

BROUILLON - SOUSTRAIRE LES PUISSANCES N° DE N NATURELS CONSÉCUTIFS

CHRISTOPHE BAL

*Document, avec son source L^AT_EX, disponible sur la page
<https://github.com/bc-writing/drafts>.*

Mentions « légales »

Ce document est mis à disposition selon les termes de la licence
Creative Commons “Attribution - Pas d’utilisation commerciale -
Partage dans les mêmes conditions 4.0 International”.



TABLE DES MATIÈRES

1.	Faire la différence avec des puissances	2
2.	Expérimentations et conjecture	3
3.	Une preuve polynomiale	5
4.	AFFAIRE À SUIVRE...	6

1. FAIRE LA DIFFÉRENCE AVEC DES PUISSANCES

Considérons l'algorithme suivant dont nous allons donner des cas d'application juste après.

Algorithme 1 : Version naturelle

Entrée : $n \in \mathbb{N}^*$

Sortie : ?

Actions

Choisir $(n + 1)$ naturels consécutifs : $k_1 < k_2 < \dots < k_{n+1}$.

$L \leftarrow [k_1^n, k_2^n, \dots, k_{n+1}^n]$

Tant Que $\text{taille}(L) \neq 1$:

$\text{new}L \leftarrow []$

Pour i **de** 1 **à** $\text{taille}(L) - 1$:

 Ajouter $(L[i + 1] - L[i])$ à droite de $\text{new}L$.

$L \leftarrow \text{new}L$

Renvoyer $L[1]$

Pour $n = 2$ avec $k_1 = 3$, $k_2 = 4$ et $k_3 = 5$, nous avons les valeurs suivantes de la liste L .

$$(1) L = [3^2, 4^2, 5^2] = [9, 16, 25]$$

$$(2) L = [16 - 9, 25 - 16] = [7, 9]$$

$$(3) L = [9 - 7] = [2]$$

L'algorithme renvoie donc 2 ici mais que se passe-t-il si l'on choisit d'autres naturels consécutifs? Avec $k_1 = 10$, $k_2 = 11$ et $k_3 = 12$, nous obtenons :

$$(1) L = [10^2, 11^2, 12^2] = [100, 121, 144]$$

$$(2) L = [121 - 100, 144 - 121] = [21, 23]$$

$$(3) L = [23 - 21] = [2]$$

L'algorithme renvoie de nouveau 2. Que se passerait-il pour d'autres triplets de naturels consécutifs? Pour se faire une bonne idée, il va falloir utiliser un programme. Ceci étant dit nous allons toute de suite faire l'hypothèse audacieuse que le choix des naturels consécutifs n'est pas important. Regardons alors ce que renvoie l'algorithme pour $n = 3$.

$$(1) L = [0^3, 1^3, 2^3, 3^3] = [0, 1, 8, 27]$$

$$(2) L = [1, 7, 19]$$

$$(3) L = [6, 12]$$

$$(4) L = [6]$$

L'algorithme renvoie 6 pour $n = 3$, et pour $n = 4$ ce qui suit nous donne que 24 est renvoyé. Ceci nous fait alors penser à $n!$ et donc nous amène à conjecturer, un peu rapidement c'est vrai, que l'algorithme va toujours renvoyer $n!$.

$$(1) L = [0^4, 1^4, 2^4, 3^4, 4^4] = [0, 1, 16, 81, 256]$$

$$(2) L = [1, 15, 65, 175]$$

$$(3) L = [14, 50, 110]$$

$$(4) L = [36, 60]$$

$$(5) L = [24]$$

Il est temps de passer aux choses un peu plus sérieuses via une expérimentation informatique bien plus poussée.

2. EXPÉRIMENTATIONS ET CONJECTURE

Le code suivant¹ permet de tester notre conjecture audacieuse : aucun test raté n'est révélé.

Code Python

```

from math import factorial
from random import randint

NMAX      = 20
POWER_MAX = 6
NB_TESTS  = 10**6

print ("TEST - START")

kmax = 10**POWER_MAX

for _ in range(NB_TESTS):
    n      = randint(1, NMAX)
    start  = randint(0, kmax)

    L = [i*n for i in range(start, start + n + 1)]

    while len(L) != 1:
        newL = []

        for i, elt in enumerate(L[:-1]):
            newL.append(L[i+1] - elt)

        L = newL[:]

    if L[0] != factorial(n):
        print (f"      * Test failed with n = {n}")
        exit()

print ("TEST - END")

```

Comme les valeurs des naturels consécutifs ne semblent pas importantes, on peut pousser l'expérimentation en travaillant avec $k_1 \in \mathbb{R}$, $k_2 = k_1 + 1$, $k_3 = k_1 + 2$, ... Ceci étant dit, il n'est pas toujours possible de travailler avec des réels en informatique : l'usage des flottants s'accompagnent de son lot d'arrondis. Nous allons donc juste tester des valeurs rationnelles via le code suivant² qui ne révèle aucun test raté.

