

DE LA RECHERCHE À L'INDUSTRIE



INTRODUCTION TO SCIENTIFIC PYTHON

OLIVIER TACHE



www.cea.fr



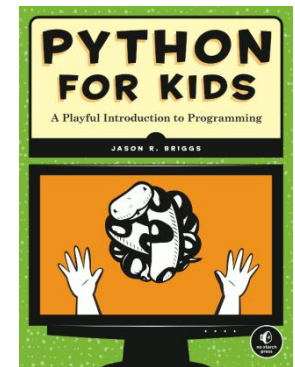
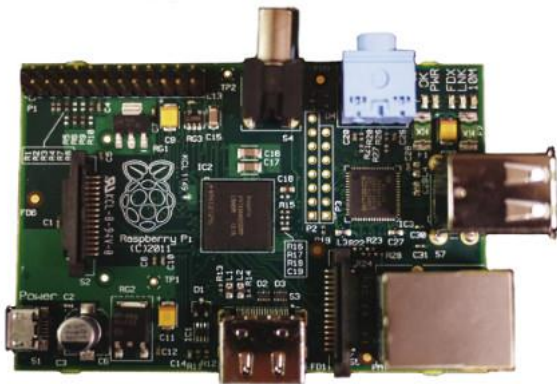
Ce(tte) œuvre est mise à disposition selon les termes de la [Licence Creative Commons Attribution - Pas d'Utilisation Commerciale - Partage dans les Mêmes Conditions 3.0 France](https://creativecommons.org/licenses/by-nc-sa/3.0/fr/).

PROGRAM IN PYTHON AND PUBLISH IN NATURE



Neil Ibata, en stage à l'observatoire de Strasbourg avec son père, a été le premier à déceler la rotation de galaxies naines autour d'Andromède grâce à un programme informatique qu'il avait mis au point. À seulement 15 ans, Neil Ibata vient de réaliser le rêve de beaucoup de chercheurs : co-signer un papier dans NATURE : "A vast, thin plane of corotating dwarf galaxies orbiting the Andromeda galaxy", A.I. Ibata *et al.*, [Nature 493 \(2013\) 62](#)". ([voir l'article du Figaro](#)).

"Je venais de faire un stage pour apprendre le langage informatique Python", raconte l'élève de 1^{ère} S du lycée international des Pontonniers à Strasbourg. [Python](#) est un langage de programmation Facile à apprendre "Open source" et gratuit. Il comporte des modules scientifiques couvrant de nombreux domaines. **Vous avez des tonnes de données inexploitées, [utilisez Python](#) !**



CONTENTS

Philosophy of Python

Scientific Python

Using Python

Syntax elements

- Types
- Conditions
- loop
- files
- Functions
- Class

Modules

- Its own module
- Scientific Modules
- Numeric Array

Plotting

- Introduction to Gnuplot
- Gnuplot and Python
- Matplotlib

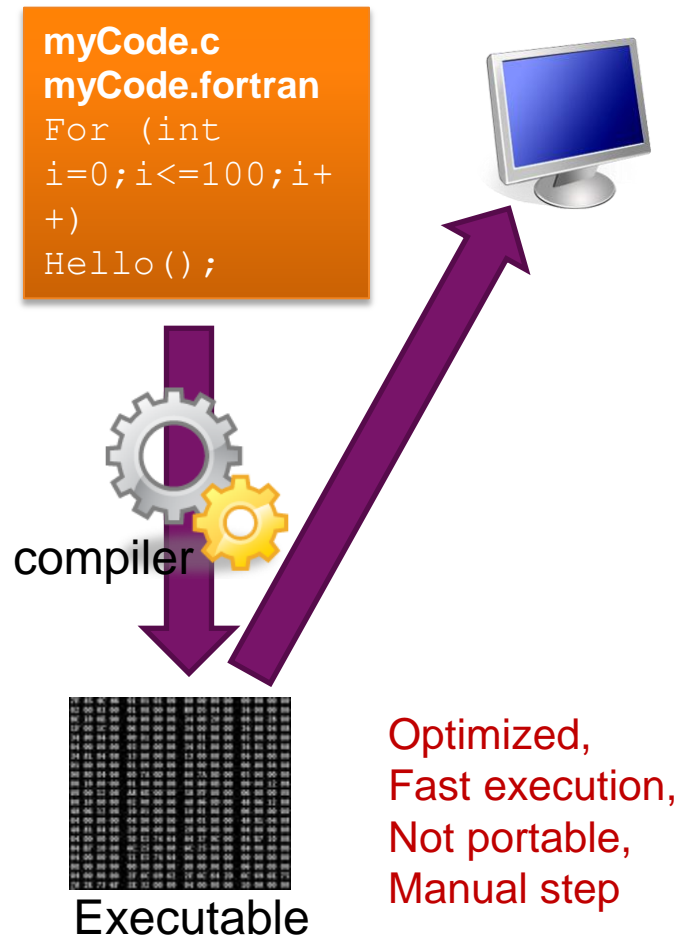
Example of use for fits

Python : oriented objects ? Classes

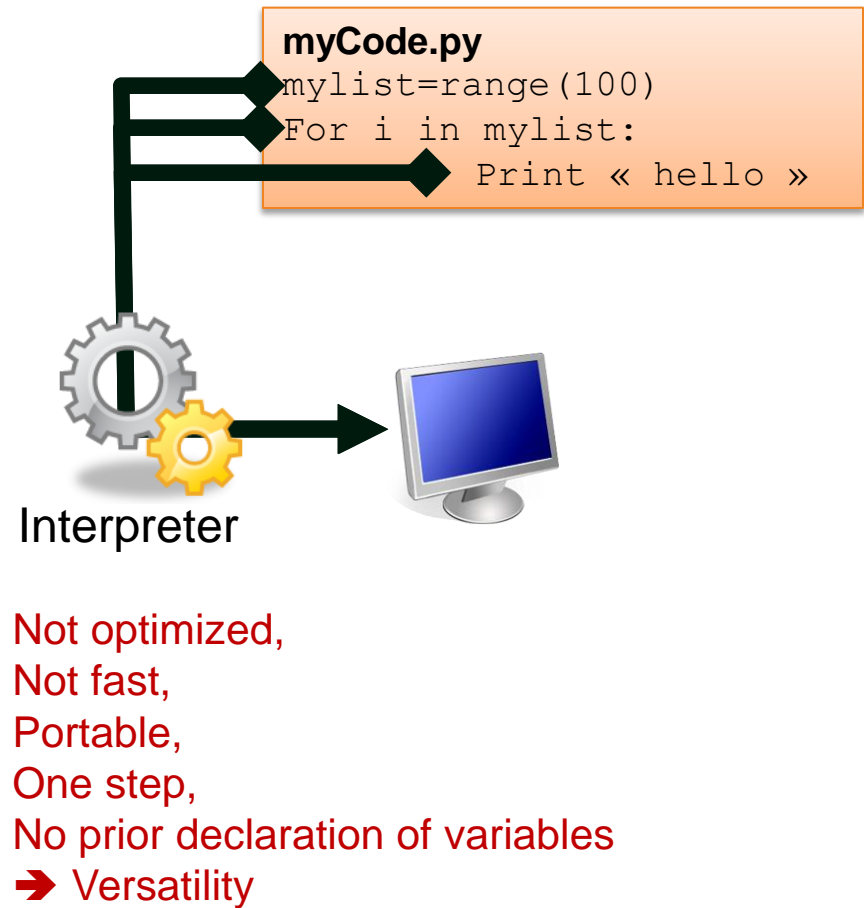
- « **simple** » programming language, which allows you to concentrate on the application and not the syntax
- Readable language, The indentation is mandatory.
- Object oriented, **evolved**
- modular, **scalable**
- Use in many contexts
- Interface with other languages (Fortran, C,...)
- **Portable** (can be used on linux, mac, windows,...)
- Many graphical interfaces
- Many scientific libraries
- Free and Open Source

PYTHON PHILOSOPHY : COMPILER / INTERPRETER

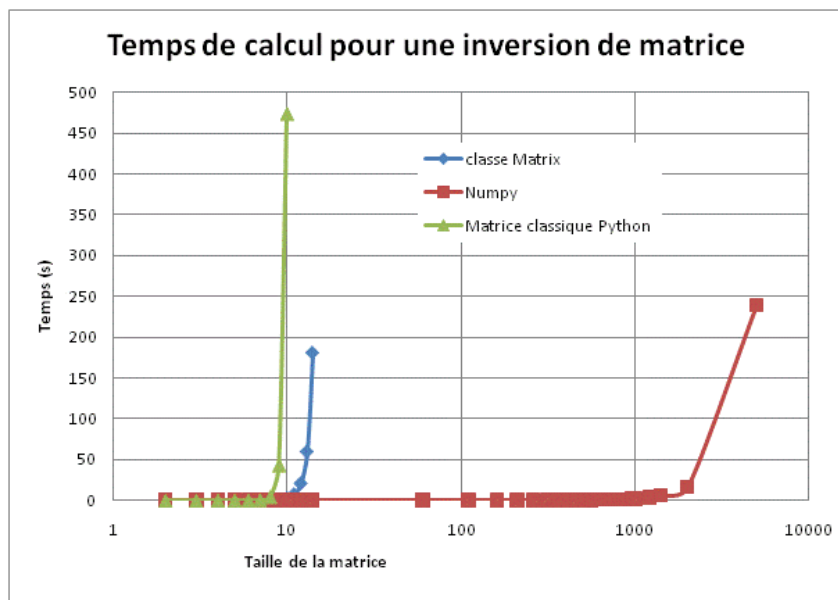
C, fortran are **compiled languages**



Python is an **interpreted language**



Interpreted language -> performances ?



Type of solution	Time taken (sec)
Python (estimate)	1500.0
Python + Psyco (estimate)	1138.0
Python + NumPy Expression	29.3
Blitz	9.5
Inline	4.3
Fast Inline	2.3
Python/Fortran	2.9
Pyrex	2.5
Matlab (estimate)	29.0
Octave (estimate)	60.0
Pure C++	2.16

solving the 2D Laplace equation
<http://www.scipy.org/PerformancePython>

→think different (no “fortran” like)

→Use Numpy et scipy

Gnuplot

Matplotlib

Plotting

Scipy

Scientific functions

Numpy

Numeric arrays

Python

Programming Language

SCIENTIFIC PYTHON: PLOTTING WITH GNUPLLOT

Gnuplot

Matplotlib

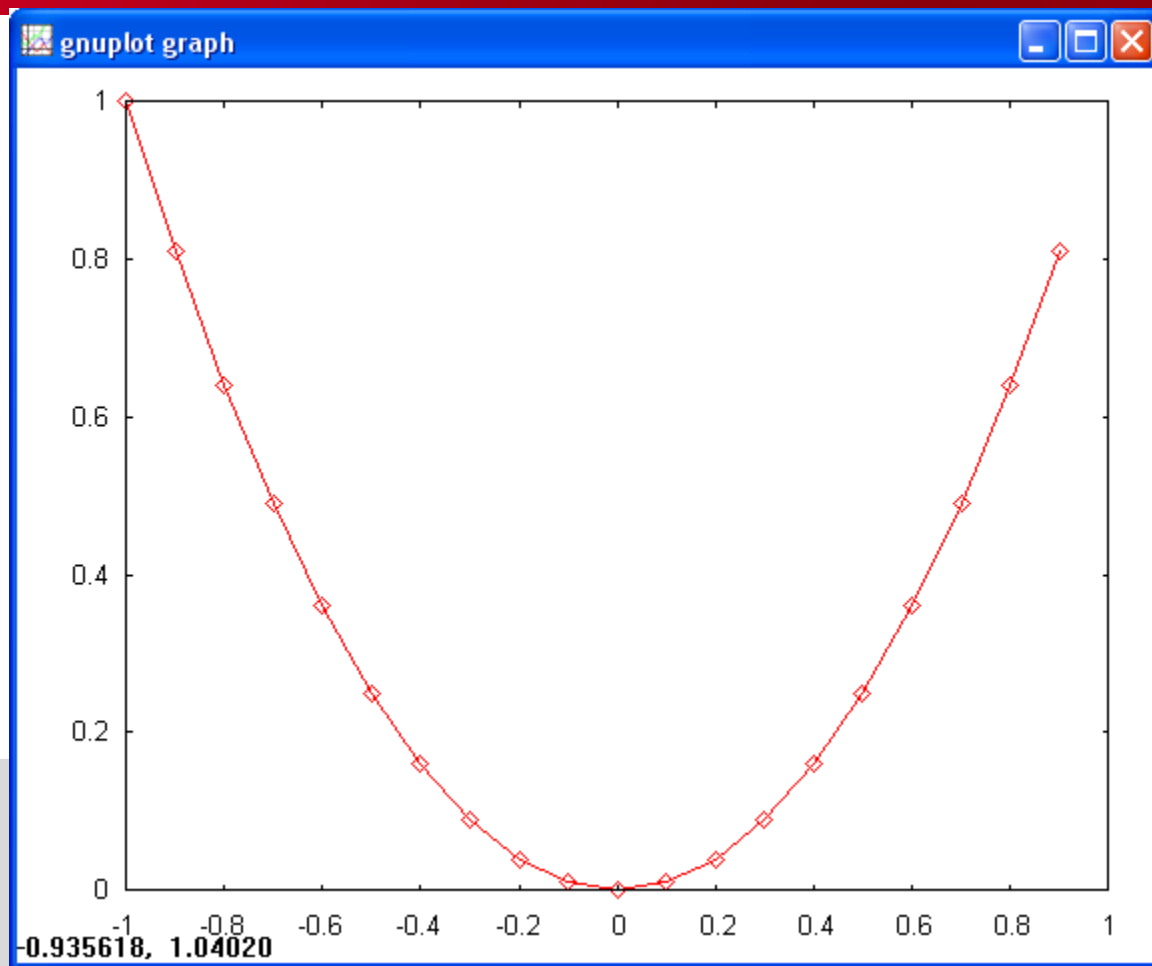
Scipy

Numpy

Python

```
from numpy import *
x=arange(-1.0,1,0.1)
y=x*x

import Gnuplot
g=Gnuplot.Gnuplot()
d=Gnuplot.Data(x,y,with_='linespoints')
g.plot(d)
```



