

Introduction à Python pour la programmation scientifique

Vincent Noel, Ludmila Klenov,
Yohann Morille, Dmitry Khvorostyanov

LMD, IPSL/CNRS-X-ENS-UPMC

Journée de formation IPSL - 2010-04-08, Ecole Polytechnique,
Palaiseau, France

Programme de la journée

Matin (maintenant)

10h30-12h30

Introduction générale à Python

Après-midi

14h-17h30

Travaux Pratiques

Est-ce que Python va dominer le monde?

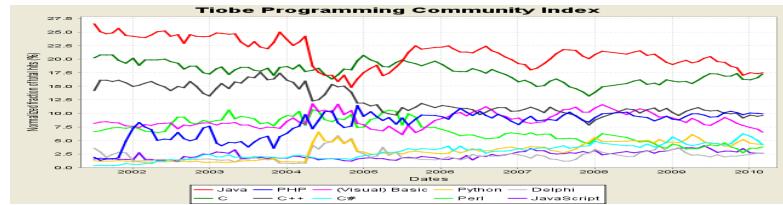
Les 30 et 31 mai 2009, AFPy : conférence à la Cité des Sciences et de l'Industrie à Paris

- AFPY: ~ 200 inscrits; rencontres, forums, formations, documents éducatifs
- "Planets" et "User Groups" de Python
- PyCon, EuroPython, PyCon UK, OSCON, ...
- Google, Plone, Zope, LaunchPad, Django - Python-powered!
- Argentine Py Group: CDPedia
- One Laptop Per Child: Sugar
- Bungeni : système d'information pour les parlements africains + ouverts et accessibles
- Python African Tour: Formations Maroc, Sénégal, Zambie, Nigéria...



The screenshot shows the homepage of the Pilot Systems website. At the top, there's a navigation bar with links for accueil, services, solutions, technologies, actualités, and contributions. The main header features the Pilot Systems logo, which includes a yellow airplane icon and the text "Pilot Systems". Below the header, there's a large image of a man in a dark suit and tie, wearing red boxing gloves and has green, spiky hair. To the right of the image, there's a sidebar with various links related to Python and open source. A calendar for April 2010 is visible in the bottom right corner, showing the 28th highlighted. At the bottom left, there are logos for ENA (Ecole Nationale d'Administration) and Crédit Municipal de Paris.

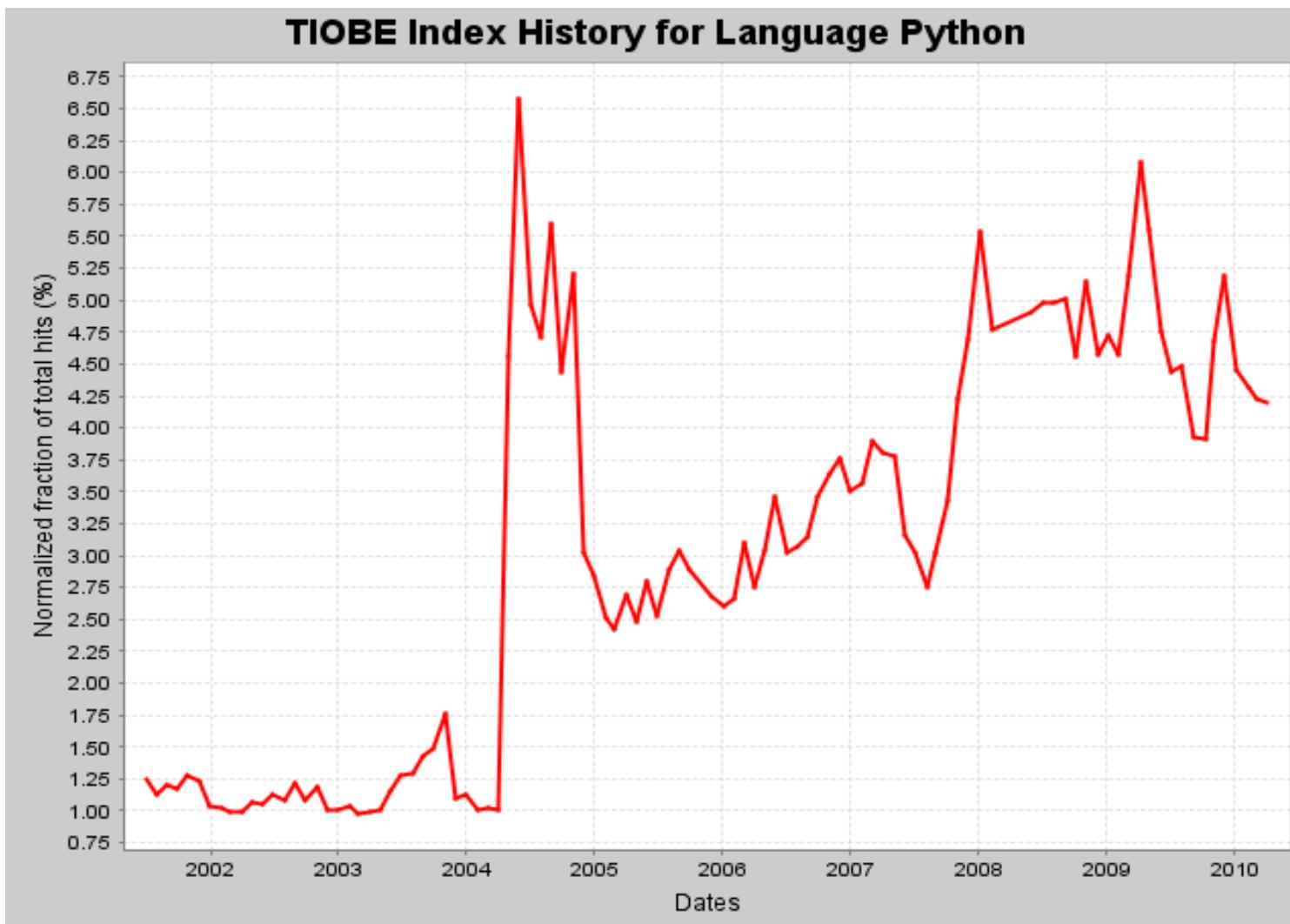
Est-ce que Python va dominer le monde?



The TIOBE Programming Community index:

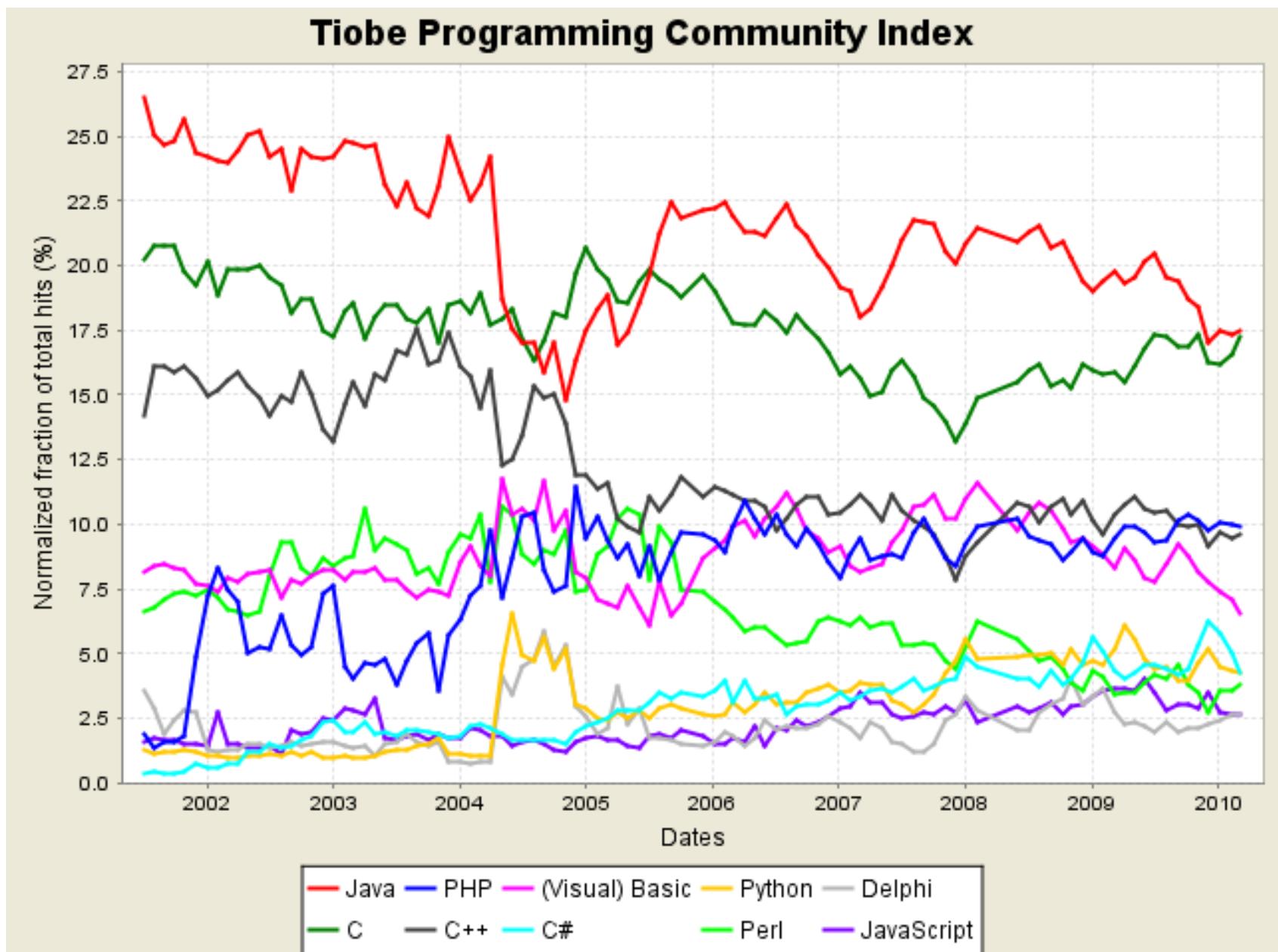
- Based on:
 - The number of skilled engineers world-wide
 - Courses
 - Third party vendors
- Ratings are calculated based on number of searches in Google, MSN, Yahoo!, Wikipedia, and YouTube
- The results are counted for the last 12 months
- The index is updated every month

Est-ce que Python va dominer le monde?



- Highest Rating (since 2001): 6.579% (7th position, May 2004)
- Lowest Rating (since 2001): 0.974% (13th position, February 2003)

Est-ce que Python va dominer le monde?

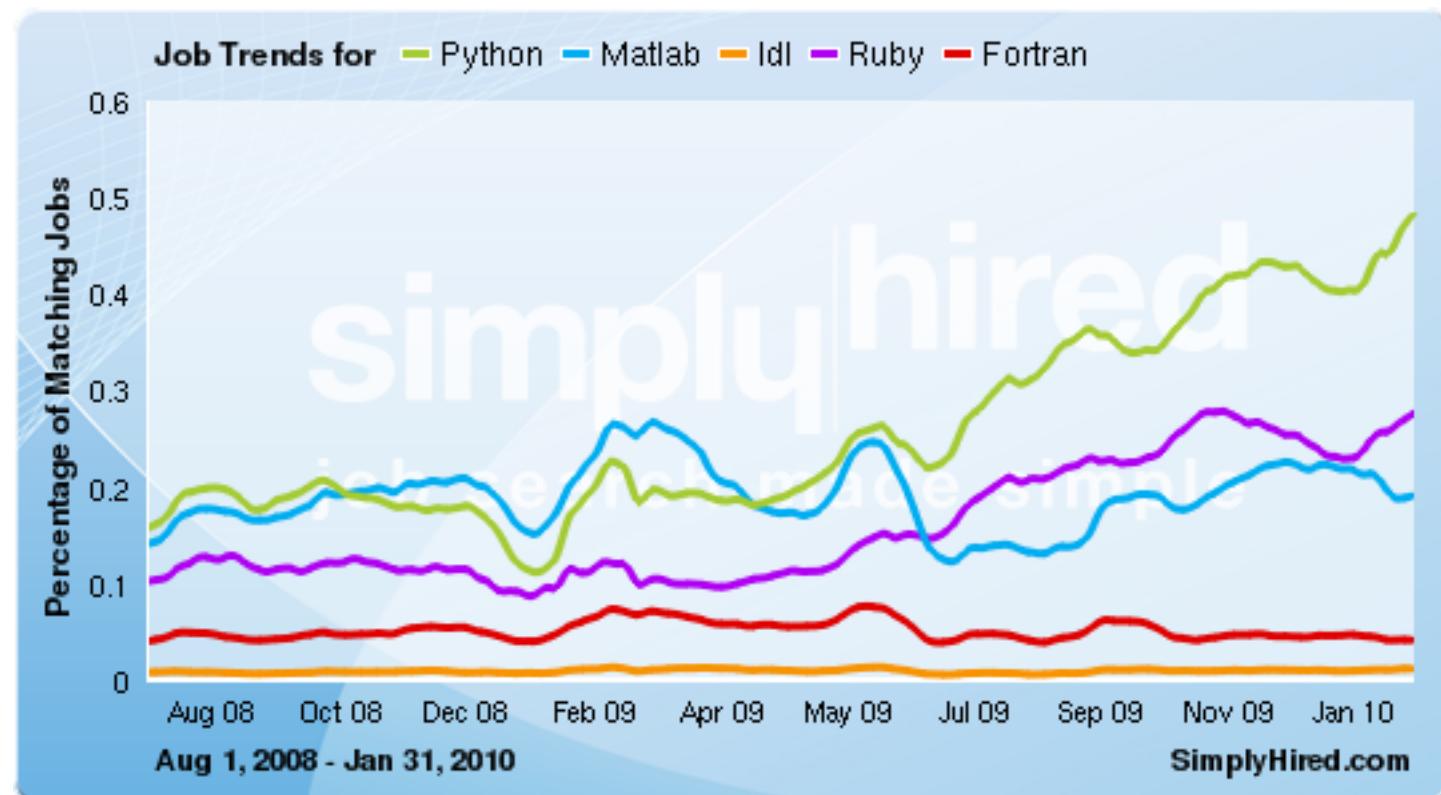


Est-ce que Python va dominer le monde?

