Utilisation scientifique de Python — Première partie : visualisation de données

Gazette Linux n°114 — Mai 2005

Anders Andreasen

<anders POINT andreasen CHEZ risoe POINT dk>

Article paru dans le n°114 de la Gazette Linux de mai 2005.

Traduction française par Cyril Buttay <cyril POINT buttay CHEZ free POINT fr>.

Relecture de la traduction française par Joëlle Cornavin <jcornavi CHEZ club TIRET internet POINT fr>.

Table des matières

1. Motivation et grandes lignes

La première étape vers une compréhension et une interprétation qualitatives des données scientifiques passe par la visualisation de celles-ci. De plus, pour atteindre une compréhension quantitative, il est nécessaire d'analyser ces données, en y ajustant par exemple un modèle physique. Pour être utilisables, les données brutes peuvent aussi exiger un traitement préliminaire, comme un filtrage, une mise à l'échelle, une calibration, etc.

Il existe plusieurs programmes *open source* pour analyser et visualiser des données : gnuplot, grace, octave, R et scigraphica. Chacun de ces programmes a ses avantages et ses inconvénients. Cependant, il est probable que vous finirez toujours par en utiliser plusieurs pour couvrir les différents besoins décrits ci-dessus, au moins si vous n'avez pas les compétences en programmation suffisantes pour

écrire vos propres programmes personnalisés à l'aide par exemple du Fortran ou du C.

J'ai récemment découvert Python et me suis aperçu que c'est un outil très puissant. Dans cet article, j'aimerais partager mon expérience et montrer que, même avec des compétences en programmation basiques (voire moins), il est toujours possible de créer des applications d'analyse et de visualisation de données très utiles à l'aide de ce langage. L'article est articulé autour de quelques exemples illustratifs et traite de la partie visualisation. L'analyse de données sera abordée dans un article à venir.

2. Python : un bref aperçu

Créé par Guido van Rossum, Python est un langage interprété (comme Perl par exemple), avec une syntaxe claire et facile à lire. Python vous permet d'écrire des applications autonomes, mais un des ses atout réside dans sa capacité à agir en tant que liant entre différentes sortes de programmes.

L'introduction classique à tout langage de programmation est le programme *Bonjour, monde !*. En Python, on l'obtient en ouvrant tout d'abord l'interpréteur Python, en saisissant **python** sur la ligne de commande. Voici ce que votre écran devrait afficher :

```
Python 2.3.3 (#2, Feb 17 2004, 11:45:40)
[GCC 3.3.2 (Mandrake Linux 10.0 3.3.2-6mdk)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Saisissez alors le code suivant :

```
print "Bonjour, monde !"
```

Il est également possible de stocker du code Python dans un fichier appelé par exemple `script.py`. Par convention, les fichiers contenant du code python ont une extension `*.py`. Pour exécuter le script, il suffit de saisir **python script.py** sur la ligne de commande. La sortie du programme s'affiche alors écrite sur la sortie standard et apparaît à l'écran. Si l'on ajoute la ligne suivante au début du fichier :