Gnuplot

Matplotlib

Scipy

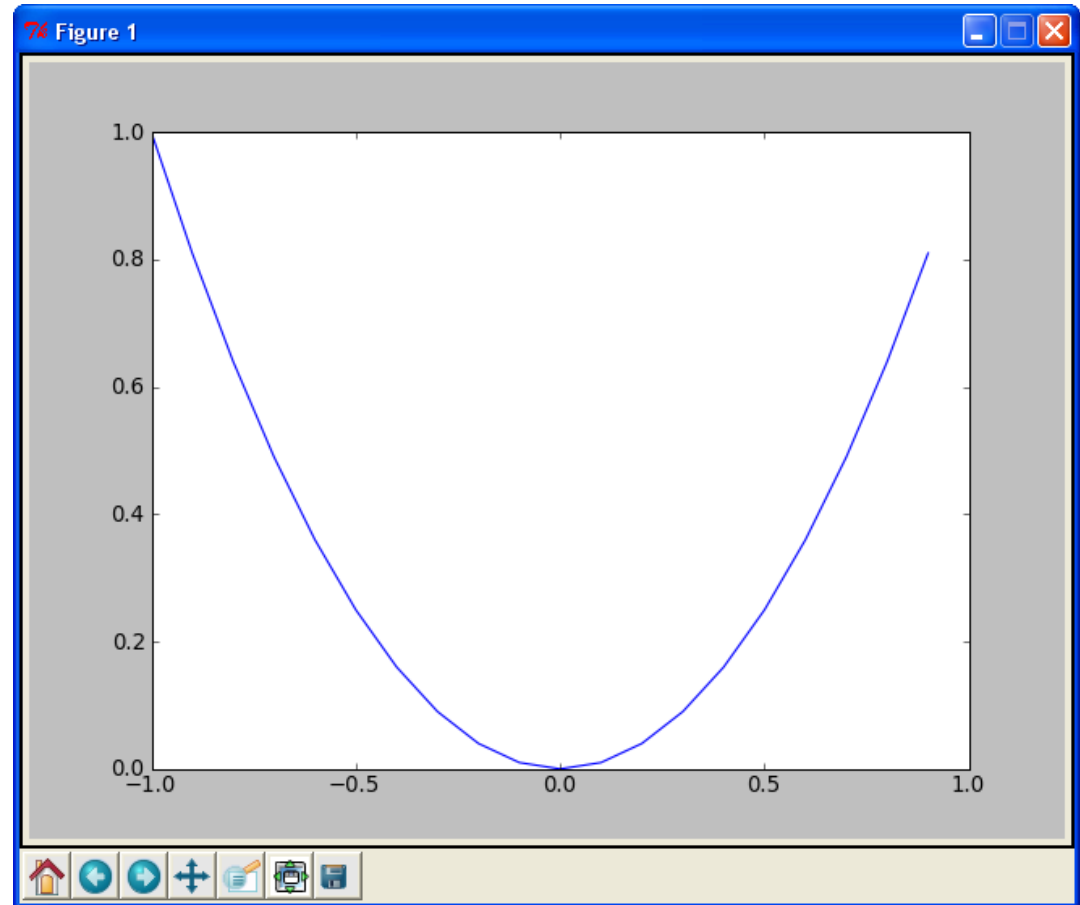
Numpy

Python

```
import pylab
from numpy import *
x=arange(-1.0,1,0.1)
y=x*x
```

```
plot(x, y)
```

```
show()
```



SCIENTIFIC PYTHON

Gnuplot

Matplotlib

0.99

Plotting

Scipy

0.7.1

Scientific functions

Numpy

1.3.0

Numeric arrays

Python

2.5

Programming Language



Pierre Raybaut (CEA / DAM)

Python(x,y) is a Python distribution Python for scientific development

1. collecting scientific-oriented Python libraries and development environment tools ;
2. collecting almost all free related documentation ;
3. providing a quick guide to get started in Python / Qt / Spyder ;
4. providing an all-in-one **setup program**, so the user can install or uninstall all these packages and features by clicking on one button only.

For windows

1- Python shell

The image shows two windows. The top window is the 'Python Shell' window, which displays the Python 2.5.4 startup screen with copyright information and a warning about firewall software. The bottom window is a 'Command Prompt - python' window showing the output of the 'dir' command in the C:\Python25 directory, listing files like LICENSE.txt, NEWS.txt, and various executables.

```
Python 2.5.4 (r254:67916, Dec 23 2008, 15:10:54) [MSC v.1310 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.

>>>

Personal firewall software may warn about the connection IDLE
makes to its subprocess using this computer's internal loopback
interface. This connection is not visible on any external
interface and no data is sent to or received from the Internet.

IDLE 1.2.4
>>>
```

```
C:\Python25>python
Python 2.5.4 (r254:67916, Dec 23 2008, 15:10:54) [MSC v.1310 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

2- execute in the shell

The image shows a text editor window displaying the contents of a file named 'test.py'. The file contains a shebang line, a docstring, and Python code that imports Gnuplot and Numeric modules, and defines a 'wait' function.

```
#!/usr/bin/Python Shell
# $Id: test.py 18 22:33:00 mhagger Exp $
# Copyright 2008 Gael Haggerty <mhagger@alum.mit.edu>
# This file is licensed under the GNU Lesser General Public License
# (LGPL). See LICENSE.txt for details.

"""test.py -- Exercise the Gnuplot.py module.

This module is not meant to be a flashy demonstration; rather it is a
thorough test of many combinations of Gnuplot.py features.

"""

__cvs_version__ = '$Revision: 2.34 $'

import os, time, math, tempfile
import Numeric
from Numeric import NewAxis

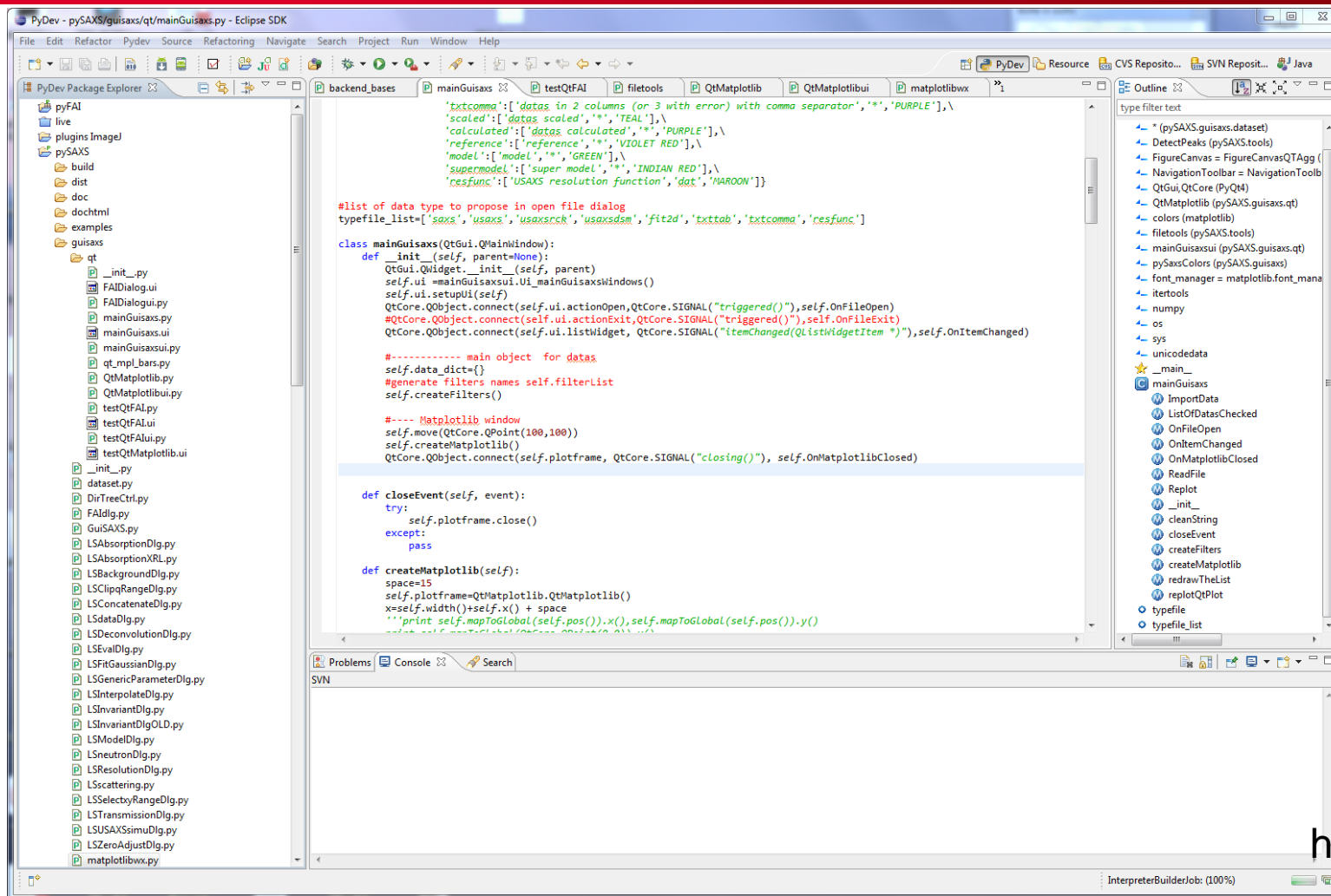
try:
    import Gnuplot, Gnuplot.PlotItems, Gnuplot.funcutils
except ImportError:
    # kludge in case Gnuplot hasn't been installed as a module yet:
    import __init__
    Gnuplot = __init__
    import PlotItems
    Gnuplot.PlotItems = PlotItems
    import funcutils
    Gnuplot.funcutils = funcutils

def wait(str=None, prompt='Press return to show results...\n'):
    if str is not None:
        print str
    raw_input(prompt)
```

3- direct execute

