BROUILLON - SOUSTRAIRE LES PUISSANCES N$^{\text{e}}$ DE N NATURELS CONSÉCUTIFS

CHRISTOPHE BAL

Document, avec son source LaTeX, disponible sur la page `https://github.com/bc-writing/drafts`.

Mentions « légales »

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

Date: 3 Octobre 2019 – 8 Octobre 2019.

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 < ··· < k_{n+1}.
L ← [k_1^n, k_2^n, ..., k_{n+1}^n]
Tant Que taille(L) ≠ 1 :
    newL ← [ ]
    Pour i de 1 à taille(L) − 1 :
        Ajouter (L[i + 1] − L[i]) à droite de newL.
    L ← newL
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 renvoyé $n!$.

(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 plus sérieuses via une expérimentaion informatique bien plus poussée.

2. Expérimentations et conjecture

Le code suivant [1] 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'accompagne de son lot d'arrondis. Nous allons donc juste tester des valeurs rationnelles via le code suivant [2] 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.

```python
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 $\mathbb{R}\left[X\right]$. 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
- $L \leftarrow [X^n, (X+1)^n, \ldots, (X+n)^n]$
- **Tant Que** $taille(L) \neq 1$ **:**
  - $newL \leftarrow [\ ]$
  - **Pour** i **de** 1 **à** $taille(L) - 1$ **:**
    - Ajouter $(L[i+1] - L[i])$ à droite de $newL$.
  - $L \leftarrow newL$
- **Renvoyer** $L[1]$

Il est aisée de traduire cet algorithme en Python. Le code suivant ne révèle aucun test raté.

Code Python

```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. Ceci va découler de la validité de l'algorithme ci-dessous où $\mathbb{R}_n\left[X\right]$ désigne l'ensemble des polynômes réels de degré $n \in \mathbb{N}^*$.

Algorithme 3 : Version polynomiale élargie

Entrée : $P \in \mathbb{R}_n\left[X\right]$ de coefficient dominant a_n
Sortie : $n!\, a_n$
Actions

- $L \leftarrow [\, P(X), P(X+1), \ldots, P(X+n) \,]$
- **Tant Que** $taille(L) \neq 1$ **:**
  - $newL \leftarrow [\ \ ]$
  - **Pour** i **de** 1 **à** $taille(L) - 1$ **:**
    - Ajouter $(L[i+1] - L[i])$ à droite de $newL$.
  - $L \leftarrow newL$
- **Renvoyer** $L[1]$

Pourquoi cet algorithme est-il valide ? Pour la suite, nous posons $P(x) = \sum\limits_{k=0}^{n} a_k X^k$.

(1) Il est immédiat que $\forall k \in \mathbb{N}^*$, $(X+1)^k - X^k = kX^{k-1} + R(X)$ où $\deg R < k-1$ avec la convention $\deg 0 = -\infty$.

(2) Le point précédent donne sans effort $P(X+1) - P(X) = na_n X^{n-1} + S(X)$ où le polynôme S vérifie $\deg S < n-1$. Tout est dit comme nous allons le voir.

(3) A la 1re itération de la boucle, la liste $[\, P(X), P(X+1), \ldots, P(X+n) \,]$ est transformée en $[\, P(X+1) - P(X), P(X+2) - P(X+1), \ldots, P(X+n) - P(X+n-1) \,]$ soit $[\, Q(X), Q(X+1), \ldots, Q(X+n-1) \,]$ si l'on pose $Q(X) = P(X+1) - P(X)$ qui a pour coefficient dominant na_n et pour degré $\deg Q = n-1$.

(4) Il est alors facile de faire une récurrence pour prouver que la boucle se finit en fournissant ce qui est annoncé.

Joli, efficace et éclairant sur le pourquoi du comment. Il se trouve que l'algorithme 1 cache aussi une formule combinatoire très intéressante comme nous allons le voir dans la section qui suit où nous allons reprendre la démarche découverte sur le site The Math Less Traveled [3]

---

4. AFFAIRE À SUIVRE...

---

3. Chercher l'article *« A combinatorial proof »* en 2019.