Programmation Scientifique en Python

2015

Vincent.Favre-Nicolin@cea.fr

Objectifs de cette UE :

- (maîtriser un nouveau language)
- connaître les “ modules ” scientifiques
- et les appliquer à un micro-projet de recherche

Séances prévues

1.5 séance d'introduction à Python

1.5 séance d'introduction aux modules scientifiques en Python

2 séances consacrées à la réalisation d'un micro-projet personnel

Quel langage connaissez-vous ?

Fortran (77, 95...)

C

C++

basic

(pascal)

Langage de haut & bas niveau

- ***Assembleur***

```
        .global _start              ; D'après wikipedia.fr
BONJ:   .ascii  "Bonjour\n"       ; Definition en mémoire de la chaîne à afficher. \n correspond au saut de ligne
_start: mov     $4      , %eax    ; Mettre 4 dans le registre eax (appel système ''Write'')
        mov     $1      , %ebx    ; Mettre 1 dans le registre ebx (descripteur de fichier ''STDOUT'')
        mov     $BONJ   , %ecx    ; Mettre l'adresse mémoire de notre chaîne de caractère dans le registre ecx
        mov     $8      , %edx    ; Mettre la taille de la chaîne dans edx
        int     $0x80             ; Interruption 0x80, éxécutant un appel système sous Linux)

        mov     $1      , %eax    ; Mettre 1 dans eax (appel système ''Exit'')
        mov     $0      , %ebx    ; Mettre 0 dans ebx (valeur de retour du programme)
        int     $0x80             ; Interruption 0x80, éxécutant un appel système sous Linux)
```

- ***Langage de bas niveau (gestion mémoire, pointeurs,..) C C++ fortran etc...***

```
#include <iostream>
int main()
{
      std::cout << "Bonjour !" << std::endl;
      return 0;
}
```

- ***Langage de haut niveau (matlab, python+modules,...)***
  - ***Pas de gestion explicite de la mémoire***
  - ***Auto-déclaration des objets***

```
print “ Bonjour ”
```

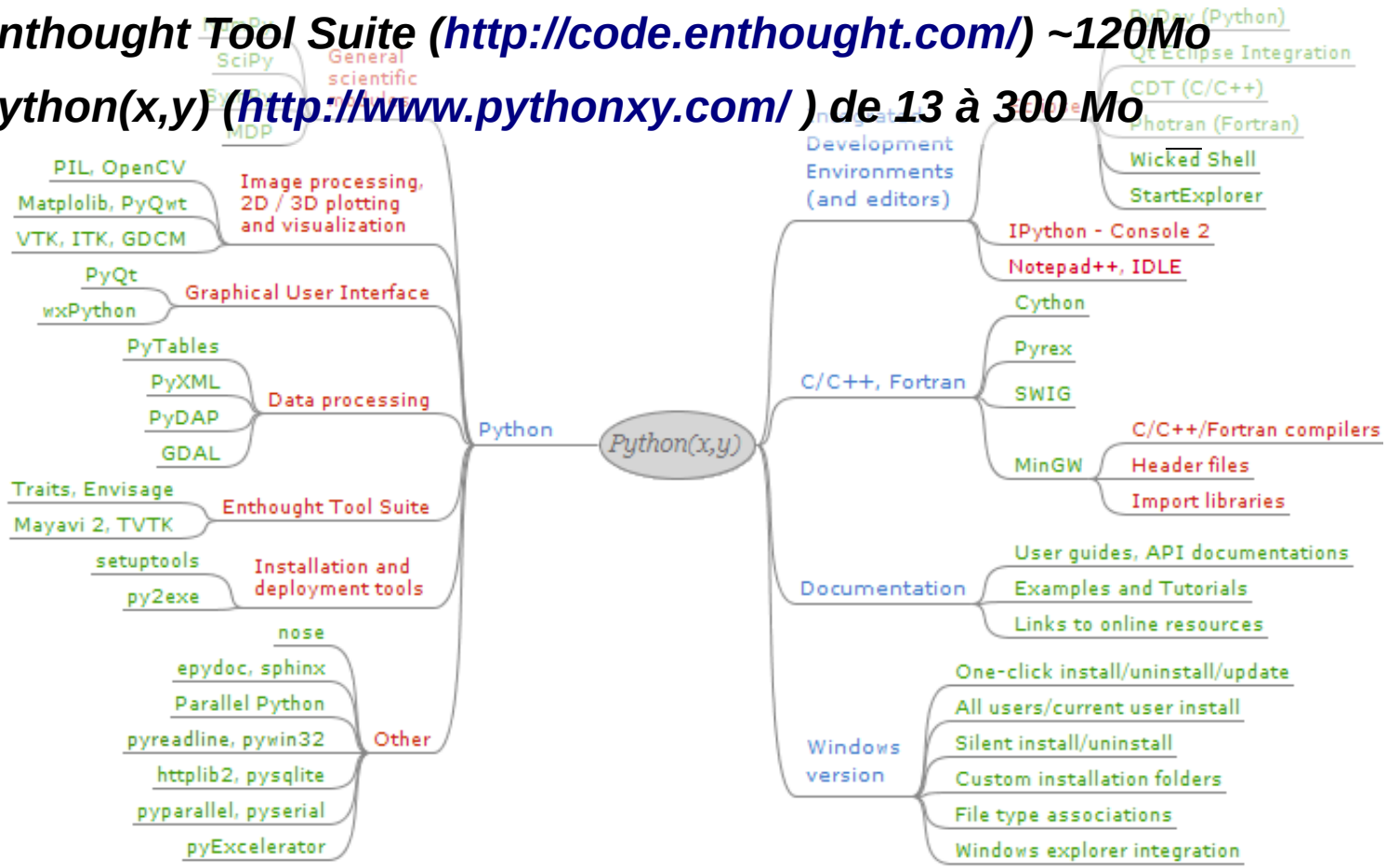
  - ***Présence (en général) d'un interpréteur de commande***

Langages : compilé / interprété / compilé au vol

- ***Compilé :***
  - ***Le programme doit être compilé avant d'être éxécuté***
- ***Interprété***
  - ***Le programme est lu au fur et à mesure de l'éxécution***
- ***Compilé au vol***
  - ***Le programme est automatiquement compilé à l'éxécution***

Installer Python sur son ordinateur

- ***Sous linux :***
  - ***Python est en général installé par défaut, et est livré avec de nombreuses librairies (Debian “ lenny ” comprend 1520 paquets avec le mot “ python ”)***
- ***Sous windows : installeurs gratuits (python + librairies scientifiques)***
  - ***Enthought Tool Suite (http://code.enthought.com/) ~120Mo***
  - ***Python(x,y) (http://www.pythonxy.com/) de 13 à 300 Mo***



Lancement de Python : en ligne de commande

- ***Sous Linux, ouvrir une console et taper “ python ”***

```
[vincent@d420 ~]$ python
Python 2.5.2 (r252:60911, Aug  5 2008, 15:33:24)
[GCC 4.2.3 (4.2.3-6mnb1)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- ***Il ne reste plus qu'à taper des commandes :***

```
>>> print 2+3
5
```

- ***On peut aussi utiliser “ ipython ” => recommandé !***

```
[vincent@d420 ~]$ ipython
Python 2.5.2 (r252:60911, Aug  5 2008, 15:33:24)
Type "copyright", "credits" or "license" for more information.

IPython 0.8.4 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object'. ?object also works, ?? prints more.

In [1]: print 2+3
5

In [2]:
```

Pour sortir du “ shell ” python : ctrl-d

Lancement de Python : éxécuter un fichier de commande

- ***Créer un fichier texte (avec*** kwrite ***ou*** kate***) qui s'appelle " prog.py ", qui contient :***

```
print 2+3
print " nous sommes mardi "
```

- ***Sauvegarder ce fichier dans un répertoire connu :***

  par exemple : /home/NFS/HOME/v/vfavreni/python/prog.py

- ***Puis éxécuter le programme :***

```
-bash-3.2$ cd /home/NFS/HOME/v/vfavreni/python/prog.py
-bash-3.2$ python prog.py
5
nous sommes mardi
-bash-3.2$
```

- ***Idem avec ipython***

```
-bash-3.2$ cd /home/NFS/HOME/v/vfavreni/python/prog.py
-bash-3.2$ ipython prog.py
5
nous sommes mardi
Python 2.5.2 (r252:60911, Aug  5 2008, 16:17:06)
Type "copyright", "credits" or "license" for more information.

IPython 0.8.1 -- An enhanced Interactive Python.
[...]
In [1]:
```

Variables

- ***Création de variable <u>sans déclaration</u> préalable !***

```
>>> a=1
>>> b=2
>>> print a+b
3
```

- ***Le “ type ” de chaque variable est automatiquement fixé***

```
>>> type(a)
<type 'int'>     # entier

>>> type(1.2)
<type 'float'>   # nombre réel à virgule flottante
```

=> typage “ fort ”, dynamique

- ***Une variable peut changer de type par simple affectation***

```
>>> a= ”Vive la Physique et la Chimie !”
>>> print a
Vive la Physique et la Chimie !
>>> type(a)
<type 'str'>     # string: chaîne de caractères
```

Types de base : nombres

- ***Entier (int ou long)*** 0,1, -1,100000000000
- ***Réel ('float')*** 1.0, 2.5, 3.14159, 1.4e10
- ***Complex ('complex')*** 1+1j, 2.5+3.75j

Opérations mathématiques

- ***Addition*** 1+2.0, 4-3j
- ***Soustraction*** 3-4, 10j-7
- ***Multiplication*** 5*4, 10.2*3.4e-9
- ***Division*** 1/2.0 89/2 ! division entière ou réelle...
- ***Puissance*** 2**8, 5.2**3.4, 3.2**(1+3.2j)

Types de base : chaînes de caratères (string)

- ‘’abc’’ ou ‘abc’
- ‘\n’ # retour à la ligne
- ‘abc’+’def’ -> ‘abcdef’
- 3*’abc’ -> ‘abcabcabc’
- ’ab cd e’.split() -> [’ab’,’cd’,’e’]
- ’1,2,3’.split(’,’) -> [’1’, ’ 2’, ’ 3’]
- ’,’.join([’1’,’2’]) -> ’1,2’ # ajoute , entre les éléments
- ’ a b c ’.strip() -> ’a b c’ # enlève les espaces de fin et début
- ’text’.find(’ex’) -> 1 #recherche
- ’Abc’.upper() -> ’ABC’
- ’Abc’.lower() -> ’abc’
- **Conversion vers des nombres:** int(’2’), float(’2.1’)
- **Conversion vers du texte** : str(3), str([1, 2, 3])

Types de base : listes

- **Création**

```
>>>a=[1,2,3, ’blabla’,[9,8]]
```

- **Concatenation (pas addition… voir array)**

```
>>>[1,2,3]+[4,5]
[1,2,3,4,5]
```

- **Ajout d’un élément**

```
>>>a.append('test')
[1,2,3, ’blabla’,[9,8],’test’]
```

- **Longueur**

```
>>>len(a)
6
```

range([start,] stop[, step]) -> liste d'entiers

```
>>>range(5)
[0,1,2,3,4]
```

- **Indexation simple**

```
>>>a[0]
1
```

- **Indexation multiple**

```
>>> a[4][1]
8
```

- **Définition d’un élément**

```
>>> a[1]=1
>>> a
[1, 1, 3, 'blabla', [9, 8], 'test']
```

- **Indices négatifs**

```
>>> a[-1]
'test'
>>> a[-2]
[9, 8]
```

- **Parties d’une liste (liste[lower:upper])**

```
>>>a[1:3]
[1,3]
>>> a[:3]
[1, 1, 3]
```

! Les indices commencent toujours à 0 !

Aide en ligne :
help(*nom de l'objet*)

>>> help(list)

```
Help on class list in module __builtin__:

class list(object)
 |  list() -> new list
 |  list(sequence) -> new list initialized from sequence's items
 |
 |  Methods defined here:
 |
 |  __add__(...)
 |      x.__add__(y) <==> x+y
 |
 |  __contains__(...)
 |      x.__contains__(y) <==> y in x
 |
 |  __delitem__(...)
 |      x.__delitem__(y) <==> del x[y]
 |
 |  __delslice__(...)
 |      x.__delslice__(i, j) <==> del x[i:j]
 |
 |      Use of negative indices is not supported.
 |
 |  __eq__(...)
 |      x.__eq__(y) <==> x==y
 |
```

L'aide en ligne est disponible pour toutes les fonctions, types, modules.
En général elle est en <u>anglais</u>

Dans ipython, mettre le nom de la variable suivi de '?' donne accès à l'aide

```
            [...]
 |  append(...)
 |      L.append(object) -- append object to end
 |
 |  count(...)
 |      L.count(value) -> integer -- return number of occurrences of value
 |
 |  extend(...)
 |      L.extend(iterable) -- extend list by appending elements from the iterable
 |
 |  index(...)
 |      L.index(value, [start, [stop]]) -> integer -- return first index of value
 |
 |  insert(...)
 |      L.insert(index, object) -- insert object before index
 |
 |  pop(...)
 |      L.pop([index]) -> item -- remove and return item at index (default last)
 |
 |  remove(...)
 |      L.remove(value) -- remove first occurrence of value
 |
 |  reverse(...)
 |      L.reverse() -- reverse *IN PLACE*
 |
 |  sort(...)
 |      L.sort(cmp=None, key=None, reverse=False) -- stable sort *IN PLACE*;
 |      cmp(x, y) -> -1, 0, 1
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:

                [...]
```

Booléens & conditions

- **Booléen** : “ True ” et “ False ”
-
- Un nombre nul <-> False, non nul <-> True

```
>>> if True: print “ toto ”
...
toto

>>> if 0:
...    print “ toto ”   # indentation !
... else:
...    print “ tata ”
...    print “ et voila ”
...
tata
et voila
```

L'indentation (espaces en début de ligne) remplace les accolades en python.

Pour sortir d'une boucle, d'une condition, il suffit de revenir à l'indentation précédente

Boucles for

```
for <variable> in <liste> :
    code
    code
```

```
>>> for i in [1,2,'a'] :
...     print i
...
1
2
'a'

>>> for i in range(4) :    # NB: mieux vaut utiliser xrange
...     print i
...
0
1
2
3
```

Boucles while

```
while <condition> :
    code
    code
```

```
>>> i=0
>>> while i<5:          # comparateurs: <  >  <=  >=  ==
...     print i
...     i=i+1
...
0
1
2
3
4
```

Programmation avec Python : 1 – ligne par ligne