```
gnuplot> set terminal windows
gnuplot> set title "A simple example"
gnuplot> set data style linespoints
gnuplot> plot 'c:\docume\1\Nache\Locals\1\temp\tnp-mkwwf' notitle
Please press return to continue...
```

Python.exe myprogram.py



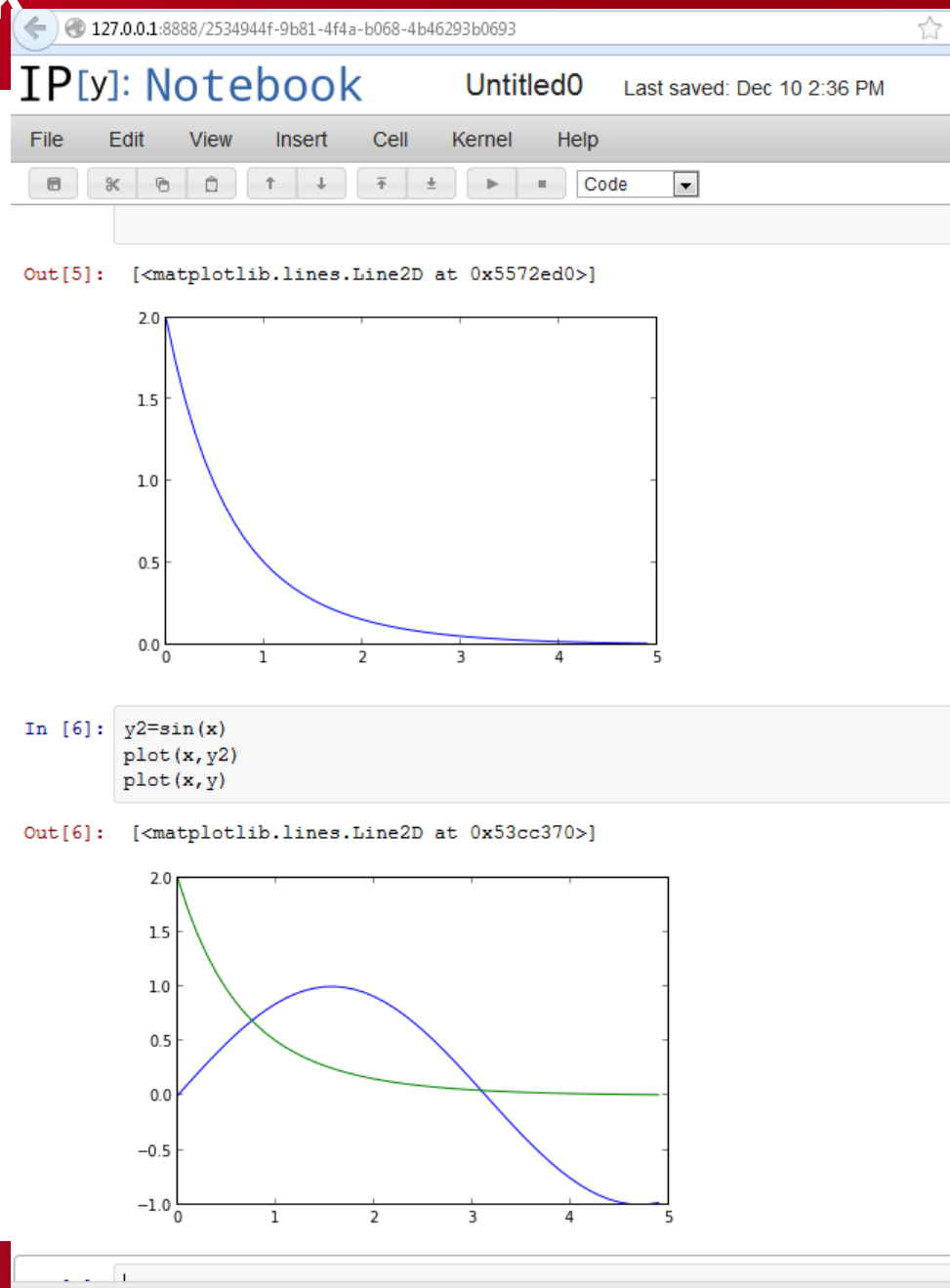
<http://pydev.org/>

What is PyDev?

PyDev is a **Python** IDE for **Eclipse**, which may be used in **Python** development.

IPYTHON NOTEBOOK

```
ipython notebook --pylab inline
```



add 2 values

```
>>> 1 + 1  
2
```

Setting a variable

```
>>> a = 1  
>>> a  
1
```

String

```
>>> s = "hello world"  
>>> print s  
hello world
```



Powerfull scientific calculator

Indentation is important:

```
If n in range(1,10):
```

```
    Print n
```



```
Print `fini`
```

```
def myfunction():
```

```
    moncode_ a_executer
```



```
    moncode_ a_executer
```

Documentation

■ Comments begins by #

■ Or ''' '''

and finished by

''' '''

VARIABLES ALLOCATION

No prior declaration of variables

How to allocate variable ?

```
>>>x=1
```

One variable refer to a memory area

```
>>>y=x
```

2 variables can refer to the same memory area

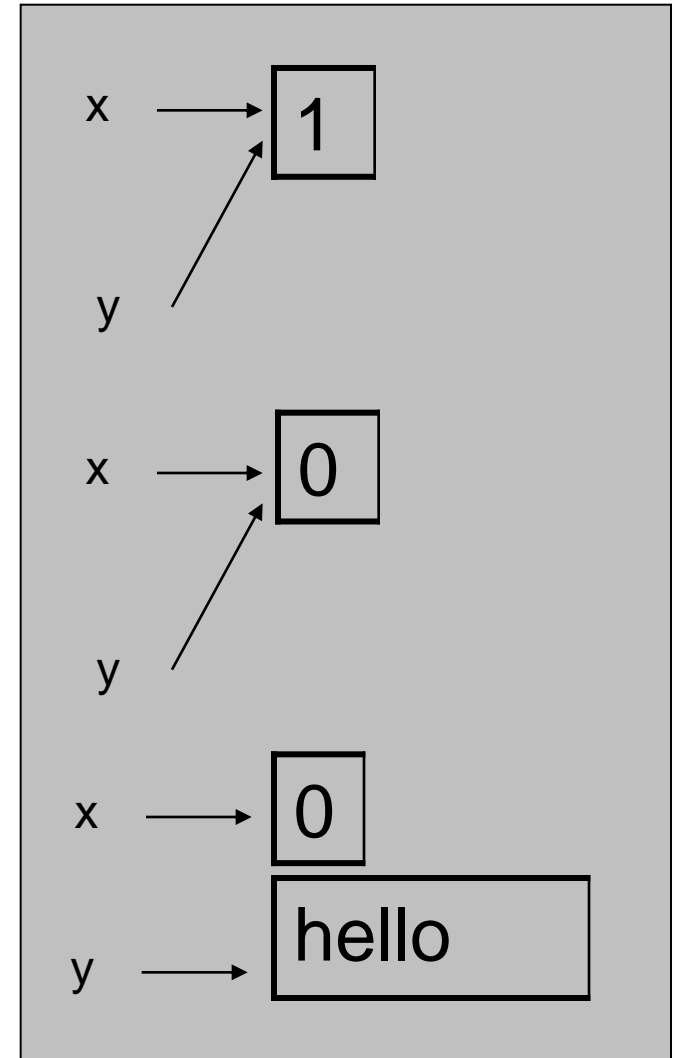
```
>>>x=0
```

```
>>>Print y
```

```
0
```

Allocate a variable :

```
>>>y=« hello »
```



Entier (int)

Réels (float)

`0, 1, 2, 3, -1, -2, -3``0., 3.1415926, -2.05e30, 1e-4`
(must contains . or exponent)

Complex

`1j, -2.5j, 3+4j`

Addition

`3+4, 42.+3, 1+0j`

substraction

`2-5, 3.-1, 3j-7.5`

Multiplication

`4*3, 2*3.14, 1j*3j`

Division

`1/3, 1./3., 5/3j`

power

`1.5**3, 2j**2, 2**-0.5`

Booleens

true ou false

```
'abc' ou 'abc'
'\n'
```

```
'abc'+'def'
3*'abc'
```

```
'abcdef'
'abcabcabc'
```

```
'ab cd e'.split()
```

```
['ab', 'cd', 'e']
```

```
'1,2,3'.split(',')
```

```
['1', ' 2', ' 3']
```

```
','.join(['1','2'])
'  a b c '.strip()
'text'.find('ex')
'Abc'.upper()
'Abc'.lower()
```

```
'1,2'
'a b c'
1
'ABC'
'abc'
```

Conversion to int or float: `int('2')`, `float('2.1')`

Conversion to string: `str(3)`, `str([1, 2, 3])`

FORMATING NUMBERS

(`%**[flag]****[width]****[.precision]****[specifier]**`):

flags:

- `-` : left justify
- `+` : Forces to precede result with + or -.
- `0` : Left pad the number with zeros instead of space (see width).

width:

Minimum number of characters to be printed. The value is not truncated if it has more characters.

precision:

- For integer specifiers (eg. `d,i,o,x`), the minimum number of digits.
- For `e, E` and `f` specifiers, the number of digits to print after the decimal point.
- For `g` and `G`, the maximum number of significant digits.
- For `s`, the maximum number of characters.

specifiers:

- `c` : character
- `d` or `i` : signed decimal integer
- `e` or `E` : scientific notation with `e` or `E`.
- `f` : decimal floating point
- `g, G` : use the shorter of `e, E` or `f`
- `o` : signed octal
- `s` : string of characters
- `u` : unsigned decimal integer
- `x, X` : unsigned hexadecimal integer

"strFormat" % number

```
>>> "%i" % (1000)
'1000'
>>> "%2.2f" % (1000)
'1000.00'
>>> "%i, %2.2f, %2.2e" % (1000, 1000, 1000)
'1000, 1000.00, 1.00e+03'
```

- **Creation**

```
>>>a=[1,2,3, 'blabla',[9,8]]
```

- **Concatenation (not addition... see array)**

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

- **Element add**

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

- **length**

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

- **range([start,] stop[, step]) -> list of integers**
Return a list containing an arithmetic progression of integers.

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

- **Simple Indexation**

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

- **Multiple Indexation**

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

- **element definition**

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

- **negatives index**

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

- **Part of list (liste[lower:upper])**

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

Index start always from 0

>>>HELP(LIST)

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(cmpfunc=None) -- stable sort *IN PLACE*; cmpfunc(x, y) -> -1, 0, 1

Usage :

```
>>>Myliste.methode (var)
```

Dictionnaires : Association from one value and a key.**Creation**

```
>>> dico = {}
>>> dico['C'] = 'carbon'
>>> dico['H'] = 'hydrogen'
>>> dico['O'] = 'oxygen'
>>> print dico
{'C': 'carbone', 'H': 'hydrogen', 'O': 'oxygen'}
```

Using

```
>>> print dico['C']
Carbone
```

A new one

```
>>> dico2={'N':'Nitrogen','Fe':'Iron'}
```

Concatenate

```
>>> dico3=dico+dico2
```

```
Traceback (most recent call last):
```

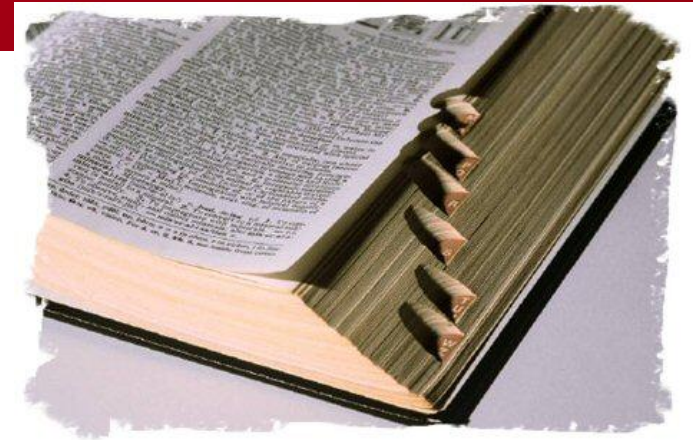
```
File "<pyshell#18>", line 1, in -toplevel-
  dico3=dico+dico2
```

```
TypeError: unsupported operand type(s) for +: 'dict' and 'dict'
```

```
>>> dico.update(dico2)
```

```
>>> dico
```

```
{'H': 'hydrogen', 'C': 'carbon', 'Fe': 'Iron', 'O': 'oxygen', 'N': 'Nitrogen '}
```



clear(...)

D.clear() -> None. Remove all items from D.

copy(...)

D.copy() -> a shallow copy of D

get(...)

D.get(k[,d]) -> D[k] if k in D, else d. d defaults to None.

has_key(...)

D.has_key(k) -> True if D has a key k, else False |

items(...)

D.items() -> list of D's (key, value) pairs, as 2-tuples

iteritems(...)

D.iteritems() -> an iterator over the (key, value) items of D

iterkeys(...)

D.iterkeys() -> an iterator over the keys of D

itervalues(...)

D.itervalues() -> an iterator over the values of D

keys(...)

D.keys() -> list of D's keys

pop(...)

D.pop(k[,d]) -> v, remove specified key and return the corresponding value

If key is not found, d is returned if given, otherwise
KeyError is raised

popitem(...)

D.popitem() -> (k, v), remove and return some (key, value)
pair as a

2-tuple; but raise KeyError if D is empty

setdefault(...)

D.setdefault(k[,d]) -> D.get(k,d), also set D[k]=d if k not in D

update(...)

D.update(E) -> None. Update D from E: for k in E.keys():
D[k] = E[k]

values(...)

D.values() -> list of D's values

if <condition>:

→ |<code>

elif <condition>:

→ |<code>

else:

→ |<code>

Examples :

```
test=True
if test:
    print 'vrai'
else:
    print 'faux'
->vrai
```

```
if i==1:
    print 'A'
elif i==2:
    print 'B'
elif i==3:
    print 'C'
else:
    print "?"
```

for <variable> **in** <list>:
 → | <code>

```
>>> for i in range(5):
    print i,
0 1 2 3 4
```

```
>>> for i in 'abcdef':
    print i,
a b c d e f
```

```
l=['C','H','O','N']
>>> for i in l:
    print i,
C H O N
```

```
>>> for i in dico:
    print i,
```

H C Fe O N

```
>>> for i in dico:
    print dico[i],
```

hydrogen carbon Iron oxygen
 Nitrogen

while <condition>:
 → | <instructions>

```
>>> test=0
>>> while test<10:
    test=test+1
    print test,
```

1 2 3 4 5 6 7 8 9 10

