Python en calcul scientifique

Romaric DAVID

Ecole d'Automne Informatique Scientifique
1er Octobre 2008

Plan

Quelques caractéristiques de Python I

- Langage interprété $\Rightarrow$ portabilité du code : il suffit d'avoir un interpréteur (Python dans notre cas) sur la machine
- réduction de la durée du cycle de développement par suppression de la phase de compilation.

Python est conçu pour inciter le programmeur à écrire des programmes clairs, documentés, et concis.
Mais encore...

- Programmes clairs : l'indentation est obligatoire en Python.
- Fonctions documentées :
  docstring, une chaîne de caractères décrivant le fonctionnement de la fonction ;

Quelques caractéristiques de Python II

- Concis : le nombre important de structures de données déjà existantes permet de se concentrer sur l'algorithme du programme à résoudre.
- Langage qui a plu à la communauté numérique

Plan

Tableaux en python I

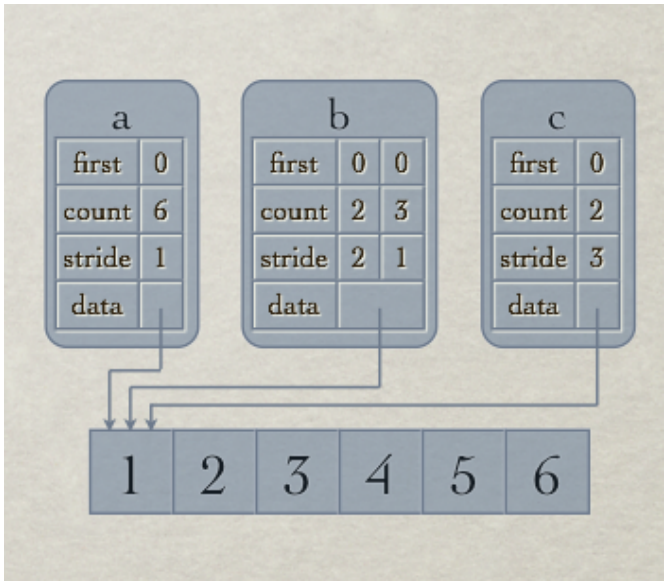
Python bien diffusé dans la communauté "numérique", en raison de ces modules permettant l'utilisation de grands talbeaux et d'opérations math courantes et facilement accessibles

- Module `array` du langage
- Tableaux Contigus
- Typecode définissant le type de données
- Pas d'opérations math ! ! !

- *Extension* numpy
- Tableaux contigus
- Opérations math (FFT, Algebre lineaire, générateurs aléatoires)
- Utilisation routines optimisées du système si présentes

Tableaux en python II

Structure en mémoire d'un tableau numpy (emprunté à Konrad Hinsen, CNRS, Orléans) :



- Exemple de code

Tableaux en python III

```
>>> import numpy
>>> a=numpy.zeros((3,4),dtype='f')
>>> a
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]], dtype=float32
```

Fichier `Exemples/Numpy/first_numpy_array`

More on numpy I

Comment est construit numpy ?

- Noyaux de calculs C/Fortran
- Interfacés avec python
- Python utilisé comme driver (+ standardistion des appels aux fonctions)

Une application de numpy : f2py

- f2py : automatisation interfaçage Python / Fortran
- Définition (assistée) d'une interface (entrées, sorties)
- Compilation automatique d'un *wrapper*, depuis un fichier source fortran ou depuis une bibliothèque
- Utilisation de ce wrapper comme d'un nouveau module Python

Fonctionnement de f2py I

Intéret du fichier d'interfaçage :

- Squelette construit par f2py
- Non intrusion dans le code . . .)
- Directives (commentaires) possibles dans le fichier source

Conséquences sur la lisibilité du code :

- Permet d'indiquer explicitement quels sont les opérandes de sortie et d'entrée (pas toujours très clair en Fortran...)
- `résultat = fonction(opérande1, opérande2, ...`
- Dépend de la spécification des variables (intent)