```
>>> print “ bonjour je suis un programme python ”
bonjour je suis un programme python
>>> import math
>>> print math.sin(3.14159)
2.65358979335e-06
>>> for lettre in ['a','b','c']: print lettre
a
b
c
```

Utile pour faire des tests rapides
... ou pour utiliser python comme **super-calculatrice**

Programmation avec Python : 2 – avec des fonctions

```
>>> def EcritPhrase(s):
...   print s
...
>>> EcritPhrase(“ bonjour je suis un programme python ”)
bonjour je suis un programme python

>>> def parab(x): return 2* x**2 + 4*x +1.5   # Parabole
...
>>> print parab(12)
337.5
```

Utile lorsqu'on doit répéter une même série d'instructions plusieurs fois

Programmation avec Python : 3 – orientée objet

```
>>> class polynome:
...     coeffs=[]
...     def valeur(self,x):
...         v=self.coeffs[0]
...         for i in xrange(1,len(self.coeffs)):
...             v+= self.coeffs[i] * x**i
...         return v
...
>>> p1=polynome()
>>> p1.coeffs=[0,0,2]
>>> print p1.valeur(0),p1.valeur(1),p1.valeur(2)
0 2 8

>>> p2=polynome()
>>> p2.coeffs=[1,0,0,1]
>>> print p2.valeur(0),p2.valeur(1),p2.valeur(2)

1 2 9
```

Utile lorsqu'on doit traiter des données complexes, avec de multiples usages

Python interactif: ipython

- ***Taper “ ipython ” dans une console :***
  - ***Rappelle de l'historique avec ↑ ↓ (même sessions précédentes)***
  - ***Documentation d'objets avec “ nom? ”***
  - ***Code d'un objet ou d'une fonction avec “ nom?? ”***
  - ***Complétion: commencer à taper un nom de variable + tabulation***
  - ***Code et coloration syntaxique (en cas d'erreur)***
  - ***Gestion des librairies graphiques en parallèle...***

Modules

- ***Python est un langage modulaire***
- ***Toutes les fonctions & types d'objets complexes sont rangés dans des modules :***

```
>>> import math              # " import " + nom du module
>>> print math.sin(1.2)      # NomDuModule.NomDelaFonction(arguments)
0.93203908596722629
>>> help(math)               # aide sur le module

>>> from math import *       # " from " + nom du module + " import *"
>>> print sin(1.2)
0.93203908596722629
```

Importance des modules

- Python est un langage populaire dans de nombreux domaines :
  - ***Scientifique***
  - ***Web (google, certains wiki, zope,...)***
  - ***Accès aux base de données***
  - ***Nombreux programmes écrits en python***

- ***... Car la plupart des libraries (scientifiques ou autres) sont interfacées avec le langage Python :***
  - ***=> pas de besoin de réinventer la roue !***
  - ***Tout code existant en c/c++/fortran est utilisable en python***
  -
  -
  - ***La semaine prochaine : présentation des modules* scipy, numpy & matplotlib**

Liens utiles

- ***http://www.python.org*** ***Le site officiel de python***
- ***http://wiki.python.org/moin/NumericAndScientific*** ***prog./libr. scientifiques***
- ***http://www.scipy.org*** ***Scientific python***
- ***http://scipy.org/Cookbook*** ***Recettes en scipy***
- ***http://www.scipy.org/Topical_Software*** ***Programmes liés à scipy, par thème***
- ***http://matplotlib.sourceforge.net/*** ***Graphisme scientifique***
- ***http://www.greenteapress.com/thinkpython/thinkpython.html*** ***Livre gratuit (*)***
- ***http://www.limsi.fr/Individu/pointal/python/pqrc/*** ***Python reference card (anglais)***
-
- ***Google :***
  - ***“ python physics ” -> 2.8 M liens...***
  - ***“ python molecule quantum ” -> 42000 résultats***
  - ***...***

Partie II : Calcul Scientifiques

Utilisation de numpy, scipy, matplotlib, mayavi :

- ***tracé de graphiques 1D, 2D, 3D...***
- ***Affinements***
- ***Résolution d'équations***
- ***etc...***

Calculs mathématiques

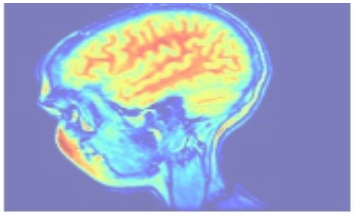
Python inclus les modules 'math' et 'cmath' (math pour complexes) :

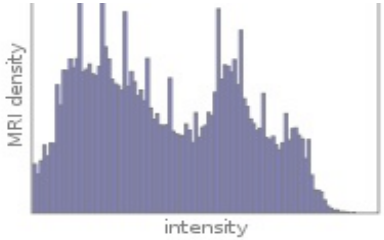
```
In [3]: import math
In [4]: print math.sin(math.pi/4)
0.707106781187
In [5]: import cmath
In [6]: print cmath.sin(2+1j)
(1.40311925062-0.489056259041j)
```

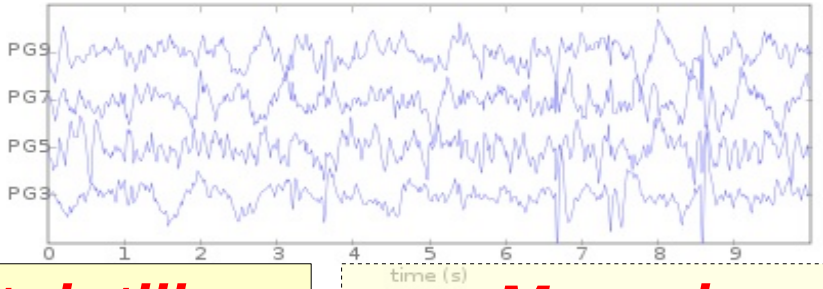
Mais :

- ***ces modules sont limités à des calculs simples (une valeur à la fois)***
- ***Il faut utiliser un module différent pour les nombres complexes***

Modules scientifiques







scipy :
calculs scientifiques avancés (affinement, transformées, résolution, intégration,...)

numpy :
opérations élémentaires sur des tableaux

Matplotlib (pylab) :
graphisme 1D/2D

Mayavi (tvtk, mlab)
graphisme 3D

Python

Librairies C/C++ Fortran
blas / lapack / VTK / wxWidgets

tableaux

Les données pour des calculs sont à ranger dans des vecteurs (numpy)

```
In [1]: import numpy
In [2]: a=numpy.array([1,2,3])        # création directe
In [3]: print a
[1 2 3]
```

Les calculs sur les tableaux s'appliquent à tous les éléments

```
In [4]: print a**2
[1 4 9]
In [5]: print numpy.sin(a*numpy.pi/2)
[  1.00000000e+00   1.22464680e-16  -1.00000000e+00]
```

Autres créations de tableaux :

```
In [10]: print numpy.arange(0,0.7,0.1)
[ 0.   0.1  0.2  0.3  0.4  0.5  0.6]

In [10]: print numpy.linspace(0,0.7,8)
[ 0.   0.1  0.2  0.3  0.4  0.5  0.6, 0.7]
```

Création de tableaux

...Autres créations de tableaux :

```
In [17]: print numpy.ones((2,2))                # tableau 2x2 remplis de 1
[[ 1.  1.]
 [ 1.  1.]]


In [18]: print numpy.zeros((2,2),dtype=int)    # tableau 2x2 d'entiers égaux à 0
[[0 0]
 [0 0]]
```

Types de base des tableaux: bool, int, uint, float complex

Types avancés (entiers signés): int8, int16, int32, int64
Types avancés (entiers non signés): uint8, uint16, uint32, uint64
Types avancés (réels flottants): float32, float64, float128
Types avancés (complexes): complex64, complex128, complex128

Dans un tableau, le type des données est identique pour tous les éléments (contrairement à une liste).

Tableaux aléatoires

```
In [16]: print numpy.random.uniform(0,2,size=(4,4))    # Distribution uniforme
[[ 1.58095991  0.15385365  0.82891886  0.0390148 ]     # entre 0 et 2
 [ 1.00491457  0.35195621  1.76908828  1.40046406]     # tableau 4x4
 [ 0.17771713  1.59835664  0.82339367  1.94511778]
 [ 0.89480039  1.31375333  1.26548763  0.83465617]]
```

```
In [17]: print numpy.random.normal(0,2,size=(4,4))     # Distribution normale
[[-4.20057151 -3.74078452 -2.68678707  1.31524717]     # moyenne=0, sigma=2
 [ 0.94090408  0.54095026  1.12982079 -0.36770016]
 [ 1.54574406  0.33812108  2.20872166  0.74243889]
 [-1.03764455  2.97095545  0.10432429 -0.31398585]]
```

```
Astuce : si on veut un vecteur de 100 éléments, avec 30% de 1 et 70% de zéros :
In [4]: (numpy.random.uniform(0,1,(100))>0.7).astype(numpy.float)
Out[4]:
array([ 0.,  1.,  0.,  1.,  1.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,
        0.,  1.,  1.,  0.,  0.,  1.,  0.,  0.,  0.,  1.,  0.,  0.,  1.,
        0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  1.,  1.,
        0.,  1.,  1.,  0.,  1.,  1.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  1.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        1.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  1.,  1.,  0.,
        1.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,
        0.,  0.,  1.,  0.,  0.,  0.,  0.,  1.,  0.])
```

Intérêt des tableaux : calcul vectoriel & rapidité

```
Un calcul simple :
import scipy
import time      # Module temps
taille=100000

# Version avec une boucle en Python
nbiter=10
a=range(taille) # Avec une boucle
b=range(taille)
c=range(taille)
t1=time.time()
for i in xrange(nbiter):
   for i in xrange(taille):
      c[i]=a[i]+b[i]

t2=time.time()
mflops=(taille*nbiter)/(t2-t1)/1e6
print "Boucle : %6.3f Mflops"%(mflops)
```

- Nécessité d'écrire une boucle
- 2.083 Mflops (Q6600 2.4GHz)

```
# Version avec des tableaux
nbiter=1000
a=scipy.ones(taille)
b=scipy.ones(taille)
c=scipy.ones(taille)
t1=time.time()
for i in xrange(nbiter):
   c=a+b

t2=time.time()
mflops=(taille*nbiter)/(t2-t1)/1e6
print "Tableau : %6.3f Mflops"%(mflops)
```

- Pas de boucle pour c=a+b
- 456.287 Mflops (Q6600 2.4GHz)
- **Vitesse * 220 !**

... et en C++ ?

- Code beaucoup plus long
- Risque d'erreurs avec des pointeurs
- 476.19 Mflops (Q6600 2.4GHz, -03)

Tableaux & matrices

Un tableau n'est pas une matrice !

```
In [9]: a= numpy.resize(numpy.arange(9),(3,3))

In [10]: b=numpy.resize(numpy.arange(10,19),(3,3))

In [11]: print a,"\n",b
[[0 1 2]
 [3 4 5]
 [6 7 8]]
[[10 11 12]
 [13 14 15]
 [16 17 18]]

In [12]: print a*b
[[  0  11  24]
 [ 39  56  75]
 [ 96 119 144]]
```

Les calculs se font toujours élément par élément pour des tableaux (array)

Pour une multiplication matricielle :

```
In [28]: print numpy.dot(a,b)
[[ 45  48  51]
 [162 174 186]
 [279 300 321]]
```

... ou utiliser le module numpy.matrix

Algèbre linéaire : scipy.linalg

```
In [1]: from numpy import matrix
In [2]: from scipy.linalg import inv, det, eig
In [3]: A=matrix([[1,1,1],[4,4,3],[7,8,5]])   # matrice 3x3
In [4]: b = matrix([1,2,1]).transpose()       # matrice 3x3

In [5]:print det(A)      # Déterminant non nul ?
1.0
In [6]: print inv(A)*b   # Solution de Ax=b
[[ 1.]
 [-2.]
 [ 2.]]

In [7]: print eig(A)     # vecteurs et valeurs propres
(array([ 10.57624887+0.j        ,  -0.28812444+0.1074048j,
        -0.28812444-0.1074048j]), array([[-0.14056873+0.j        ,
-0.58724949-0.17776272j,
        -0.58724949+0.17776272j],
       [-0.48042175+0.j        ,  0.0035321 +0.16590709j,
         0.0035321 -0.16590709j],
       [-0.86569936+0.j        ,  0.77201089+0.j        ,  0.77201089-
0.j        ]]))
```

Algèbre linéaire : scipy.linalg

```
Linear Algebra Basics:

inv --- Find the inverse of a square matrix
solve      --- Solve a linear system of equations
solve_banded --- Solve a linear system of equations with a banded matrix
det --- Find the determinant of a square matrix
norm--- matrix and vector norm
lstsq      --- Solve linear least-squares problem
pinv--- Pseudo-inverse (Moore-Penrose) using lstsq
pinv2      --- Pseudo-inverse using svd

     Eigenvalues and Decompositions:

eig --- Find the eigenvalues and vectors of a square matrix
eigvals    --- Find the eigenvalues of a square matrix
eig_banded --- Find the eigenvalues and vectors of a band matrix
eigvals_banded --- Find the eigenvalues of a band matrix
lu  --- LU decomposition of a matrix
lu_factor  --- LU decomposition returning unordered matrix and pivots
lu_solve   --- solve Ax=b using back substitution with output of lu_factor
svd --- Singular value decomposition of a matrix
svdvals    --- Singular values of a matrix
diagsvd    --- construct matrix of singular values from output of svd
orth--- construct orthonormal basis for range of A using svd
cholesky   --- Cholesky decomposition of a matrix
cho_factor --- Cholesky decomposition for use in solving linear system
cho_solve  --- Solve previously factored linear system
qr  --- QR decomposition of a matrix
schur      --- Schur decomposition of a matrix
rsf2csf    --- Real to complex schur form
hessenberg --- Hessenberg form of a matrix

     matrix Functions:

expm--- matrix exponential using Pade approx.
expm2      --- matrix exponential using Eigenvalue decomp.
```

Classe scipy.sparse pour des matrices discrètes

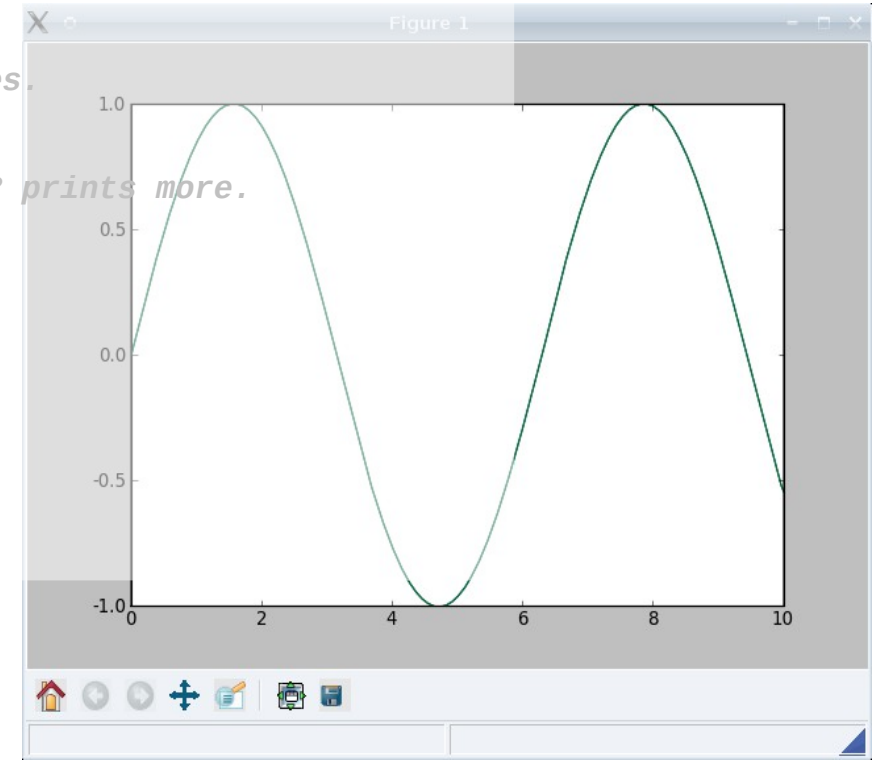
Graphiques 1D avec matplotlib

Matplotlib ***: librairie graphique en python, clone les fonctionnalités de Matlab.***
http://matplotlib.sourceforge.net
Nombreux exemples @ http://matplotlib.sourceforge.net/screenshots.html

Pour l'utiliser : lancer ipython avec l'option " -pylab " :

```
[vincent@d420 doc]$ ipython -pylab
Python 2.5.2 (r252:60911, Aug  5 2008, 15:33:24)
Type "copyright", "credits" or "license" for more information.

