Scientific programming in Python

2011/2012

Vincent.Favre-Nicolin@cea.fr

Aims of this course :

- (learn a new language)
- know the main scientific modules
- and apply them for your own project

5 Lectures/tutorials

1.5 Introduction to Python

1.5 Introduction to the scientific modules

2 slots devoted to your personal project

Which lnguages do you know ?

Fortran (77, 95...)

C

C++

basic

(pascal)

High and low-level languages

- **Assembly**

```
        .global _start              ; from wikipedia
BONJ:   .ascii  "Bonjour\n"
_start: mov     $4      , %eax
        mov     $1      , %ebx
        mov     $BONJ   , %ecx
        mov     $8      , %edx
        int     $0x80

        mov     $1      , %eax
        mov     $0      , %ebx
        int     $0x80
```

- **Low-level language (explicit memory handling, pointers,..) C C++ fortran etc...**

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

- **High-level language (matlab, python+modules,...)**
  - **No explicit memory handling**
  - **Auto-declaration of objects**

```
print “Hello”
```

  - **Command-line interpreter**

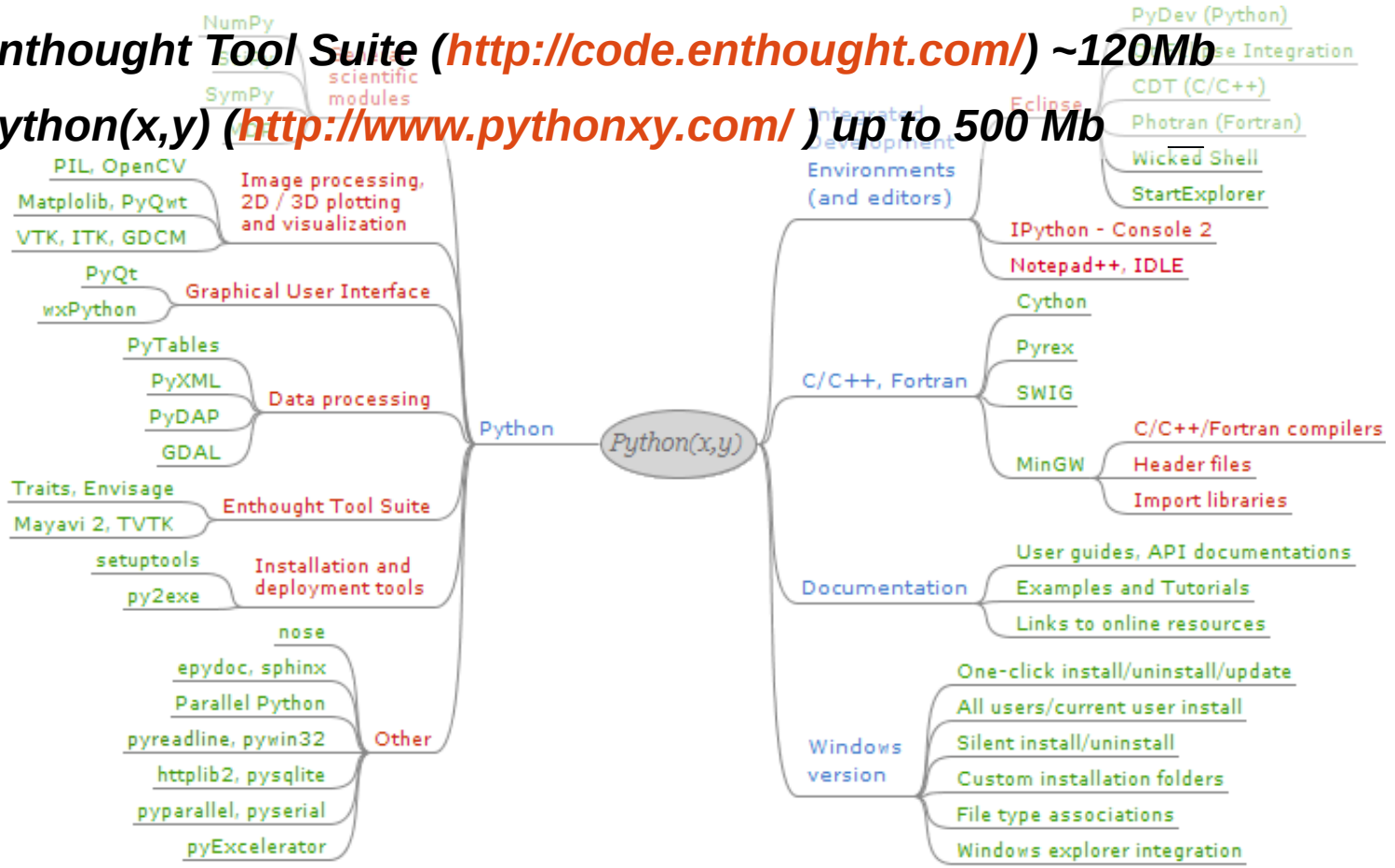
Languages : compiled / interpreted / just-in-time compiled

- ***Compiled :***
  - ***The programmed must be compiled prior to being run***
- ***Interpreted***
  - The program is read line-by-line
- ***Just-in-time compiled***
  - ***The program is compiled just before being run***

Install Python on your computer

- *Linux :*
  - *Python is generally pre-installed, with many libraries available*

    (Debian " lenny " includes 1520 packages with the python keyword)
- *Windows : free installers (python + scientific libraries)*
  - *Enthought Tool Suite (http://code.enthought.com/) ~120Mb*
  - *Python(x,y) (http://www.pythonxy.com/) up to 500 Mb*



Start Python :
With the command line

- ***For Linux, open a console and type " 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.
>>>
```

- ***Then you can type instructions :***

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

- ***You can (should !) also use" ipython " => recommanded !***

```
[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]:
```

To exit the python " shell " :
ctrl-d

Start Python :
Run a program file

- ***Create a text file (using your favorite editor) named " prog.py ", with :***

```
print 2+3
print " this is Monday "
```

- ***Save this file in a directory***

- ***Then run the program:***

```
-bash-3.2$ cd /where/the/file/was/saved
-bash-3.2$ python prog.py
5
This is Monday
-bash-3.2$
```

- ***Same with ipython***

```
-bash-3.2$ ipython prog.py
5
This is Monday
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

• ***Create variables without a prior declaration !***

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

• ***The “ type ” of each variable is automatically fixed***

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

>>> type(1.2)
<type 'float'>  # real, floating-point number
```

=> strong, dynamic typing

• ***The type of a variable can change upon a new assignment:***

```
>>> a= ”I love Physics !”
>>> print a
I love Physics !
>>> type(a)
<type 'str'>    # string
```

Basic types : numbers

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

Mathematical operations

• ***Addition***	1+2.0,	4-3j	
• ***Subtraction***	3-4,	10j-7	
• ***Multiplication***	5*4,	10.2*3.4e-9	
• ***Division***	1/2.0	89/2	! integer or floating point...
• ***Power***	2**8,	5.2**3.4,	3.2**(1+3.2j)

Basic types : String of characters

- ‘’abc’’ ou ‘abc’
- ‘\n’ # newline
- ‘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’ # adds ',' between elements
- ’ a b c ’.strip() -> ’a b c’ # removes leading and trailing spaces
- ’text’.find(’ex’) -> 1 # search
- ’Abc’.upper() -> ’ABC’
- ’Abc’.lower() -> ’abc’
- **Conversion to numbers:** int(’2’), float(’2.1’)
- **Conversion to text** : str(3), str([1, 2, 3])

Basic types : lists

• **Creation**
>>>a=[1,2,3, ’blabla’,[9,8]]
• **Concatenation (not addition… see array)**
>>>[1,2,3]+[4,5]
[1,2,3,4,5]
•**Add an element**
>>>a.append('test')
[1,2,3, ’blabla’,[9,8],’test’]
• **Length**
>>>len(a)
6
range([start,] stop[, step]) -> integer list
>>>range(5)
[0,1,2,3,4]
• **Simple indexing**
>>>a[0]
1•
• **Multiple indexing (nest lists)**
>>> a[4][1]
8

• **Element definition**
>>> a[1]=1
>>> a
[1, 1, 3, 'blabla', [9, 8], 'test']
• **Negative indices (from the end)**
>>> a[-1]
'test'
>>> a[-2]
[9, 8]
• **Parts of a list (liste[lower:upper])**S
>>>a[1:3]
[1,3]
>>> a[:3]
[1, 1, 3]

! Indices begin at 0 !

Inline help :
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
 |
                [...]
 |  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:

                [...]
```

Inline help is available for all functions, types, modules.

With ipython, use the variable name followed by '?' gives access to the help

Booleans & conditions

- **Boolean** : “ True ” and “ False ”
-
- ANy null number <-> False, not null <-> True

```
>>> if True: print “Yo”
...
toto

>>> if 0:
...     print “ Yo ”  # indentation !
... else:
...     print “ Ya ”
...     print “ Shazam ! ”
...
 Ya
 Shazam !
```

***Indentation* (spaces at the line beginning) replaces brackets {} in python.**

To get out of a loop, you just need to go back to the prior indentation level

For loop

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

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

```
>>> for i in range(4) :    # NB: better use xrange
...     print i
...
0
1
2
3
```

While loops

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

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

Python programming: 1 – line by line

```
>>> print “ this is a python program ”
this is a python program
>>> import math
>>> print math.sin(3.14159)
2.65358979335e-06
>>> for letter in ['a','b','c']: print letter
a
b
c
```

Helpful for quick tests
... or to use python as a **super-calculator**

Python programming: 2 – using functions

```
>>> def WriteSentence(s):
...     print s
...
>>> WriteSentence(“ Hello, world ”)
Hello, world


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

Useful when a series of instructions needs to be repeated for different values

Python programming: 3 – object-oriented

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

>>> p2=polynomial()
>>> p2.coeffs=[1,0,0,1]
>>> print p2.value(0),p2.value(1),p2.value(2)

1 2 9
```

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

Python, interactive: ipython

- *Type “ ipython ” in a console (or use a GUI launcher):*
  - *Call back instructions from history ↑ ↓ (incl. prior sessions)*
  - *Object help using “objname?”*
  - *Cod for an object or function using “name?? ”*
  - *Completion: begin typing + tabulation*
  - *Syntax coloring for code (en cas d'erreur)*
  - *Smart, parallel handling of graphical libraries...*

