Introduction à Python scientifique.

Quelques repaires.

Denis Arrivault

Laboratoire d'Excellence Archimède
Aix Marseille Université

14 Décembre 2016 / LIFTech' III.

Introduction

Introduction

Généralités.

- Première version développée par Guido en 1989.
- Depuis 2001 (2.1) : développé et distribué par la Python Software Fundation. Open-source, licence libre.
- Semi-interprété (bytecode interprété) multi-plateformes.
- Orienté objet avec d'autres paradigmes possibles : impérative, fonctionnelle.
- Typage dynamique fort ($\neq$ Matlab).
- Rupture de compatibilité entre versions 2.x et 3.x.
- Dernière version : python 3.6 sortie prévue le 16 décembre 2016.

Version de python

Sauf précision, tous les exemples présentés ici sont écrits en python 3.

Introduction

Python pour le développement scientifique.

- Libre, gratuit, interactif, complet et rapide à prendre en main.
- Grosse communauté : association francophone python, python@services.cnrs.fr
- Extensions dédiées au développement scientifique : NumPy, SciPy et Matplotlib.
- Très bonnes performances dues, entre autres, à l'intégration des bibliothèques optimisées que sont blas, atlas blas, lapack, arpack, Intel MKL,...
- Support pour le multithreading, MPI, OpenCL et CUDA.
- Une multitude de bibliothèques dédiées à une discipline : astropy, biopython, sage, panda, scikit-learn...

Introduction

L'interpréteur python

```
$ python
>>> print('Hello World!')
Hello World!
```

```
# Fichier HelloWorld.py
print('Hello World!')
```

```
$ python HelloWorld.py
Hello World!
```

Les options

- Exécuter une instruction en ligne de commande : `-c`
- Passer en mode interactif après l'exécution d'un script : `-i`
- Lister les options de l'interpréteur : `-h`

Introduction

Installer Python

Linux

- Utiliser de préférence le gestionnaire de paquets pour installer python et ses modules.
- En cas d'utilisation de pip, il est vivement conseillé d'installer en local (`pip install -user monmodule`)
- Conseil : virtualenv et son extension virtualenvwrapper.

```
$ sudo apt-get install python3
$ sudo apt-get install ipython3 ipython3-notebook
$ sudo apt-get install python3-numpy python3-scipy
    python3-matplotlib
$ sudo apt-get install spyder3

$ sudo apt-get install python3-pip
$ pip3 install --user monmodule
```

MacOS

Mêmes conseils que pour linux en passant par `port`.

```
$ sudo port install python34
$ sudo port install py34-ipython, py34-notebook
$ sudo port install py34-scipy py34-matplotlib py34-numpy
$ sudo port install py34-spyder

$ sudo port install py34-pip
$ py34-pip install --user monmodule
```

