Puissances de 2

Année 2022-2023

Auteurs : BARATEAU Titouan, BODIN Noan, BOIRE Ylane (Première générale).
Établissement : Lycée Paul Guérin, Niort (79).
Encadrés par : Fabien AOUSTIN, Thomas FORGET.

Dans cet article, nous nous sommes intéressés à différentes propriétés suivies par les chiffres qui composent les différentes puissances de 2. Elles ont été conjecturées grâce à des algorithmes.

1) Un comportement d'apparence naturel

Pour commencer, nous avons calculé les premières puissances de 2, pour y trouver des propriétés qui semblaient apparaître.

Nous avons remarqué que le chiffre des unités est répété toutes les 4 étapes, en partant de 2^1, et en suivant l'ordre 2-4-8-6, qui est l'ordre obtenu lorsque l'on multiplie par 2.
($2\times2=4$, $4\times2=8$, $8\times2=16$ qui se finit par 6, et $6\times2=12$ qui se finit par 2.)

Si l'on s'intéresse aux deux derniers chiffres, nous avons conjecturé qu'ils seraient répétés toutes les 20 étapes, en partant de 2^2.

À l'aide d'un algorithme, nous avons observé que les trois derniers chiffres semblent être répétés toutes les 100 étapes, en partant de 2^3.

Nous avons alors conjecturé que les k derniers chiffres semblent répétés toutes les $4\times5^{k-1}$ étapes, en partant de 2^k.

Cette conjecture semble se confirmer en utilisant notre algorithme (voir page suivante) pour les quatre derniers chiffres, qui semblent bien répétés toutes les 500 étapes en partant de 2^4, et pour les cinq derniers chiffres qui semblent répétés toutes les 2 500 étapes en partant de 2^5.

n	2^n	chiffre des unités	2 derniers chiffres	3 derniers chiffres
0	1	1	1	1
1	2	2	2	2
2	4	4	4	4
3	8	8	8	8
4	16	6	16	16
5	32	2	32	32
6	64	4	64	64
7	128	8	28	128
8	256	6	56	256
9	512	2	12	512
10	1024	4	24	24
11	2048	8	48	48
12	4096	6	96	96
13	8192	2	92	192
14	16384	4	84	384
15	32768	8	68	768
16	65536	6	36	536
17	131072	2	72	72
18	262144	4	44	144
19	524288	8	88	288
20	1048576	6	76	576
21	2097152	2	52	152
22	4194304	4	4	304
23	8388608	8	8	608

Dans notre algorithme Python, nous avons utilisé les calculs de quotients et de restes.
Par exemple : 123456789 // 100 renvoie le quotient de la division par 100, donc 1234567.
123456789 % 100 renvoie le reste de la division par 100, donc 89.

Nous avons généré l'affichage d'une liste des restes, selon le nombre de derniers chiffres que l'on souhaite garder. Cette liste s'arrêtant dés la première répétition.
Puis, par souci de lisibilité, nous avons supprimé les premiers termes de cette liste qui ne sont pas répétés comme par exemple le 02 associé à 2^1 lorsque l'on s'intéresse aux deux derniers chiffres :

```
nbchiffres = 7

L = [1]
a = 2
compteur = 1
while a not in L :
    L.append(a)
    a = (2*a)%(10**nbchiffres)
    compteur = compteur + 1
debut = L.index(a)
ecart = compteur - debut

# On veut retirer de la liste les nombres qui ne sont pas répétés.
# Pour les nbchiffres premiers chiffres, on veut donc retirer les nbchiffres premiers termes de la liste.

del L[0:nbchiffres]
print(a,ecart)

# la liste n'a conservé que les nbchiffres premiers chiffres de chaque puissance de 2
# Si l'on veut garder le chiffre de rang nbchiffes, il nous suffit de saisir
# n//(10**(nbchiffres-1))
# Par exemple 1789 devient 789, et 789//100 (qui est 10**3) renvoie 7

Lreduit = [n//10**(nbchiffres -1) for n in L]

# Et on veut dénombrer le nombre de répétitions d'un certain chiffre à cette position
# Nous affichons chaque chiffre possible, en indiquant sa fréquence d'apparition.
nb0 = Lreduit.count(0)
print (0,nb0/ecart)
nb1 = Lreduit.count(1)
print (1,nb1/ecart)
nb2 = Lreduit.count(2)
print (2,nb2/ecart)
nb3 = Lreduit.count(3)
print (3,nb3/ecart)
nb4 = Lreduit.count(4)
print (4,nb4/ecart)
nb5 = Lreduit.count(5)
print (5,nb5/ecart)
nb6 = Lreduit.count(6)
print (6,nb6/ecart)
nb7 = Lreduit.count(7)
print (7,nb7/ecart)
nb8 = Lreduit.count(8)
print (8,nb8/ecart)
nb9 = Lreduit.count(9)
print (9,nb9/ecart)

#codage raccourci de ce tableau de valeurs :
#for chiffre in range(10):
#    nb = Lreduit.count(chiffre)
#    print (chiffre, nb/ecart)
```

