les Entrées/Sorties Fichier ASCII étendu

- Exemple : **ISO-8859-15** (latin9, source : fr.wikipedia.org/wiki/ISO_8859-15)

ISO 8859-15																
	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1x	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2x	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6x	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7x	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL
8x	PAD	HOP	BPH	NBH	IND	NEL	SSA	ESA	HTS	HTJ	VTS	PLD	PLU	RI	SS2	SS3
9x	DCS	PU1	PU2	STS	CCH	MW	SPA	EPA	SOS	SGCI	SCI	CSI	ST	OSC	PM	APC
Ax	NBSP	¡	¢	£	€	¥	Š	§	š	©	ª	«	¬		®	¯
Bx	°	±	²	³	Ž	µ	¶	·	ž	¹	º	»	Œ	œ	Ÿ	¿
Cx	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
Dx	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
Ex	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
Fx	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Ouverture d'un fichier

- Ouverture du fichier avec la *built-in* fonction `open(nom, mode)` :
  - ▷ `nom` : chaîne de caractères, nom du fichier ;
  - ▷ `mode` : chaîne de caractères, accès au fichier ;

    (`'r'` : *read*, `'w'` : *write*, `'a'` : *append*)

mode	*fichier*	*accès*	*position*	*remarque*
`'r'`	ASCII	R	début	le fichier doit exister
`'a'`	ASCII	W	fin (ajout)	le fichier doit exister
`'w'`	ASCII	W	début (écrasement)	le fichier est créé s'il n'existe pas

  en ajoutant `'b'`, on précise l'accès à un fichier **binaire** (`'rb'`, `'wb'`...)

  ▷ `open` appliquée à un fichier texte renvoie un objet de type `TextIOWrapper`.

- Le **séparateur de lignes** dans les fichiers ASCII

  Gnu/Linux `\n`
  Mac Os X `\r` (*canal historique)* puis `\n`
  Windows `\r\n`

  Transparent pour Python ! on peut utiliser `\n` quel que soit l'OS.

Utilisation d'un objet `TextIOWrapper`

- Principales méthodes (• : objet `file`)
  - `.read()` lit tout le fichier, renvoie un `str`
  - `.read(n)` lit au moins `n` octets, renvoie un `str`
  - `.readline()` lit une ligne, renvoie un `str`
  - `.readlines()` lit toutes les lignes du fichier, renvoie un objet `list`
  - `.write(s)` écrit l'objet `str` `s` dans le fichier
  - `.writelines(lines)` écrit l'objet `lines` (`list` de `str`) dans le fichier
  - `.close()` ferme le fichier
  - `.next()` renvoie la ligne suivante
  - `.seek(0)` repositionne la lecture/écriture au début du fichier
- Un objet `TextIOWrapper` est itérable :

```
>>> stream = open("Fichier1.txt", "r")
>>> for line in stream:    # une ligne est lue à chaque tour de boucle
      print(line.strip()) # strip enlève espace/sauts en début/fin de ligne
>>> stream.close()
```

Lecture d'un fichier ASCII délimité (1/3)

data2.txt

```
1  # Time [s]      température [Deg. C]
2  0.000000e+00     3.321566e+01
3  1.700000e-01     6.013489e+01
4  3.400000e-01     1.154852e+02
5  5.100000e-01     1.710066e+02
6  6.800000e-01     2.265946e+02
7  # Création : Jeudi 3 novembre 2011, 23:09:24
```

- Lecture du fichier ASCII ligne par ligne avec une boucle `for`

```
stream = open("data2.txt", "r")
data = []
for line in stream:
    if line[0] != "#":
        data.append(line)
        print(repr(line), line.split())
stream.close()
```

- `open` : ouvrir un fichier, `"r"` : lecture
- `stream` est un objet itérable
- `data` : objet `list` vide
- `line` : objet `str` (ligne du fichier)
- `append` : méthode de la classe `list`
- `split` : méthode de la classe `str`

```
'0.000000e+00\t3.321566e+01\r\n' ['0.000000e+00', '3.321566e+01']
'1.700000e-01\t6.013489e+01\r\n' ['1.700000e-01', '6.013489e+01']
...
'6.800000e-01\t2.265946e+02\r\n' ['6.800000e-01', '2.265946e+02']
```

Lecture d'un fichier ASCII délimité (2/3)

- liste de `str` ⇝ liste de `float` : fonction `map` ou **liste en compréhension**

```
stream = open("data2.txt", "r")
data = []
for line in stream:
    if line[0] != "#":
        L = list(map(float,line.split()))
        data.append(L)
stream.close()
print(data)
```

- `split` : méthode de la classe `str`
- `map` : built-in function (applique la fonction `float` à la liste `line.split()`)

```
stream = open("data2.txt", "r")
data = []
for line in stream:
    if line[0] != "#":
        L = [float(x) for x in line.split()]
        data.append(L)
stream.close()
print(data)
```

- L : liste en compréhension

```
[[0.0, 33.21566], [0.17, 60.13489], [0.34, 115.4852], [0.51, 171.0066],
 [0.68, 226.5946]]
```

Lecture d'un fichier ASCII délimité (3/3)

- Lecture universelle : bloc `try` pour éliminer les lignes de commentaires

```
def read(fileName):
    with open(fileName,"r",encoding="utf8") as stream:
        data = []
        for i, line in enumerate(stream):
            try :
                L = [float(x) for x in line.split()]
                data.append(L)
            except:
                mess = "line [{}] skipped : <{}>"
                print(mess.format(i+1, line.strip()))
    from numpy import array
    return array(data)

if __name__ == "__main__":
    M = read("data2.txt");  print(M)
```

- `with` : close automatique en fin de bloc
- encoding : **Python3** !
- `try` : bloc pour essayer...
- `except` : si `try` échoue !

```
line [1] skipped : <# Time [s]      température [Deg. C]>
line [7] skipped : <# Création : Jeudi 3 novembre 2011, 23:09:24>
[[  0.00000000e+00   3.32156600e+01]
 ...
 [  6.80000000e-01   2.26594600e+02]]
```

Écriture dans un fichier ASCII

- La méthode `write` de la classe `TextIOWrapper` écrit un `str` dans un fichier ASCII

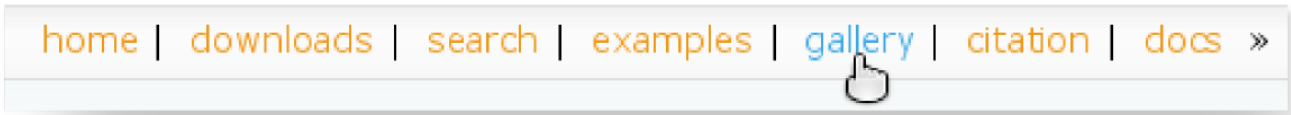
```
# ouverture du fichier en écriture
stream = open("foo.txt", "w")
stream.write("Hello every one !\n");
for i in range(1,5):
   data="{0:6.2E}\t{1:9.3E}\n".format(i*0.1,i*i)
   stream.write(data)
stream.close()

# Ré-ouverture en lecture pour vérification
stream = open("foo.txt", "r")
data = stream.read()
stream.close()
print(data)
```

