R1.06 TP 4

Suites aliquotes

2. Les nombres premiers à la rescousse

13 novembre 2025 – B. Colombel

Sommaire

Dans le TP précédent, nous avons utilisé un algorithme *naïf* pour déterminer les diviseurs stricts d'un entier positif n en calculant le reste de la division euclidienne de n par k avec $2 \leqslant k \leqslant n-1$.
Cette méthode s'avérant trop lente, l'objectif de ce TP est d'en améliorer l'efficacité. Pour cela, nous allons utiliser les nombres premiers.

Dans le cours, nous avons vu le résultat suivant :

> **Théorème 1 (Théorème des diviseurs premiers)**
> Soit n un entier naturel avec $n \geqslant 2$. Alors :
>
> 1. n admet au moins un diviseur premier : son plus petit diviseur dans $\mathbb{N}$ autre que 1.
> 2. Si n est composé, il admet un diviseur premier p tel que $2 \leqslant p \leqslant \sqrt{n}$.

Ainsi, il suffit de tester la divisibilité de n par tous les nombres premiers jusqu'à $\sqrt{n}$ pour déterminer les parties aliquotes de n.

Encore faut-il connaitre la liste des nombres premiers inférieurs ou égaux à $\sqrt{n}$. Nous allons étudier deux méthodes.

1 Test de primalité

Du théorème précédent découle le résultat suivant :

> **Théorème 2 (Test de primalité)**
> Soit un entier $n \geqslant 2$.
> Si n n'est divisible par aucun nombre premier p tel que $2 \leqslant p < \sqrt{n}$ alors n est premier.

Exercice 1.

1. Écrire une fonction `IsPrime(n)` qui retourne un booléen égal à `True` si n est premier et `False` sinon en utilisant le crirère ci-dessus.
   Indication. Pour simplifier, on pourra tester si les nombres impairs inférieurs ou égaux à $\sqrt{n}$ sont des diviseurs de n.
2. Écrire une fonction `premiers(n)` qui retourne la liste des nombres premiers inférieurs ou égaux à n.

Exemples d'utilisation.

[2]:
```
IsPrime(13)
```

[2]:
```
True
```

[3]:
```
IsPrime(6)
```

[3]:
```
False
```

[5]:
```
L1 = premiers(20); print(L1)
```

[5]:
```
[2, 3, 5, 7, 11, 13, 17, 19]
```

[6]:
```
L1 = premiers(100); print(L1)
```

[6]:
```
[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]
```

2 Crible d'Eratosthène

Rappels de cours. Nous avons vu en cours le crible d'Ératosthène. En voici le principe :

Étant donné la liste A des entiers de 2 à n, on procède comme suit :

- *première étape :* on considère le premier élément de A, soit 2, et on supprime de la liste A tous ses multiples sauf lui-même. Ainsi, les éléments 4, 6, 8, 10, 12, ..., c'est-à-dire tous les nombres pairs de la liste A sauf 2 sont supprimés. La liste A commence donc maintenant par 2, 3, 5, 7, 9, ...
- *deuxième étape :* on considère alors le deuxième élément de la liste A, soit 3 et on supprime de A tous ses multiples sauf lui-même. On remarque que 6 est déjà parti, que 9 sera supprimé, ..., etc. La liste A deviendra alors 2, 3, 5, 7, 11, 13, 17, 19, 23, 25, ...
- *et ainsi de suite :* on itère le procédé avec le troisième puis le quatrième élément de A, ..., etc.

Ce procédé a pour but d'éliminer les nombres composés de la liste A de sorte qu'il ne reste que les nombres premiers.

- *Condition d'arrêt :* Au k^{e} passage, on doit supprimer de la liste A les entiers $2A(k)$, $3A(k)$, ... mais en fait $A(k)^2$ est le premier de ces nombres qui n'a pas déjà été supprimé (pourquoi ?) ; de plus, on ne le supprime effectivement que s'il est inférieur ou égal au dernier terme de la liste A. Sinon, cette liste reste inchangée et le crible d'Eratosthène s'arrête.

Exercice 2. Écrire une fonction `eratosthene(n)` qui retourne la liste des nombres premiers inférieurs à n.

Vous pourrez commencer par initialiser une liste avec les entiers de 1 à n puis affecter la valeur `0` aux multiples que vous voulez supprimer. Pour initialiser une telle liste, vous pouvez utiliser la commande :

[8]:
```
L = [1..n] # ou L = list(range(1, n+1))
```

3 Comparaison des deux méthodes

Exercice 3.

1. Comparer les temps de calcul des deux méthodes avec $n = 1\ 000\ 000$.
2. Quel est le plus grand nombre premier inférieur à 1 000 000 ?
3. Combien y-a-t-il de nombres premiers inférieurs à 1 000 000 ?

Pour mesurer le temps d'exécution du programme, on peut utiliser la commende `%time`. Voici ce que j'obtiens sur on ordinateur personnel :

[9]:
```
%time L1 = premiers(1000000)
```

[9]:
```
CPU times: user 2min 45s, sys: 206 ms, total: 2min 46s
Wall time: 2min 46s
```

[10]:
```
%time L1 = eratosthene(1000000)
```

[10]:
```
CPU times: user 790 ms, sys: 30.9 ms, total: 821 ms
Wall time: 821 ms
```