



énergie atomique • énergies alternatives

# Visualisation 2D avec Python

Matplotlib, PyQt, PyQt5

**Pierre RAYBAUT**

- **Visualiser des données 2D avec Python**

- > **Introduction**

- Visualisation 2D : spécificités de la solution Python

- > **Le choix des armes**

- Environnements de développement interactifs
    - Bibliothèques
    - Interfaces graphiques

- > **Visualisation embarquée dans des interfaces graphiques**

- Programmation orientée objet

- > **Démonstrations**

- Exemples simples
    - Intégration dans des interfaces graphiques

- **Les bibliothèques de visualisation 2D, en deux mots :**

- > **Matplotlib**

Le choix de la polyvalence et du haut niveau

- > **PyQwt**

Le choix de la performance et du bas niveau

- > **guiqwt**

Le choix de la performance et du haut niveau

- **Visualiser des données 2D avec Python**

- > **Introduction**

- Visualisation 2D : spécificités de la solution Python

- Langage Python
    - Interpréteur
    - Bibliothèque standard
    - Bibliothèques exogènes
    - IPython (interpréteur amélioré)
    - Interfaces graphiques
    - ...



- **La distribution officielle du langage Python**
  - « Python est fourni avec les piles »
    - > **Langage Python**
    - > **Interpréteur**
    - > **Bibliothèque standard**
    - > **Environnement de développement IDLE**
- **La distribution officielle est insuffisante**

En tout cas pour tracer des graphiques ou afficher des images

  - > **Calcul numérique quasi-inexistant**
  - > **Interactivité limitée**
  - > **Ah oui, au fait : pas de bibliothèque de représentation graphique...**
- **Questions pertinentes :**
  - > **Quels sont les spécificités de la solution Python ?**

Que faut-il savoir avant de l'utiliser pour visualiser des données 2D ?
  - > **Comment améliorer la distribution officielle ?**

Oui, c'est possible et en plus c'est très facile  
(Nous verrons cela plus tard)



- **Le langage Python**

Langage de programmation ayant six caractéristiques essentielles :

dynamique

généraliste

libre

très haut niveau

orienté objet

gratuit

- **Autres langages dynamiques (non compilés) :** **Java, Ruby, MATLAB®, IDL®**
- **Autres langages généralistes :** **C/C++, Fortran, Java, Ruby**
- **Autres langages orientés objet par conception :** **C++, Java, Ruby**
- **Autres langages libres et gratuits :** **C/C++, Fortran, Java, Ruby**



Source : <http://www.python.org>

- **Le langage Python :**

- > **Exécution :**

Python est un langage interprété

- Exécution d'un script : `python myscript.py`
- Interpréteur en mode interactif : `python`

- > **Comptage de références, objets muables/immuables :**

En Python, tout est objet

```
x = [1, 2]
y = x
x[1] += 3
print y # [1, 5]
```

- > **Mécanisme de la déclaration `import` :**

- Recherche du module (`imp`)
- Mise en cache (`sys.modules`)
- Mise à jour de l'espace de nom global (`globals()`)



- **Les modules ne sont importés qu'une seule fois par session** : pas de rechargement complet des modules à chaque déclaration de type `import`
- **À éviter dans un script** : `from toto import *`  
(réservé ce type de raccourci au mode interactif de l'interpréteur)

- **Le langage Python :**

> **Types de données :** *Built-in data types*     `datatypes = (1, 1., 1+1j,  
                  "a", 'a', u"a", r"a", ""a"", (),  
                  [], {})`

> **Types de données NumPy :**     `np_datatypes = (np.ndarray, np.matrix)`  
NumPy est la bibliothèque des tableaux     `x = np.linspace(-10, 10)`  
à N dimensions et de l'algèbre linéaire     `y = x**2`  
   `z = np.array(x, copy=True)`

> **Rôle de l'indentation :**     `# Rôle de l'indentation : blocs`  
Blocs     `if x:`  
           `pass`

> **Opérateurs numériques :**     `# Opérateurs : +, -, *, /, //, **, %`

> **Affectation multiple :**     `# Affectation multiple :`  
                                   `a, b = range(2)`

> **Opérateurs de test :**     `# Opérateurs de test : <, <=, >, >=, !=, in, is`  
                                   `liste = ['toto', 'tata', 'tutu']`  
                                   `liste2 = liste`  
                                   `print liste is liste2 and 'tata' in liste`



- **Le langage Python :**

- > **Déclaration if... elif... else... :**

```
if a == 1:
    print "c"
elif b == 1:
    print "e"
else:
    print "a"
```

- > **Déclaration de fonctions et objet None :**

Une fonction sans `return` renvoie `None`

```
def gaussian(x, x0=0., sigma=1., a=None):
    """
    Gaussian function
    x0: center (default: 0.)
    sigma: std deviation (default: 1.)
    a: amplitude (default: None)
    --> if a is None: the function is integral-normalized
    """
    ygauss = np.exp(-.5*((x-x0)/sigma)**2)
    if a is None:
        a = 1./ygauss.sum()
    return a*ygauss

def do_and_return_nothing():
    pass
print do_and_return_nothing()
```

- **Le langage Python :**

> **Séquences :**

```
# Séquences
liste = ['toto', 'tata', 'tutu']
for chaine in liste:
    if 'a' in chaine:
        break
```

> **Compréhensions de liste :**

```
# Compréhensions de liste :
liste = []
for fname in os.listdir(os.getcwd()):
    if osp.splitext(fname)[1] == '.txt' and fname.startswith('DATA'):
        liste.append(osp.dirname(fname))

liste = [osp.dirname(fname) for fname in os.listdir(os.getcwd())
        if osp.splitext(fname)[1] == '.txt' and fname.startswith('DATA')]
```

> **Boucles :**

```
# Boucles:
while False:
    pass
for index, value in enumerate(liste):
    print liste
```



- **Le langage Python :**

- > **Gestion des exceptions :**

```
# Gestion des erreurs (exceptions) :
try:
    print u
except NameError:
    # Normal, 'u' n'est pas déclaré...
    pass
except (TypeError, IOError):
    pass
except:
    import traceback
    traceback.print_exc()
finally:
    pass
```

- > **Exceptions utilisateur**

- > **L'exception NotImplementedError :**

```
def super_fonction(x):
    raise NotImplementedError
```

## 7.1. Exception hierarchy

The class hierarchy for built-in exceptions is:

```
BaseException
+-- SystemExit
+-- KeyboardInterrupt
+-- GeneratorExit
+-- Exception
    +-- StopIteration
    +-- StandardError
        | +-- BufferError
        | +-- ArithmeticError
        | | +-- FloatingPointError
        | | +-- OverflowError
        | | +-- ZeroDivisionError
        | +-- AssertionError
        | +-- AttributeError
        | +-- EnvironmentError
        | | +-- IOError
        | | +-- OSError
        | | | +-- WindowsError (Windows)
        | | | +-- VMSError (VMS)
        | +-- EOFError
        | +-- ImportError
        | +-- LookupError
        | | +-- IndexError
        | | +-- KeyError
        | +-- MemoryError
        | +-- NameError
        | | +-- UnboundLocalError
        | +-- ReferenceError
        | +-- RuntimeError
        | | +-- NotImplementedError
        | +-- SyntaxError
        | | +-- IndentationError
        | | +-- TabError
        | +-- SystemError
        | +-- TypeError
        | +-- ValueError
        | | +-- UnicodeError
        | | | +-- UnicodeDecodeError
        | | | +-- UnicodeEncodeError
        | | | +-- UnicodeTranslateError
    +-- Warning
        +-- DeprecationWarning
        +-- PendingDeprecationWarning
        +-- RuntimeWarning
        +-- SyntaxWarning
        +-- UserWarning
        +-- FutureWarning
        +-- ImportWarning
        +-- UnicodeWarning
        +-- BytesWarning
```