- `open` : ouvrir un fichier, `'w'` : écriture (écrasement)
- `write` : méthode de la classe `file`
- `data` : chaîne formatée
- `read` : méthode de la classe `file`, lit le fichier en une seule fois

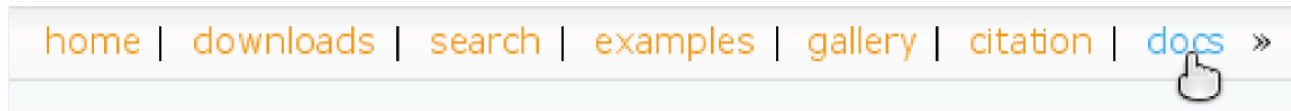
```
Hello every one !
1.00E-01 1.000E+00
2.00E-01 4.000E+00
3.00E-01 9.000E+00
4.00E-01 1.600E+01
```

Le module **matplotlib** propose des outils de tracés (cf http://matplotlib.org) :

- *l*ook & feel "à la MatLab®"
- Gallerie impressionnante de tracés possibles : http://matplotlib.org/gallery.html



  - ▷ Tous les graphiques sont accompagnés du programme Python correspondant
  - ▷ Approche par l'exemple (copier/coller/modifier...)
- Toute la doc est sur le site matplotlib.org/contents.html



Exemple doc > pyplot tutorial http://matplotlib.org/users/pyplot_tutorial.html

matplotlib.org/users/pyplot_tutorial.html

matplotlib

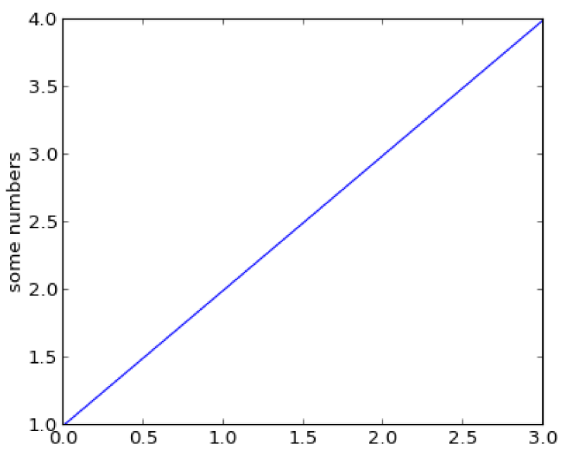
home | downloads | search | examples | gallery | citation | docs »

Pyplot tutorial

matplotlib.pyplot is a collection of command style functions that ma
create a figure, create a plotting area in a figure, plot some lines in a plot
keeps track of the current figure and plotting area, and the plotting func

```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4])
plt.ylabel('some numbers')
plt.show()
```

(Source code, png, hires.png, pdf)



matplotlib.org/users/pyplot_tutorial.html

You may be wondering why the x-axis ranges from 0-3 and the y-axis from 1-4. If y
is a sequence of y values, and automatically generates the x values for you. Since py
starts with 0. Hence the x data are `[0,1,2,3]`.

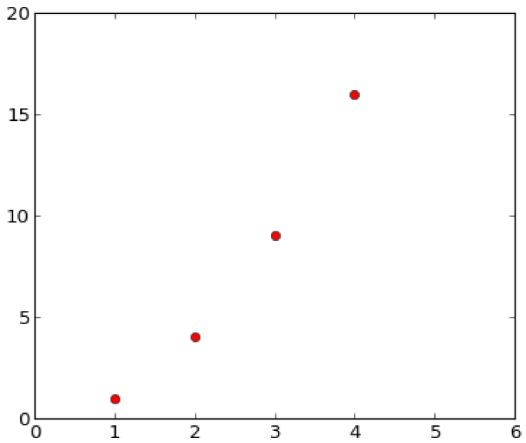
plot() is a versatile command, and will take an arbitrary number of arguments. For

```
plt.plot([1,2,3,4], [1,4,9,16])
```

For every x, y pair of arguments, there is an optional third argument which is the for
symbols of the format string are from MATLAB, and you concatenate a color string w
For example, to plot the above with red circles, you would issue

```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4], [1,4,9,16], 'ro')
plt.axis([0, 6, 0, 20])
```

(Source code, png, hires.png, pdf)



Tracé d'une courbe y=f(x)

[Plot_XY_21.py]

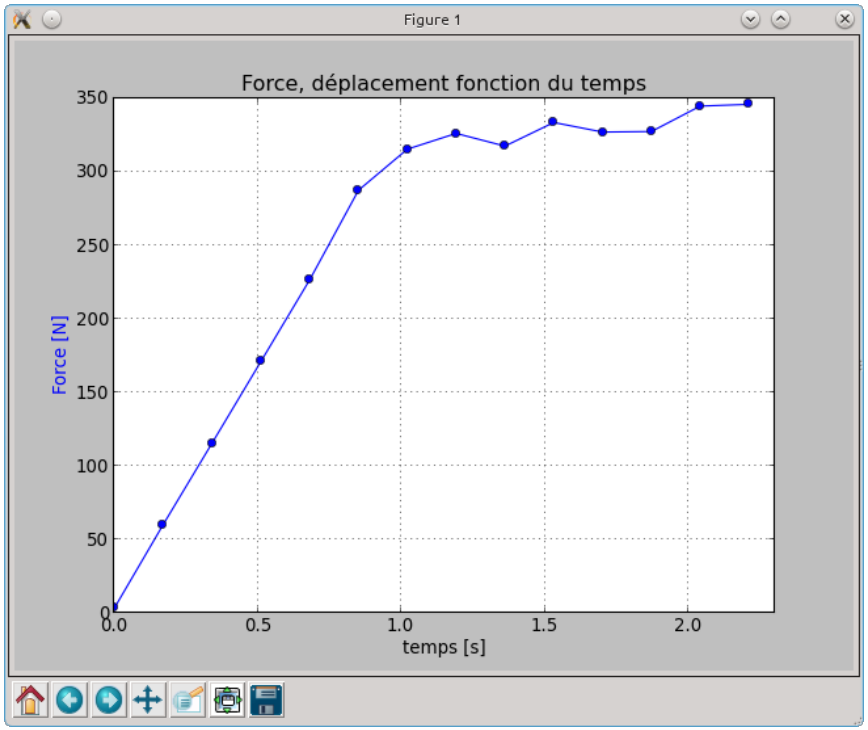
3 colonnes dans le fichier `data1.txt` : temps (s), force (N), déplacement (m).