```
#! /usr/bin/python
```

(en partant du principe que l'exécutable python ou un lien symbolique qui pointe dessus existe) et en donnant au fichier le mode exécutable avec **chmod u+x script.py**, on peut exécuter le script peut être lancé en saisissant **./script.py** sur la ligne de commande.

Python est livré avec de nombreux modules, soit incorporés, soit disponibles en téléchargement et installation séparés. Dans cet article, nous utiliserons SciPy, une bibliothèque très puissante de modules destinés à la visualisation, à la manipulation et à l'analyse de données. Elle ajoute de la fonctionalité à Python en le rendant comparable à Octave ou Matlab par exemple.

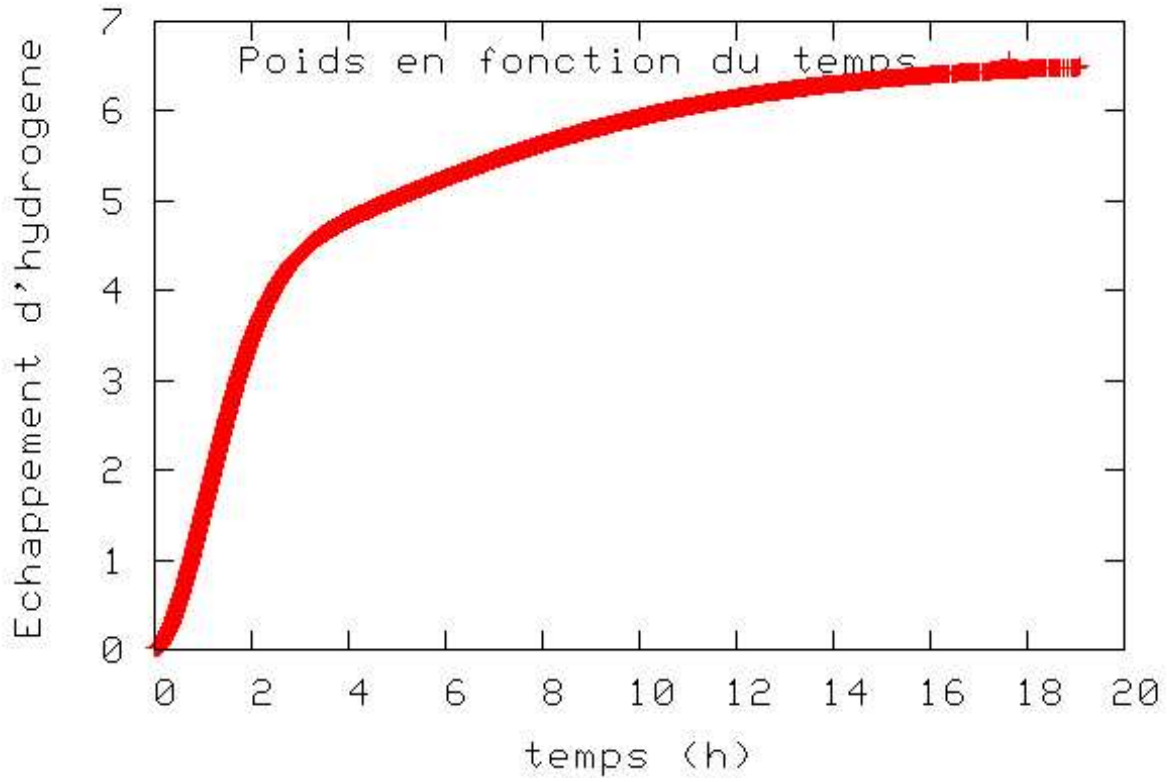
3. Tracer des données en 2D

3.1. Exemple n°1 : tracer des données x,y

Le premier exemple illustre le tracé d'une série de données en 2D. Les données à tracer sont enregistrées dans le fichier `tgdata.dat` et représentent la perte de poids (en pourcentage massique) en fonction du temps. La routine de tracé se trouve dans le fichier `tgdata.py` et le code Python correspondant est affiché ci-dessous. Les numéros de ligne ont été ajouté pour la lisibilité.

