
Informatique et Python

Version 2013-2014

Yannick Copin

12/01/14, 18:09

1	Introduction	1
1.1	Pourquoi Python ?	1
1.2	Organisation des TDs	1
2	Initiation à Python	3
2.1	Introduction	3
2.2	Types de base	5
2.3	Structures de programmation	6
2.4	Les chaînes de caractères	7
2.5	Objets itérables	8
2.6	Fonctions	10
2.7	Bibliothèques et scripts	11
2.8	Classes	12
2.9	Exceptions	14
2.10	Entrées-sorties	15
2.11	Éléments passés sous silence	16
3	Bibliothèque standard	17
3.1	Gestion des arguments/options de la ligne de commande	17
3.2	[c]Pickle : sérialisation des données	18
3.3	<i>Batteries included</i>	19
3.4	<i>Text/Graphical User Interfaces</i>	19
4	Bibliothèques numériques	21
4.1	Numpy	21
4.2	Scipy	28
4.3	Bibliothèques graphiques	29
4.4	Autres librairies scientifiques	30
5	Développer en python	33
5.1	Le zen du python	33
5.2	Outils de développement	35
6	Références supplémentaires	39
6.1	Documentation générale	39
6.2	Forums de discussion	39
6.3	Listes de liens	39
6.4	Livres libres	40
6.5	Cours en ligne	40
6.6	Sphinx & co.	41
7	Exemples	43
7.1	Cercle circonscrit	43

7.2	Filtres du 2nd ordre	46
8	Exercices	51
8.1	Manipulation de listes	51
8.2	Flocon de Koch	51
8.3	Jeu de la vie	52
8.4	Manipulation de tableaux (arrays)	52
8.5	Algorithmes numériques	54
8.6	Exercices supplémentaires	54
9	Micro-projets	57
9.1	Introduction	57
9.2	Formation de pistes de fourmis sur un pont à 2 branches	57
9.3	Formation d'agrégats	57
9.4	Modèle d'Ising	57
9.5	Méthode de Hückel	59
9.6	Évacuation d'une salle & déplacement d'une foule dans une rue	59
9.7	Densité d'états d'un nanotube	60
9.8	Solitons	60
9.9	Auto-organisation d'un banc de poissons	61
9.10	Diagramme de phase du potentiel de Lennard-Jones	63
9.11	Suivi de particule(s)	63
9.12	Autres possibilités	64
10	Projets 2013	67
11	Carré magique	69
12	Flocon de Koch	71
13	Jeu de la vie	73
14	<i>Median Absolute Deviation</i>	75
15	<i>Distribution du pull</i>	77
16	Quartet d'Anscombe	79
17	Suite logistique	81
18	Ensemble de Julia	83

Introduction

Version ENSL - L3 du 12/01/14, 18 :09

Auteur Yannick Copin <(at) ipnl.in2p3.fr>

1.1 Pourquoi Python ?

Les principales caractéristiques du langage `python` :

- Syntaxe simple et lisible : langage pédagogique et facile à apprendre et à utiliser ;
- Langage interprété : utilisation interactive ou script exécuté ligne à ligne, pas de processus de compilation ;
- Haut niveau : typage dynamique, gestion active de la mémoire, pour une plus grande facilité d'emploi ;
- Multi-paradigme : langage impératif et/ou orienté objet, selon les besoins et les capacités de chacun ;
- Logiciel libre et ouvert, largement répandu (multi-plateforme) et utilisé (forte communauté) ;
- Riche bibliothèque standard : *Batteries included* ;
- Riche bibliothèque externe : de nombreuses bibliothèques de qualité, dans divers domaines (y compris scientifiques), sont déjà disponibles.

L'objectif est bien d'apprendre *un seul* langage de haut niveau, permettant tout aussi bien des analyses rapides dans la vie de tous les jours – quelques lignes de code en interactif – que des programmes les plus complexes (projets de plus de 100000 lignes).

Liens :

- [About Python](#)
- [Python Advocacy](#)

1.2 Organisation des TDs

- *TD1* : Introduction, types de base, structures de programmation
- *TD2* : Chaînes de caractères, objets itérables, fonctions, bibliothèques
- *TD3* : Classes, exceptions, entrées-sorties
- *TD4* : Librairie standard, révisions
- *TD5* : Numpy
- *TD6* : Scipy
- *TD7* : Zen du python, révisions
- *Page projets* : Exemple de sujets de micro-projets


Initiation à Python

Table des matières

- Initiation à Python
 - Introduction
 - Installation
 - Notions d'Unix
 - L'invite de commande
 - Types de base
 - Structures de programmation
 - Les chaînes de caractères
 - Indexation
 - Méthodes
 - Formatage
 - Objets itérables
 - Fonctions
 - Bibliothèques et scripts
 - Bibliothèques externes
 - Bibliothèques personnelles et scripts
 - Classes
 - Exceptions
 - Entrées-sorties
 - Interactif
 - Fichiers
 - Éléments passés sous silence

2.1 Introduction

Liens :

- [Python Tutorial \(v2.7\)](#)
- [Tutoriel Python \(v2.4\)](#) 
- [Beginner's Guide](#)

2.1.1 Installation

Cette introduction repose sur les outils suivants :

- [Python](#) p.ex. 2.7¹ (inclus l'interpréteur de base et la bibliothèque standard)

1. Le développement actuel de Python se fait uniquement sur la branche 3.x, qui constitue une remise à plat *non-rétrocompatible* du langage. À moins d'être activement impliqué dans le développement de modules 3.x, il est encore conseillé – en 2011 pour un utilisateur scientifique lambda – de développer en python 2.x, car certaines bibliothèques utiles n'ont pas encore été portées ou pleinement testées en 3.x.

- Bibliothèques scientifiques : [Numpy](#) et [Scipy](#),
- Bibliothèque graphique : [matplotlib](#),
- Interpréteur évolué, p.ex. [ipython](#),
- Éditeur de texte évolué, p.ex. [emacs](#)

Ces logiciels peuvent être installés indépendamment, de préférence sous Linux (p.ex. [Ubuntu](#), [Fedora](#) ou [open-SUSE](#)), ou sous Windows ou MacOS. Il existe également des distributions « clés en main » :

- [Python\(x,y\)](#) (Windows)
- [Enthought Canopy](#) (Windows, MacOS, Linux, gratuite pour les étudiants du supérieur)

2.1.2 Notions d'Unix

Les concepts suivants sont supposés connus :

- ligne de commande : exécutables et options
- arborescence : chemin relatif ([./]...) et absolu (/...), navigation (**cd**)
- gestion des fichiers (**ls**, **rm**, **mv**) et répertoires (**mkdir**)
- gestion des exécutables : \$PATH, **chmod +x**
- gestion des processus : **&**, Control-c, Control-z + **bg**
- variables d'environnement : **export**, **.bashrc**

Liens :

- [Quelques notions et commandes d'UNIX](#) 
- [Introduction to Unix Study Guide](#)

2.1.3 L'invite de commande

Il existe principalement deux interpréteurs interactifs de commandes Python :

- **python** : l'interpréteur de base :

```
$ python
Python 2.7.1+ (r271:86832, Apr 11 2011, 18:05:24)
[GCC 4.5.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- Control-d pour sortir
- `help(commande)` pour obtenir l'aide d'une commande
- *A priori*, pas d'historique des commandes ni de complétion automatique

L'interpréteur de base permet également d'interpréter un « script », c'est-à-dire un ensemble de commandes regroupées dans un fichier texte (généralement avec une extension `.py`) : `python mon_script.py`

- **ipython** : interpréteur évolué (avec historique et complétion automatique des commandes) :

```
$ ipython
Python 2.7.1+ (r271:86832, Apr 11 2011, 18:05:24)
Type "copyright", "credits" or "license" for more information.

IPython 0.10.1 -- 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.
```

In [1]:

- Control-d pour sortir
- Tab pour la complétion automatique
- Haut et Bas pour le rappel des commandes
- Aide ipython : `object?` pour une aide sur un objet, `object??` pour une aide plus complète (au niveau source)
- Commandes *magic* (voir `%magic`) :

- `%run mon_script.py` pour exécuter un script *dans* l'interpréteur
- `%debug` pour lancer le mode débogage interactif *post-mortem*
- `%cpaste` pour coller et exécuter un code pré-formaté

Liens :

- [Tutorial](#)
- [IPython Tip Sheet](#)

2.2 Types de base

- None (rien)
- Chaînes de caractères : `str`
 - Entre (simples ou triples) apostrophes `'` ou guillemets `"` : `'Calvin'`, `"Calvin'n'Hobbes"`, `'''Two\nlines'''`, `"""'Why?' he asked."""`
 - Conversion : `str(3.2)`
- Objets numériques :
 - Booléens `bool` (vrai/faux) : `True`, `False`, `bool(3)`
 - Entiers `int` (pas de valeur limite explicite, correspond *au moins* au long du C) : `-2`, `int(2.1)`, `int("4")`
 - Réels `float` (entre $\pm 1.7e\pm 308$, correspond au double du C) : `2.`, `3.5e-6`, `float(3)`
 - Complexes `complex` : `1+2j`, `5.1j`, `complex(-3.14)`, `complex('j')`

```
>>> 3/2          # Attention! Division euclidienne par défaut
1
>>> float(3)/2   # ou 'from __future__ import division'
1.5
>>> (1+2j)**-0.5  # puissance entière, réelle ou complexe
(0.5688644810057831-0.3515775842541429j)
```

- Objets itérables :
 - Listes `list` : `['a', 3, [1,2], 'a']`
 - Listes immuables `tuple` : `(2, 3.1, 'a', [])` (selon les conditions d'utilisation, les parenthèses ne sont pas toujours nécessaires)
 - Listes à clés `dict` : `{ 'a':1, 'b':[1,2], 3:'c' }`
 - Ensembles non ordonnés `set` : `set([1, 2, 3, 2])`

```
>>> x,y = 1,2      # Affectations multiples via tuples (ici les parenthèses ne sont pas nécessaires)
>>> l1 = ['a','b']; l2=['c']; l1+l2 # Concaténation de listes
['a', 'b', 'c']
>>> range(5)       # liste de 5 entiers commençant par 0
[0, 1, 2, 3, 4]
>>> set([2,1,3]) | set([1,2,'a']) # Union d'ensembles
set(['a', 1, 2, 3])
```

- `type(obj)` retourne le type de l'objet, `isinstance(obj, type)` teste le type de l'objet.

```
>>> l = [1,2,3]
>>> type(l)
<type 'list'>
>>> isinstance(l,tuple)
False
```

2.3 Structures de programmation

- Une instruction par ligne en général (ou instructions séparées par `;` en interactif).
- Les blocs sont définis par l'**indentation** (en général par pas de 4 espaces).
- Les commentaires commencent par `#`, et vont jusqu'à la fin de la ligne.

- Expression booléenne : une condition est une expression s'évaluant à True ou False :
 - False : test logique faux (p.ex. 3==4), 0, '' (chaîne vide), [] (liste vide), etc.
 - True : test logique vrai (p.ex. 3==3), toute valeur/objet non-nul (et donc s'évaluant par défaut à True)
 - Tests logiques : ==, >, >=, etc.

Attention : Ne pas confondre = (affectation d'une variable) et == (test logique d'égalité).

- Opérateurs logiques : and, or, not
- Opérateur ternaire : *value if condition else altvalue*
- Expression conditionnelle if/elif/else :

```
if i>0:      # Condition principale
    print "positif"
elif i<0:   # Condition secondaire (si nécessaire)
    print "négatif"
else:      # Cas final (si nécessaire)
    print "nul"
```

- Boucle for : *for element in iterable:*, s'exécute sur chacun des éléments d'un objet itérable :

```
>>> for i in range(1,10,2): # entiers de 1 (inclus) à 10 (exclu) par pas de 2
...     print i**2          # Affichage de i**2
1
9
25
49
81
```

- continue : interrompt l'itération courante, et reprend la boucle à l'itération suivante
- break : interrompt complètement la boucle
- Boucle while : *while condition:* se répète tant que la *condition* est vraie, ou après une sortie explicite avec break

Attention : aux boucles infinies, dont la condition d'exécution reste invariablement vraie (typiquement un critère de convergence qui n'est jamais atteint). On peut toujours s'en protéger en testant *en plus* sur un nombre maximal (raisonnable) d'itérations :

```
>>> niter = 0
>>> while error>1e-6 and niter<100:
...     error = ... # A priori, error va décroître, et la boucle s'interrompt
...     niter += 1 # Mais on n'est jamais assez sûr!
```

2.4 Les chaînes de caractères

2.4.1 Indexation

Les chaînes sont des objets *itérables* – i.e. constitués d'éléments (ici les caractères) sur lesquels il est possible de « boucler » (p.ex. avec *for*) – et *immuables* – i.e. dont les éléments individuels ne peuvent pas être modifiés.

```
>>> alpha = 'abcdefghijklmnopqrstuvwxy'
>>> len(alpha)
26
>>> alpha[0]      # 1er élément (l'indexation commence à 0)
'a'
>>> alpha[-1]     # Dernier élément (-2: avant dernier, etc.)
'z'
>>> alpha[5:10]   # *Slice*, du 5ème élément (inclu) au 10ème élément (exclu)
'fghij'
>>> alpha[:5]     # Slice, du 1er élément (par défaut) au 5ème élément (exclu)
```

```
'abcde'
>>> alpha[20:] # Slice, du 20ème élément (inclu) au dernier inclu (défaut)
'uvwxyz'
>>> alpha[:5] # Slice, du 1er au dernier élément (défauts), tous les 5 éléments
'afkpuz'
```

2.4.2 Méthodes

Les chaînes de caractères disposent de nombreuses propriétés (des « méthodes ») facilitant leur manipulation :

```
>>> n,t = "Calvin","Hobbes"
>>> titre = n + ' et ' + t; titre # += Concaténation de chaînes
'Calvin et Hobbes'
>>> titre.replace('et','&') # Remplacement de sous-chaînes
'Calvin & Hobbes'
>>> ' & '.join(titre.split(' et ')) # Découpage et jonction
'Calvin & Hobbes'
>>> 'Hobbes' in titre # in: Test d'inclusion
True
>>> titre.find("Hobbes") # str.find: Recherche de sous-chaîne
10
>>> titre.center(30, '-')
'-----Calvin et Hobbes-----'
>>> dir(str) # Liste toutes les méthodes des chaînes
```

2.4.3 Formatage

Le système de **formatage** des chaînes de caractères s'appuie sur l'opérateur %, et est très semblable au `printf` du C/C++ :

```
>>> "%s a %d ans" % ('Calvin',6) # 'format' % (tuple)
'Calvin a 6 ans'
>>> pi = 3.1415926535897931
>>> "%(x)f %(x).2f %(y)f %(y)g" % dict(x=pi, y=pi*1e9) # 'format' % {dict}
'3.141593 3.14 3141592653.589793 3.14159e+09'
```

`print` affiche à l'écran (plus spécifiquement la sortie standard) la conversion d'une variable en chaîne de caractères :

```
>>> print "Calvin and Hobbes\nScientific progress goes 'boink'"
Calvin and Hobbes
Scientific progress goes 'boink'
>>> print "%d fois %d font %d" % (3, 4, 3*4) # Formatage et affichage
3 fois 4 font 12
```

Exercice : tables de multiplications.

2.5 Objets itérables

Les chaînes de caractères, listes, tuples et dictionnaires sont les objets itérables de base en Python. Les listes et dictionnaires sont *mutables* – leurs éléments constitutifs peuvent être changés à la volée – tandis que chaînes de caractères et les tuples sont *immuables*.

– Accès indexé : conforme à celui des chaînes de caractères

```

>>> l = range(1,6) # De 1 (inclus) à 6 (exclu) = [1,2,3,4,5]
>>> len(l)         # Nb d'éléments dans la liste (i varie de 0 à 4)
5
>>> l[0],l[-2]     # 1er et avant-dernier élément (l'indexation commence à 0)
(1, 4)
>>> l[5]           # Erreur: indice hors-bornes
IndexError: list index out of range
>>> d = dict(a=1, b=2) # Création du dictionnaire {'a':1, 'b':2}
>>> d['a']          # Accès à une entrée via sa clé
1
>>> d['c']          # Erreur: clé inexistante!
KeyError: 'c'
>>> d['c'] = 3; d    # Ajout d'une clé et sa valeur
{'a': 1, 'c': 3, 'b': 2} # Noter qu'un dictionnaire N'est PAS ordonné!

```

– Sous-listes (*slices*) :

```

>>> l[1:-1]        # Du 2ème ('1') *inclus* au dernier ('-1') *exclu*
[2, 3, 4]
>>> l[1:-1:2]      # Idem, tous les 2 éléments
[2, 4]
>>> l[:2]          # Tous les 2 éléments (avec /start/ et /stop/ par défaut)
[1, 3, 5]

```

– Modification d'éléments d'une liste (chaînes et tuples sont **immuables**) :

```

>>> l[0]='a'; l    # Remplacement du 1er élément
['a', 2, 3, 4, 5]
>>> l[1:2]='x','y'; l # Remplacement d'éléments par /slices/
['a', 'x', 3, 'y', 5]
>>> l + [1,2]; l    # Concaténation (l reste inchangé)
['a', 'x', 3, 'y', 1, 2]
['a', 'x', 3, 'y']
>>> l += [1,2]; l    # Concaténation (modification de l)
['a', 'x', 3, 'y', 5, 1, 2]
>>> del l[-3:]; l    # Efface les 3 derniers éléments de la liste
['a', 'x', 3, 'y']

```

Attention : à la modification des objets mutables :

```

>>> l = range(3)    # l pointe vers la liste [0,1,2]
>>> m = l; m        # m est un *alias* de la liste l
[0, 1, 2]
>>> id(l); id(m); m is l
171573452          # id(/obj/) retourne le n° d'identification en mémoire
171573452          # m et l ont le même id:
True               # ils correspondent donc bien au même objet en mémoire
>>> l[0] = 'a'; m    # puisque l a été modifiée, il en est de même de m
['a', 1, 2]
>>> m = l[:];        # copie des *éléments* de l dans une nouvelle liste m (clonage)
>>> id(l); id(m); m is l
171573452          # m a un id différent de l: il s'agit de 2 objets distincts
171161228          # (contenant éventuellement la même chose!)
False
>>> del l[-1]; m     # les éléments de m n'ont pas été modifiés
['a', 1, 2]

```

– Liste en compréhension :