IPython 0.8.4 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object'. ?object also works, ?? prints more.

  Welcome to pylab, a matplotlib-based Python environment.
  For more information, type 'help(pylab)'.

In [1]: x=linspace(0,10,1000)      # de 0 à 10, 1000 points

In [2]: plot(x,sin(x))             # tracé de la courbe
Out[2]: [<matplotlib.lines.Line2D instance at 0x8f675ac>]

In [3]: savefig("sinus.png")
```

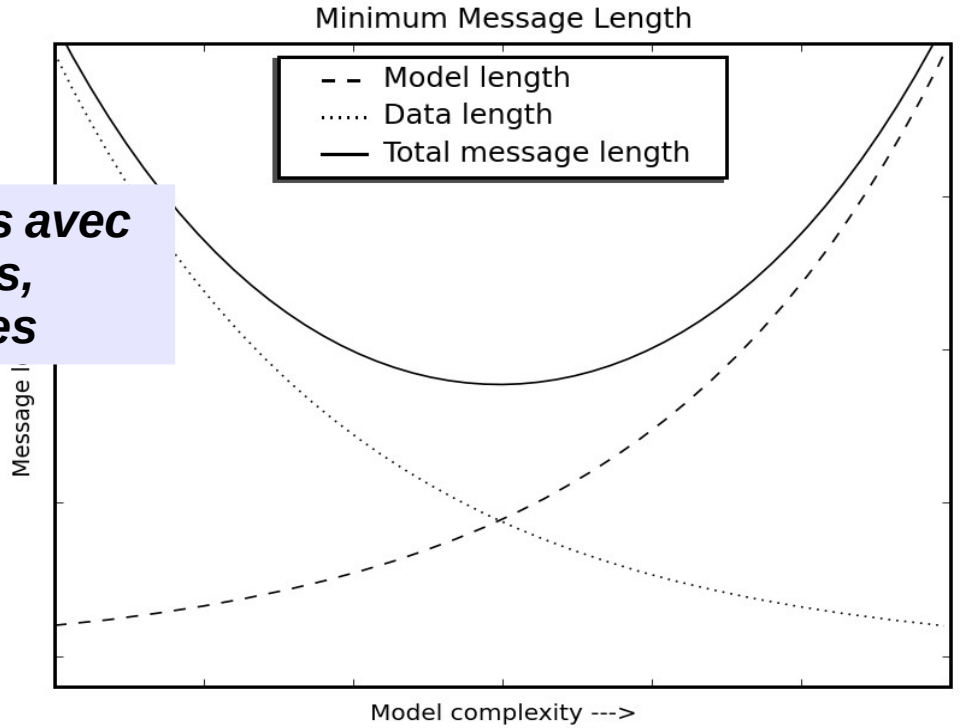
ipython -pylab éxécute automatiquement
" from numpy import * "



Graphiques 1D avec matplotlib

```
# Thanks to Charles Twardy for this example
#
#See http://matplotlib.sf.net/examples/legend_demo2.py for an example
#controlling which lines the legend uses a

from pylab import *

a = arange(0,3,.02)
b = arange(0,3,.02)
c = exp(a)
d = c.tolist()
d.reverse()
d = array(d)

ax = subplot(111)
plot(a,c,'k--',a,d,'k:',a,c+d,'k')
legend(('Model length', 'Data length', 'To
       'upper center', shadow=True)
ax.set_ylim([-1,20])
ax.grid(0)
xlabel('Model complexity --->')
ylabel('Message length --->')
title('Minimum Message Length')
ax.set_xticklabels([])   # Pas de coordonné
ax.set_yticklabels([])
```

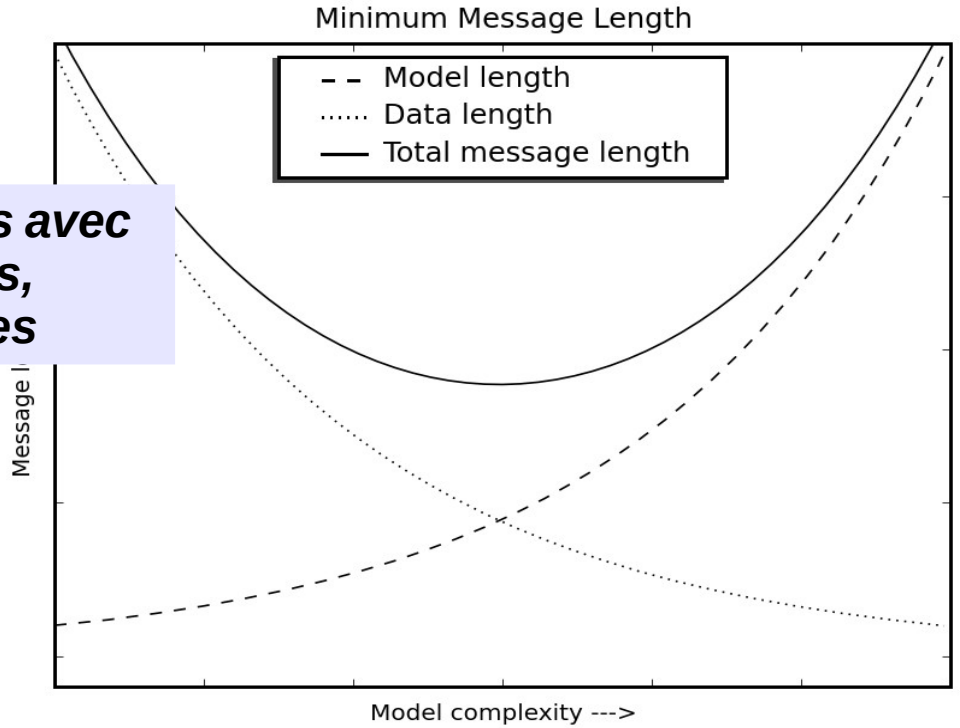
Tracer plusieurs courbes avec des styles différents, des titres & légendes



Graphiques 1D avec matplotlib

```
# Thanks to Charles Twardy for this example
#
#See http://matplotlib.sf.net/examples/legend_demo2.py for an example
#controlling which lines the legend uses a

from pylab import *

a = arange(0,3,.02)
b = arange(0,3,.02)
c = exp(a)
d = c.tolist()
d.reverse()
d = array(d)

ax = subplot(111)
plot(a,c,'k--',a,d,'k:',a,c+d,'k')
legend(('Model length', 'Data length', 'To
       'upper center', shadow=True)
ax.set_ylim([-1,20])
ax.grid(0)
xlabel('Model complexity --->')
ylabel('Message length --->')
title('Minimum Message Length')
ax.set_xticklabels([])   # Pas de coordonné
ax.set_yticklabels([])
```

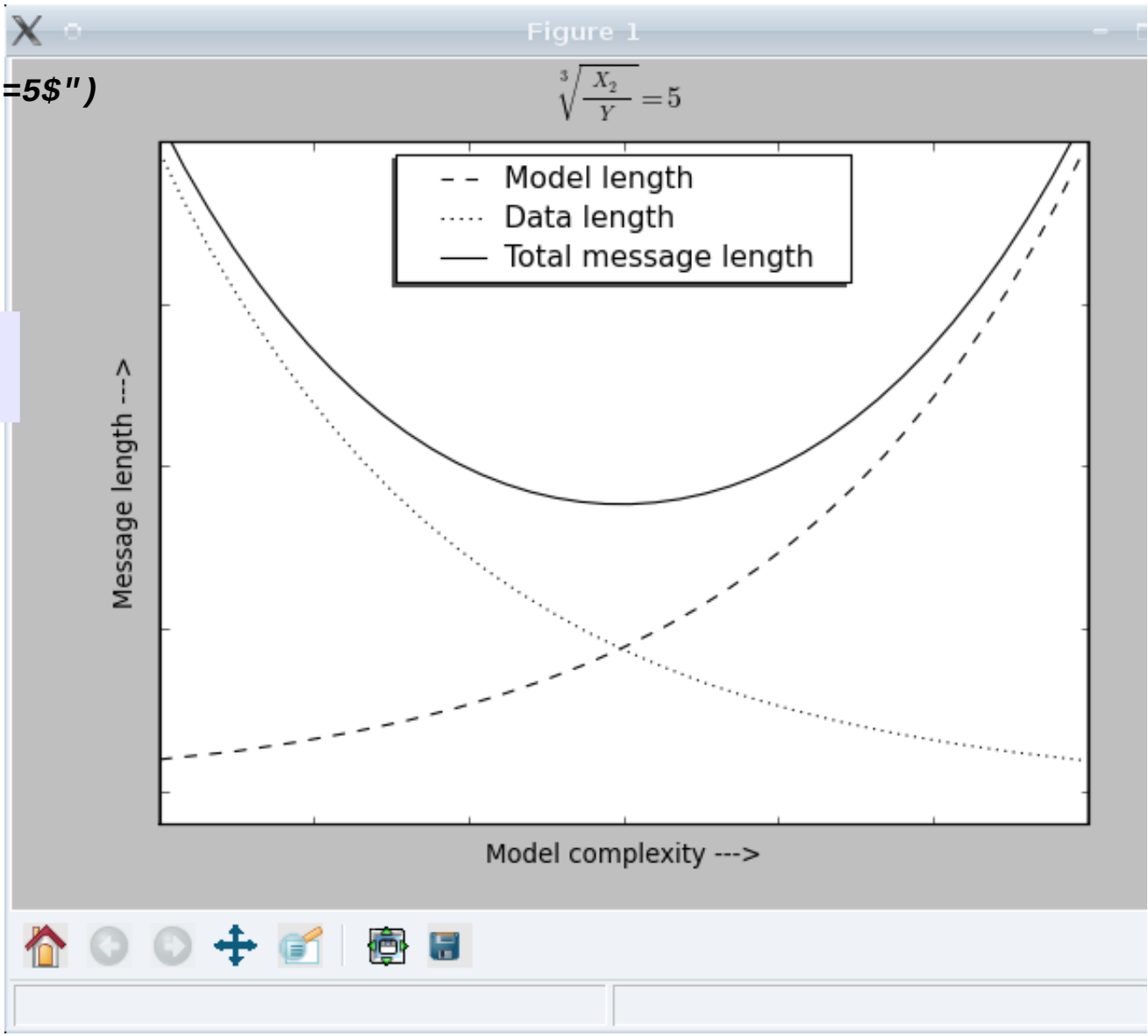
Tracer plusieurs courbes avec des styles différents, des titres & légendes



Graphiques 1D avec matplotlib
utilisation de TeX

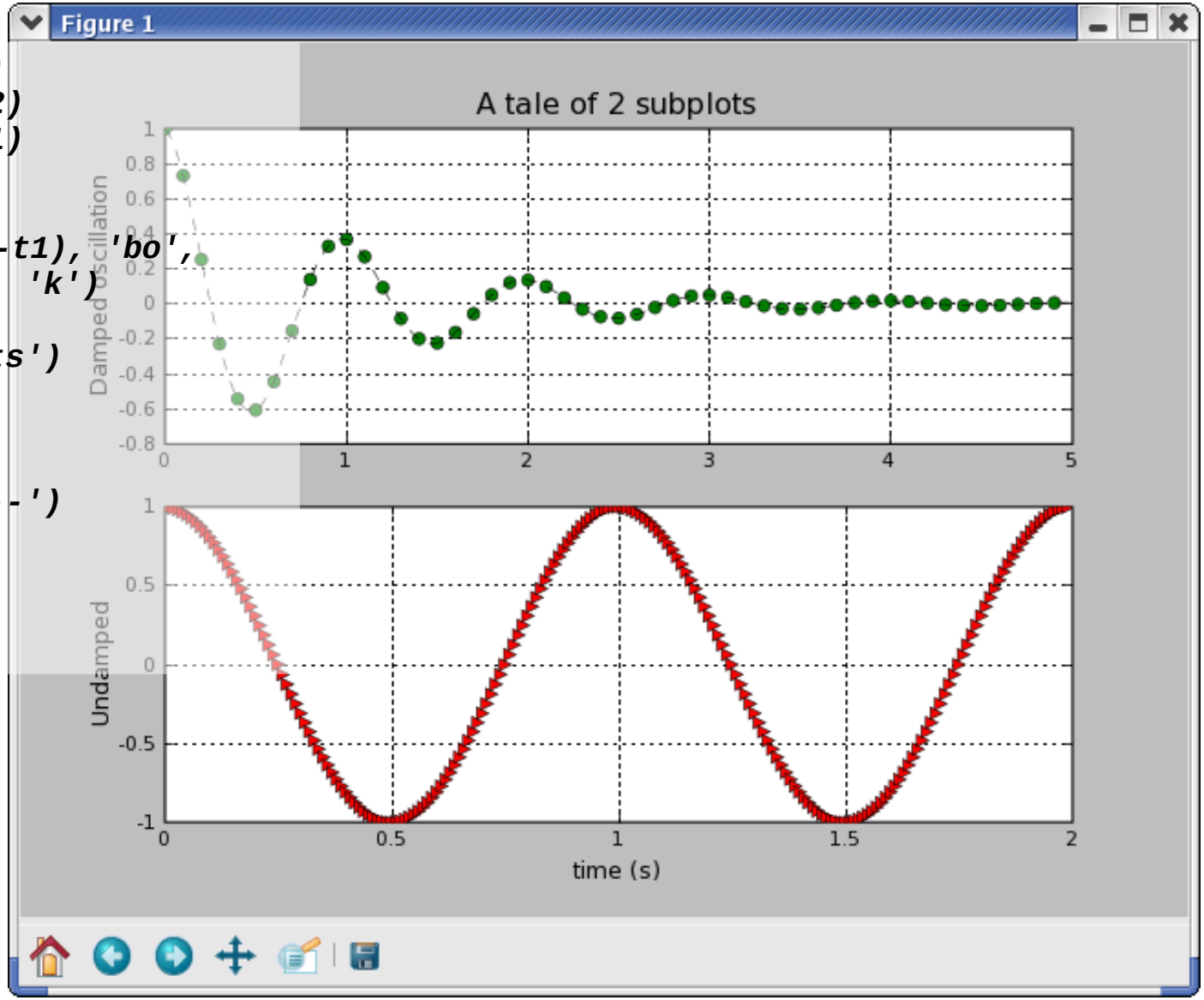
```
title(r"$\sqrt[3]{\frac{X_2}{Y}}=5$")
```

Toutes les légendes, titres, peuvent être écrites en TeX



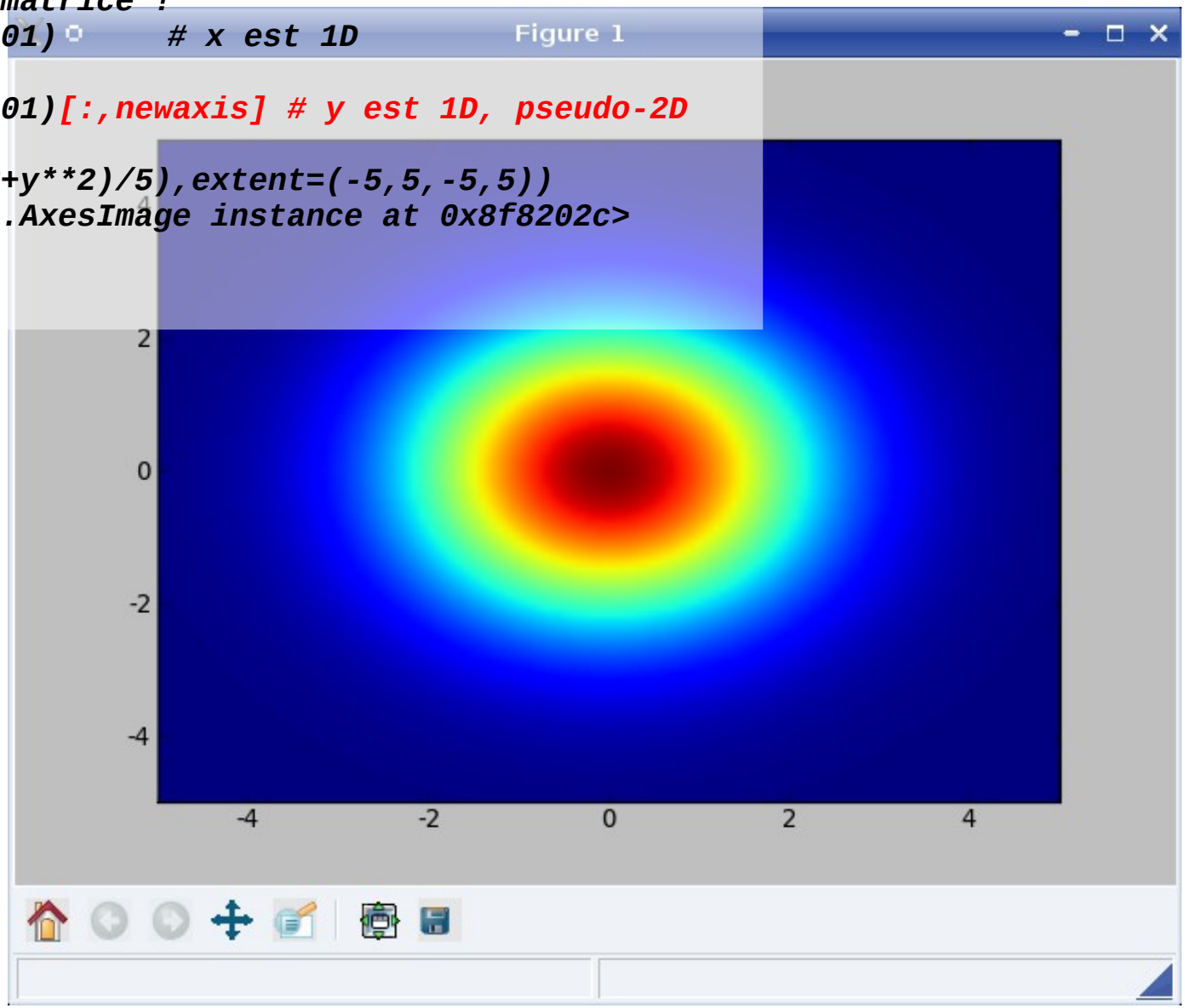
Graphiques 1D avec matplotlib
plusieurs graphiques dans une figure