```
>>> i = 0
>>> while 1:
    if i<3 :
        print i,
    else :
        break
    i=i+1
```

0 1 2

READING TEXT FILES

How to read a text file:

- Open a text file
- Read line by line
- One line set to a variable
- Close the file

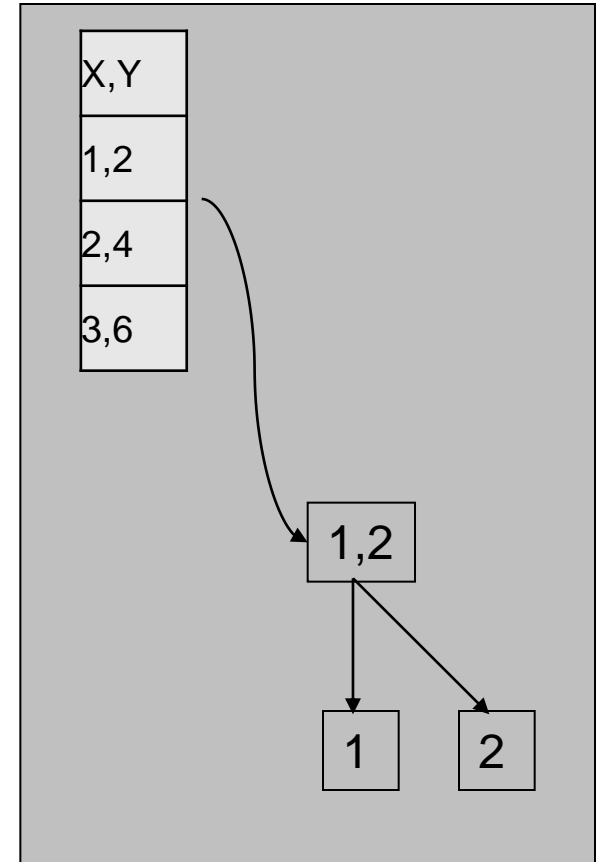
Python :

```
>>>f=open('filename.txt','r')
>>>lines=f.readlines()
>>>f.close()
readlines read all the lines and return a list.
```

Each element must be treated.

In the file there is 2 columns separated by a ,

```
>>>x=[ ] #list
>>>y=[ ]
>>>for s in lines[1:]:We don't want the first line
                        element=s.split(',')
                        x.append(float(element[0]))
                        y.append(float(element[1]))
```



How to write a text file

- Open the file (r, a, w)
- Write line by line
- Close the file

Python :

```
>>>f=open('filename.txt','w')  
>>>f.write('line1\n')  
>>>f.write('line2\n')  
>>>f.close()
```

NUMPY LOADTXT

numpy.loadtxt(*fname*, *dtype*=<type 'float'>, *comments*='#', *delimiter*=None, *converters*=None, *skiprows*=0, *usecols*=None, *unpack*=False) Load data from a text file.

Each row in the text file must have the same number of values.

Parameters:

fname : file or str

File or filename to read. If the filename extension is .gz or .bz2, the file is first decompressed.

dtype : dtype, optional

Data type of the resulting array. If this is a record data-type, the resulting array will be 1-dimensional, and each row will be interpreted as an element of the array. In this case, the number of columns used must match the number of fields in the data-type.

comments : str, optional

The character used to indicate the start of a comment.

delimiter : str, optional

The string used to separate values. By default, this is any whitespace.

converters : dict, optional

A dictionary mapping column number to a function that will convert that column to a float. E.g., if column 0 is a date string: `converters = {0: datestr2num}`. Converters can also be used to provide a default value for missing data: `converters = {3: lambda s: float(s or 0)}`.

skiprows : int, optional

Skip the first *skiprows* lines.

usecols : sequence, optional

Which columns to read, with 0 being the first. For example, `usecols = (1,4,5)` will extract the 2nd, 5th and 6th columns.

unpack : bool, optional

If True, the returned array is transposed, so that arguments may be unpacked using `x, y, z = loadtxt(...)`. Default is False.

Returns: out : ndarray

Data read from the text file.

savetxt(fname, X, fmt='%.18e', delimiter=' ', newline='\n')

Save an array to a text file.

Parameters

fname : filename or file handle

If the filename ends in ``.gz``, the file is automatically saved in compressed gzip format. `loadtxt` understands gzipped files transparently.

X : array_like

Data to be saved to a text file.

fmt : str or sequence of str

A single format (%10.5f), a sequence of formats, or a multi-format string, e.g. 'Iteration %d -- %10.5f', in which case `delimiter` is ignored.

delimiter : str Character separating columns.

newline : str Character separating lines.

Examples

```
>>> x = y = z = np.arange(0.0, 5.0, 1.0)
>>> np.savetxt('test.out', x, delimiter=',')    # X is an array
>>> np.savetxt('test.out', (x,y,z))    # x,y,z equal sized 1D arrays
>>> np.savetxt('test.out', x, fmt='%1.4e')    # use exponential notation
```

Syntax :

```
def nameOfFunction(arguments):  
    → | instructions  
      | return something # not mandatory
```

```
>>> def addition(a,b):  
        c=a+b  
        return c  
  
>>> addition(2,3)  
5  
>>>  
addition('name','firstname')  
'namefirstname'  
>>> addition(1.25,5.36)  
6.6100000000000003
```


FUNCTIONS : FLEXIBILITY

```
def addition(a,b):
    → return a+b
```

```
>>> addition(1,2)
3
```

```
def addition(a,b=1):
    return a+b
```

```
>>> addition(1)
2
```

```
def addition(a,b=1,c=1):
    return a+b+c
```

```
>>> addition(1,c=2,b=3)
6
```

```
def addition(a,b):
    return a+b
```

```
>>> addition('hello',' world')
'hello world'
```

```
def addition(a,b):
    '''
    add a and b
    '''
    return a+b
```

```
>>> addition(
```

(a,b)
add a and b

```
def addition(a,b):
    return a+b
```

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

Create its own module:

```
operations.py :
def addition(a,b):
    c=a+b
    return c
```

Using

```
>>>import operations
>>> operations.addition(1,2)
3
```

Reload a module

```
>>>reload(operations)
```

Using some part

```
>>>from operations import addition
addition(1,2)
```

Using every compounds

```
>>>from operations import *
```

```
>>>import numpy
>>> numpy.sin(3.14)
0.0015926529164868282
```

Cleaner because we
know what is it used

Or

```
>>>import numpy as N
>>>N.sin(1)
```

Where are the modules ?

- Windows:
 - C:\Python27
 - where it is executed
 - C:\Python27\Lib\site-packages
- Linux : ???
- How to know ?
 - Module.__path__
 - Module.__file__

How to know the locals objects ?

```
>>>locals()
```

Update and return a dictionary
containing the current scope's
local variables.

TESTS AND DOCUMENTATION (IN THE CODE)

- An easy way to make test with code is :

```
# add this code at the end
if __name__ == '__main__':
    test()
```

```
# add this code at the end
if __name__ == '__main__':
    print sys.argv[1:] #print the args from
                        the command line
```

Will execute this code if the module is run as « main »

- An easy way to make a documentation is to use **docstrings**

```
"""
a set of usefull function for learning python
"""

def addition(a,b):
    """
    add a and b
    """
    return a+b

def substraction(a,b):
    """
    sub a and b
    """
    return a-b

# add this code at the end
if __name__ == '__main__':
    print "1+1=",addition(1,1)
    print "2-1=",substraction(2,1)
```

```
>>> import operations
>>> help(operations)
Help on module operations:

NAME
    operations - a set of usefull function for learning python

FILE
    s:\présentations\formation python\operations.py

FUNCTIONS
    addition(a, b)
        add a and b

    substraction(a, b)
        sub a and b
```

USER INTERACTIONS, EVAL

```
>>> ret=raw_input("How old are you ?")
How old are you ?18

>>> formula=raw_input("Enter the formula : ")
Enter the formula : sin(x)*2

>>> x=float(raw_input("Enter x : "))
Enter x : 3.14

>>> eval(formula)
0.0031853058329736565
```

Raw_input : Read a string from standard input

Eval : evaluate the source

How to catch exceptions (errors, or problems raised) ?

```
try:
    code
except [exception name] :
    code
else:
    code if no exceptions are raised
finally:
    always run on the way out
```

```
'''
tools package for numbers
'''
def isNumeric(val):
    ''' return true if val is a numeric value
    '''
    try:
        i = float(val)
    except ValueError, TypeError:
        # not numeric
        return False
    else:
        # numeric
        return True
```

```
>>> isNumeric(3)
True
>>> isNumeric('3')
True
>>> isNumeric('3.0')
True
>>> isNumeric('3.0x')
False
```

Gnuplot

Matplotlib

Plotting

Scipy

Scientific functions

Numpy

Numeric arrays

Python

Programming Language

Different from a list

All the objects are numerics (floats or int) and identicals

Creation :

```
>>> array([1,2,3,4])
array([1, 2, 3, 4])
```

```
>>> array([1,2,3,4.])
array([ 1.,  2.,  3.,  4.])
```

```
>>> arange(0,20,2)
array([ 0,  2,  4,  6,  8, 10,
       12, 14, 16, 18])
```

From a list

```
>>> l=range(1,10)
>>> a=array(l)
>>> a
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

- Addition

```
>>> t=array([1,2,3,4.])
>>> t2=array([1,2,3,4.])
>>> t+t2
array([ 2.,  4.,  6.,  8.])
```

- Using mathematical functions

```
>>> t=arange(0,10,2)
>>> t
array([0, 2, 4, 6, 8])
>>> t*pi
array([0., 6.28318531, 12.56637,
       18.84955592, 25.13274123])
```

Arrays with multiple dimensions

```
>>> t=arange(0,15)
>>> t.shape
(15,)
>>> array2d=reshape(t, (3,5))
>>> array2d
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
```

Data access

```
>>> array2d[0,3]
3
```

Warning tableau[y,x] !!!

Parts

```
>>> array2d[:,3]
array([ 3,  8, 13])
```


ARRAY FUNCTIONS

Functions

- array
- zeros
- shape
- rank
- size
- fromstring
- take
- put
- putmask
- reshape
- repeat
- choose
- cross_correlate
- searchsorted
- sum
- average
- cumsum
- product
- cumproduct
- alltrue
- sometrue
- allclose
- NumPy Array construction
- Return an array of all zeros
- Return shape of sequence or array
- Return number of dimensions
- Return number of elements in entire array or a certain dimension
- Construct array from (byte) string
- Select sub-arrays using sequence of indices
- Set sub-arrays using sequence of 1-D indices
- Set portion of arrays using a mask
- Return array with new shape
- Repeat elements of array
- Construct new array from indexed array tuple
- Correlate two 1-d arrays
- Search for element in 1-d array
- Total sum over a specified dimension
- Average, possibly weighted, over axis or array.
- Cumulative sum over a specified dimension
- Total product over a specified dimension
- Cumulative product over a specified dimension
- Logical and over an entire axis
- Logical or over an entire axis
- Tests if sequences are essentially equal

More Functions:

- arange (arange)
- asarray
- sarray
- convolve
- swapaxes
- concatenate
- transpose
- sort
- argsort
- argmax
- argmin
- innerproduct
- dot
- outerproduct
- resize
- indices
- fromfunction
- diagonal
- trace
- dump
- dumps
- load
- loads
- ravel
- nonzero
- shape
- where
- compress
- clip
- zeros
- ones
- identity
- Return regularly spaced array
- Guarantee NumPy array
- Guarantee a NumPy array that keeps precision
- Convolve two 1-d arrays
- Exchange axes
- Join arrays together
- Permute axes
- Sort elements of array
- Indices of sorted array
- Index of largest value
- Index of smallest value
- Innerproduct of two arrays
- Dot product (matrix multiplication)
- Outerproduct of two arrays
- Return array with arbitrary new shape
- Tuple of indices
- Construct array from universal function
- Return diagonal array
- Trace of array
- Dump array to file object (pickle)
- Return pickled string representing data
- Return array stored in file object
- Return array from pickled string
- Return array as 1-D
- Indices of nonzero elements for 1-D array
- Shape of array
- Construct array from binary result
- Elements of array where condition is true
- Clip array between two values
- Array of all zeros
- Array of all ones
- 2-D identity array (matrix)

help(numpy)

help(function)

SCIPY FUNCTIONS

Gnuplot

Matplotlib

Scipy

Numpy

Python

[Cluster](#) : vector quantization / kmeans

[Fftpack](#) : discrete fourier transform algorithms

[Integrate](#) : integration routines and differential equation solvers

[Interpolate](#) : interpolation tools

[Linalg](#) : linear algebra routines

[Misc](#) : various utilities (including Python Imaging Library)