```
 1  from scipy import *
 2
 3  data=io.array_import.read_array('tgdata.dat')
 4  plotfile='tgdata.png'
 5
 6  gplt.plot(data[:,0],data[:,1],'title "Poids en fonction du temps" with points')
 7  gplt.xtitle('Temps (h)')
 8  gplt.ytitle("Echappement d'hydrogene")
 9  gplt.grid("off")
10  gplt.output(plotfile,'png medium transparent picsize 600 400'
```

Pour exécuter le programme, téléchargez le fichier `tgdata.py`, renommez-le en `tgdata.py` et lancez le avec **python tgdata.py**. En plus de Python, il faut aussi que SciPy et gnuplot soient installés. Tout au long de cet article, j'ai utilisé la versio 4.0 de gnuplot. La sortie du programme est un tracé sur l'écran comme affiché ci-dessous. Ce tracé est également enregistré sur le disque sous le nom de `tgdata.png` grâce à la commande de la ligne 4 du listing ci-dessus.



Dans la ligne 1, tout ce que contient le module SciPy est importé. Pour se servir des diverses fonctions d'un module, il faut importer ce dernier en ajoutant une ligne `import nom-du-module` au script python. Dans ce cas, il aurait suffi d'importer les paquetages gplt et io.array_import. Dans la ligne 3, le paquetage io.array_import sert à importer le fichier de données `tgdata.dat` dans la variable appelée `data` sous la forme d'un tableau, avec la variable indépendante stockée dans la colonne 0 (notez que les indices de tableau débutent à 0 comme en C, non à 1 comme avec Fortran/Octave/Matlab) et la variable dépendante dans la colonne 1. Dans la ligne 4, une variable contenant le nom du fichier (une chaîne de caractères) dans lequel le tracé devrait être stocké. Dans les lignes 6 à 10, le paquetage gplt fait office d'interface pour piloter gnuplot. La ligne 6 ordonne à gnuplot d'utiliser la colonne 0 pour désigner les valeurs `x` et la colonne 1 pour les valeurs `y`. La notation **data[:,0]** signifie : utiliser/afficher toutes les lignes de la colonne 0. Par ailleurs, **data[0,:]** fait référence à toutes les colonnes de la première ligne.

L'option `png picsize` de gnuplot permettant de générer des fichiers `png` peut s'avérer un peu délicate. L'exemple ci-dessus fonctionne lorsque gnuplot est compilé avec **libpng** + **zlib**. Si votre version de gnuplot a été compilée avec **libgd**, la syntaxe devient **size**, la largeur et la hauteur spécifiées doivent être séparées par une virgule.

Pour tracer le contenu d'un fichier de données ayant un autre nom, nous devons ouvrir le source python dans un éditeur de texte et changer manuellement le nom du fichier à importer. Il faut également changer le nom de la copie du fichier de sortie pour ne pas écraser le tracé précédent. C'est une opération assez fastidieuse. Il est facile d'ajouter cette fonctionnalité à notre script python en permettant que les noms de fichiers soient passés comme arguments de ligne de commande. Le script ainsi modifié est appelé `tgdata1.py` et affiché ci-dessous.