Introduction

Installer Python

Windows

On peut trouver des installeurs sur
https://www.python.org/downloads/windows/

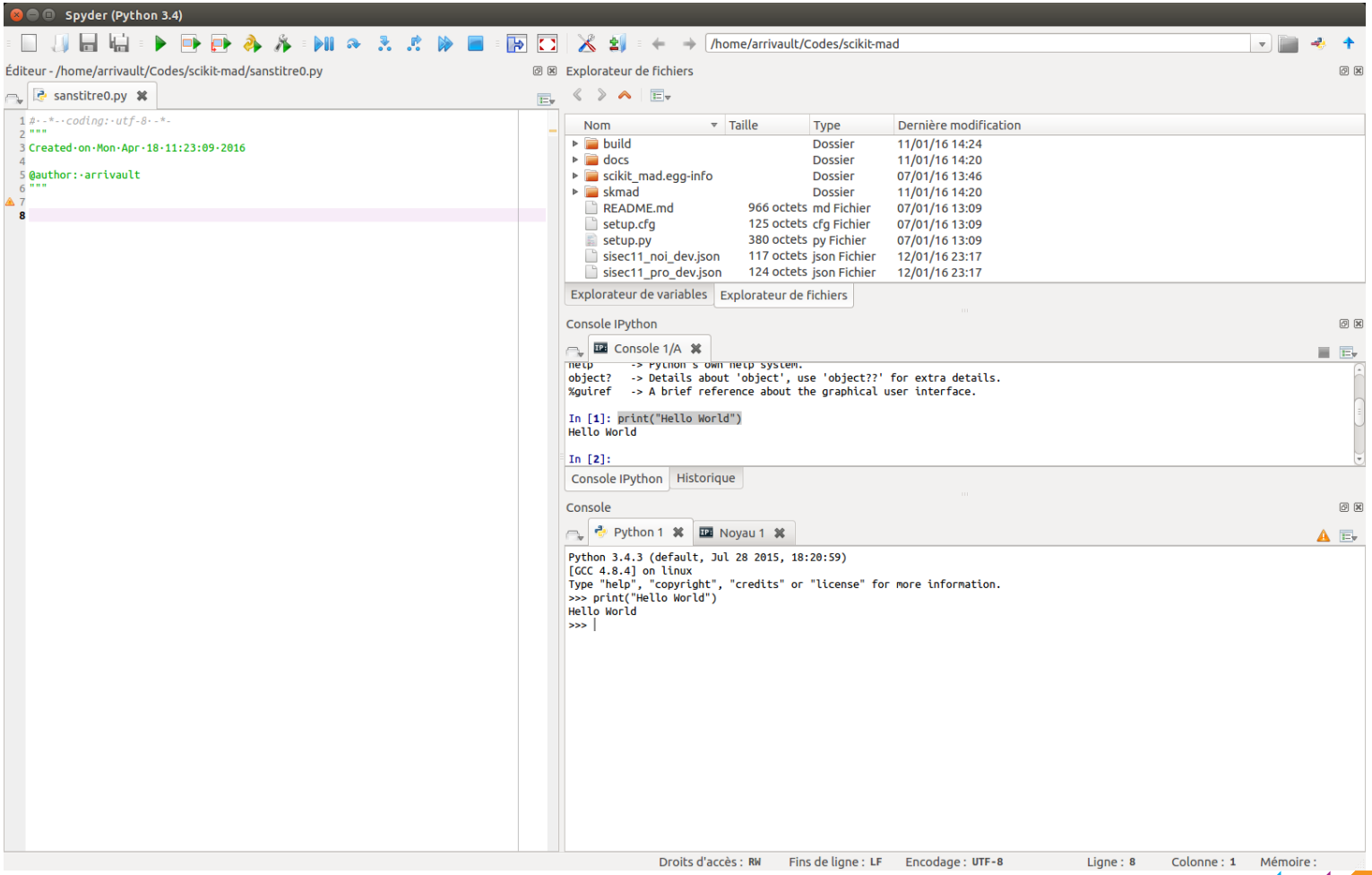
Une solution multi-plateformes : Anaconda

https://store.continuum.io/cshop/anaconda/

Introduction

IDE

Spyper fourni avec Anaconda.



Introduction

IDE

Mais aussi :

- Pycharm
- PyDev pour Eclipse.
- ...

Introduction

Quelques variables d'environnement

PYTHONSTARTUP : un fichier qui sera executé au lancement de l'interpréteur

PYTHONPATH : la liste de répertoires séparés par ' :' où se trouvent les modules qui seront chargés par le code.

PYTHONHOME : le répertoire contenant l'interpréteur python.

Le langage

Le langage

Les types et opérateurs

Propriétés

- Tout est objet.
- Le type d'un objet détermine les opérations que l'objet supporte et défini les valeurs possibles de cet objet.
- Le typage est dynamique.
- Notion de mutabilité : si une donnée peut changer de valeur en gardant le même `id` alors son type est mutable.

Exemple :

```
>>> a = 12
>>> id(a)
 10455392
>>> type(a)
 <class 'int'>
```

```
>>> a
 12
>>> a = 45
>>> id(a)
 10456448
```

Les nombres

Tous les nombres sont **non mutables**.

Type	Construction	Propriétés
Entier	`a = 10`	aucune limite (python 3)
Flottant	`a = 0.1`	64 bits, 17 digits, $2,2 \times 10^{-308} <= f <= 1,8 \times 10^{308}$
Complexe	`a = (4+5j) * 1J`	2 flottants

```
>>> a = 0.1 + 0.2
>>> a
 0.30000000000000004
>>> type(a)
 <class 'float'>
>>> 1.8e+308 + 1
 inf
```

Le langage

Les types et opérateurs

Les opérateurs

Affectation :

```
>>> a = 4
>>> a,b = 6,True
>>> a = b = 8
```

Logique :

```
>>> a = True
>>> not a
False
>>> b = False
>>> a and b
False
>>> a or b
True
```

Comparaison :

```
>>> a = 4
>>> b = 3
>>> a > b
True
>>> a <= b
False
>>> a = 5.0
>>> b = 5.0
>>> a == b
True
>>> a is b
False
```

- `is` teste les id.