# Introduction



- Le langage Python : PEP 008

bad.py

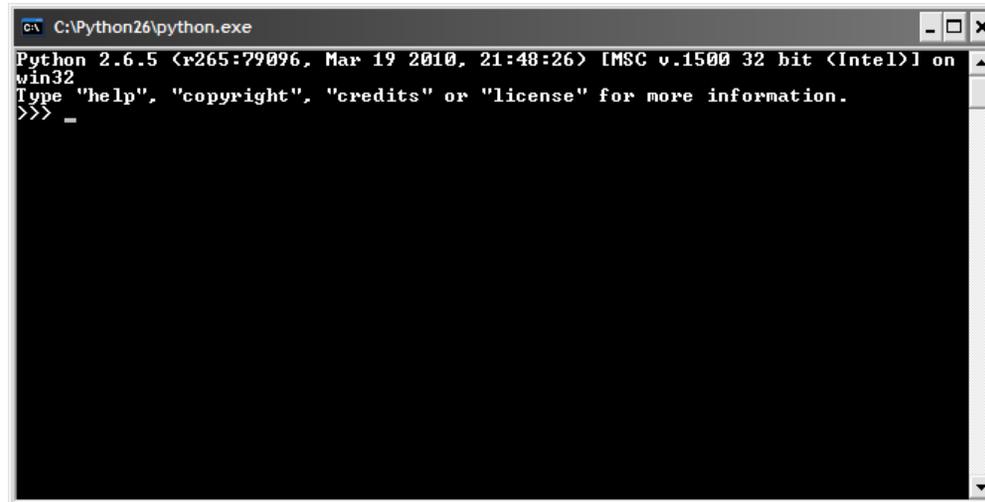
```
1 from pylab import *
2
3 def gaussian(x, x0=0., sigma=1., a=None):
4     ygauss = exp(-.5*((x-x0)/sigma)**2)
5     if a is None:
6         a = 1./ygauss.sum()
7     return a*ygauss
8
9 x = linspace(-10, 10)
10 y = gaussian(x, 2., 3.)
11 plot(x, y)
12 show()
```

good.py

```
1 # -*- coding: utf-8 -*-
2 """
3 Good Module - Documentation
4 =====
5
6 This is a good example of how to write a Python script following
7 the official Python Language development guidelines, `PEP 008`_
8 (aka `Guido's style guide`_).
9 """
10
11 import numpy as np
12
13 def gaussian(x, x0=0., sigma=1., a=None):
14     """
15     Gaussian function
16     x0: center (default: 0.)
17     sigma: std deviation (default: 1.)
18     a: amplitude (default: None)
19     --> if a is None: the function is integral-normalized
20     """
21     ygauss = np.exp(-.5*((x-x0)/sigma)**2)
22     if a is None:
23         a = 1./ygauss.sum()
24     return a*ygauss
25
26 def test():
27     """Testing our gaussian function to see if it works"""
28     import matplotlib.pyplot as plt
29     x = np.linspace(-10, 10)
30     y = gaussian(x, 2., 3.)
31     plt.plot(x, y)
32     plt.show()
33
34 if __name__ == '__main__':
35     test()
```

- **L'interpréteur Python :**

- > **Change la vie des habitués des langages compilés**
- > **A tendance à faire fuir les habitués de MATLAB ou IDL... en particulier sous Windows :**

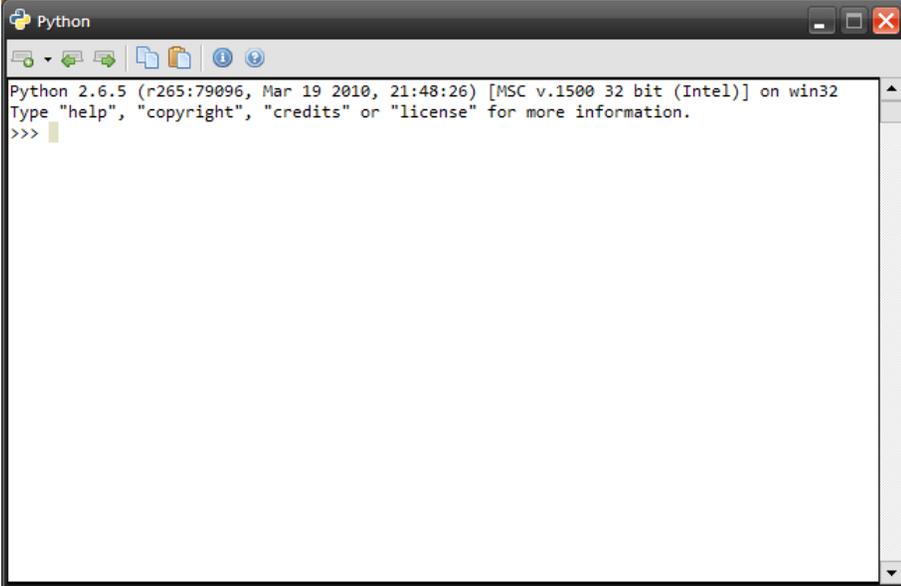


```
C:\Python26\python.exe
Python 2.6.5 (r265:79096, Mar 19 2010, 21:48:26) [MSC v.1500 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>> -
```

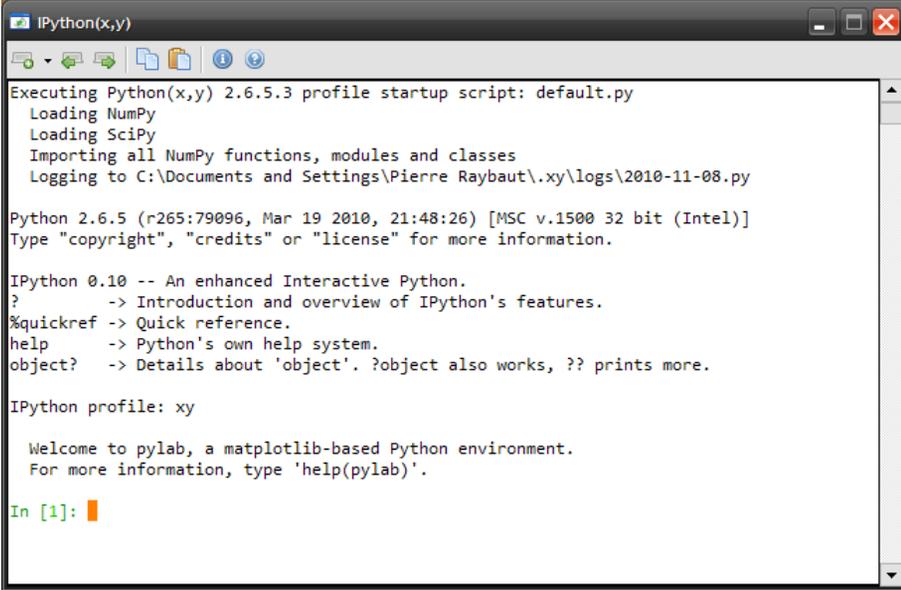
- **Utiliser l'interpréteur Python en mode interactif :**

- > **Sans option :**  
`python`
- > **En exécutant un script, avec l'option `-i` :**  
`python -i myscript.py`

- **Interpréteur Python :**  
Limité mais standard
- **Interpréteur IPython :**  
Amélioré mais non standard
  - > **Commandes magiques**  
(à double tranchant !)
  - > **Mode 'pylab'**
  - > **Débogage plus facile**
  - > **Affichage amélioré**



```
Python 2.6.5 (r265:79096, Mar 19 2010, 21:48:26) [MSC v.1500 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```



```
IPython(x,y)
Executing Python(x,y) 2.6.5.3 profile startup script: default.py
Loading NumPy
Loading SciPy
Importing all NumPy functions, modules and classes
Logging to C:\Documents and Settings\Pierre Raybaut\.xy\logs\2010-11-08.py

Python 2.6.5 (r265:79096, Mar 19 2010, 21:48:26) [MSC v.1500 32 bit (Intel)]
Type "copyright", "credits" or "license" for more information.