```
 1  import sys, glob
 2  from scipy import *
 3
 4  plotfile = 'plot.png'
 5
 6  if len(sys.argv) > 2:
 7          plotfile = sys.argv[2]
 8  if len(sys.argv) > 1:
 8          datafile = sys.argv[1]
10  else:
11          print "Nom de fichier de données absent. Veuillez en spécifier un"
12          datafile = raw_input("-> ")
13  if len(sys.argv) <= 2:
14          print "Aucun fichier de sortie spécifié. Utilisation de plot.png par défaut"
15  if len(glob.glob(datafile))==0:
16          print "Le fichier de données %s n'a pas été trouvé. Abandon" % datafile
17          sys.exit()
18
19  data=io.array_import.read_array(datafile)
20
21  gplt.plot(data[:,0],data[:,1],'title "Poids en fonction du temps" with points')
22  gplt.xtitle('Temps (h)')
23  gplt.ytitle("Echappement d'hydrogene")
24  gplt.grid("off")
25  gplt.output(plotfile,'png medium transparent picsize 600 400')
```

Dans la première ligne, nous avons importé deux nouveaux modules : **sys** et **glob**, afin d'ajouter la souplesse souhaitée. En Python, **sys.argv** contient les arguments de ligne de commande au moment où l'exécution du script débute. **sys.argv[0]** contient le nom de fichier du script python exécuté, **sys.argv[1]** contient le premier argument de ligne de commande, **sys.argv[2]** le second, etc. La fonction `glob.glob()` se comporte comme **ls** dans les environnements *nix en ce qu'elle fournit les caractères de remplacement (`*`) des noms de fichiers. Si aucun fichier correspondant n'est trouvé, elle renvoie la liste vide (et comporte donc une `len()` égale à zéro), sinon elle contient une liste des noms de fichiers correspondants. Le script peut être exécuté avec autant d'arguments de ligne de commande que l'on souhaite. Si exécuté avec deux arguments, par exemple **python tgdata1.py tgdata.dat tgdata1.png**, le premier argument est le nom du fichier contenant les données à tracer et le second argument correspond au nom de la copie de fichier du tracé.

Le script fonctionne comme suit : un nom de fichier pour la copie par défaut du tracé est stocké dans la variable `plotfile` (ligne 4). Certaines conditions sur le nombre d'arguments de ligne de commande indiqués sont vérifiées. Tout d'abord, si l'on a indiqué deux arguments de ligne de commande ou plus, le contenu de `plotfile` est écrasé par l'argument n°2 (lignes 6 et 7). Les éventuels arguments qui suivent le deuxième sont ignorés silencieusement. Pour un ou plusieurs arguments spécifié(s), l'argument 1 sert de nom de fichier de données (lignes 8 et 9). Si aucun argument de ligne de commande n'a été passé, l'utilisateur est invité à indiquer le nom du fichier de données (lignes 10 à 12). An cas

d’utilisation d’un nom de fichier invalide pour le fichier de données, le script renvoie un message d’erreur et s’arrête.

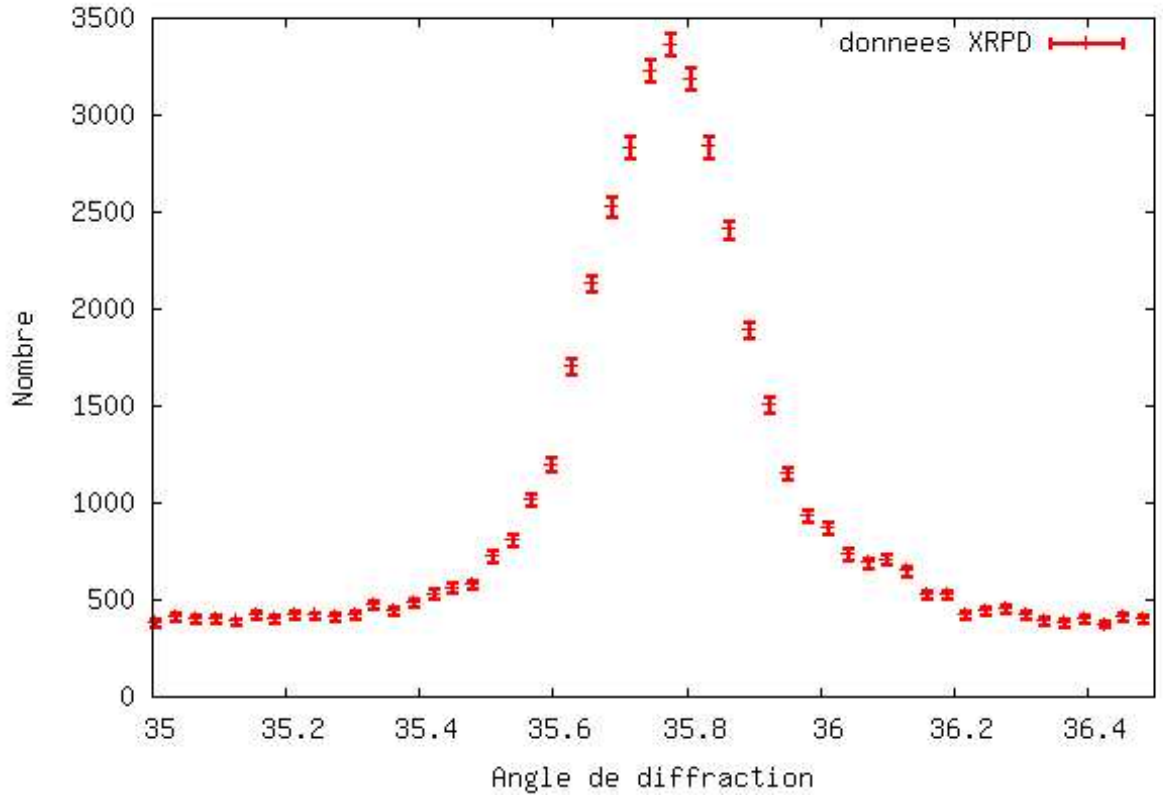
3.2. Exemple n°2 : tracer des données `x,y` avec des barres d’erreur

Jusqu’ici, nous avons montré que le paquetage gplt s’interface aisément avec gnuplot et qu’il est très efficace. Cependant, pour une utilisation scientifique, il est souvent souhaitable de représenter des incertitudes pour chaque point de données. Bien que ce soit possible avec gnuplot, l’interface gplt n’offre pas cette fonctionnalité. À la place, nous utiliserons le paquetage popen présent dans le module os. popen permet de se connecter à l’entrée (ou à la sortie) standard d’un programme via un tube.

Le code ci-dessous (également disponible dans `xrddata.py.txt` ainsi que le fichier de données appelé `xrddata.dat`) montre approximativement comment l’exemple n°1 est reproduit en utlisant **popen** à la place du paquetage gplt. La principale différence est que **popen** ne nécessite pas d’importer les données à tracer dans Python puisque c’est gnuplot qui les lit directement.