Code Python

```

from math import factorial
from random import randint

```

1. Ce fichier `diff-n-cons-int-to-the-power-of-n/exploring-int-version.py` est disponible dans le sous-dossier sur le lieu de téléchargement de ce document.

2. Ce fichier `diff-n-cons-int-to-the-power-of-n/exploring-frac-version.py` est disponible dans le sous-dossier sur le lieu de téléchargement de ce document.

```

from sympy import S

NMAX      = 20
POWER_MAX = 6
NB_TESTS  = 10**3

print("TEST - START")

kmax = 10**POWER_MAX

for _ in range(NB_TESTS):
    n      = randint(1, NMAX)
    start  = S(randint(0, kmax)) / S(randint(1, kmax))

    L = [(start + i)**n for i in range(n + 1)]

    while len(L) != 1:
        newL = []

        for i, elt in enumerate(L[:-1]):
            newL.append(L[i+1] - elt)

        L = newL[:]

    if L[0] != factorial(n):
        print(f"      * Test failed with n = {n}")
        exit()

print("TEST - END")

```

Il devient donc normal de penser que le phénomène est purement polynomial. Nous allons explorer ceci dans la section suivante.

3. UNE PREUVE POLYNOMIALE

Voici une légère modification de l'algorithme 1 où l'on manipule des polynômes dans $\mathbb{R}[X]$. Notons au passage que nous passons d'un algorithme a priori indéterministe, car on fait un choix de naturels consécutifs, à un autre complètement déterministe.

Algorithme 2 : Version polynomiale**Entrée** : $n \in \mathbb{N}^*$ **Sortie** : ?**Actions** $L \leftarrow [X^n, (X+1)^n, \dots, (X+n)^n]$ **Tant Que** $\text{taille}(L) \neq 1$: $\quad \text{newL} \leftarrow []$ $\quad \text{Pour } i \text{ de } 1 \text{ à } \text{taille}(L) - 1$: $\quad \quad \text{Ajouter } (L[i+1] - L[i]) \text{ à droite de } \text{newL}.$ $\quad L \leftarrow \text{newL}$ **Renvoyer** $L[1]$

Il est aisée d'implémenter cet algorithme en Python. Le code suivant ne révèle aucun test raté.

Code Python

```

from math import factorial
from random import randint

from sympy import poly
from sympy.abc import x

NMAX      = 20
NB_TESTS  = 10**4

print("TEST - START")

for _ in range(NB_TESTS):
    n = randint(1, NMAX)
    L = [poly((x + i)**n) for i in range(n + 1)]

    while len(L) != 1:
        newL = []

        for i, elt in enumerate(L[:-1]):
            newL.append(L[i+1] - elt)

        L = newL[:]

    result = L[0]
    result = result.expand()

    if result.degree() != 0 \
    or L[0] != factorial(n):

```

```

    print(f"      * Test failed with n = {n}")
    exit()

print("TEST - END")

```

Il est clair que si l'on prouve que l'algorithme 2 renvoie $n!$, il en sera de même pour 1. Nous allons en fait prouver la validité de l'algorithme ci-dessous où $\mathbb{R}_n[X]$ désigne l'ensemble des polynômes réels de degré $n \in \mathbb{N}^*$.

Algorithme 3 : Version polynomiale élargie

Entrée : $P \in \mathbb{R}_n[X]$ de coefficient dominant a_n

Sortie : $a_n n!$

Actions

$L \leftarrow [P(X), P(X+1), \dots, P(X+n)]$

Tant Que $\text{taille}(L) \neq 1$:

$\text{newL} \leftarrow []$

Pour i **de** 1 **à** $\text{taille}(L) - 1$:

 Ajouter $(L[i+1] - L[i])$ à droite de newL .

$L \leftarrow \text{newL}$

Renvoyer $L[1]$

PREUVE : TODO

4. AFFAIRE À SUIVRE...