IPython 0.10 -- 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.

IPython profile: xy

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

In [1]:
```



- **Bibliothèques standard :**

- > **Types de données de la bibliothèque standard :**

```
datatypes = (1, 1., 1+1j,  
             "a", 'a', u"a", r"a", """a""", (),  
             [], {})
```

- **Bibliothèques exogènes :**

- > **IPython**

- Interpréteur amélioré

- > **numpy : (import numpy as np)**

```
np_datatypes = (np.ndarray, np.matrix)  
x = np.linspace(-10, 10)  
y = x**2  
z = np.array(x, copy=True)
```

- > **PIL : Python Imaging Library**

- Traitement (de base) et affichage d'images

- > **matplotlib, PyQt4.Qwt5, guiqwt**  
(encore un peu de patience)

# Introduction



- **Environnement de développement IDLE :**

Très vite insuffisant pour effectuer calculs et visualisation de manière interactive

The image shows two overlapping windows from the Python IDLE environment. The top window is the 'Python Shell' terminal, displaying the Python 2.6.5 startup screen with a warning about firewall software. The bottom window is a script editor for 'good.py', containing a Gaussian function definition and a test function that plots the function.

```
Python Shell
File Edit Shell Debug Options Windows Help
Python 2.6.5 (x265:79096, Mar 19 2010, 21:48:26) [MSC v.1500 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 2.6.5
>>> |

good.py - D:\Python\good.py
File Edit Format Run Options Windows Help

x0: center (default: 0.)
sigma: std deviation (default: 1.)
a: amplitude (default: None)
--> if a is None: the function is integral-normalized
"""
ygauss = np.exp(-.5*((x-x0)/sigma)**2)
if a is None:
    a = 1./ygauss.sum()
return a*ygauss

def test():
    """Testing our gaussian function to see if it works"""
    import matplotlib.pyplot as plt
    x = np.linspace(-10, 10)
    y = gaussian(x, 2., 3.)
    plt.plot(x, y)
    plt.show()