```
t1 = arange(0.0, 5.0, 0.1)
t2 = arange(0.0, 5.0, 0.02)
t3 = arange(0.0, 2.0, 0.01)

subplot(211)
plot(t1, cos(2*pi*t1)*exp(-t1), 'bo',
t2, cos(2*pi*t2)*exp(-t2), 'k')
grid(True)
title('A tale of 2 subplots')
ylabel('Damped')

subplot(212)
plot(t3, cos(2*pi*t3), 'r--')
grid(True)
xlabel('time (s)')
ylabel('Undamped')
```



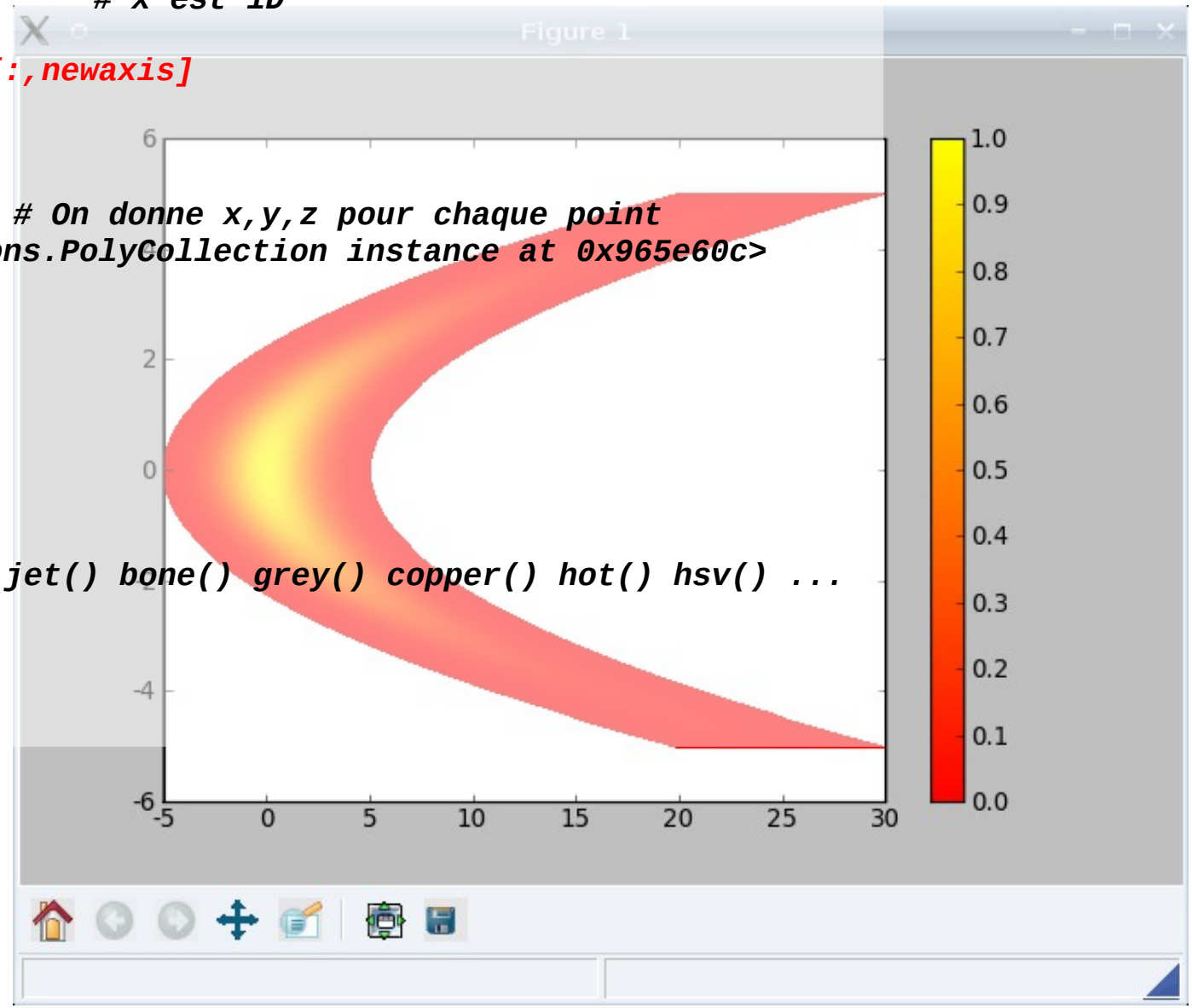
Graphiques 2D avec matplotlib
imshow

```
Un tableau n'est pas une matrice !
In [1]: x=linspace(-5,5,101)     # x est 1D

In [2]: y=linspace(-5,5,101)[:,newaxis] # y est 1D, pseudo-2D

In [6]: imshow(exp(-(x**2+y**2)/5),extent=(-5,5,-5,5))
Out[6]: <matplotlib.image.AxesImage instance at 0x8f8202c>
```



Graphiques 2D avec matplotlib imshow

```
n [1]: x=linspace(-5,5,101)        # x est 1D

In [2]: y=linspace(-5,5,101)[:,newaxis]

In [3]: z=exp(-(x**2+y**2)/5)

In [4]: pcolor(x+y**2,y,z)    # On donne x,y,z pour chaque point
Out[4]: <matplotlib.collections.PolyCollection instance at 0x965e60c>
```




```
Pour changer de couleur

In [15]: autumn()    # Aussi jet() bone() grey() copper() hot() hsv() ...

In [16]: colorbar()
```

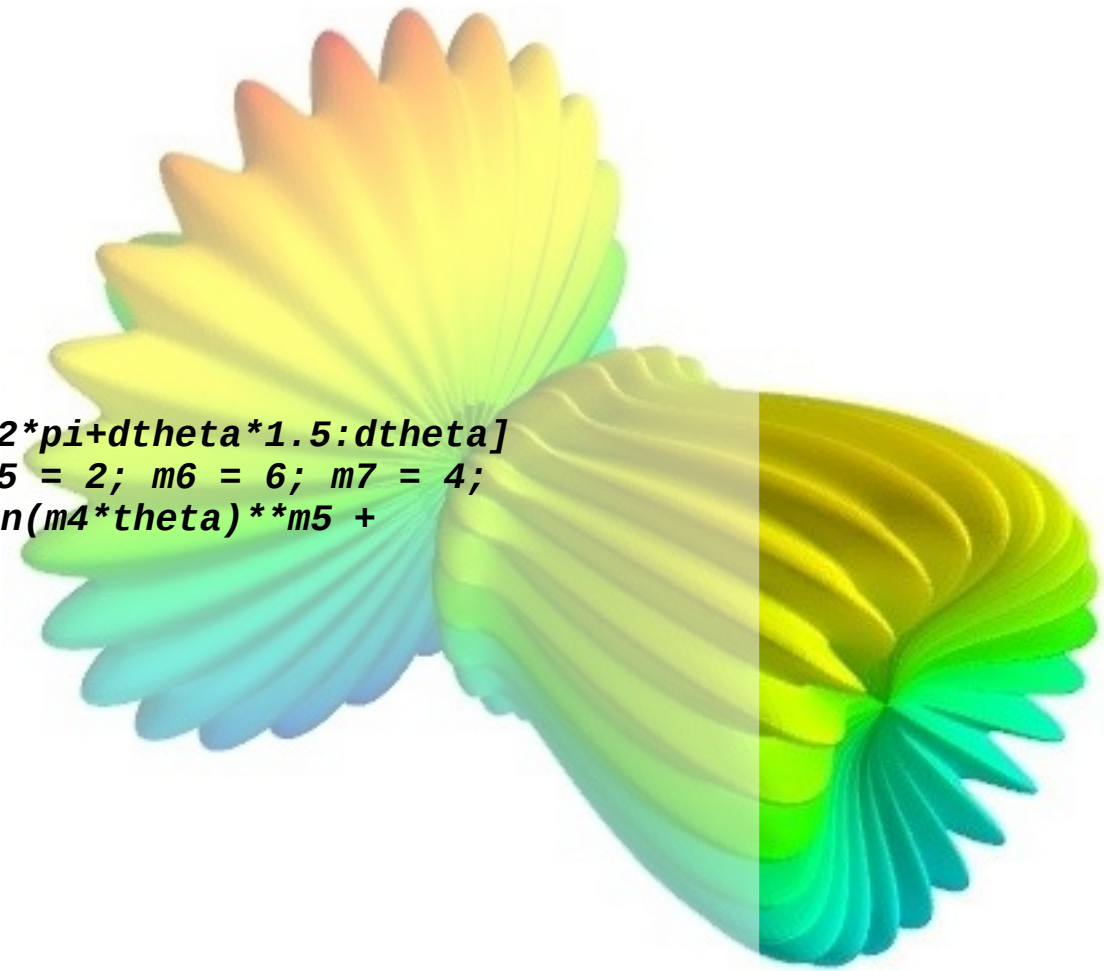
Graphiques 3D avec mayavi

Graphiques 3D avec mayavi.mlab

http://code.enthought.com/projects/mayavi/docs/development/html/mayavi/
" Simple scripting with mlab "

Lancer ipython avec " ipython -wthread "