```

>>> [ i**2 for i in range(5) ] # Carré de tous les éléments de [0,...,4]
[0, 1, 4, 9, 16]
>>> [ 2*i for i in range(10) if i%3!=0 ] # Compréhension conditionnelle ('%' = reste euclidien)
[2, 4, 8, 10, 14, 16]
>>> [ 10*i+j for i in range(3) for j in range(4) ] # Double compréhension

```

```
[0, 1, 2, 3, 10, 11, 12, 13, 20, 21, 22, 23]
>>> [ [ 10*i+j for i in range(3) ] for j in range(4) ] # Compréhensions imbriquées
[[0, 10, 20], [1, 11, 21], [2, 12, 22], [3, 13, 23]]
```

– Utilitaires sur les itérables :

```
>>> l1 = ['Calvin', 'Wallace', 'Boule']
>>> for i in range(len(l1)):      # Boucle sur les indices de l1
...     print i, l1[i]           # Accès explicite, pas pythonique :-()
0 Calvin
1 Wallace
2 Boule
>>> for i, name in enumerate(l1): # Boucle sur indice, valeur de l1
...     print i, name           # Pythonique! :-)
0 Calvin
1 Wallace
2 Boule
>>> l2 = ['Hobbes', 'Gromit', 'Bill']
>>> for n1, n2 in zip(l1, l2):    # Boucle simultanée sur 2 listes (ou +)
...     print n1, 'et', n2
Calvin et Hobbes
Wallace et Gromit
Boule et Bill
>>> sorted(zip(l1, l2)) # Tri, ici sur le 1er élément de chaque tuple de la liste
[('Boule', 'Bill'), ('Calvin', 'Hobbes'), ('Wallace', 'Gromit')]
```

Exercices : *Crible d'Ératosthène, Carré magique*

2.6 Fonctions

Une fonction est un regroupement d'instructions impératives – assignations, branchement, boucles – s'appliquant sur des arguments d'entrée. C'est le concept central de la programmation *impérative*.

def permet de définir une fonction : `def fonction(arg1, arg2, ..., option1=valeur1, option2=valeur2, ...)`. Les « args » sont des arguments nécessaires (i.e. obligatoires), tandis que les « kwargs » – arguments de type `option=valeur` – sont optionnels, puisqu'ils possèdent une valeur par défaut. Si la fonction doit retourner une valeur, celle-ci est spécifiée par `return`.

Exemples :

```
1 def temp_f2c(tf):
2     """Conversion d'une température Fahrenheit 'tf' en température Celsius.
3
4     Exemple:
5     >>> temp_f2c(104)
6     40.0
7     """
8
9     tc = (tf - 32.)/1.8      # Conversion Fahrenheit → Celsius
10
11    return tc

1 def mean(alist, power=1):
2     """Retourne la racine 'power' de la moyenne des éléments de
3     'alist' à la puissance 'power':
4
5     .. math:: \mu = (\frac{1}{N} \sum_{i=0}^{N-1} x_i^p)^{1/p}
6
7     'power=1' correspond à la moyenne arithmétique, 'power=2' au
8     *Root Mean Squared*, etc.
9
```

```
10  Exemples:
11  >>> mean([1,2,3])
12  2.0
13  >>> mean([1,2,3], power=2)
14  2.160246899469287
15  """
16
17  s = 0.                                # Initialisation de la variable *s* comme *float*
18  for val in alist:                    # Boucle sur les éléments de *alist*
19      s += val**power                  # *s* est augmenté de *val* puissance *power*
20  mean = (s / len(alist))**(1./power) # *mean* = (somme valeurs / nb valeurs)**(1/power)
21
22  return mean
```

Note : Python est un langage à typage *dynamique*, p.ex., le type des arguments d'une fonction n'est pas fixé *a priori*. Dans l'exemple précédent, `alist` peut être une `list`, un `tuple` ou tout autre itérable contenant des éléments pour lesquels les opérations effectuées – somme, exponentiation, division par un entier – ont été définies (p.ex. des entiers, des complexes, des matrices, etc.).

2.7 Bibliothèques et scripts

2.7.1 Bibliothèques externes

Une bibliothèque (ou module) est un code fournissant des fonctionnalités supplémentaires – p.ex. des fonctions prédéfinies – à Python. Ainsi, le module `math` définit les fonctions et constantes mathématiques usuelles (`sqrt`, `pi`, etc.)

Une bibliothèque « s'importe » avec la commande `import module`. Les fonctionnalités supplémentaires sont alors accessibles dans l'espace de noms `module` via `module.fonction` :

```
>>> sqrt(2)                                # sqrt n'est pas une fonction standard de python
NameError: name 'sqrt' is not defined
>>> import math                            # Importe tout le module 'math'
>>> dir(math)                             # Liste les fonctionnalités de 'math'
['__doc__', '__name__', '__package__', 'acos', 'acosh', 'asin',
'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos',
'cosh', 'degrees', 'e', 'exp', 'fabs', 'factorial', 'floor', 'fmod',
'frexp', 'fsum', 'hypot', 'isinf', 'isnan', 'ldexp', 'log', 'log10',
'loglp', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt',
'tan', 'tanh', 'trunc']
>>> math.sqrt(math.pi)                    # Les fonctionnalités sont disponibles sous 'math'
1.7724538509055159
>>> import math as M                      # Importe 'math' dans l'espace 'M'
>>> M.sqrt(M.pi)
1.7724538509055159
>>> from math import sqrt, pi             # Importe uniquement 'sqrt' et 'pi' dans l'espace courant
>>> sqrt(pi)
1.7724538509055159
```

Attention : Il est possible d'importer toutes les fonctionnalités d'une bibliothèque dans l'espace de noms courant :

```
>>> from math import *
>>> sqrt(pi)
1.7724538509055159
```

Cette pratique est cependant fortement *déconseillée* ! (possibilité de conflit dans les espaces de noms)

Nous verrons par la suite quelques exemples de modules de la *Bibliothèque standard*, ainsi que des *Bibliothèques numériques* orientées analyse numérique.

Exercice : *Flocon de Koch* (difficile)

2.7.2 Bibliothèques personnelles et scripts

Vous pouvez définir vos propres bibliothèques en regroupant les fonctionnalités au sein d'un même fichier *monfichier.py*. Si ce fichier est importé (p.ex. `import monfichier`), il agira comme une bibliothèque ; si ce fichier est exécuté (p.ex. `python ./monfichier.py`), il agira comme un *script*.

Attention : Toutes les instructions d'un module qui ne sont pas encapsulées dans le `__main__` (voir plus bas) sont interprétées et exécutées lors de l'import du module. Elles doivent donc en général se limiter à la définition de variables, de fonctions et de classes.

Un code Python peut donc être :

- un module – s'il n'inclut que des définitions mais pas d'instruction exécutable en dehors d'un éventuel `main`,
- ou un exécutable – s'il inclut un `main` ou des instructions exécutables,
- ou les deux à la fois.

Exemple : *circonscriit.py*

Ce code peut être importé comme une bibliothèque (p.ex. `from circonscriit import Point`) dans un autre code python, ou bien être exécuté depuis la ligne de commande (p.ex. `./circonscriit.py`), auquel cas la partie `__main__` sera exécutée.

- `#!` (**Sharp-bang**) : la première ligne d'un script définit l'interpréteur à utiliser :

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

- Un fichier incluant des caractères non-ASCII (p.ex. caractères accentués, ou symboles UTF tel que \pm) doit définir le système d'encodage, généralement `utf-8` :

```
# -*- coding: utf-8 -*-
```

Notez que les noms de variables, fonctions, etc. doivent être purement ASCII (a-zA-Z0-9_). De manière générale, favorisez la langue anglaise (variables, commentaires, affichages).

- `if __name__ == '__main__':` permet de séparer le `__main__` (i.e. le corps du programme, à exécuter lors d'une utilisation en script) des définitions de fonctions et classes, permettant une utilisation en module.

2.8 Classes

Un objet est une entité de programmation, disposant de ses propres états et fonctionnalités. C'est le concept central de la *programmation orientée objet*.

Au concept d'objet sont liées les notions de :

- **Classe** : il s'agit d'un modèle d'objet, dans lequel sont définis ses propriétés usuelles. P.ex. la classe `Point` peut représenter un point dans le plan, caractérisé par ses coordonnées `x`, `y`, et disposant de fonctionnalités propres (p.ex. `distance`).
- **Instantiation** : c'est le fait générer un objet concret (une instance) à partir d'un modèle (une classe). P.ex. `M = Point(2, 3)` crée le point *M* à partir de la classe `Point` et des coordonnées (2,3).
- **Attributs** : variables internes décrivant l'état de l'objet. P.ex. `M.x` donne la coordonnée en *x* du point *M*.
- **Méthodes** : fonctions internes, s'appliquant en premier lieu sur l'objet lui-même (`self`), décrivant le comportement de l'objet. P.ex. `M.distance(N)` retourne la distance entre les points *M* et *N*.

Attention : Toutes les méthodes d'une classe doivent au moins prendre `self` – représentant l'objet lui-même – comme premier argument.

- **Surcharge d'opérateurs** : cela permet de redéfinir les opérateurs et fonctions usuels (`+`, `abs()`, `str()`, etc.), pour simplifier l'écriture d'opérations sur les objets. Ainsi, on peut redéfinir l'opérateur `+` dans la classe `Vector` pour que `v1 + v2` ait un sens.

Liste des méthodes standard et des surcharges d'opérateur

- **Héritage de classe** : il s'agit de définir une classe à partir d'une (ou plusieurs) classe(s) parente(s). La nouvelle classe *hérite* des attributs et méthodes de sa (ses) parente(s), que l'on peut alors modifier ou compléter. P.ex. la classe `Vector` hérite de la classe `Point` (elle partage la notion de coordonnées), et lui ajoute des méthodes propres à la notion de vecteur (p.ex. l'addition vectorielle).

Attention : Toutes les classes doivent au moins hériter de la classe principale `object`.

Exemple de définition

```
1 class Point(object): # *object* est la classe dont dérivent toutes les autres
2     """Classe définissant un 'Point' du plan, caractérisé par ses
3     coordonnées 'x', 'y'.
4     """
5
6     def __init__(self, x, y):
7         """Méthode d'instanciation à partir de deux coordonnées."""
8
9         try:                                     # Convertit les coords en 'float'
10             self.x = float(x)
11             self.y = float(y)
12         except ValueError:
13             raise ValueError("Incompatible input coordinates")
14
15     def __str__(self):
16         """Surcharge de l'opérateur 'str': l'affichage *informel* de
17         l'objet dans l'interpréteur, p.ex. 'print self' sera résolu
18         comme 'self.__str__()'
19
20         Retourne une chaîne de caractères.
21         """
22
23         return "Point (x=%f, y=%f)" % (self.x, self.y)
24
25     def isOrigin(self):
26         """Méthode booléenne, vrai si le point est à l'origine."""
27
28         return ((self.x == 0) and (self.y == 0))
29
30     def distance(self, other):
31         """Méthode de calcul de la distance du point ('self') à un
32         autre ('other')."""
33
34         return hypot(self.x - other.x, self.y - other.y) # sqrt(dx² + dy²)
```

Exemple d'héritage

```
1 class Vector(Point):
2     """Un 'Vector' hérite de 'Point' avec des méthodes additionnelles
3     (p.ex. l'addition)."""
4
5     def __init__(self, A, B=0):
6         """Définit le vecteur :math:\vec{AB} si B!=0, ou
7         :math:\vec{OA} sinon."""
8
9         # Initialisation de la classe parente
10        if B.isOrigin():                # B = 0
11            Point.__init__(self, A.x, A.y)
12        else:                            # B != 0
13            Point.__init__(self, B.x-A.x, B.y-A.y)
```



```

14
15     # Attribut propre à la classe dérivée
16     self.norm2 = self.x**2 + self.y**2 # Norme du vecteur au carré
17
18     def __str__(self):
19         """Surcharge de la fonction 'str()': ainsi, 'print self' sera
20         résolu comme 'Vector.__str__(self)' (et non pas comme
21         'Point.__str__(self)')
22         """
23
24         return "Vector (x=%f, y=%f)" % (self.x, self.y)
25
26     def __add__(self, other):
27         """Surcharge de l'opérateur '+': l'instruction 'self + other'
28         sera résolue comme 'self.__add__(other)'.

```

Études de cas

- turtle.Vec2D
- fractions.Fraction

Exercice : *Jeu de la vie*

2.9 Exceptions

Lorsqu'il rencontre une erreur dans l'exécution d'une instruction, l'interpréteur génère (raise) une erreur (Exception), de *nature différente* selon la nature de l'erreur : `KeyError`, `ValueError`, `AttributeError`, `NameError`, `TypeError`, `IOError`, `NotImplementedError`, etc. La levée d'une erreur n'est cependant pas nécessairement fatale, puisque Python dispose d'un mécanisme de *gestion des erreurs*.

Il est d'usage en Python d'utiliser la philosophie EAFP (Easier to Ask for Forgiveness than Permission)² : plutôt que de tester explicitement toutes les conditions de validité d'une instruction, on « tente sa chance » d'abord, quitte à gérer les erreurs *a posteriori*. Cette gestion des Exception se fait par la construction `try ... except`.

```

def lireEntier():
    while True:
        try:
            chaine = raw_input('Entrez un entier: ') # Lecture du clavier
            # La conversion en type entier lève 'ValueError' si impossible
            return int(chaine)
        except ValueError:
            # Gestion de l'exception ValueError
            print "'%s' n'est pas un entier" % chaine

```

2. Par opposition au LBYL (Look Before You Leap) du C/C++, basé sur une série *exhaustive* de tests *a priori*.

```
>>> lireEntier()
Entrez un entier: toto
'toto' n'est pas un entier
Entrez un entier: 3,4
'3,4' n'est pas un entier
Entrez un entier: 4
4
```

Dans l'élaboration d'un programme, gérez explicitement les erreurs que vous auriez pu tester *a priori* et pour lesquels il existe une solution évidente, et laissez passer les autres (ce qui provoquera finalement l'interruption du programme)

Attention : Évitez à tout prix les `except nu`, càd ne spécifiant pas la ou les exceptions à gérer, car ils intercepteraient alors *toutes* les exceptions, y compris celles que vous n'aviez pas prévues !

```
>>> y = 2
>>> try:
...     x = z                # Copie y dans x
...     print "Tout va bien"
... except:
...     print "Rien ne va plus"
```

Vos procédures doivent également générer des exceptions (*documentées*) – avec l'instruction `raise Exception` – si elles ne peuvent conclure leur action, à charge pour la procédure appelante de les gérer :

```
def diffsq(x,y):
    """Returns x**2 - y**2 for x >= y, raises ValueError if x < y."""

    if x < y:
        raise ValueError("x=%f < y=%f" % (x,y))

    return x**2 - y**2
```

Avant de se lancer dans un calcul long et complexe, on peut vouloir tester la validité de certaines hypothèses fondamentales, soit par une structure `if ... raise`, ou plus facilement à l'aide d'`assert` (qui, si l'hypothèse n'est pas vérifiée, génère une `AssertionError`) :

```
def diffsq(x,y):
    """Returns x**2 - y**2 for x >= y."""

    assert x >= y, "x=%f < y=%f" % (x,y) # Test et msg d'erreur
    return x**2 - y**2
```

Note : La règle générale à retenir concernant la gestion des erreurs : **Fail early, fail often, fail better !**

2.10 Entrées-sorties

2.10.1 Interactif

Comme nous avons pu le voir précédemment, l'affichage à l'écran se fait par `print`, la lecture du clavier par `raw_input`.

2.10.2 Fichiers

La gestion des fichiers (lecture et écriture) se fait à partir de la fonction `open` retournant un objet `file` :

```

# ===== ÉCRITURE =====
outfile = open("carres.dat", 'w') # Ouverture du fichier "carres.dat" en écriture
for i in range(1,10):
    outfile.write("%d %d\n" % (i, i**2)) # Noter la présence du '\n' (non-automatique)
outfile.close() # Fermeture du fichier (nécessaire)

# ===== LECTURE =====
infile = open("carres.dat") # Ouverture du fichier "carres.dat" en lecture
for line in infile: # Boucle sur les lignes du fichier
    if line.strip().startswith('#'): # Ne pas considérer les lignes "commentées"
        continue
    try: # Essayons de lire 2 entiers sur cette ligne
        x,x2 = [ int(tok) for tok in line.split() ]
    except ValueError: # Gestion des erreurs
        print "Cannot decipher line '%s'" % line
        continue
    print "%d**3 = %d" % (x, x**3)

```

2.11 Éléments passés sous silence

Cette (brève) introduction à Python se limite à des fonctionnalités relativement simples et anciennes – disons 2.5 – du langage. De nombreuses fonctionnalités du langage n'ont pas été abordées (y compris les développements introduits dans les dernières versions de la branche 2.x) :

- Variables globales
- Arguments anonymes : `*args` and `**kwargs`
- Fonction anonyme : `lambda x,y: x+y`
- Itérateurs et générateurs : `yield`
- Gestion de contexte : `with ... as` (**PEP 243**)
- Nouveau système de formatage de chaîne : `str.format`
- Décorateurs : fonction sur une fonction ou une classe (`@staticmethod`, etc.)
- Héritages multiples et méthodes de résolution
- Etc.

Ces fonctionnalités peuvent évidemment être très utiles, mais ne sont généralement pas strictement indispensables pour une utilisation « courante » de Python dans un contexte scientifique.

Bibliothèque standard

Table des matières

- Bibliothèque standard
 - Gestion des arguments/options de la ligne de commande
 - [c]Pickle : sérialisation des données
 - *Batteries included*
 - *Text/Graphical User Interfaces*

Python dispose d'une très riche bibliothèque de modules étendant les capacités du langage dans de nombreux domaines : nouveaux types de données, interactions avec le système, gestion des fichiers et des processus, protocoles de communication (internet, mail, FTP, etc.), multimédia, etc.

- The Python Standard Library (v2.7)
- Python by Example !

3.1 Gestion des arguments/options de la ligne de commande

Le module `sys` permet un accès direct aux arguments de la ligne de commande, via la liste `sys.argv` : `sys.argv[0]` contient le nom du script exécuté, `sys.argv[1]` le nom du 1er argument (s'il existe), etc.

Exemple : *crible.py*

```
import sys

if sys.argv[1:]: # Présence d'au moins un argument sur la ligne de commande
    try:
        n = int(sys.argv[1]) # Essayer de lire le 1er argument comme un entier
    except ValueError:
        raise ValueError("L'argument '%s' n'est pas un entier" % sys.argv[1])
else:
    # Pas d'argument sur la ligne de commande
    n = 101 # Valeur par défaut
```

Pour une gestion avancée des arguments et/ou options de la ligne de commande, il est préférable d'utiliser le module `optparse`.

– **Exemple :** *koch.py*

```
from optparse import OptionParser

desc = u"Tracé (via 'turtle') d'un flocon de Koch d'ordre arbitraire."