Percentage of jobs in US job listings since August 2008

Langages de Programmation Scientifique

- Python : + **206%**
- Matlab : + 35%
- IDL : + 43%
- Ruby : + **170%**
- Fortran : + 4%

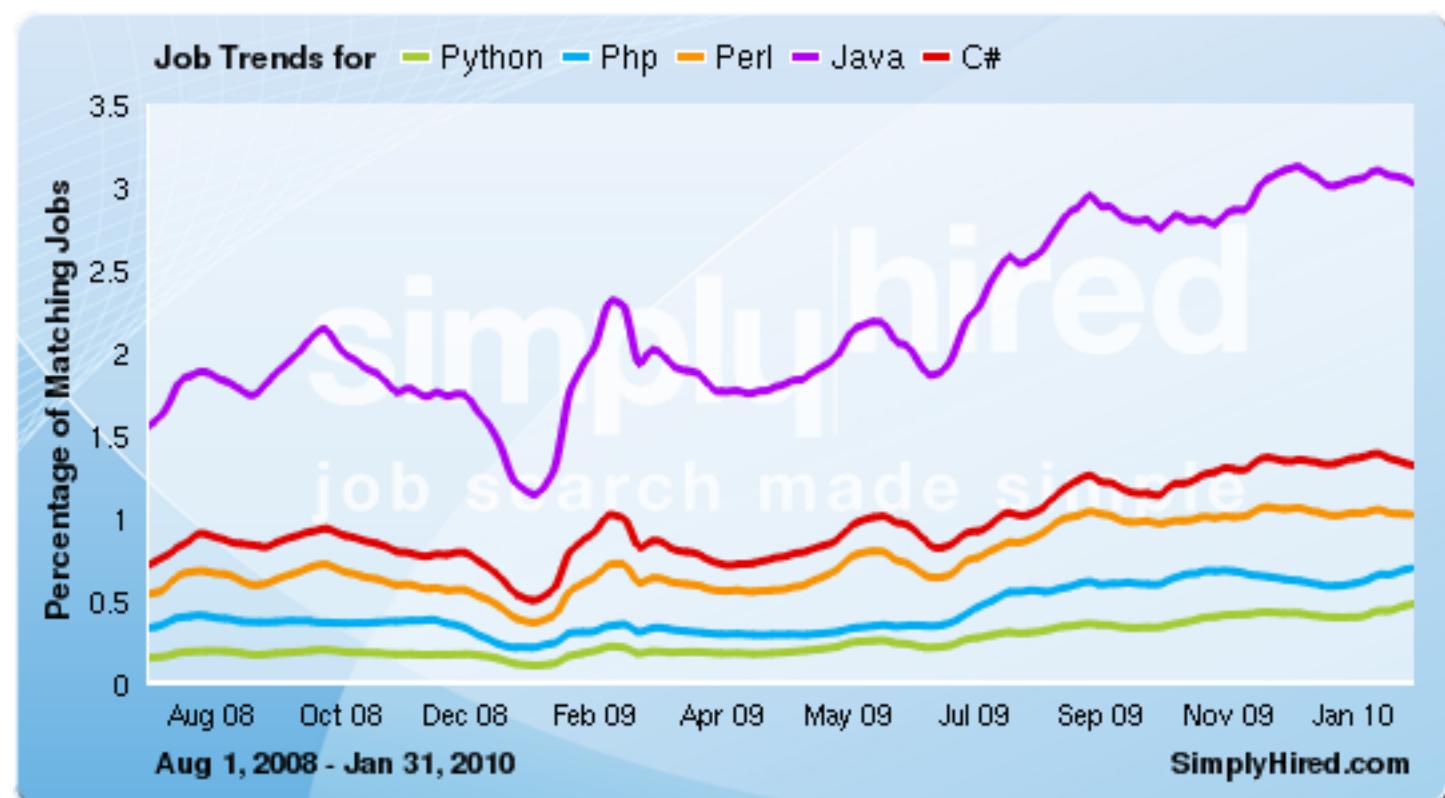


Est-ce que Python va dominer le monde?

Percentage of jobs in US job listings since August 2008

Langages de Programmation Web

- Python : + **206%**
- PHP : + **105%**
- Perl : + 89%
- Java : + 94%
- C# : + 84%

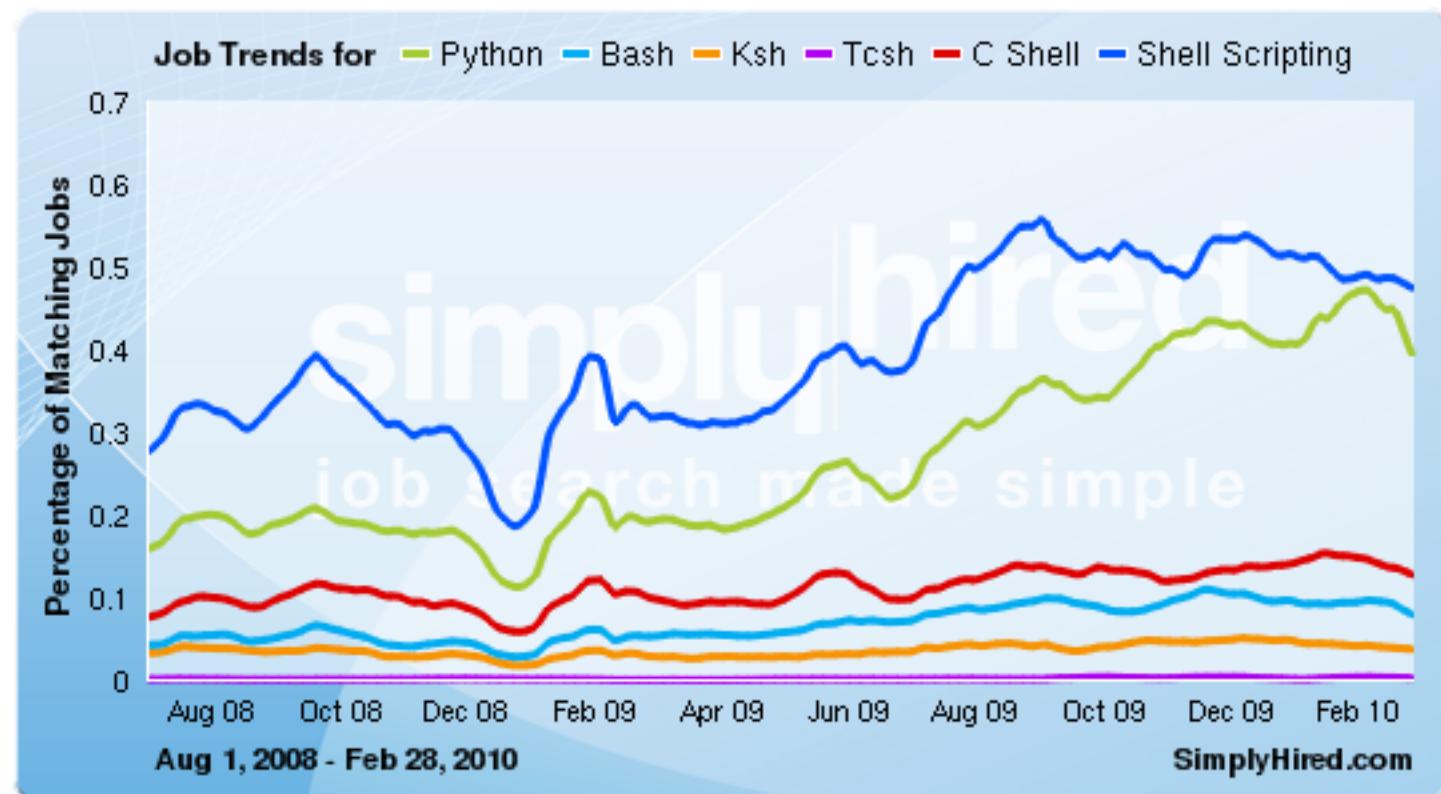


Est-ce que Python va dominer le monde?

Percentage of jobs in US job listings since August 2008

Langages de Programmation Shell

- Python : **+152%**
- Bash : +87%
- tcsh : + 135%
- Ksh : + 11%
- C shell: +69%
- Shell scripting:
+71%



Est-ce que Python va dominer le monde?

Percentage of jobs in US job listings since August 2008

Langages tout court ;-)

- Python : + **206%**
- French : + 36%
- English : + 45%
- German : + 21%



Plan de la présentation

1. Qu'est-ce-que Python ?
2. Brève introduction à la syntaxe
3. Python pour la science : libraries et outils



I. Qu'est-ce-que Python ?

Python est un langage...

- stable et mature : v. 1.0 en 1994
- activement développé : v. 2.7 en 2010
- gratuit et libre de droits
- disponible sous Linux (en standard), Mac OS X (en standard), Windows, etc.
- à la fois simple à apprendre, et très extensible
- doté de nombreuses librairies
- utilisé dans de nombreux domaines: programmation système, réseaux, automatisation, ingénierie, serveurs web, bases de données, images de synthèse, manipulation de texte, jeux vidéos, data mining, applications commerciales...
... et en sciences



Python est un langage...

- interprété - sans compilation
 - plus lent qu'un langage compilé
 - ~ équivalent en lecture/écriture
 - développement plus rapide, plus souple
 - adapté à un processus itératif
- avec gestion automatique de la mémoire
 - pas de allocate/deallocate, malloc/dealloc
- à la syntaxe claire et lisible
 - facile à apprendre, maintenir, partager
 - tout le monde n'est pas d'accord
- disponible en standard sous linux, Mac OS X



Python est polyvalent

- programmation shell et système
- analyse scientifique : statistique, Fourier, réseaux de neurones...
- parallel processing
- lecture/écriture
- utilisation interactive ou automatisée
- traitement d'images
- création de plots complexes
- ...