```
 1  import os
 2
 3  DATAFILE='xrddata.dat'
 4  PLOTFILE='xrddata.png'
 5  LOWER=35
 6  UPPER=36.5
 7
 8  f=os.popen('gnuplot' ,'w')
 9  print >>f, "set xrange [%f:%f]" % (LOWER,UPPER)
10  print >>f, "set xlabel 'Angle de diffraction'; set ylabel 'Nombre'"
11  print >>f, "plot '%s' using 1:2:(sqrt($2)) with errorbars title 'donnees XRPD' lw 3" % DATAFILE
12  print >>f, "set terminal png  large transparent size 600,400; set out '%s'" % PLOTFILE
13  print >>f, "pause 2; replot"
14  f.flush()
```

Le code de l’exemple n°2 produit la sortie affichée ci-dessous. Le tracé de la barre d’erreur est obtenu avec **plot ’filename’ using 1:2:(sqrt($2)) with errorbars** parce que dans xrddata.dat, l’écart-type standard est égal à la racine carrée de des valeurs y. C’est un cas spécial. D’habitude, les valeurs d’incertitude sont écrites explicitement sous la forme d’une troisième colonne dans le fichier de données. Ainsi, un tracé de barre est créé à l’aide de **plot ’filename’ using 1:2:3 with errorbars**.



4. Tracer des données en 3D

4.1. Exemple n°3

Nous allons maintenant étudier la manière de représenter des données en 3D en utilisant une combinaison Python/gnuplot. Pour ce faire, gnuplot exige que les données lui soient fournies par une expression mathématique ou qu'elles soient stockées dans un fichier de données. Ce dernier doit avoir soit un format *matrix* dans lequel les valeurs de `z` sont indiquées sous forme de matrice, avec les valeurs `x` et `y` égales au numéro de ligne et de colonne, respectivement, correspondant à chaque valeur de `z`, soit un format à trois colonnes dans lequel chaque ligne représente un triplet (`x, y, z`) correspondant aux première, deuxième et troisième colonnes, respectivement. Pour plus de détails, reportez-vous au gnuplot manual.

Dans cet exemple, les données à représenter en 3D sont en fait un ensemble de fichiers de données en 2D (comme celui de l'exemple n°2). Comme chaque fichier de données correspond à un enregistrement à un moment différent (avec des intervalles de 150 s entre deux enregistrements), nous avons deux variables indépendantes et une variable dépendante : x (numéro de fichier/temps), y (angle de diffraction) et z (nombre d'unités) réparties dans plusieurs fichiers [1 [p 10]]. Ce comportement fait qu'il n'est pas — *encore* — possible de tracer des données en 3D.

Le script 3ddata_1.py affiché ci-dessous trouve tous les fichiers ayant une extension donnée (`*.x_y` dans ce cas) dans le répertoire de travail actuel et crée une liste contenant leurs noms de fichiers (ligne 5, `FILELIST`). Dans la ligne 6, le nombre de lignes de données dans chaque fichier est déterminé (`SIZEX`). Ces informations, y compris le nombre de fichier de données, sert ensuite à construire un tableau (`DATAMATRIX`) avec une taille de `SIZEX` par `len(FILELIST)`. Dans les lignes 11 et 12, nous parcourons les fichiers de données en copiant la seconde colonne du fichier de données

numéro y dans la colonne y de DATAMATRIX. Le tableau contient à présent toutes les valeurs de z. Il ne s'agit que d'un tableau temporaire convenant pour traîter les données avant que soit écrit le fichier de données proprement dit destiné au tracé en 3D.

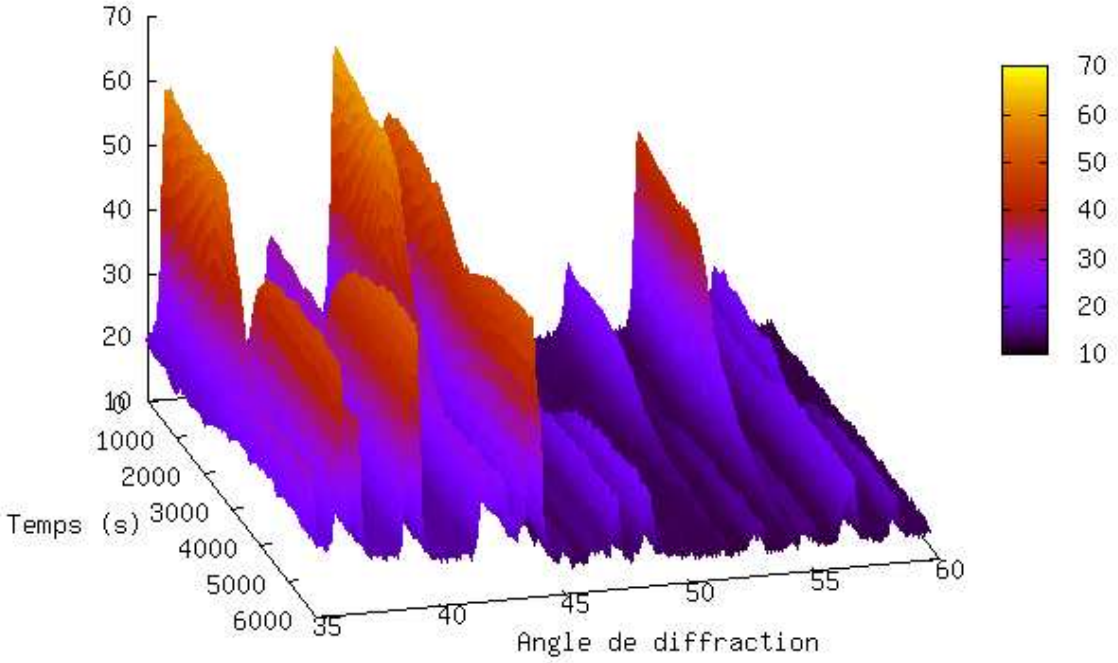
Dans les lignes 14 à 22, le fichier de données est écrit au format (x,y,z), où x correspond au temps (obtenu en multipliant le numéro de fichier par le pas de temps), y correspond à l'angle de diffraction tel qu'il est indiqué par TWOTHETA et z correspond au nombre d'éléments. Dans ce cas, nous voulons seulement tracer les données ayant des angles de diffraction compris entre 35 et 60° (ce qui correspond aux lignes de données 1126 à 1968). Par conséquent, seule cette plage sera écrite dans le fichier, afin d'accélérer à la fois le processus d'écriture dans le fichier et le tracé. Dans les lignes 24 à 29, l'ensemble est envoyé à gnuplot à l'aide du paquetage popen.