Arithmétique :

```
>>> a=2
>>> a **= 3
>>> a
8
...
```

Le langage

Les chaînes de caractères

Le type `str` est **non mutable**.

Construction

```
>>> s = 'Hello World.'
>>> print(s)
Hello World.
>>> s = "Hello World."
>>> print(s)
Hello World.
>>> s = """Hello
...  World"""
>>> print(s)
Hello
World
>>> type(s)
<class 'str'>
```

Opérations

```
>>> s1 = 'Hello '
>>> s2 = 'World.'
>>> print(s1 + s2)
Hello World.
>>> s = 'Hello World.'
>>> print(s[0])
H
>>> print(s[0:4])
Hell
>>> 'Hello' in s
True
```

Pour plus d'informations : `help(str)`.

Le langage

Les Collections

Type	Construction	Propriétés
Liste	`L = [10, "aa"]`	Ordonnée, mutable
Tuple	`T = (10, "aa")`	Ordonnée, non mutable
Dictionnaire	`D = {"a" : 10, "b" : 2.0}`	Non ordonnée (indexée), mutable
Set	`S = {"a", 2.0}`	Non ordonnée, éléments uniques, mutable

```
>>> l = ['Paul', 'Paul', 'Eric', 'Solange', 'Eric', 'Eric', 'Paul
    ', 'Solange']
>>> s = set(l)
>>> s
{'Solange', 'Eric', 'Paul'}
```

Le langage

Les structures de contrôle

Définitions

- Les structures de contrôle se terminent par deux points " **:** "
- Les blocs d'instructions séquentielles qui suivent une structure de contrôle sont indentés de 4 espaces (tabulation déconseillée) vers la droite.
- C'est l'indentation et seulement l'indentation qui indique à l'interpréteur les instructions qui font partie d'un bloc.

```
structure 1:
    instruction 1.1
    instruction 1.2
    sous-structure 1.3:
        instruction 1.3.1
        instruction 1.3.2
    instruction 1.4
structure 2:
    instruction 2.1
    instruction 2.2
```

Le langage

Les structures de contrôle

Conditionnelles :

```
a = 10
if a > 0:
    print("Entier positif.")
elif a < 0:
    print("Entier negatif.")
else:
    print("Entier null.")
```

Boucles :

```
for i in range(0, 10):
    if i % 2:
        continue
    print(i, end='... ')
```

```
while True:
    print("Infinite loop... ")
```

Pour aller plus loin :

```
l = [x for x in range(10,15)]
for i,v in enumerate(l):
    print("{0:d} - {1:d}"
          .format(i,v))
```

```
l1 = [x for x in range(10,15)]
l2 = [x for x in range(85,90)]
for x,y in zip(l1,l2):
    print("{0:d} - {1:d}"
          .format(x,y))
```

Définitions

- Une fonction se définit à l'aide du mot clef `def`
- Elle est appelée par son nom avec les paramètres entre parenthèses. Le nom des paramètres doit être précisé si l'ordre de déclaration n'est pas respecté.
- Elle peut retourner une valeur ou plusieurs, mises alors dans un tuple.

Exemple

```
def divide(a,b):
    q = a // b
    r = a % b
    return(q,r)
```

```
result = divide(1548, 56)
#result est un tuple
#Autres possibilités:
quotien, reste = divide(1548, 56)
quotien, reste = divide(b=56,a=1548)
```

Les paramètres

Par défaut :

```
def divide(a=10,b=2):
    q = a // b
    r = a % b
    return(q,r)
```

```
quotien, reste = divide()
#prend les paramètres par défaut
quotien, reste = divide(20)
#on précise a mais b reste 2
quotien, reste = divide(b=5)
#on précise b avec son nom, a reste 10
```

Inconnus :

```
def anyargsfun(*args, **kwargs):
```

```
anyargsfun(45, "dr", [4, 8, 9], e=8, fgcolor="orange")
```

```
args est un tuple qui contient :
45 | dr | [4, 8, 9] |
kwargs est un dict qui contient :
fgcolor->orange | e->8 |
```

Le langage

Les modules

Un module est classiquement un fichier `.py`

Exemple d'un module exécutable

```
# -*- coding: utf-8 -*-
# file : gaussian.py

from math import exp, sqrt, pi


def gaussian(x, m=0, std=1):
    return exp(-(x - m) ** 2 / (2 * std ** 2))/(std * sqrt(2 *
        pi))


if __name__ == '__main__':
    print(gaussian(4))
    print(gaussian(4, 2, 2))
```

```
$ python gaussian.py
0.00013383022576488537
0.1209853622595716 8
```