Python est extensible

- autorise l'écriture de scripts simples, quelques lignes
- mais supporte la création d'architectures logicielles complexes
 - Twisted : ~ 300 000 lignes de code (network framework)
 - Zope : ~ 620 000 (web app framework)
 - Plone : ~ 210 000



Python est flexible

Python s'adapte à plusieurs styles de programmation

- séquentielle
- procédurale
- orientée objet
- test-driven
- bête/intuitive (populaire dans les labos)



Distributions Python

- python.org = python + librairies standard
 - numpy, etc à installer soi-même : complexe
- distributions "intégrées" python scientifique
 - python + librairie standard + numpy + lecture/écriture + etc.
 - plus facile, à privilégier si possible
 - vérifier la license... (parfois payant)
- Python-CDAT
- SAGE (calcul formel) <http://www.sagemath.org>
- Enthought Python Distribution (conseillé) <http://www.enthought.com>
- python(x,y) <http://www.pythonxy.com>



II. La syntaxe : brève intro

Généralités (1)

- La commande **python** seule lance l'interpréteur Python (utilisation interactive, similaire à Matlab)
- chaque ligne entrée est exécutée immédiatement
- les variables restent en mémoire pendant la session
- Pratique pour calculs, tests, mise au point de code

```
~ > python
Enthought Python Distribution -- http://www.enthought.com
Version: 6.1-1 (32-bit)

Python 2.6.4 |EPD 6.1-1 (32-bit)| (r264:75706, Dec 11 2009, 10:58:54)
[GCC 4.0.1 (Apple Inc. build 5465)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> x = 3      • La commande 'python' seule lance l'interpréteur
>>> 4*x        Python (utilisation interactive, similaire à Matlab)
12
>>> █
```



Généralités (2)

- Le code python est non compilé (=script), un programme Python est donc un fichier texte ASCII
- On le distingue par l'extension .py (convention)
 - exemple : script.py
- On le lance de 2 façons
 - python script.py
 - ./script.py (après un chmod +x)



Généralités (3)

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

# commentaire
print 'Hello ' + "World!"

# exemple de boucle for
for i in [0, 1, 2, 3]:
    j = i * 2
    print 'i = ', i, ' et j = ', j
    if i == 2:
        print 'Ici i == 2.'

list_of_words = ['Hello', 'Word', 'Again']
for word in list_of_words:
    print word

print "Goodbye World!"
```



- le blanc est significatif : l'indentation détermine la logique (améliore la lisibilité par 10)
- Pas de fin de ligne
- Pas besoin de déclarer les variables
- Python devine leur type, et les élimine de la mémoire quand elles ne sont plus utilisées



Généralités (4)

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

# commentaire
print 'Hello ' + "World!"

# exemple de boucle for
for i in [0, 1, 2, 3]:
    j = i * 2
    print 'i = ', i, ' et j = ', j
    if i == 2:
        print 'Ici i == 2.'

list_of_words = ['Hello', 'Word', 'Again']
for word in list_of_words:
    print word

print "Goodbye World!"
```

- indispensable : indique quel interpréteur python utiliser (ici celui par défaut)

Généralités (5)

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

# commentaire
print 'Hello ' + "World!"

# exemple de boucle for
for i in [0, 1, 2, 3]:
    j = i * 2
    print 'i = ', i, ' et j = ', j
    if i == 2:
        print 'Ici i == 2.'

list_of_words = ['Hello', 'Word', 'Again']
for word in list_of_words:
    print word

print "Goodbye World!"
```

- Commentaires définis par #
- Chaines définies par ' ou "(aucune différence)"



Généralités (6)

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

# commentaire
print 'Hello ' + "World!"

# exemple de boucle for
for i in [0, 1, 2, 3]:
    j = i * 2
    print 'i = ', i, ' et j = ', j
    if i == 2:
        print 'Ici i == 2.'

list_of_words = ['Hello', 'Word', 'Again']
for word in list_of_words:
    print word

print "Goodbye World!"
```

for itère sur les éléments d'une liste [] (cf la suite)

L'indentation délimite le contenu de la boucle (pas de enddo, endif, { et }...)

On itère sur tout type de liste

le nombre d'espaces de l'indentation est sans importance,
mais il faut rester cohérent dans un même fichier



Les variables (1)

Python choisit le type d'une variable en fonction de ce qu'on met dedans

Python comprend les types de variable classiques

<code>x = 1</code>	Entier entier / entier -> entier : $1/2 \rightarrow 0$
<code>x = 1.</code>	Flottant / Réel entier / flottant -> flottant : $1./2 \rightarrow 0.5$
<code>x = 'bla'</code>	Chaine de caractères $x[0] \rightarrow 'b'$
<code>x1 = [1, 2, 3] x2 = ['bla1', 'bla2'] x3 = ['bl', 1, x2]</code>	Listes de données, 1 dimension $x1[0] \rightarrow 1$ (cf la suite) Une liste peut contenir des données de n'importe quel type, même d'autres listes



Les variables (2)

Python comprend d'autres types plus exotiques qui ne seront pas utilisés aujourd'hui, mais qui sont très pratiques dans certaines situations

<code>x = (1, 2, 3) y = ('bla', 23, x)</code>	Tuples = "Read-only" lists
<code>x ={'prenom':'Albert', 'nom':'Einstein', 'age':42}</code>	Dictionnaires <code>x['nom'] -> Einstein</code>

- Les types peuvent être combinés à l'infini : dictionnaires de listes de strings et de variables numériques...



Les variables (3)

Python alloue automatique la mémoire pour stocker la variable,
et la libère lorsque la variable n'est plus utilisée

On peut libérer explicitement la mémoire :

del(x)

(Ça peut être utile un jour)

listes (1) : indices

- Une liste est un ensemble ordonné à 1 dimension...
 - $x = [4,3,-5,6,-7,8,9,10]$
- ...dont on extrait un ou plusieurs éléments par *indice*
 - $x[0] \rightarrow 4, x[3] \rightarrow 6, x[7] \rightarrow 10$
- Les indices commencent à 0, comme en C
 - $x = [\text{item}_0, \dots, \text{item}_{N-1}]$
 - Matlab, Fortran : indices commencent à 1
- Les indices négatifs comptent depuis la fin.
 - $x[-2] \rightarrow 9$
 - **En pratique**, surtout utile pour accéder au dernier élément par $x[-1]$



listes (2) : slicing (intervalles)

- On extrait plusieurs éléments à la fois en spécifiant un intervalle d'indices ("slicing") `x[start:stop]`
 - `x = [4,3,-5,6,-7,8,9,10]`
 - `x[0:5]` -> `[4,3,-5,6,-7]`
- Le dernier indice d'un intervalle est exclu : `[0:5)`
 - □ \neq Matlab, Fortran
 - `x[7]` -> 10
 - `x[0:7]` -> `[4,3,-5,6,-7,8,9]` (confusion)
 - `x[0:8]` -> `[4,3,-5,6,-7,8,9,10]`
- On peut omettre les extremes
 - `x[:5]=x[0:5]` = "5 premiers éléments"
 - `x[5:]=x[5:-1]` = "de l'élément 5 à la fin"



listes (3) : slicing

- $x = [4,3,-5,6,-7,8,9,10]$
- les indices d'un slice peuvent être négatifs
 - $x[-5:-2] \rightarrow [6,-7,8]$
 - (Si cela vous surprend, rappelez-vous que l'indice de droite est exclu de l'intervalle...)
- **En pratique**, surtout utile pour récupérer les n derniers éléments d'une liste, avec $x[-n:]$
 - $x[-5:] \rightarrow [6,-7,8,9,10]$
 - $x[-2:] \rightarrow [9, 10]$



listes (4) : slicing

- $x = [4,3,-5,6,-7,8,9,10]$
- Slicing avec incrément
 - $x[\text{start}:\text{stop}:\text{step}]$
 - $x[::2]=x[0:-1:2]= \rightarrow [4,-5,-7,9]$
 - l'incrément est à la fin (\neq Matlab : au milieu)
- Slicing avec incrément négatif
 - $x[8:0:-2] \rightarrow [10,8,6,3]$
 - équivalent à $x[:: -2]$
 - **En pratique**, surtout utile pour inverser une liste entière avec $x[:: -1]$



listes (5) : ce qu'il faut vraiment retenir de l'indexation...

- $x[0]$ = premier élément
- $x[-1]$ = dernier élément
- $x[-3:]$ = 3 derniers éléments
- $x[:3]$ = 3 premiers éléments
- $x[::-1]$ = liste inversée

listes (6) : fonctions utiles

Python procure beaucoup de fonctions utiles pour manipuler des listes

x=[5,2,8,3]

sort(x) -> x=[2,3,5,8]

x=['ba', 'ac', 'ab']

sort(x) -> x=['ab', 'ac', 'ba']

append(x, 'ba') -> x=['ab', 'ac', 'ba', 'ba']

Voir aussi : index, count, reverse, remove, extend...

Attention : Ces fonctions sont spécifiques aux listes python standard, numpy se comporte différemment (cf la suite)

Flow control (1)

- Toute commande de contrôle (for, while, if) est suivie de deux points ":" et d'un bloc indenté
- **for** itère sur les éléments d'une liste

```
listex = [0,1,2,3]
```

```
for x in listex:
```

```
    print 'x*2 = ', x*2
```

```
files = ['d1.nc', 'd2.nc', 'd3.nc']
```

```
for file in files:
```

```
    # do stuff with files
```

- Astuce : une chaîne est une liste

```
for char in "abcd":
```

- **while** itère tant qu'une condition est vraie

```
while x < 10:
```

```
    x = x + 2
```



Flow control (2)

if, elif, else vérifient une ou plusieurs condition(s)

```
if x < 2:  
    print "x est petit"  
  
elif x < 4:  
    print "x est moyen"  
  
else:  
    print "x est grand"
```



Flow control (3)

Exemples de conditions

- if x:
- if not x:
- if x==y:
- if x!=y: ou if x<>y: ou if not x==y:
- if x<y:
- if x<y<z: (équivalent à if x<y and y<z:)
- if x in liste:
- if x not in liste:
- if x is True: (équivalent à if x:)



Exceptions (1)

En Python, les erreurs déclenchent des exceptions

```
~ > python LMD - CNRS tmpx Read Later G reader G mail G Docs Simplenote  
Enthought Python Distribution -- http://www.enthought.com G Docs  
Version: 6.1-1 (32-bit) Web more ▾  
  
Python 2.6.4 |EPD 6.1-1 (32-bit)| (r264:75706, Dec 11 2009, 10:58:54)  
[GCC 4.0.1 (Apple Inc. build 5465)] on darwin  
Type "help", "copyright", "credits" or "license" for more information.  
>>> a='abcd'  
>>> a + 3  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: cannot concatenate 'str' and 'int' objects  
>>> █ raw control [2]
```

- Une exception interrompt le déroulement du programme
- Dans la plupart des cas, une exception indique une erreur de programmation

Exceptions (2)

On peut "attraper" les exceptions avec try... except :

```
try:  
    # actions  
except NomException:  
    # actions en cas d'erreur
```

Similaire à try... catch en Matlab

except sans nom d'exception attrape toutes les exceptions : à utiliser **avec précaution**, car on court le risque d'ignorer des erreurs

Exceptions (3)

Une variante de except permet de récupérer le message d'erreur pour l'afficher :

```
try:  
    # actions  
except Exception, msg:  
    print "Erreur : ", msg
```

Utile pour débugger tout en évitant un plantage

Encore une fois : à utiliser **avec précaution**, mais très utile si vous savez ce que vous faites

Les fonctions

```
def multiply_by_two(x, and_add_one=False):
    xprime = x * 2.
    if (and_add_one is True):
        xprime = xprime + 1
    return xprime

y = multiply_by_two(3)                      # y = 6
y = multiply_by_two(3, and_add_one=True)     # y = 7
y = multiply_by_two(3, True)                 # y = 7
```

- On définit une fonction par **def** fonction(args):
- retourne n'importe quel type de variable (dont listes)
- and_add_one: argument avec **valeur par défaut**
 - facultatif lors de l'appel de la fonction
 - utile pour des fonctions à beaucoup d'argument
 - utile pour des arguments facultatifs ou rarement utilisés



Objets (1)

- Python permet la Programmation Orientée Objet (POO)
- La POO est très utile pour stocker des propriétés similaires (min, max, médiane, moyenne, standard déviation, contrôle qualité des données...) pour plusieurs variables différentes.
- On crée un objet qui contient ces résultats par **class** :

```
class GridData:
```

```
    def __init__(self):  
        self.data=[]  
        self.mean=[]  
        self.min=[]  
        self.max=[]  
        self.flag_qc=[]
```

```
    ...
```

Objets (2)

On crée des variables de **classe** GridData par:

```
temp=GridData # temp et hmdt sont des objets
hmdt=GridData # de la classe GridData
```

Ces variables contiennent ensuite les mêmes propriétés, on y accède par ".", e.g. temp.flag_qc :

```
temp.mean.append(mean(data_une_heure))
temp.min.append(min(data_une_heure))
temp.max.append(max(data_une_heure))
temp.flag_qc.append(flag_de_controle_qualite)
```

```
hmdt.mean.append(mean(data_une_heure))
```

...