En partant du principe que cette conjecture est vraie, nous en avons déduit un moyen pour déterminer rapidement les derniers chiffres de n'importe quelle puissance de 2.

- Par exemple, si on s'intéresse au dernier chiffre de 2^{2023} :

On sait que le dernier chiffre est répété toutes les 4 étapes en partant de 2^1. On effectue donc la division euclidienne de 2023 par 4, qui est $2023 = 4 \times 505 + 3$.

Mais, afin de prendre en compte le fait que l'on commence au rang 1, nous allons le prendre en compte en écrivant plutôt cette égalité : $2023 = 1 + (4 \times 505 + 2)$.
Nous en déduisons que 2^{2023} a le même dernier chiffre que $2^{1+2} = 2^3 = 8$.

- Pour les deux derniers chiffre de 2^{2023} :

On sait que les deux derniers chiffres sont répétés toutes les 20 étapes en partant de 2^2. Il faut prendre en compte le fait que l'on commence au rang 2, ce qui nous conduit à $2023 = 2 + (20 \times 101 + 1)$. Et on en déduit que 2^{2023} a les deux mêmes derniers chiffres que $2^{2+1} = 08$.

- Pour les deux derniers chiffre de 2^{2021} :

Comme $2021 = 2 + (20 \times 100 + 19)$, ce qui ressemble à l'exemple précédent.
Comme 2^1 n'est pas répété, alors 2^{2021} a les deux mêmes derniers chiffre que $2^{2+19} = 2^{21}$ qui sont 52.

- Pour les trois derniers chiffres de 2^{2023} :

Les répétitions se font toutes les 100 étapes en partant de 2^3.
Comme $2023 = 3 + (100 \times 20 + 20)$, les trois derniers chiffres de 2^{2023} sont les mêmes que ceux de $2^{3+20} = 2^{23}$ qui sont 608.

Après cela, nous avons cherché à estimer la fréquence d'apparition de chaque chiffre pour chaque position en partant de la droite, dans toutes les puissances de 2.
Pour le dernier chiffre, nous retrouvons sans surprise 2, 4, 6 et 8, qui semblent apparaître chacun 25 % du temps.
Et, à chacune des autres positions (en partant de la droite), les simulations réalisées nous conduisent à conjecturer que les dix chiffres de 0 à 9 apparaissent chacun 10 % du temps.

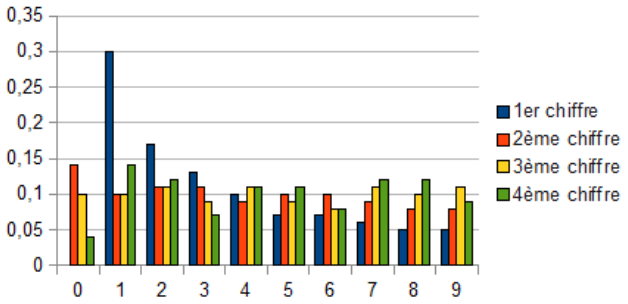
Ce constat ne semble pas se généraliser au puissances de n'importe quel entier.
Par exemple, il semble être mis en défaut pour les puissances de 4, en utilisant une version adaptée de l'algorithme précédent.