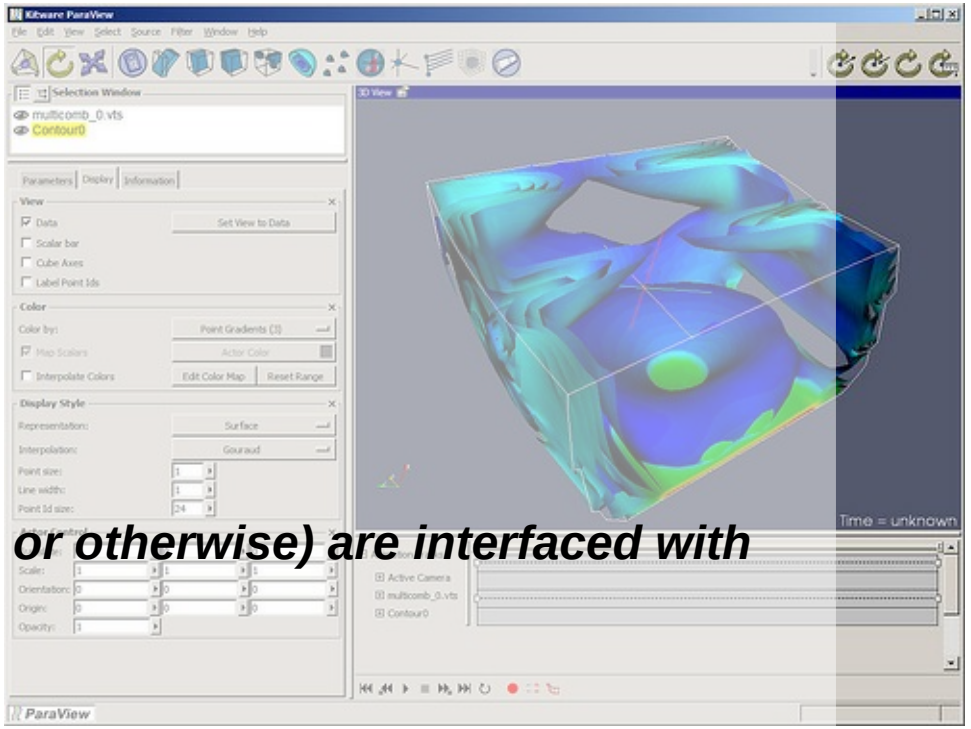
Modules

- ***Python is a modular language***
- ***All functions and complex objects are available from various modules :***

```
>>> import math              # " import " + module name
>>> print math.sin(1.2)      # ModuleName.FunctionName(parameters)
0.9320390859672629
>>> help(math)               # help on the module

>>> from math import *       # " from " + module name + " import *"
>>> print sin(1.2)
0.9320390859672629
```

Importance of modules

- Python is popular in many domains :
  - ***Scientific***
  - ***Web (google, some wikis, zope,...)***
  - ***Databases***
  - ***Many programs written in python***



- ***... because most existing libraries (scientific or otherwise) are interfaced with the Python language :***
  - ***=> no need to reinvent the wheel !***
  - ***Existing c/c++/fortran code is re-usable from Python***
  -
  -
  - ***Next slot : modules* scipy, numpy *&* matplotlib**

Useful links

- *http://www.python.org* — *Official python site*
- *http://wiki.python.org/moin/NumericAndScientific* — *Scientific libraries*
- *http://www.scipy.org* — *Scientific python*
- *http://scipy.org/Cookbook* — *Scipy cookbook*
- *http://www.scipy.org/Topical_Software* — *Programs related to scipy*
- *http://matplotlib.sourceforge.net/* — *2D graphics*
- *http://www.greenteapress.com/thinkpython/thinkpython.html* — *Free book*
- *http://www.limsi.fr/Individu/pointal/python/pqrc/* — *Python reference card*
-
- *Google :*
  - *“ python physics ” -> 2.8 M links...*
  - *“ python molecule quantum ” -> 42000 results*
  - *...*

Partie II : Scientific calculations

Using numpy, scipy, matplotlib, mayavi :

- ***1D, 2D, 3D graphics...***
- ***Refinements***
- ***Solving equations***
- ***etc...***

Mathematical calculations

Python includes the modules: 'math' et 'cmath' (math for complex numbers) :

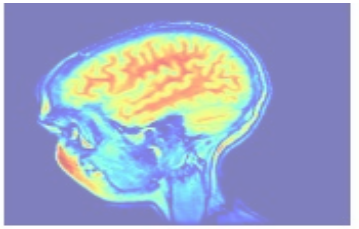
```
In [3]: import math
In [4]: print math.sin(math.pi/4)
0.707106781187
In [5]: import cmath
In [6]: print cmath.sin(2+1j)
(1.40311925062-0.489056259041j)
```

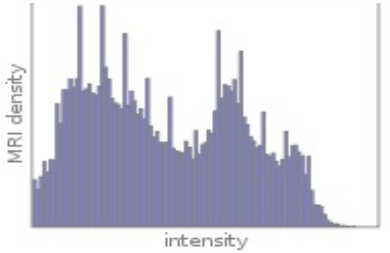
But :

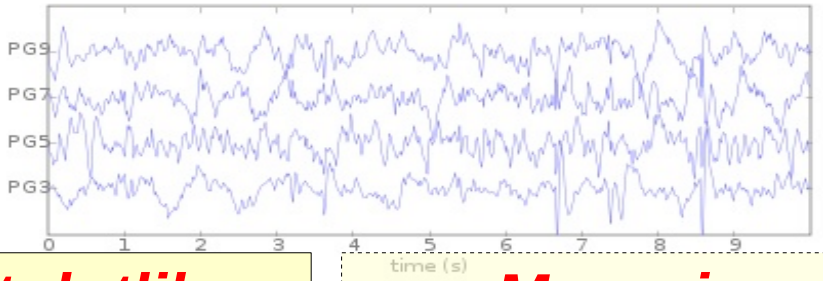
- ***These are limited to simple calculations (single values at at time)***
- ***A different module must be used for complex numbers calculations***

(you can forget them for scientific applications)

Scientific modules







scipy **:**
Advanced scientific calculations (refinement, transforms, solving, integration,...)

numpy **:**
Basic operations on arrays

Matplotlib (pylab) **:**
1D/2D graphics

Mayavi (tvtk, mlab)
3D graphics

Python

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

Arrays

Data needed for a calculation are stored in an array (numpy)

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

Calculations on a array are made element-wise

```
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]
```

Other array declarations :

```
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]
```

Array creations...

...Other array creations :

```
In [17]: print numpy.ones((2,2))                # 2x2 array, filled with 1
[[ 1.  1.]
 [ 1.  1.]]


In [18]: print numpy.zeros((2,2),dtype=int)    # 2x2 array of null integers
[[0 0]
 [0 0]]
```

Simple types for arrays: bool, int, uint, float complex

Advanced types (signed integers): int8, int16, int32, int64
Advanced types (unsigned integers): uint8, uint16, uint32, uint64
Advanced types (floating point): float32, float64, float128
Advanced types (complex numbers): complex64, complex128, complex128

In an array, the data type is the same for all elements (contrary to a list).

Random arrays

```
In [16]: print numpy.random.uniform(0,2,size=(4,4))    # Uniform distribution
[[ 1.58095991  0.15385365  0.82891886  0.0390148 ]    # between 0 et 2
 [ 1.00491457  0.35195621  1.76908828  1.40046406]    # 4x4 array
 [ 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))    # Normal distribution
[[-4.20057151 -3.74078452 -2.68678707  1.31524717]    # average=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]]


Trick : if you want a 100-elements vector, with 30% 1 and 70% zeros :
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.])
```

Array advantages : Vector calculations & speed

```
Simple computation :
import scipy
import time      # Module temps
taille=100000

# With a python loop
nbiter=10
a=range(taille) # loop
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)
```

- Need to write a loop
- 2.083 Mflops (Q6600 2.4GHz)

```
# Using arrays
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)
```

- No loop for c=a+b
- 456.287 Mflops (Q6600 2.4GHz)
- **Speedup * 220 !**

... and with C++ ?

- Much longer code
- Errors with pointers
- 476.19 Mflops (Q6600 2.4GHz, -03)

Arrays vs matrices

An array is not a matrix !

```
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]]
```

Calculations are made elementwise for an array

For a matrix multiplication:

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

... or used numpy.matrix

Linear algebra : scipy.linalg

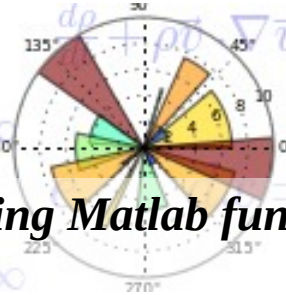
```
In [1]: from numpy import matrix
In [2]: from scipy.linalg import inv, det, eig
In [3]: A=matrix([[1,1,1],[4,4,3],[7,8,5]])   # matrix 3x3
In [4]: b = matrix([1,2,1]).transpose()       # matrix 3x3

In [5]:print det(A)      # not nul ?
1.0
In [6]: print inv(A)*b   # Solution of Ax=b
[[ 1.]
 [-2.]
 [ 2.]]

In [7]: print eig(A)     # vectors and eigenvalues
(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         ]]))
```

Linear algebra : 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.
```

Use scipy.sparse for sparse matrices

1D graphics using matplotlib



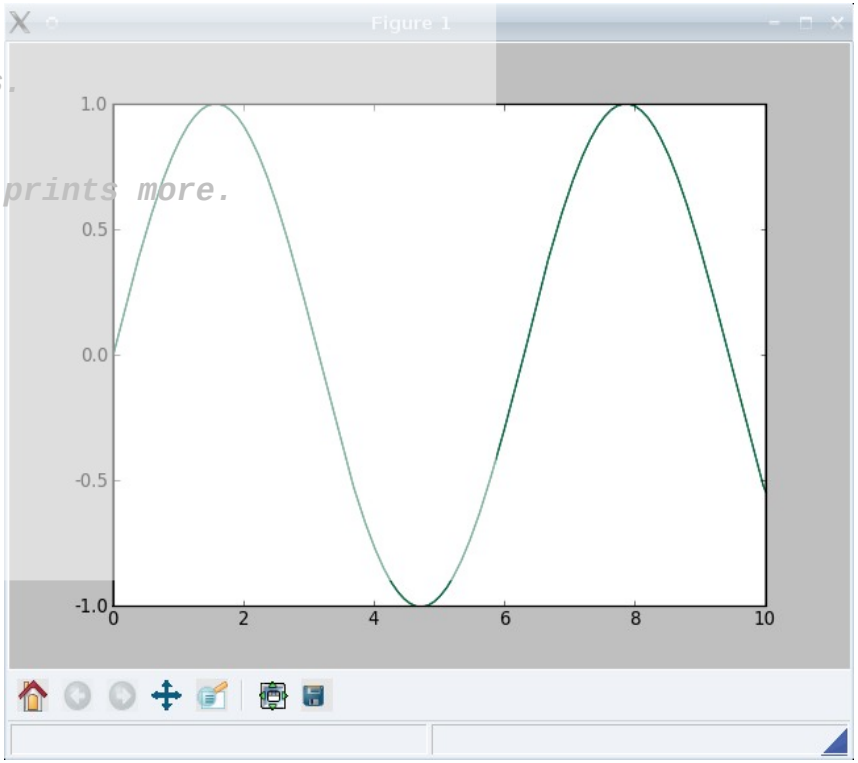
matplotlib

***Matplotlib** : graphical library, cloning Matlab functionalities*
http://matplotlib.sourceforge.net
Lots of examples @ http://matplotlib.sourceforge.net/screenshots.html

To use : start ipython with " -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")
```

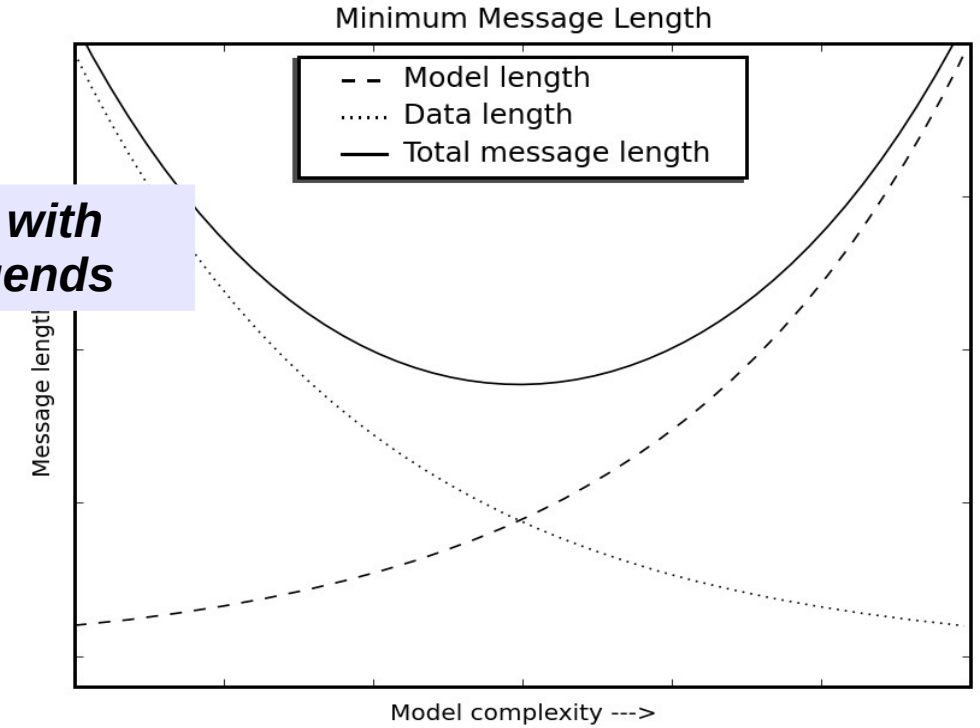
ipython -pylab automatically performs
" from numpy import * "