Objets (3)

On peut ensuite appliquer le même traitement à toute variable de la même classe (e.g. temp, hmdt de classe GridData). Par exemple en utilisant un dictionnaire **dict_param** et une liste de propriétés **lst_param**:

```
lst_param=['temp','hmdt','ws']

for param in lst_param:
    # creation d'un objet
    dict_param[param]=GridData
    #calcul
    dict_param[param].mean.append(
        mean(data_une_heure))
```

Objets (4)

Pour info, toute variable Python est en réalité un objet, qui possède des propriétés et fonctions associées :

```
str = 'Hello World!'
str.split()    -> ['Hello', 'World!']
str.split('o') -> ['Hell', ' W', 'rld!']
'Hello World!'.split() # valide
```

```
x = [4,1,3,'bla', 3]
x.count(3)      -> 2
x.index('bla') -> 3
```

Découvrir les propriétés et fonctions d'un objet est simple en utilisant la fonction de complétion d'ipython (cf part 3)

☐ À noter (pour finir)

Pour Python, une variable est d'abord son contenu,
le nom de la variable n'en est qu'un attribut

a=[0,1,2,3]	
b=a	b [0,1,2,3] a
b[1]=5	b [0,5,2,3] a

L'instruction "b=a" donne juste un deuxième nom à la meme variable (ici la liste [0,1,2,3])

a=[0,1,2,3]	
b=a[:]	b -> [0,1,2,3] a -> [0,1,2,3]
b[1]=5	b -> [0,5,2,3] a -> [0,1,2,3]

Si on souhaite une nouvelle liste, il faut demander explicitement sa création par "b=a[]" (ou "b=a.copy()" si a est un tableau numpy)

Suivant votre style de programmation, il faudra faire plus ou moins attention à ce comportement



Ouf
10 minutes de pause !
Questions ?

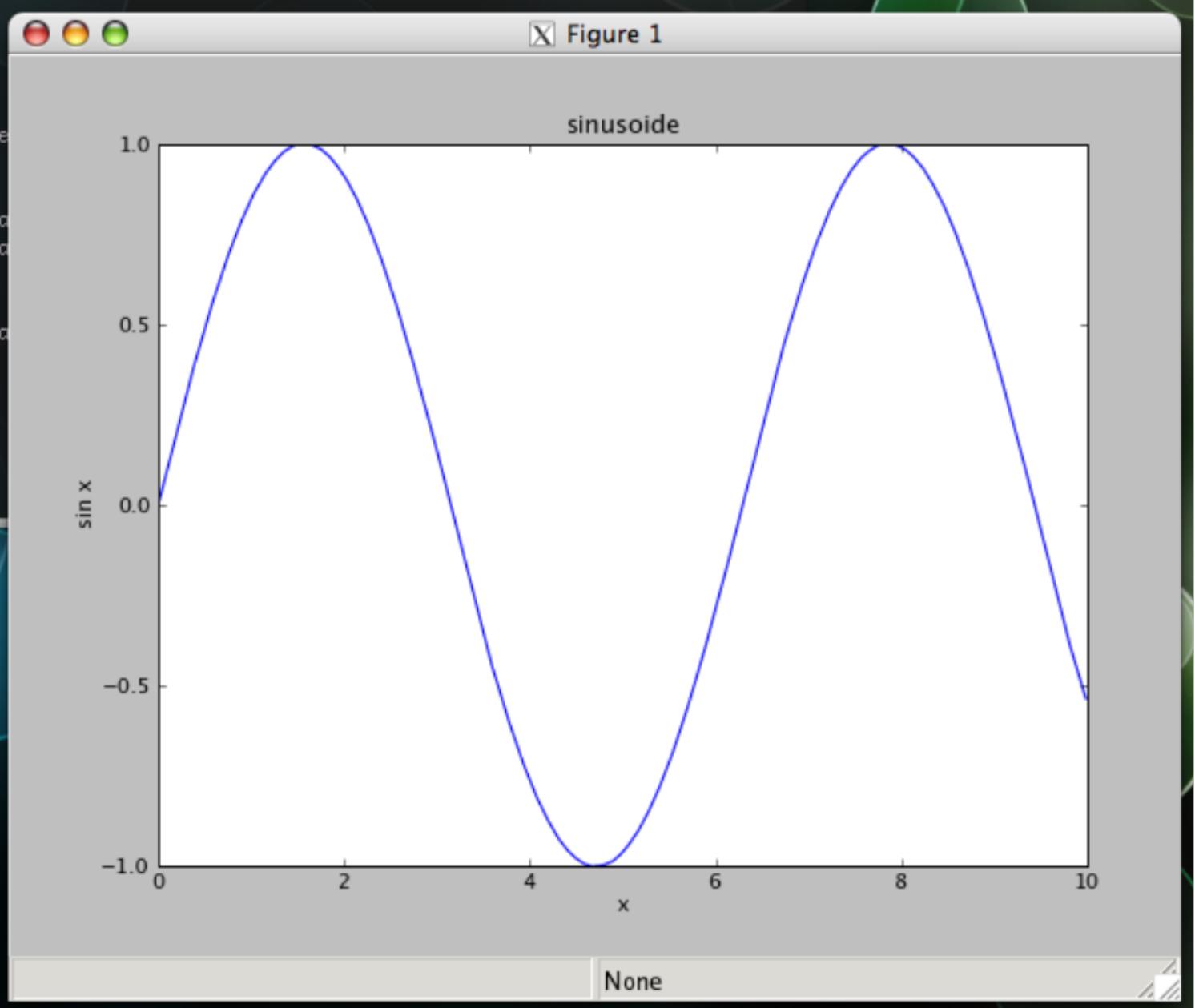
III. Python pour la science : outils et librairies

1. IPython

- L'interpréteur Python de base est pratique mais un peu limité (~ interpréteur IDL) : pas de cd, ls, rappels de lignes, etc.
- **IPython** : un interpréteur Python amélioré
 - complète les commandes, fichiers/dossiers avec Tab
 - aide intégrée : **help plot**
 - prévu pour l'affichage numérique et graphique interactif (cf la suite)
 - raccourcis à la Matlab : cd, ls, pwd, !mkdir, who
 - ça serait dommage de ne pas s'en servir
 - se lance par **ipython -pylab** (cf la suite)
- Astuce : ^d pour quitter



```
In [3]: x = r_[0:10:0.01]
In [4]: y = sin(x)
In [5]: plot(x, y)
Out[5]: []
In [6]: xlabel('x'); ylabel('sin x')
Out[6]: <matplotlib.text.Text object at 0x...>
Out[6]: <matplotlib.text.Text object at 0x...>
In [7]: title('sinusoide')
Out[7]: <matplotlib.text.Text object at 0x...>
In [8]: savefig('sinusoide.png')
In [9]:
In [10]: []
```



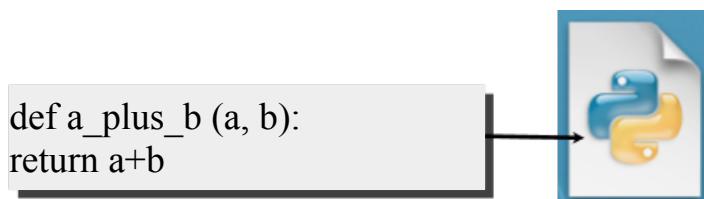
2. Modules (1)

- les fonctions supplémentaires de python proviennent de **modules** (~= modules Fortran, toolbox Matlab)
- Un module est un fichier python normal contenant fonctions, variables, objets, etc.
 - **tout script python est un module**
- Le nom du fichier est le nom du module
- On importe le module par
import module
- On appelle ensuite une fonction du module
module.fonction()



2. Modules (2)

- Il est simple de créer son propre module, qu'on peut ensuite importer comme les modules standards



```
~ > python
Enthought Python Distribution -- http://www.enthought.com
Version: 6.1-1 (32-bit)

Python 2.6.4 |EPD 6.1-1 (32-bit)| (r264:75706, Dec 11 2009, 10:58:54)
[GCC 4.0.1 (Apple Inc. build 5465)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import mon_module
>>> mon_module.a_plus_b(4,3)
7
>>> 
```



2. Modules (3)

- Il existe plusieurs variantes de l'importation

commande d'importation	Appel de a_plus_b()
import mon_module	mon_module.a_plus_b()
import mon_module as mon (conseillé)	mon.a_plus_b()
from mon_module import a_plus_b	a_plus_b()
from mon_module import * à éviter dans les scripts (collisions de noms possible)	a_plus_b()



2. Modules (4)

- pour info : un module peut contenir d'autres modules
 - les règles précédentes s'appliquent
- exemple : scipy contient les modules io, stats, etc.
 - import scipy.stats
scipy.stats.nanmean()
- d'où l'intérêt de **import ... as ...**
 - import scipy.stats as st
st.nanmean()



2. Modules (5) : exemple

- Python fournit beaucoup de modules en standard, qui constituent la Python Standard Library
- des modules existent pour de nombreuses tâches : manipulation de chaînes, transferts réseaux, serveurs web, lecture/écriture, analyses variées

2 exemples : **os**, **filecmp**

Modules dédiés à la manipulation de répertoires et fichiers, utiles pour une utilisation "script shell"

- `os.path` - manipulation des chemins d'accès
- `os.shutil`, `filecmp` - manipulation des fichiers

Exemple:

```
import os  
os.path.basename()
```

3. Modules Scientifiques

Ce matin : focus sur 2 modules importants

- **numpy**
- **matplotlib**

Nous survolerons brièvement

- **scipy**
- **netCDF4**
- **Basemap**
- **Bases de données**



3.1 numpy

Par défaut, Python ne gère pas bien les nombres

```
~ > python
LMD ✓ CNRS ✓ tmp ✓ Read Later G reader G mail G Docs Simplenote
Enthought Python Distribution -- http://www.enthought.com G Docs
Version: 6.1-1 (32-bit) Web more ▾

Python 2.6.4 |EPD 6.1-1 (32-bit)| (r264:75706, Dec 11 2009, 10:58:54)
[GCC 4.0.1 (Apple Inc. build 5465)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> x = [0,1,2,3,4]
>>> x * 2
[0, 1, 2, 3, 4, 0, 1, 2, 3, 4]
>>> █ Modules (4)
```

- Ne fait pas les opérations sur les éléments de la liste
- Listes limitées à 1 dimension
- Calculs par boucles, lents
- Peu adapté au travail scientifique



3.1 numpy



- numpy : module développé depuis 1995
 - descendant de Numeric et numarray
 - v.1.3 (2.0 en 2010 = standardisation)
- type "tableau de type fixe" à n dimensions
 - `x = array([0, 1, 2, 3])`
 - raccourcis : `x = r_[0, 1, 2, 3]`, `x = r_[0:4]`
 - optimisé pour la manipulation numérique
 - librairie compilée = très rapide
- la base pour scipy, matplotlib et la plupart des modules de lecture/écriture de formats scientifiques
 - ces modules créent des variables de type "array"
 - leur création manuelle est donc souvent évitée



3.1 numpy

- IPython importe tout numpy avec l'option -pylab

```
~ > ipython -pylab
Enthought Python Distribution -- http://code.enthought.com
Slide 51 of 76
Python 2.6.4 |EPD 6.1-1 (32-bit)| (r264:75706, Dec 11 2009, 10:58:54)
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.

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

In [1]: x = array([0, 1, 2, 3, 4])
Out[1]: array([0, 1, 2, 3, 4])

In [2]: x * 2
Out[2]: array([0, 2, 4, 6, 8])

In [3]: 
```

- Convention "officielle" : **import numpy as np**



3.1 numpy

On indexe un tableau numpy comme une liste, avec la différence que plusieurs dimensions sont possibles.

Quelques exemples :

- `x = r_[0:2*pi:0.01]`
- `x[0], x[50:], x[:-10]`
- `x[:, :, :, 3] -> x[..., 3]`
- `x[x < 3] = 0`
- `x=x.T # transpose, x=x' en Matlab`
- `x=a*b # x=a.*b en Matlab`
- `x**3 # x^3 en Matlab`



3.1 numpy : manipulation de tableaux

np. zeros ([d1,d2,...]) np. ones ([d1,d2,...])	crée des tableaux de 0 ou 1, de dimensions d1,d2...
np. shape (a1) np. reshape (a1, [...])	Dimensions du tableau a1 redimensionne le tableau a1
np. concatenate ([a1,a2,...],axis=n)	Concatène des tableaux le long de la dimension n
np. mean (a1), np. std (a1) np. min (a1), np. max (a1)	moyenne, écart-type, min et max du tableau a1 axis=n travaille le long de la dimension n
np. sin (), np. cos (), np. tan (), np. transpose (), np. flipud (), np. tile ()...	