```
 1  import os, glob
 2  from scipy import *
 3
 4  EXT='*.x_y'
 5  FILELIST=glob.glob(EXT)
 6  SIZEX = len(io.array_import.read_array(FILELIST[0]))
 7  DATAMATRIX = zeros((SIZEX,len(FILELIST)), Float)
 8  TWOTHETA=io.array_import.read_array(FILELIST[0])[:,0]
 9  TIMESTEP=150
10
11  for y in range(len(FILELIST)):
12                  DATAMATRIX[:,y]=sqrt(io.array_import.read_array(FILELIST[y])[:,1])
13
14  file = open("3ddata.dat", "w")
15
16  for y in range(len(FILELIST)):
17          for x in range(1126,1968):
18                  file.write(repr(TIMESTEP*y)+" "\
19                  +repr(TWOTHETA[x])+" "+repr(DATAMATRIX[x,y]))
20                  file.write("\n")
21          file.write("\n")
22  file.close()
23
24  f=os.popen('gnuplot' ,'w')
25  print >>f, "set ticslevel 0.0 ;set xlabel 'Temps (s)'; set ylabel 'Angle de diffraction'"
26  print >>f, "set pm3d; unset surface; set view 60,75; splot '3ddata.dat' notitle"
27  print >>f, "set terminal png large transparent; set out '3ddata_1.png'"
28  print >>f, "replot"
29  f.flush()
```

Si vous voulez écrire un fichier de données en 3D au format *matrice*, il faut remplacer les lignes 14 à 22 par le code suivant :