if __name__ == '__main__':
    test()
```

- **Visualiser des données 2D avec Python**

- > **Introduction**

- Visualisation 2D : spécificités de la solution Python

- > **Le choix des armes**

- Environnements de développement interactifs
    - Bibliothèques
    - Interfaces graphiques

- > **Visualisation embarquée dans des interfaces graphiques**

- Programmation orientée objet

- > **Démonstrations**

- Exemples simples
    - Intégration dans des interfaces graphiques

- **Visualisation 2D et environnement de développement :**

- > **Première approche : le calcul itératif par script interposé**

L'utilisateur modifie son script de calcul dans un environnement de développement avec éditeur et console.

Après chaque modification, il exécute son script et n'interagit pas ou peu avec les données.

- > **Deuxième approche : le calcul interactif en ligne de commande avec [IPython](#)**

L'utilisateur s'appuie éventuellement sur un script mais complète ce dernier de manière interactive (en ligne de commande) et/ou exécute ce script à chaque fois dans la même session [IPython](#).

Il interagit directement avec les données (calculs et visualisations).

- > **Unification des deux approches :**



- **Environnements de développement pour Python :**

- > **Une dizaine de logiciels valables (libres et commerciaux)**

Exemples :

- Eclipse/Pydev (libre, bien adapté au développement de gros projets... mais un peu lourd [Java inside])
- Wing IDE (léger, rapide mais... commercial)
- Spyder (libre, orienté scientifique)

- > **Parmi ces logiciels, seuls 3 ou 4 permettent également de développer avec C/C++ et Fortran**

- > **Un seul prend en charge le calcul interactif (à la MATLAB) :**



- **Visualiser des données 2D avec Python**

- > **Introduction**

- Visualisation 2D : spécificités de la solution Python

- > **Le choix des armes**

- Environnements de développement interactifs
    - Bibliothèques
    - Interfaces graphiques

- > **Visualisation embarquée dans des interfaces graphiques**

- Programmation orientée objet

- > **Démonstrations**

- Exemples simples
    - Intégration dans des interfaces graphiques

# Le choix des armes : bibliothèques



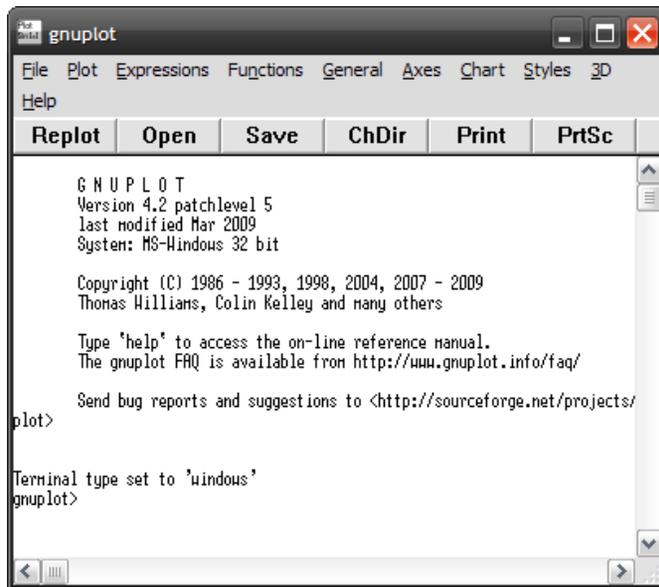
- **Bibliothèques de visualisation 2D pour Python :**

- > **Les plus recommandables :**

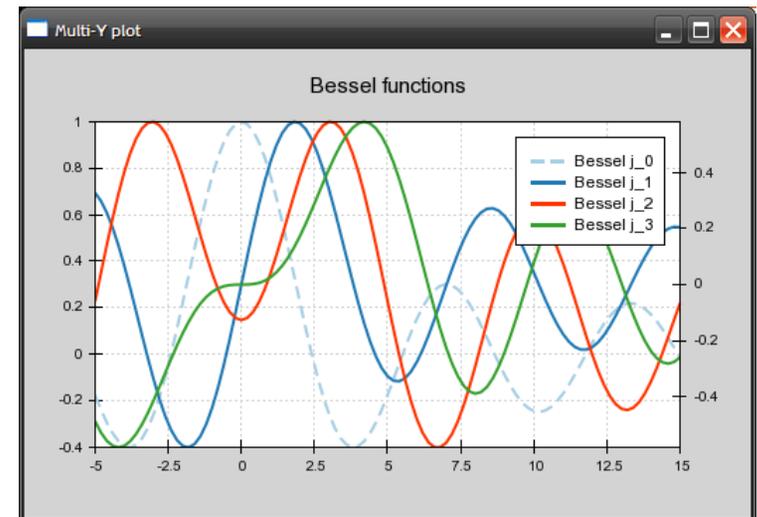
- Matplotlib
    - PyQwt (et guiqwt)

- > **Les autres :**

## gnuplot



## Chaco



**Dépendance douteuse !**



- **Matplotlib :**

- > **Avantages :**

- Bibliothèque la plus riche en types de graphiques 2D (environ 30 !)
    - Bon rendu graphique
    - Export immédiat en PNG, PDF, etc.
    - Documentation riche et nombreux exemples
    - Forte communauté

- > **Inconvénients :**

- Performances aussi mauvaises que celles de MATLAB
    - Interactivité quasi inexistante

- > **Spécificités :**

- Interface 'pylab' : reproduit fidèlement la syntaxe MATLAB
    - Bibliothèque haut niveau : idéale pour le calcul interactif

# Le choix des armes : bibliothèques



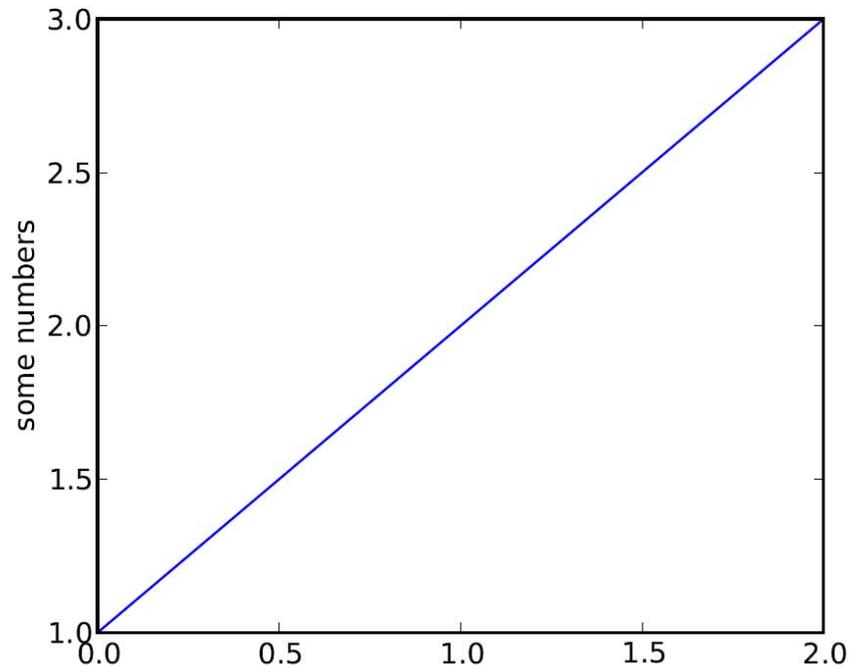
- **Matplotlib :**

- > **Structure interne :**

- Interface *pylab*
    - Frontend (API de base de Matplotlib)
    - Backends

- > **Exemple de base :**

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



- **Matplotlib :**

- > **Courbes :**

- [Matplotlib User's Guide / « Pyplot Tutorial »](#)
    - [Démonstration](#)

- > **Images :**

- [Matplotlib User's Guide / « Image Tutorial »](#)

- > **Aperçu de la documentation et des exemples**



- **PyQwt :**

- > **Avantages :**

- Bibliothèque la plus performante en 2D
    - Très bonne intégration dans les interfaces graphiques modernes (Qt, GTK)
    - Basée sur la bibliothèque C++ Qwt (grand nombre d'utilisateurs)
    - API claire et bien documentée

- > **Inconvénients :**

- Interactivité quasi inexistante
    - Choix limité de types de graphiques 2D

- > **Spécificités :**

- Bibliothèque bas niveau : idéale pour la conception d'applications

- **guiqwt :**

- > **Avantages de PyQwt :**

- Bibliothèque la plus performante en 2D
    - Très bonne intégration dans les interfaces graphiques modernes (Qt, GTK)
    - Basée sur la bibliothèque C++ Qwt (grand nombre d'utilisateurs)
    - API claire et bien documentée

- > **Autres avantages :**

- Interactivité très forte avec les objets graphiques (sélection, édition de paramètres, etc.)
    - Interface haut niveau avec des widgets prêts à l'emploi

- > **Inconvénient : choix limité de types de graphiques 2D**

- > **Spécificités :**

- **Bibliothèque bas niveau** : idéale pour la conception d'applications
    - **Interface haut niveau également** : fonctionne sur le même principe que 'pylab' (remplacer `'import matplotlib.pyplot as plt'` par `'import guiqwt.pyplot as plt'`)



- **guiqwt :**

- > **Aperçu de la documentation et des exemples**

- > **Démonstration de `guiqwt.pyplot`**



- **guidata :**

- > **Modification et affichage graphique de jeux de paramètres**

- Via des interfaces graphiques générées automatiquement

- > **Paramètres de type variable**

- Nombres réels (grandeurs physiques), entiers (indices de tableaux), chaîne de caractères (noms de fichier), booléen (activation d'une option), etc.

- > **Manipulations courantes :**

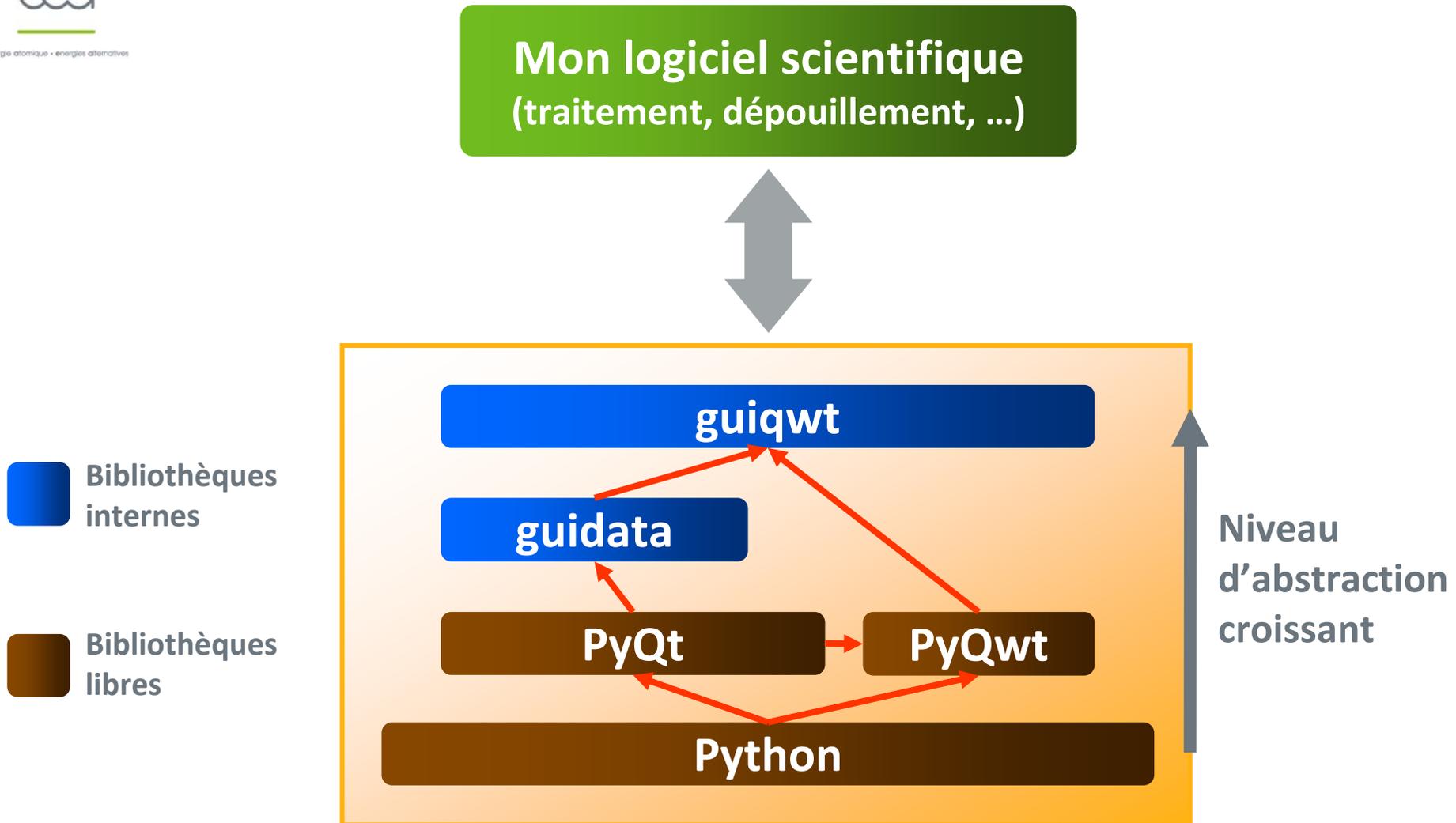
- **Saisie** de chaque paramètre via une interface graphique en adaptant chaque *widget* au type du paramètre concerné
    - **Stockage** des valeurs saisie : convention de stockage
    - **Utilisation des valeurs** pour des calculs (par exemple) sans perdre l'information d'appartenance au jeu de paramètres
    - **Affichage de ces valeurs** dans une interface graphique, en même temps que les résultats de calculs (par exemple), en adaptant l'affichage au type du paramètre concerné

# Le choix des armes : bibliothèques



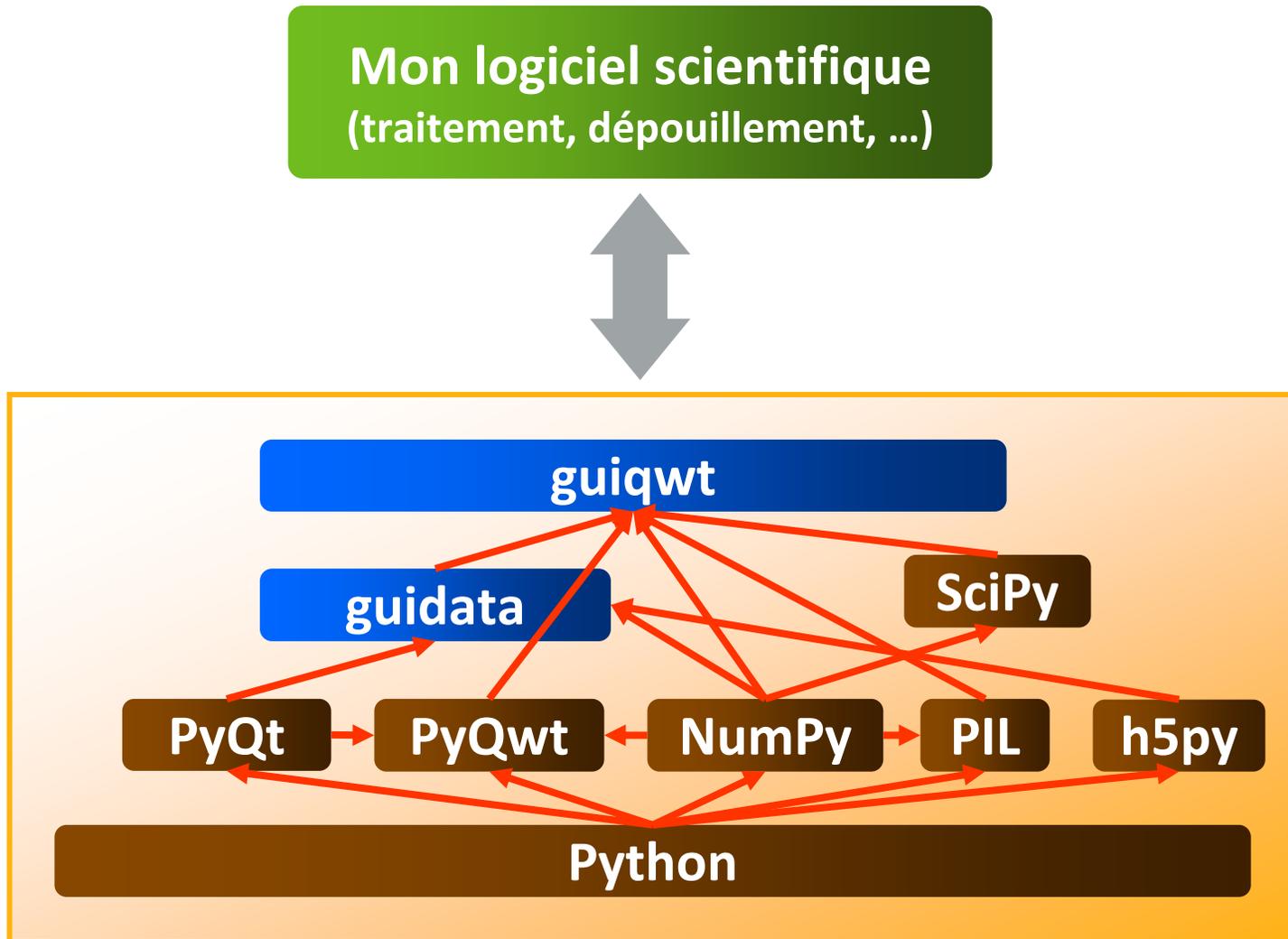
energie atomique • energies alternatives

- **Dépendances de guidata et guiqwt :**



# Le choix des armes : bibliothèques

- **Dépendances de guidata et guiqwt :**

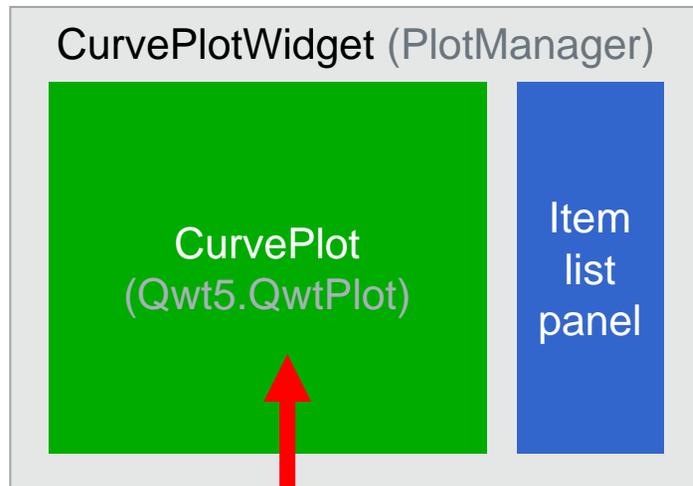


# Le choix des armes : bibliothèques

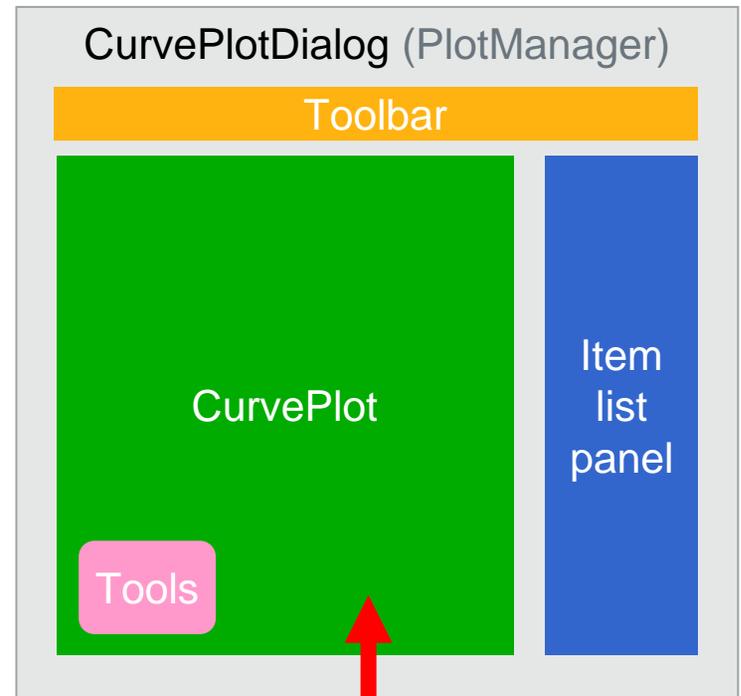


- **guiqwt :**

- > **Widgets intégrant la fonctionnalité *PlotManager* pour la représentation graphique de courbes :**



**Add *plot items***  
(i.e. curves, images, shapes, ...)



**Add *plot items***  
(i.e. curves, images, shapes, ...)

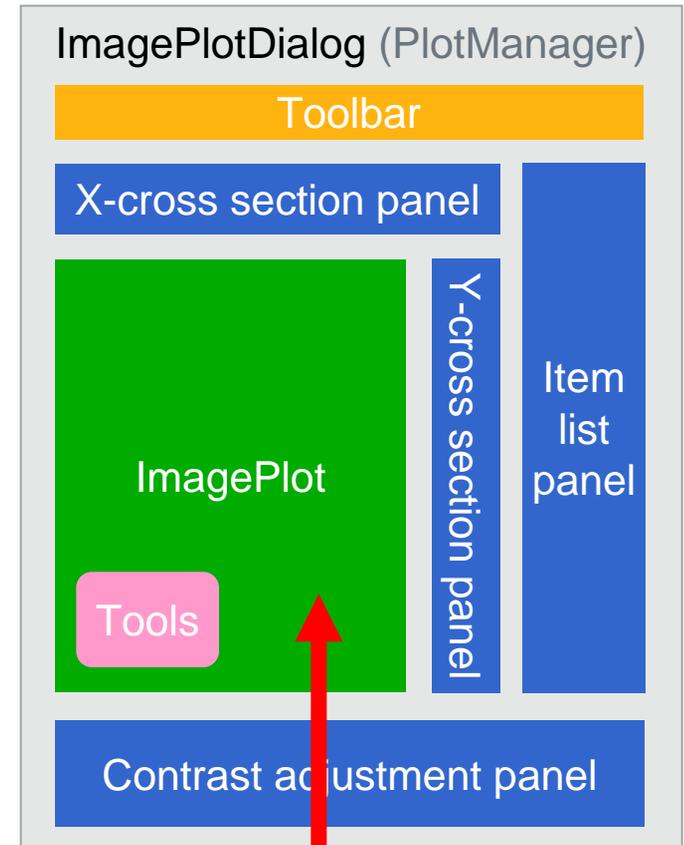
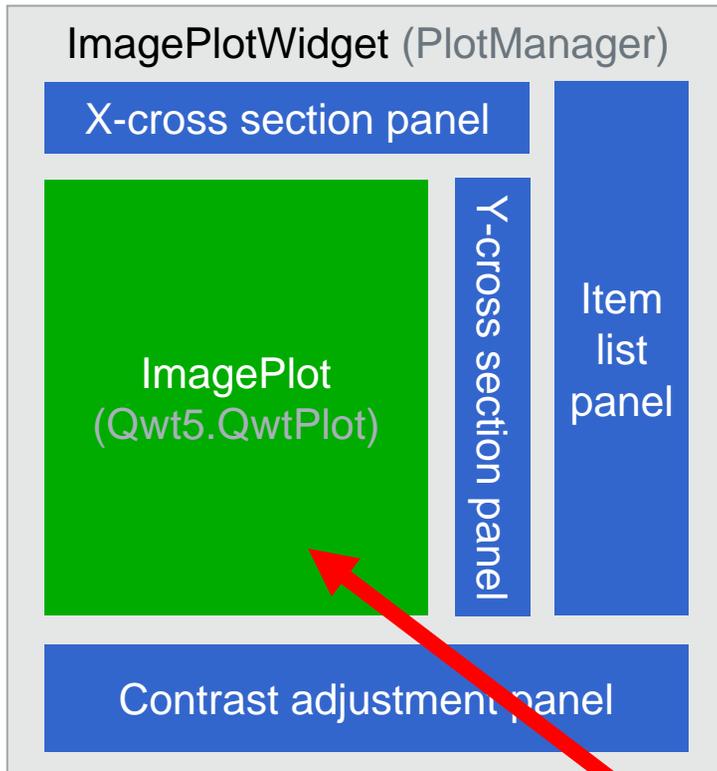
```
guiqwt.curve:      CurvePlot
guiqwt.plot:      CurvePlotWidget
                   CurvePlotDialog
                   (PlotManager)
```

# Le choix des armes : bibliothèques



- **guiqwt :**

- > **Widgets intégrant un *PlotManager* pour l'affichage d'images :**