1D graphics using 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 ar

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([])
```

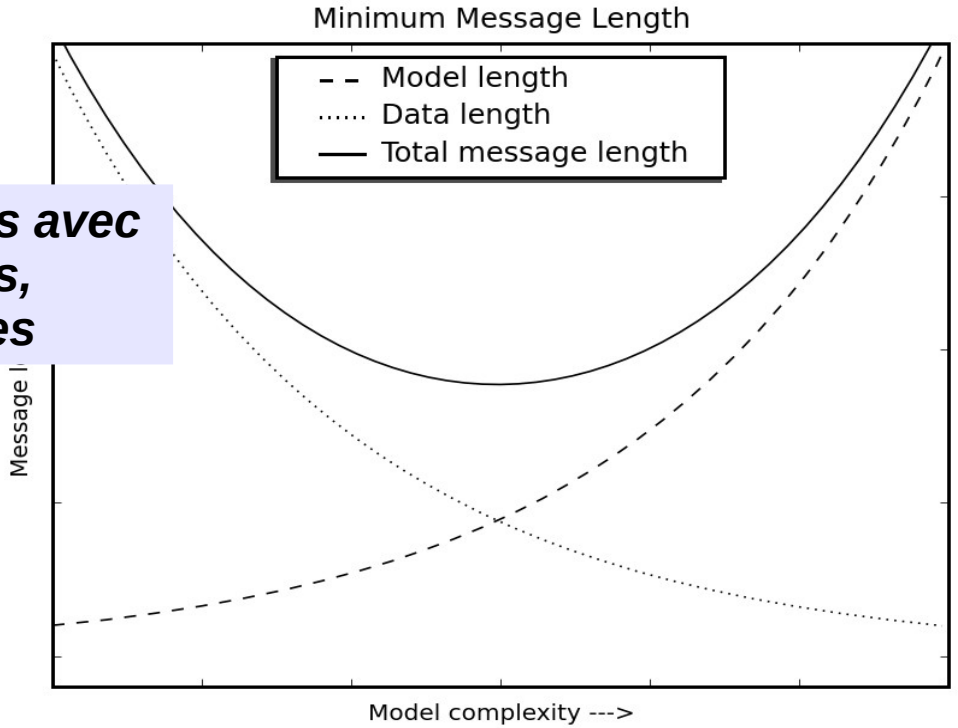
Draw several curves with different styles & legends



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 ai

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([])
```

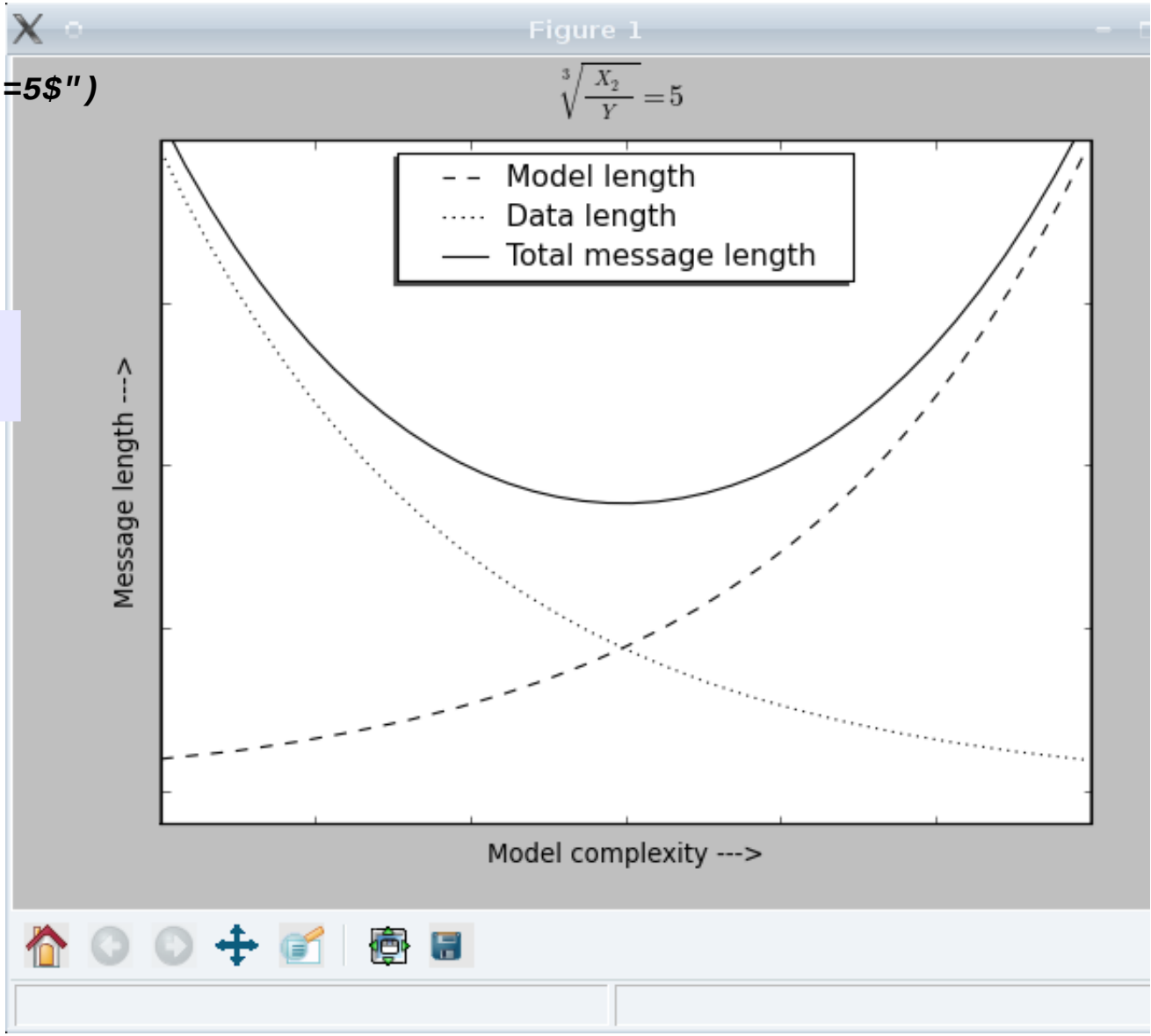
Tracer plusieurs courbes avec des styles différents, des titres & légendes



1D graphics using matplotlib with TeX labels

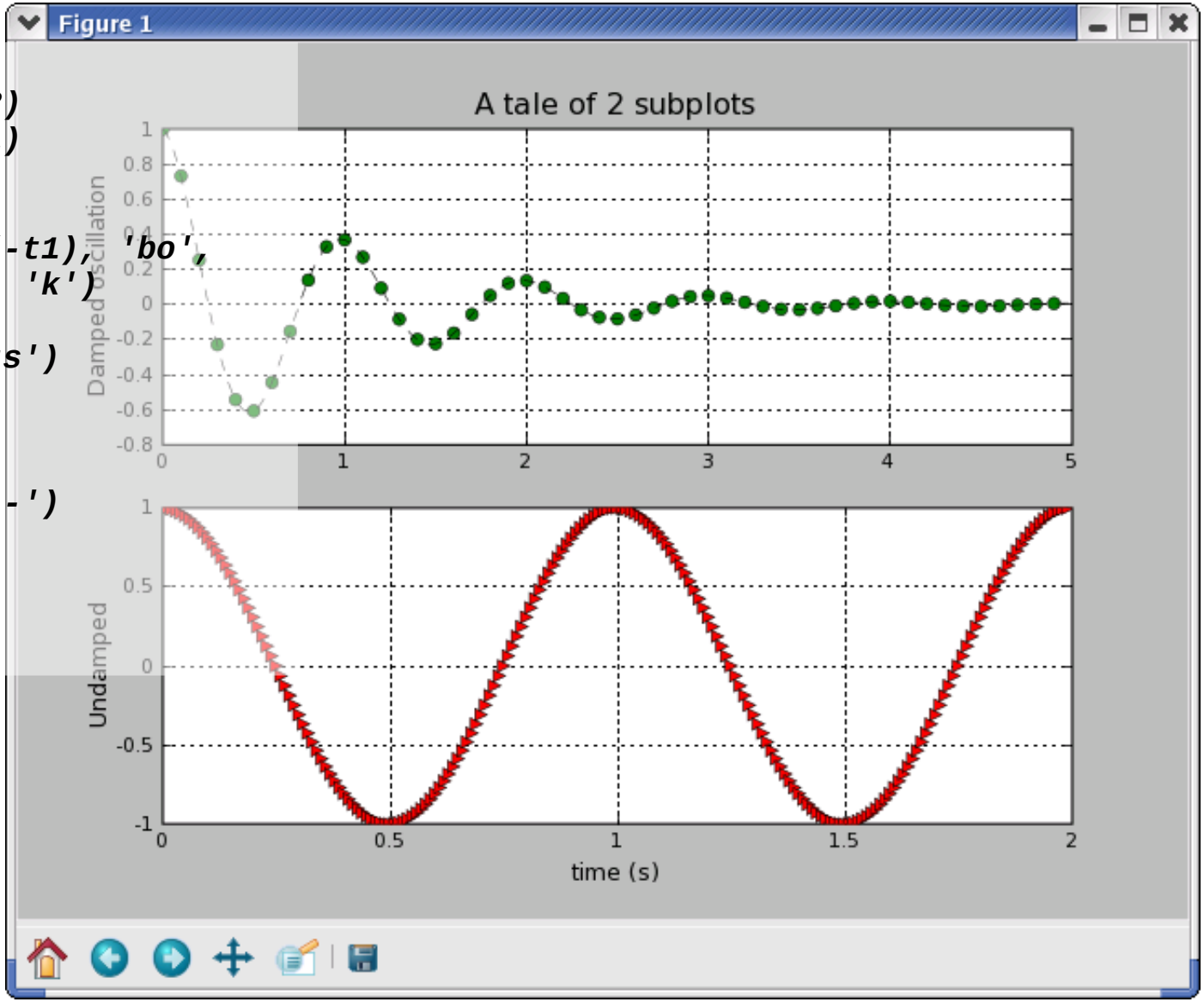
```
title(r"$\sqrt[3]{\frac{X_2}{Y}}=5$")
```