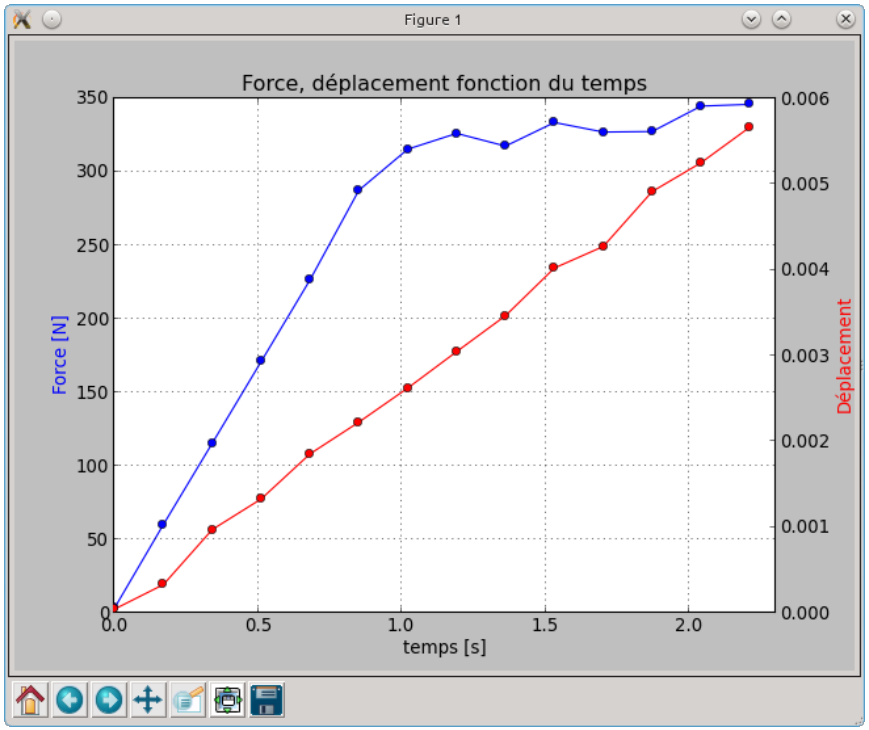
Le tracé "force=f(temps)" dans le premier système d'axe est fait par :

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

t, f, d = np.loadtxt("data1Full.txt", unpack = True)
plt.figure()     # Créer une nouvelle figure
plt.title("Force, déplacement fonction du temps")
plt.grid(True)
plt.xlabel("temps [s]")
plt.xlim(0,2.3)
plt.ylabel("Force [N]", color="b")
plt.ylim(0,350.0)
plt.plot(t, f,"o-b")
# copie du tracé dans un fichier :
plt.savefig("data1.png", format = "png")
plt.show()
```

La figure est sur la diapo suivante...

Tracé d'une courbe y=f(x)





Le tracé complet ([Plot_XY_2.py], à droite) est fait en rajoutant les lignes :

```
plt.twinx()    # second système d'axes :
plt.xlim(0, 2.3)
plt.ylim(0, 6.e-3)
plt.ylabel("Déplacement", color="r")
plt.plot(t, d, "o-r")
```

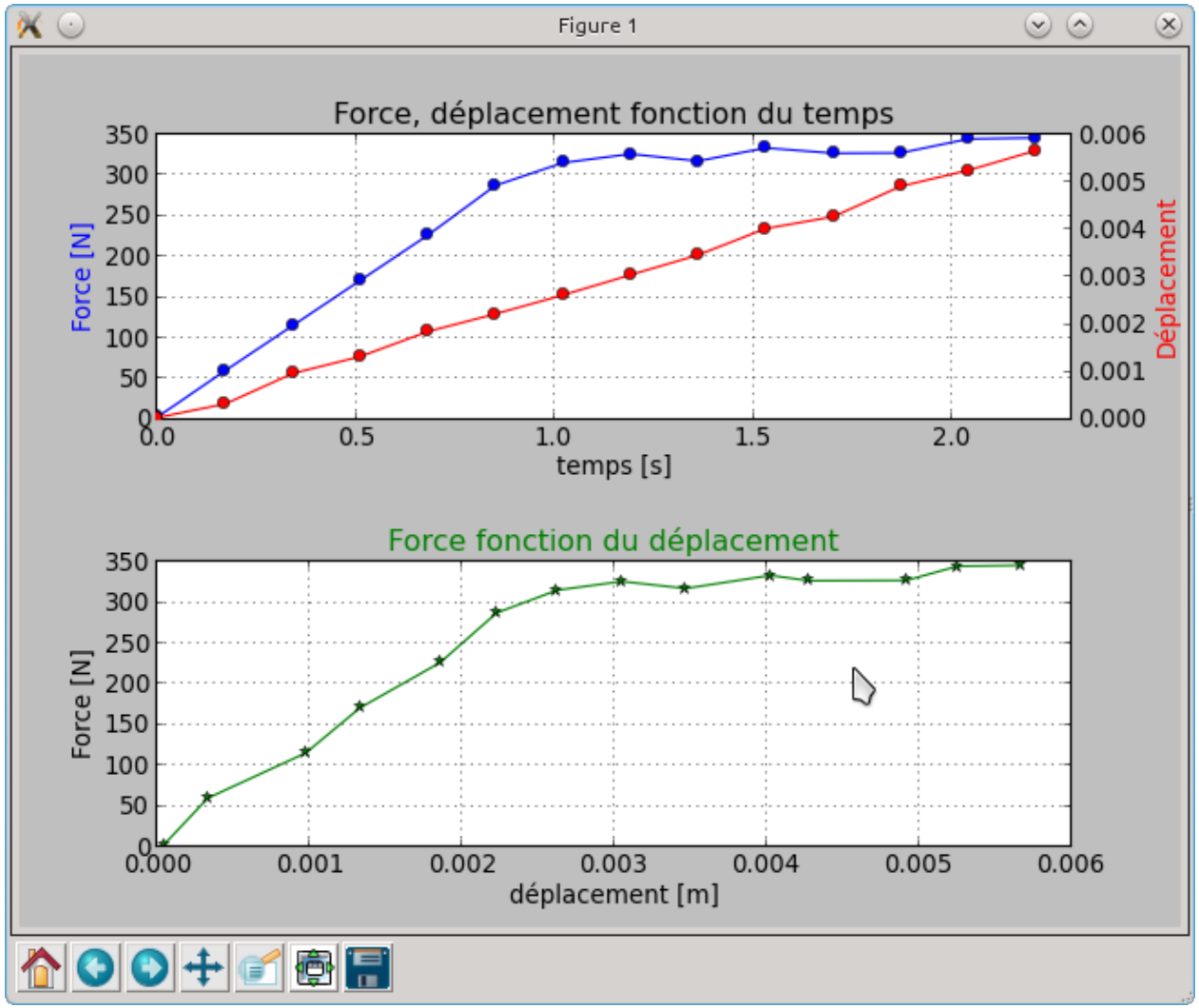
Tracé d'une courbe y=f(x)

[Plot_XY_3.py]

Plusieurs graphes sur une même `figure` :