```
from numpy import *
from enthought.mayavi import mlab

# Create the data.
dphi, dtheta = pi/250.0, pi/250.0
[phi,theta] = mgrid[0:pi+dphi*1.5:dphi,0:2*pi+dtheta*1.5:dtheta]
m0 = 4; m1 = 3; m2 = 2; m3 = 3; m4 = 6; m5 = 2; m6 = 6; m7 = 4;
r = sin(m0*phi)**m1 + cos(m2*phi)**m3 + sin(m4*theta)**m5 +
cos(m6*theta)**m7
x = r*sin(phi)*cos(theta)
y = r*cos(phi)
z = r*sin(phi)*sin(theta)

# View it.
s = mlab.mesh(x, y, z)

mlab.show()
```



Intégration avec scipy

```
In [20]:from scipy.integrate import quad
In [21]: def f(x): return x**2
   ....:

In [22]: quad(f, 0, 1)
Out[22]: (0.33333333333333331, 3.7007434154171879e-15)
```

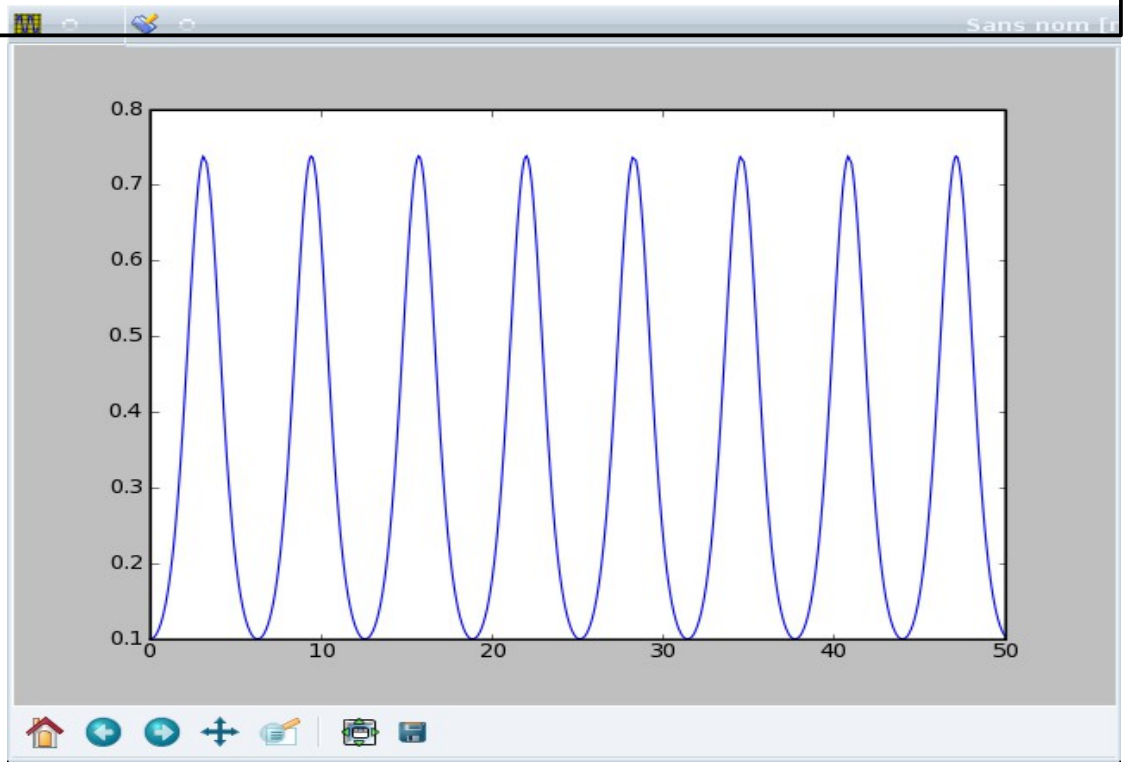
$$\int_0^1 x^2 dx = \frac{1}{3}$$

```
help(scipy.integrate)
    Integration routines
    ====================
     Methods for Integrating Functions given function object.

       quad          -- General purpose integration.
       dblquad       -- General purpose double integration.
       tplquad       -- General purpose triple integration.
       fixed_quad    -- Integrate func(x) using Gaussian quadrature of order n.
       quadrature    -- Integrate with given tolerance using Gaussian quadrature.
       romberg       -- Integrate func using Romberg integration.

     Methods for Integrating Functions given fixed samples.

       trapz         -- Use trapezoidal rule to compute integral from samples.
       cumtrapz      -- Use trapezoidal rule to cumulatively compute integral.
       simps         -- Use Simpson's rule to compute integral from samples.
       romb          -- Use Romberg Integration to compute integral from
                        (2**k + 1) evenly-spaced samples.

       See the special module's orthogonal polynomials (special) for Gaussian
          quadrature roots and weights for other weighting factors and regions.

     Interface to numerical integrators of ODE systems.

       odeint        -- General integration of ordinary differential equations.
       ode           -- Integrate ODE using vode routine.
```

Intégration avec scipy : équation différentielle du 1er degré

```
                    Résolution d'équation différentielle avec odeint
# Equation du premier degré, exemple:

from scipy.integrate import odeint
def f(y,t): return sin(t)*y        # Derivée de y(t)
t=linspace(0,50,501)               # Valeurs de t souhaitées
y=odeint(f,[0.1],t)       # Calcul en partant de y=0.1
plot(t,y)                          ## Affichage du résultat
```

$$\frac{\partial y}{\partial t} = y * \sin(t)$$



NB: l'intégration se fait avec un pas adaptatif (plus petit là où les dérivées sont grandes), indépendamment des abscisses demandées

Intégration avec scipy : équation différentielle du 2nd degré

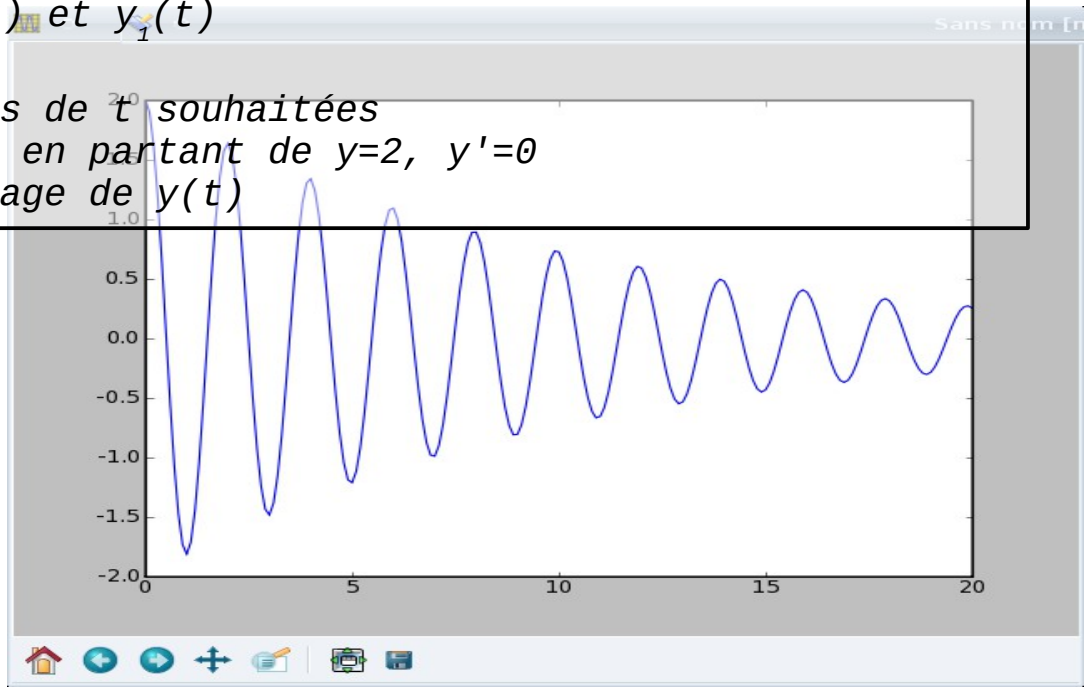
```
# Equation du second degré, exemple :
```

$$\frac{\partial^2 y}{\partial t^2} = -10 * y^2 - \frac{0.2 * \partial y}{\partial t}$$

```
# Changement de variable  - y devient [y0, y1] : pseudo-premier degré :
```

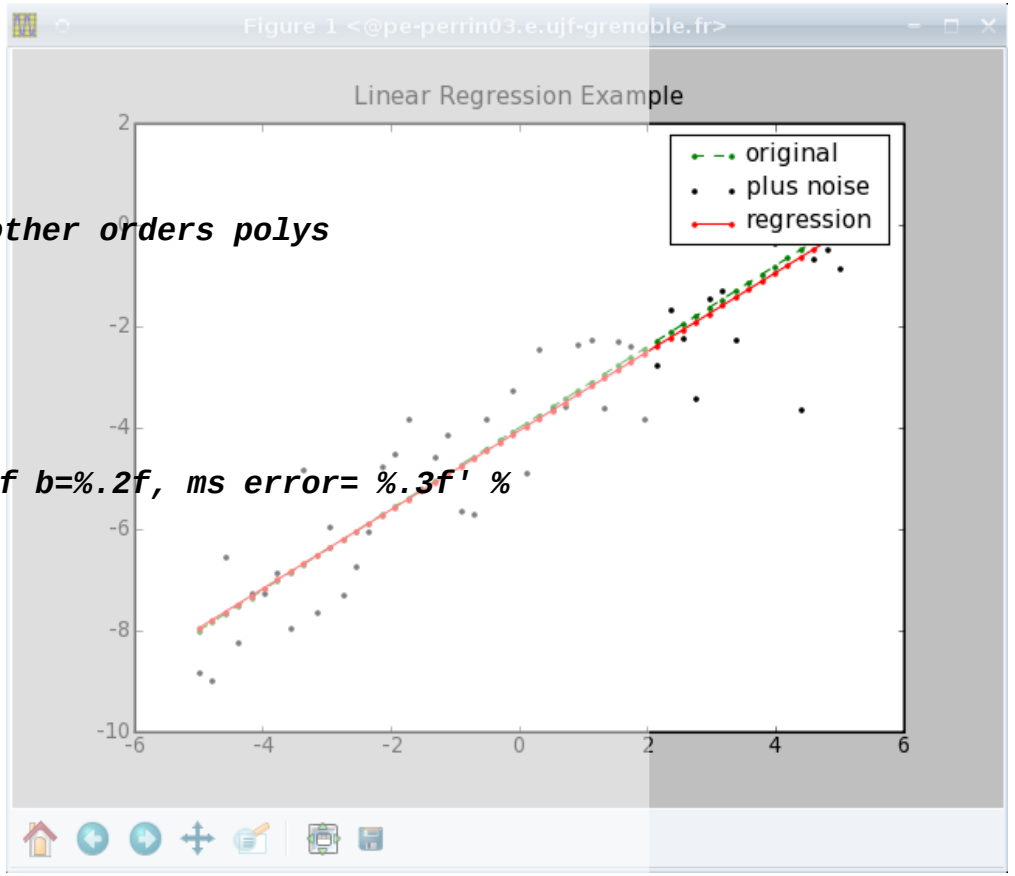
$$\left\{ \begin{array}{l} y_0 = y \\ y_1 = \frac{\partial y}{\partial t} \end{array} \right\} \qquad \left\{ \begin{array}{l} \frac{\partial y_0}{\partial t} = y_1 \\ \frac{\partial y}{\partial t} = -10 * y_0^2 - 0.2 * y_1 \end{array} \right\}$$

```
def f(y,t):            # Dérivée de y0(t) et y1(t)
    return [y[1],-10*y[0]-0.2*y[1]]
t=linspace(0,20,201)          # Valeurs de t souhaitées
y=integrate.odeint(f,[2,0],t) # Calcul en partant de y=2, y'=0
plot(t,y[:,0])                # Affichage de y(t)
```



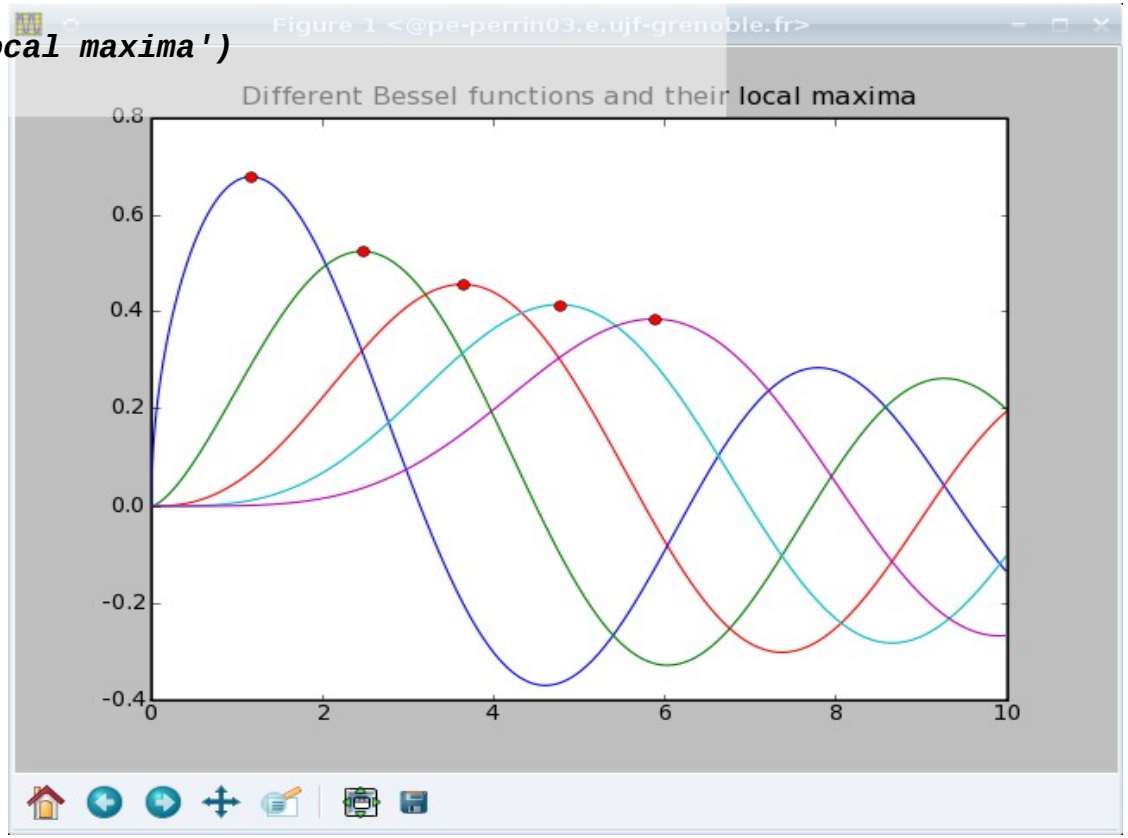
Affinement de données régression linéaire