All labels, titles can be written using TeX



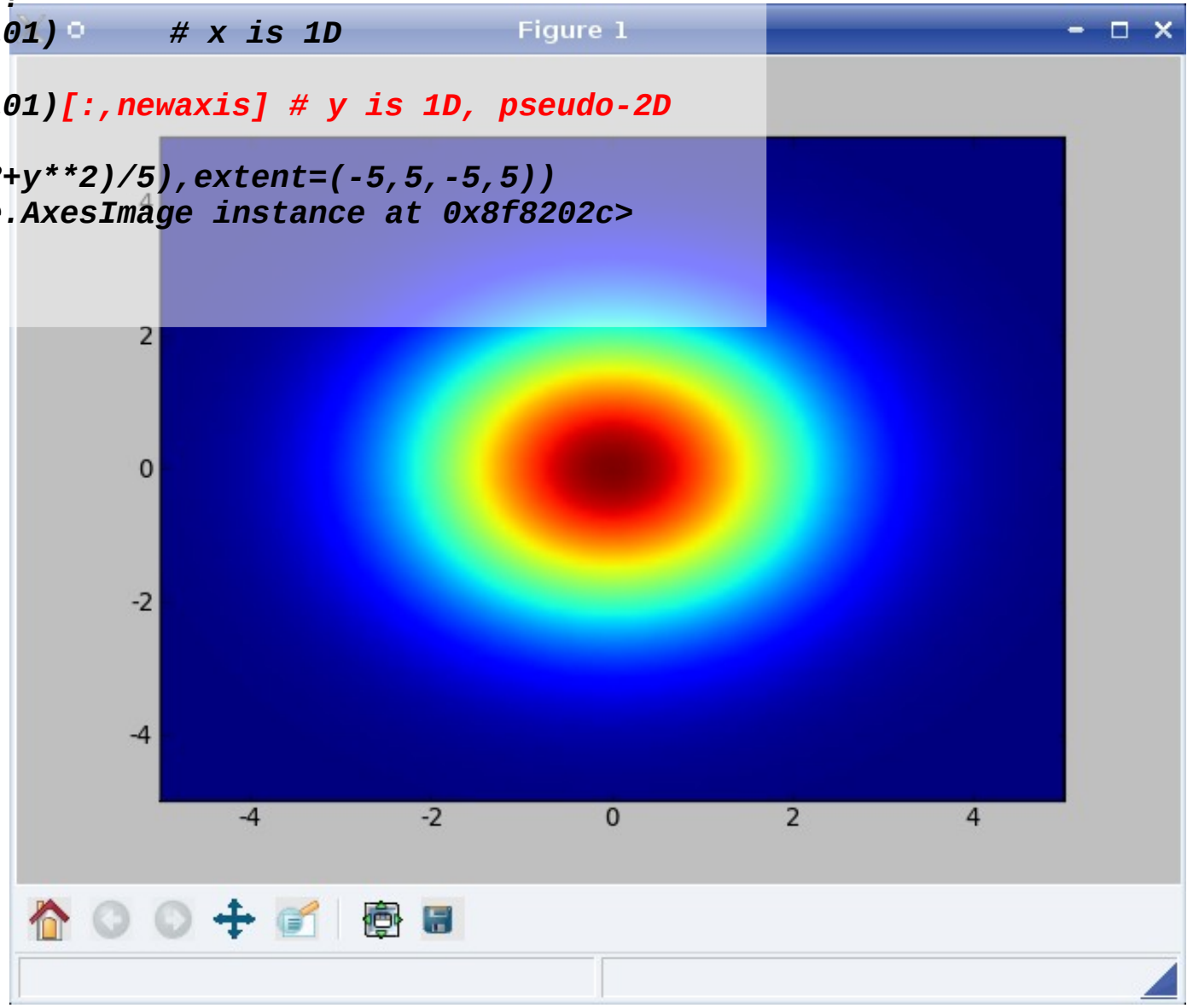
1D graphics using matplotlib
Multiple plots in a single figure

```
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')
```



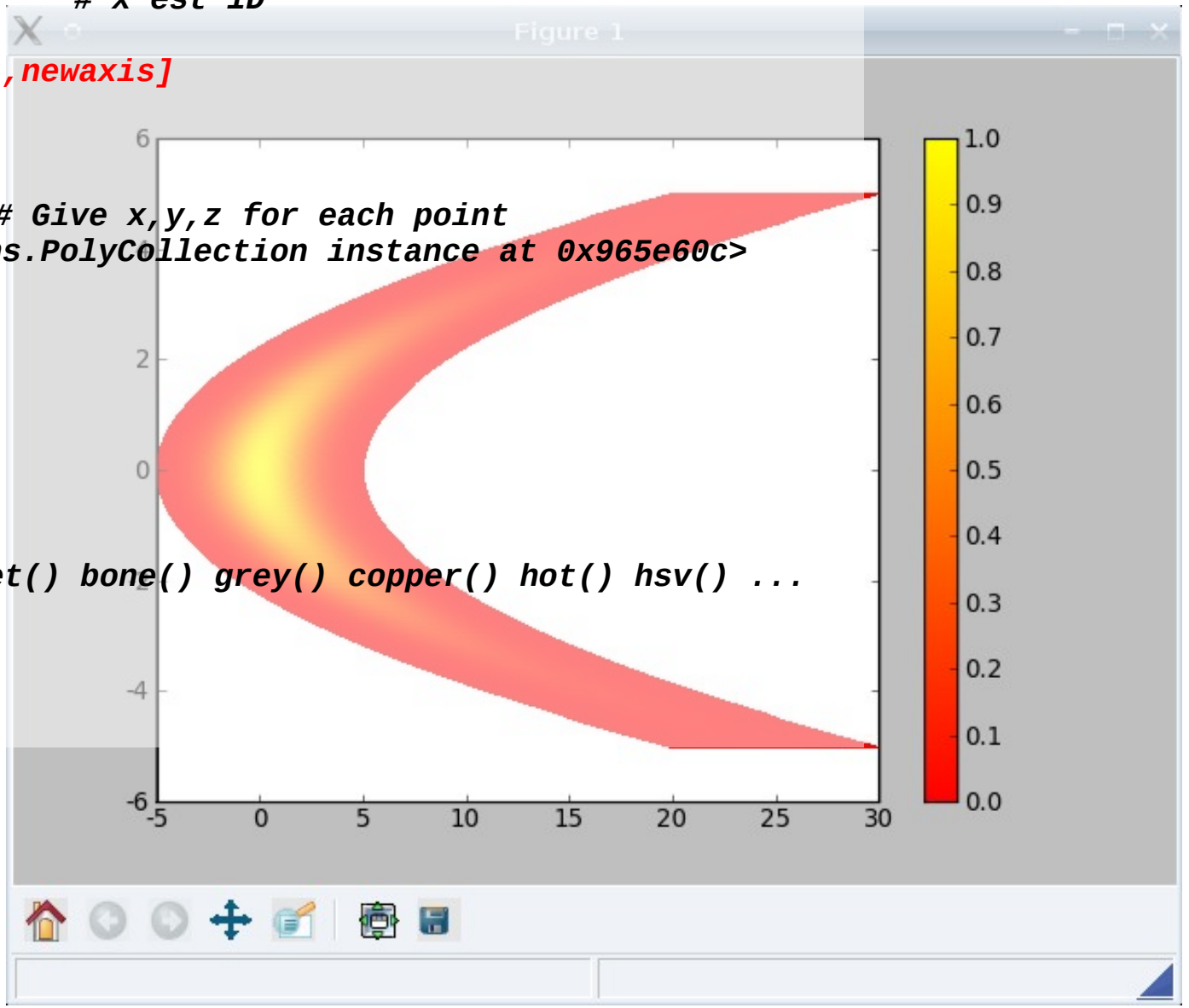
2D graphics using matplotlib imshow

```
AN array is not a matrix !
In [1]: x=linspace(-5,5,101)      # x is 1D

In [2]: y=linspace(-5,5,101)[:,newaxis] # y is 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>
```



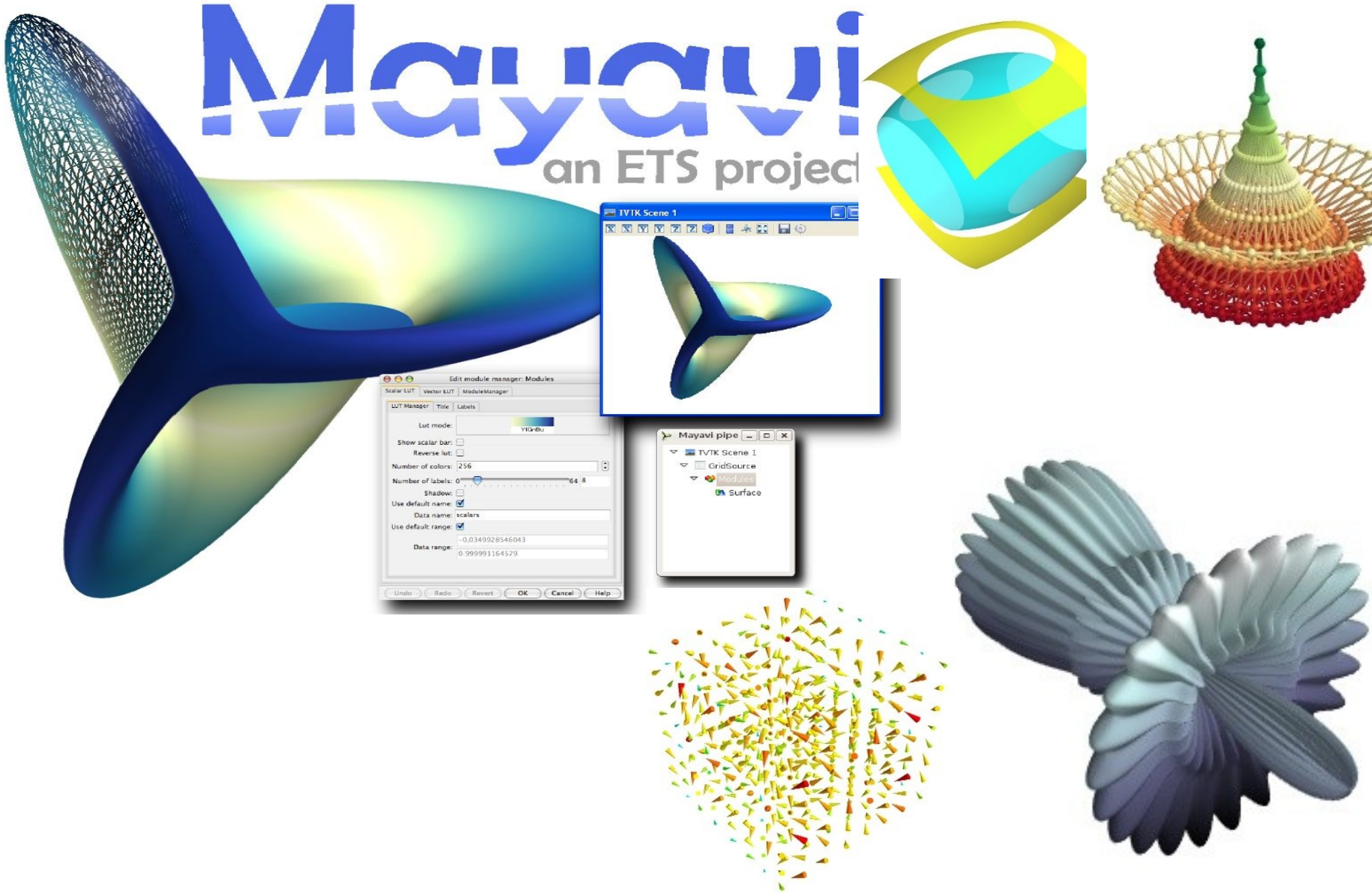
2D graphics using matplotlib
pcolor

```
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)    # Give x,y,z for each point
Out[4]: <matplotlib.collections.PolyCollection instance at 0x965e60c>
```




```
Change the color

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

In [16]: colorbar()
```