Syntaxe des intent

- intent(in) : variable d'entree dans la procédure
- intent(out) : variable de sortie. Sera récupérée comme résultat dans le code python

Fonctionnement de f2py II

- intent(copy) : la valeur de la variable d'origine sera conservée
- intent(overwrite) : la valeur de la variable d'origine n'est pas recopiée avant utilisation $\Rightarrow$ ecrasée par celle de la valeur de sortie. Utile pour éviter de copier. Fonctionnement par defaut

Le principe de fonctionnement général est représenté ci-dessous :

Fonctionnement de f2py III

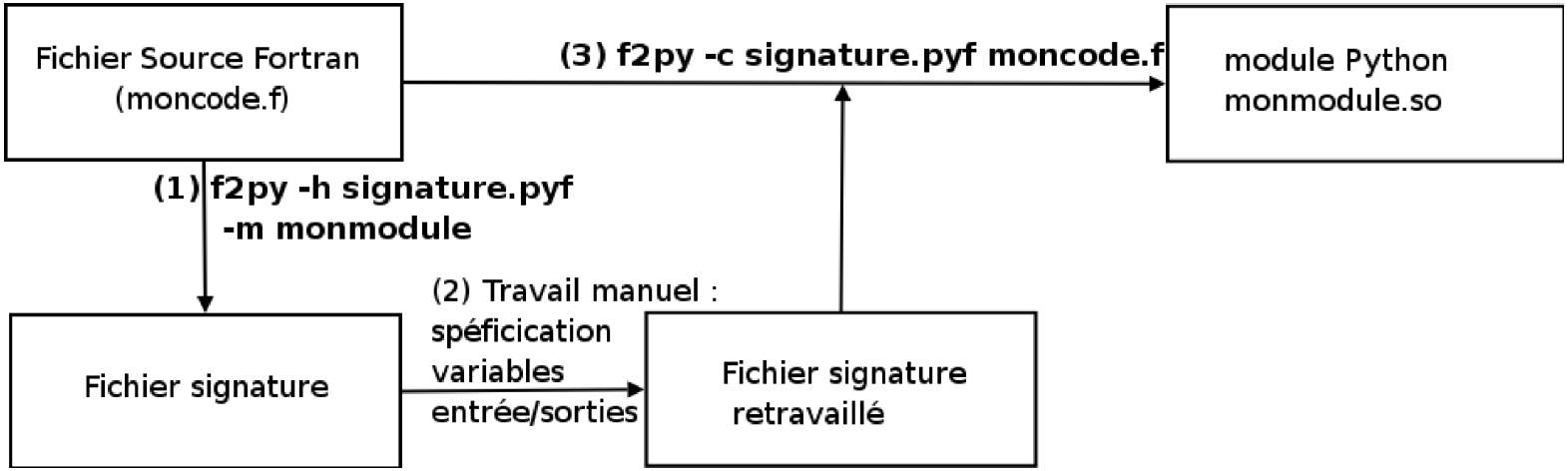


FIG.: Principe de fonctionnement de f2py

f2py : dernières remarques

Caractéristiques principales :

- Du petit exemple aux grosses bibliothèques
- Interfaçage de code C également
- Limitations : Types dérivés Fortran 90
- Autre générateur d'interfaces : Forthon

Quelles fonctions wrapper avec f2py ?

- les routines de calcul consommatrices en temps, du coeur du problème
- les routines d'E/S en Fortran (joke)

Dans un code couplé python/fortran, python est chargé du housekeeping (allant jusqu'au pré/post traitement), fortran du calcul "dur". En fortran, penser fonctions plutot que code complet.