3.1 numpy sait faire autre chose

- Lecture/écriture de formats simples
 - **np.loadtxt()**, **np.savetxt()** - ASCII
 - np.load(), np.save() - format numpy "npz", utile pour fichiers temporaires
- Calcul matriciel
 - np.matrix()
- Interpolation
 - np.interp()
- Histogrammes à 1, 2, n dimensions
 - np.histogram(), np.histogram2d(), np.histogramdd()
- Masked arrays
 - arrays avec valeurs invalides (module numpy.ma)



3.2 Matplotlib



- Beaucoup de modules Python pour faire des plots
 - PyNGL, Chaco, Veusz, gnuplot, Rpy, Pychart...
- Matplotlib émerge comme un "standard"
 - "all-purpose" plot package, interactif ou publication-ready
 - EPS, PNG, TIFF
 - forte interaction avec numpy, scipy
 - extensible, populaire, stable
 - "*Matplotlib tries to make easy things easy and hard things possible*"
 - utilisation "matlab-like" ou orientée objet



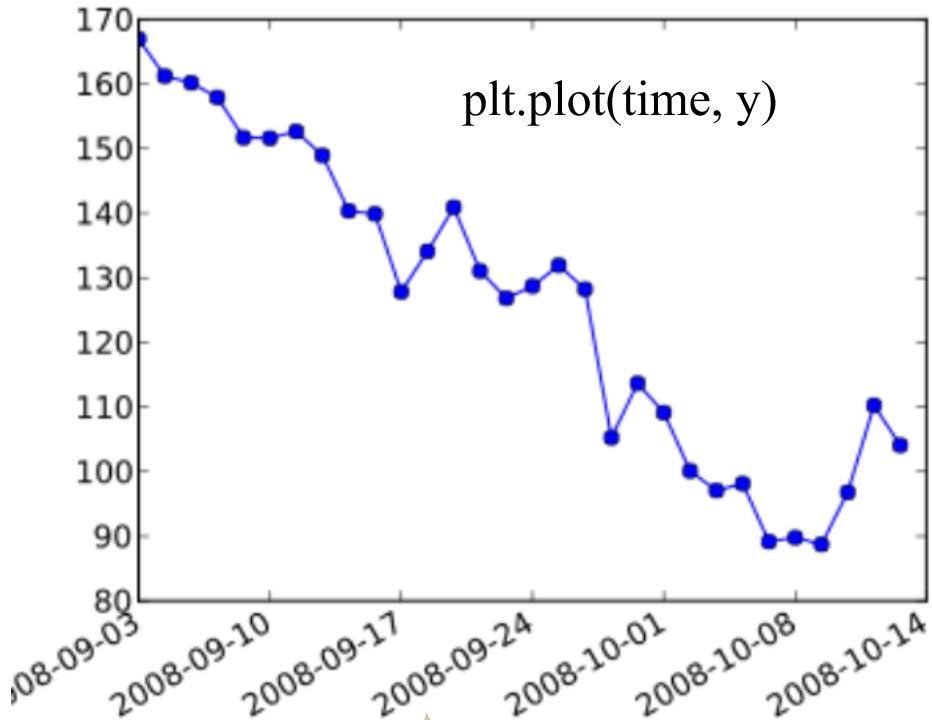
3.2 Matplotlib et Matlab

- Le module `pyplot` de Matplotlib offre une large série de commandes à la syntaxe volontairement proche de Matlab
 - `plot`, `semilogx`, `semilogy`, `pcolor` (shading flat), `xlabel`, `ylabel`, `title`, `legend`, `hist`, `figure`, `axis`, `subplot`, `contour`, `colorbar`, `quiver`, `axes`, `xlim`, `ylim`, `xticks`, `yticks`...
- Interactif : IPython importe Matplotlib.pyplot avec l'option `-pylab`
- Scripts : convention "officielle"
`import matplotlib.pyplot as plt`



3.2 Matplotlib : exemples (1)

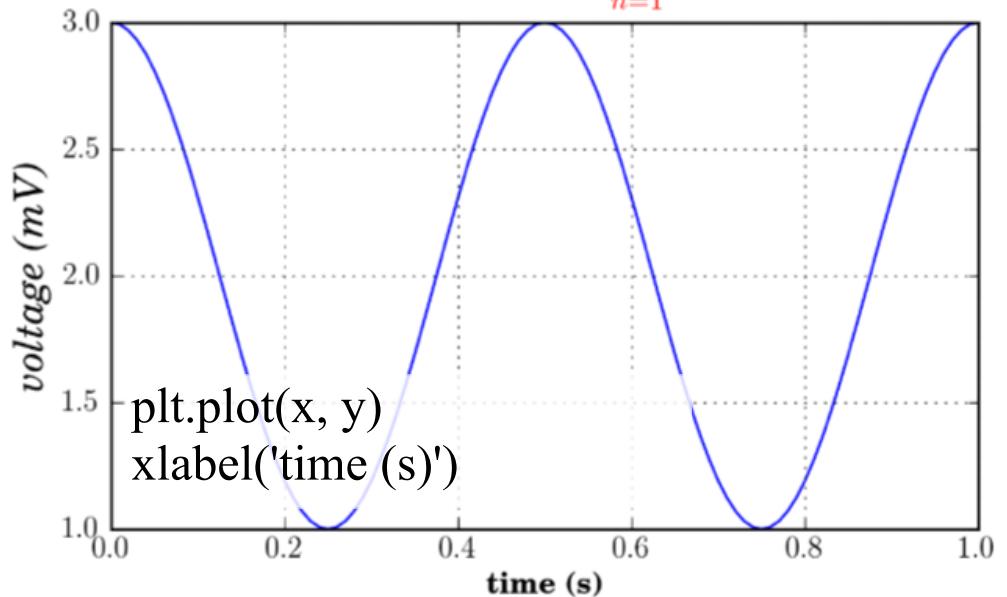
- Séries Temporelles simples



Python et matplotlib
comprennent les dates

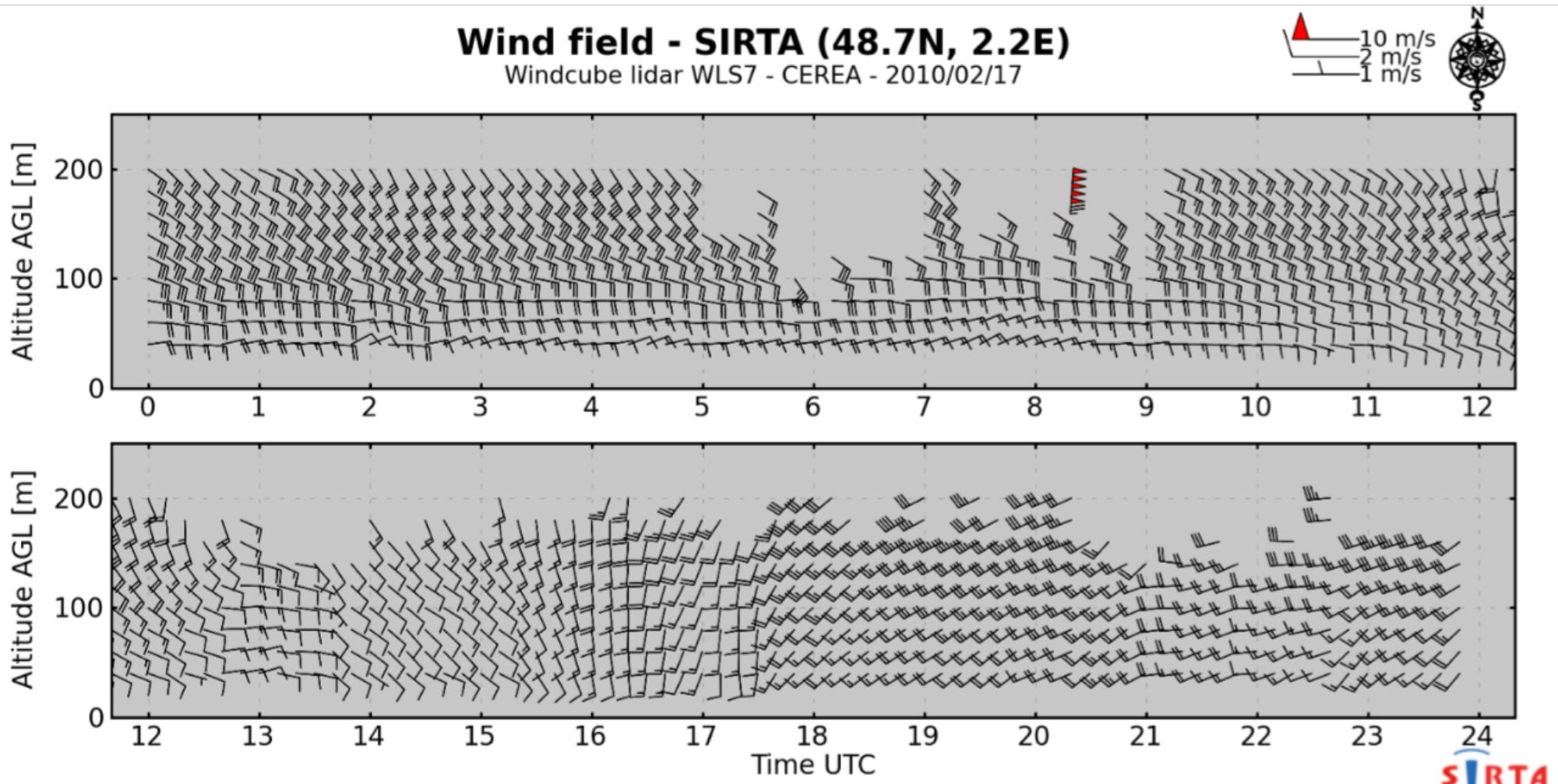
Matplotlib comprend Latex

TEX is Number $\sum_{n=1}^{\infty} \frac{-e^{i\pi}}{2^n}!$



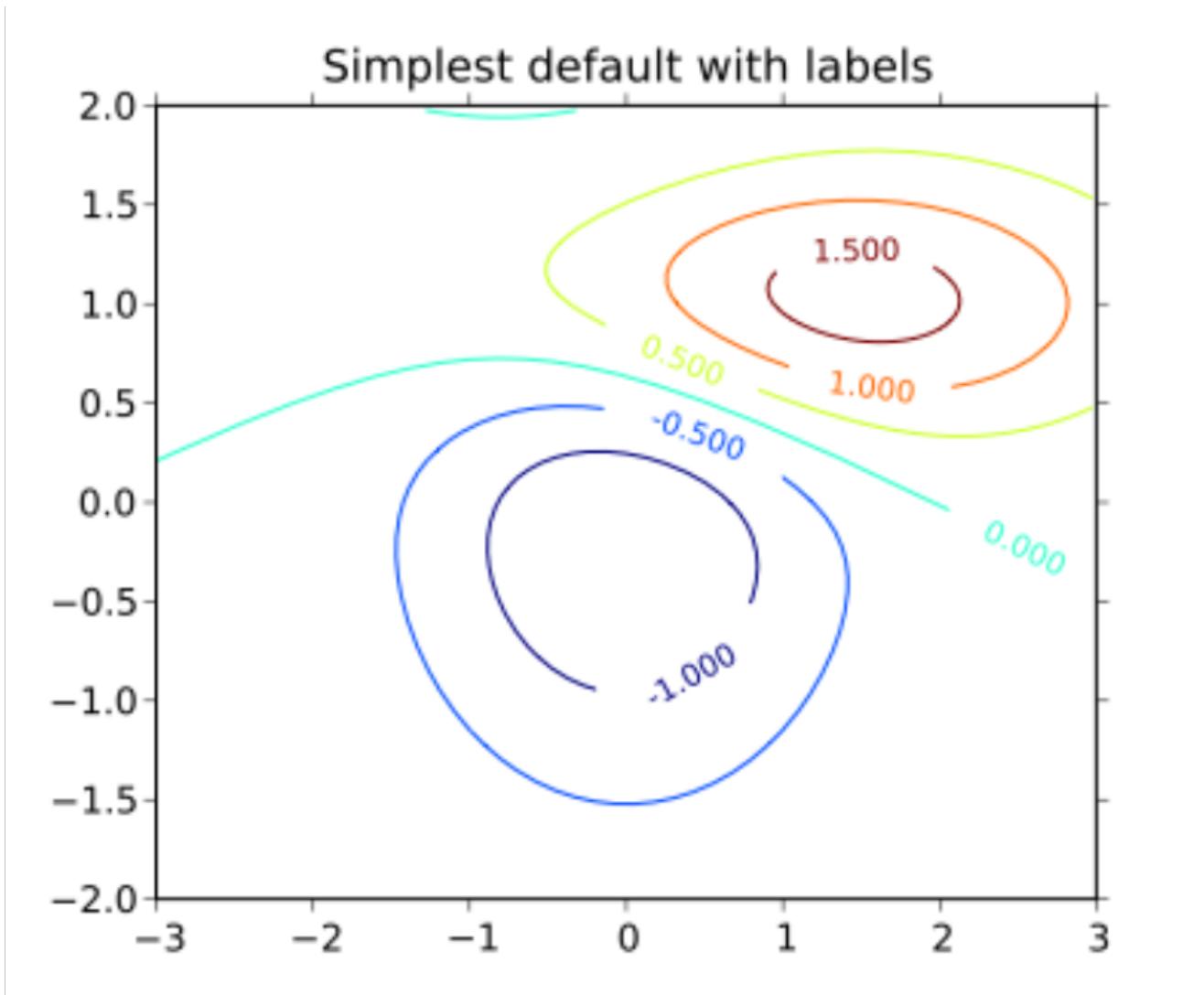
3.2 Matplotlib : exemples (2)