3D graphics with Mayavi



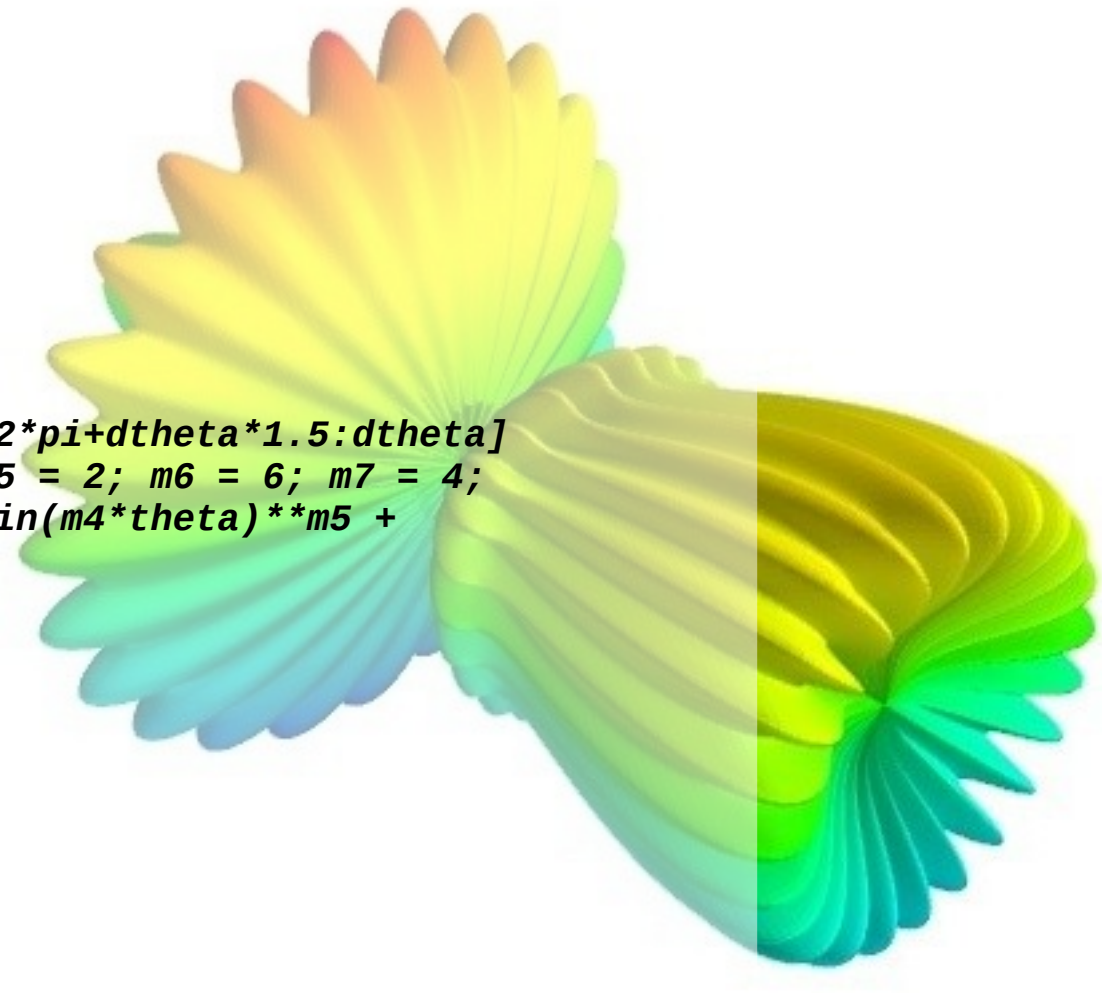
3D graphics with mayavi.mlab

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

" Simple scripting with mlab "

Start ipython with " 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()
```



Integration with 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)
```

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

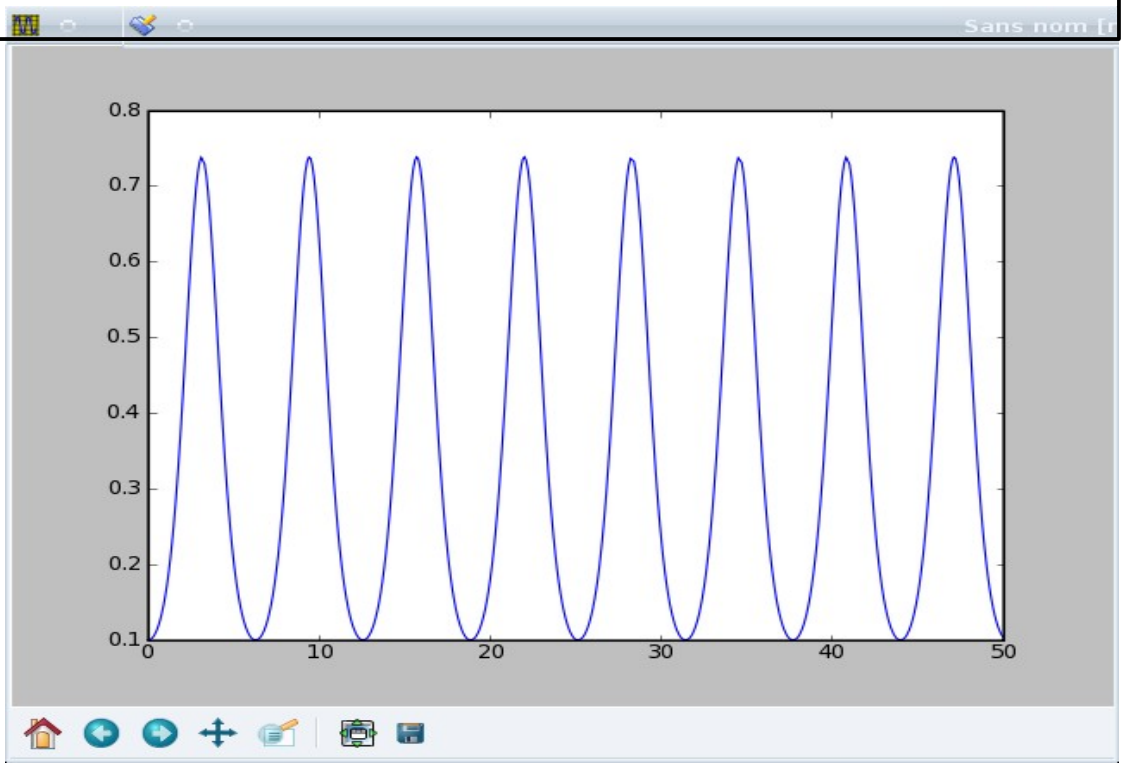
```
help(scipy.integrate)
    Integration routines
    ====================
     Methods for Integrating Functions given function object.

       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.
```

Integration with scipy : 1st order differential equation

Solving a diffrential equation with odeint

```
# First degree equation, example:

from scipy.integrate import odeint
def f(y,t): return sin(t)*y        # First derivative of y(t)
t=linspace(0,50,501)               # list of values for t where we want y(t)
y=integrate.odeint(f,[0.1],t)      # Start at y=0.1
plot(t,y)                          # Display the result
```

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



NB: the integration actually uses an adaptative step, independently of the values given for t

Integration with scipy : 1st order differential equation

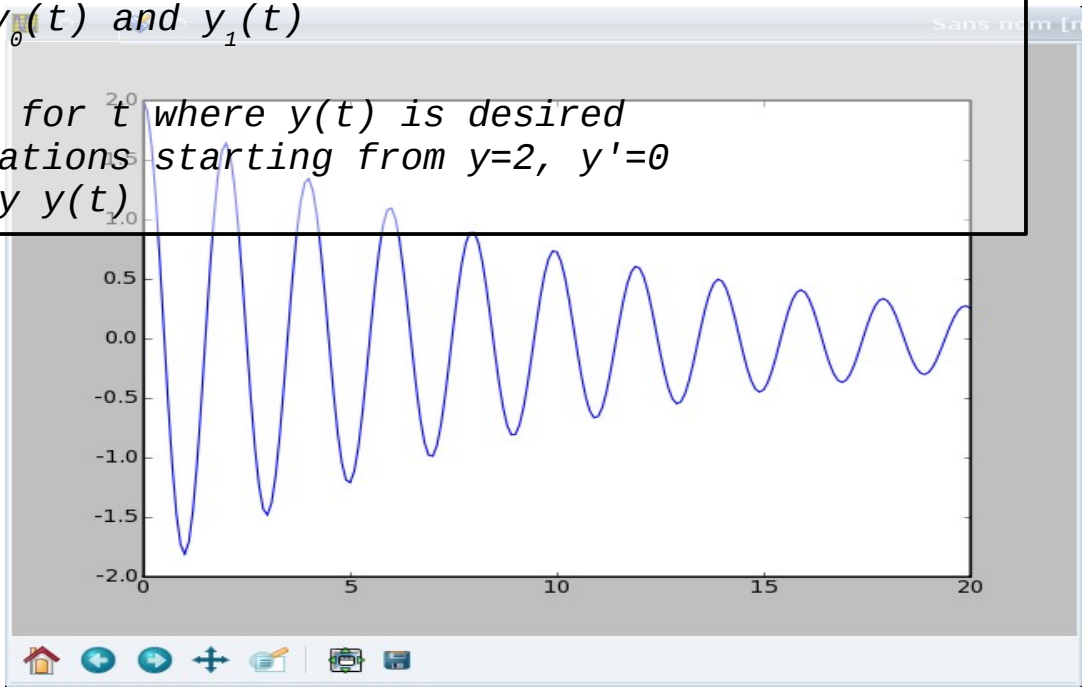
```
# 2nd degree equation, example :
```

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

```
# Change of variable  - y becomes [y0, y1] : pseudo-first degree :
```

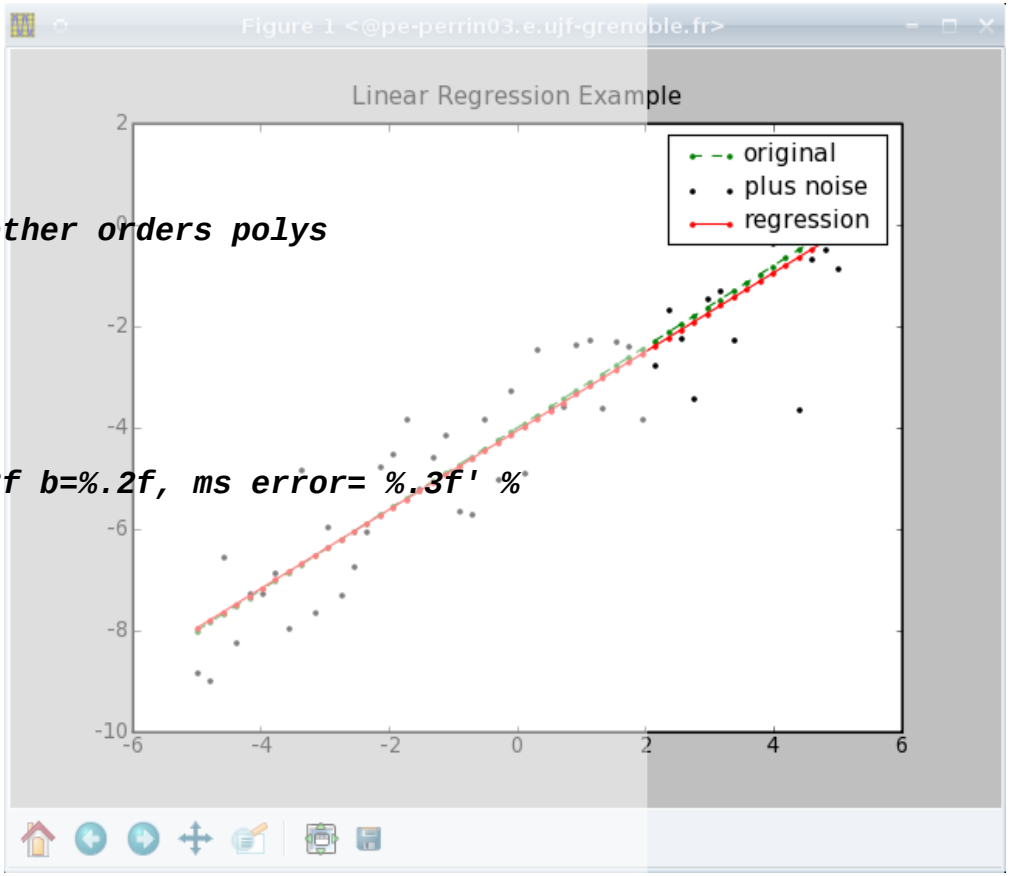
$$\begin{cases} y_0 = y \\ y_1 = \dfrac{\partial y}{\partial t} \end{cases} \qquad \begin{cases} \dfrac{\partial y_0}{\partial t} = y_1 \\ \dfrac{\partial y}{\partial t} = -10 * y_0^2 - 0.2 * y_1 \end{cases}$$

```
def f(y,t):           # Derivative of y0(t) and y1(t)
    return [y[1],-10*y[0]-0.2*y[1]]
t=linspace(0,20,201)          # Values for t where y(t) is desired
y=integrate.odeint(f,[2,0],t) # Calculations starting from y=2, y'=0
plot(t,y[:,0])                # Display y(t)
```