# Définition des options
parser = OptionParser(usage="%prog [options] ordre",
                      version=__version__, description=desc)
parser.add_option("-t", "--taille", type=int,
```

```
        help="Taille [%default px]",
        default=500)
parser.add_option("-d", "--delta", type=float,
        help="Delta [%default deg]",
        default=0.)
parser.add_option("-f", "--figure",
        help="Figure de sortie (format EPS)")
parser.add_option("-T", "--turbo",
        action="store_true", default=False,
        help="Mode Turbo")

# Déchiffrage des options et arguments
opts,args = parser.parse_args()

# Quelques tests sur les args et options
try:
    niveau = int(args[0])
    assert niveau >= 0
except (IndexError,                # Pas d'argument
        ValueError,               # Argument non-entier
        AssertionError):          # Argument entier < 0
    parser.error("Niveau d'entrée %s invalide" % args)

if opts.taille < 0:
    parser.error("La taille de la figure doit être positive")
```

– Génère automatiquement une aide en ligne :

```
$ python koch.py -h
Usage: koch.py [options] ordre
```

Tracé (via 'turtle') d'un flocon de Koch d'ordre arbitraire.

Options:

```
--version          show program's version number and exit
-h, --help         show this help message and exit
-t TAILLE, --taille=TAILLE
                   Taille [500 px]
-d DELTA, --delta=DELTA
                   Delta [0.0 deg]
-f FIGURE, --figure=FIGURE
                   Figure de sortie (format EPS)
-T, --turbo        Mode Turbo
```

3.2 [c]Pickle : sérialisation des données

Les modules `pickle`/`cPickle` permettent la sauvegarde pérenne d'objets python (« sérialisation »).

```
>>> d = dict(a=1, b=2, c=3)
>>> l = range(10000)
>>> import cPickle as pickle          # 'cPickle' est + rapide que 'pickle'
>>> pk1 = open('archive.pkl', 'w')    # Overture du fichier en écriture
>>> pickle.dump((d,l), pk1, protocol=-1) # Sérialisation du tuple (d,l)
>>> pk1.close()                       # *IMPORTANT!* Fermeture du fichier
>>> d2,l2 = pickle.load(open('archive.pkl')) # Désérialisation (relecture)
>>> d==d2
True
>>> l==l2
True
```

Attention : les *pickles* ne sont pas un format d'échange de données. Ils sont spécifiques à python, et peuvent dépendre de la machine utilisée.

3.3 Batteries included

- `math` : accès aux fonctions mathématiques (réelles)

```
>>> math.asin(math.sqrt(2)/2)/math.pi*180
45.00000000000001
```

- Interaction système
 - `sys, os` : interface système
 - `shutil` : opérations sur les fichiers (*copy, move, etc.*)
 - `subprocess` : exécution de commandes système
 - `glob` : méta-caractères du *shell* (p.ex. `toto?.*`)
- Expressions régulières : `re`
- Gestion du temps (`time`) et des dates (`datetime, calendar`)
- Fichiers compressés et archives : `gzip, bzip2, zipfile, tarfile`
- Lecture & sauvegarde des données (outre `pickle/cPickle`)
 - `csv` : lecture/sauvegarde de fichiers CSV (Comma Separated Values)
 - `ConfigParser` : fichiers de configuration
- Lecture d'une URL (p.ex. page web) : `urllib2`

3.4 Text/Graphical User Interfaces

- TUI (Text User Interface) : `curses`
- GUI (Graphical User Interface) : `Tkinter` (voir également `easygui`),
- Bibliothèques externes : `PyGTK, PyQt, wxPython`

Bibliothèques numériques

Table des matières

- Bibliothèques numériques
 - Numpy
 - Tableaux
 - Création de tableaux
 - Manipulations sur les tableaux
 - Opérations de base
 - Tableaux évolués
 - Entrées/sorties
 - Sous-modules
 - Scipy
 - Quelques exemples complets
 - Bibliothèques graphiques
 - Matplotlib (2D/3D)
 - Mayavi (3D)
 - Autres librairies scientifiques

- Standard Numeric and Mathematical Modules

4.1 Numpy

- Référence Numpy
- Tutoriel numpy
- Exemples annotés

numpy est une bibliothèque *numérique* apportant le support efficace de larges tableaux multidimensionnels, et de routines mathématiques de haut niveau (fonctions spéciales, algèbre linéaire, statistiques, etc.).

Note : La convention d'import utilisé par la suite est `import numpy as N`.

4.1.1 Tableaux

Un `ndarray` (généralement appelé `array`) est un tableau multidimensionnel *homogène* : tous les éléments doivent avoir le même type, en général numérique. Les différentes dimensions sont appelées des *axes*, tandis que le nombre de dimensions – 0 pour un scalaire, 1 pour un vecteur, 2 pour une matrice, etc. – est appelé le *rang*.

```
>>> import numpy as N      # Import de la bibliothèque numpy avec le surnom N
>>> a = N.array([1,2,3])   # Création d'un array 1D à partir d'une liste d'entiers
>>> a.ndim                 # Rang du tableau
```

```
1
>>> a.shape          # Format du tableau: par définition, len(shape)=ndim
(3,)                 # Vecteur 1D de longueur 3
>>> a.dtype          # Type des données du tableau
dtype('int32')       # Python 'int' = numpy 'int32' = C 'long'
>>> # Création d'un tableau 2D de float (de 0. à 12.) de shape 4x3
>>> b = N.arange(12, dtype=float).reshape(4,3); b
array([[ 0.,  1.,  2.],
       [ 3.,  4.,  5.],
       [ 6.,  7.,  8.],
       [ 9., 10., 11.]])
>>> b.shape          # Nb d'éléments le long de chacune des dimensions
(4, 3)              # 4 lignes, 3 colonnes
>>> b.size           # Nb *total* d'éléments dans le tableau
12                  # Par définition, size=prod(shape)
>>> b.dtype          # Python 'float' = numpy 'float64' = C 'double'
dtype('float64')
```

Création de tableaux

- `N.array` : convertit une liste d'éléments homogènes ou coercitibles

```
>>> N.array([[1.,2.],[3,4]]) # Liste de listes d'entiers et de réels
array([[ 1.,  2.],
       [ 3.,  4.]])          # Tableau 2D de réels
```

- `N.zeros` (resp. `N.ones`) : crée un tableau de format donné rempli de zéros (resp. de uns)

```
>>> N.zeros((2,1))      # Shape (2,1): 2 lignes, 1 colonne, float par défaut
array([[ 0.],
       [ 0.]])
>>> N.ones((1,2), dtype=bool) # Shape (1,2): 1 ligne, 2 colonnes, type booléen
array([[True, True]], dtype=bool)
```

- `N.arange` : crée une séquence de nombres, en spécifiant éventuellement le *start*, le *end* et le *step* (similaire à `range` pour les listes)

```
>>> N.arange(10, 30, 5)   # De 10 à 30 (exclu) par pas de 5, type entier par défaut
array([10, 15, 20, 25])
>>> N.arange(0.5, 2.1, 0.3) # Accepte des réels en argument, DANGER!
array([ 0.5,  0.8,  1.1,  1.4,  1.7,  2. ])
```

- `N.linspace` : répartition uniforme d'un nombre fixe de points entre un *start* et un *end* (préférable à `N.arange` sur des réels).

```
>>> N.linspace(0, 2*N.pi, 5) # 5 nb entre 0 et 2π (inclus), type réel par défaut
array([ 0.,  1.57079633,  3.14159265,  4.71238898,  6.28318531])
```

- `N.meshgrid(x,y)` est similaire à `N.linspace` en 2D.

```
>>> # 5 points entre 0 et 2 en "x", et 3 entre 0 et 1 en "y"
>>> z = N.meshgrid(N.linspace(0,2,5), N.linspace(0,1,3)); z
(array([[ 0. ,  0.5,  1. ,  1.5,  2. ], # Tableau 2D des x
       [ 0. ,  0.5,  1. ,  1.5,  2. ],
       [ 0. ,  0.5,  1. ,  1.5,  2. ]]),
 array([[ 0. ,  0. ,  0. ,  0. ,  0. ], # Tableau 2D des y
       [ 0.5,  0.5,  0.5,  0.5,  0.5],
       [ 1. ,  1. ,  1. ,  1. ,  1. ]]))
```

- `N.mgrid` permet de générer des rampes d'indices (entiers) ou de coordonnées (réels) de rang arbitraire avec une notation spéciale faisant appel aux *Index tricks*. Équivalent à `N.linspace` en 1D et *similaire (mais différent)* à `N.meshgrid` en 2D.

```

>>> N.mgrid[0:4:2,1:5:3] # Grille 2D d'indices (entiers)
array([[0, 0],          # 0:4:2=[0,2] selon l'axe 0
       [2, 2]],
       [[1, 4],          # 1:5:3=[1,4] selon l'axe 1
       [1, 4]])
>>> N.mgrid[0:2*N.pi:5j] # Rampe de coordonnées (réels): 5 nb de 0 à 2π (inclus)
array([ 0.,  1.57079633,  3.14159265,  4.71238898,  6.28318531])
>>> # 3 points entre 0 et 1 selon l'axe 0, et 5 entre 0 et 2 selon l'axe 1
>>> z = N.mgrid[0:1:3j, 0:2:5j]; z
array([[ 0.,  0.,  0.,  0.,  0.], # Axe 0 variable, axe 1 constant
       [ 0.5,  0.5,  0.5,  0.5,  0.5],
       [ 1.,  1.,  1.,  1.,  1.]],
       [[ 0.,  0.5,  1.,  1.5,  2.], # Axe 0 constant, axe 1 variable
       [ 0.,  0.5,  1.,  1.5,  2.],
       [ 0.,  0.5,  1.,  1.5,  2.]])
>>> z.shape
(2,3,5) # 2 plans 2D (x & y) × 3 lignes (y) × 5 colonnes (x)
>>> N.mgrid[0:1:5j, 0:2:7j, 0:3:9j].shape
(3, 5, 7, 9) # 3 volumes 3D (x, y, z) × 5 plans (z) × 7 lignes (y) × 9 colonnes (x)

```

Attention : à l'ordre de variation des indices dans les tableaux multidimensionnel, et aux différences entre `meshgrid` et `mgrid`.

- `N.random.rand` crée un tableau d'un format donné de réels aléatoires dans $[0,1[$; `N.random.randn` génère un tableau d'un format donné de réels tirés aléatoirement d'une distribution gaussienne (normale) standard ($\mu=0, \sigma=1$).
Le sous-module `numpy.random` fournit des générateurs de nombres aléatoires pour de nombreuses distributions discrètes et continues.

Manipulations sur les tableaux

Les `array` 1D sont indexables comme les listes standard. En dimension supérieure, chaque axe est indéxable indépendamment.

```

>>> x = N.arange(10); # Rampe 1D
>>> x[1::3] *= -1; x # Modification "in-place"
array([ 0, -1,  2,  3, -4,  5,  6, -7,  8,  9])

```

Slicing

Les « tranches » de rang $< N$ d'un tableau de rang N sont appelées *slices* : le ou les axes selon lesquels la *slice* a été découpée, devenus de longueur 1, sont éliminés.

```

>>> y = N.arange(2*3*4).reshape(2,3,4); y # 2 plans, 3 lignes, 4 colonnes
array([[[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]],
       [[12, 13, 14, 15],
       [16, 17, 18, 19],
       [20, 21, 22, 23]])
>>> y[0,1,2] # 1er plan (axe 0), 2ème ligne (axe 1), 3ème colonne (axe 2)
6 # scalaire, shape *()**, ndim 0
>>> y[0,1] # 1er plan (axe 0), 2ème ligne (axe 1) = y[0,1,:]
array([4, 5, 6, 7]) # Shape (4,)
>>> y[0] # 1ère slice (3,4) selon le 1er axe = y[0,:,:]
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>> y[0][1][2] # 1er plan (axe 0), 2ème ligne (axe 1), 3ème colonne (axe 2)

```

```
6
>>> y[:, -1]      # Dernière slice (2,4) selon le 2ème axe = y[:,2,:]
array([[ 8,  9, 10, 11],
       [20, 21, 22, 23]])
>>> y[:, :, 0]     # 1ère slice (2,3) selon le 3ème axe = y[:, :, 0]
array([[ 0,  4,  8],
       [12, 16, 20]])
>>> # On peut vouloir garder explicitement la dimension "tranchée"
>>> y[:, :, 0:1]   # 1ère slice (2,3) selon le 3ème axe *en gardant le format original*
array([[[ 0],
        [ 4],
        [ 8]],
       [[12],
        [16],
        [20]]])
>>> y[:, :, 0:1].shape
(2, 3, 1)  # L'axe 2 a été conservé, il ne contient pourtant qu'un seul élément
```

Modification de format

```
>>> # *reshape* modifie le format sans modifier le nb total d'éléments
>>> y = N.arange(6).reshape(2,3); y # Shape (6,) → (2,3) (*size* inchangé)
array([[0, 1, 2],
       [3, 4, 5]])
>>> y.reshape(2,4) # Format incompatible (*size* serait modifié)
ValueError: total size of new array must be unchanged
>>> y.ravel()      # *ravel* déroule tous les axes (*1st axis slowest*)
array([ 0, 1, 2, 3, 4, 5]) # Shape (6,) = 2 × 3 = *size*
>>> y.ravel('F')   # Ordre 'Fortran' (*last axis slowest*)
array([0, 3, 1, 4, 2, 5])

>>> y.T           # Transposition = y.transpose() (voir aussi *rollaxis*)
array([[0, 3],
       [1, 4],
       [2, 5]])

>>> y[:, :, 0:1].squeeze() # *squeeze* élimine les axes de dimension 1 (anton: *expand_dims*)
array([0, 3])

>>> # *resize* modifie le format en modifiant le nb total d'éléments
>>> N.resize(N.arange(4), (2,4)) # Complétion avec des copies du tableau
array([[0, 1, 2, 3],
       [0, 1, 2, 3]])
>>> N.resize(N.arange(4), (4,2))
array([[0, 1],
       [2, 3],
       [0, 1],
       [2, 3]])
```

Attention : `N.resize(arr, shape)` (complétion avec des copies de `arr`) est différent de `arr.resize(shape)` (complétion avec des 0).

Stacking

```
>>> a = N.arange(5); a
array([0, 1, 2, 3, 4])
>>> N.hstack((a,a)) # Stack horizontal (le long des colonnes)
```

```

array([0, 1, 2, 3, 4, 0, 1, 2, 3, 4])
>>> N.vstack((a,a)) # Stack vertical (le long des lignes)
array([[0, 1, 2, 3, 4],
       [0, 1, 2, 3, 4]])
>>> N.dstack((a,a)) # Stack en profondeur (le long des plans)
array([[[0, 0],
        [1, 1],
        [2, 2],
        [3, 3],
        [4, 4]]])

```

Broadcasting

L'array **broadcasting** définit les règles selon lesquelles deux tableaux de formats *différents* peuvent éventuellement s'apparier.