- Séries temporelles de profils verticaux de vent



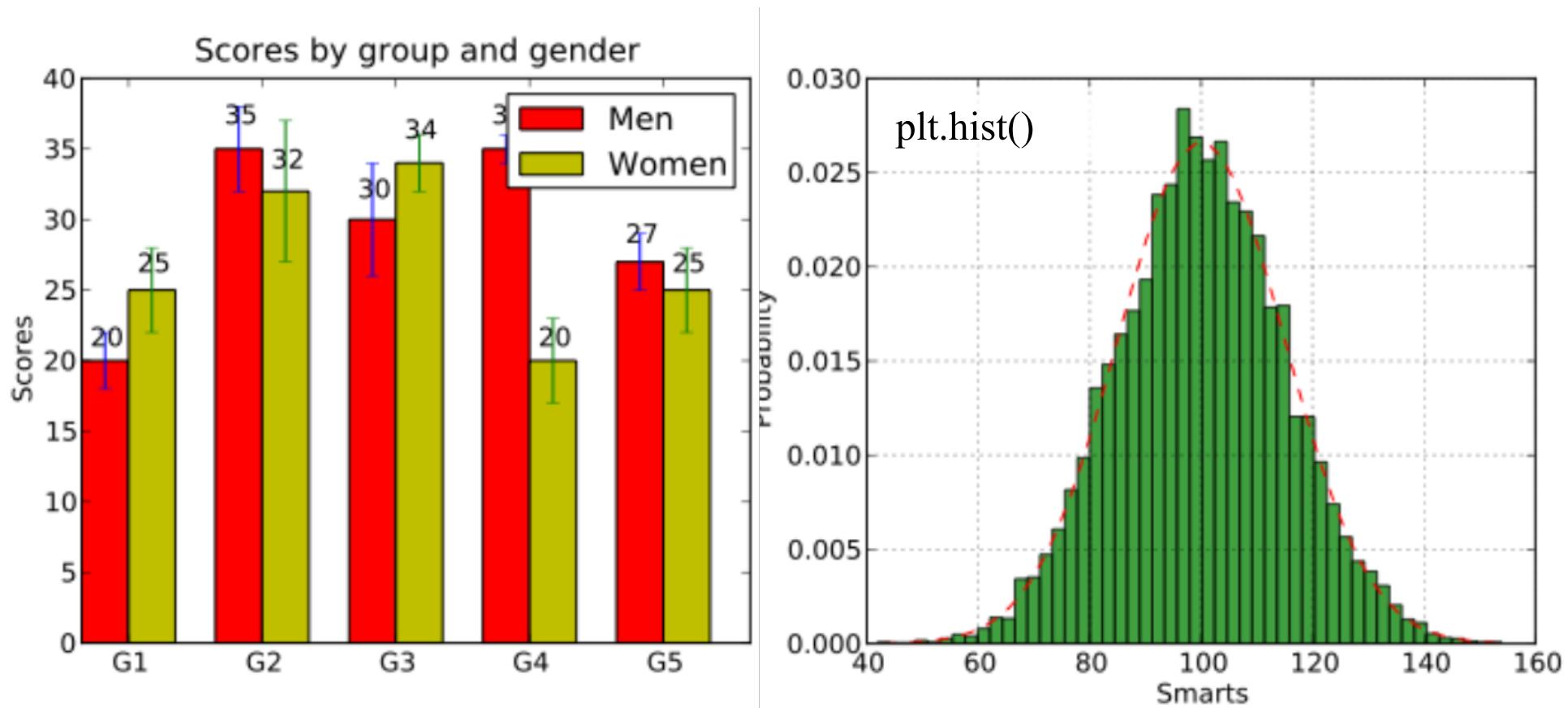
3.2 Matplotlib : exemples (3)

- Contours avec labels



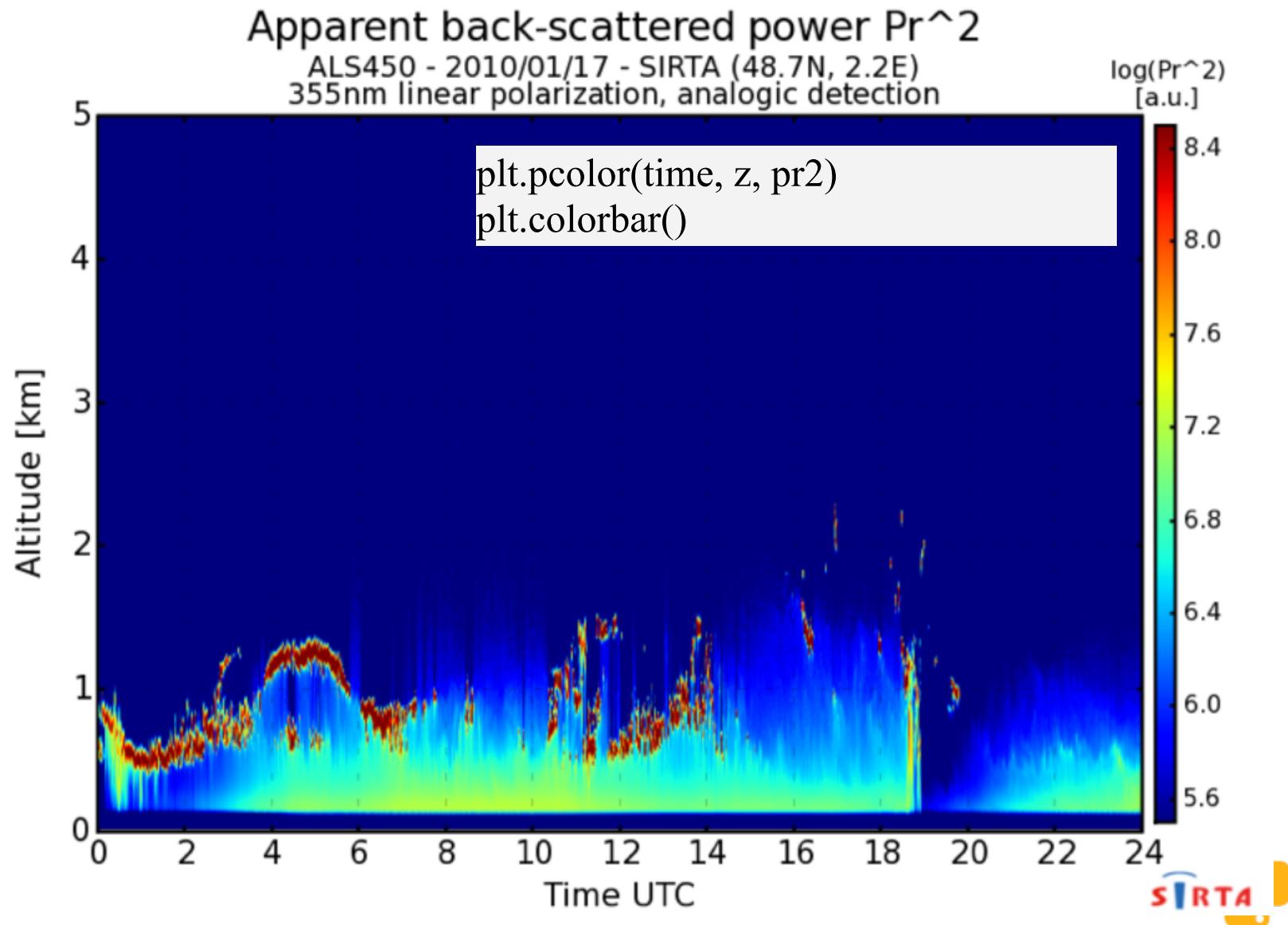
3.2 Matplotlib : exemples (4)

- Histogrammes



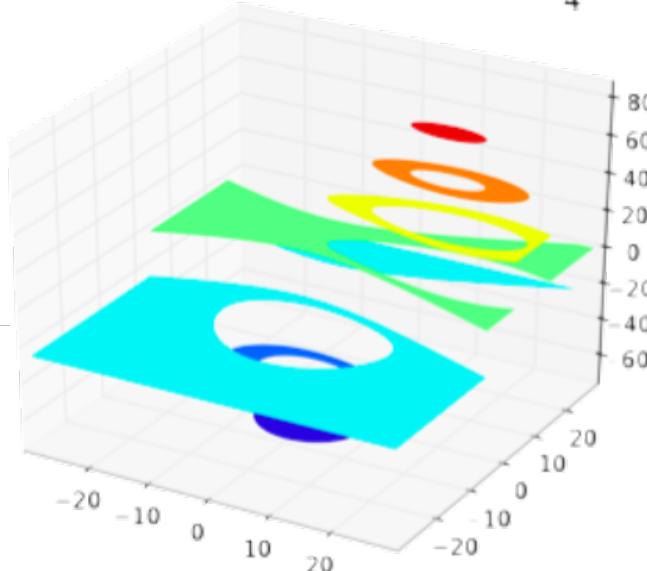
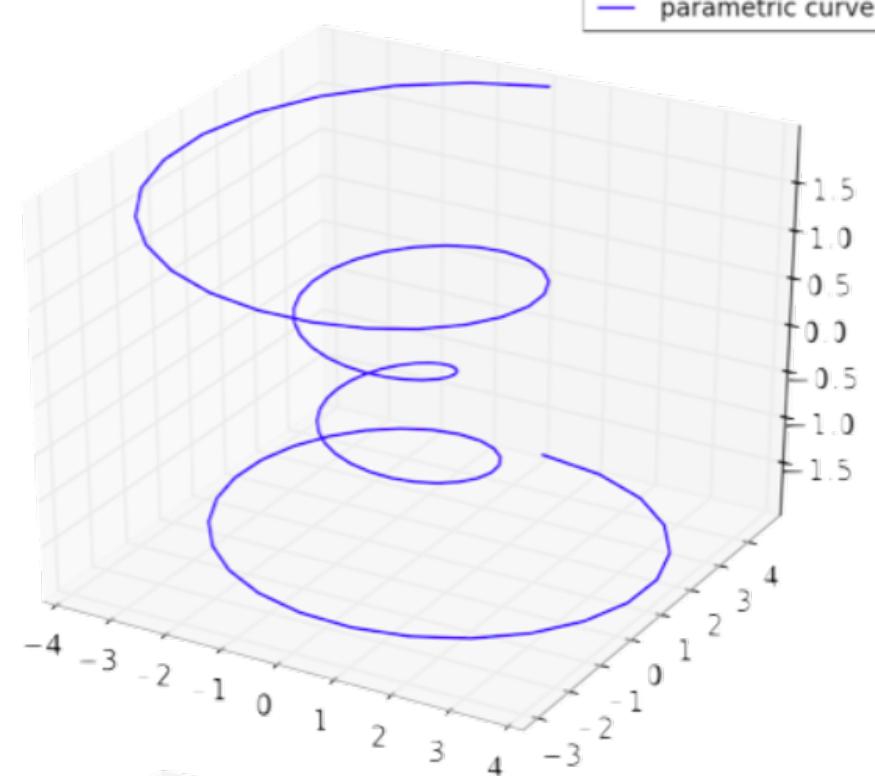
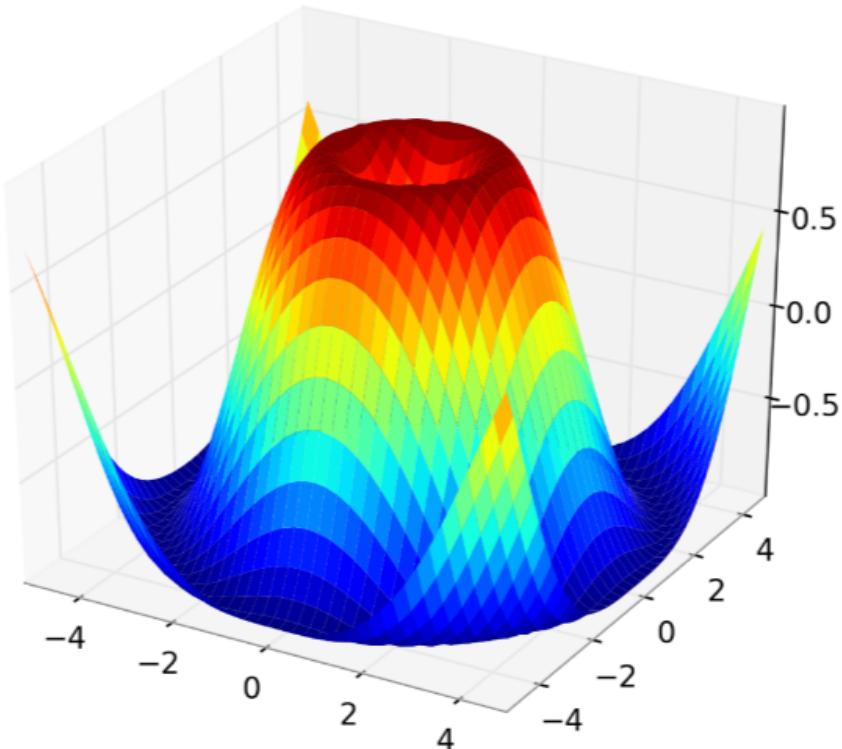
3.2 Matplotlib : exemples (5)