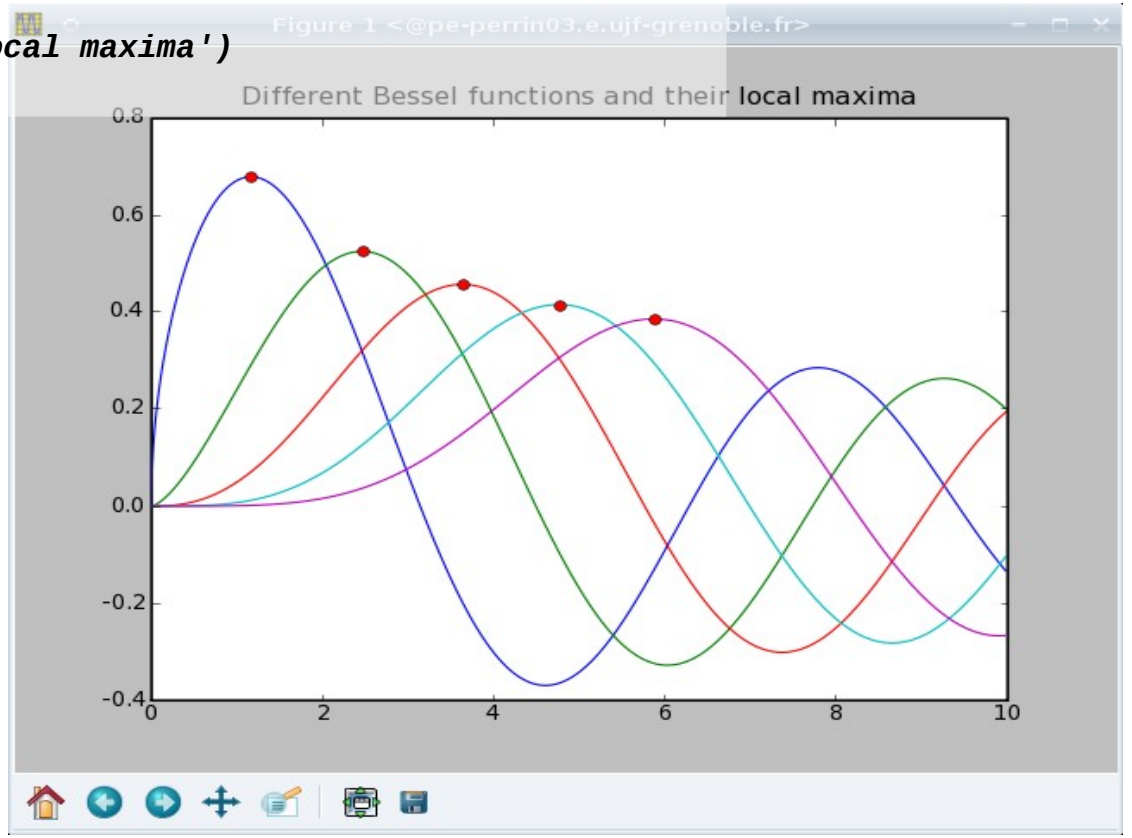
Data refinement
Linear regression

```
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'])
```



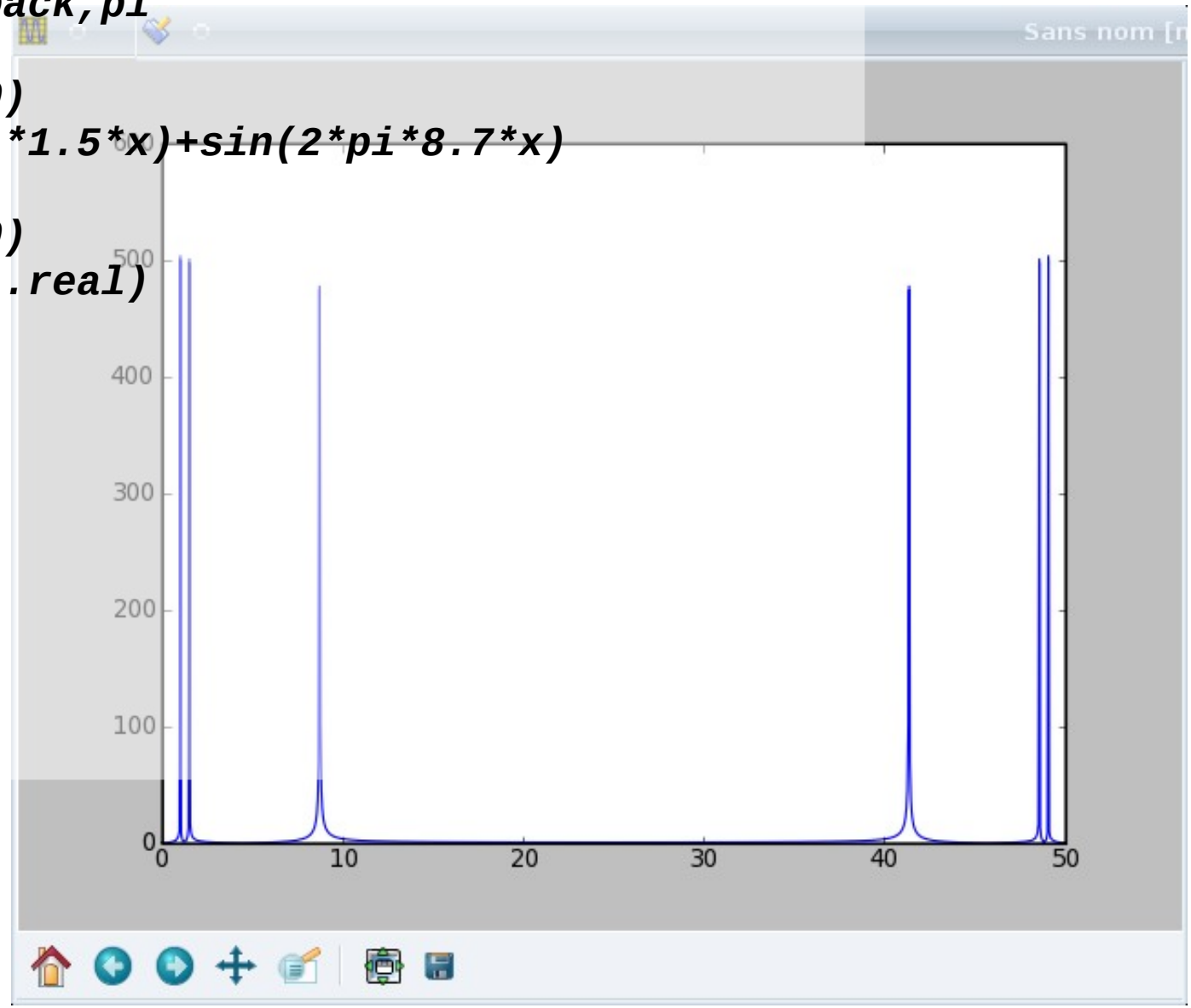
Data refinement
Search for an extremum

```
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')
```



(Fast) Fourier Transform

```
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)
```



Available :
FFT 1D, 2D, 3D,...

Links

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

Part III

- ***Object Oriented Programming (principles)***
- ***File input/output***

Object Oriented Programming (OOP)

- *In OOP, new data types can be created as "classes"*
- *An object belonging to this class is an "instance"*
- *A class includes: data members (attributes) and function members (methods)*

Example:

```
class MyClass:
    x=12


a=MyClass()      # Create an object of type MyClass
print a.x        # Access to attribute x in object a
a.x=25           # Change attribute x in object a
b=MyClass()      # Another object of type MyClass
print b.x        # Access to attribute x in object a
12
```

Data & classes

- *In python, you can dynamically add data inside an object:*

```
class MyClass:
    x=12

a=MyClass()      # Create an object of type MyClass
print a.x        # Access to attribute x in object a
a.x=25
a.y=12           # Add attribute y
a.name='toto'    # Add attribute 'name'
```

Note : for the clarity of the code, it is advised to add data members at the beginning of the code, or in the constructor

Note 2: an added attribute is only available in the object it was added to !

Classes & documentation

- Class can be easily documented:

```
class MyClass:
    """ This is an example class """
     x=12


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

    class MyClass
     |  This is an example class
     |
     |  Data and other attributes defined here:
     |
     |  x = 12
```

Classes & member functions

- *Member functions are declared in the class declaration. They muist used as first paramleter the keyword 'self' which represents the instance of the class (the object)*

```
class MyClass:
    """ This is an example class """
   x=12
    def function1(self):
        print "Value: ",self.x
    def valuex(self):
        return self.x          # Returns a value
    def add(self,y=1):         # Adds a value (default: 1)
        self.x += y
a=Myclass()
a.function1()  # returns: "Value: 12"
a.add(20)  # adds 20 to a.x
a.add()# adds 1 to a.x
```

Classes & special methods

- ***Some special methods use predefined names (with two '_' before & after):***
- ***__init__ : class constructor – this function is automatically called whenever an object is created. It can be used to pass parameters:***

```
class MyClass:
    def __init__(self, x, x2=5):  # constructor – one default value
        self.x=x
        self.y=x2

a=MyClass(1,2)      # Creates an object with a.x=1 and a.y=2
```

Classes & special methods

- ***Some special methods use predefined names (with two '_' before & after):***
- ***__init__ : class constructor – this function is automatically called whenever an object is created. It can be used to pass parameters:***

```
class MyClass:
    def __init__(self, x, x2=5):  # constructor – one default value
        self.x=x
        self.y=x2

a=MyClass(1,2)      # Creates an object with a.x=1 and a.y=2
```

Classes & special methods

- Some special methods use predefined names (with two '_' before & after):
- __str__ : allows interpreting the object as a string
- __add__ : allows adding two objects using '+'
- __eq__ : allows testing equality of two objects using '=='
- etc...

```
class MyClass:
    x=12
    def __str__(self): return "MyClasse, value=%d"%(self.x)
    def __add__(self,b):
        a=MyClass()    # Creates a new object to be returned
        a.x=self.x+b.x
        return a
    def __eq__(self,b):
        return self.x==b.x


a=MyClass()
b=MyClass()
print str(a)   # equivalent to a.__str__()
print a+b      # equivalent to a.__add__(b)  => operator overloading: +
print a==b     # equivalent to a.__eq__(b)   => operator overloading: ==
```

Why classes ?

