

Mesurer des temps de calcul en Python

L2 MPCI - Valentin Emiya

Novembre 2022

Pour mesurer des temps de calcul, on utilisera le module `time`¹ qui fournit différentes fonctions pour mesurer le temps. On utilisera en particulier la fonction `process_time()` qui donne des mesures précises du temps écoulé dans le programme courant.

Principe : repérez le code que vous souhaitez chronométrier, il suffit ensuite d'encadrer ce code par deux appels à `process_time()` en stockant les résultats `t0` et `t1`. Chaque appel renvoyant l'instant courant, la différence `t1-t0` entre les deux mesures donne ensuite le temps passé à exécuter le code (voir exemple ci-dessous).

```
from time import process_time
import numpy as np
import matplotlib.pyplot as plt

n_range = 2**np.arange(2, 13)
t = np.empty(n_range.size)
for i, n in enumerate(n_range):
    a = np.random.randn(n, n)
    b = np.random.randn(n, n)
    t0 = process_time() # début chronométrage
    c = np.dot(a, b) # Code à chronométrier
    t1 = process_time() # fin chronométrage
    t[i] = t1 - t0
    print(f'Temps pour multiplier 2 matrices {n}x{n}: {t[i]:.6f}s')

plt.plot(n_range, t)
plt.xlabel('taille $n$')
plt.ylabel('temps (s)')
plt.xscale('log')
plt.yscale('log')
plt.grid(True)
plt.savefig('exemple_process_time.png')
```

Résultat affiché :

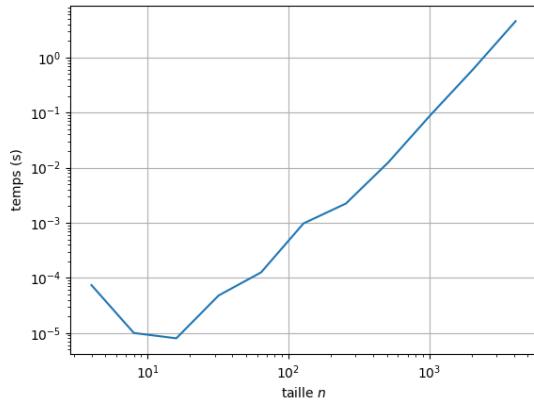
```
Temps pour multiplier 2 matrices 4x4: 0.000074s
Temps pour multiplier 2 matrices 8x8: 0.000010s
Temps pour multiplier 2 matrices 16x16: 0.000008s
Temps pour multiplier 2 matrices 32x32: 0.000048s
Temps pour multiplier 2 matrices 64x64: 0.000126s
```

1. <https://docs.python.org/3/library/time.html>

```

Temps pour multiplier 2 matrices 128x128: 0.000975s
Temps pour multiplier 2 matrices 256x256: 0.002238s
Temps pour multiplier 2 matrices 512x512: 0.012525s
Temps pour multiplier 2 matrices 1024x1024: 0.091418s
Temps pour multiplier 2 matrices 2048x2048: 0.614037s
Temps pour multiplier 2 matrices 4096x4096: 4.561713s

```



Le profiling. Si l'on souhaite aller plus loin dans l'analyse des temps d'exécution, on peut utiliser des outils de profiling. Il existe un profiler² intégré à Python (et les environnements de développement comme Visual Studio Code et PyCharm). Le paquet `line_profiler`³ est un outil encore plus détaillé qui permet de mesurer le temps d'exécution de chaque ligne de code dans des fonctions ou méthodes préalablement sélectionnées. Cela permet notamment de repérer les lignes de code gourmandes en temps en vue de cibler les parties à optimiser.

Les exemples fournis illustrent deux façons d'utiliser `line_profiler` :

- `demo_line_profiler.py` : sans toucher au code original, en indiquant les éléments à sélectionner via un objet `LineProfiler`.
- `demo_line_profiler_with_decorator.py` : en ajoutant des décorateurs `@profile` dans le code au niveau des éléments à sélectionner.

Dans les deux cas, on compare trois versions différentes de la multiplication de deux matrices : sans boucle, avec deux boucles et avec trois boucles. Un tableau affiche ligne par ligne une analyse des temps de calcul : nombre de passage dans la ligne (Hits), temps, temps par passage, pourcentage du temps total. On peut par exemple voir que la version sans boucle prend 0.6 ms alors que la version avec trois boucles prend 17 s (28000 fois plus lente!).

Line #	Hits	Time	Per Hit	% Time	Line Contents
<hr/>					
45					def dot(self):
46	3	0.0	0.0	0.0	A = self.A
47	3	0.0	0.0	0.0	B = self.B
48	3	0.6	0.2	0.0	x = mul_no_loop(A, B)
49	3	17178.3	5726.1	98.4	y = mul_3loops(A, B)
50	3	284.0	94.7	1.6	z = mul_2loops(A, B)
51	3	2.8	0.9	0.0	print(np.max(np.abs(x - y)))
52	3	0.4	0.1	0.0	print(np.max(np.abs(x - z)))

2. <https://docs.python.org/3/library/profile.html>

3. https://github.com/rkern/line_profiler