```
guiqwt.image:      ImagePlot
guiqwt.plot:      ImagePlotWidget
                  ImagePlotDialog
                  (PlotManager)
```

Add plot items  
(i.e. curves, images, shapes, ...)

Add plot items  
(i.e. curves, images, shapes, ...)

# Le choix des armes : bibliothèques

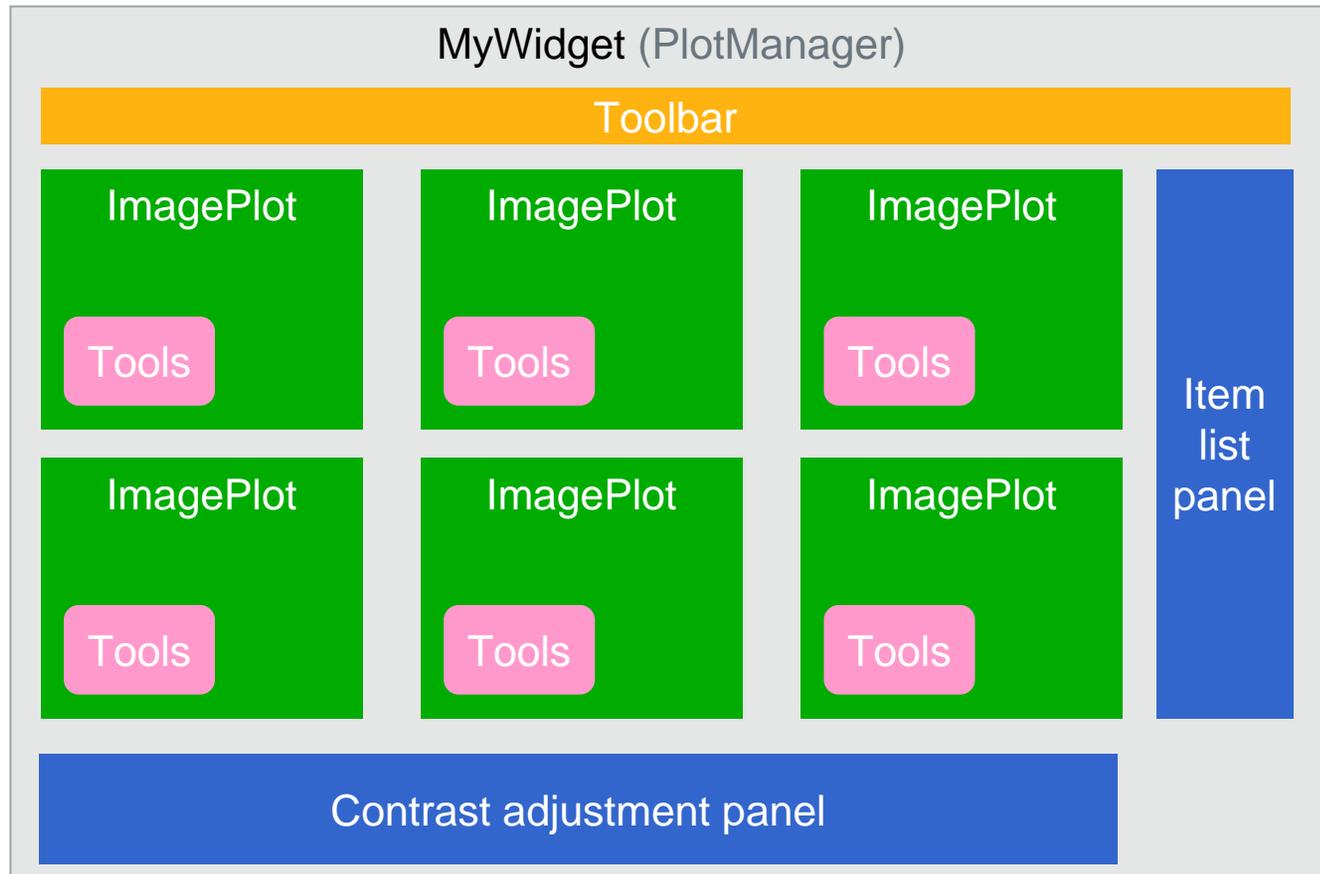


energie atomique • energies alternatives

- **guiqwt :**

- > **Écrire son propre *PlotManager* :**