```
from scipy import linspace, polyval, polyfit, sqrt, stats, randn

#Linear regression example
# This is a very simple example of using two scipy tools
# for linear regression, polyfit and stats.linregress

#Sample data creation
#number of points
n=50
t=linspace(-5,5,n)
#parameters
a=0.8; b=-4
x=polyval([a,b],t)
#add some noise
xn=x+randn(n)

#Linear regressison -polyfit - polyfit can be used other orders polys
(ar,br)=polyfit(t,xn,1)
xr=polyval([ar,br],t)
#compute the mean square error
err=sqrt(sum((xr-xn)**2)/n)

print('Linear regression using polyfit')
print('parameters: a=%.2f b=%.2f \nregression: a=%.2f b=%.2f, ms error= %.3f' %
(a,b,ar,br,err))

#matplotlib ploting
title('Linear Regression Example')
plot(t,x,'g.--')
plot(t,xn,'k.')
plot(t,xr,'r.-')
legend(['original','plus noise', 'regression'])
```



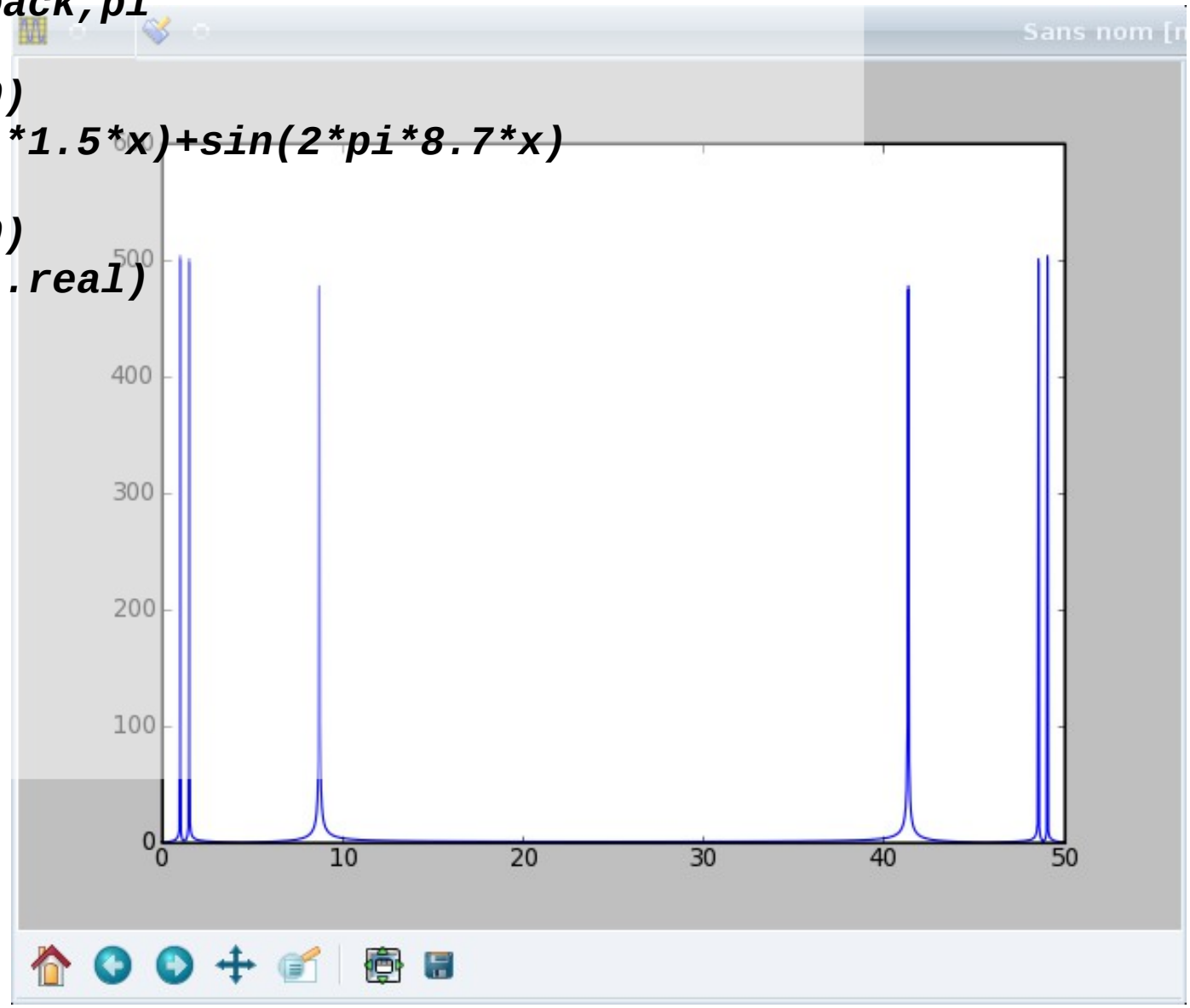
Affinement de données
Recherche de maximum

```
from scipy import *
from numpy import *

x = arange(0,10,0.01)

for k in arange(0.5,5.5):
     y = special.jv(k,x)                # scipy.special : fonctions spéciales
     plot(x,y)
     f = lambda x: -special.jv(k,x)
     x_max = optimize.fminbound(f,0,6)
     plot([x_max], [special.jv(k,x_max)],'ro')

title('Different Bessel functions and their local maxima')
```



Transformée de Fourier (rapide)

```
from scipy import fftpack,pi

x=linspace(-10,10,1000)
y=sin(2*pi*x)+sin(2*pi*1.5*x)+sin(2*pi*8.7*x)

y2=linspace(0,100,1000)
plot(y2,fftpack.fft(y).real)
```



Présentes :
FFT 1D, 2D, 3D,...

Liens

http://www.scipy.org

http://www.scipy.org/Documentation

http://www.scipy.org/Cookbook

http://www.scipy.org/Topical_Software

http://www.scipy.org/Numpy_Example_List_With_Doc

http://matplotlib.sourceforge.net

http://code.enthought.com/projects/mayavi/docs/development/html/mayavi/

https://svn.enthought.com/enthought/wiki/MayaVi

Partie III

- ***Programmation Orientée Objet (principes)***
- ***Lecture/écriture de fichiers***

Notion de Programmation Orientée Objet (POO)

- ***En POO, on peut créer de nouveaux types de données sous forme de "classe"***
- ***Un objet appartenant à une classe est une "instance" de cette classe***
- ***Une classe contient deux sortes d'attributs : des variables membres et des fonctions membres (appelées méthodes)***

Exemple:

```
class MaClasse:
    x=12

a=MaClasse()     # Création d'un objet de type MaClasse
print a.x        # Accès à la donnée x de l'objet a
a.x=25
b=MaClasse()     # deuxième objet de type MaClasse
print b.x        # Accès à la donnée x de l'objet b
12
```

Classes et données

- ***En python, on peut ajouter dynamiquement des données dans un objet:***

```
class MaClasse:
    x=12

a=MaClasse()     # Création d'un objet de type MaClasse
print a.x        # Accès à la donnée x de l'objet a
a.x=25
a.y=12           # Ajout d'une variable membre y
a.nom='toto'     # Ajout d'une variable membre 'nom'
```

***Note** : pour la clarté du code, il est néanmoins conseillé d'ajouter et de documenter toutes les variables membres au début de la classe*

***Note 2**: Attention, une variable ajoutée n'est disponible que dans l'instance où elle a été ajoutée !!*

Classes & documentation

- *Il est possible de documenter la classe*

```
class MaClasse:
    """ Ceci est une classe exemple """
     x=12

help(MaClasse)
    Help on class MaClasse in module __main__:

    class MaClasse
     |  Ceci est une classe exemple
     |
     |  Data and other attributes defined here:
     |
     |  c = 12
```

Classes & fonctions membres

- ***Les méthodes sont des fonctions déclarées dans le corps de la classe. Elles doivent avoir comme première variable le mot-clef 'self' qui représente l'instance de la classe pour laquelle on appelle la fonction***

```
class MaClasse:
    """ Ceci est une classe exemple """
   x=12
    def fonction1(self):
        print "Valeur de l'objet: ",self.x
    def valeurx(self):
        return self.x          # Renvoie une valeur
    def ajoute(self,y=1):      # Ajoute une valeur (par défaut : 1)
        self.x += y
a=Maclasse()
a.fonction1()  # réponse: "Valeur de l'objet: 12"
a.ajoute(20)   # ajoute 20 à a.x
a.ajoute() # ajoute 1 à a.x
```

Classes & méthodes spéciales

- *Il existe des méthodes spéciales avec des noms prédéfinis (utilisant deux '_' avant et après):*
- *__init__ : constructeur de la classe – cette fonction est appelée lorsqu'on crée une instance de la classe – cela permet de passer des paramètres lors de la construction d'un objet :*

```
class MaClasse:
    def __init__(self, x, x2=5):  # constructeur – une valeur par défaut
        self.x=x
        self.y=x2

a=Maclasse(1,2)     # Crée un objet avec a.x=1 et a.y=2
```

Classes & méthodes spéciales

- *Il existe des méthodes spéciales avec des noms prédéfinis (utilisant deux '_' avant et après):*
- *__str__ : méthode permettant d'interpréter l'objet sous forme d'une chaîne de caratère*
- *__add__ : méthode permettant l'addition de deux objets*
- *__eq__ : méthode permettant de tester l'égalité de deux objets*
- *etc...*

```
class MaClasse:
    x=12
    def __str__(self): return "MaClasse, valeur=%d"%(self.x)
    def __add__(self,b):
        a=Maclasse()
        a.x=self.x+b.x
        return a
    def __eq__(self,b): return self.x==b.x


a=MaClasse()

b=MaClasse()

print str(a)   # équivalent à a.__str__()

print a+b      # équivalent à a.__add__(b)   => Surcharge d'opérateur +

print a==b     # équivalent à a.__eq__(b)    => Surcharge d'opérateur ==
```

Pourquoi des classes ?

- ***Pour stocker des données avec des fonctions qui permettent d'interpréter ou de traiter ces données :***

```
a=Molecule()
a.Energie()
a.NombreAtome()
a.FormuleBrute()
len(a)            # Nombre d'atomes
a[0].x            # Coordonnée x du premier atome
a.SpectreRMN()    # Renvoie le spectre RMN
a.save("nom.mol",format='mol')   # Enregistrement dans un fichier mol
```

Pourquoi des classes ?

- ***Pour bien "ordonner" son code : une fonction est toujours associée à l'objet à laquelle elle se rapporte.***

- ***Si la manière dont sont stockées les données changent, les fonctions qui sont codées avec sont également modifiées – MAIS les fonctions doivent toujours prendre la même forme (même arguments, mêmes résultat) => les fonctions sont une couche d'abstraction qui cachent la manière dont sont stockées et traitées les données***

- ***Il n'est pas nécessaire de connaître le détail d'une fonction / d'une classe pour l'utiliser. Il suffit de connaître sa documentation (arguments, valeurs renvoyées)***

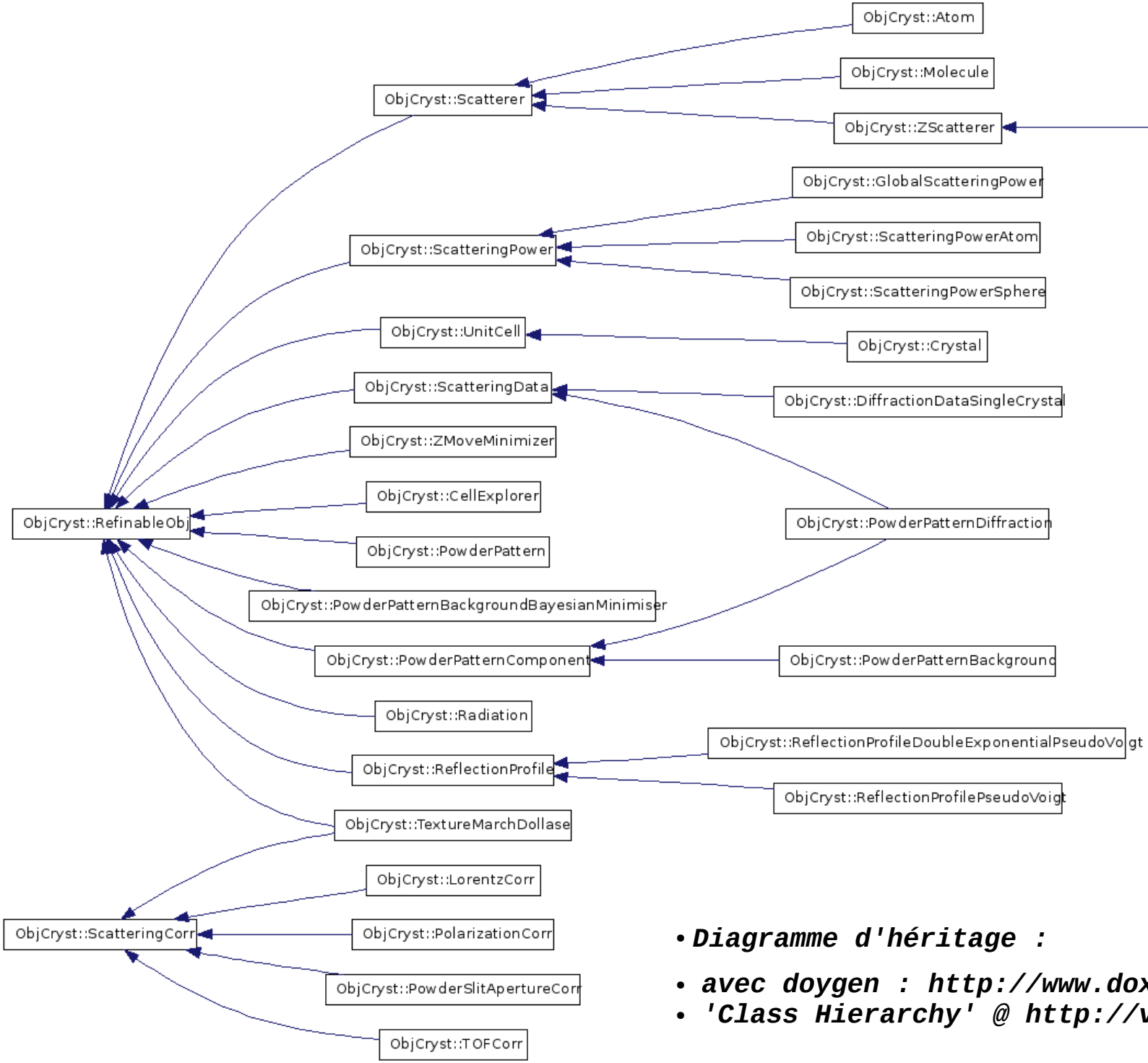
Classe & héritage

- Une classe peut hériter d'une autre : elle possède alors toutes les propriétés (données, fonctions) de la classe parente, et peut en ajouter ou en remplacer :