Le langage

Les entrées/sorties

Entrée standard

```
>>> name = input("Entrez votre prénom : ")
Entrez votre prénom : Denis
>>> name
'Denis'
```

Sortie standard

```
>>> import sys
>>> print("Hello World!",file=sys.stdout) #flux par défaut
Hello World!
>>> print("Erreur!",file=sys.stderr)
Erreur!
```

Le langage

Les entrées/sorties

Ouverture d'un fichier

```
f = open("Couleurs.txt") # lecture
f = open("Couleurs.txt","r") # lecture
f = open("Couleurs.txt","w") # écriture
```

Lecture d'un fichier

```
for line in f:    #iteration sur toutes les lignes de f
    print(line)
```

Écriture dans un fichier

```
print('With tangerine trees and marmalade skies',file=f)
```

Fermeture d'un fichier

```
f.close()
```

Le langage

Les modules numériques et mathématiques standards

- Fonctions mathématiques à valeurs réelles : https://docs.python.org/3.6/library/math.html
- Fonctions mathématiques à valeurs complexes : https://docs.python.org/3.6/library/cmath.html
- Générateurs pseudo-aléatoires : https://docs.python.org/3.6/library/random.html
- Fonctions statistiques : https://docs.python.org/3.6/library/statistics.html

Python scientifique

Python scientifique

Historique

1995 Numeric (by Jim Huguni & Col.).

2001 Numarray (Perry Greenfield, Rick White and Todd Miller, Paul Barrett) : complémentaire à Numeric pour les gros tableaux.

2006 Numpy (Travis Oliphant) : Synthèse des deux premiers et intégration à Scipy.

Qu'est-ce que Numpy ?

- La bibliothèque fondamentale du calcul scientifique en Python.
- Elle implémente des objets tableaux multi-dimensionnels, des dérivés de ces objets ainsi que des routines permettant de réaliser des opérations rapides sur ces objets.
- Dernière version : 1.11 avec Python 3.5. La version 1.12 compatible Python 3.6 est en béta.

Propriétés fondamentales

- Taille des tableaux fixée à la création. Changer le nombre d'éléments d'un tableau = créer un nouveau tableau et supprimer l'ancien.
- Tous les éléments d'un tableau sont de même type. On peut cependant créer des tableaux de `object` et avoir ainsi des éléments de tailles différentes.
- La couche mémoire est compact et compatible C/Fortran.
- Les opérations ont des syntaxes simples, intuitives et invariables suivant les dimensions des tableaux.

Les 2 concepts fondamentaux de Numpy :

La vectorisation : Numpy est optimisé pour ne jamais utiliser de boucles.

La diffusion (broadcasting) : Numpy gère les différences de tailles de tableaux durant les opérations arithmétiques.

Python scientifique

Numpy

Quelques exemples

```
>>> from numpy import *
```

```
>>> a = array([[1,2,3],[4,5,6]])
>>> a
array([[1, 2, 3],
       [4, 5, 6]])
>>> a[0,0]   # indexation
1
>>> a.shape
(2, 3)
>>> a.dtype
dtype('int64')
```

```
>>> a = array([[1,2,3],[4,5,6]], dtype='f') # flottant
>>> a
array([[ 1.,  2.,  3.],
       [ 4.,  5.,  6.]], dtype=float32)
```

Python scientifique

Numpy

Quelques exemples un peu plus poussés...

```
>>> a=arange(10,step=2)
array([0, 2, 4, 6, 8])
```

```
>>> a=zeros((2,4),dtype='f')
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]], dtype=float32)
```

```
>>> a=fromfunction(lambda i,j:100+10*i+j,(5,3))
>>> a
array([[ 100.,  101.,  102.],
       [ 110.,  111.,  112.],
       [ 120.,  121.,  122.],
       [ 130.,  131.,  132.],
       [ 140.,  141.,  142.]])
```

```
>>> I=zeros((512,512), dtype=[('r','u1'),('g','u1'),('b','u1')])
>>> I['r']=256
>>> I['g']=255
>>> I[15,58]
(0, 255, 0)
```

Introduction

- Bibliothèque de visualisation graphique de données 2D.
- Créée en 2007 par John D. Hunter (décédé en 2012).
- Très proche de la syntaxe Matlab.
- Très bon tuto de Nicolas P. Rougier : http://www.labri.fr/perso/nrougier/teaching/matplotlib/