- time-range plot (pcolor en matlab) avec colorbar



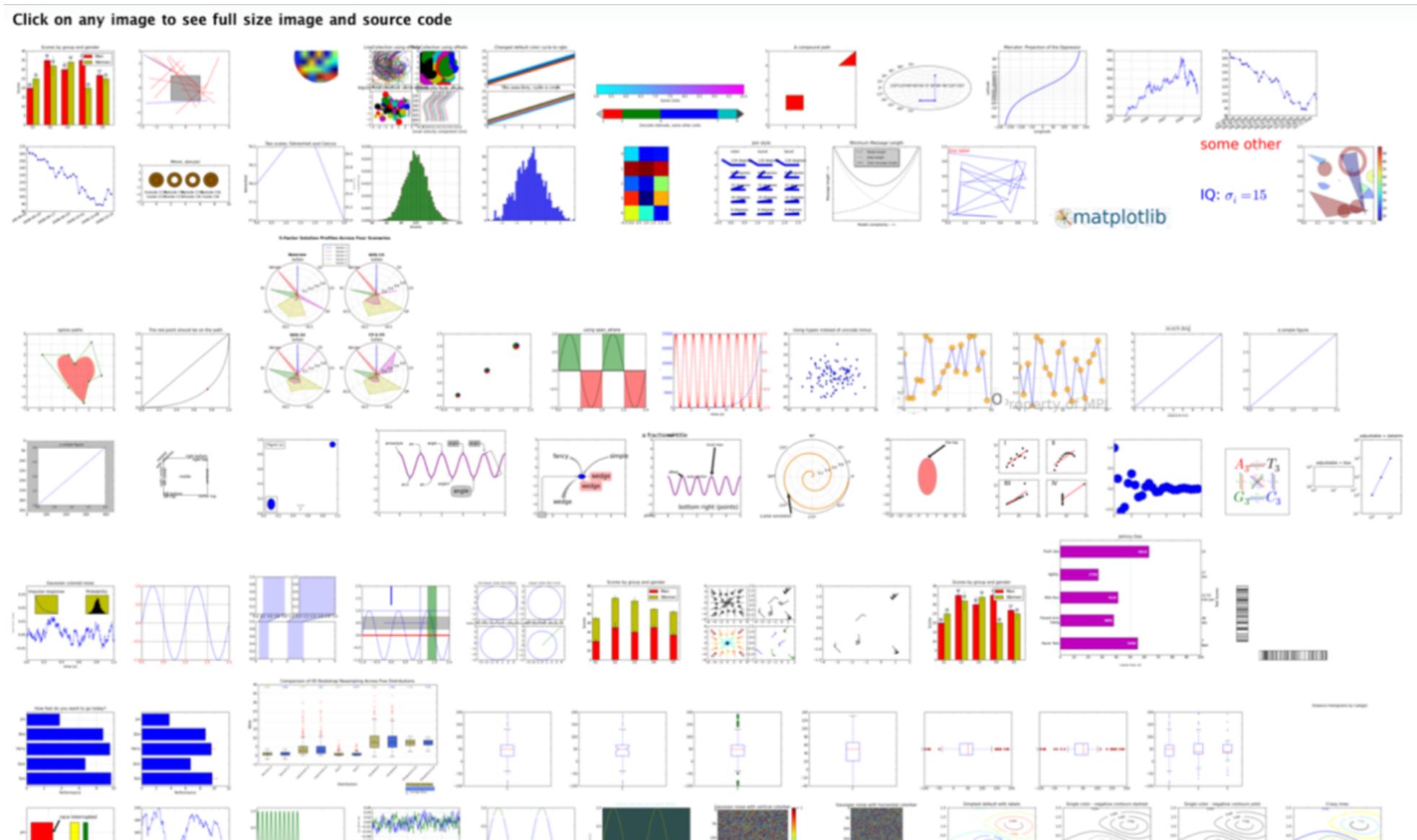
3.2 Matplotlib : exemples

- 3D plots (mplot3d Toolkit)



3.2 Matplotlib : etc.

- <http://matplotlib.sourceforge.net/gallery.html>



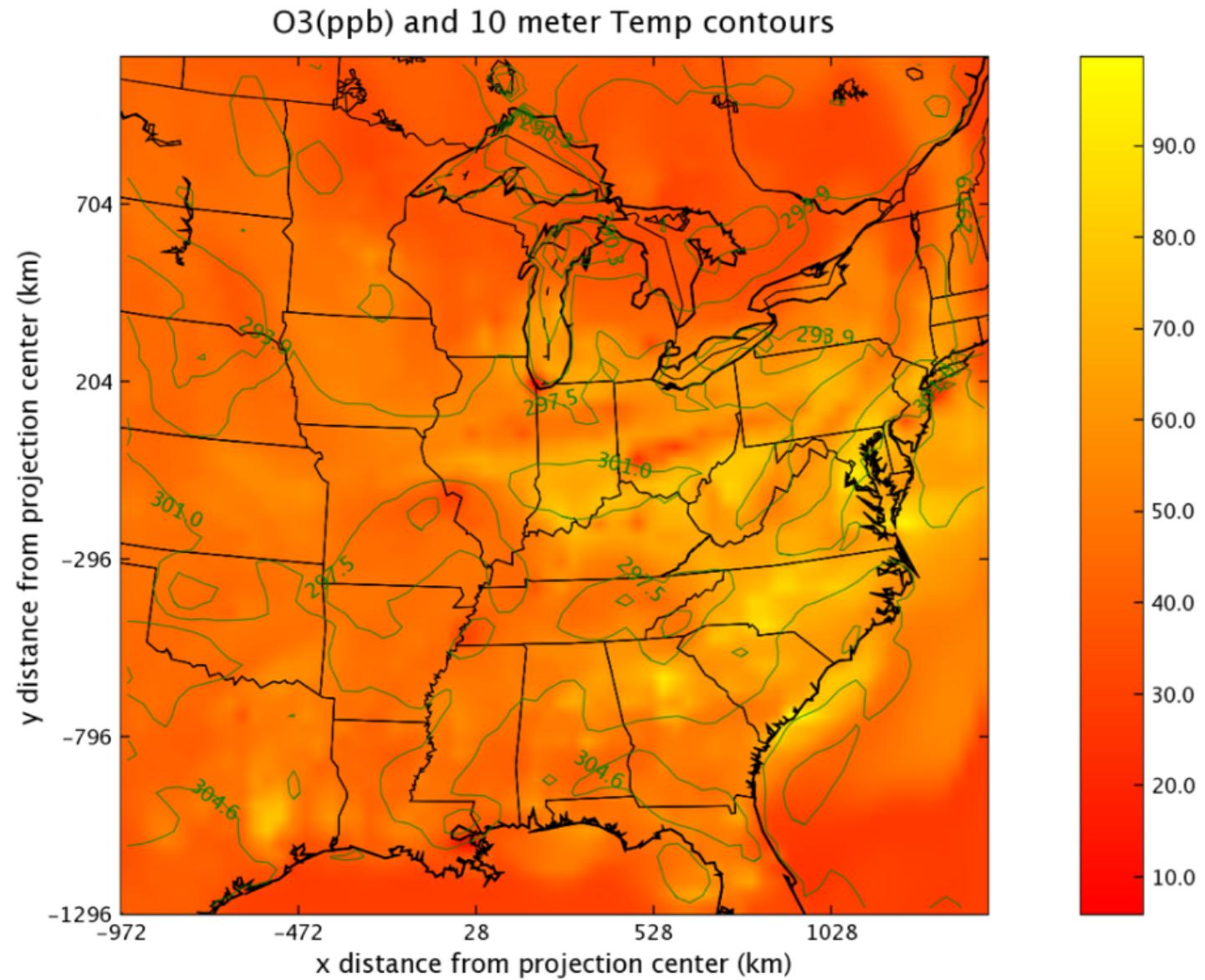
3.2 Matplotlib : Basemap

- from mpl_toolkits import basemap
- Toolkit de Matplotlib
- projections : (x, y) <-> (lon, lat)
- affichage en f(lon,lat)
 - plots
 - cartes
 - contours
 - cotes, frontières, géodésie

```
Azimuthal Equidistant
Polyconic
Gnomonic
Mollweide
Transverse Mercator
North-Polar Lambert Azimuthal
Gall Stereographic Cylindrical
Miller Cylindrical
Mercator
Stereographic
North-Polar Stereographic
Geostationary
van der Grinten
Lambert Azimuthal Equal Area
McBryde-Thomas Flat-Polar Quartic
Sinusoidal
South-Polar Stereographic
Lambert Conformal
North-Polar Azimuthal Equidistant
Equidistant Conic
Cylindrical Equidistant
Oblique Mercator
Albers Equal Area
South-Polar Azimuthal Equidistant
Orthographic
Cassini-Soldner
South-Polar Lambert Azimuthal
Robinson
```