```

>>> a = N.arange(6).reshape(2,3) # Shape (2,3)
>>> b = N.array([10,20,30])      # Shape (3,) ~ (1,3) → (2,3) = (1,3) copié 2 fois
>>> a+b
array([[10, 21, 32],
       [13, 24, 35]])           # Shape (2,3)
>>> c = N.array([10,20])         # Shape (2,) ~ (1,2) incompatible avec (2,3)
>>> a+c
ValueError: shape mismatch: objects cannot be broadcast to a single shape
>>> a+c[:,N.newaxis]            # = c.reshape(-1,1) → (2,1) → (2,3) = (2,1) copié 3 fois
array([[10, 11, 12],
       [23, 24, 25]])

```

Indexation évoluée

```

>>> a = N.linspace(-1,1,5); a
array([-1. , -0.5,  0. ,  0.5,  1. ])
>>> a>=0 # Test logique: tableau de booléens
array([False, False,  True,  True,  True], dtype=bool)
>>> (a>=0).nonzero() # Indices des éléments ne s'évaluant pas à False
(array([2, 3, 4]),) # Indices des éléments >= 0
>>> a[(a>=0).nonzero()] # Indexation par un tableau d'indices
array([ 0. ,  0.5,  1. ])
>>> a[a>=0] # Indexation par un tableau de booléens, **préférable**
array([ 0. ,  0.5,  1. ])
>>> a[a<0] -= 10; a # Soustrait 10 à toutes les valeurs <0 (= where(a<0, a-10, a))

```

Opérations de base

```

>>> a = N.arange(3); a # Shape (3,), type *int*
array([0, 1, 2])
>>> b = 1. # ~ Shape (), type *float*
>>> c = a+b; c # *Broadcasting*: () → (1,) → (3,)
array([ 1.,  2.,  3.]) # *Upcasting*: int → float
>>> a += 1; a # Modification *in-place* (plus efficace si possible)
array([ 1.,  2.,  3.])
>>> a.mean() # *ndarray* dispose de nombreuses méthodes numériques de base
2.0

```

Opérations sur les axes

```
>>> x = N.random.permutation(6).reshape(3,2); x # 3 lignes, 2 colonnes
array([[3, 4],
       [5, 1],
       [0, 2]])
>>> x.min(axis=None) # minimum global (comportement par défaut)
0
>>> x.min(axis=0)     # minima le long de l'axe 0 (i.e. l'axe des lignes)
array([0, 1])        # ce sont donc les minima par colonne (2)
>>> x.min(axis=1)     # minima le long de l'axe 1 (i.e. l'axe des colonnes)
array([3, 1, 0])     # ce sont donc les minima par ligne (3)
```

Opérations matricielles

Les opérations de base s'appliquent sur les *éléments* des tableaux, et n'ont pas une signification matricielle par défaut :

```
>>> m = N.arange(4).reshape(2,2); m # Tableau de rang 2
array([[0, 1],
       [2, 3]])
>>> i = N.identity(2, dtype=int); i # Tableau "identité" de rang 2 ~ N.eye
array([[1, 0],
       [0, 1]])
>>> m*i                                # Attention! opération * sur les éléments
array([[0, 0],
       [0, 3]])
>>> N.dot(m,i)                        # dot = multiplication matricielle sur des tableaux
array([[0, 1],
       [2, 3]])
>>> # Utiliser 'N.matrix' pour des opérations matricielles par défaut
>>> N.matrix(m)*N.matrix(i) # Opération * entre matrices
matrix([[0, 1],
        [2, 3]])
```

Le sous-module `numpy.linalg` fournit des outils spécifiques au calcul matriciel (inverse, déterminant, valeurs propres, etc.).

Ufuncs

`numpy` fournit les fonctions mathématiques de base (`exp`, `atan2`, etc.) qui s'appliquent sur les éléments des tableaux d'entrée :

```
>>> x = N.linspace(0,2*N.pi,5) # [0,  $\pi/2$ ,  $\pi$ ,  $3\pi/2$ ,  $2\pi$ ]
>>> y = N.sin(x); y             #  $\sin(x) = [0, 1, 0, -1, 0]$ 
array([ 0.00000000e+00,  1.00000000e+00,  1.22460635e-16,
        -1.00000000e+00, -2.44921271e-16])
>>> y == [0,1,0,-1,0]          # Test d'égalité stricte (élément par élément)
array([ True,  True, False,  True, False], dtype=bool) # Attention! aux calculs en réels (float)
>>> N.all(N.sin(x)==[0,1,0,-1,0]) # Test d'égalité stricte de tous les éléments
False
>>> N.allclose(y, [0,1,0,-1,0]) # Test d'égalité numérique de tous les éléments
True
```

Exercices : *Median Absolute Deviation, Distribution du pull*

4.1.2 Tableaux évolués

Types composés

Outre les types scalaires élémentaires – `bool`, `int`, `float`, `complex`, `str`, etc. – `numpy` supporte les tableaux de types composés :

```
>>> dt = N.dtype([('nom','S10'),      # 1er élément: chaîne de 10 caractères
...              ('age','i'),        # 2ème élément: entier
...              ('taille','d')])    # 3ème élément: réel (double)
>>> arr = N.array([('Calvin',6,1.20),('Hobbes',5,1.80)], dtype=dt); arr
array([('Calvin', 6, 1.2), ('Hobbes', 6, 1.8)],
      dtype=[('nom', '<S10'), ('age', '<i4'), ('taille', '<f8')])
>>> arr[0]                          # Accès par élément
('Calvin', 6, 1.2)
>>> arr['nom']                      # Accès par sous-type
array(['Calvin', 'Hobbes'], dtype='<S10')
>>> rec = arr.view(N.recarray); arr # Vue de type 'recarray'
rec.array([('Calvin', 6, 1.2), ('Hobbes', 5, 1.8)],
          dtype=[('nom', '<S10'), ('age', '<i4'), ('taille', '<f8')])
>>> rec.nom                         # Accès direct par attribut
chararray(['Calvin', 'Hobbes'], dtype='<S10')
```

Tableaux masqués

Le sous-module `numpy.ma` ajoute le support des tableaux masqués (*Masked Arrays*). Imaginons un tableau (4,5) de réels (positifs ou négatifs), sur lequel nous voulons calculer pour chaque colonne la moyenne des éléments positifs uniquement :

```
>>> x = N.random.randn(4,5); x
array([[ -0.55867715,  1.58863893, -1.4449145 ,  1.93265481, -0.17127422],
       [ -0.86041806,  1.98317832, -0.32617721,  1.1358607 , -1.66150602],
       [ -0.88966893,  1.36185799, -1.54673735, -0.09606195,  2.23438981],
       [ 0.35943269, -0.36134448, -0.82266202,  1.38143768, -1.3175115 ]])
>>> x[x>=0]                        # Donne les éléments >0 du tableau, sans leurs indices
array([ 1.58863893,  1.93265481,  1.98317832,  1.1358607 ,  1.36185799,
        2.23438981,  0.35943269,  1.38143768])
>>> (x>=0).nonzero()               # Donne les indices ([i],[j]) des éléments >0
(array([0, 0, 1, 1, 2, 2, 3, 3]), array([1, 3, 1, 3, 1, 4, 0, 3]))
>>> y = N.ma.masked_less(x, 0); y  # Tableau où les éléments <0 sont masqués
masked_array(data =
  [[-- 1.58863892701 -- 1.93265481164 --] # Données
  [-- 1.98317832359 -- 1.13586070417 --]
  [-- 1.36185798574 -- -- 2.23438980788]
  [0.359432688656 -- -- 1.38143767743 --]],
            mask =
  [[ True False  True False  True]
  [ True False  True False  True]
  [ True False  True  True False]
  [False  True  True False  True]],
            fill_value = 1e+20)
>>> m0 = y.mean(axis=0); m0        # Moyenne sur les lignes (axe 0)
masked_array(data = [0.359432688656 1.64455841211 -- 1.48331773108 2.23438980788],
            mask = [False False  True False False],
            fill_value = 1e+20)      # Le résultat est un *Masked Array*
>>> m0.filled(-1)                 # Conversion du *Masked Array* en tableau normal
array([ 0.35943269,  1.64455841, -1.         ,  1.48331773,  2.23438981])
```

4.1.3 Entrées/sorties

numpy peut lire (`N.loadtxt`) ou sauvegarder (`N.savetxt`) des tableaux dans un simple fichier ASCII :

```
>>> x = N.linspace(-1,1,100)
>>> N.savetxt('archive_x.dat', x)      # Sauvegarde dans le fichier 'archive_x.dat'
>>> y = N.loadtxt("archive_x.dat")     # Relecture à partir du fichier 'archive_x.dat'
>>> (x==y).all()
True
```

Attention : `N.loadtxt` supporte les types composés, mais ne supporte pas les données manquantes ; utiliser la fonction `N.genfromtxt` dans ce cas.

Le format ASCII n'est pas optimale pour de gros tableaux : il peut alors être avantageux d'utiliser le format binaire `.npy`, beaucoup plus compact (mais non *human readable*) :

```
>>> x = N.linspace(-1,1,1e6)
>>> N.save('archive_x.npy', x)         # Sauvegarde dans le fichier 'archive_x.npy'
>>> y = N.load("archive_x.npy")        # Relecture à partir du fichier 'archive_x.npy'
>>> (x==y).all()
True
```

Il est enfin possible de *sérialiser* les tableaux à l'aide de la bibliothèque standard [*\[c\]*Pickle](#).

4.1.4 Sous-modules

numpy fournit en outre quelques fonctionnalités supplémentaires, parmi lesquelles les sous-modules suivants :

- `numpy.random` : générateurs de nombres aléatoires
- `numpy.fft` : *Discrete Fourier Transform*
- `numpy.lib.polynomial` : manipulation des polynômes (racines, polynômes orthogonaux, etc.)

4.2 Scipy

scipy est une bibliothèque *numérique*¹ d'algorithmes et de fonctions mathématiques, basée sur les tableaux numpy, complétant ou améliorant (en terme de performances) les fonctionnalités numpy. P.ex. :

- Fonctions spéciales : `scipy.special` (fonctions de Bessel, erf, gamma, etc.)
- Intégration numérique : `scipy.integrate` (intégration numérique ou d'équations différentielles)
- Méthodes d'optimisation : `scipy.optimize` (minimisation, moindres-carrés, zéros d'une fonction, etc.)
- Interpolation : `scipy.interpolate` (interpolation, splines)
- Transformées de Fourier : `scipy.fftpack`
- Traitement du signal : `scipy.signal` (convolution, corrélation, filtrage, ondelettes, etc.)
- Algèbre linéaire : `scipy.linalg`
- Statistiques : `scipy.stats` (fonctions et distributions statistiques)
- Traitement d'images multi-dimensionnelles : `scipy.ndimage`
- Entrées/sorties : `scipy.io`

Liens :

- [Référence Scipy](#)
- [Cookbook Scipy](#)
- [Scipy Course Outline](#)

– Calcul numérique haut-niveau 

Voir également :

- [Pandas](#) : structuration et analyse de données
- [Scikits](#) : modules supplémentaires, parmi lesquels :
 - [scikit-learn](#) : *machine learning*
 - [scikits-image](#) : *image processing*

1. Python dispose également d'une librairie de calcul *formel*, *sympy*, et d'un environnement de calcul mathématique, *sage*.

- statsmodel : modèles statistiques
- Scipy Topical softwares
- Numpy/Scipy projects

Exercice : *Quartet d'Anscombe*

4.2.1 Quelques exemples complets

- Interpolation
- Integration (intégrales numériques, équations différentielles)
 - Oscillation amortie
 - Zombie Apocalypse
- Optimisation (moindres carrés, ajustements, recherche de zéros)
- Traitement du signal (splines, convolution, filtrage)
- Algèbre linéaire (systèmes linéaires, moindres carrés, décompositions)
- Statistiques (variables aléatoires, distributions, tests)

4.3 Bibliothèques graphiques

4.3.1 Matplotlib (2D/3D)

Matplotlib est une bibliothèque graphique de visualisation 2D (et marginalement 3D), avec support interactif et sorties de haute qualité.

Note : Utiliser **ipython -pylab** pour l'utilisation interactive des figures.

Il existe deux interfaces pour deux types d'utilisation :

- pylab : interface procédurale, très similaire à MATLAB et généralement réservée à l'analyse interactive :

```
>>> from pylab import *          # À NE PAS FAIRE DANS UN SCRIPT!
>>> x = linspace(-pi, pi, 100)   # pylab importe numpy dans l'espace courant
>>> plot(x, sin(x))              # Trace la courbe y = y(x)
>>> show()                       # Affiche le résultat
```

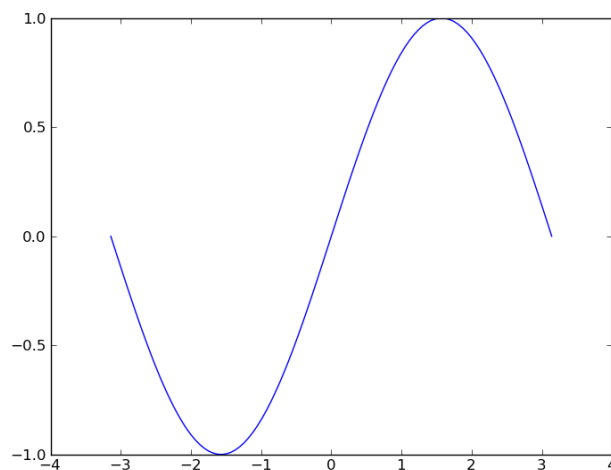
- pyplot : interface orientée objet, généralement pour les scripts :

```
import numpy as N
import matplotlib.pyplot as P

x = N.linspace(-N.pi, N.pi, 100)
y = N.sin(x)

fig, ax = P.subplots()
ax.plot(x, y)
P.show()
```

Dans les deux cas, le résultat est le même :



Liens

- [Tutorial matplotlib](#)
- [Gallery](#)
- [Examples](#)

Exemple : *filtres2ndOrdre.py*

Exercices : *Ensemble de Julia, Diagramme de bifurcation : la suite logistique*

4.3.2 Mayavi (3D)

[mayavi.mlab](#) est une bibliothèque graphique de visualisation 3D s'appuyant sur [Mayavi](#).

Note : Utiliser **ipython -wthread** pour l'utilisation interactive des figures.

Voir également :

- [VPython](#) : *3D Programming for Ordinary Mortals*

4.4 Autres librairies scientifiques

Une liste **non-exhaustive** de librairies scientifiques dans différents domaines de la physique/chimie :

- Mécanique quantique : [QuTiP](#)
- Électromagnétisme : [EMpy](#)
- Astronomie : [astropy](#), [Kapteyn](#) et autres modules astronomiques
- *PDE solver* : [FiPy](#), [SfePy](#)
- *Machine Learning* : [mlpy](#), [milk](#), [MDP](#) et autres modules d'intelligence artificielle
- Calcul symbolique : [sympy](#) (voir également ce [tutoriel sympy](#))
- [PyROOT](#)
- Etc.

Développer en python

Table des matières

- Développer en python
 - Le zen du python
 - Us et coutumes
- Outils de développement
 - *Integrated Development Environment*
 - Vérification du code
 - Documentation
 - Les méthodes de tests
 - Débugage et optimisation
 - *Python packages*
 - Python 2 vs. python 3

5.1 Le zen du python

Le *zen du Python* (**PEP 20**) est une série de 20 aphorismes ¹ donnant les grands principes du Python :

```
>>> import this
```

1. Beautiful is better than ugly.
2. Explicit is better than implicit.
3. Simple is better than complex.
4. Complex is better than complicated.
5. Flat is better than nested.
6. Sparse is better than dense.
7. Readability counts.
8. Special cases aren't special enough to break the rules.
9. Although practicality beats purity.
10. Errors should never pass silently.
11. Unless explicitly silenced.
12. In the face of ambiguity, refuse the temptation to guess.
13. There should be one– and preferably only one –obvious way to do it.
14. Although that way may not be obvious at first unless you're Dutch.
15. Now is better than never.

1. Dont seulement 19 ont été écrits.

16. Although never is often better than *right* now.
17. If the implementation is hard to explain, it's a bad idea.
18. If the implementation is easy to explain, it may be a good idea.
19. Namespaces are one honking great idea – let's do more of those !

Une traduction libre en français :

1. Préférer le beau au laid,
2. l'explicite à l'implicite,
3. le simple au complexe,
4. le complexe au compliqué,
5. le déroulé à l'imbriqué,
6. l'aéré au compact.
7. La lisibilité compte.
8. Les cas particuliers ne le sont jamais assez pour violer les règles,
9. même s'il faut privilégier l'aspect pratique à la pureté.
10. Ne jamais passer les erreurs sous silence,
11. ou les faire taire explicitement.
12. Face à l'ambiguïté, ne pas se laisser tenter à deviner.
13. Il doit y avoir une – et si possible une seule – façon évidente de procéder,
14. même si cette façon n'est pas évidente à première vue, à moins d'être Hollandais.
15. Mieux vaut maintenant que jamais,
16. même si jamais est souvent mieux qu'immédiatement.
17. Si l'implémentation s'explique difficilement, c'est une mauvaise idée.
18. Si l'implémentation s'explique facilement, c'est peut-être une bonne idée.
19. Les espaces de nommage sont une sacrée bonne idée, utilisons-les plus souvent !

5.1.1 Us et coutumes

- *Easier to Ask for Forgiveness than Permission* (`try ... except`)
- *Fail early, fail often, fail better!* (`raise`)
- le *Style Guide for Python Code* (**PEP 8**)
- *Idioms and Anti-Idioms in Python*
- *Code Like a Pythonista : Idiomatic Python*
- *Google Python Style Guide*

Quelques conseils supplémentaires

- « Ne réinventez pas la roue, sauf si vous souhaitez en savoir plus sur les roues » (Jeff Atwood²) : cherchez si ce que vous voulez faire n'a pas déjà été fait (éventuellement en mieux...), construisez dessus (en citant évidemment vos sources), et contribuez si possible !
- Codez *proprement* : commentez votre code, utilisez des noms de variable qui ont un sens, créez des objets si nécessaire, etc.
- Codez proprement *dès le début* : ne croyez pas que vous ne relirez jamais votre code (ou même que personne n'aura jamais à le lire), ou que vous aurez le temps de le refaire mieux plus tard...
- « L'optimisation prématurée est la source de tous les maux » (Donald Knuth³) : mieux vaut un code lent mais juste et maintenable qu'un code rapide et faux ou incompréhensible. Dans l'ordre absolu des priorités :
 1. *Make it work.*
 2. *Make it right.*
 3. *Make it fast.*
- *Respectez le zen du python*, il vous le rendra.

2. « *Don't reinvent the wheel, unless you plan on learning more about wheels* » – Jeff Atwood

3. « *Premature optimization is the root of all evil* » – Donald Knuth

5.2 Outils de développement

Je fournis ici essentiellement des liens vers des outils pouvant être utiles pour développer en python.

- [Best Practices, Development Methodologies and the Zen of Python](#)

5.2.1 Integrated Development Environment

- `idle`, l'IDE intégré à Python
- `emacs` + `python-mode` + `ropemacs` pour l'édition, et `ipython` pour l'exécution de code (voir [Python Programming In Emacs](#))
- `spyder`
- `PythonToolKit`
- Etc.

5.2.2 Vérification du code

Il s'agit d'outils permettant de vérifier *a priori* la validité syntaxique du code, de mettre en évidence des constructions dangereuses, les variables non-définies, etc. Ces outils ne testent pas nécessairement la validité des algorithmes et de leur mise en oeuvre...

- `pyflakes`
- `pychecker`
- `pylint`
- `pep8`

5.2.3 Documentation

- Outils de documentation, ou comment transformer automatiquement un code-source bien documenté en une documentation fonctionnelle.
 - `Sphinx`
 - `reStructuredText` for `Sphinx`
- Conventions de documentation :
 - Docstring convention : **PEP 257**
 - [Documenting Your Project Using Sphinx](#)
 - [A Guide to NumPy/SciPy Documentation](#)

5.2.4 Les méthodes de tests

Le but est de tester chaque partie du code, en incluant des tests directement dans la documentation ou en écrivant du code spécifique :

1. Les tests unitaires vérifient individuellement chacune des fonctions, méthodes, etc.
2. Les tests d'intégration évaluent les interactions entre différentes unités du programmes.
3. Les tests système assurent le bon fonctionnement du programme dans sa globalité.

- **doctest** : tests unitaires inclus à la documentation
- `unittest`
- `py.test`
- `nose`

5.2.5 Débugage et optimisation

Débugage

Les débogueurs permettent de « plonger » dans un code au cours d'exécution ou juste après une erreur (analyse post-mortem).

- Modules de la bibliothèque standard : `pdb`, `timeit`

Ainsi, pour déboguer un script, il est possible de l'exécuter sous le contrôle du débogueur `pdb` en s'interrompant dès la 1ère instruction :