[Ndimimage](#) : n-dimensional image tools

[Optimize](#) : optimization tools

[Signal](#) : signal processing tools

[Sparse](#) : sparse matrices

[Stats](#) : statistical functions

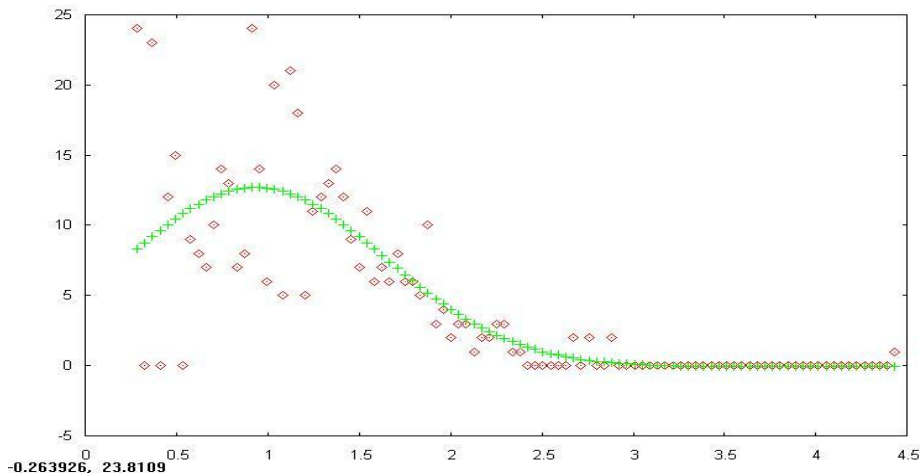
SCIPY : EXAMPLE

```

from scipy import *
From numpy import *
import Gnuplot
g=Gnuplot.Gnuplot()

d=loadtxt("c:/datas/results.txt",separator="\t")
y=d[1]
rayons=(y/pi)**.5
h=numpy.histogram(rayons,bins=100)
hx=h[1]
hy=h[0]
curveh=Gnuplot.Data(hx,hy)
g.plot(curveh)

```



	Area	Mean	Min	Max	IntDen	%Area
1	2.755	255	255	255	702.551	100
2	0.184	255	255	255	46.837	100
3	22.408	252.91	0	255	5667.245	99.18
4	6.245	255	255	255	1592.449	100
5	3.673	255	255	255	936.735	100
6	2.755	255	255	255	702.551	100
7	6.245	255	255	255	1592.449	100
8	13.224	255	255	255	3372.245	100
9	0.551	255	255	255	140.51	100
10	14.878	255	255	255	3793.776	100
11	0.184	255	255	255	46.837	100
12	8.816	255	255	255	2248.163	100
13	5.51	255	255	255	1405.102	100
14	2.571	255	255	255	655.714	100
15	2.571	255	255	255	655.714	100
16	3.857	255	255	255	983.571	100
17	6.612	255	255	255	1686.122	100
18	5.694	255	255	255	1451.939	100
19	4.224	255	255	255	1077.245	100
20	3.306	255	255	255	843.061	100
21	0.184	255	255	255	46.837	100
22	4.224	255	255	255	1077.245	100

GNU PLOT INTRODUCTION

Gnuplot is a portable command-line driven graphing utility

Gnuplot

Matplotlib

Scipy

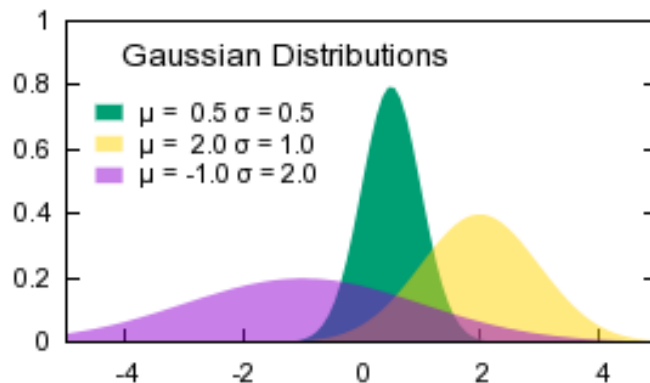
Numpy

Python

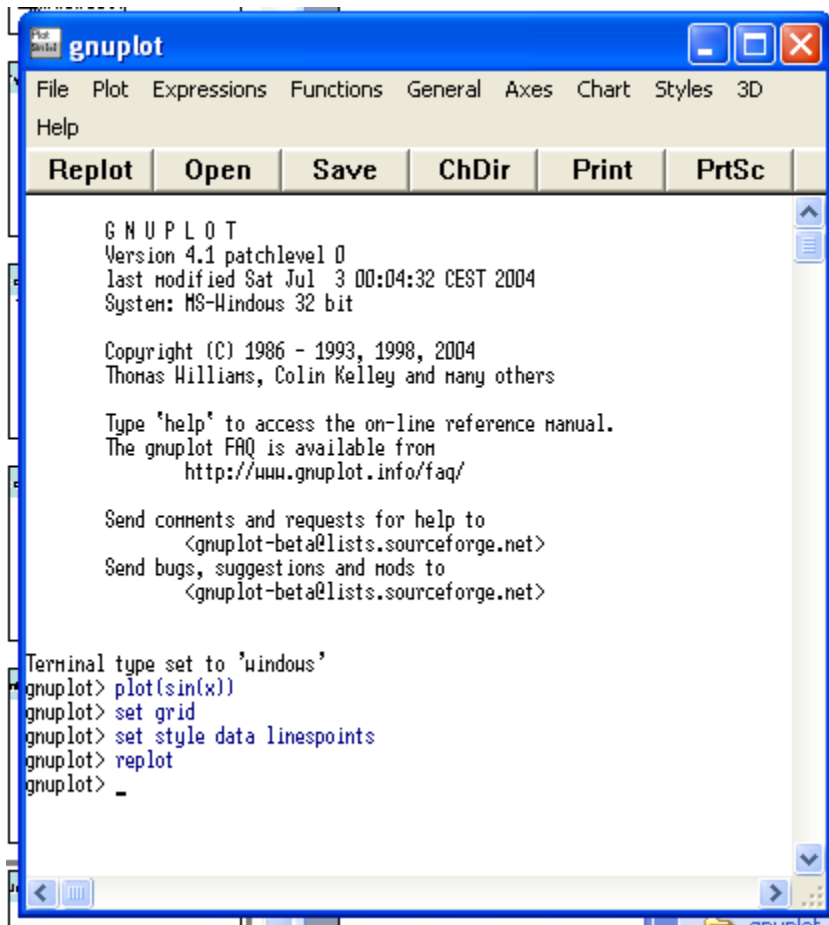
Gnuplot is not a Python software

- Gnuplot is a software that can plot correctly datas
- Command line → not very user friendly
- Load datas from text file
- Can automatize data plotting
- many different types of 2D and 3D plots
- Free and Open Source
- cross-platform(windows, linux, mac)

■ Python can interface Gnuplot → Gnuplot do the plotting work



USING GNUPLOT



```

gnuplot
File Plot Expressions Functions General Axes Chart Styles 3D
Help
Replot Open Save ChDir Print PrtSc

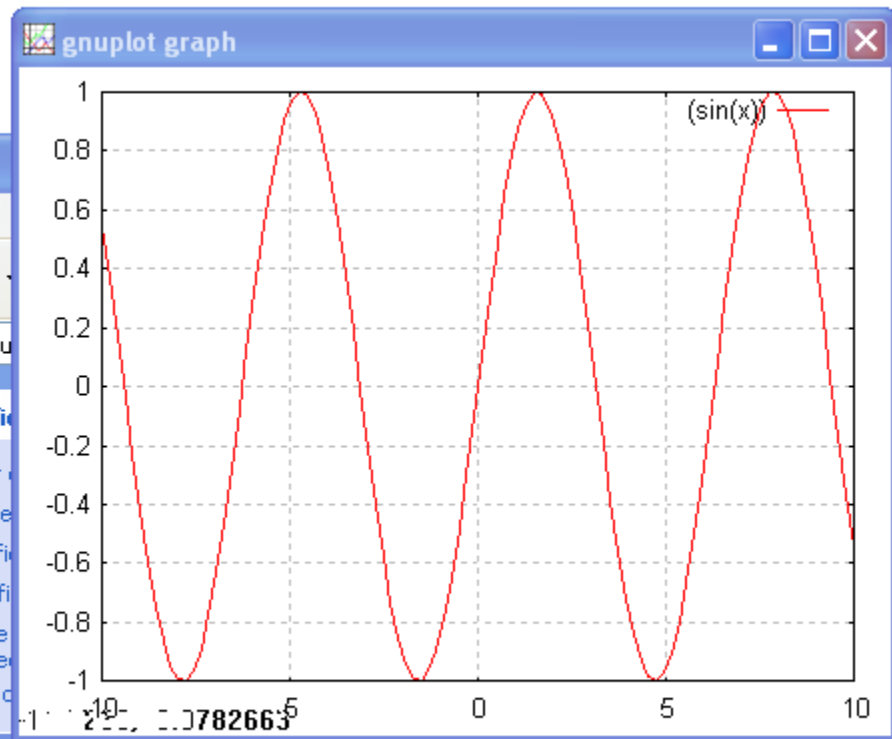
GNU PLOT
Version 4.1 patchlevel 0
last modified Sat Jul 3 00:04:32 CEST 2004
System: MS-Windows 32 bit

Copyright (C) 1986 - 1993, 1998, 2004
Thomas Williams, Colin Kelley and many others

Type 'help' to access the on-line reference manual.
The gnuplot FAQ is available from
  http://www.gnuplot.info/faq/

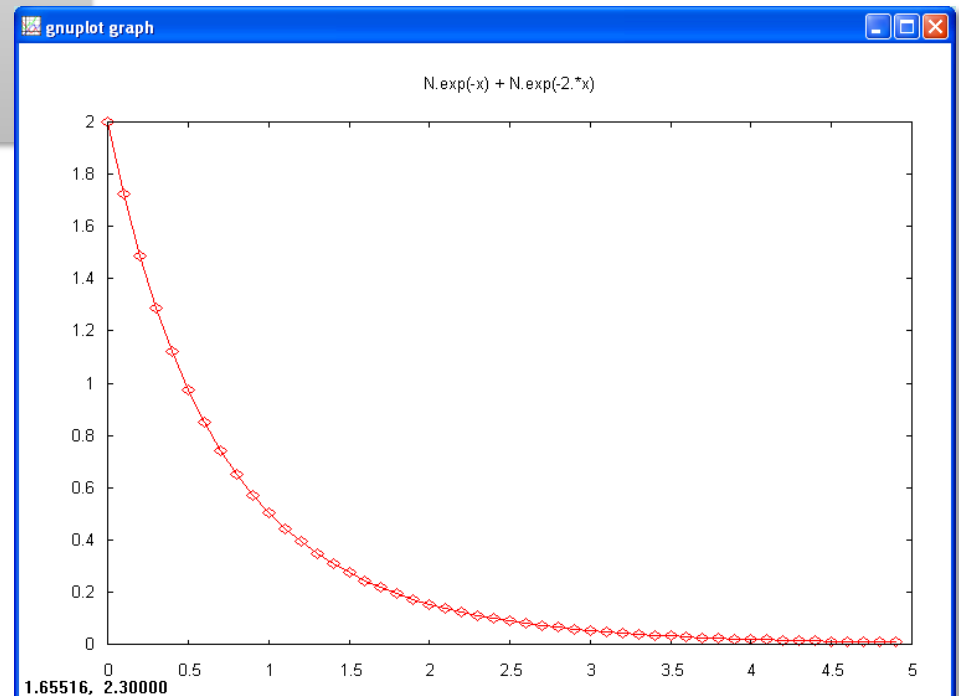
Send comments and requests for help to
  <gnuplot-beta@lists.sourceforge.net>
Send bugs, suggestions and mods to
  <gnuplot-beta@lists.sourceforge.net>

Terminal type set to 'windows'
gnuplot> plot(sin(x))
gnuplot> set grid
gnuplot> set style data linespoints
gnuplot> replot
gnuplot> _
  
```



GNU PLOT AND PYTHON

```
import Gnuplot
import numpy
# Initialize Gnuplot
g = Gnuplot.Gnuplot()
# Set up data
x = numpy.arange(0., 5., 0.1)
y = numpy.exp(-x) + numpy.exp(-2.*x)
curve=Gnuplot.Data(x, y,with='linespoints')
g.title('numpy.exp(-x) + numpy.exp(-2.*x)')
g.plot(curve)
```



Méthodes :

plot : clear the old plot and old **PlotItems**, then plot the arguments in a fresh plot command. Arguments can be: a **PlotItem**, which is plotted along with its internal options; a string, which is plotted as a 'Func'; or anything else, which is plotted as a Data.

Splot : like plot, except for 3-d plots.

Hardcopy : replot the plot to a postscript file (if filename argument is specified) or pipe it to the printer as postscript otherwise. If the option color is set to true, then output color postscript.

Replot : replot the old items, adding any arguments as additional items as in the plot method.

Refresh : issue (or reissue) the plot command using the current **PlotItems**.

xlabel, ylabel, title : set corresponding plot attribute.