```
class Classe1:
    x=2
    def __init__(self, x=5):
        self.x=x

class ClasseEnfant(Classe1):
    y=2
    def __init__(self,x=5, nom='toto'):
        self.__init__(x)      # Appel au constructeur de Classe1
        self.nom=nom

a=ClasseEnfant(6,"tata")
Print a.x,a.y,a.nom
```

Note: l'héritage ne doit être utilisé que si les deux classes ont réellement des propriétés en commun. Par exemple une classe "Proteine" peut hériter de "Molecule", mais pas de "Atome" . Par contre Molecule va contenir des objets Atome.

Classe & héritage



- ***Diagramme d'héritage :***
- ***avec doygen : http://www.doxygen.org***
- ***'Class Hierarchy' @ http://vincefn.net/ObjCryst/***

Classe & polymorphisme

- Polymorphisme : lorsqu'une classe hérite d'une autre, elle peut réécrire une fonction de la classe parente:

```
class Rectangle:
    def __init__(self, a=5,b=2):
        self.x=a
        self.y=b
    def Surface(self): return self.x*self.y
    def Circonference(self): return 2*(self.x+self.y)

class Carre(Rectangle):
    def __init__(self, a=5):
        self.x=a
    def Surface(self): return self.x*self.x
    def Circonference(self): return 4*self.x
```

L'intérêt du polymorphisme est qu'on peut appeler une méthode sans savoir si l'objet est de type Carre ou Rectangle

Lecture/écriture de données

- **Fonctions de base :**

```
f=open("data.txt",'r')        # Ouverture d'un fichier pour lecture
ll=f.readlines()              # Lit toutes les lignes d'un fichier
for l in ll: print l      # Imprime toutes les lignes
f.close()                     # Fermeture du fichier

f=open("data.txt",'r')        # Ouverture d'un fichier
l=f.readline()                # Lit une ligne sous forme d'une chaîne de car.
vx,vy=[],[]
While len(l)>0:               # A la fin du fichier la chaîne est vide
    s=l.split()               # On sépare les champs (fichier avec 2 colonnes)
    vx.append(float(s[0]))
    vy.append(float(s[1]))
    l=f.readline()
f.close()


f=open("data.txt",'w')            # Ouverture d'un fichier pour écriture
f.write("Ligne 1 du fichier\n")
f.write("%f %f %f\n"%(1,2,3))     # Écriture de données formatées
f.close()
```

Lecture/écriture avec Pickle/cPickle

Les module "pickle" et 'cPickle" permettent de stocker dans un fichier n'importe quel type d'objet, et de les relire ensuite. Documentation à : http://docs.python.org/library/pickle.html

```
class Rectangle:
    def __init__(self, a=5,b=2):
        self.x=a
        self.y=b
    def Surface(self): return self.x*self.y
    def Circonference(self): return 2*(self.x+self.y)

a=Rectangle()                    # Création de deux rectangles
b=Rectangle()
import pickle
f=open("rect.pickle",'w')
pickle.dump((a,b),f)             # Sauvegarde d'un tuple de deux rectangles
f.close()

f=open("rect.pickle",'r')
c,d=pickle.load(f)               # relecture
```

Note: le module cPickle est similaire à pickle, mais écrit en C et plus efficiace !
Note 2: il vaut mieux utiliser `cPickle.dump((a,b),f, protocol=-1)` qui permet d'être encore plus efficace (compression)...
Note 3: lec fichiers enregistrés ne sont pas portables ! Il ne peuvent a priori être relus que sur le même ordinateur (version de python,système, architecture)...

Lecture/écriture avec scipy/numpy

Dans scipy il existe un module 'io' (input/output) :

Voir : http://www.scipy.org/Cookbook/InputOutput

```
Data input and output
=====================
  Classes
        npfile -- a class for reading and writing numpy arrays from / to binary files
        Cache
        DataSource
        Repository
  Functions
        read_array -- reading ascii streams into NumPy arrays
        write_array -- write an array to an ascii stream
        loadmat -- read a MATLAB style mat file (version 4 and 5)
        savemat -- write a MATLAB (version <= 4) style mat file
        fread -- low-level reading
        fwrite -- low-level writing
        bswap -- in-place byte-swapping
        packbits -- Pack a binary array of 1's and 0's into an array of bytes
        unpackbits -- Unpack an array packed by packbits.
        save --- simple storing of Python dictionary into module that can then be imported
                 and the data accessed as attributes of the module.
        mminfo   -- query matrix info from Matrix Market formatted file
        mmread   -- read matrix from Matrix Market formatted file
         mmwrite  -- write matrix to Matrix Market formatted file
```

Partie IV

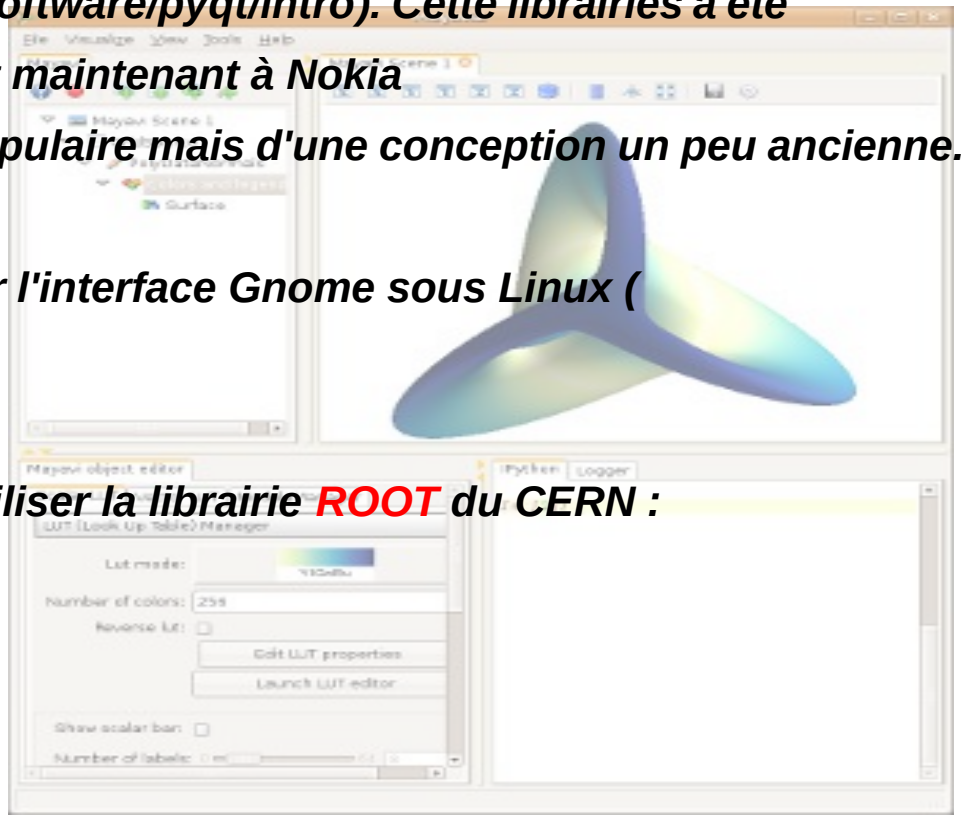
- ***Interfaces graphiques***
- ***Acquisition de données***
- ***Calcul parallèle***

Interfaces graphiques (GUI) en Python

Il existe de nombreuses interfaces graphiques disponibles en Python. Il s'agit en général de libraires écrites en C ou C++, et interfacées avec Python.

Les plus importantes (toutes multi-plateformes) sont :

- ***WxPython : basé sur la librairie wxWidgets (http://www.wxpython.org)***
- ***PyQt : basé sur la librairie Qt – la librairie sur laquelle est construit l'environnement KDE sous Linux (http://www.riverbankcomputing.co.uk/software/pyqt/intro). Cette librairies a été développée par TrollTech, société appartenant maintenant à Nokia***
- ***Python-tkinter : basé sur Tcl/Tk – a été très populaire mais d'une conception un peu ancienne. (http://wiki.python.org/moin/TkInter)***
- ***PyGTK : basé sur la librairie GTK, utilisée pour l'interface Gnome sous Linux (http://www.pygtk.org/)***

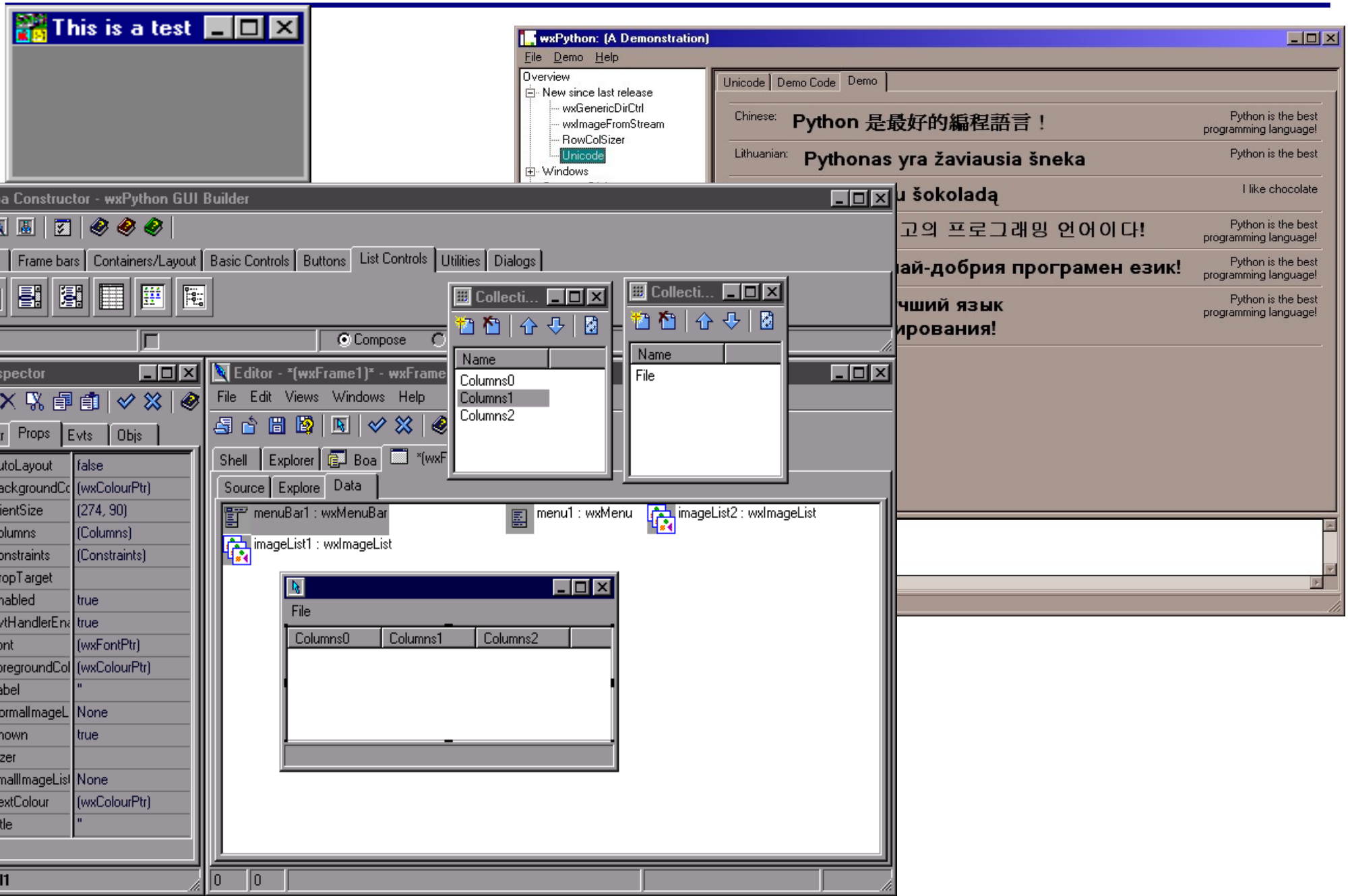


Signalons aussi l'interface python permettant d'utiliser la librairie ROOT du CERN :

http://wlav.web.cern.ch/wlav/pyroot/

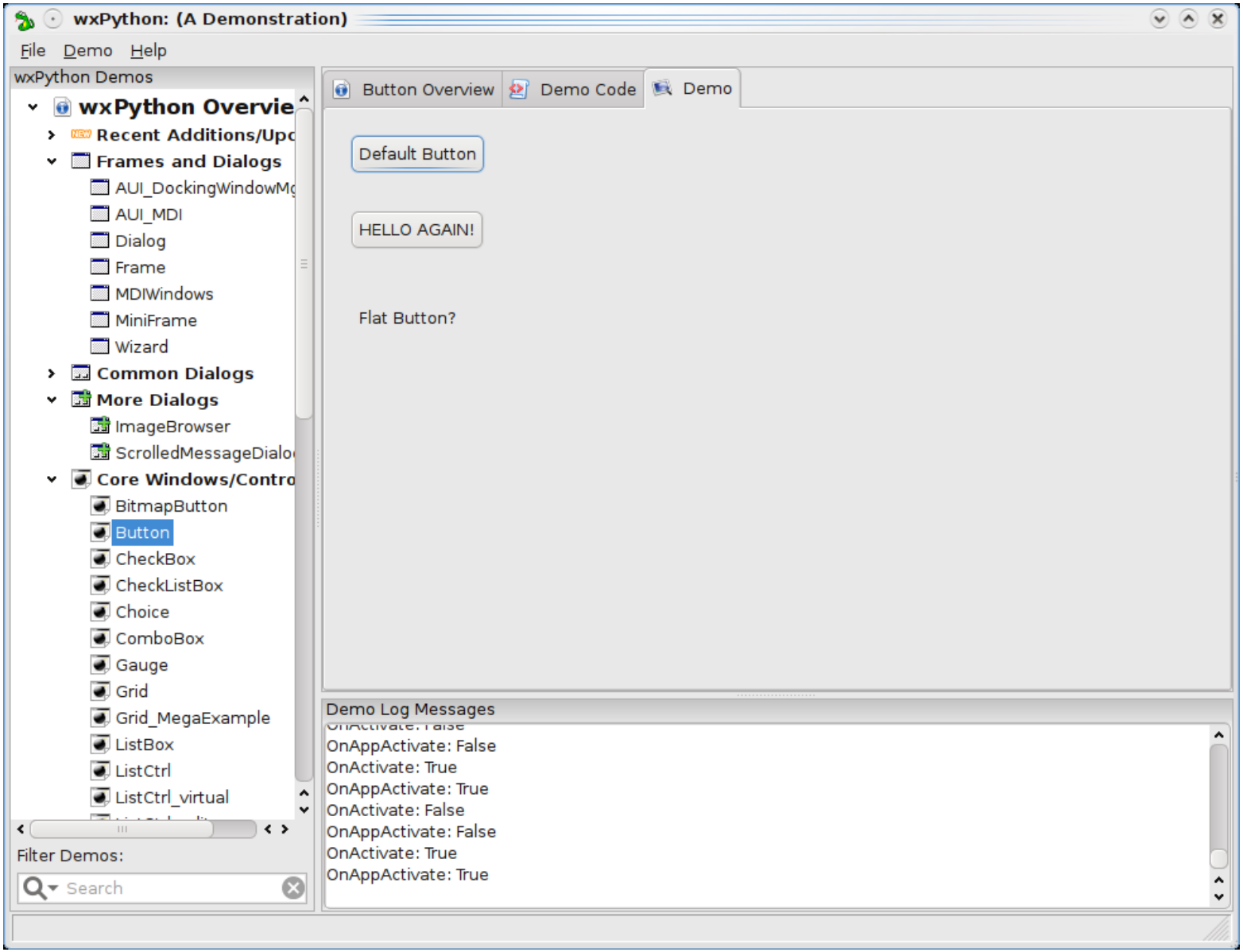
etc...

Exemples (avec wxPython)



Exemples (avec wxPython)

python /usr/share/doc/wx2.8-examples/examples/wxPython/demo.py

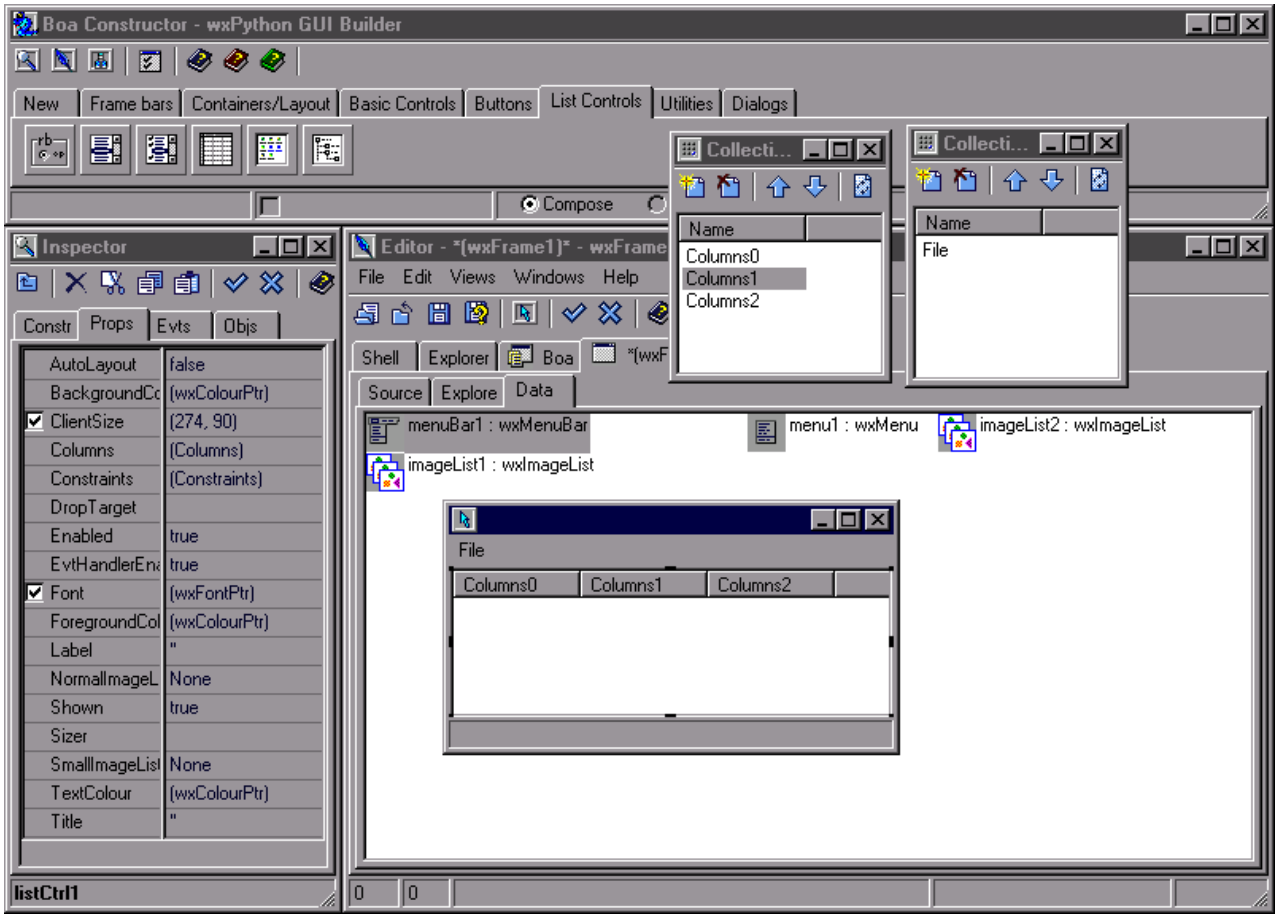


Différents types de fenêtres & dialogues

Les notions de base les plus importantes sont :

- les différents types de fenêtres et de dialogues
  - http://docs.wxwidgets.org/trunk/hierarchy.html
- L'agencement des fenêtres
- La notion d'**évènements**

=> voir exemples de code



Acquisition de données

Il existe des librairies spécialisées pour l'acquisition de données

Avec l'USB, en utilisant des cartes d'acquisition :

- **PyUL (python-Universal Library) avec un boîtier de "measurement computing"**

http://www.scipy.org/Cookbook/Data_Acquisition_with_PyUL

- **Avec une librairie et un boîtier de NI:**

http://www.scipy.org/Cookbook/Data_Acquisition_with_NIDAQmx

Avec les ports série & parallèle :

- **http://pyserial.wiki.sourceforge.net/**

Pour tous les ports(usb, série, gpib) :

- **http://pyvisa.sourceforge.net/**

Calculs parallèles en python

Il est possible d'effectuer des calculs parallèles en python à l'aide de :

threading
(module standard python)
http://docs.python.org/library/threading.html

Multiprocessing
(module standard python)
http://docs.python.org/library/multiprocessing.html

MPI
pypar *: http://code.google.com/p/pypar/*
mpi4py *: http://mpi4py.scipy.org/*

Calcul sur GPU (CUDA)
PyCUDA :
http://documen.tician.de/pycuda/

PyPI – Python Package index

http://pypi.python.org

> 28000 paquets

Installation : “`pip install` *`packagename`*”

Topic

Adaptive Technologies (10) Artistic Software (46) Communications (473) Database (623) Desktop Environment (100) Documentation (146) Education (116)
Games/Entertainment (148) Home Automation (17) Internet (3546) Multimedia (614) Office/Business (567) Other/Nonlisted Topic (46) Printing (33) Religion (10)
Scientific/Engineering (1321) Security (264) Sociology (9) Software Development (8200) System (1099) Terminals (128) Text Editors (64) Text Processing (814)
Utilities (1692)

Création d'un Module Python

1) écrire un fichier avec les déclarations de variables, fonctions et classes

2) sauvergarder comme “ModuleExemple`.py`”

3) Importation avec : “`Import ModuleExemple`”

L'importation fonctionne tant que le fichier se trouve dans le chemin recherché par python, que l'on peut consulter via :