2) Des apparences trompeuses

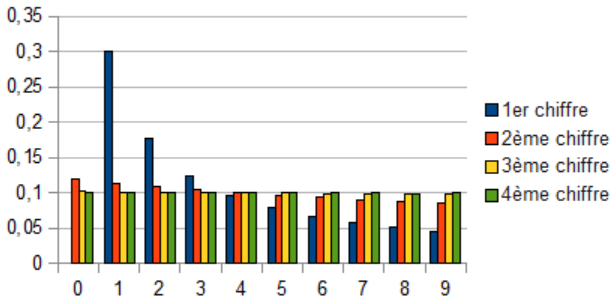
Les constats précédents indiquent une régularité de la fréquence d'apparition de chaque chiffre qui composent les puissances de 2, lorsque l'on observe leur position en partant de la droite.
Nous nous sommes alors intéressés aux chiffres apparaissant aux positions en partant de la gauche.
De nouveau, nous avons utilisé des algorithmes pour observer les différentes fréquences pour différentes positions.

```
# Calculer la fréquence d'apparition de chaque chiffre pour le premier chiffre d'une puissance de 2

# Dans un premier temps, on crée la liste des premiers chiffres des n premières puissances de 2

p = 100000
a = 0
b = 100
N = []


Lreduit = [int(str(2**n)[0:2]) for n in range(4,p+4) ]

while a!=b :
    a = a+1
    c = Lreduit.count(a)
    N.append(c/p)


print(Lreduit)
print(N)
```

Nous avons observé les résultats pour les $n = 100$, et les $n = 100\,000$ premières puissances de 2, en déterminant la fréquence du 1er, du 2nd, du 3ième et du 4ième chiffre des puissances de 2 (à partir de la gauche), qui sont résumées dans les graphiques ci-dessous :



$n = 100$



$n = 100\,000$

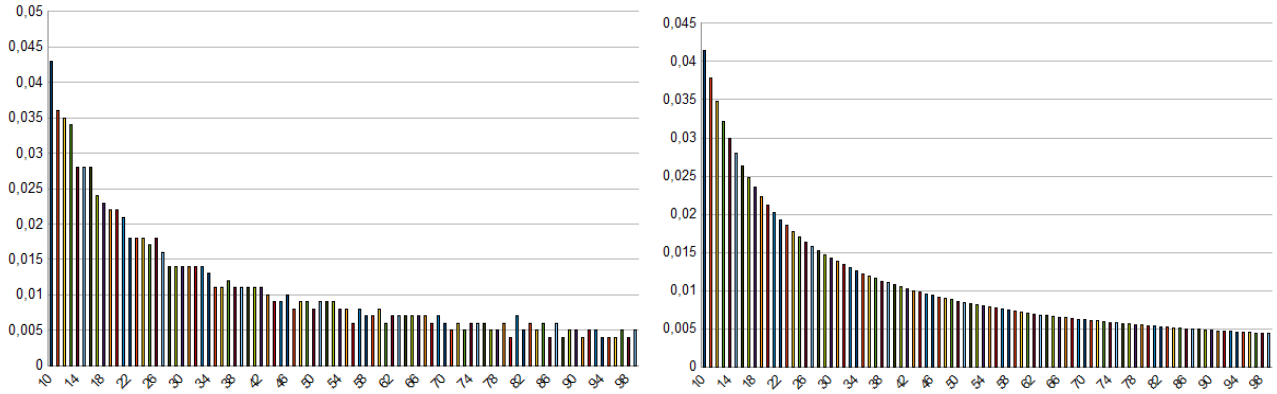
Nous observons que pour le premier chiffre d'une puissance de 2 (en partant de la gauche), la fréquence d'apparition du 1 est très nettement supérieure à celle des autres chiffres, et que cette fréquence est de plus en plus faible quand ce chiffre augmente (le chiffre 0 étant évidemment exclu à cette position).

De plus, les fréquences de chaque chiffres tendent à devenir identiques à mesure que l'on progresse dans la position que observée dans les puissances de 2.