```
file = open("3ddata_matrix.dat", "w")
for x in range(SIZEX):
        for y in range(len(FILELIST)):
                file.write(repr(DATAMATRIX[x,y])+" ")
        file.write("\n")
file.close()
```

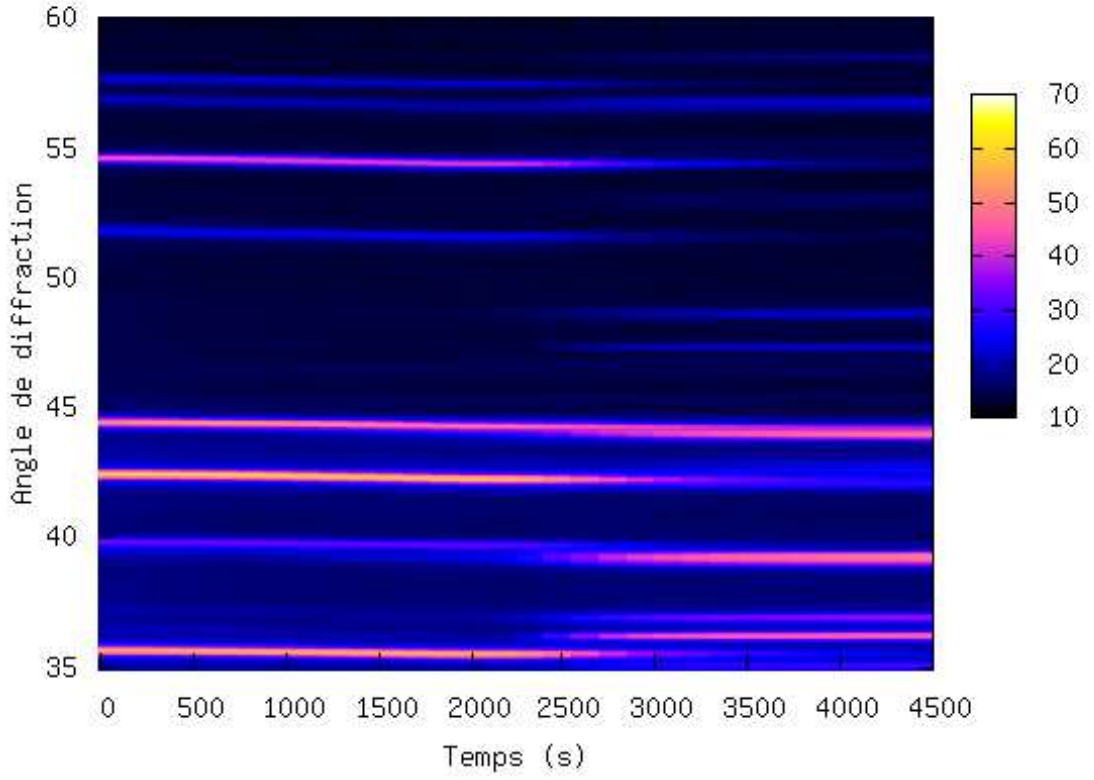
Voici le tracé 3D obtenu avec le programme ci-dessus :



On voit ici qu'il peut être difficile de produire un tracé en 3D de données non monotoniques qui montre tous les détails des données — quelques-uns des pics plus petits sont masqués par de plus grands. Il n'est pas non plus évident de voir l'évolution de la position des pics en fonction du temps. Pour faire ressortir ces détails, il est parfois plus judicieux de créer un tracé de contour en 2D en projetant les valeurs de `z` sur le plan `x,y`. Pour ce faire, il faut remplacer les lignes 24 à 29 de 3ddata_1.py par le code ci-dessous (3ddata_2.py).