Python scientifique

Matplotlib

Introduction

- Bibliothèque de visualisation graphique de données 2D.
- Créée en 2007 par John D. Hunter (décédé en 2012).
- Très proche de la syntaxe Matlab.
- Très bon tuto de Nicolas P. Rougier : http://www.labri.fr/perso/nrougier/teaching/matplotlib/

Exemple simple

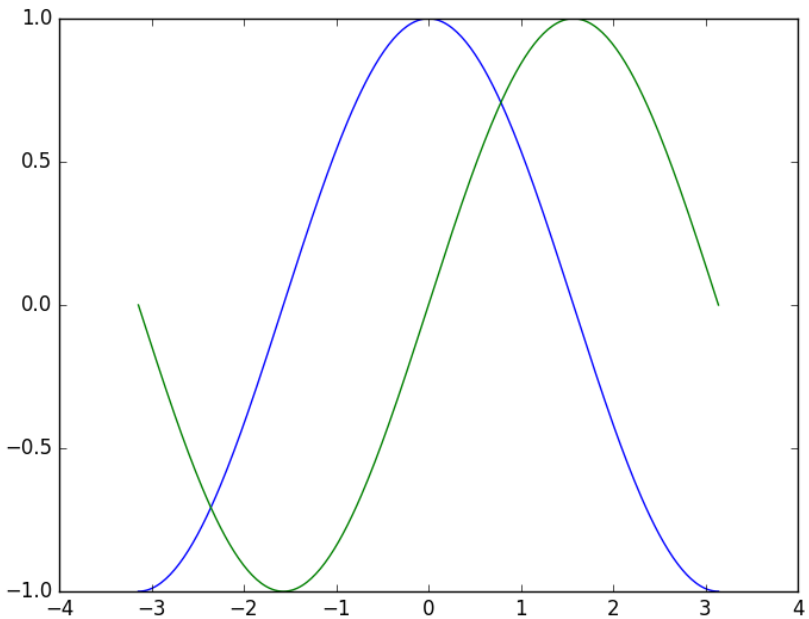
```
>>> import numpy as np
>>> X = np.linspace(-np.pi, np.pi, 256,endpoint=True)
>>> C, S = np.cos(X), np.sin(X)
>>> import matplotlib.pyplot as plt
>>> plt.plot(X,C)
 [<matplotlib.lines.Line2D object at 0x7f7020052b70>]
>>>  plt.plot(X,S)
 [<matplotlib.lines.Line2D object at 0x7f7020029438>]
```

Python scientifique

Matplotlib

Exemple simple

```
>>> import numpy as np
>>> X = np.linspace(-np.pi, np.pi, 256,endpoint=True)
>>> C, S = np.cos(X), np.sin(X)
>>> import matplotlib.pyplot as plt
>>> plt.plot(X,C)
 [<matplotlib.lines.Line2D object at 0x7f7020052b70>]
>>>  plt.plot(X,S)
 [<matplotlib.lines.Line2D object at 0x7f7020029438>]
```