Plan

Matplotlib : Un tracé simple I

Voici le code du sinus :

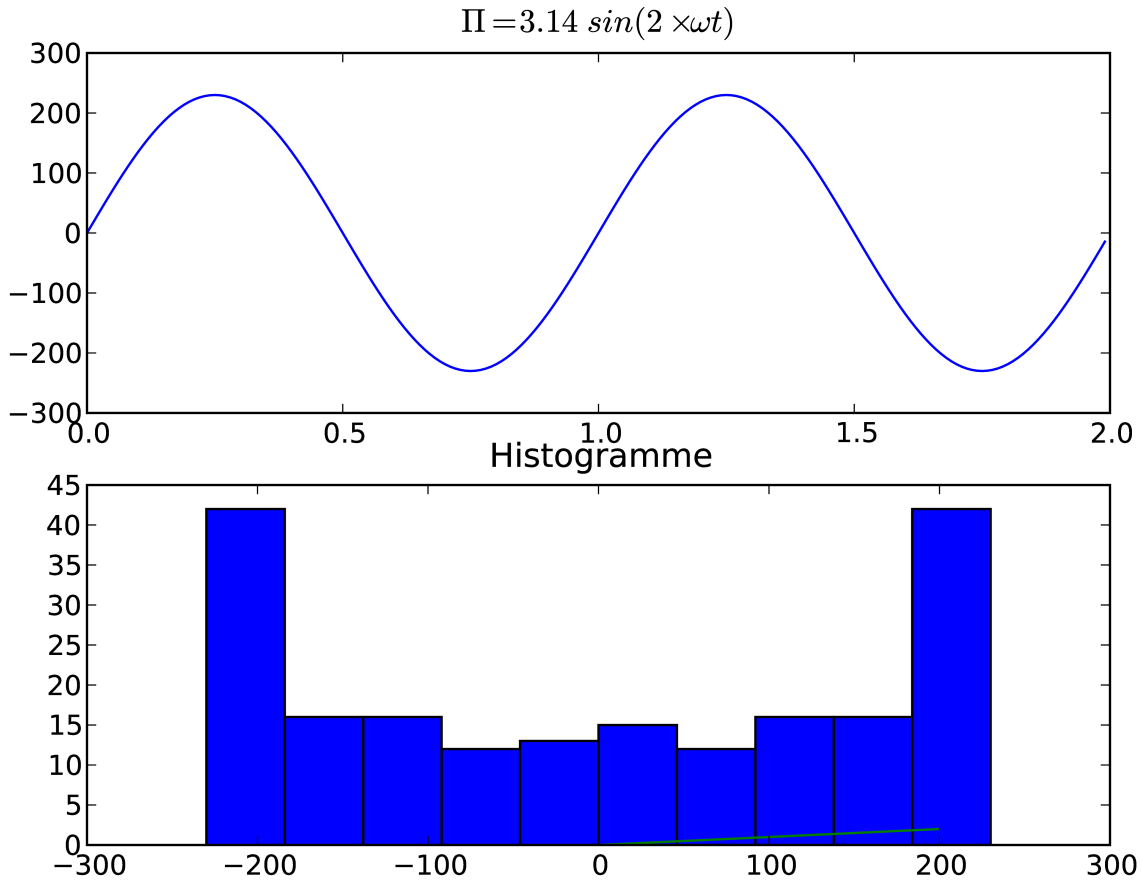
```
from pylab import *

# Definition de la serie de donnees en x : de 0
t = arange(0.0, 2.0, 0.01)

# Courant alternatif parfait
s = 230 * sin(2*pi*t)
# Plot absisces, ordonnees
subplot(211)
plot(t,s)
title(r'$\Pi =3.14 \ \ \ \ sin(2 \times \omega t)$


subplot(212)
```

Matplotlib : Un tracé simple II

```
n,h,p=hist(s)
plot(t)
title(r'Histogramme')
savefig('sinus.pdf')
show()
```

Fichier `Exemples/Matplotlib/sinus.py`

Matplotlib : Un tracé simple III



Plan

Plan

Forme que peut prendre un module

- un fichier `mod.so` (cas que nous avons vu jusqu'à présent)
- un fichier `mod.py`
- un répertoire `mod` contenant obligatoirement un fichier `__init.py__` qui sera interprété. Ce répertoire est prioritaire sur le fichier. S'il est présent, le fichier `mod.py` sera ignoré.