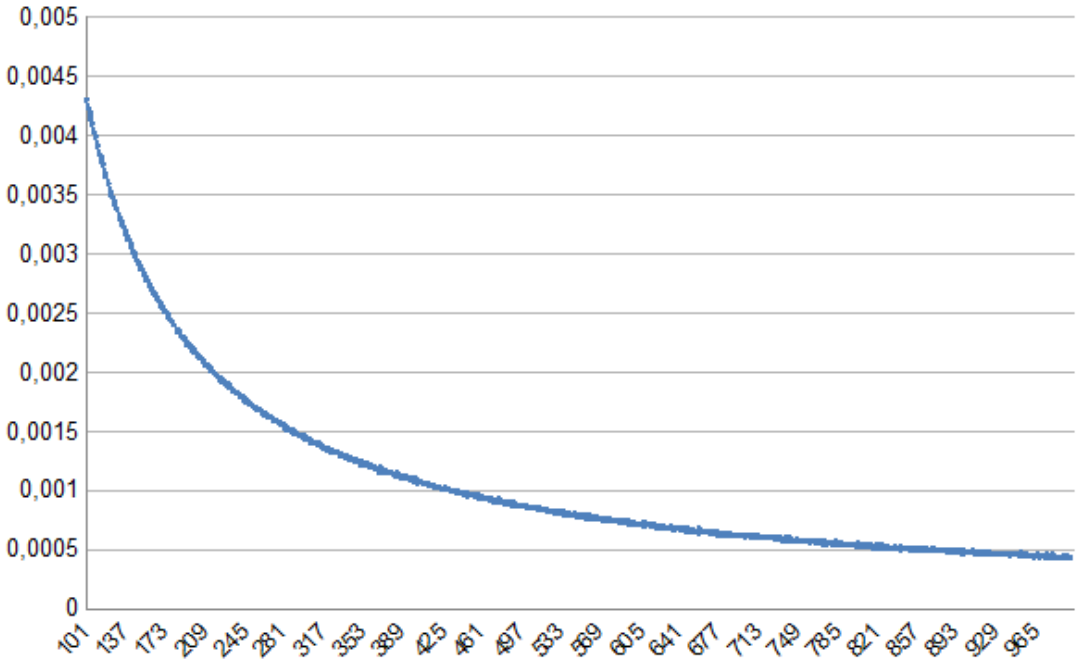
Ce résultat nous semble surprenant, car les positions observées en partant de la gauche ne correspondent pas aux positions que l'on aurait en regardant le chiffre à partir de la droite. Pourtant on semble retrouver assez rapidement une fréquence égale d'apparition de chacun des 10 chiffres.
Nous avons retrouvé ces fréquences en complétant l'algorithme précédent :

```
# Calculer la fréquence d'apparition de chaque chiffre pour le premier chiffre d'une puissance de 2

# Dans un premier temps, on crée la liste des premiers chiffres des n premières puissances de 2

p = 10000

Lreduit = [int(str(2**n)[0]) for n in range(p) ]

nb0 = Lreduit.count(0)
nb1 = Lreduit.count(1)
nb2 = Lreduit.count(2)
nb3 = Lreduit.count(3)
nb4 = Lreduit.count(4)
nb5 = Lreduit.count(5)
nb6 = Lreduit.count(6)
nb7 = Lreduit.count(7)
nb8 = Lreduit.count(8)
nb9 = Lreduit.count(9)

print (nb0/p,nb1/p,nb2/p,nb3/p,nb4/p,nb5/p,nb6/p,nb7/p,nb8/p,nb9/p)

#codage raccourci de ce tableau de valeurs :
#for chiffre in range(10):
#    nb = Lreduit.count(chiffre)
#    print (chiffre, nb/ecart)
```

Pour finir, nous nous sommes intéressés aux fréquences d'apparition de chiffres aux deux premières positions (en partant de la gauche) des puissances de 2, puis aux fréquences d'apparition des chiffres aux trois premières positions.
Les fréquences observées semblent suivre une progression assez remarquable :



fréquences des deux premiers chiffres des $n = 1000$ *et des* $n = 100\,000$ *premières puissances de* 2



fréquence des trois premiers chiffres des $n = 100\,000$ *premières puissances de* 2

On remarque que la courbe qui se dessine à mesure que n augmente ressemble à celle d'une décroissance radioactive.

Lorsque l'on s'intéresse aux n premiers chiffres des puissances de 2, le plus probable semble être de trouver 10...0, et le moins probable semble être 99...9 (avec une probabilité qui semble se rapprocher de 0).
Étant donnés les résultats obtenus pour les 100 000 premières puissances de 2, il semble que, pour les 2 premiers chiffres, la fréquence d'apparition de 10 semble être approximativement 0,0414 et, pour les 3 premiers chiffres, la fréquence d'apparition de 100 semble être approximativement 0,00432.