```
...
plt.figure()
plt.subplots_adjust(hspace=0.5) # ajustement des espaces
plt.subplot(211)          # 2 lignes, 1 col, plot 1 : 211
plt.title("Force, déplacement fonction du temps")
plt.axis([0, 2.3, 0, 350]) ; plt.grid(True)
plt.xlabel("temps [s]")
plt.ylabel("Force [N]", color="b")
plt.plot(t, f,"o-b")

plt.twinx()
plt.axis([0, 2.3, 0, 6.e-3])
plt.ylabel("Déplacement", color="r")
plt.plot(t, d,"o-r")

plt.subplot(212)          # 2 lignes, 1 col, plot 2 : 212
plt.title("Force fonction du déplacement", color="g")
plt.axis([0, 6e-3, 0, 350.0]) ; plt.grid(True)
plt.xlabel("Déplacement [m]")
plt.ylabel("Force [N]")
plt.plot(d, f,"*-g")
```

Tracé d'une courbe y=f(x)



Tracé d'un histogramme

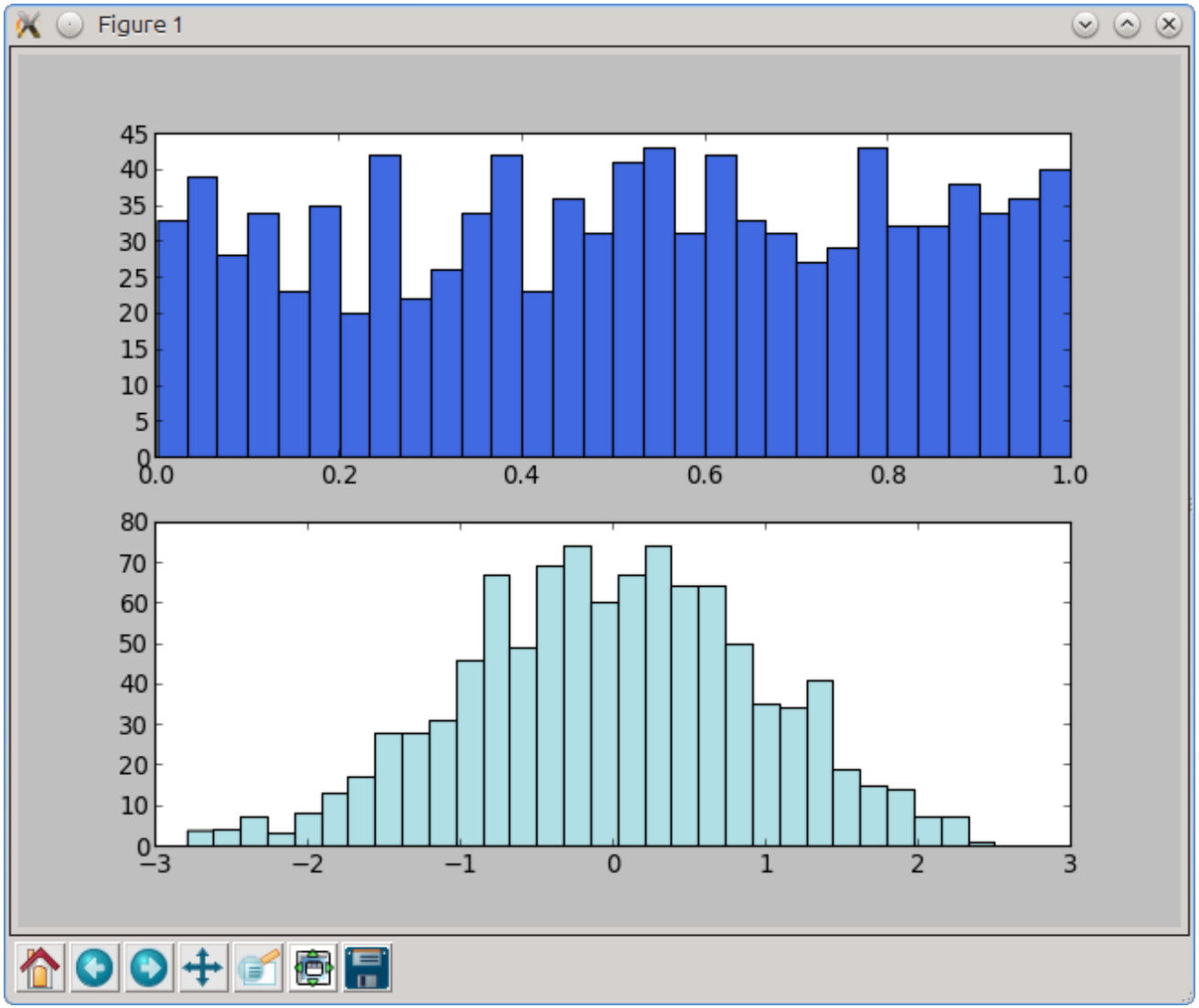
[Plot_Hist_1.py]

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

y = np.random.rand(1000)  # tirage aleatoire uniforme dans [0,1]
print(type(y))
print(y.shape)

plt.subplot(211)
plt.hist(y, 30, color="RoyalBlue")

y = np.random.randn(1000) # loi normale centree reduite
plt.subplot(212)
plt.hist(y, 30, color="PowderBlue")

plt.show()
```

couleurs : 'r','b' ou 0x10AABE (*Red, Green, Blue*), ou couleurs HTML

```
<class 'numpy.ndarray'>
(1000,)
```