```
f=os.popen('gnuplot' ,'w')
print >>f, "set pm3d map; set palette rgbformulae 30,31,32; set xrange[0:4500]"
print >>f, "set xlabel 'Time [s]'; set ylabel 'Diffraction angle'"
print >>f, "splot '3ddata.dat' notitle"
print >>f, "set terminal png large transparent size 600,500; set out '3ddata.png'"
print >>f, "replot"
f.flush()
```

Voici le tracé obtenu :



Ces exemples de tracés en 3D ont été obtenus à partir de 39 fichiers contenant chacun 4 096 lignes de données.

5. Résumé

J'ai présenté dans cet article quelques exemples pour illustrer que Pyhton est vraiment un outil puissant de visualisation de données scientifiques en combinant la puissance en termes de tracé de gnuplot avec celle d'un vrai langage de programmation. On notera que tous les exemples indiqués ici pourraient certainement avoir été résolus en combinant bash, gawk et gnuplot. Il m'apparaît que Python est beaucoup plus simple et que les scripts en résultant sont plus transparents, faciles à lire et à maintenir. Si un lourd traitement de données s'avères nécessaire, la combinaison bash/gawk/gnuplot pourrait également nécessiter la fonctionnalité ajoutée d'un langage comme octave. Avec Python, cette fonctionnalité réside dans SciPy.

6. Autres suggestions de lecture

Manuels, tutoriaux, livres, etc. :

1. Guido van Rossum, *Python tutorial* : http://docs.python.org/tut/tut.html
2. Guido van Rossum, *Python Library Reference* : http://docs.python.org/lib/lib.html
3. Mark Pilgrim, *Dive into Python* : http://diveintopython.org/toc/index.html
4. Thomas Williams & Colin Kelley, *Gnuplot - An Interactive Plotting Program* : http://www.gnuplot.info/docs/gnuplot.html
5. Travis E. Oliphant, *SciPy tutorial* : http://www.scipy.org/documentation/tutorial.pdf
6. David Ascher, Paul F. Dubois, Konrad Hinsen, Jim Hugunin et Travis Oliphant, *Numerical Python* : http://numeric.scipy.org/numpydoc/numdoc.htm

Consultez également les articles précédents sur Python publiés dans la *Linux Gazette* [2 [p 10]].

7. Autres bibliothèques graphiques pour Python

Dans cet article, nous avons utilisé la version 4.0 de gnuplot. Python fonctionne également avec d'autres moteurs et/ou bibliothèques graphiques :

1. Grace : http://www.idyll.org/~n8gray/code/
2. PGPLOT : http://efault.net/npat/hacks/ppgplot/
3. PLplot : http://www.plplot.org/
4. matplotlib : http://matplotlib.sourceforge.net/ (bibliothèque spécifiquement conçue pour Python)

Anders utilise linux depuis environ 6 ans. Il a commencé avec RedHat 6.2, est passé aux versions 7.0, 7.1, 8.0, à Knoppix, a fait quelques expérimentations avec Mandrake, Slackware, FreeBSD et utilise actuellement Gentoo sur son poste de travail (sans double amorçage) au travail et Debian Sarge sur son portable à la maison. Anders a (un peu) d'expérience en programmation C, Pascal, Bash, HTML, LaTeX, Python et Matlab/Octave.

Anders est titulaire d'un diplôme d'ingénieur en chimie et prépare actuellement une thèse dans le Materials Research Department, du National Laborary de Risö (Danemark). Il a également la charge du site web Hydrogen storage at Risö.

[1 [p 6]] Les fichiers sont disponibles dans l'archive 3ddata.tar.gz (N. d. T.).

[2 [p 10]] Certains articles sont disponibles en français sur le site du projet traduc.org (N. d. T.).