Load : load a file (using the gnuplot load command).

Save : save gnuplot commands to a file (using gnuplot save command) If any of the 'PlotItem's is a temporary file, it will be deleted at the usual time and the save file will be pretty useless :-).

Clear : clear the plot window (but not the itemlist).

Reset : reset all gnuplot settings to their defaults and clear the current itemlist.

set_string : set or unset a gnuplot option whose value is a string.

_clear_queue : clear the current **PlotItem** list.

_add_to_queue : add the specified items to the current **PlotItem** list.

__call__ : pass an arbitrary string to the gnuplot process, followed by a newline.

```
g("replot")
```

```
g.Itemlist
g.itemlist.append(...)
```

Itemlist : liste des **PlotItems**

PlotItems : éléments that can be plot (data, file, grid)

Data (*set, *keyw)

Create and return a `_FileItem` representing the data from `*set`.

```
curve=Gnuplot.Data(y)
curve=Gnuplot.Data(x,y,sigma)
curve=Gnuplot.Data(x, y,with='linespoints')
d=transpose([x,y])
curve=Gnuplot.Data(d)
```

File (filename, **keyw)

Construct a `_FileItem` object referring to an existing file.

```
Gnuplot.File(filename1)
```

GridData (data, xvals=None, yvals=None, inline=_unset, **keyw,)

GridData represents a function that has been tabulated on a rectangular grid. The data are written to a file; no copy is kept in memory.

Data : the data to plot: a 2-d array with dimensions (numx,numy).

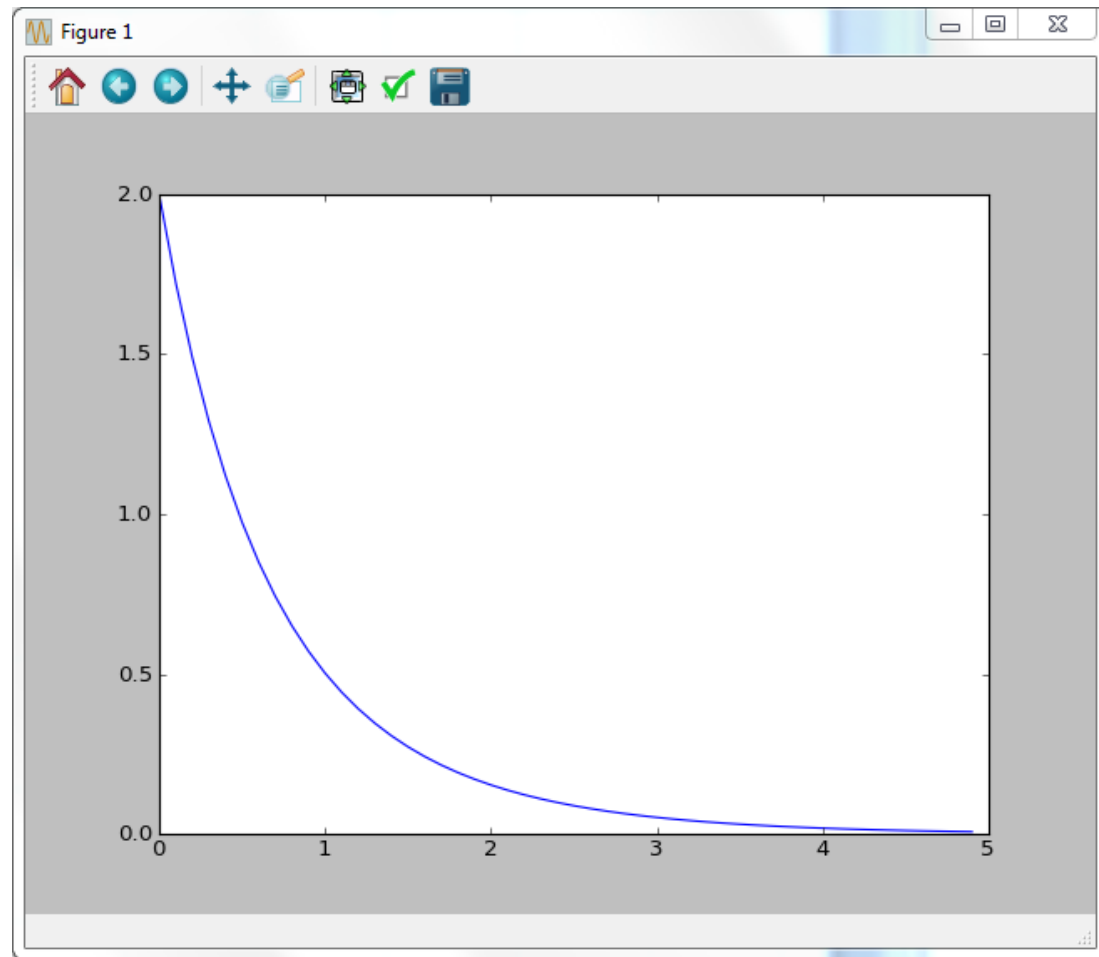
Xvals : a 1-d array with dimension numx

Yvals : a 1-d array with dimension numy

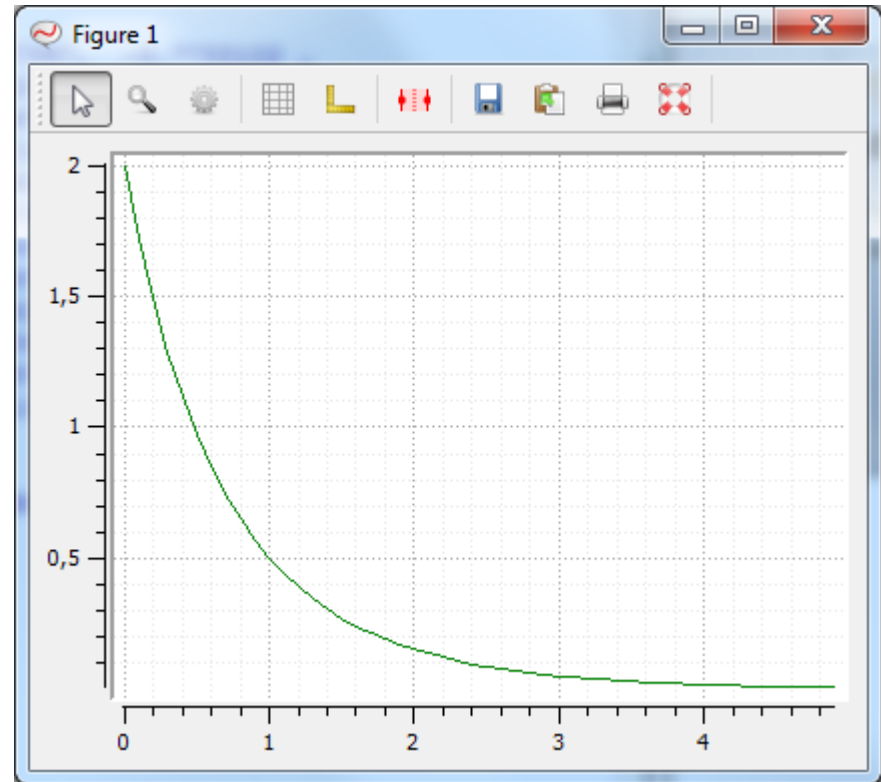
`python/Lib/site-packages/Gnuplot test.py`


```
from numpy import *  
from pylab import *  
x = arange(0., 5.,  
0.1)  
y=exp(-x)+exp(-2*x)  
plot(x,y)  
show()
```

<http://matplotlib.sourceforge.net>



```
from numpy import *  
from guiqwt.pyplot import *  
x = arange(0., 5., 0.1)  
y = exp(-x) + exp(-2*x)  
plot(x, y)  
show()
```



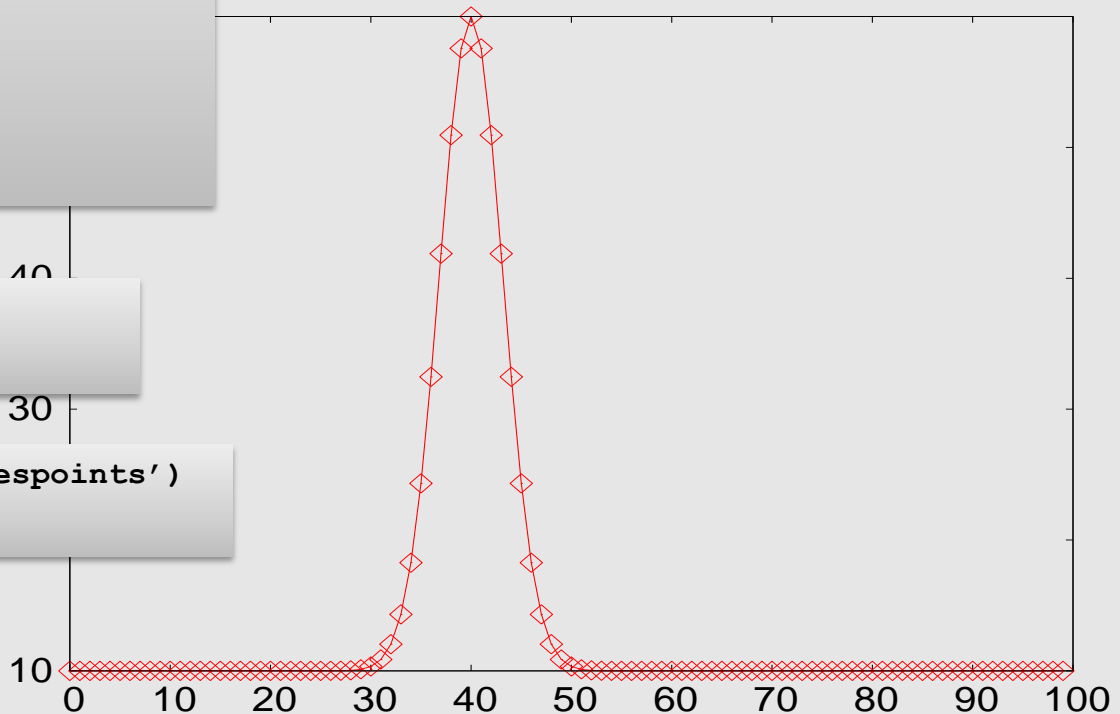
```
def Gaussian(x,par):
    """
    Gaussian model to fit the peak to get
    exact zero position
    par[0] : height of gaussian
    par[1] : is related to the FWHM
    par[2] : center of gaussian
    par[3] : background
    """
    return par[0]*exp(-((x-
    par[2])**2)/par[1]**2)+par[3]
```

```
>>> x=arange(0,100,1)
```

```
>>> y=gaussian((50,0.05,40,10),x)
```

```
>>> curve=Gnuplot.Data(x,y,with='linespoints')
```

```
>>> g.plot(curve)
```



HOW TO FIT ?

```

From numpy import *

def Gaussian(x,par):
    """
        Gaussian model to fit the peak to get
        exact zero position
        par[0] : height of gaussian
        par[1] : is related to the FWHM
        par[2] : center of gaussian
        par[3] : background
    """
    return (par[0]-par[3])*numpy.exp(-(x-
    par[2])**2)/par[1]**2)+par[3]

```

Experimentals datas

```

>>> x=arange(0,100.0,1)
>>> y=gaussian([50,5.0,40,10],x)

```

Minimization function

```

def residuals(r,x,y):
    """
        calculate difference between datas (x) and
        model (Gaussian(r))
    """
    print r
    err=(y-Gaussian(x,r))
    return err

```

```

#fit....
>>>p0=[52,5.0,35,10.1] #initial parameters
>>>fit,succes=optimize.leastsq(residuals,p0,args
=(x,y)) #fit