```
python -m pdb script.py
(Pdb)
```

- Commandes `ipython` : `%run`, `%debug`, `%timeit`

Si un script exécuté sous `ipython` (commande `%run`) lève une exception, il est possible d'inspecter l'état de la mémoire au moment de l'erreur avec la commande `%debug`.

Profilage et optimisation

Warning : *Premature optimization is the root of all evil* – Donald Knuth

Le profilage permet de déterminer le temps passé dans chacune des sous-fonctions d'un code, afin d'y identifier les parties à optimiser.

- `python -O, __debug__, assert`
- [Performance tips](#)
- [Tutoriel de profilage](#)
- `cython` : *write C-extensions in python-like language* ([Cython for NumPy users](#))

5.2.6 Python packages


Comment installer/créer des modules externes :

- `pip`
- [Hitchhiker's Guide to Packaging](#)

5.2.7 Python 2 vs. python 3

- [Python 2 or python 3 ?](#)
- [Porting Python 2 Code to Python 3](#)

Références supplémentaires

Voici une liste très partielle de documents Python disponibles en ligne. La majorité des liens sont en anglais, quelques uns  sont en français.



6.1 Documentation générale

- Python
- The Python Package Index
- Python 2.7 Quick Reference
- Python Documentation
- Python Frequently Asked Questions

6.2 Forums de discussion

- numpy
- scipy
- matplotlib
- Python recipes

6.3 Listes de liens

- Python facile (2005) 
- Python références et astuces : (2009) 
- IPython en ligne
- IPython cookbook
- Starter Kit (py4science)



6.4 Livres libres

- How to Think Like a Computer Scientist
 - Wikibook
 - Interactive edition
- Dive into Python (*lien mort*)
- Non-Programmer's Tutorial for Python 2.6
- A Python Book
- Start Programming with Python



- ‘Python for Informatics : Exploring Information <<http://www.pythonlearn.com/>>’_
- Apprendre à programmer avec Python  (Wikilivre)
- Plongez au coeur de Python 

6.5 Cours en ligne

6.5.1 Python

- Cours Python 3 + exercices (voir également le cours d’algorithmique en C, et de easygui) 
- DIY python workshop
- Exemples de scripts Python 
- Google’s Python Class
- CheckIO, pour apprendre la programmation Python en s’amusant !

6.5.2 Scientifique

- Python Scientific Lecture Notes
- Handbook of the Physics Computing Course (2002)
- Practical Scientific Computing in Python
- Computational Physics with Python (avec exercices)
- SciPy 2011 tutorials : numpy, scipy, matplotlib, ipython
- Practical Python for Astronomers
- Lectures on Computational Economics (avec exercices)
- L’informatique scientifique avec Python 
- La programmation scientifique avec Python 

6.5.3 Autres

- Visualization

6.6 Sphinx & co.

- Sample doc

Exemples

7.1 Cercle circonscrit

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  __author__ = "Yannick Copin <y.copin@ipnl.in2p3.fr>"
5  __version__ = "Time-stamp: <2011-09-27 12:17:25 ycopin>"
6
7  """
8  Calcul du cercle circonscrit à 3 points du plan.
9  """
10
11 from math import sqrt, hypot
12
13 # Définition d'une classe =====
14
15 # start-classPoint
16 class Point(object): # *object* est la classe dont dérivent toutes les autres
17     """Classe définissant un 'Point' du plan, caractérisé par ses
18     coordonnées 'x', 'y'.
19     """
20
21     def __init__(self, x, y):
22         """Méthode d'instanciation à partir de deux coordonnées."""
23
24         try:                                # Convertit les coords en 'float'
25             self.x = float(x)
26             self.y = float(y)
27         except ValueError:
28             raise ValueError("Incompatible input coordinates")
29
30     def __str__(self):
31         """Surcharge de l'opérateur 'str': l'affichage *informel* de
32         l'objet dans l'interpréteur, p.ex. 'print self' sera résolu
33         comme 'self.__str__()'
34
35         Retourne une chaîne de caractères.
36         """
37
38         return "Point (x=%f, y=%f)" % (self.x, self.y)
39
40     def isOrigin(self):
41         """Méthode booléenne, vrai si le point est à l'origine."""
42
43         return ((self.x == 0) and (self.y == 0))
44

```

```
45     def distance(self, other):
46         """Méthode de calcul de la distance du point ('self') à un
47         autre ('other')."""
48
49         return hypot(self.x - other.x, self.y - other.y) # sqrt(dx2 + dy2)
50 # end-classPoint
51
52 # Définition du point origine O
53 O = Point(0,0)
54
55 # Héritage de classe =====
56
57 # start-classVector
58 class Vector(Point):
59     """Un 'Vector' hérite de 'Point' avec des méthodes additionnelles
60     (p.ex. l'addition)."""
61
62     def __init__(self, A, B=0):
63         """Définit le vecteur :math:\vec{AB} si B!=0, ou
64         :math:\vec{OA} sinon."""
65
66         # Initialisation de la classe parente
67         if B.isOrigin(): # B = O
68             Point.__init__(self, A.x, A.y)
69         else: # B != O
70             Point.__init__(self, B.x-A.x, B.y-A.y)
71
72         # Attribut propre à la classe dérivée
73         self.norm2 = self.x**2 + self.y**2 # Norme du vecteur au carré
74
75     def __str__(self):
76         """Surcharge de la fonction 'str()': ainsi, 'print self' sera
77         résolu comme 'Vector.__str__(self)' (et non pas comme
78         'Point.__str__(self)')
79         """
80
81         return "Vector (x=%f, y=%f)" % (self.x, self.y)
82
83     def __add__(self, other):
84         """Surcharge de l'opérateur '+': l'instruction 'self + other'
85         sera résolue comme 'self.__add__(other)'.
```

```

108     Retourne: centre (Point), rayon (float)"""
109
110     # Diamètre
111     m = N.distance(P)
112     n = P.distance(M)
113     p = M.distance(N)
114
115     d = (m+n+p) * (-m+n+p) * (m-n+p) * (m+n-p)
116     if d>0:
117         diam = 2*m*n*p / sqrt(d)
118     else:
119         raise ValueError("Undefined circumscribed circle diameter.")
120
121     # Centre
122     dm2 = Vector(M).norm2
123     dn2 = Vector(N).norm2
124     dp2 = Vector(P).norm2
125
126     MN = Vector(M,N)
127     NP = Vector(N,P)
128     PM = Vector(P,M)
129
130     d = -2*( M.x*NP.y + N.x*PM.y + P.x*MN.y )
131     if d==0:
132         raise ValueError("Undefined circumscribed circle center.")
133
134     x0 = -( dm2*NP.y + dn2*PM.y + dp2*MN.y ) / d
135     y0 = ( dm2*NP.x + dn2*PM.x + dp2*MN.x ) / d
136
137     return Point(x0,y0),diam/2.          # Centre,R
138
139
140 if __name__=='__main__':
141
142     # start-optparse
143     from optparse import OptionParser
144
145     parser = OptionParser(usage="%prog [-p/--plot] "
146                           "[-i/--input coordfile | x1,y1 x2,y2 x3,y3]",
147                           version=__version__)
148     parser.add_option("-i", "--input",
149                      help="Input coordinate file (one 'x,y' per line)")
150     parser.add_option("-p", "--plot",
151                      action="store_true", default=False,
152                      help="Plot circumscribed circle")
153
154     opts,args = parser.parse_args()
155     # end-optparse
156
157     if opts.input:          # Lecture du fichier d'entrée
158         try:
159             args = file(opts.input).readlines()
160         except IOError:
161             parser.error("Cannot read coordinates from %s" % opts.input)
162
163     if len(args) != 3:      # Vérifie le nb de points
164         parser.error("Specify 3 input points by their 'x,y' coordinates")
165
166     points = [None]*3       # Génère la liste de 3 points
167     for i,arg in enumerate(args):
168         try:
169             x,y = ( float(t) for t in arg.split(',') )
170         except ValueError:

```

```

171         parser.error("Cannot parse x,y coordinates '%s'" % arg)
172
173     points[i] = Point(x,y)
174     print "#%d: %s" % (i+1, str(points[i]))
175
176     # Calcul du cercle circonscrit
177     try:
178         center, radius = circumCircle(*points)
179         print "Circumscribed circle: %s, radius=%f" % (str(center), radius)
180     except ValueError:
181         raise ValueError("Undefined circumscribed circle")
182
183     if opts.plot:
184         # Figure
185         import matplotlib.pyplot as P
186
187         fig = P.figure()
188         ax = fig.add_subplot(1,1,1, aspect='equal', adjustable='datalim')
189         # Point
190         ax.plot([ p.x for p in points], [ p.y for p in points ], 'ko')
191         # Cercle circonscrit
192         c = P.matplotlib.patches.Circle((center.x, center.y), radius=radius,
193                                         fc='none')
194         ax.add_patch(c)
195         ax.plot(center.x, center.y, 'r+')
196
197         P.show()

```

Source : circonscrit.py

7.2 Filtres du 2nd ordre

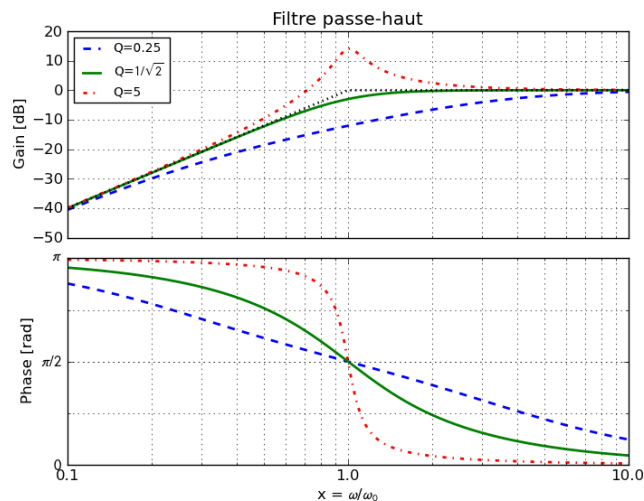


FIGURE 7.1 – Figure : Filtre passe-haut du 2nd ordre.

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  # Time-stamp: <2011-10-20 23:09 yannick@lyopc469>
4
5  import numpy as N
6  import matplotlib.pyplot as P

```

```

7
8 def passeBas(x, Q=1):
9     """Filtre passe-bas en pulsation réduite x=omega/omega0, facteur
10    de qualité Q."""
11
12    return 1/(1-x**2 + x/Q*1j)
13
14 def passeHaut(x, Q=1):
15
16    return -x**2/(1-x**2 + x/Q*1j)
17
18 def passeBande(x, Q=1):
19
20    return 1/(1 + Q*(x-1/x)*1j)
21
22 def coupeBande(x, Q=1):
23
24    return (1-x**2)/(1-x**2 + x/Q*1j)
25
26 def gainNphase(f, dB=True):
27     """Retourne le gain (éventuellement en dB) et la phase [rad] d'un
28    filtre de fonction de transfert complexe f."""
29
30    g = N.abs(f)                # Gain
31    if dB:                      # [dB]
32        g = 20*N.log10(g)
33    p = N.angle(f)              # [rad]
34
35    return g,p
36
37 def asympGain(x, pentes=(0,-40)):
38
39    lx = N.log10(x)
40    return N.where(lx<0, pentes[0]*lx, pentes[1]*lx)
41
42 def asympPhase(x, phases=(0,-N.pi)):
43
44    return N.where(x<1, phases[0], phases[1])
45
46 def diagBode(x, fs, Qs, title='', plim=None, gAsymp=None, pAsymp=None):
47     """Trace le diagramme de Bode -- gain [dB] et phase [rad] -- du
48    filtre de fonction de transfert complexe f en fonction de la
49    pulsation réduite x."""
50
51    fig = P.figure()
52    axg = fig.add_subplot(2,1,1,          # Axe des gains
53                          xscale='log',
54                          ylabel='Gain [dB]')
55    axp = fig.add_subplot(2,1,2,          # Axe des phases
56                          sharex=axg,
57                          xlabel='x = $\omega$/$\omega_0$', xscale='log',
58                          ylabel='Phase [rad]')
59
60    lss=['--','-', '-.', ':']
61    for f,Q,ls in zip(fs,Qs,lss):        # Tracé des différentes courbes
62        g,p = gainNphase(f, dB=True)
63        axg.plot(x, g, lw=2, ls=ls, label="Q="+str(Q)) # Gain
64        axp.plot(x, p, lw=2, ls=ls)           # Phase
65
66    # Asymptotes
67    if gAsymp is not None:                # Gain
68        axg.plot(x, asympGain(x,gAsymp), 'k:', lw=2, label='_')
69    if pAsymp is not None:                # Phase

```

```
70     #axp.plot(x, asympPhase(x,pAsymp), 'k:')
71     pass
72
73     axg.legend(loc='best', prop=dict(size='small'))
74
75     # Labels des phases
76     axp.set_yticks(N.arange(-2,2.1)*N.pi/2)
77     axp.set_yticks(N.arange(-4,4.1)*N.pi/4, minor=True)
78     axp.set_yticklabels([r'$-\pi$', r'$-\pi/2$', r'$0$', r'$\pi/2$', r'$\pi$'])
79     # Domaine des phases
80     if plim is not None:
81         axp.set_ylim(plim)
82
83     # Ajouter les grilles
84     for ax in (axg, axp):
85         ax.grid() # x et y, majors
86         ax.xaxis.grid(True, which='minor') # x, minors
87         ax.yaxis.grid(True, which='minor') # y, minors
88
89     # Ajustements fins
90     gmin, gmax = axg.get_ylim()
91     axg.set_ylim(gmin, max(gmax, 3))
92
93     fig.subplots_adjust(hspace=0.1)
94     axg.xaxis.set_major_formatter(P.matplotlib.ticker.ScalarFormatter())
95     P.setp(axg.get_xticklabels(), visible=False)
96
97     if title:
98         axg.set_title(title)
99
100    return fig
101
102 if __name__ == '__main__':
103
104     P.rc('mathtext', fontset='stixsans')
105
106     x = N.logspace(-1, 1, 1000) # de 0.1 à 10 en 1000 pas
107
108     qs = [0.25, 1/N.sqrt(2), 5] # Valeurs numériques
109     Qs = [0.25, r'$1/\sqrt{2}$', 5] # Chaînes
110     # Calcul des fonctions de transfert complexes
111     pbs = [ passeBas(x, Q=q) for q in qs ]
112     phs = [ passeHaut(x, Q=q) for q in qs ]
113     pcs = [ passeBande(x, Q=q) for q in qs ]
114     cbs = [ coupeBande(x, Q=q) for q in qs ]
115
116     # Création des 4 diagrammes de Bode
117     figPB = diagBode(x, pbs, Qs, title='Filtre passe-bas',
118                     plim=(-N.pi, 0),
119                     gAsymp=(0, -40), pAsymp=(0, -N.pi))
120     figPH = diagBode(x, phs, Qs, title='Filtre passe-haut',
121                     plim=(0, N.pi),
122                     gAsymp=(40, 0), pAsymp=(N.pi, 0))
123     figPC = diagBode(x, pcs, Qs, title='Filtre passe-bande',
124                     plim=(-N.pi/2, N.pi/2),
125                     gAsymp=(20, -20), pAsymp=(N.pi/2, -N.pi/2))
126     figCB = diagBode(x, cbs, Qs, title='Filtre coupe-bande',
127                     plim=(-N.pi/2, N.pi/2),
128                     gAsymp=(0, 0), pAsymp=(0, 0))
129
130     P.show()
```

Source : `filtres2ndOrdre.py`

Exercices

8.1 Manipulation de listes

8.1.1 Crible d'Ératosthène

Implémenter le [crible d'Ératosthène](#) pour afficher les nombres premiers compris entre 1 et un entier fixe, p.ex. :