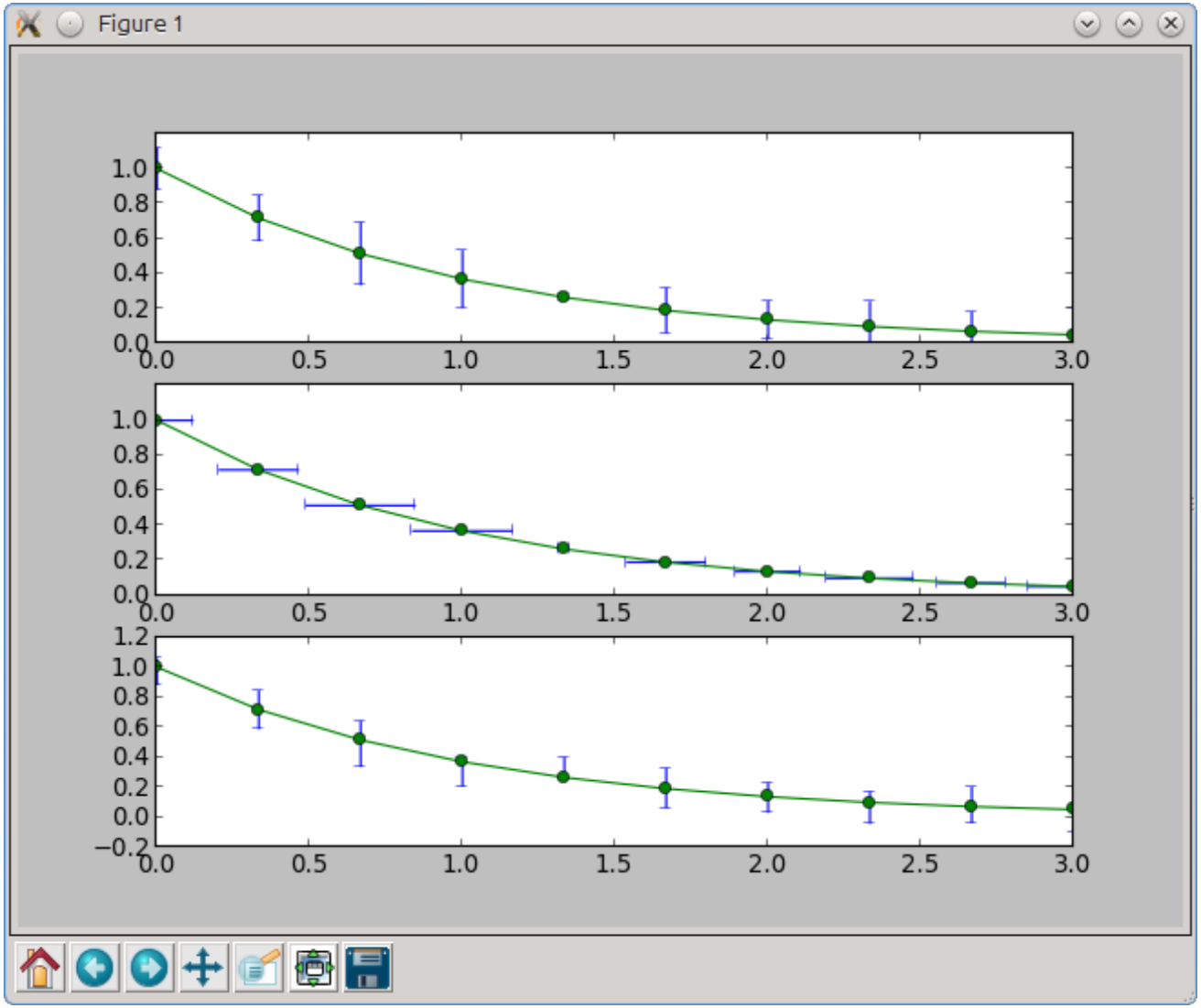
Tracé d'un histogramme



Tracé d'une courbe avec barres d'erreurs [Plot_ErrorBars_1.py]

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

x = np.linspace(0, 3, 10) # abscisses : 10 pts dans [0, 3]
y = np.exp(-x)            # ordonnées
e1 = 0.2*np.abs(np.random.rand(len(x)))
e2 = 0.2*np.abs(np.random.rand(len(x)))
# barre d"erreur Y symétrique
plt.axis([0, 3, 0, 1.2])
plt.subplot(311)
plt.errorbar(x, y, yerr=e1, fmt="go-", ecolor="b")
# barre d"erreur X symétrique
plt.axis([0, 3, 0, 1.2])
plt.subplot(312)
plt.errorbar(x, y, xerr=e1, fmt="go-", ecolor="b")
# barre d"erreur Y asymétrique
plt.axis([0, 3, 0, 1.2])
plt.subplot(313)
plt.errorbar(x, y, yerr=[e1,e2], fmt="go-", ecolor="b")
plt.show()
```

Tracé d'une courbe avec barres d'erreurs

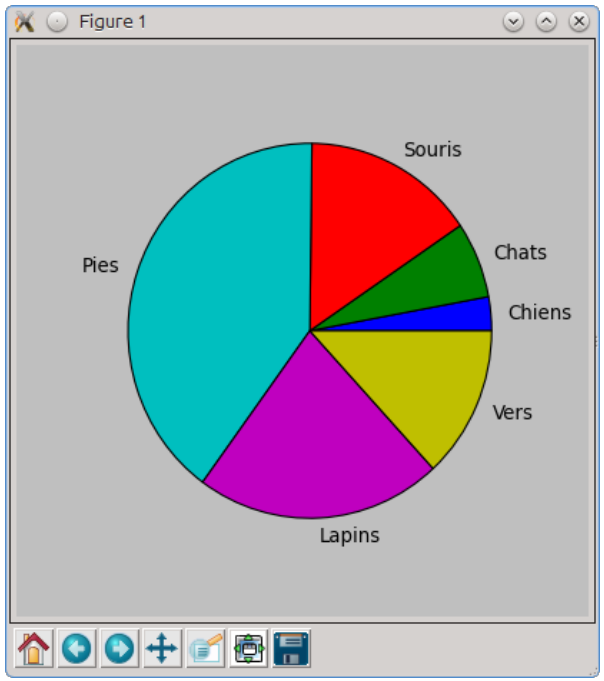


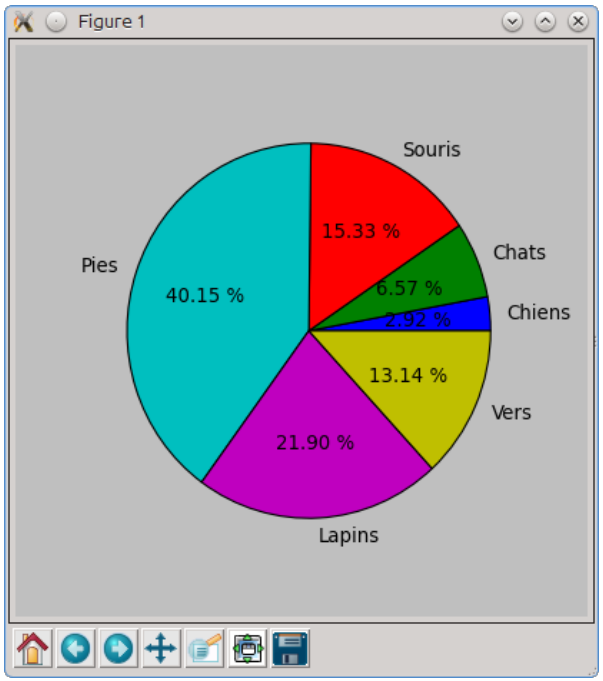
Tracé d'un 'camembert'

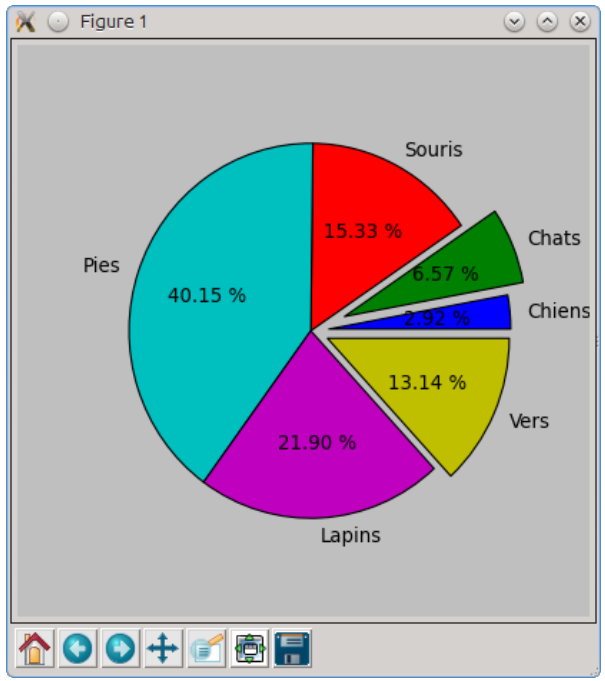
[Plot_PieChart_1.py]

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

x = [4, 9, 21, 55, 30, 18]

plt.figure(figsize=(5,5))
L = ["Chiens", "Chats","Souris", "Pies", "Lapins", "Vers"]
plt.pie(x, labels=L)
plt.show()

# Labels dans les 'portions':
plt.figure(figsize=(5,5))
plt.pie(x, labels=L, autopct="%.2f %%")
plt.show()

# Explode de certaines portions:
e = [0.1, 0.2, 0, 0, 0, 0.1]
plt.figure(figsize=(5,5))
plt.pie(x, labels=L, autopct="%.2f %%", explode=e)
plt.show()
```

Tracé d'un 'camembert'