```

```

import numpy
import Gnuplot
from scipy import optimize

def GaussianFunction(x,par):
    """
    Gaussian model to fit the peak to get exact zero position
    par[0] : height of gaussian
    par[1] : is related to the FWHM
    par[2] : center of gaussian
    par[3] : background
    """
    return (par[0]-par[3])*numpy.exp(-(x-par[2])**2/par[1]**2)+par[3]

def residuals(r,x,y):
    """
    calculate difference between datas (x) and model (Gaussian(r))
    """
    print r
    err=(y-GaussianFunction(x,r))
    return err

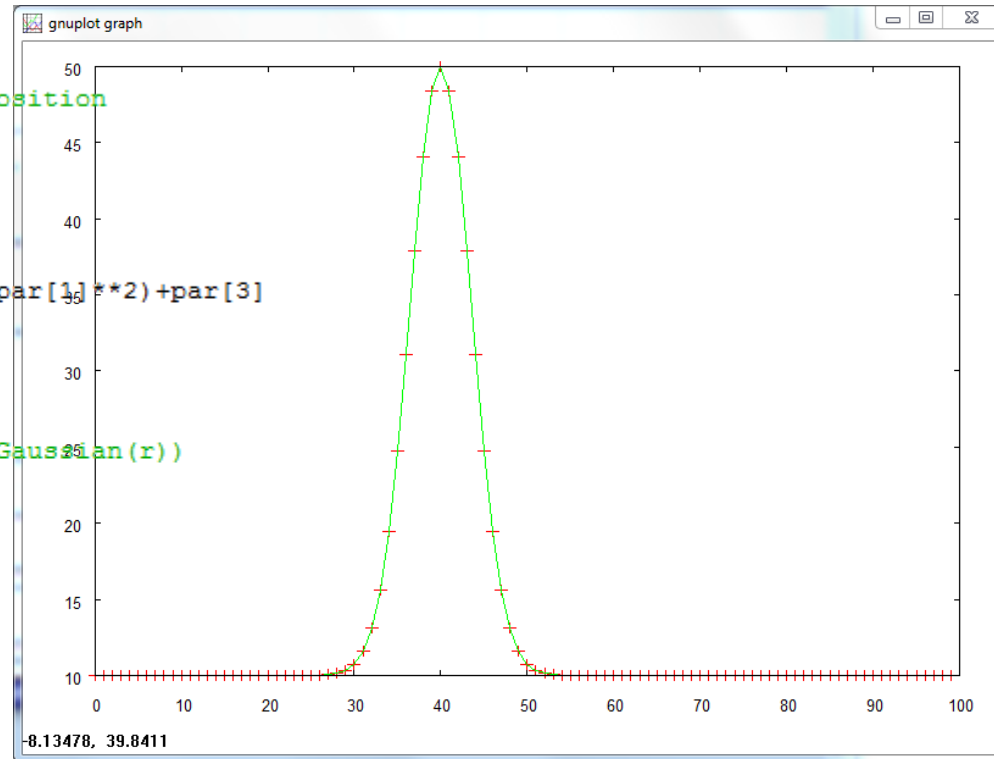
x=numpy.arange(0,100.0,1)
y=GaussianFunction(x,[50.0,5,40,10])

g=Gnuplot.Gnuplot()
curve=Gnuplot.Data(x,y,with_='linespoints')
g.plot(curve)

p0=[52,5.0,35,10.1] #initial parameters
fit,succes=optimize.leastsq(residuals,p0,args=(x,y)) #fit
print fit

curveFit=Gnuplot.Data(x,GaussianFunction(x,fit),with_='lines')
g.plot(curve,curveFit) #plot multiple curve

```



```

from numpy import *
#from scipy import *
from scipy import optimize
import Gnuplot

def Gaussian(x,par):
    """
    Gaussian model to fit the peak to get exact zero position
    par[0] : height of gaussian
    par[1] : is related to the FWHM
    par[2] : center of gaussian
    par[3] : background
    """
    return par[0]*exp(-(x-par[2])**2/par[1]**2)+par[3]

def residuals(par, x, y):
    #err = (y-Gaussian(x,par))
    err = (y-Gaussian(x,par))
    return err

def rnd(start,end):
    return random.rand()*(end-start)+start

#generate datas
p=[rnd(0,100),rnd(3,6),rnd(1,6),rnd(0,10)]
x=arange(0,10,0.1)
y=Gaussian(x,p)
percent=0.1
r=y+(y*percent)*(random.rand(size(y))-1/2.)
#print y
pfit=[max(y),1,5,min(y)]#initials parameters
fitParam,succes = optimize.leastsq(residuals,pfit,args=(x,r))
print fitParam, succes

#plotting
g=Gnuplot.Gnuplot()
curve=Gnuplot.Data(x,y,with='lines')
curve2=Gnuplot.Data(x,r,with='points')
curvefit=Gnuplot.Data(x,Gaussian(x,fitParam),with='linespoints')
g.plot(curve,curve2,curvefit)

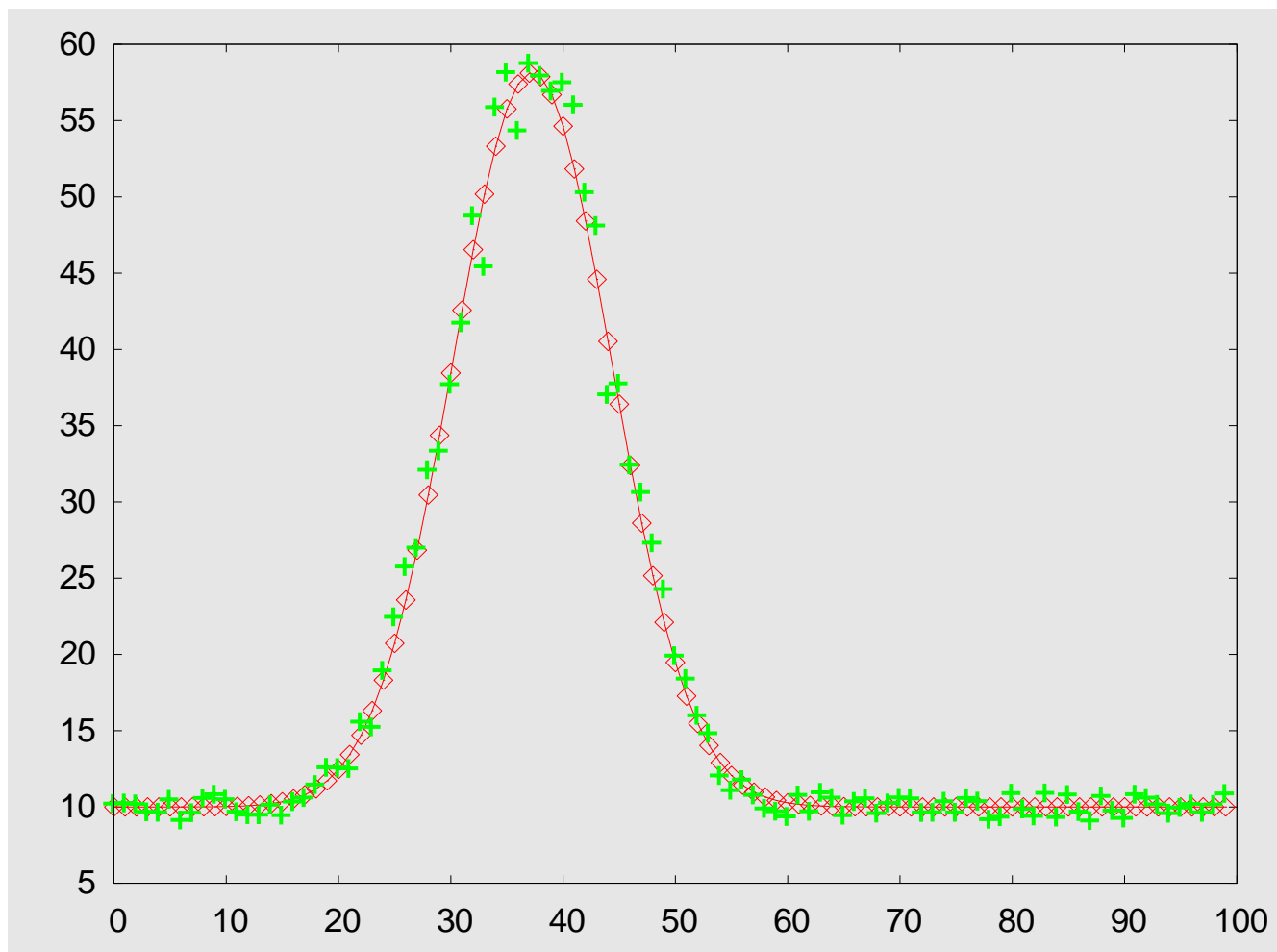
```

Creation of random datas

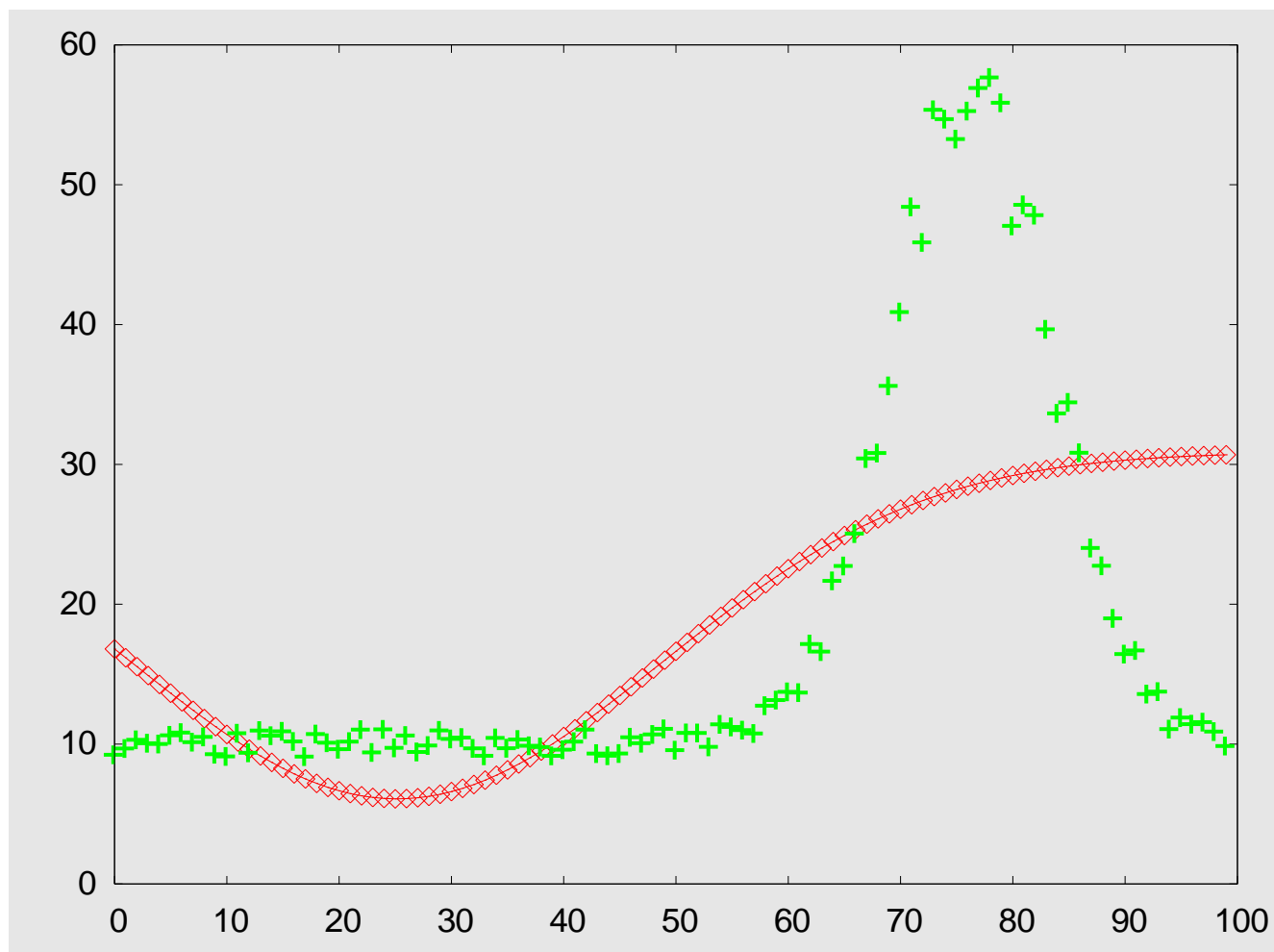
Fit

Plot

FIT : RESULT



FIT : RESULTS ?!