- *To store data, along with functions to interpret/import/export this data :*

```
a=Molecule()
a.Energy()
a.NbAtom()
a.Formula()
len(a)           # Number of atoms
a[0].x           # x coordinate of the first atom
a.NMRSpectra()   # Calculates the NMR spectra
a.save("name.mol",format='mol')  # Save as mol format
```

Why classes ?

- *To "order" your code : a function is always associated with the data associated to it.*

- *If the format for the data changes, the functions handling the data will also be modified – BUT the functions will always keep the same signature (same parameters, same results) => member functions are an abstraction layer which hide the way data are stored and analyzed.*

- *One does not need to know the details of a function / a class to use it. Only the documentation (parameters, return values) need to be well documented*

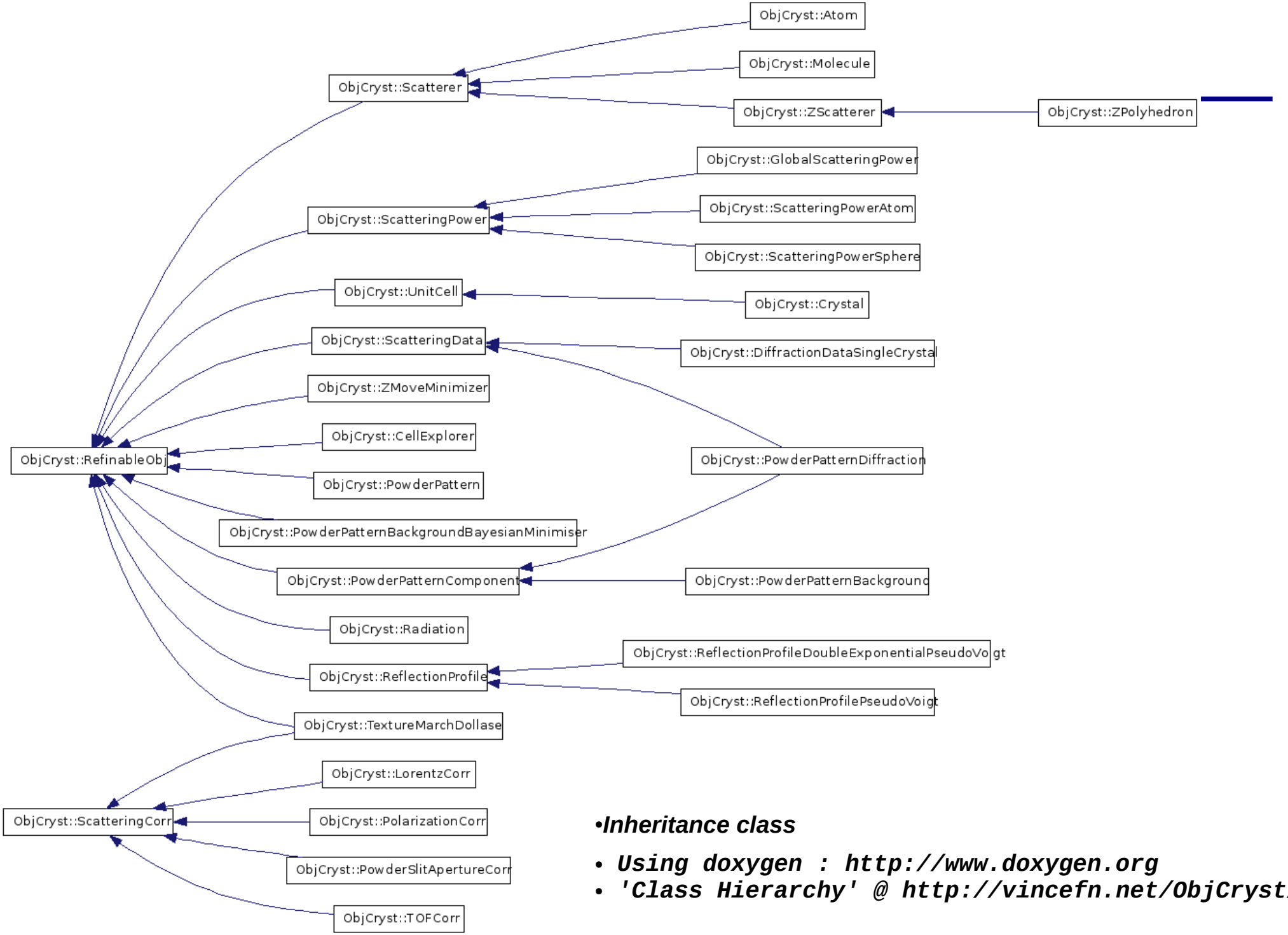
Classes & inheritance

- A class can inherit from another: it gets all the properties (data, functions) of the parent class, and can add or replace some (polymorphism):

```
class Class1:
    x=2
    def __init__(self, x=5):
        self.x=x

class ChildClass(Class1):
    y=2
    def __init__(self,x=5, name='toto'):
        Class1.__init__(x)    # Call the base constructor
        self.name=name

a=ChildClass(6,"tata")
print a.x,a.y,a.nom
```

Note: inheritance should be used only if classes have common properties. For example, a Protein class my inherit from Molecule, but not from Atom (hoever a Molecul will include Atoms)

Classes & inheritance



- ***Inheritance class***
- ***Using doxygen : http://www.doxygen.org***
- ***'Class Hierarchy' @ http://vincefn.net/ObjCryst/***

Classes & polymorphism

- *Polymorphism: when a class inherits from another, it can rewrite a function from the parent:*

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

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

The interest of polymorphism lies in the fact that the function can be called without knowing if the object actually is a Square or Rectangle

Data input/output

- **Basic functions :**

```
f=open("data.txt",'r')          # Open a file, read mode
ll=f.readlines()                # Read all the lines
for l in ll: print l            # Print all the lines
f.close()                       # Close the file

f=open("data.txt",'r')          # Opens a file
l=f.readline()                  # Reads a single line as a string
vx,vy=[],[]
While len(l)>0:                 # End of file => empty line
    s=l.split()                 # Separate data (2-column file)
    vx.append(float(s[0]))
    vy.append(float(s[1]))
    l=f.readline()
f.close()


f=open("data.txt",'w')              # Opens a file for writing
f.write("Ligne 1 du fichier\n")
f.write("%f %f %f\n"%(1,2,3))       # Write formatted data
f.close()
```

Read/write using Pickle/cPickle

The "pickle" and 'cPickle" modules allows to save & read back any type of python object. Documentation : 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 Circumference(self): return 2*(self.x+self.y)

a=Rectangle()                   # Create two rectangles
b=Rectangle()
import pickle
f=open("rect.pickle",'w')
pickle.dump((a,b),f)            # Save the two rectangles
f.close()

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

Note:cPickle is similar to pickle, but written in C and more efficient !
Note 2: better use `cPickle.dump((a,b),f, protocol=-1)` which is more efficient (compression)...
Note 3: saved files are NOT portable ! They can only be read back on the same computer (same python version, same architecture...)

Read/write data using scipy/numpy

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

Useful functions:

- *numpy.loadtxt and numpy.savetxt (simple i/o from/to well-formatted text files)*

- *numpy.save: save a single array in binary form (.npy)*
- *numpy.savez: save one or more arrays in binary form, in compressed format (.npz)*
- *numpy.load: read a .npy or .npz file*

- *Module scipy.io: reads various types of special data (netcdf, Matlab, ...)*

Part IV

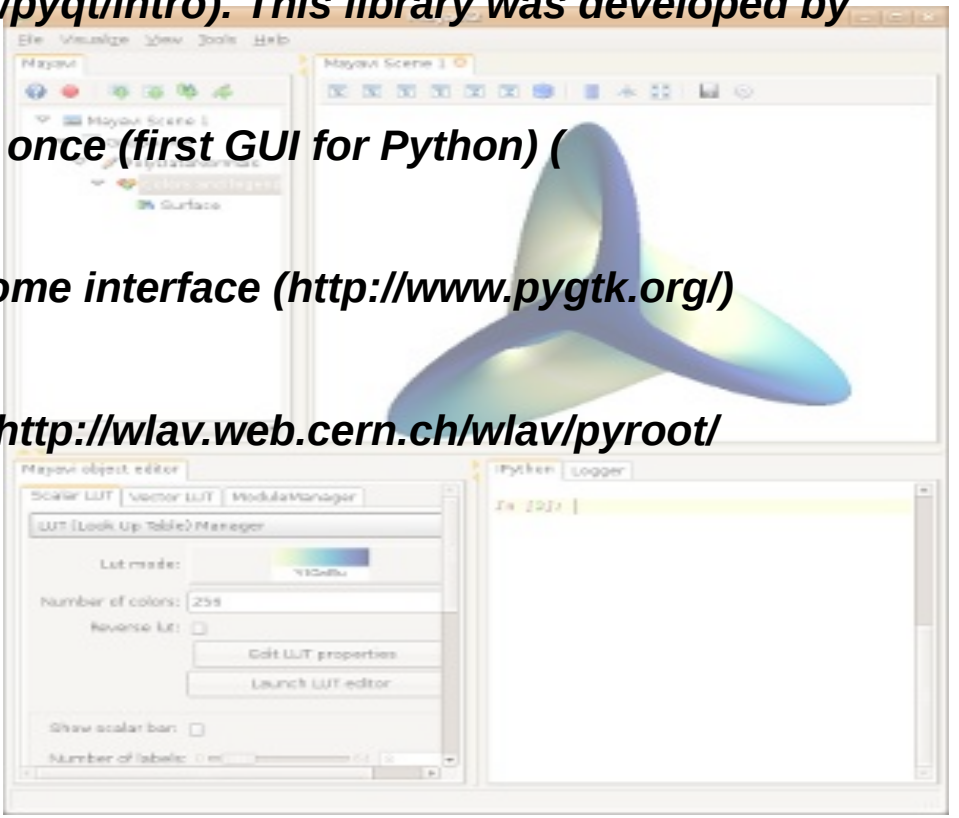
- ***Graphical interfaces***
- ***Data acquisition***
- ***Parallel computing***

Graphical User Interfaces (GUI) with Python

Many Graphical User Interfaces are available with Python. Generally they were written in C or C++, and the interfaced with Python

The main ones (all multi-platforms) are :