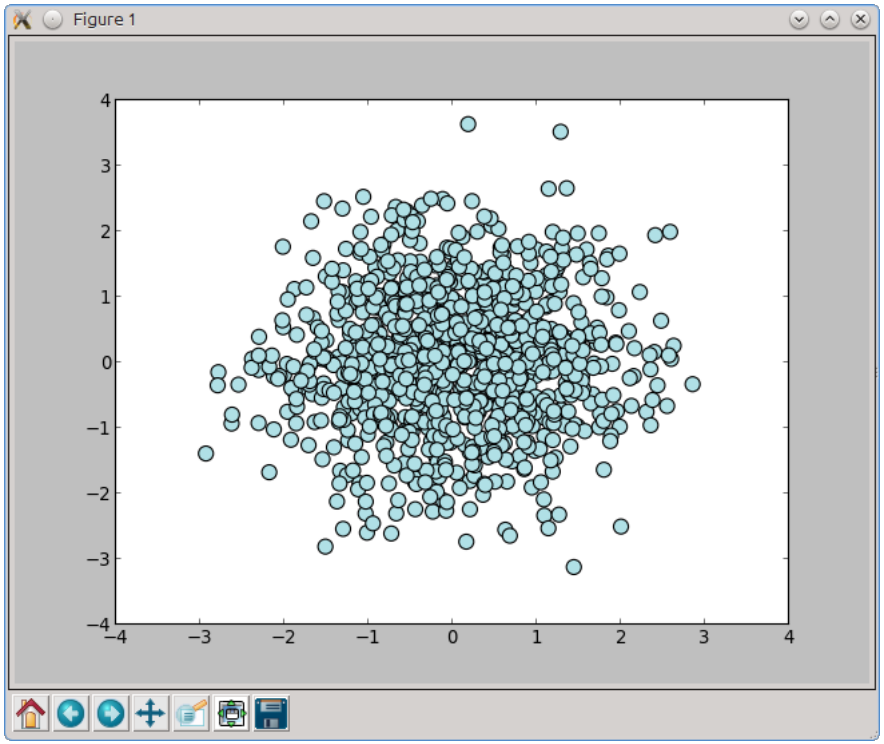


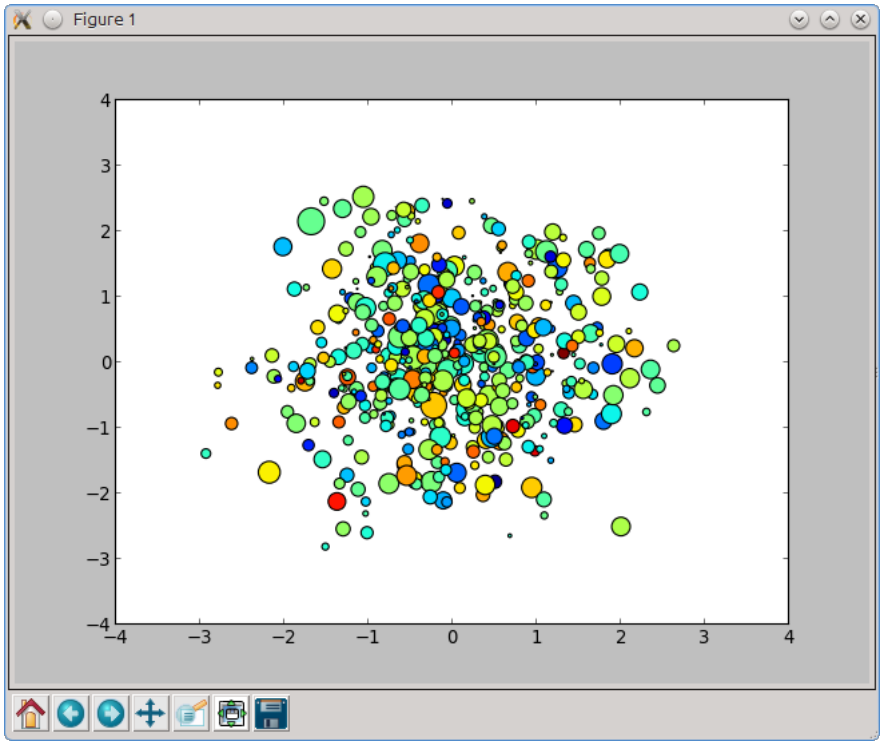
Tracé d'un nuage de points (Scatter)

[Plot_Scatter_1.py]

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

x = np.random.randn(1000) # 1000 tirages random (loi normale
y = np.random.randn(1000) # centrée réduite)

plt.scatter(x, y, s=100, c="PowderBlue")
plt.show()

sizes =  100*np.random.randn(1000)
colors = np.random.randn(1000)

plt.scatter(x, y, s=sizes, c=colors)
plt.show()
```

Tracé d'un nuage de points (Scatter)



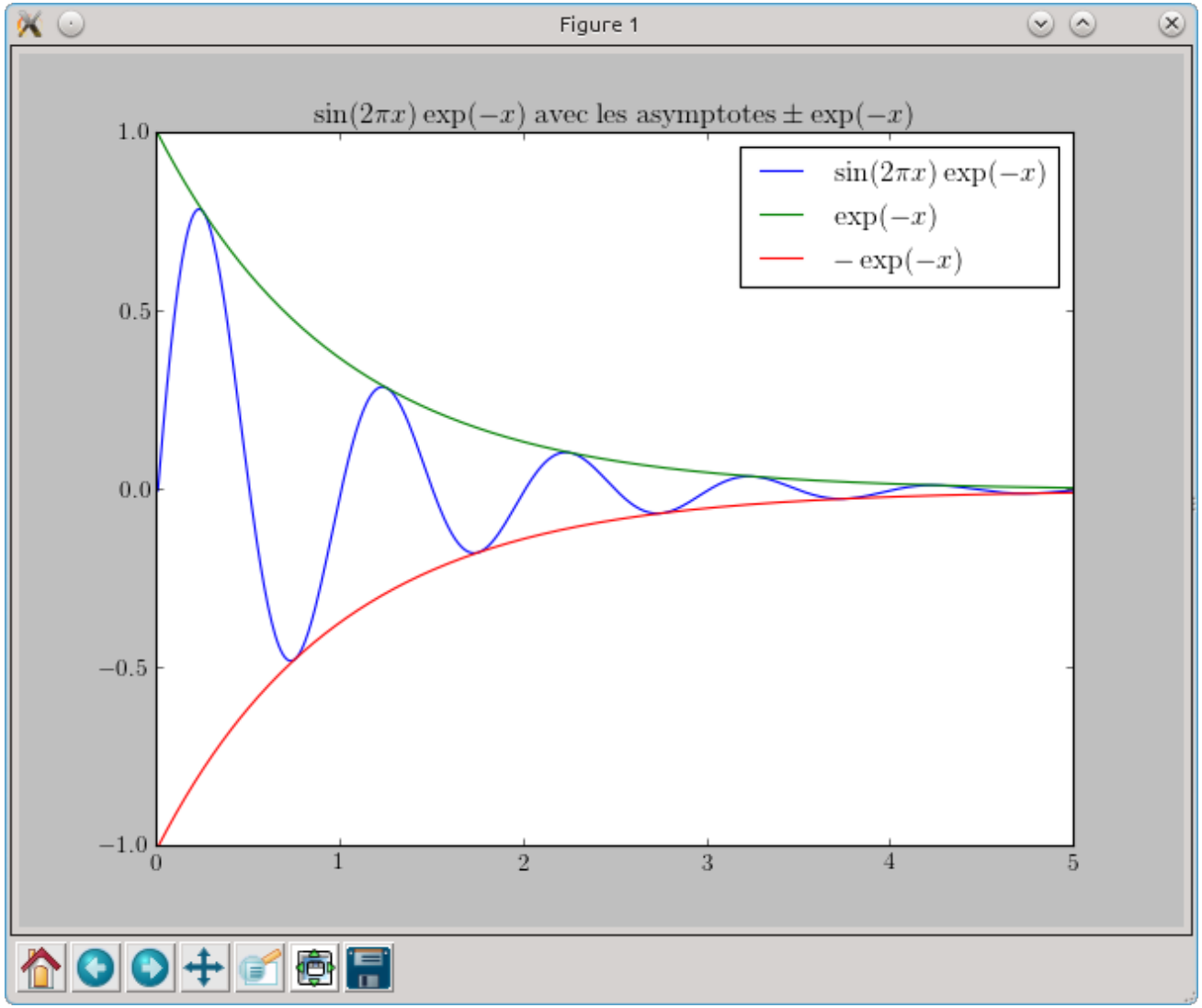


LaTeX dans les Tracés

[Plot_LaTeX_1.py]