```
Liste des entiers premiers <= 41
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41]
```

8.1.2 Carré magique

Un carré magique d'ordre n est un tableau carré $n \times n$ dans lequel on écrit une et une seule fois les nombres entiers de 1 à n^2 , de sorte que la somme des n nombres de chaque ligne, colonne ou diagonale principale soit constante. P.ex. le carré magique d'ordre 5, où toutes les sommes sont égales à 65 :

11	18	25	2	9
10	12	19	21	3
4	6	13	20	22
23	5	7	14	16
17	24	1	8	15

Pour les carrés magiques d'ordre impair, on dispose de l'algorithme suivant – (i,j) désignant la case de la ligne i , colonne j du carré ; on se place en outre dans une indexation « naturelle » commençant à 1 :

1. la case $(n, (n+1)/2)$ contient 1 ;
2. si la case (i,j) contient la valeur k , alors on place la valeur $k+1$ dans la case $(i+1, j+1)$ si cette case est vide, ou dans la case $(i-1, j)$ sinon. On respecte la règle selon laquelle un indice supérieur à n est ramené à 1.

Programmer cet algorithme pour pouvoir construire un carré magique d'ordre impair quelconque.

8.2 Flocon de Koch

En utilisant les commandes `left`, `right` et `forward` de la bibliothèque graphique standard `turtle` dans une fonction récursive, générer à l'écran un [flocon de Koch](#) d'ordre arbitraire.

8.3 Jeu de la vie

On se propose de programmer l'automate cellulaire le plus célèbre, le [Jeu de la vie](#).

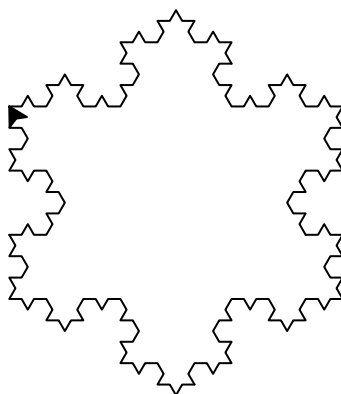


FIGURE 8.1 – **Figure** : Flocon de Koch d'ordre 3.

Pour cela, vous créerez une classe `Life` qui contiendra la grille du jeu ainsi que les méthodes qui permettront son évolution. Vous initialiserez la grille aléatoirement à l'aide de la fonction `random.choice([True, False])`, et vous afficherez l'évolution de l'automate dans la sortie standard du terminal, p.ex. :

. # . # . ## . . .
 . ##
 # . . #
 . . # . #
 ## . . .
 . # . # . ## . # .
 . . ## . ## . . .
 . ## . ## . . .
 ## .
 . #

N.B. : Pour que l’affichage soit agréable à l’oeil, vous marquerez des pauses entre l’affichage de chaque itération grâce à la fonction `time.sleep`.

8.4 Manipulation de tableaux (arrays)

8.4.1 Median Absolute Deviation

En statistique, le *Median Absolute Deviation* (MAD) est un estimateur robuste de la dispersion d'un échantillon

ID: $\text{MAD} = \text{median}(|x - \text{median}(x)|)$

À l'aide de la fonction `numpy.median`, écrire une fonction `mad(x, axis=None)` calculant le MAD d'un tableau le long d'un de ses axes.

8.4.2 Distribution du *pull*

Le *pull* est une quantité statistique permettant d'évaluer la conformité des erreurs par rapport à une distribution de valeurs (typiquement les résidus d'un ajustement). Pour un échantillon $\mathbf{x} = [x_i]$ et les erreurs associées $d\mathbf{x} = [\sigma_i]$, le *pull* est défini par :

- Moyenne optimale (pondérée par la variance) : $E = (\sum_i x_i / \sigma_i^2) / (\sum_i 1 / \sigma_i^2)$
 - Erreur sur la moyenne pondérée : $\sigma_E^2 = 1 / \sum (1 / \sigma_i^2)$
 - Définition du *pull* : $p_i = (x_i - E_i) / (\sigma_{E_i}^2 + \sigma_i^2)^{1/2}$, où E_i et σ_{E_i} sont calculées *sans* le point i .
- Si les erreurs σ_i sont correctes, la distribution du *pull* est centrée sur 0 avec une déviation standard de 1.

Écrire une fonction `pull(x, dx)` calculant le *pull* de tableaux 1D.

8.4.3 Quartet d'Anscombe

Après chargement des données, calculer et afficher les propriétés statistiques des quatre jeux de données du Quartet d'Anscombe :

- Moyenne et variance des x (`N.mean` et `N.var`)
- Moyenne et variance des y
- Corrélation entre les x et les y (`scipy.stats.pearsonr`)
- Équation de la droite de régression linéaire (`scipy.stats.linregress`)

TABLE 8.1 – Quartet d'Anscombe

I		II		III		IV	
x	y	x	y	x	y	x	y
10.0	8.04	10.0	9.14	10.0	7.46	8.0	6.58
8.0	6.95	8.0	8.14	8.0	6.77	8.0	5.76
13.0	7.58	13.0	8.74	13.0	12.74	8.0	7.71
9.0	8.81	9.0	8.77	9.0	7.11	8.0	8.84
11.0	8.33	11.0	9.26	11.0	7.81	8.0	8.47
14.0	9.96	14.0	8.10	14.0	8.84	8.0	7.04
6.0	7.24	6.0	6.13	6.0	6.08	8.0	5.25
4.0	4.26	4.0	3.10	4.0	5.39	19.0	12.50
12.0	10.84	12.0	9.13	12.0	8.15	8.0	5.56
7.0	4.82	7.0	7.26	7.0	6.42	8.0	7.91
5.0	5.68	5.0	4.74	5.0	5.73	8.0	6.89

Pour chacun des jeux de données, tracer y en fonction de x , ainsi que la droite de régression linéaire.

8.4.4 Diagramme de bifurcation : la suite logistique

Écrivez une fonction qui calcule la valeur d'équilibre de la [suite logistique](#) pour un x_0 (nécessairement compris entre 0 et 1) et un paramètre r (parfois noté μ) donné.

Générez l'ensemble de ces points d'équilibre pour des valeurs de r comprises entre 0 et 4 :

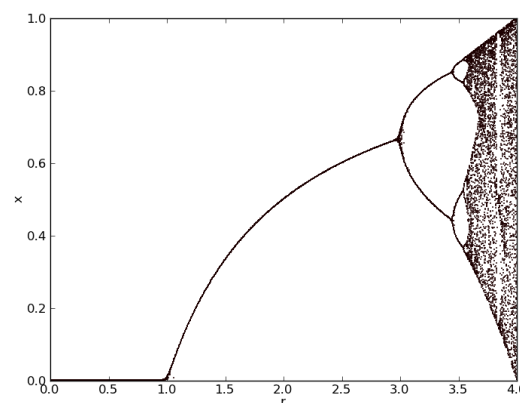


FIGURE 8.2 – **Figure** : Diagramme de bifurcation.

N.B. : Vous utiliserez la bibliothèque `matplotlib` pour tracer vos résultats.

8.4.5 Ensemble de Julia

Représentez l'[ensemble de Julia](#) pour la constante complexe $c = 0.284 + 0.0122j$:

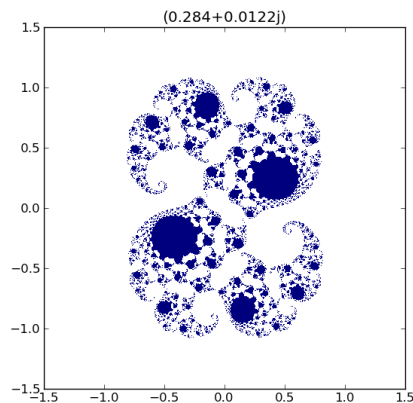


FIGURE 8.3 – **Figure :** Ensemble de Julia pour $c = 0.284 + 0.0122j$.

N.B. : On utilisera la fonction `numpy.meshgrid` pour construire le plan complexe, et l'on affichera le résultat grâce à la fonction `matplotlib.pyplot.imshow`.

8.5 Algorithmes numériques

8.5.1 Intégration numérique de Romberg

Écrire une fonction `integ_romberg(f, a, b, epsilon=1e-6)` permettant de calculer l'intégrale numérique de la fonction f entre les bornes a et b avec une précision $epsilon$ selon la [méthode de Romberg](#).



Tester sur des solutions analytiques et en comparant à `scipy.integrate.romberg`.

8.5.2 Méthode de Runge-Kutta

Développer un algorithme permettant d'intégrer numériquement une équation différentielle du 1er ordre en utilisant la [méthode de Runge-Kutta](#) d'ordre quatre.

Tester sur des solutions analytiques et en comparant à `scipy.integrate.odeint`.

8.6 Exercices supplémentaires

- Exercices de base 
- Entraînez-vous ! 
- [Learn Python The Hard Way](#)
- [Google Code Jam](#)

8.6.1 Protein Data Bank

On cherche à réaliser un script qui analyse un fichier de données de type [Protein Data Bank](#).

La banque de données [Worldwide Protein Data Bank](#) regroupe les structures obtenues par diffraction aux rayons X ou par RMN. Le format est parfaitement défini et conventionnel ([Documentation](#)).

On propose d'assurer une lecture de ce fichier pour calculer notamment :

- le barycentre de la biomolécule
- le nombre d'acides aminés ou nucléobases

- le nombre d'atomes
- la masse moléculaire
- les dimensions maximales de la protéine
- etc.

On propose de considérer par exemple la structure résolue pour la [GFP](#) (*Green Fluorescent Protein*, Prix Nobel 2008) ([Fichier PDB](#))

Micro-projets

9.1 Introduction

Les micro-projets seront évalués sur la base d'un rapport d'une quinzaine de pages et d'une soutenance orale. Il vous est demandé de travailler par binome. En plus des 4 séances prévues pour discuter de l'avancée du projet, un travail personnel est attendu.

Note : L'utilisation de bibliothèques externes (scientifiques, visualisation, etc.) est fortement encouragée.

Une liste de sujets possibles est donnée ci-dessous. Elle est non-exhaustive, c'est-à-dire que vous pouvez *en accord avec votre chargé de TD* proposer une autre étude. Les sujets donnés sont volontairement ouverts : ce sont des *pistes* sur lesquelles vous êtes sensés partir et évoluer.

Les modalités pratiques sont disponibles sur la page [Projets 2013](#).

9.2 Formation de pistes de fourmis sur un pont à 2 branches

Si on propose à une colonie de fourmis de choisir entre 2 branches pour rejoindre une source de nourriture la branche finalement choisie est toujours la plus courte. Le projet consiste à modéliser et caractériser ce comportement.

Indication : on peut étudier ce système avec des EDOs. Cela peut aussi donner lieu à une simulation individu centré et éventuellement une comparaison entre les deux types de modèle.

9.3 Formation d'agrégats

La formation d'agrégats est par essence un sujet interdisciplinaire, où la modélisation joue un rôle certain comme « microscope computationnel ». Pour un projet en ce sens, un soin particulier sera donné à la contextualisation. P. ex., on pourra tester les limites de la règle de Wade pour la structure de clusters métalliques, ou bien dans un contexte plus [biologique](#).

9.4 Modèle d'Ising

Le modèle d'Ising est le modèle le plus simple du magnétisme. Le modèle 1D est exactement soluble par la méthode de la matrice de transfert. La généralisation à 2 dimensions a été faite par Lars Onsager en 1944, mais la solution est assez compliquée. Il n'existe pas de solution analytique en 3D. On va ici considérer un système de spins sur réseau. Chaque spin σ_i peut prendre 2 valeurs (« *up* » et « *down* »). L'hamiltonien du système,

$$H = -J \sum_{i,j} \sigma_i \sigma_j - h \sum_i \sigma_i$$

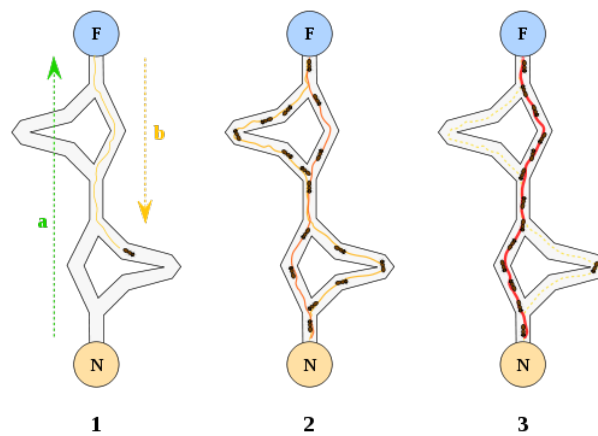


FIGURE 9.1 – **Figure :** 1) la première fourmi trouve la source de nourriture (F), via un chemin quelconque (a), puis revient au nid (N) en laissant derrière elle une piste de phéromone (b). 2) les fourmis empruntent indifféremment les 4 chemins possibles, mais le renforcement de la piste rend plus attractif le chemin le plus court. 3) les fourmis empruntent le chemin le plus court, les portions longues des autres chemins voient la piste de phéromones s'évaporer. Source : [Johann Dréo](#) via Wikimedia Commons.

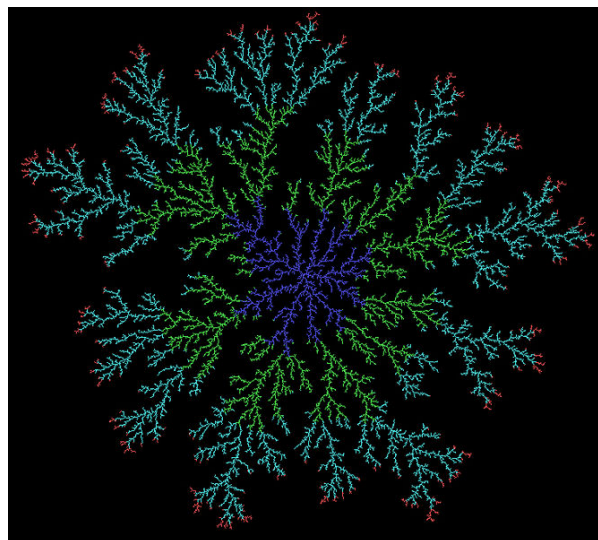


FIGURE 9.2 – **Figure :** Résultat d'une agrégation limitée par la diffusion d'environ 33 000 particules obtenue en permettant à des marcheurs aléatoires d'adhérer à une semence centrale. Les couleurs indiquent le temps d'arrivée des marcheurs. Source : [WingkLEE](#) (Own work) [Public domain], via Wikimedia Commons.

contient deux contributions : l'interaction entre premiers voisins et le couplage à un champ magnétique. On va considérer un réseau carré avec une interaction ferromagnétique ($J > 0$). L'objectif du projet sera d'étudier le diagramme de phase du système en fonction de la température et du champ magnétique par simulation de Monte-Carlo.

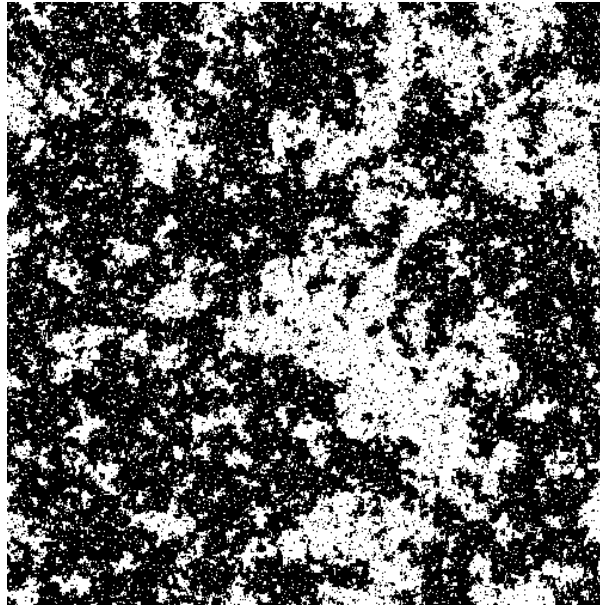


FIGURE 9.3 – **Figure** : Modèle d'Ising au point critique. Source : [Paul Coddington](#).

9.5 Méthode de Hückel

La spectroscopie et la réactivité des électrons π est centrale en chimie. Un outil efficace pour les appréhender est l'approche développée par Hückel. Il vous est demandé ici de mettre en œuvre cette méthode pour l'analyse des orbitales et de l'énergie d'une famille de molécules répondant aux hypothèses sous-jacentes. On discutera notamment du choix de la paramétrisation du système.

9.6 Évacuation d'une salle & déplacement d'une foule dans une rue

Le comportement d'une foule est un problème aux applications multiples : évacuation d'une salle, couloir du métro aux heures de pointes, manifestations... On peut en imaginer des modèles simples. P. ex., on peut décrire chaque individu par sa position, sa vitesse, et comme étant soumis à des « forces » :

- Une force qui spécifie la direction dans laquelle l'individu *veut* se déplacer, $\mathbf{f}_{dir} = (\mathbf{v}_0 - \mathbf{v}(t))/\tau$, où \mathbf{v}_0 est la direction et la vitesse que la personne veut atteindre, \mathbf{v} sa vitesse actuelle, et τ un temps caractéristique d'ajustement.
- Une force qui l'oblige à éviter des obstacles qui peuvent être fixes (un mur, un massif de fleurs, ...), ou qui peuvent être les autres individus eux-mêmes. On pourra essayer $f_{obs}(d) = a \exp(-d/d_0)$, où d est la distance entre le piéton et l'obstacle, d_0 la « portée » de la force, et a son amplitude.

On pourra varier les différents paramètres apparaissant ci-dessus, tout en leur donnant une interprétation physique réelle, et étudier leur influence dans des situations concrètes. P. ex., à quelle vitesse, en fonction de \mathbf{v}_0 et de la densité de piétons, se déplace une foule contrainte à avancer dans un couloir si chaque individu veut maintenir une vitesse \mathbf{v}_0 ? Comment s'organise l'évacuation d'une salle initialement uniformément peuplée, avec une ou plusieurs sorties, et en la présence éventuels d'obstacles ?

Il est également possible d'essayer d'autres expressions pour les forces.

Il existe une littérature conséquente sur le sujet, que l'on pourra explorer si besoin (p. ex : [Décrypter le mouvement des piétons dans une foule](#)).

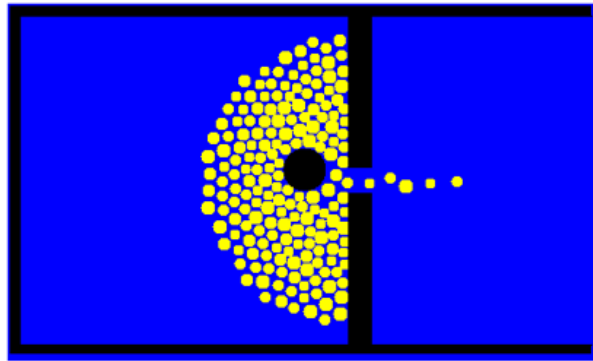


FIGURE 9.4 – **Figure :** Un obstacle aide à l'évacuation. [Source](#).

9.7 Densité d'états d'un nanotube

Les nanotubes de carbone ont été découverts bien avant celle du graphène. Ce sont des matériaux très résistants et durs qui possèdent une conductivité électrique et thermique élevées. Un nanotube de carbone monofeuillet consiste d'une couche de graphène enroulé selon un certain axe. L'axe d'enroulement détermine la chiralité du nanotube et, par la suite, les propriétés électroniques : selon la chiralité, le nanotube peut être soit semi-conducteur, soit métallique. L'objectif du projet sera de calculer la densité d'états de nanotubes de carbone de différentes chiralités et d'établir le lien entre la chiralité et le fait que le nanotube soit semiconducteur ou métallique.

9.8 Solitons

On considère un câble sous tension auquel sont rigidement et régulièrement attachés des pendules. Les pendules sont couplés grâce au câble à travers sa constante de torsion. Dans un tel système on peut observer une large gamme de phénomènes ondulatoires. Le but de cet projet est d'étudier une solution très particulière : le *soliton*.

Imaginons qu'une des extrémités du câble est attachée à une manivelle qui peut tourner librement. Il est alors possible de donner une impulsion au système en faisant un tour rapide ce qui déclenche la propagation d'un soliton. Dans ce projet, on considérera les pendules individuellement. Il n'est pas demandé de passer au modèle continu et de résoudre l'équation obtenue.

Pour chaque pendule n dont la position est décrite par θ_n , l'équation d'évolution s'écrit :

$$\frac{d^2\theta_n}{dt^2} = \alpha \sin \theta_n + \beta(\theta_{n-1} + \theta_{n+1} - 2\theta_n)$$

où α, β sont des paramètres physiques. On résoudra numériquement cette équation pour chaque pendule. En donnant un « tour de manivelle numérique », on essayera d'obtenir la solution soliton. On cherchera en particulier à ajuster la solution par une équation du type $\theta_n = a \tan^{-1}(\exp(b(n - n_0)))$ où a, b, n_0 sont des paramètres à déterminer.