```
guiqwt.curve:      CurvePlot
                   PlotItemList
guiqwt.histogram: ContrastAdjustment
guiqwt.image:      ImagePlot
guiqwt.plot:       PlotManager
```



- **Visualiser des données 2D avec Python**

- > **Introduction**

- Visualisation 2D : spécificités de la solution Python

- > **Le choix des armes**

- Environnements de développement interactifs
    - Bibliothèques
    - Interfaces graphiques

- > **Visualisation embarquée dans des interfaces graphiques**

- Programmation orientée objet

- > **Démonstrations**

- Exemples simples
    - Intégration dans des interfaces graphiques

- **Bibliothèques d'interfaces graphiques pour Python :**

- > **Tkinter**

Bibliothèque standard (implémentation Python de Tcl/Tk)

- > **PyQt**

Basée sur la bibliothèque C++ Qt

- > **PyGTK**

Basée sur la bibliothèque C GTK

- > **wxPython**

Basée sur la bibliothèque C++ wxWindows

# Le choix des armes : les interfaces graphiques



energie atomique • energies alternatives

Excellents choix

	Points forts	Points faibles
<b>Tkinter</b>	<ul style="list-style-type: none"> <li>- Simplicité</li> <li>- Disponibilité (intégrée à la bibliothèque standard de Python)</li> </ul>	<ul style="list-style-type: none"> <li>- Simplicité (choix limité de <i>widgets</i>)</li> <li>- Apparence austère et mauvaise intégration aux systèmes d'exploitation modernes</li> </ul>
<b>PyQt</b>	<ul style="list-style-type: none"> <li>- Documentation disponible sous plusieurs formes (html, <i>Qt Assistant</i>, <i>Eclipse</i>, livres)</li> <li>- Apparence graphique moderne et native</li> <li>- Éditeur d'interfaces graphiques : Qt Designer</li> <li>- Intégration de Qt Designer dans Eclipse</li> <li>- Excellente disponibilité sur tous les systèmes d'exploitation, Windows compris</li> </ul>	<p>Copie conforme de la bibliothèque C++, PyQt n'exploite pas toutes les possibilités offertes par Python</p>
<b>PyGTK</b>	<ul style="list-style-type: none"> <li>- Documentation claire et complète</li> <li>- Apparence graphique moderne et native</li> <li>- Éditeur d'interfaces graphiques : Glade</li> <li>- La seule bibliothèque d'interfaces graphiques moderne à exploiter pleinement Python</li> </ul>	<p>L'intégration et la disponibilité de PyGTK sont insuffisantes sous Windows</p>
<b>wxPython</b>	<p>Apparence graphique moderne et native</p>	<p>L'architecture, les choix de conception et la portabilité de cette bibliothèque sont réputés moins bons que ceux de PyQt et PyGTK</p>

- **Visualiser des données 2D avec Python**

- > **Introduction**

- Visualisation 2D : spécificités de la solution Python

- > **Le choix des armes**

- Environnements de développement interactifs
    - Bibliothèques
    - Interfaces graphiques

- > **Visualisation embarquée dans des interfaces graphiques**

- Programmation orientée objet

- > **Démonstrations**

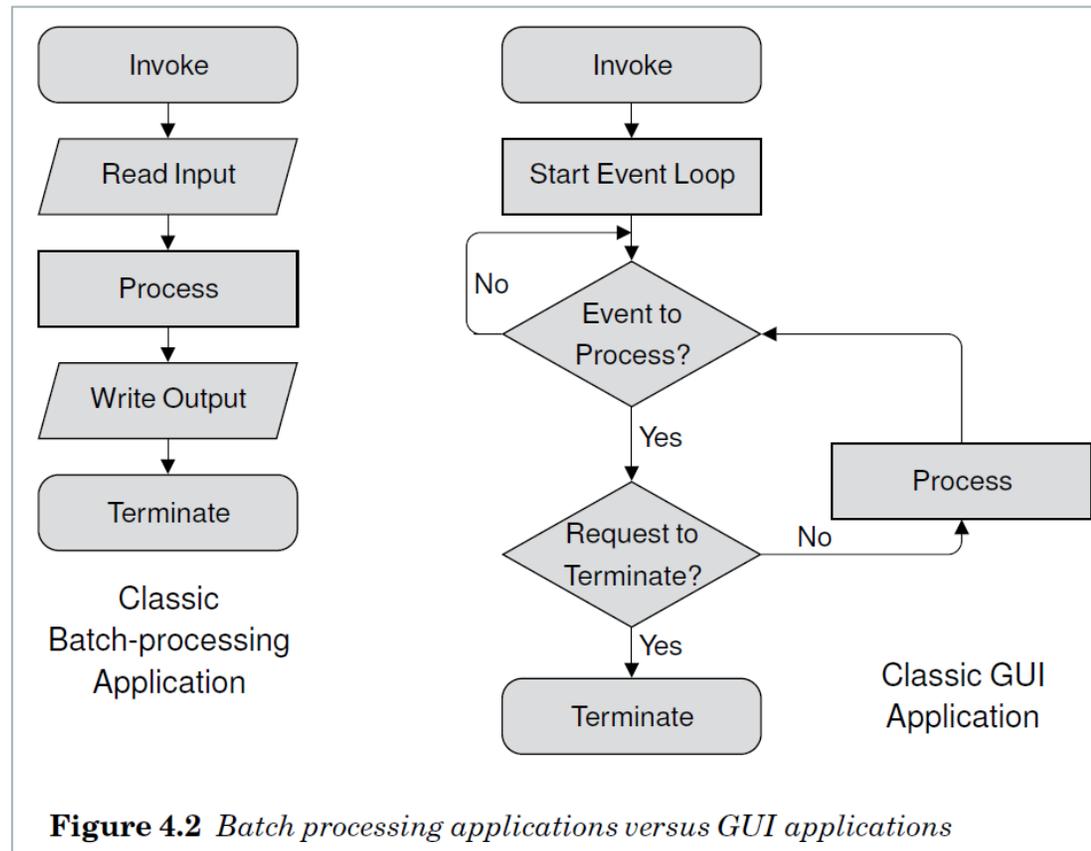
- Exemples simples
    - Intégration dans des interfaces graphiques



- **Prérequis :**
  - > **Notions de programmation d'interfaces graphiques**
  - > **Notions de programmation orientée objet**
  
- **Deux stratégies de programmation :**
  - > **Conception via un *GUI designer***  
*QtDesigner* pour PyQt, *Glade* pour PyGTK
  - > **Conception directe**  
Tout est dans le code source

- **Programmation d'interfaces graphiques**

Démonstrations avec Qt



Source : **Rapid GUI Programming with Python and Qt**  
Mark Summerfield, Ed.: Prentice Hall (2007)



- **Programmation d'interfaces graphiques**  
Démonstrations avec Qt

> **En oubliant la boucle d'événement :**

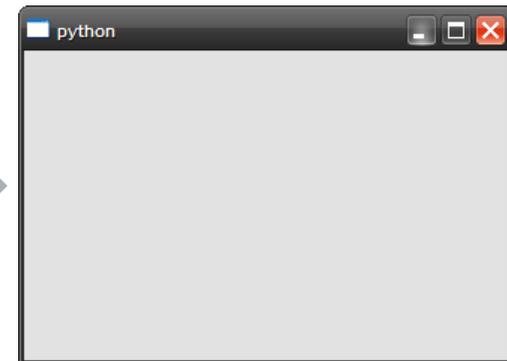
```
from PyQt4.QtGui import QApplication, QWidget  
  