```
from numpy import pi, arange, sin, exp

import matplotlib.pyplot as plt
import matplotlib as mpl

# validation de la compilation LaTeX :
mpl.rcParams["text.usetex"] = True

x = arange(0., 5., .01)                  # x values
y = [sin(2*pi*xx)*exp(-xx) for xx in x] # liste par compréhension

# Le texte pour LaTeX est préfixé par "r" (raw) :
plt.plot(x, y, label=r"$\sin(2\pi x)\exp(-x)$")
plt.plot(x,  exp(-x), label=r"$\ \exp(-x)$")
plt.plot(x, -exp(-x), label=r"$-\exp(-x)$")
t1  = r"$\sin(2\pi x)\exp(-x)\ \mathrm{avec\ les\ asymptotes} "
t1 += r"\pm\exp(-x)$"
plt.title(t1)
plt.legend()
plt.show()
```

LaTeX dans les Tracés



La classe ndarray du module numpy

- La classe **ndarray** : *n-dimensional array*

```
>>> import numpy as np
>>> m1 = np.ndarray((2,2)) ; print(m1)
[[ 0.00000000e+000  4.94944794e+173]
 [ 1.93390228e-309  4.10074486e-322]]
>>> print(m1[0,0], m1[0,1])
0.0 4.94944794e+173
>>> m1[0,1], m1[1,1] = 2, 4
>>> m1[0,0], m1[1,0] = 1, 3
>>> m1
array([[ 1.,  2.],
       [ 3.,  4.]])
```

- `m1` : matrice de float, 2 lignes, 2 col.
- Les éléments des `ndarray` ne sont pas initialisés !
- `m[i,j]` renvoie l'élément ligne `i`, colonne `j`

- Les fonctions numpy `zeros` et `ones`

```
>>> m1 = np.zeros(4) ; print(m1)
[ 0.,  0.,  0.,  0.]
>>> m2 = np.ones((2,3)) ; print(m2)
([[ 1.,  1.,  1.],
  [ 1.,  1.,  1.]])
>>> type(m1)
<class 'numpy.ndarray'>
```

- `m1` : vecteur à 4 composantes
- `m2` : matrice 2 lignes, 3 colonnes.

La classe ndarray du module numpy

- La fonction `array` convertit un objet `list` en objet `ndarray` :

```
>>> L1 = [1., 2., 3.] ; L2 = [3., 4., 5.]
>>> m1 = np.array(L1)
>>> L1 is m1
False
>>> print("m1:", m1)
m1: [[ 1.  2.  3.]]
>>> m2 = np.array([L1, L2])
>>> print(m2)
[[ 1.  2.  3.]
 [ 3.  4.  5.]]
```

- Les attributs `size` et `shape` donnent le nombre d'éléments et la dimension et d'un objet `ndarray` :

```
>>> m1.size, m2.size
(3, 6)
>>> m1.shape, m2.shape
((3,), (2, 3))
>>> m2.shape = 6
>>> print(m2)
[ 1.  2.  3.  3.  4.  5.]
```

- shape permet aussi de redimensionner un objet ndarray

La classe ndarray du module numpy

- La fonction `tolist` convertit un objet `ndarray` en objet `list` :

```
>>> m = np.ndarray((3,2))
>>> m.fill(5)
>>> m
array([[ 5.,  5.],
       [ 5.,  5.],
       [ 5.,  5.]])
>>> print(m)
[[ 5.  5.]
 [ 5.  5.]
 [ 5.  5.]]
>>> L = m.tolist() ; L
[[5.0, 5.0], [5.0, 5.0], [5.0, 5.0]]
```

- La fonction `linspace` crée un vecteur de `float` aux coordonnées régulièrement espacées :

```
 >>> np.linspace(0,10,5)
array([ 0., 2.5, 5., 7.5, 10.])
```

  - 5 nombres entre 0 et 10 inclus

Attention : comportement différent de `range` ou de `numpy.arange` pour la borne supérieure.

La classe ndarray du module numpy

- Indexation des objets `ndarray` -> comme les objets list :

```
>>> m2 = np.array([[ 1.,  2.,  3.],[ 4.,  5., 6.]])
```

```
>>> print(m2)
[[ 1.  2.  3.]
 [ 4.  5.  6.]]
>>> m2[0]
array([ 1.,  2.,  3.])
>>> m2[1]
array([ 4.,  5.,  6.])
>>> m2[0,:]
array([ 1.,  2.,  3.])
>>> m2[:,0]
array([ 1.,  4.])
>>> m2[:,1]
array([ 2.,  5.])
>>> m2[:-1,:-1]
array([[ 1.,  2.]])
```

- `[0]` 1re ligne
- `[1]` 2me ligne
- `[0,:]` 1re ligne, toutes les colonnes
- `[:,0]` toutes les lignes, 1re colonne
- `[:,1]` toutes les lignes, 2me colonne
- `[:-1,:-1]` toutes les lignes sauf la dernière, toutes les colonnes sauf la dernière

Lecture/écriture d'un fichier ASCII délimité avec numpy

Fonctions `loadtxt` et `savetxt` du module numpy [prog51.py]

```
import numpy as np

m1 = np.loadtxt("data1.txt")
print("m1:\n", m1)

m2 = np.loadtxt("data1.txt", usecols=(0,1))
print("m2:\n", m2)
```

```
m1:
[[  0.00000000e+00   3.32156600e+00   4.34020300e-05]
 [  1.70000000e-01   6.01348900e+01   3.29935900e-04]
 [  3.40000000e-01   1.15485200e+02   9.71830100e-04]
 [  5.10000000e-01   1.71006600e+02   1.32835300e-03]
 [  6.80000000e-01   2.26594600e+02   1.85442800e-03]]
m2:
[[  0.00000000e+00   3.32156600e+00]
 [  1.70000000e-01   6.01348900e+01]
 [  3.40000000e-01   1.15485200e+02]
 [  5.10000000e-01   1.71006600e+02]
 [  6.80000000e-01   2.26594600e+02]]
```

Lecture/écriture d'un fichier ASCII délimité avec numpy

Fonctions `loadtxt` et `savetxt` du module numpy [prog52.py]

