Python pour le calcul scientifique

Un pas vers l'optimisation

Loïc Gouarin

Laboratoire de Mathématiques d'Orsay

Python, c'est lent !

oui mais ...

- Python est un langage facile à apprendre.
- Il existe énormément de modules pour le calcul scientifique.
- On peut faire bien plus que du calcul scientifique.
- La communauté est gigantesque.

Reste donc aux développeurs le soin de faire l'optimisation de leurs codes pour avoir des utilisateurs conquis.

Comment optimiser son code

- trouver les régions lentes du code en faisant du profiling,
- optimiser ces parties en se ramenant à un langage bas niveau,
- l'interfaçage se fait via l'API Python/C.

Comment faire l'interface ?

- en écrivant tout seul l'interface,
- en utilisant SWIG,
- en utilisant f2py,
- en utilisant cython.

Cython : comment ça marche ?

- On écrit un fichier `.pyx` indiquant comment s'opère l'interfaçage entre les objets Python et du C.
- On utilise la commande `cython` pour créer l'interface `.c` écrite en API Python/C.
- On compile ce fichier `.c` en `.o`.
- On crée la librairie `.so` associée.
- On peut maintenant importer cette librairie dans Python et utiliser les fonctions ou classes ainsi créées.

Cython : la compilation ?

- Utiliser `cython` manuellement.
- Ecrire un fichier `setup.py` utilisant `distutils`.
- Utiliser `pyximport`.
- Utiliser Sage.

Cython : principe général

Définir des types statiques permettant à Cython de comprendre que nous ne sommes plus dans une partie Python mais dans une partie pouvant être écrite facilement en C et donc optimisée.

Premier exemple concret

On souhaite approcher π en calculant l'intégrale suivante

$$\pi = \int_0^1 f(x)dx \text{ avec } f(x) = \frac{4}{1+x^2}dx.$$

Pour ce faire, on approche l'intégrale en utilisant une méthode des rectangles

$$\pi \approx \frac{1}{n}\sum_{i=0}^{n-1} f(x_i), \text{ avec } x_i = \frac{i+0.5}{n} \text{ pour } i = 0, \cdots, n-1.$$

Premier exemple concret

```
import time

def f(x):
    return 4./(1 + x**2)

def calculPI(n):
    h, pi = 1./n, 0.
    for i in xrange(n):
        pi += f(h*(i+.5))
    return h*pi

if __name__=="__main__":
    n, nrep = 100000, 10
    t1 = time.time()
    for i in xrange(nrep):
        pi = calculPI(n)
    print pi, (time.time() - t1)/nrep
```

Passage à Cython

- Récupérer l'archive des TPs Cython se trouvant à l'adresse suivante
  http ://devlog.cnrs.fr/jdev2013/t7a2-3.
- Recopier les fonctions `f` et `calculPI` du fichier `pi.py` dans un fichier `piCython.pyx`.
- Utiliser le fichier `setup.py` pour créer la librairie en utilisant la commande

```
python setup.py build_ext --inplace
```

- Importer la fonction `calculPI` se trouvant dans la librairie créée et tester dans le script `pi.py`.

La commande magique

- Faire `cython -a piCython.pyx`.
- Ouvrir dans votre navigateur le fichier `piCython.html` ainsi créé.

Quelques directives importantes

- boundscheck
  si `False`, Cython assume que les indices demandés existent (`True` par défaut).

- wraparound
  si `False`, Cython ne vérifie pas si l'indice est négatif (`True` par défaut).

- cdivision
  si `True`, Cython fait une division en C (`False` par défaut).

Comment les utiliser ?

- En ligne de commande

```
cython -X boundscheck=True monfichier.pyx
```

- A toutes les fonctions du fichier `.pyx`

```
#cython: boundscheck=True
```

- uniquement à une fonction

```
@cython.boundscheck(True)
def mafonction(...):
```

Deuxième exemple : Cython + numpy

Pour illustrer l'utilisation de Cython avec des tableaux numpy, nous allons programmer le très connu **jeu de la vie**.

Rappel du principe

- Si une cellule a exactement trois voisines vivantes, elle est vivante à l'étape suivante.
- Si une cellule a exactement deux voisines vivantes, elle reste dans son état actuel à l'étape suivante.
- Si une cellule a strictement moins de deux ou strictement plus de trois voisines vivantes, elle est morte à l'étape suivante.

Deuxième exemple : Cython + numpy

Cython est livré avec un fichier `numpy.pyd` permettant de définir des types de tableaux numpy.

On y accède en utilisant

```
cimport numpy
```

Exemple d'utilisation

```
cimport numpy as np
def mafonction(np.ndarray[np.int_t, ndim=2] x):
    ...
```