De très nombreuses questions se posent (il ne vous est pas demandé de répondre à chacune d'entre elle) :

- Est-il toujours possible d'obtenir un soliton ?
- Sa vitesse est-elle constante ?
- Le soliton conserve-t-il sa forme ?
- Que se passe-t-il avec des pendules plus lourds ? ou plus rapprochés ? avec un câble plus rigide ? avec un frottement ?
- Comment le soliton se réfléchit-il si l'extrémité du câble est rigidement fixée ? et si elle tourne librement ?
- Dans ce système, le soliton est chiral. En effet, on peut tourner la manivelle à gauche ou à droite. Un anti-soliton a-t-il les mêmes propriétés (taille, vitesse, énergie) qu'un soliton ?

- Si on place une manivelle à chaque extrémité, on peut faire se collisionner des solitons. Cette étude est très intéressante et pleine de surprises. Que se passe-t-il lors de la collision de deux solitons ? Entre un soliton et un anti-soliton ?



FIGURE 9.5 – **Figure** : Un mascaret, une vague soliton, dans un estuaire de Grande Bretagne. Source : [Arnold Price](#) [CC-BY-SA-2.0], via Wikimedia Commons.

orphan

9.9 Auto-organisation d'un banc de poissons

Auteur de la section : Hanna Julienne <hanna.julienne@gmail.com>

La coordination d'un **banc de poissons** ou d'un **vol d'oiseaux** est tout à fait frappante : les milliers d'individus qui composent ces structures se meuvent comme un seul. On observe aussi, dans les bancs de poisson, d'impressionnants comportements d'évitement des prédateurs (*flash expansion*, *fountain effect*).

Pourtant ces mouvements harmonieusement coordonnés ne peuvent pas s'expliquer par l'existence d'un poisson leader. Comment pourrait-il être visible par tous ou diriger les *flash expansion* qui ont lieu à un endroit précis du banc de poisson ? De la même manière on ne voit pas quelle contrainte extérieure pourrait expliquer le phénomène.

Une hypothèse plus vraisemblable pour rendre compte de ces phénomènes est que la cohérence de l'ensemble est due à la somme de comportements individuels. Chaque individu adapte son comportement par rapport à son environnement proche. C'est ce qu'on appelle *auto-organisation*. En effet, on a établi expérimentalement que les poissons se positionnent par rapport à leurs k plus proches voisins de la manière suivante :

- ils s'éloignent de leurs voisins très proches (zone de répulsion en rouge sur la figure ci-dessous)
- ils s'alignent avec des voisins qui sont à distance modérée (zone jaune)
- ils s'approchent de leur voisins s'ils sont à la fois suffisamment proches et distants (zone verte)

Dans notre modèle, nous allons prendre en compte l'influence des k plus proches voisins. On calculera la contribution de chaque voisin selon la zone dans laquelle il se situe. Le déplacement du poisson sera la moyenne de ces contributions. Il est à noter qu'un voisin en dehors des trois zones d'influence n'a pas d'effet.

L'environnement proche d'un poisson est modélisé par des sphères imbriquées qui présentent une zone aveugle (voir figure).

Par ailleurs, si un individu n'a pas de voisins dans son environnement proche il adopte un comportement de recherche. Il explore aléatoirement les alentours jusqu'à ce qu'il repère le banc de poissons et finalement s'en rapproche.

Ce projet vise à :

- Coder le comportement des poissons et à les faire évoluer dans un environnement **2D**.
- On essaiera d'obtenir un comportement collectif cohérent (similaire à un banc de poisson) et d'établir les conditions nécessaires à ce comportement.
- On étudiera notamment l'influence du nombre d'individus pris en compte. Est-ce que le positionnement par rapport au plus proche voisin ($k = 1$) est suffisant ?

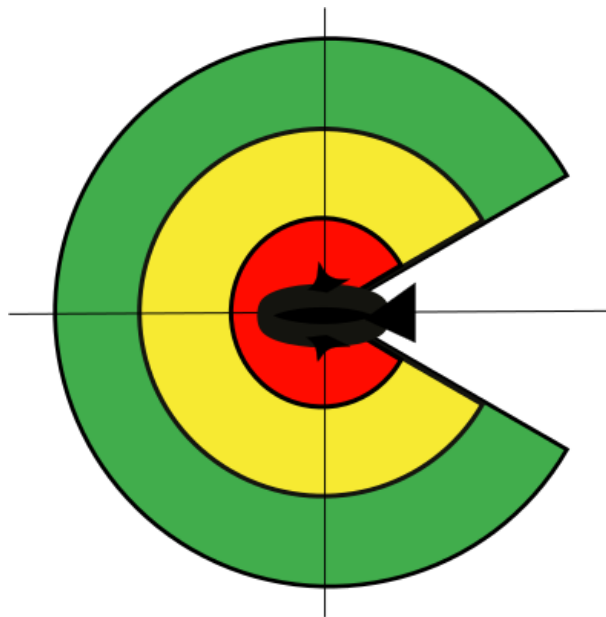


FIGURE 9.6 – **Figure** : Environnement proche du poisson : zones dans lesquelles le positionnement d'un voisin provoque une réponse de la part de l'individu au centre

- On pourra se servir de la visualisation pour rendre compte de la cohérence du comportement et éventuellement inventer des mesures pour rendre compte de manière quantifier de cette cohérence.

Suggestions :

Attention : Les suggestions sont difficiles à réaliser. On cherchera à obtenir des résultats sur le projet de base avant de s'y risquer.

- Passer le projet en 3D
- modéliser la réponse des individus face à un prédateur

Liens :

- [Craig Reynolds Boids](#)
- [Décrypter les interactions entre poissons au sein d'un banc](#)
- [Nuées d'oiseaux et bancs de poissons](#)

orphan

9.10 Diagramme de phase du potentiel de Lennard-Jones

Auteur de la section : Mathieu Leocmach <mathieu.leocmach@ens-lyon.fr>

Le potentiel de Lennard-Jones est souvent utilisé pour décrire les interactions entre deux atomes au sein d'un système monoatomique de type gaz rare. Son expression en fonction de la distance r entre les deux noyaux atomiques est :

$$E_p(r) = 4E_0 \left[\left(\frac{r_0}{r} \right)^{12} - \left(\frac{r_0}{r} \right)^6 \right]$$

avec r_0 la distance pour laquelle $E_p(r_0) = 0$.

On programmera un simulateur de dynamique moléculaire pour N particules identiques dans un cube périodique de taille fixe L et à une température T . On prendra soin d'adimensionner toutes les grandeurs et d'imposer des conditions aux limites périodiques. On se renseignera sur les façons possibles de déterminer les conditions initiales et d'imposer la température.

Les positions et vitesses des particules seront exportées de façon régulières pour visualisation (par exemple dans [Paraview](#)).

- On pourra observer les collisions de 2 ou 3 particules à différentes températures avant de passer à des N plus grands (100 particules ?).
- On fera varier $V = L^3$ et T pour déterminer les frontières des différentes phases.
- On pourra aussi essayer d’aller vers de plus grands N pour tester l’influence de la taille finie de l’échantillon. Des optimisations seront alors sûrement nécessaires pour accélérer le programme.
- On pourra aussi tester d’autres types de potentiels comme celui de Weeks-Chandler-Anderson et discuter des différences observées.

orphan

9.11 Suivi de particule(s)

Auteur de la section : Mathieu Leocmach <mathieu.leocmach@ens-lyon.fr>

Dans de nombreux domaines de recherche expérimentale, on a besoin de localiser des particules dans une image ou de reconstituer leurs trajectoires à partir d’une vidéo. Il peut s’agir de virus envahissant une cellule, de traceurs dans un écoulement, d’objets célestes fonçant vers la terre pour la détruire, etc.

Dans ce projet, on essaiera d’abord de localiser une particule unique dans une image à 2 dimensions (niveaux de gris) en utilisant l’algorithme de Crocker & Grier décrit [ici](#). On utilisera sans retenue les fonctions de la bibliothèque `scipy.ndimage`.

On essaiera d’obtenir une localisation plus fine que la taille du pixel. On essaiera ensuite de détecter plusieurs particules dans une image.

Afin de pouvoir traiter efficacement une séquence d’images de même taille, on privilégiera une implémentation orientée objet. L’objet de la classe `Finder` sera construit une seule fois en début de séquence et il contiendra les images intermédiaires nécessaire au traitement. On nourrira ensuite cet objet avec chaque image de la séquence pour obtenir les coordonnées des particules.

Enfin, on pourra essayer de relier les coordonnées dans des images successives pour constituer des trajectoires.

On contactera le créateur du sujet pour obtenir des séquences d’images expérimentales de particules Browniennes.

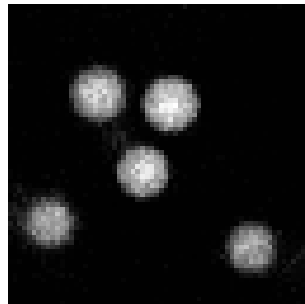


FIGURE 9.7 – **Figure** : Exemple d’image test où on voudra localiser les particules.

9.12 Autres possibilités

Attention : ces sujets n’ont pas été validés. Discutez-en avec votre enseignant avant de vous lancer.

- [Reaction-Diffusion by the Gray-Scott Model](#)
- Équation de Cahn–Hilliard (voir l’exemple NIST sous FiPy : [A Finite Volume PDE Solver Using Python](#))
- [Computational Methods for Nonlinear Systems](#)

9.12.1 États de diffusion pour l'équation de Schrödinger 1D stationnaire

On s'intéresse à la diffusion d'une particule de masse m à travers un potentiel carré défini par $V(x) = V_0$ pour $0 \leq x \leq a$, et 0 sinon.

Les solutions de cette équation en dehors de la région où règne le potentiel sont connues. Les paramètres d'intégration de ces fonctions d'onde peuvent se déterminer par les relations de continuité aux frontières avec la région où règne le potentiel. En résolvant l'équation différentielle dans la région du potentiel pour x allant de a à 0 on peut obtenir une autre valeur pour ces paramètres d'intégration. Il faut ensuite appliquer un algorithme de minimisation pour déterminer les constantes d'intégration.

Les objectifs de ce projet sont :

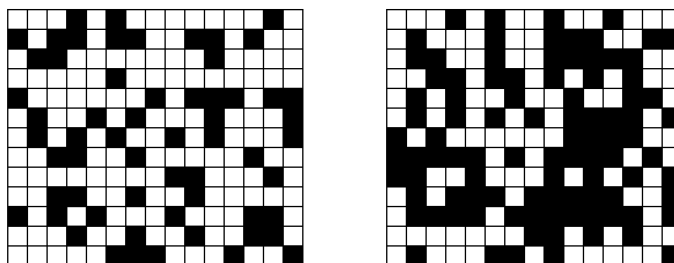
- Écrire un programme qui résolve l'équation de Schrödinger.
- En déduire les coefficients de transmission et de réflexion.

Références :

- *A numerical method for quantum tunnelling*, Pang T., Computers in Physics, 9, p 602-605.
- Équation de Schrödinger 1D

9.12.2 Percolation

Ce sujet propose d'étudier le phénomène de percolation. La percolation est un processus physique qui décrit pour un système, une transition d'un état vers un autre. Le système que nous étudierons est composé ici d'une grille carrée dont les cases sont soit vides, soit pleines. Initialement, la matrice est vide et l'on tire aléatoirement une case que l'on remplit. On définit la concentration comme le rapport du nombre de cases noires sur le nombre total de cases. À partir d'une certaine concentration critique un chemin continu de cases noires s'établit entre deux bords opposés du système (haut et bas, ou gauche et droite) et on dit alors que le système percole. Le but du sujet est d'étudier la transition d'un système qui ne percole pas (à gauche sur la figure) vers un système qui percole (à droite). Pour ce faire, on établira un algorithme qui pour une configuration donnée détermine si le réseau de cases noires percole ou non. On étudiera également la taille et le nombre des amas de cases noires en fonction de la concentration. On étudiera aussi les effets de la taille du système.



Cette étude repose sur un tirage pseudo aléatoire et pas conséquent nécessite un traitement statistique. On ne pourra pas se contenter d'étudier un cas particulier mais on prendra soin au contraire d'effectuer des moyennes sur un très grand nombre de tirages (plusieurs centaines).

Sources :

- Percolation theory
- Concepts fondamentaux de la percolation
- Percolation exercises : 2006, 2012

9.12.3 Modèle de Potts 3D

Modèle de Potts en 3D dans un univers carré à condition périodique. Le but est la mise en évidence de la transition de phase pour plusieurs jeux de paramètres avec 3 types de spins différents.

1. Reproduire des résultats connus du modèle d'Ising en 2D pour valider le code.
2. Passer à un algorithme en *cluster* pour évaluer la différence avec un algorithme classique.
3. Passer en 3D

4. Changer le nombre de type de spins (de 2 à 3).

Jeux de paramètres à tester :

- Ising en 2D (2 types de spins, algorithme de Glauber) : Transition de phase attendue à $T \sim 227K$ pour un couplage $J=100$ et un champ externe nul
- Toujours Ising, mais avec l'algorithme de Wolff
- Ising en 3D avec Wolff
- Potts (changer $q=2$ par $q=3$) en 3D avec Wolff

Références : [Computational Studies of Pure and Dilute Spin Models](#)

Projets 2013

Warning : Il s'agit ici d'informations **préliminaires**, qui restent à confirmer.

- Avant le 16 novembre : choix d'un sujet de projet et envoi d'un email à votre chargé de TD
- Novembre/décembre : 4 séances de discussion
- 13 janvier 2014, 17 :00 : envoi des codes (fichiers sources python) et du rapport (fichier PDF) au chargé de TD sous forme électronique. Une version imprimée (agrafée ou reliée) du rapport sera également déposée au secrétariat.
- Deuxième quinzaine de janvier : soutenance (15 mn + questions)

Le rapport, rédigé en **LaTeX** (imposé pour validation ultérieure **C2i2e**) doit faire entre 10 et 15 pages, éventuellement avec quelques annexes (p.ex. version commentée du code). Il doit logiquement inclure une brève introduction à la problématique physique, une présentation du problème numérique et des algorithmes retenus pour sa résolution, une analyse détaillée des expérimentations numériques réalisées et des différents résultats obtenus (p.ex. vitesse de convergence, précision numérique, influence des différents paramètres de la simulation, comparaison avec la théorie, etc.), une conclusion et des perspectives de développement. Une section bibliographique est conseillée.

Binôme	Sujet
<i>Groupe du mercredi matin</i>	
Guillaume Blot, Gabriel Rocheman	Ising
Ménil Reboud, Sami Jouabert	Percolation
Nicolas Auvray, Samuel Boury	Foule
Maxime Lombart, Sophie Michel	Poissons
Louis Garbe	Poissons
Juliette Monsel, Barbara Pascal	Fourmis
Marion Teyssedre, Perceval Desforges	Fourmis
<i>Groupe du jeudi matin</i>	
Thibaut Coudarchet, Lilian Chabrol	Potentiel de Lennard-Jones
Paul Haddad, David Germain	Les solitons
Florine Dubourg, Raphaëlle Taub	Les fourmis
Mohamed Bekhtaoui, Pierre Martin-Dussaud	Modèle d'Ising
Éric Beaucé, Céline Boucly	Les solitons

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  # Time-stamp: <2013-11-05 17:19:01 ycopin>
4
5  """
6  Crible d'Ératosthène.
7
8  Source: http://fr.wikibooks.org/wiki/Exemples\_de\_scripts\_Python#Impl.C3.A9mentation\_du\_crible\_d.2
9  """
10
11 # Gestion simplifiée d'un argument sur la ligne de commande (TD4)
12 # start-sys
13 import sys
14
```

```
15 if sys.argv[1:]: # Présence d'au moins un argument sur la ligne de commande
16     try:
17         n = int(sys.argv[1]) # Essayer de lire le 1er argument comme un entier
18     except ValueError:
19         raise ValueError("L'argument '%s' n'est pas un entier" % sys.argv[1])
20 else:
21     n = 101 # Pas d'argument sur la ligne de commande
22     # Valeur par défaut
23
24 # Liste des entiers *potentiellement* premiers. Les nb non-premiers
25 # seront étiquetés par 0.
26 l = range(n+1) # <0,...,n>
27 l[1] = 0 # 1 n'est pas premier
28
29 i = 2 # Entier à tester
30 while i**2 <= n: # Inutile de tester jusqu'à n
31     if l[i]: # Si i n'est pas étiqueté...
32         l[2*i::i] = [0]*len(l[2*i::i]) # ...étiqueter tous les multiples de i
33     i += 1 # Passer à l'entier à tester suivant
34
35 # Afficher la liste des entiers premiers (c-à-d non-étiquetés)
36 print "Liste des entiers premiers <=", n
37 print [ i for i in l if i ]
```