```
import sys
print sys.path
```

Pour qu'un module soit disponible sur un ordinateur, il faut le mettre dans le répertoire 'dist-packages', soit pour python 2.7 sous Linux : '/usr/lib/python2.7/dist-packages''

cProfile

Comment déterminer combien de temps prend chaque partie d'un programme python ?
=> http://docs.python.org/2/library/profile.html

Pour 'profiler' une fonction 'maFonction()', simplement utiliser :

```
import cProfile
cProfile.run('maFonction()')
```

Il est aussi possible d'obtenir une représentation graphique des temps d'éxécution :

http://code.google.com/p/jrfonseca/wiki/Gprof2Dot

Partie V

- ***Modules os, sys***
- ***Exceptions***
- ***! Notion & danger de la copie par référence !***
- ***Python 3***

Le module 'sys'

=> contient les fonctions & paramètres spécifiques au système :

```
import sys
sys.path        # Chemin de recherche des modules python
sys.argv        # Arguments de la ligne de commande python
sys.modules # Listes des modules importés
sys.version   # Version de python
sys.version_info #Version de python, plus facile à interpréter
```

Le module 'os'

=> fonctions & paramètres spécifiques au système d'exploitation:

```
import os
os.getcwd()            # 'Current Working Directory'
os.chdir(dirname)      # Change le répertoire à dirname
os.environ             # Variables d'environnement (dictionnaire)
os.listdir(dirname)    # Liste des fichiers&répertoires dans dirname
os.mkdir(dirname)      # Création du répertoire dirname
os.system('ls -la')    # Exécute la commande 'ls -la' dans l'OS
```

Exceptions

=> En cas de problème, python renvoie des erreurs :

```
>>> print 10/0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

Chaque erreur correspond à une exception spécifique, qu'il est possible de détecter à l'aide des instructions try/except:

```
try:
    print 10/0
except:
    print "Oups!"
```

Plus d'informations : http://docs.python.org/2/tutorial/errors.html

Exceptions

Il est possible de ne détecter qu'un type d'erreur:

```
try:
    print 10/0
except ZeroDivisionError:
    print "Oups!"
```

On peut enchaîner les tests d'exceptions :

```
try:
   print 10+"toto"
except ZeroDivisionError:
   print "Oups ! Division par zéro"
except:
   print "Exception !!"
```

Exceptions

Il est possible de récupérer l'objet correspondant à l'exception :

```
try
    print 10/0
except ZeroDivisionError as err:
    print "Oups ! Division par zéro"
    print err
```

Les exceptions forment une hiérarchie de classes :

```
>>> help(ZeroDivisionError)
Help on class ZeroDivisionError in module exceptions:

class ZeroDivisionError(ArithmeticError)
 |  Second argument to a division or modulo operation was zero.
 |
 |  Method resolution order:
 |      ZeroDivisionError
 |      ArithmeticError
 |      StandardError
 |      Exception
 |      BaseException
 |      __builtin__.object
```

Copie par référence

Par défaut dans python, tout objet complexe (définit dans une classe) est ***copié par référence****. Par exemple :*

```
from numpy import zeros
a=zeros(10)
b=a
```

Après ces instructions, b est une copie par référence de a, et non une copie indépendante :

```
print a,b
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.] [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
b[1]=10
print a,b
[  0.  10.  0.  0.  0.  0.  0.  0.  0.  0.] [  0.  10.  0.  0.  0.  0.  0.  0.  0.  0.]
```

- *Une copie par référence permet d'éviter d'utiliser trop de mémoire.*
- *Toute modification d'une copie par référence modifie aussi l'original !*

Copie par référence

```
from numpy import zeros
class MaClasse :
    var=zeros(10)

a=MaClasse()
print a.var
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
b=MaClasse()
print b.var
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
b.var[1]=10
print a.var,b.var
[ 0.  10.  0.  0.  0.  0.  0.  0.  0.  0.] [ 0.  10.  0.  0.  0.  0.  0.  0.  0.  0.]
```

NB : si 'var' avait été créée dans un constructeur (__init__), ce ne serait pas la même variable pour a et b, et les données seraient distinctes !

Copie par référence, copy & deepcopy

=> *faire attention lorsqu'une variable est copiée et ensuite modifiée !*

Afin de faire une copie indépendante, il est possible :

Pour un tableau

```
from numpy import zeros
a=zeros(10)
b=a*1          # Pas très élégant, mais b n'est pas une référence de a
b=a.copy()     # Plus élégant et rapide !
```

Pour toute classe : utiliser copy.deepcopy()

```
b=copy.deepcopy(a)    # Copie intégrale de a, sans référence
```

Python 3

- *Python 3 est sorti en 12/2008*
- *version actuelle : 3.4.2*
-
- *numpy+python3 OK depuis 1.5 (8/2010)*
- *scipy depuis 0.9 (2/2011)*
- *matplotlib depuis 1.2 (11/2012)*

Python 3 : principaux changements

***print** est une fonction*

- *Python 2.x :* *print “ toto ”*
- *Python 3:* *print(“ toto ”)*

*Plus de **xrange** → **range**(..) ne renvoie plus une liste :*

- *Python 3 :*

  for i in xrange(1000000000) : print(i)
- *type(range(10)) → builtins.range*

*Plus de “ **long** ” → un seul type d'entier : **int**(5)*

*Plus de **cPickle** → **pickle***

Python 3 : principaux changements

exec *est une fonction*

- *Python 2.x :* *exec “ toto ”*
- *Python 3:* *exec(“ toto ”)*

Division en virgule flottante *par défaut*

- *1/2 → 0.5*
- *1//2 → 0*
- *Utiliser “ from __future__ import division ” en python 2.7*

Nombres en base 8 : “ 056 ” → “ 0o56 ”
(aussi en base 2 “ 0b11 ”, 16 “ 0x11 ”)

Python 3 : principaux changements

exec *est une fonction*

- *Python 2.x :* *exec “ toto ”*
- *Python 3:* *exec(“ toto ”)*

execfile *disparaît → exec()*

- *Python 2.x :* *execfile(“toto.py”)*
- *Python 3:*
  - ***exec****(open(“toto”,'r').read())*
- *Mieux :*
  - *code =* ***compile****(f.read(), "somefile.py", 'exec')*
  - *exec(code)*

Python 3 : principaux changements

dict.keys() *renvoie une 'vue' (builtins.dict_keys) et plus une liste*
Idem pour dict.values(), dict.items()
• Disparition de iteritems() iterkeys() itervalues()

•De manière générale, beaucoup plus d'objets de type 'view' ou d'itérateurs.

*Beaucoup des changement dans les fonctions/modules standards web (**http**, **urllib**)*

Python 3 : principaux changements

Support plus natif pour les chaînes de caractère unicode.

Les classes dérivent toutes d'un type de base 'object'

Python 3 : resources

https://docs.python.org/release/3.0.1/whatsnew/3.0.html

Script “ 2to3 ” qui converti un code Python 2.x en 3.x

http://python3porting.com

http://www.diveintopython3.net

http://docs.pythonsprints.com/python3_porting/py-porting.html