Python scientifique

Matplotlib

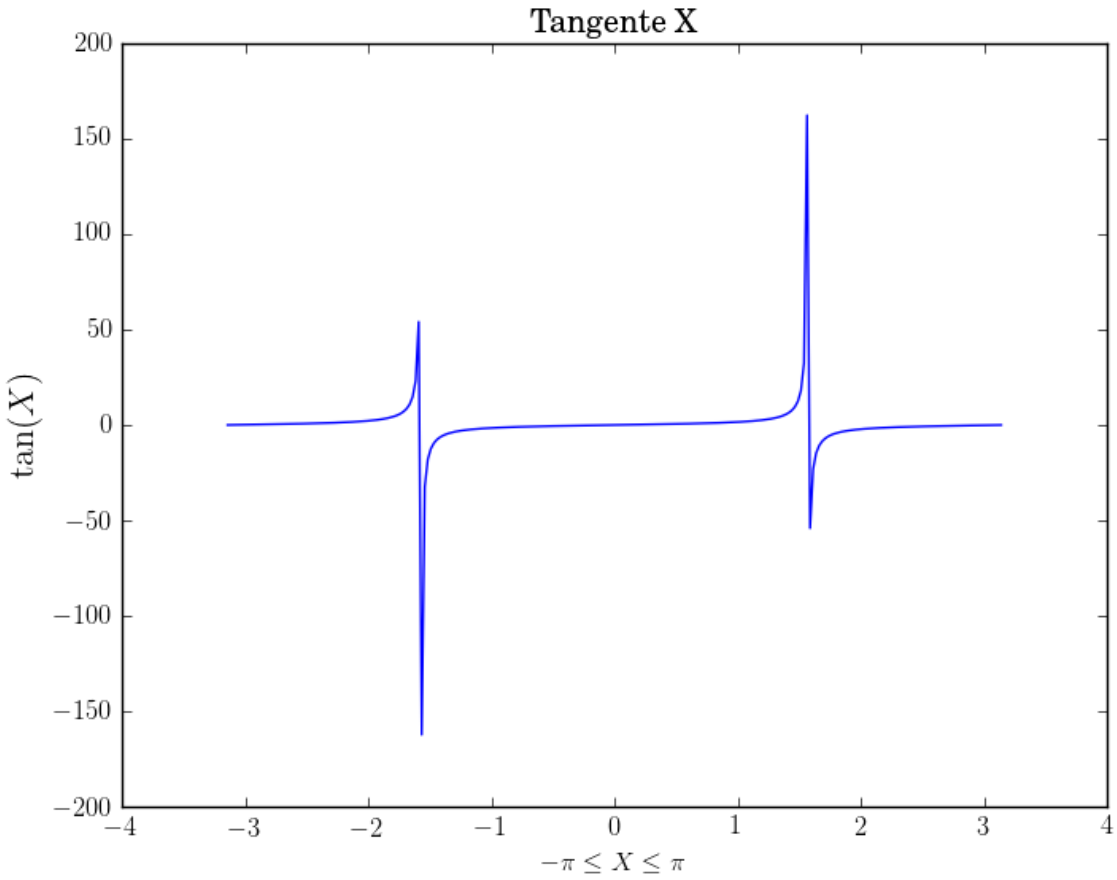
Axes et titre avec LaTeX

```
>>> import numpy as np
>>> X = np.linspace(-np.pi, np.pi, 256,endpoint=True)
>>> import matplotlib.pyplot as plt
>>> from matplotlib import rc
>>> plt.rc('text', usetex=True)
>>> plt.rc('font', family='serif')
>>> T = np.tan(X)
>>> plt.plot(X,T)
>>> plt.title(r'Tangente X')
>>> plt.xlabel(r'$-\pi \leq X \leq \pi$')
>>> plt.ylabel(r'$\tan(X)$', fontsize=16)
```

Python scientifique

Matplotlib

Axes et titre avec LaTeX



Python scientifique

Les bibliothèques de scipy

Scipy est le module scientifique de Python basé sur Numpy.

Qu'est-ce qu'on trouve dans scipy ?

Clustering package (`scipy.cluster`)
Constants (`scipy.constants`)
Discrete Fourier transforms (`scipy.fftpack`)
Integration and ODEs (`scipy.integrate`)
Interpolation (`scipy.interpolate`)
Input and output (`scipy.io`)
Linear algebra (`scipy.linalg`)
Miscellaneous routines (`scipy.misc`)
Multi-dimensional image processing (`scipy.ndimage`)
Orthogonal distance regression (`scipy.odr`)
Optimization and root finding (`scipy.optimize`)
Signal processing (`scipy.signal`)
Sparse matrices (`scipy.sparse`)
Sparse linear algebra (`scipy.sparse.linalg`)
Compressed Sparse Graph Routines (`scipy.sparse.csgraph`)
Spatial algorithms and data structures (`scipy.spatial`)
Special functions (`scipy.special`)
Statistical functions (`scipy.stats`)
Statistical functions for masked arrays (`scipy.stats.mstats`)
C/C++ integration (`scipy.weave`)

Conclusion

Conclusion

- Un langage libre, open-source et performant.
- Une communauté dynamique régie par des règles collaboratives (PEP).
- Utilisé pour tout : web (Django, Flask), HPC (LoOps), Data Mining (Orange)...
- Un dépôt tiers officiel facile d'utilisation : Pypi.

credits

credits

credits

- Supports des cours de la formation continue AMU "Python Scientifique" : D. Arrivault & F.X. Dupé.
- https://docs.python.org/3.6/
- "Python Essential Reference", David M. Beazley, 4th Edition, Addison-Wesley Professional, ISBN-13 : 978-0672329784
- "Python Pocket Reference", Mark Lutz, 5th Edition, O'Reilly, ISBN-13 :978-1449357016