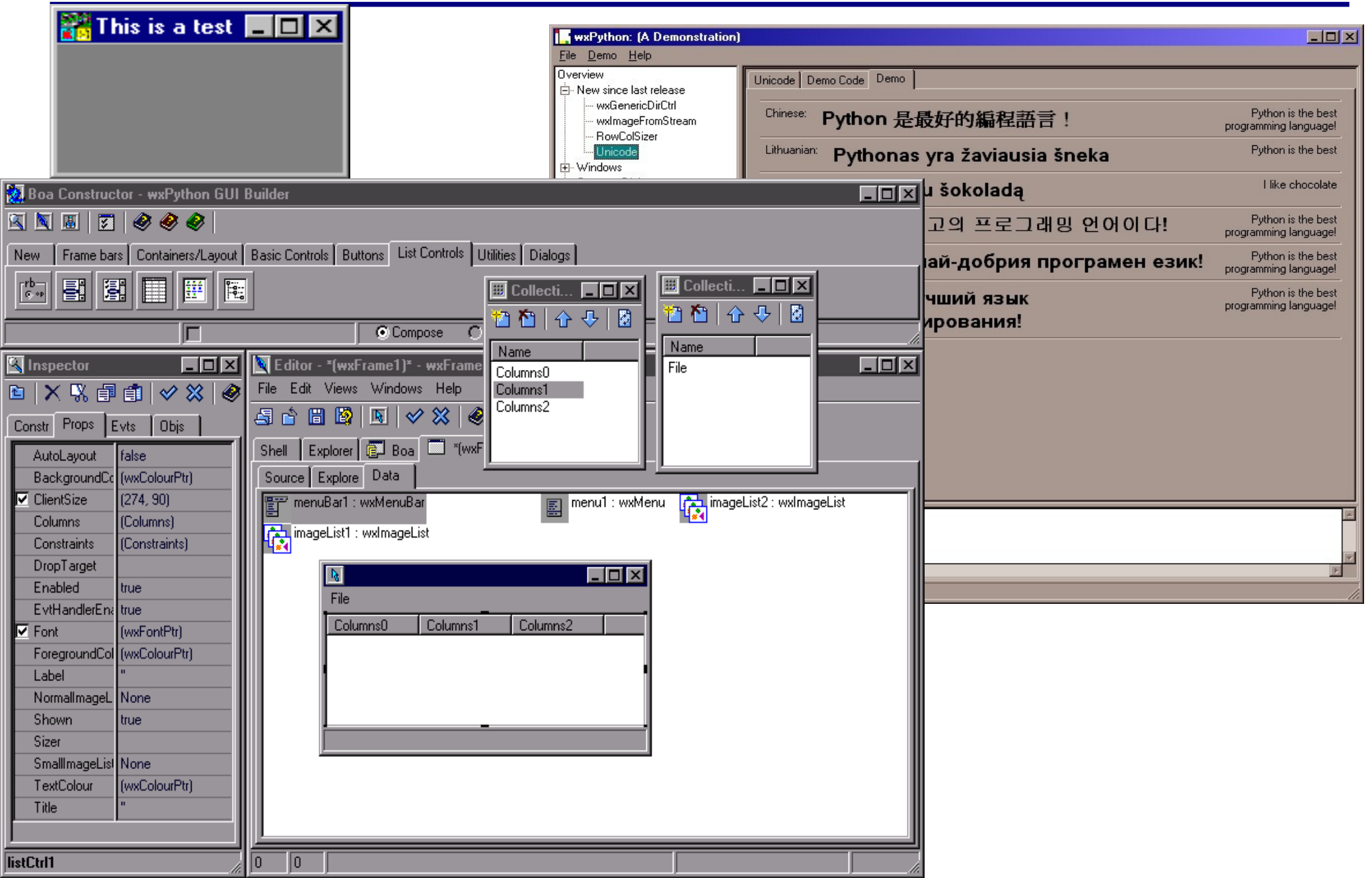
- ***WxPython :based on wxWidgets (http://www.wxpython.org)***
- ***PyQt : based on Qt – the library around which the Linux KDE library is built (http://www.riverbankcomputing.co.uk/software/pyqt/intro). This library was developed by TrollTech, now part of Nokia***
- ***Python-tkinter: based on Tcl/Tk – was popular once (first GUI for Python) (http://wiki.python.org/moin/TkInter)***
- ***PyGTK : based on GTK, used for the Linux Gnome interface (http://www.pygtk.org/)***



Another noteworthy interface: ROOT from CERN : http://wlav.web.cern.ch/wlav/pyroot/

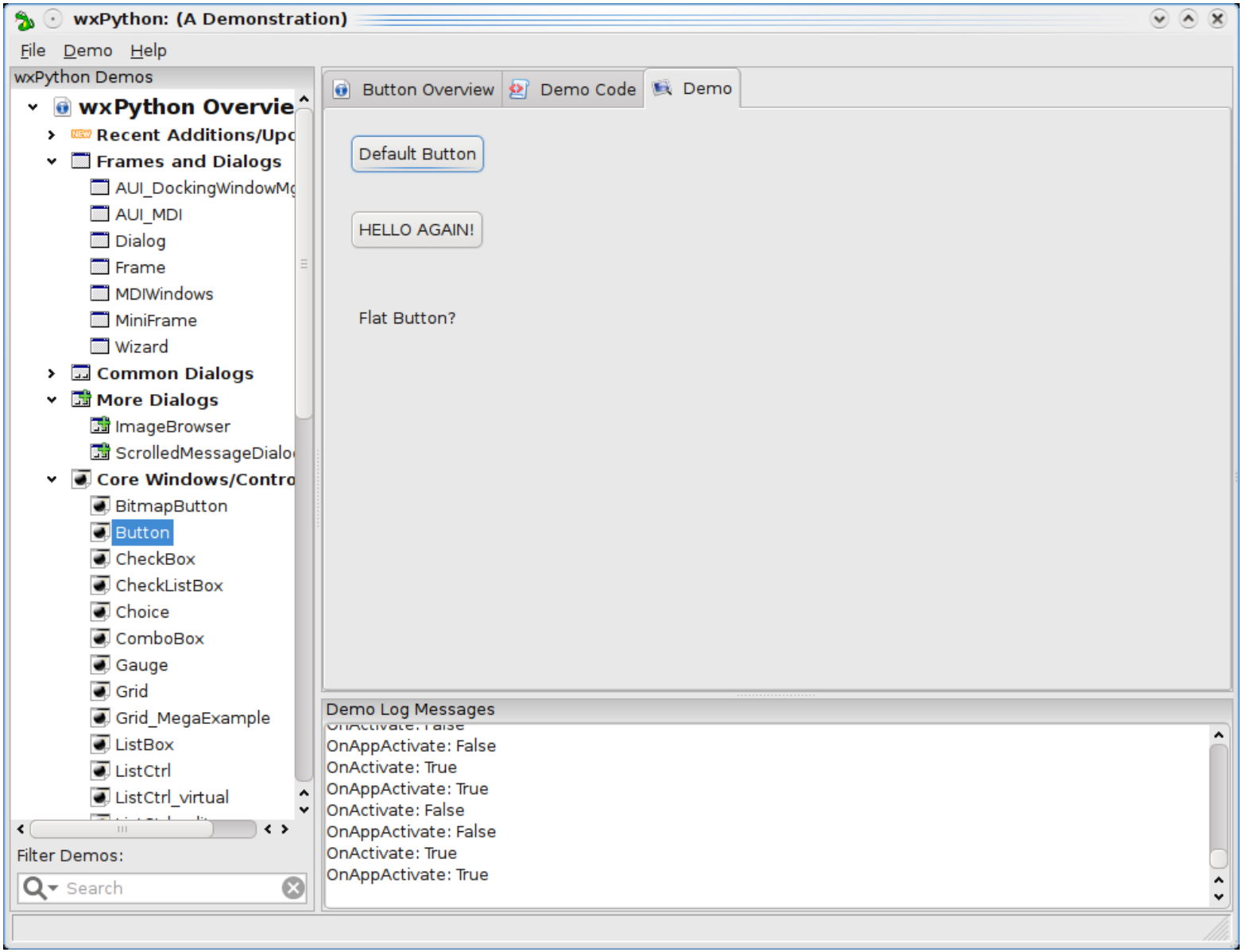
etc...

Examples (using wxPython)



Examples (avec wxPython)

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

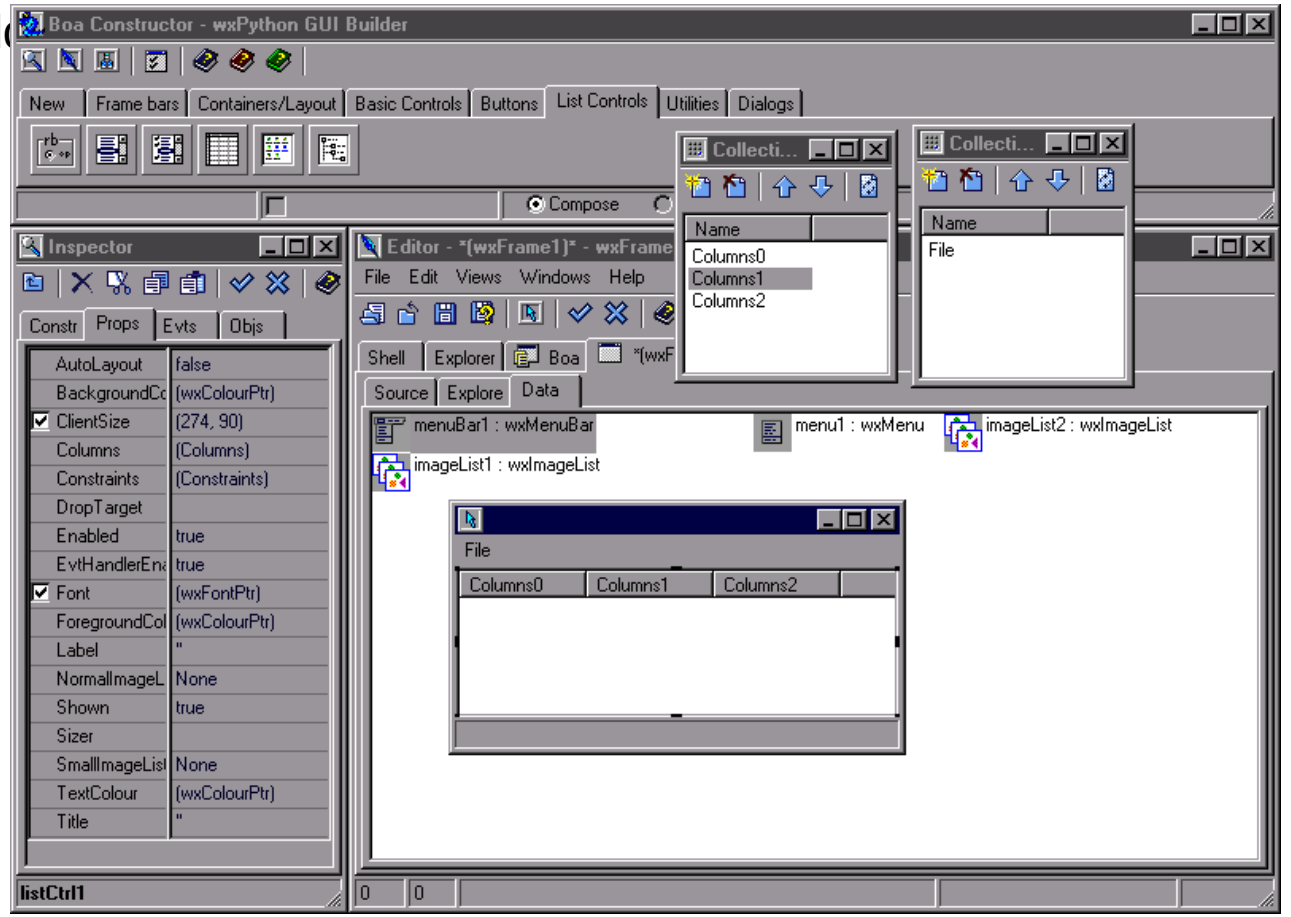


Different types of windows & dialogs

Basic principles:

- Different types of windows and dialogs:
  - http://docs.wxwidgets.org/trunk/hierarchy.html
- Sub-window size&ordering
- Notion of message passing (broad

=> see code examples



Data acquisition

A few libraries for data acquisition (from hardware)

Using USB, from data acquisition cards :

- **PyUL (python-Universal Library) with a "measurement computing" card**

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

- **Using a National Instruments hardware:**

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

Using serial & parallel ports:

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

For all ports(usb, série, gpib) :

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

Parallel computing using Python

Main libraries for parallel computing:

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

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

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

GPU computing
PyCUDA (CUDA):
http://documen.tician.de/pycuda/
PyOpenCL (OpenCL):
http://documen.tician.de/pyopencl/