app = QApplication([])  
widget = QWidget()  
widget.show()
```



Une fenêtre s'affiche  
mais elle disparaît aussitôt

> **Avec la boucle d'événement :**

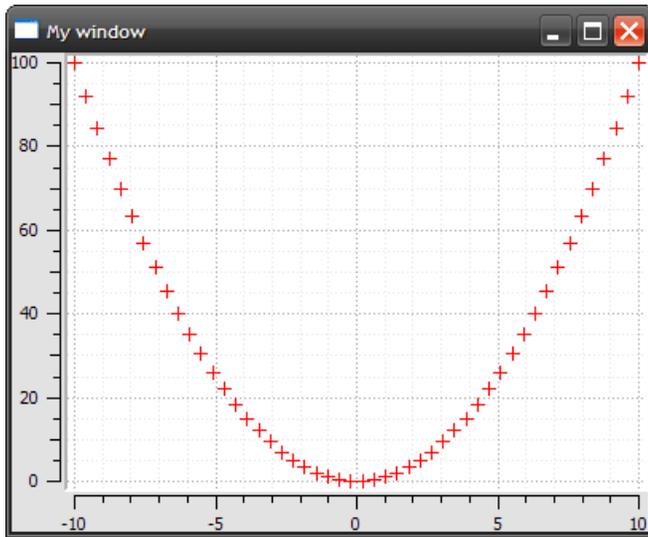
```
from PyQt4.QtGui import QApplication, QWidget  
  