```
import numpy as np
t1, c1, d1 = np.loadtxt("data1.txt", unpack=True)
print("temps :", t1)
print("force :", c1)
print("dépl. :", d1)

print("\nLecture avec usecols=(0,2) :")
t2, d2 = np.loadtxt("data1.txt", usecols=(0,2), unpack=True)
print("temps :", t2)
print("dépl. :", d2)
```

```
temps : [ 0.    0.17  0.34  0.51  0.68]
force : [   3.321566   60.13489   115.4852    171.0066    226.5946  ]
dépl. : [  4.34020300e-05  3.29935900e-04  ... 1.32835300e-03 1.85442800e-03]

Lecture avec usecols=(0,2) :
temps : [ 0.    0.17  0.34  0.51  0.68]
dépl. : [ 4.34020300e-05  3.29935900e-04  ...  1.32835300e-03 1.85442800e-03]
```

Lecture/écriture d'un fichier ASCII délimité avec numpy

Fonctions `loadtxt` et `savetxt` du module numpy [prog53.py]

```
import numpy as np
def read(fileName):                    # fonction pour relire
    print("fichier <"+fileName+">") # et afficher un fichier.
    print(np.loadtxt(fileName))
    return
# 9 points dans [1., 10.] et dans [1., 2.]
a1, a2 = np.linspace(1,10,9), np.linspace(1,2,9)
a1.shape = (3,3)
np.savetxt("f1.txt", a1, fmt="%f")
read("f1.txt")
a1.shape = 9
np.savetxt("f2.txt", np.transpose([a1,a2]), fmt="%12.6e", delimiter="\t")
read("f2.txt")
```

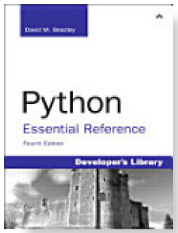
```
fichier <f1.txt> :
[[  1.      2.125   3.25 ]
 [  4.375   5.      6.625]
 [  7.75    8.875  10.   ]]
```

```
fichier <f2.txt> :
[[  1.      1.   ]
 [  2.125   1.125]
...
 [  8.875   1.875]
 [ 10.      2.   ]]
```

Références bibliographiques

http://docs.python.org/index.html
fr.openclassrooms.com/informatique/python/cours
fr.openclassrooms.com/informatique/cours/apprenez-a-programmer-en-python

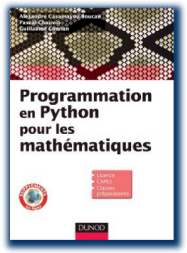
Apprenez à programmer en Python
Vincent Le Goff
Simple IT éd. (Le livre du zéro)
ISBN 979-10-90085-03-9
≈25 €



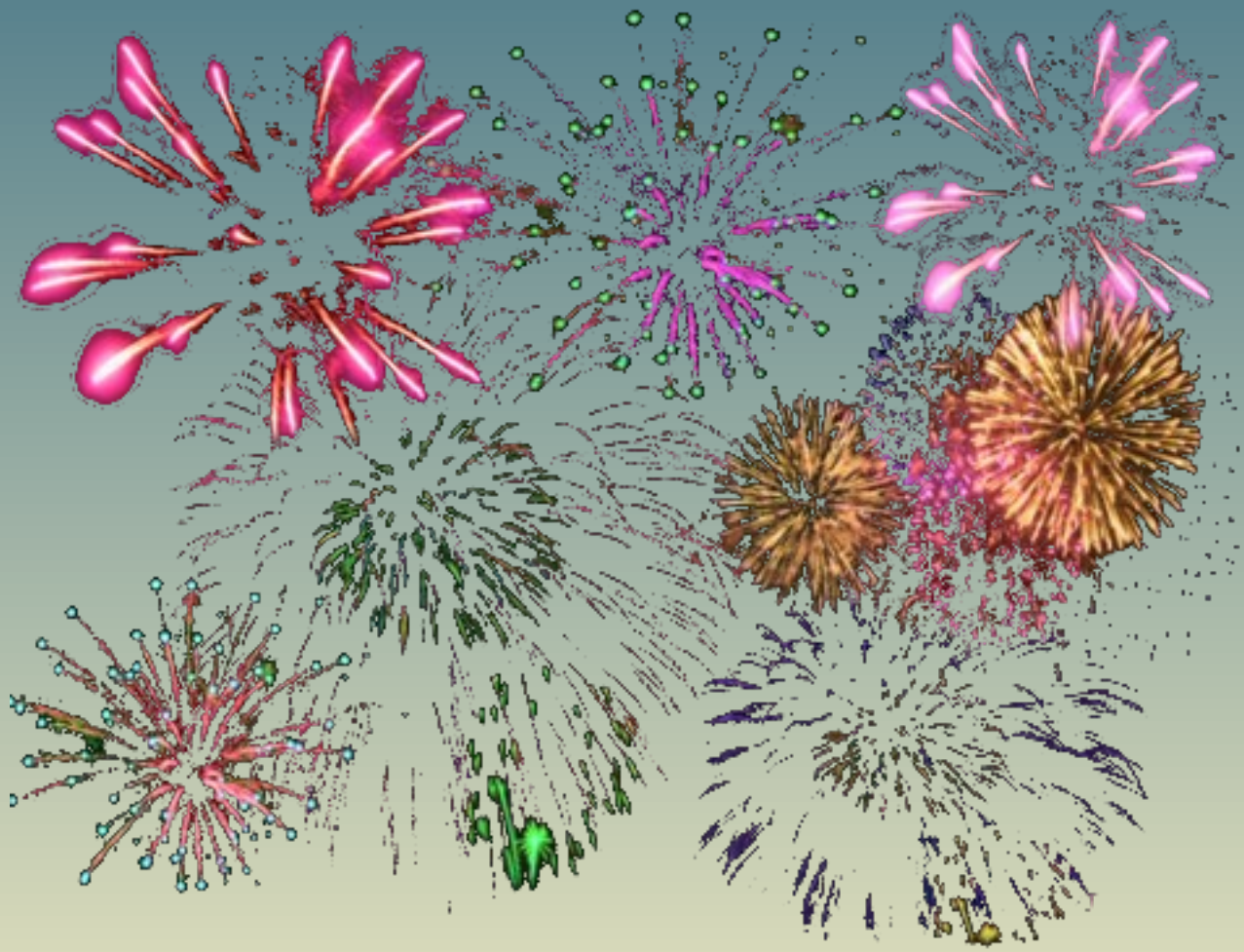
Python Essential Reference
David M. Beazley
Addison Wesley
ISBN 0-672-32978-4

Matplotlib for Python Developers
Sandro Tosi
PACTK publishing
ISBN 1847197906

Apprendre à programmer avec Python 3
Gérard Swinnen
Télécharger le PDF



Programmation en Python
pour les mathématiques
A. Casamayou-Boucau
Dunod
ISBN 978-2-10-057422-3



✉ jean-luc.charles@ensam.eu

✉ eric.ducasse@ensam.eu