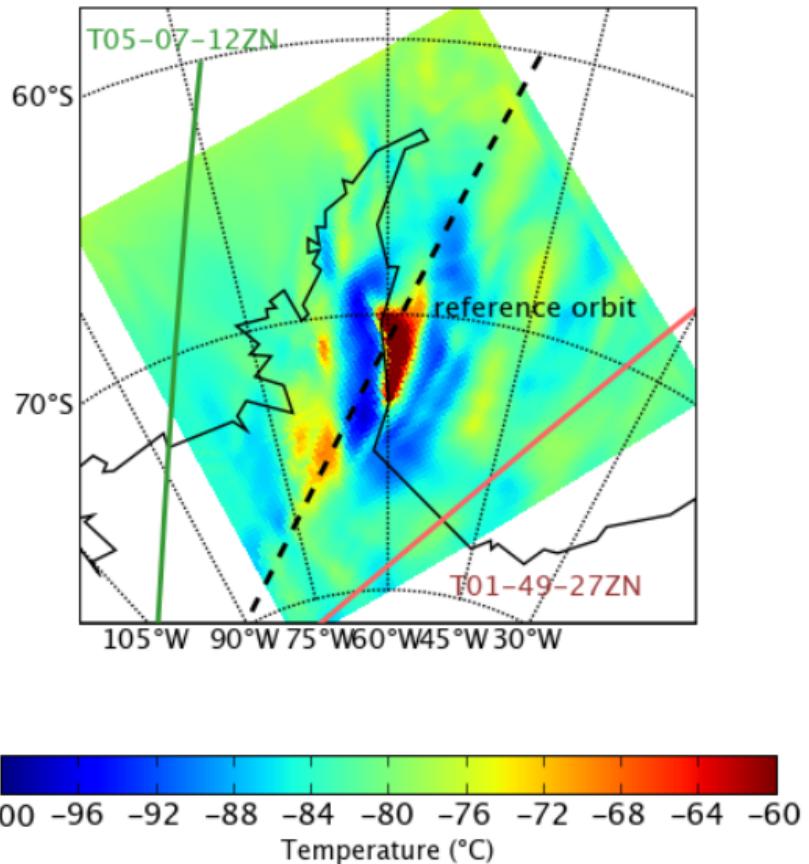


3.2 Matplotlib+Basemap : exemple (1)

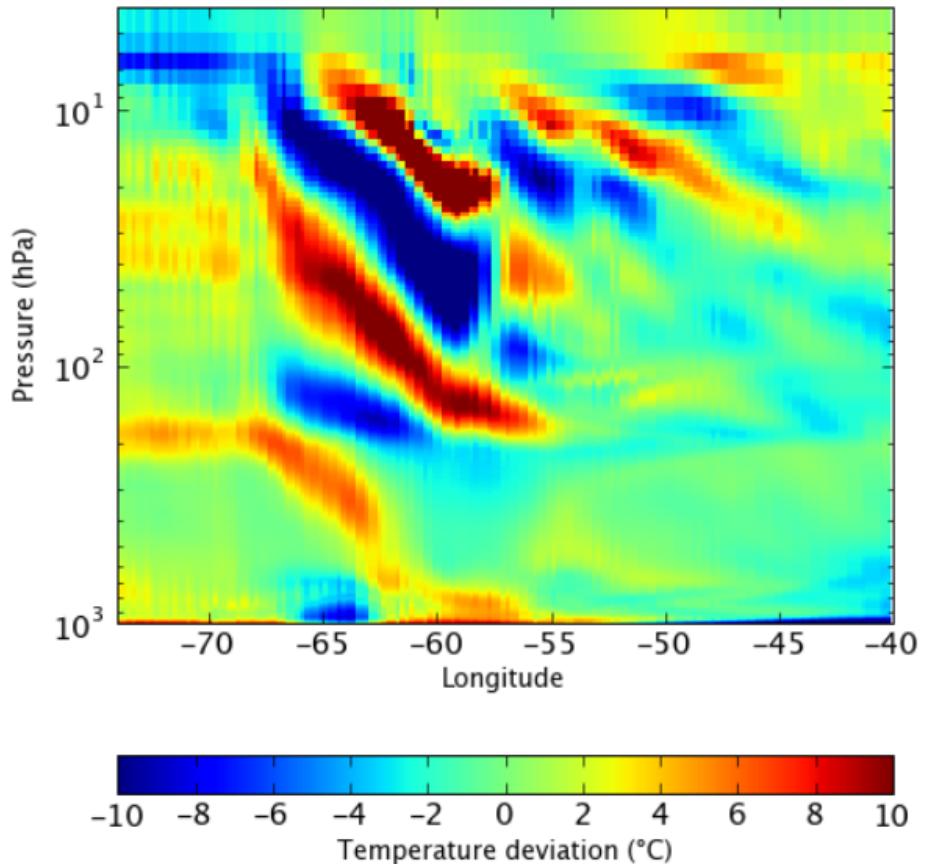


3.2 Matplotlib + Basemap : exemple (2)

WRF Temperature at 20 hPa, 0300 UTC



Temperature deviation from pressure-level zonal average at 72°S



3.3 scipy

- Fournit avec numpy des fonctions mathématiques utiles
- Les fonctions sont regroupées par thèmes dans des sous-modules à importer séparément
 - ndimage - multi-dimensional image processing
 - stats - statistiques
 - io - lecture/écriture (netCDF3, ASCII, Matlab)
 - signal - signal processing
 - interpolate - interpolations n-dimensions linéaires, cubiques, quadratiques, splines et autres
 - linsolve, odeint - linear equation / ordinary differential equations solvers
 - fftpack - transformees de Fourier
 - integrate...
- Impossible de tout couvrir, Google is your friend
- Cette après-midi, vous aurez l'occasion d'utiliser quelques fonctions de scipy



3.4 netcdf4-python

- Lecture/écriture de fichiers netCDF 3 ou 4
- D'autres modules existent (PyNIO, scipy.io, pycdf...), netcdf4-python est maintenu, fonctionnel, documenté
- Manipulation directe de variables netCDF comme des variables Python

```
import netCDF4
```

```
nc = netCDF4.Dataset('test.nc')
```

```
liste_var = nc.variables
```

```
for nomvar in liste_var:
```

```
    print 'Nom de variable : ', nomvar
```

```
    var = liste_var[nomvar] # liste_var est un dictionnaire
```

```
    print 'Unites : ', var.units # Acces aux attributs
```

```
    valeurs = var[:] # Acces aux valeurs
```



All together now

numpy + matplotlib + basemap + netCDF4

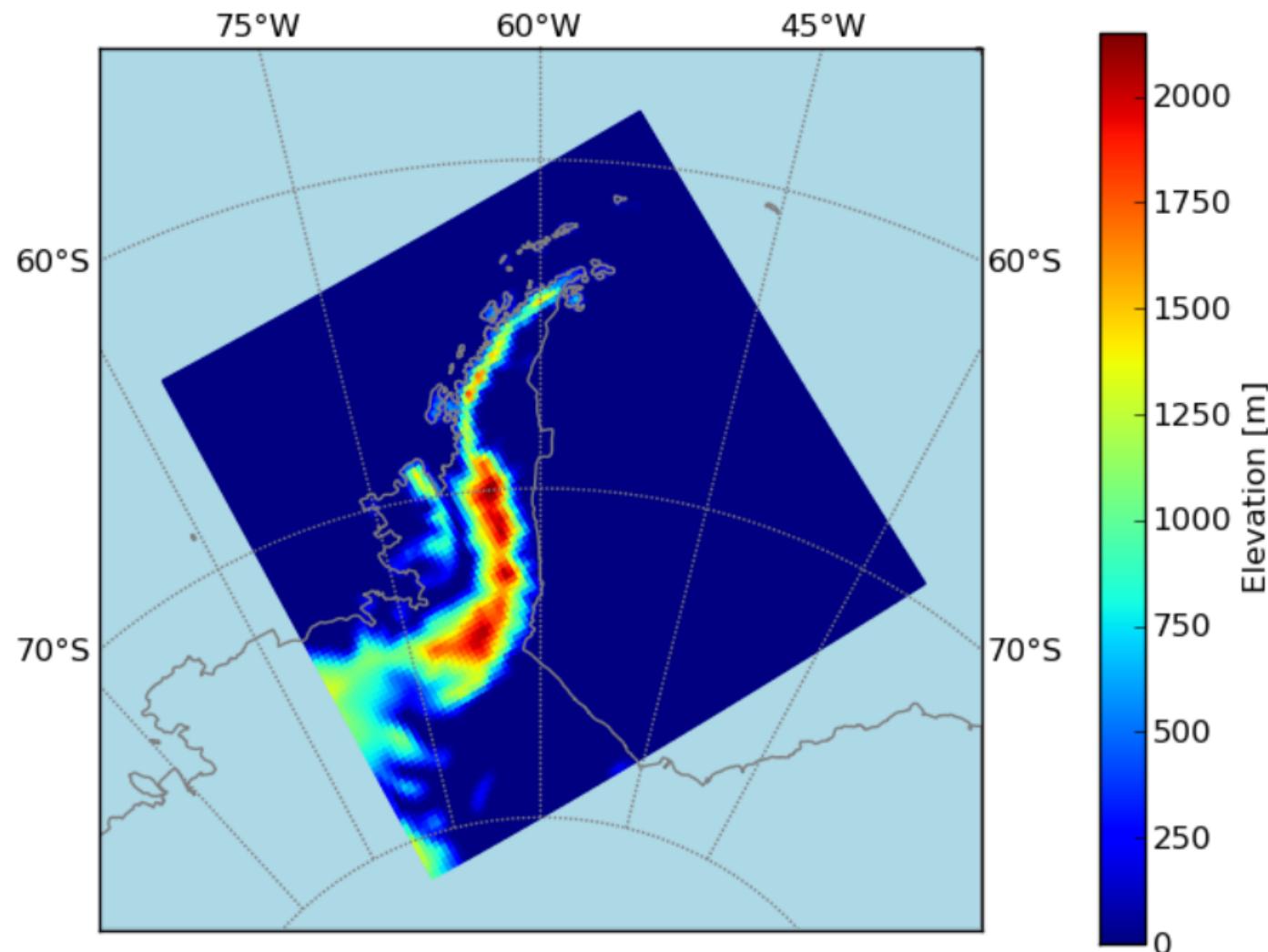
```
#!/usr/bin/env python
import numpy as np
import matplotlib.pyplot as plt
import netCDF4
from mpl_toolkits.basemap import Basemap
nc = netCDF4.Dataset('geo_em.d01.nc')
lat = nc.variables['XLAT_M'][0,:,:]
lon = nc.variables['XLONG_M'][0,:,:]
hgt = nc.variables['HGT_M'][0,:,:]
m = Basemap(projection='lcc',lat_0=-70,lon_0=-60,
            width=3000*1000,height=3000*1000, resolution='i')
x, y = m(lon, lat)

m.pcolor(x, y, hgt)
m.drawmapboundary(fill_color='lightblue')
m.drawmeridians(np.r_[-105:-15:15], labels=[0,0,1,0], color='grey')
m.drawparallels(np.r_[-80:90:10], labels=[1,1,0,0], color='grey')
m.drawcoastlines(color='grey')

plt.colorbar(pad=0.1).set_label('Elevation [m]')
plt.savefig('example_basemap.png')
```



18 lignes de Python



pour rire : essayer de faire la même chose en Fortran



3.5 bases de données (1)

Les SGBDR (Systèmes de Gestion de Bases de Données Relationnelles) sont des programmes informatiques capables de gérer efficacement un ensemble de données:

- complexe
- hétérogène

Les SGBDR permettent :

- d'éviter les doublons et les incohérences des données
- de manipuler efficacement un grand volume de données

3.5 Bases de données (2)

python propose la création et la manipulation des bases de données sous SGBDR Oracle, MySQL etc via les modules :

- **MySQLdb** pour MySQL
- **cx_oracle** pour Oracle

Ces modules permettent :

- la connexion à une SGBDR (MySQL par exemple)
- la création d'une nouvelle base de données
- la mise à jour des données dans une base existante
- extraction de données de la base

Toutes ces tâches peuvent être automatisées.

Autres modules

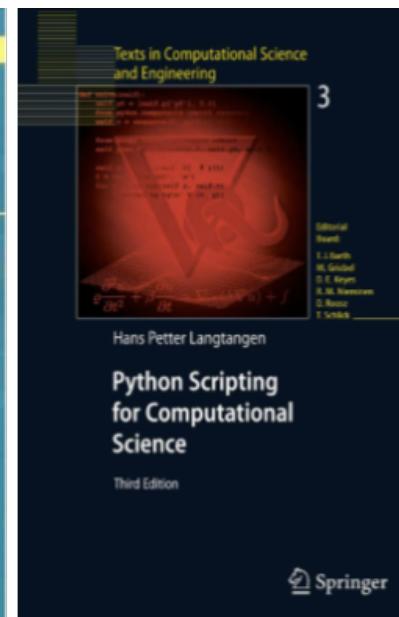
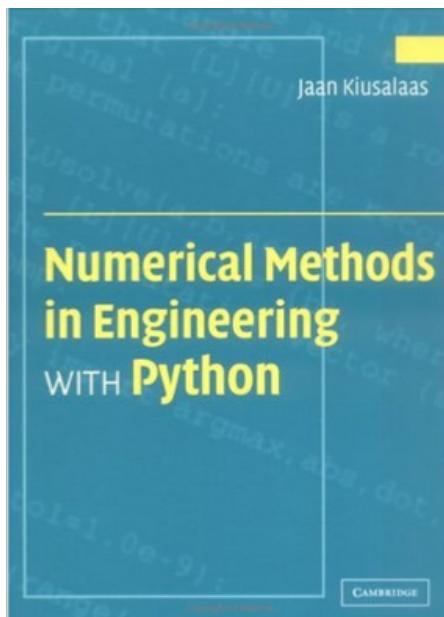
Il existe beaucoup d'autres modules pertinents dans un contexte de recherche scientifique, en voici quelques-uns que nous ne pourrons présenter :

- **pyhdf** - lecture/écriture HDF4
- **PyTables** - lecture/écriture HDF5
- **PyNIO** - lecture/écriture GRIB1 et 2, HDFEOS
- **PyWavelets** - transformées en ondelettes
- **shapely** - manipulation et analyse de géométrie planaire
- **GIS-Python** - geographic information systems
- **Sympy** - calcul symbolique à la Mathematica/Maple
- **PyClimate** - analyses de séries climatiques
- ...



Pour aller plus loin

- Restez pour l'après-midi
- Google : Python, science, numpy, matplotlib, tutorials...
- Bouquins (souvent en anglais, mais pas toujours)



Merci pour votre attention
Questions ?

TP

login machine
python1... python20

connection serveur de calcul
ssh -X tpclims01@loholt2.ipsl.polytechnique.fr
...
ssh -X tpclims20@loholt2.ipsl.polytechnique.fr