\$ python crible.py

Liste des entiers premiers <= 101

[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, ...]

Source : crible.py

Carré magique

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  # Time-stamp: <2013-11-05 16:56:47 ycopin>
4
5  """
6  Création et affichage d'un carré magique d'ordre impair.
7  """
8
9  __author__ = "Yannick Copin <y.copin@ipnl.in2p3.fr>"
10
11  n = 5                                # Ordre du carré magique
12
13  # On vérifie que l'ordre est bien impair
14  assert n%2 == 1, "L'ordre %d n'est pas impair" % n
15
16  # Le carré sera stocké dans une liste de n listes de n entiers
17  # Initialisation du carré: liste de n listes de n zéros.
18  array = [ [ 0 for j in range(n) ] for i in range(n) ]
19
20  # Initialisation de l'algorithme
21  i, j = n, (n+1)/2                    # Indices de l'algo (1-indexation)
22  array[i-1][j-1] = 1                  # Attention: python utilise une 0-indexation
23  # Boucle sur valeurs restant à inclure dans le carré (de 2 à n**2)
24  for k in range(2, n**2+1):
25      # Test de la case i+1, j+1 (modulo n)
26      i2 = (i+1) % n
27      j2 = (j+1) % n
28      if not array[i2-1][j2-1]: # La case est vide: l'utiliser
29          i, j = i2, j2
30      else:
31          i = (i-1) % n            # La case est déjà remplie: utiliser la case i-1, j
32          array[i-1][j-1] = k      # Remplissage de la case
33
34  # Affichage, avec vérification des sommes par ligne et par colonne
35  print "Carré magique d'ordre %d:" % n
36  for row in array:
37      print ' '.join( '%2d' % k for k in row ), '=', sum(row)
38  print ' '.join( '==' for k in row )
39  print ' '.join( str(sum( array[i][j] for i in range(n) )) for j in range(n) )

```

\$ python carre.py

```

Carré magique d'ordre 5:
11 18 25  2  9 = 65
10 12 19 21  3 = 65
 4  6 13 20 22 = 65
23  5  7 14 16 = 65

```

```
17  24   1   8  15 = 65
==  ==  ==  ==  ==
65  65  65  65  65
```

Source : `carre.py`

Flocon de Koch

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  from __future__ import division          # Pas de division euclidienne par défaut
5
6  __version__ = "Time-stamp: <2012-08-31 19:18:23 ycopin>"
7  __author__ = "Yannick Copin <y.copin@ipnl.in2p3.fr>"
8
9  """
10 Dans le même genre:
11
12 - courbe de Peano (http://fr.wikipedia.org/wiki/Courbe\_de\_Peano)
13 - courbe de Hilbert (http://fr.wikipedia.org/wiki/Courbe\_de\_Hilbert)
14 - île de Gosper (http://fr.wikipedia.org/wiki/Île\_de\_Gosper)
15
16 Voir également:
17
18 - L-système: http://fr.wikipedia.org/wiki/L-système
19
20 Autres exemples à http://natesoares.com/tutorials/python-fractals/
21 """
22
23 import turtle as T
24
25 def koch(niveau=3, iter=0, taille=100, delta=0):
26     """Tracé du flocon de Koch de niveau 'niveau', de taille 'taille'
27     (px).
28
29     Cette fonction récursive permet d'initialiser le flocon (iter=0,
30     par défaut), de tracer les branches fractales (0<iter<=niveau) ou
31     bien juste de tracer un segment (iter>niveau).
32     """
33
34     if iter==0:                                # Dessine le triangle de niveau 0
35         T.title("Flocon de Koch - niveau %d" % niveau)
36         koch(iter=1, niveau=niveau, taille=taille, delta=delta)
37         T.right(120)
38         koch(iter=1, niveau=niveau, taille=taille, delta=delta)
39         T.right(120)
40         koch(iter=1, niveau=niveau, taille=taille, delta=delta)
41     elif iter<=niveau:                        # Trace une section _/\_ du flocon
42         koch(iter=iter+1, niveau=niveau, taille=taille, delta=delta)
43         T.left(60 + delta)
44         koch(iter=iter+1, niveau=niveau, taille=taille, delta=delta)
45         T.right(120 + 2*delta)
46         koch(iter=iter+1, niveau=niveau, taille=taille, delta=delta)
47         T.left(60 + delta)

```

```
48     koch(iter=iter+1, niveau=niveau, taille=taille, delta=delta)
49     else:                                     # Trace le segment de dernier niveau
50         T.forward(taille/3**(niveau+1))
51
52 if __name__=='__main__':
53
54     # start-optparse
55     from optparse import OptionParser
56
57     desc = u"Tracé (via 'turtle') d'un flocon de Koch d'ordre arbitraire."
58
59     # Définition des options
60     parser = OptionParser(usage="%prog [options] ordre",
61                           version=__version__, description=desc)
62     parser.add_option("-t", "--taille", type=int,
63                      help="Taille [%default px]",
64                      default=500)
65     parser.add_option("-d", "--delta", type=float,
66                      help="Delta [%default deg]",
67                      default=0.)
68     parser.add_option("-f", "--figure",
69                      help="Figure de sortie (format EPS)")
70     parser.add_option("-T", "--turbo",
71                      action="store_true", default=False,
72                      help="Mode Turbo")
73
74     # Déchiffrage des options et arguments
75     opts,args = parser.parse_args()
76
77     # Quelques tests sur les args et options
78     try:
79         niveau = int(args[0])
80         assert niveau >= 0
81     except (IndexError,                               # Pas d'argument
82          ValueError,                                 # Argument non-entier
83          AssertionError):                             # Argument entier < 0
84         parser.error("Niveau d'entrée %s invalide" % args)
85
86     if opts.taille < 0:
87         parser.error("La taille de la figure doit être positive")
88     # end-optparse
89
90     if opts.turbo:
91         T.hideturtle()
92         T.speed(0)
93
94     # Tracé du flocon de Koch de niveau 3
95     koch(niveau=niveau, taille=opts.taille, delta=opts.delta)
96     if opts.figure:
97         # Sauvegarde de l'image
98         print "Sauvegarde de la figure dans '%s'" % opts.figure
99         T.getscreen().getcanvas().postscript(file=opts.figure)
100
101     T.exitonclick()
```

Source : koch.py

Jeu de la vie

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  # Time-stamp: <2012-09-05 09:31 ycopin@lyopc469>
4
5  import random
6
7  class Life(object):
8
9      cells = {False: ".", True: "#"} # Dead and living cell representations
10
11     def __init__(self, h, w, periodic=False):
12         """
13         Create a 2D-list (the game grid *G*) with the wanted size (*h*
14         rows, *w* columns) and initialize it with random booleans
15         (dead/alive). The world is periodic if *periodic* is True.
16         """
17
18         self.h = int(h)
19         self.w = int(w)
20         assert self.h > 0 and self.w > 0
21         # Random initialization of a h×w world
22         self.world = [ [ random.choice([True,False])
23                         for j in range(self.w) ]
24                       for i in range(self.h) ] # h rows of w elements
25         self.periodic = periodic
26
27     def get(self, i, j):
28         """
29         This method returns the state of cell (*i*,*j*) safely, even
30         if the (*i*,*j*) is outside the grid.
31         """
32
33         if self.periodic:
34             return self.world[i%self.h][j%self.w] # Periodic conditions
35         else:
36             if 0<=i<self.h and 0<=j<self.w:      # Inside grid
37                 return self.world[i][j]
38             else:                                  # Outside grid
39                 return False                       # There's nobody out there...
40
41     def __str__(self):
42         """
43         Convert the grid to a visually handy string.
44         """
45
46         return '\n'.join([ ''.join([ self.cells[val] for val in row ])
47                             for row in self.world ])

```

```
48
49 def evolve_cell(self,i,j):
50     """Tells if cell (*i*,*j*) will survive during game iteration,
51     depending on the number of living neighboring cells."""
52
53     alive = self.get(i,j)          # Current cell status
54     # Count living cells *around* current one (excluding current one)
55     count = sum( self.get(i+ii,j+jj)
56                 for ii in [-1,0,1]
57                 for jj in [-1,0,1]
58                 if (ii,jj) != (0,0) )
59
60     if count==3:
61         # A cell w/ 3 neighbors will either stay alive or resuscitate
62         future = True
63     elif count<2 or count>3:
64         # A cell w/ too few or too many neighbors will die
65         future = False
66     else:
67         # A cell w/ 2 or 3 neighbors will stay as it is (dead or alive)
68         future = alive          # Current status
69
70     return future
71
72 def evolve(self):
73     """
74     Evolve the game grid by one step.
75     """
76
77     # Update the grid
78     self.world = [ [ self.evolve_cell(i,j)
79                     for j in range(self.w) ]
80                   for i in range(self.h) ]
81
82 if __name__=="__main__":
83
84     import time
85
86     h,w = (20,60)          # (nrows,ncolumns)
87     n = 100                # Nb of iterations
88
89     life = Life(h, w, periodic=True) # Instantiation (including initialization)
90
91     for i in range(n):      # Iterations
92         print life          # Print current world
93         print "\n"
94         time.sleep(0.1)      # Pause a bit
95         life.evolve()        # Evolve world
```

Source : life.py

Median Absolute Deviation

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  import numpy as N
5
6  def mad(a, axis=None):
7      """Compute *Median Absolute Deviation* of an array along given axis."""
8
9      med = N.median(a, axis=axis)          # Median along given axis
10     if axis is None:
11         umed = med                        # med is a scalar
12     else:
13         umed = N.expand_dims(med, axis)    # Bring back the vanished axis
14     mad = N.median(N.absolute(a - umed), axis=axis) # MAD along given axis
15
16     return mad
17
18 if __name__ == '__main__':
19
20     x = N.arange(5*7, dtype='f').reshape(5,7)
21
22     print "x =\n",x
23     print "MAD(x, axis=None) =", mad(x)
24     print "MAD(x, axis=0)      =", mad(x, axis=0)
25     print "MAD(x, axis=1)      =", mad(x, axis=1)
```

Source : mad.py

Distribution du *pull*

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  import numpy as N
5
6  def pull(x, dx):
7      """Compute the pull from x, dx.
8
9      * Input data: x = [x_i], error dx = [s_i]
10     * Optimal (variance-weighted) mean: E = sum(x_i/s_i^2)/sum(1/s_i^2)
11     * Variance on weighted mean: var(E) = 1/sum(1/s_i^2)
12     * Pull: p_i = (x_i - E_i)/sqrt(var(E_i) + s_i^2)
13       where E_i and var(E_i) are computed without point i.
14
15     If errors s_i are correct, the pull distribution is centered on 0
16     with standard deviation of 1."""
17
18     assert x.ndim==dx.ndim==1, "pull works on 1D-arrays only."
19     assert len(x)==len(dx), "dx should be the same length as x."
20
21     n = len(x)
22
23     i = N.resize(N.arange(n), n*n) # 0,...,n-1,0,...n-1,..., n times (n^2,)
24     i[:,n+1] = -1                  # Mark successively 0,1,2,...,n-1
25     j = i[i>=0].reshape((n,n-1))  # Remove marked indices & reshape (n,n-1)
26
27     v = dx**2                      # Variance
28     w = 1/v                        # Variance (optimal) weighting
29
30     Ei = N.average(x[j], weights=w[j], axis=1) # Weighted mean (n,)
31     vEi = 1/N.sum(w[j], axis=1)               # Variance of mean (n,)
32
33     p = (x - Ei)/N.sqrt(vEi + v)              # Pull (n,)
34
35     return p
36
37 if __name__=='__main__':
38
39     n = 100
40     mu = 1.
41     sig = 2.
42
43     # Normally distributed random sample of size n, with mean=mu and std=sig
44     x = N.random.normal(loc=mu, scale=sig, size=n)
45     dx = N.ones_like(x)*sig                # Formal (true) errors
46
47     p = pull(x, dx)                        # Pull computation

```

48

49 `print "Pull: mean=%.2f, std=%.2f" % (p.mean(), p.std(ddof=1))`

Source : pull.py

Quartet d'Anscombe

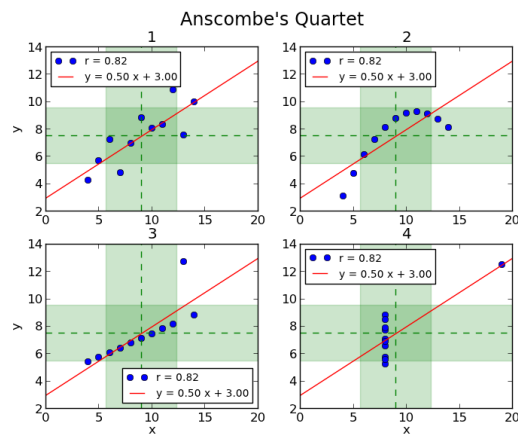
```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  import numpy as N
5  import scipy.stats as SS
6  import matplotlib.pyplot as P
7
8  def printStats(x, y):
9
10     assert N.shape(x)==N.shape(y), "Incompatible input arrays"
11
12     print "x: mean=%.2f, variance=%.2f" % (N.mean(x), N.var(x))
13     print "y: mean=%.2f, variance=%.2f" % (N.mean(y), N.var(y))
14     print "y vs. x: corrcoeff=%.2f" % (SS.pearsonr(x, y)[0])
15     # slope, intercept, r_value, p_value, std_err
16     a,b,r,p,s = SS.linregress(x,y)
17     print "y vs. x: y = %.2f x + %.2f" % (a, b)
18
19  def plotStats(ax, x, y, title=''):
20
21     assert N.shape(x)==N.shape(y), "Incompatible input arrays"
22
23     # slope, intercept, r_value, p_value, std_err
24     a,b,r,p,s = SS.linregress(x, y)
25
26     # Data + corrcoeff
27     ax.plot(x, y, 'bo', label="r = %.2f" % r)
28
29     # Add mean line ± stddev
30     m = N.mean(x)
31     s = N.std(x, ddof=1)
32     ax.axvline(m, color='g', ls='--', label='_') # Mean
33     ax.axvspan(m-s, m+s, color='g', alpha=0.2, label='_') # Std-dev
34
35     m = N.mean(y)
36     s = N.std(y, ddof=1)
37     ax.axhline(m, color='g', ls='--', label='_') # Mean
38     ax.axhspan(m-s, m+s, color='g', alpha=0.2, label='_') # Std-dev
39
40     # Linear regression
41     xx = N.array([0,20])
42     yy = a*xx + b
43     ax.plot(xx, yy, 'r-', label="y = %.2f x + %.2f" % (a, b))
44
45     # Title and labels
46     ax.set_title(title)
47     if ax.is_last_row():

```

```

48     ax.set_xlabel("x")
49     if ax.is_first_col():
50         ax.set_ylabel("y")
51     leg = ax.legend(loc='best', fontsize='small')
52
53 if __name__=='__main__':
54
55     quartet = N.genfromtxt("anscombe.dat") # Read Anscombe's Quartet
56
57     fig = P.figure()
58
59     for i in range(4): # Loop over quartet sets x,y
60         ax = fig.add_subplot(2,2,i+1)
61         print "Dataset %d " % (i+1) + "="*20
62         x,y = quartet[:,2*i:2*i+2].T
63         printStats(x, y) # Print main statistics
64         plotStats(ax, x, y, title=str(i+1)) # Plots
65
66     fig.suptitle("Anscombe's Quartet", fontsize='x-large')
67
68     P.show()
    
```



Source : anscombe.py

Suite logistique

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  # Time-stamp: <2012-09-05 02:37 ycopin@lyopc469>
4
5  import numpy as np
6  import random
7  import matplotlib.pyplot as plt
8
9  def iteration(r, niter=100):
10
11     x = random.uniform(0,1)
12     i = 0
13     while i < niter and x < 1:
14         x = r*x*(1-x)
15         i += 1
16
17     return x if x<1 else -1
18
19  def generate_diagram(r, ntrials=50):
20     """
21     Cette fonction retourne (jusqu'à) *ntrials* valeurs d'équilibre
22     pour les *r* d'entrée. Elle renvoie un tuple:
23
24     + le premier élément est la liste des valeurs prises par le paramètre *r*
25     + le second est la liste des points d'équilibre correspondants
26     """
27
28     r_v = []
29     x_v = []
30     for rr in r:
31         j = 0
32         while j<ntrials:
33             xx = iteration(rr)
34             if xx > 0: # A convergé: il s'agit d'une valeur d'équilibre
35                 r_v.append(rr)
36                 x_v.append(xx)
37             j += 1 # Nouvel essai
38
39     return r_v,x_v
40
41  r = np.linspace(0,4,1000)
42  x,y = generate_diagram(r)
43
44  plt.plot(x, y, 'r,')
45  plt.xlabel('r')
46  plt.ylabel('x')
47  plt.show()

```

Source : `logistique.py`

Ensemble de Julia

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  # Time-stamp: <2012-09-05 02:10 ycopin@lyopc469>
4
5  """Visualisation de l'ensemble de julia
6  <http://fr.wikipedia.org/wiki/Ensemble\_de\_Julia>`_.
7
8  Exercice: proposer des solutions pour accélérer le calcul.
9  """
10
11 import numpy as np
12 import matplotlib.pyplot as plt
13
14 c = complex(0.284,0.0122)          # Constante
15
16 xlim = 1.5                         # [-xlim,xlim] × i[-xlim,xlim]
17 nx = 1000                         # Nb de pixels
18 niter = 100                       # Nb d'itérations
19
20 x = np.linspace(-xlim, xlim, nx)   # nx valeurs de -xlim à +xlim
21 xx,yy = np.meshgrid(x, x)         # Tableaux 2D
22 z = xx + 1j*yy                    # Portion du plan complexe
23 for i in range(niter):            # Itération:  $z(n+1) = z(n)**2 + c$ 
24     z = z**2 + c
25
26 # Visualisation
27 plt.imshow(np.abs(z), extent=[-xlim,xlim,-xlim,xlim], aspect='equal')
28 plt.title(c)
29 plt.show()

```

Source : julia.py