app = QApplication([])  
widget = QWidget()  
widget.show()  
app.exec_()
```



- **Programmation d'interfaces graphiques**

Démonstrations avec Qt

- > **Exemple simple de visualisation embarquée :**



```
from PyQt4.QtGui import QApplication, QMainWindow
```

```
from guiqwt.plot import CurvePlotWidget
from guiqwt.builder import make
```

```
class MyWindow(QMainWindow):
    def __init__(self):
        QMainWindow.__init__(self)
        self.setWindowTitle("My window")

        plotwidget = CurvePlotWidget()
        self.setCentralWidget(plotwidget)

        self.plot = plotwidget.plot

    def plot_data(self, *args):
        self.plot.add_item(make.mcurve(*args))
```

```
if __name__ == '__main__':
    app = QApplication([])
```

```
    win = MyWindow()
```

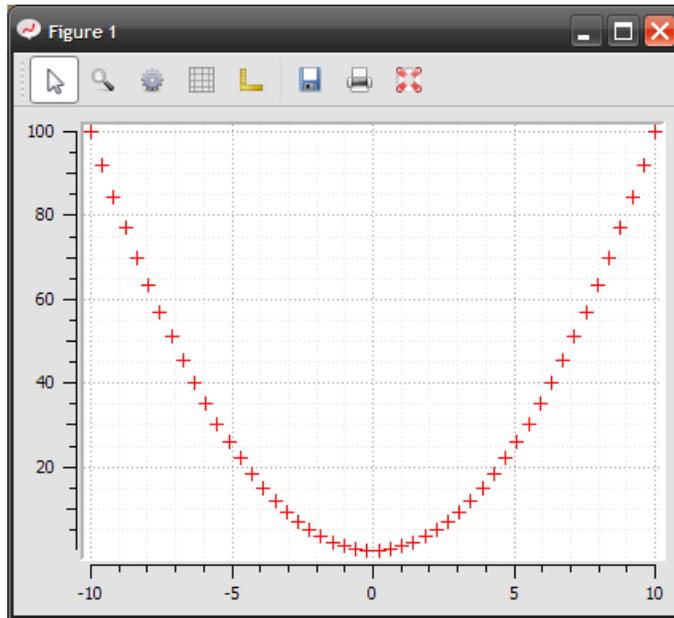
```
    import numpy as np
    x = np.linspace(-10, 10)
    y = x**2
    win.plot_data(x, y, 'r+')
```

```
    win.show()
    app.exec_()
```

- **Programmation d'interfaces graphiques**

Démonstrations avec Qt

> L'interface graphique réalisée avec l'exemple précédent peut être obtenu beaucoup plus facilement via l'interface interactive de **guiqwt** (`guiqwt.pyplot`) ou **matplotlib** (`matplotlib.pyplot`) :



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

x = np.linspace(-10, 10)
y = x**2

plt.plot(x, y, 'r+')
plt.show()
```

# Parenthèse



- **Programmation orientée objet avec Python**

À travers des exemples simples d'interfaces graphiques

- > **filtertest\_mpl.py**

Widgets *Matplotlib* embarqués dans une interface graphique PyQt

- > **filtertest1\_guiqwt.py**

Widgets *guiqwt* embarqués dans une interface graphique PyQt (équivalent de l'exemple basé sur Matplotlib)

- > **filtertest2\_guiqwt.py**

Widgets *guiqwt* embarqués dans une interface graphique PyQt (version améliorée en utilisant un *plot manager*)

- **Visualiser des données 2D avec Python**

- > **Introduction**

- Visualisation 2D : spécificités de la solution Python

- > **Le choix des armes**

- Environnements de développement interactifs
    - Bibliothèques
    - Interfaces graphiques

- > **Visualisation embarquée dans des interfaces graphiques**

- Programmation orientée objet

- > **Démonstrations**

- Exemples simples
    - Intégration dans des interfaces graphiques

- **Le langage Python est un outil formidable :**
  - > Agréable à lire et à écrire, un véritable jeu intellectuel
  - > Simple mais puissant
  - > Interprété (prototypage rapide) mais très performant (extensions C/C++ ou Fortran)
  - > Doté d'une grande et active communauté d'utilisateurs
  
- **La communauté Python scientifique est très active : rejoignez-la !**

Les projets ne manquent pas : numpy, scipy, Python(x,y), Spyder, etc.
  
- **Un dernier conseil pour bien utiliser Python**

Gardez votre indépendance ! (dans la mesure du possible)

  - > **Dans vos habitudes de développement**

Ne soyez jamais dépendant d'un environnement de développement ou d'un système d'exploitation
  - > **Dans le code que vous écrivez**

Minimisez le nombre de dépendances externes  
Évaluez les risques d'obsolescence des modules dont votre code dépend