$\Rightarrow$ Une variable ou un fonction v d'un module m sera référencée par m.v.

Une hierarchie volontairement complexe...

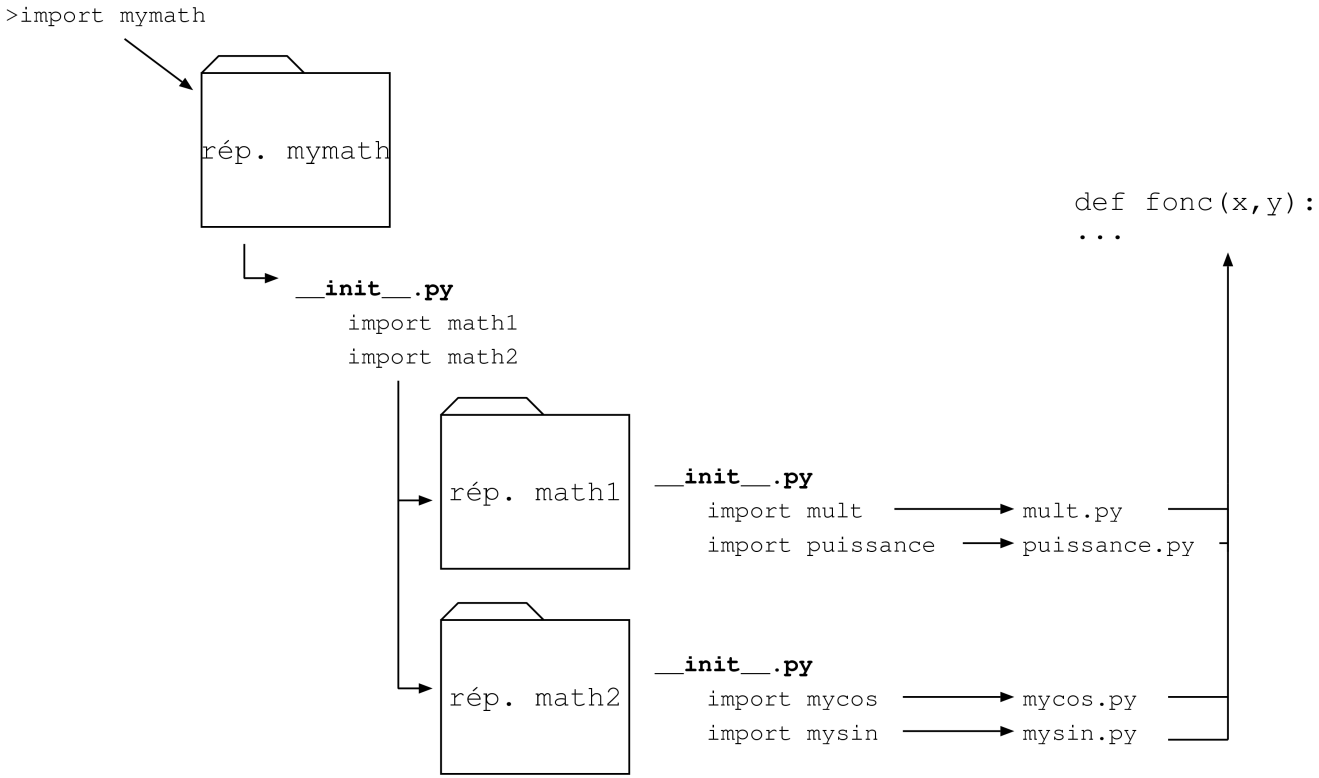


FIG.: Hiérarchie de modules