Object have some things in common

Class

- Datas (variables)
- Constructor (variables initialization)
- Methods (fonctions)

```
class OurClass(object):
    """Class docstring."""
    arg1=0#not necessary
    arg2=0#not necessary
    def __init__(self, arg1, arg2):
        """Method docstring."""
        self.arg1 = arg1
        self.arg2 = arg2
    def printargs(self):
        """Method docstring."""
        print self.arg1
        print self.arg2
```

Using :

```
>>>P=OurClass('one',two')
>>>P.printargs()
one
Two
```

```
>>>'ab cd e'.split() -> ['ab', 'cd', 'e']
```



It is an object

OBJECTS : EXAMPLE

```
# author : Olivier Taché
```

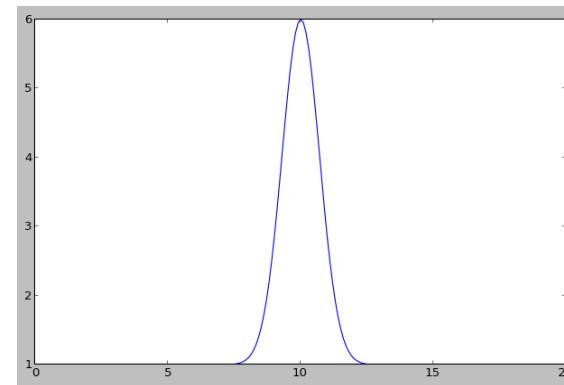
```
"""
gaussian class
"""

class gaussian:
    def __init__(self,height=10,fwmh=1,center=10,offset=1.0):
        """
        Gaussian model to fit the peak to get exact zero position
        height of gaussian
        is related to the FWHM
        center of gaussian
        background
        """
        self.height=height
        self.fwmh=fwmh
        self.center=center
        self.offset=offset

    def calculate(self,x):
        return self.height*exp(-((x-self.center)**2)/self.fwmh**2)+self.offset

    def getFormula(self):
        return "height*exp(-((x-center)**2)/fwmh**2)+offset"

if __name__=='__main__':
    import numpy
    from pylab import *
    x=numpy.arange(0,20,0.1)
    #define Gaussian
    g=gaussian()
    #set a parameter
    g.height=5.0
    y=g.calculate(x)
    #plot
    plot(x,y)
    show()
```



EXAMPLE : CLASS « MODEL » WITH INHERITANCE

```
Class model:
    #here list of datas
    author=«  »
    description=«  »
    modelFunction=None

    def __init__(self)
        #initialization

    def fit(self):
        #here a fitting routine
        code

    def fitWithBounds(self):
        #here another routine
        autre code

    ... .
```

```
Class myModel(model):
    #
    author=« olivier »
    description=« mymodele »
    modelFunction=myfunction

    def myfunction(self):
        #,,,
```

Classe model

Classe myModel

```
>>> mod=myModel()
>>> mod.fit()
```

INHERITANCE AND FIT MODEL

```
import numpy
from scipy import optimize
```

```
class model:
    author="me"
    description=""
    parameters=None
    def __init__(self):
        pass

    def modelFunction(self,par,x):
        #need to be redefined in child class
        y=x
        return y

    def residuals(self,par,x,y):
        '''
        calculate difference between datas (x) and model
        '''
        print par
        err=(y-self.modelFunction(par,x))
        return err

    def fit(self,p0,x,y):
        '''
        fit by model
        p0 : initials parameters for fit
        x and y are values to be fitted
        '''
        fitvalues,success=optimize.leastsq(self.residuals,p0,args=(x,y))
        return fitvalues,success

    def evaluate(self,par,x):
        return self.modelFunction(par,x)
```

Base model

```
class modelGaussian(model):
    def GaussianFunction(self,par,x):
        '''
        Gaussian model to fit the peak to
        get exact zero position
        '''
        return (par[0]-par[3])*numpy.exp(-((x-par[2])**2))

    '''
    parameters definition
    '''

    modelFunction=GaussianFunction #function
    author="Olivier Tache"
    description="example of class for fit by gaussian m
```

```
if __name__=='__main__':
    g=modelGaussian()
    #-----
    x=numpy.arange(0,100.0,1)
    y=g.evaluate([50.0,5,40,10],x)
    #-----
    values,success=g.fit([40.0,3,42,11],x,y)
```

www.python.org

Numpy (<http://numeric.scipy.org/>)

www.SciPy.org

www.gnuplot.info

Python xy www.pythonxy.com

Python.developpez.com

Transparents inspirés de “Introduction to Scientific Computing with Python”



Mon premier programme :

Écrire un programme qui définit la fonction $y=\cos(x/5)$ et qui calcule toutes les valeurs de y pour x variant de -10 et +10

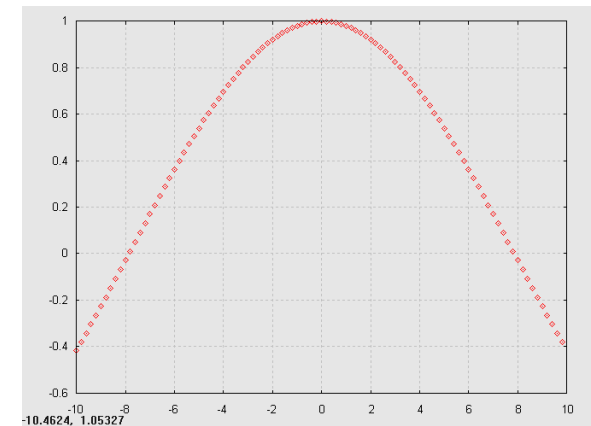
Calculer l'intégrale de la fonction

Tracer la fonction

Introduire une variable r pour faire varier la fonction précédente

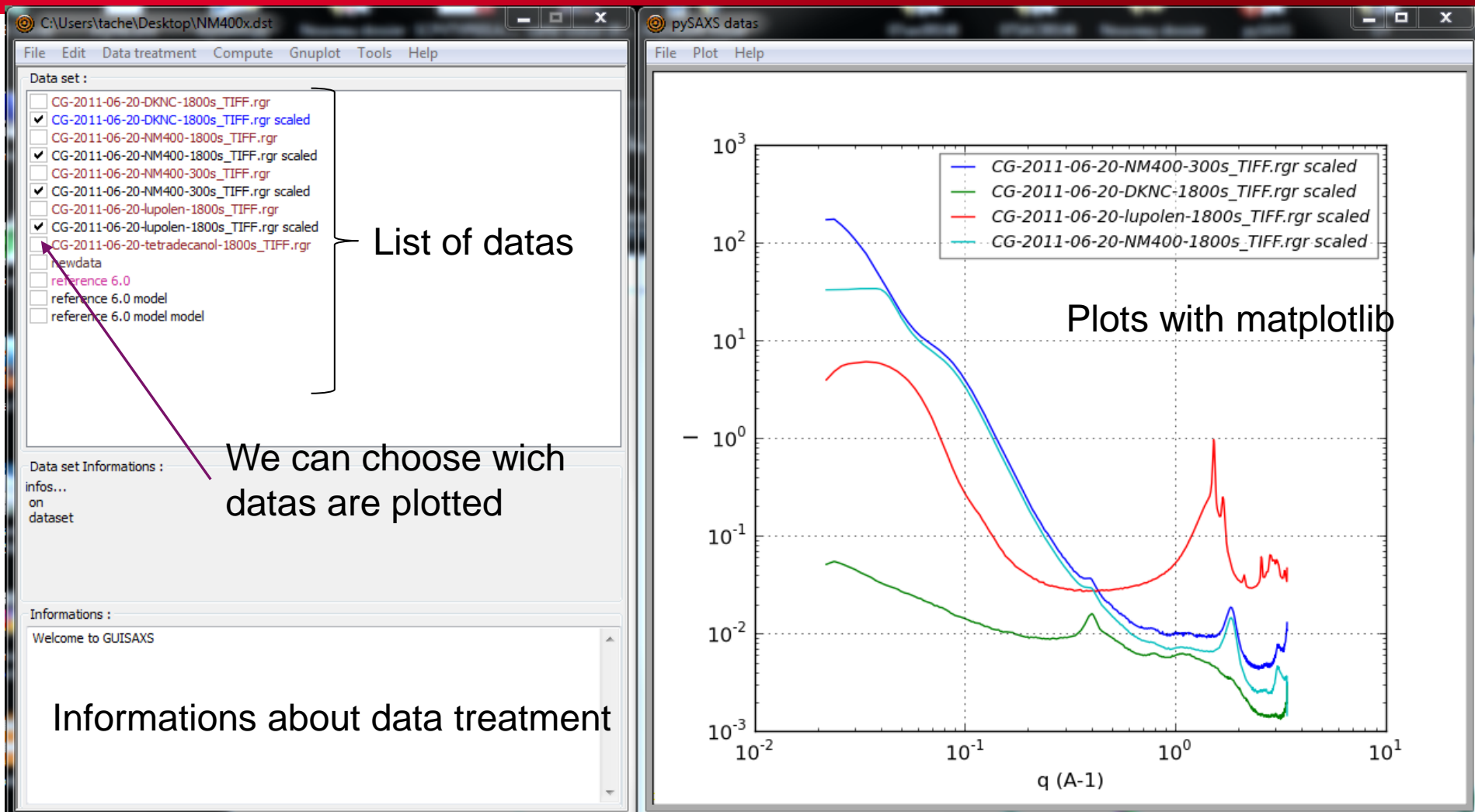
Calculer l'intégrale et tracer la fonction

Fitter la fonction pour retrouver la variable r .

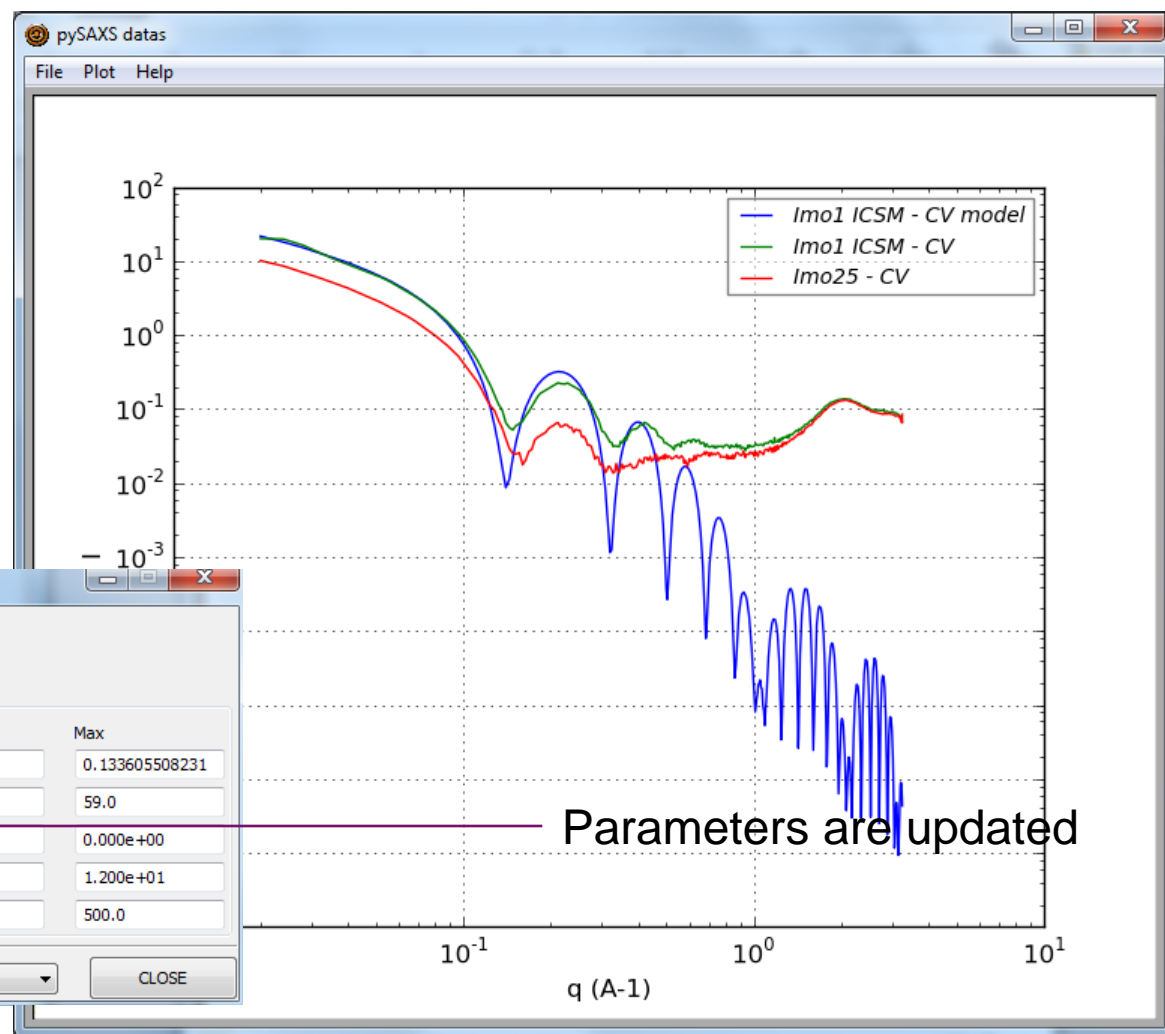


pySAXS

- packages of libraries for SAXS data treatment and modeling
- Graphic interface
- With python, researchers can validate and modify the source code
- With GuiSAXS, standard users can analyze easily datas



FITTING WITH MODELS



Parameter for Imogolite Single Wall Si/Ge

Description : Imogolite single wall variable thickness

Author : Antoine Thill

Chi carre : 3419036.97777

Model parameters :

Parameter	Value	Fit ?	Min	Max
Si or Ge concentration (mol/l) :	1.336e-01	<input checked="" type="checkbox"/> fit	0.000e+00	0.133605508231
Si or Ge in tube circumference :	2.016e+01	<input checked="" type="checkbox"/> fit	0.000e+00	59.0
ratio of Si in the wall :	0.000e+00	<input type="checkbox"/> fit	0.000e+00	0.000e+00
Wall thickness (Å) :	6.000e+00	<input type="checkbox"/> fit	0.000e+00	1.200e+01
imogolite tube length (Å) :	4.099e+02	<input checked="" type="checkbox"/> fit	0.000e+00	500.0

Fit Fit with bounds Y type : normal CLOSE