Pi2

February 16, 2019

1 La millionième décimale de π (II)

Marc Lorenzi
16 février 2019

```
In [1]: from sympy import *
        init_printing()
        x = Symbol('x')
        import math
```

Nous allons dans ce notebook nous intéresser à la question suivante :
Est-il possible de calculer la millionième décimale de π

- **SANS calculer les décimales précédentes**, et
- **En effectuant uniquement des calculs en précision standard** ?

En 1997, David Bailey, Peter Borwein et Simon Plouffe ont démontré que ceci était possible, *en base 16*. L'aricle original se trouve facilement sur Internet, et il est tout à fait lisible. Pour les lecteurs motivés, en voici la référence :

D. Bailey, P. Borwein, S. Plouffe, *On the Rapid Computation of Various Polylogarithmic Constants*, Mathematics of Computation, Vol 66, Number 218, April 1997, Pages 903-913.

On peut donc déterminer le $d$ième chiffre de π en base 16, ainsi que quelques chiffres suivants, en un temps quasi-linéaire et en calculant en précision standard. Nous allons tout d'abord établir la formule qui permet ce calcul, puis faire le dit calcul ...

1.1 1. Une intégrale

1.1.1 1.1 Intégrons une fraction rationnelle

Considérons la fraction rationnelle suivante.

```
In [2]: F = (-8 * x ** 5 - 4 * sqrt(2) * x ** 4 - 8 * x ** 3 + 4 * sqrt(2)) / (1 - x ** 8)
        F
```

Out[2]:

$$\frac{-8x^5 - 4\sqrt{2}x^4 - 8x^3 + 4\sqrt{2}}{-x^8 + 1}$$

Le dénominateur de F est immédiat à factoriser.

In [3]:
```
factor(1 - x ** 8)
```

Out[3]:

$$-(x-1)(x+1)\left(x^2+1\right)\left(x^4+1\right)$$

La décomposition de F en éléments simples est

In [4]:
```
apart(F, x)
```

Out[4]:

$$-\frac{4\left(x-\sqrt{2}\right)}{x^2-\sqrt{2}x+1}+\frac{2}{x+1}+\frac{2}{x-1}$$

On en déduit facilement une primitive de F.

In [5]:
```
G = integrate(F, x)
G
```

Out[5]:

$$2\log\left(x^2-1\right)-2\log\left(x^2-\sqrt{2}x+1\right)+4\,\mathrm{atan}\left(\sqrt{2}x-1\right)$$

Calculons $\int_0^{\sqrt{(2)}/2} F(x)dx$.

In [6]:
```
G.subs(x, sqrt(2) / 2) - G.subs(x, 0)
```

Out[6]:

$$\pi$$

In [7]:
```
init_printing(pretty_print=False)
```

1.1.2 1.2 Échange d'une intégrale et d'une somme

Faisons maintenant un petit calcul ... On a pour tout $x \in [0, \frac{\sqrt{2}}{2}]$

$$\frac{-8x^5-4\sqrt{2}x^4-8x^3+4\sqrt{2}}{1-x^8}=(-8x^5-4\sqrt{2}x^4-8x^3+4\sqrt{2})\sum_{i=0}^{\infty}x^{8i}$$

La série convergeant uniformément sur $[0, \frac{\sqrt{2}}{2}]$, on peut échanger la série et l'intégrale :

$$\int_0^{\sqrt{2}/2}F(x)dx=\sum_{i=0}^{\infty}\left(-8\int_0^{\sqrt{2}/2}x^{8i+5}dx-4\sqrt{2}\int_0^{\sqrt{2}/2}x^{8i+4}dx-8\int_0^{\sqrt{2}/2}x^{8i+3}dx+4\sqrt{2}\int_0^{\sqrt{2}/2}x^{8i}dx\right)$$

Laissons le reste au lecteur. Le membre de gauche vaut π et les intégrales du membre de droite sont faciles à calculer. On obtient

$$\sum_{i=0}^{\infty} \frac{1}{16^i}\left(\frac{4}{8i+1} - \frac{2}{8i+4} - \frac{1}{8i+5} - \frac{1}{8i+6}\right) = \pi$$

C'est cette dernière formule qui va nous permettre de calculer des décimales de π. Remarquez les puissances de 16. Si nous voulons calculer le $(d+1)$-ième chiffre hexadécimal de π,

- Nous multiplions π par 16^d.
- Nous ne gardons que la partie fractionnaire du résultat.
- Nous prenons le premier chiffre hexadécimal de ce que nous venons de trouver.

1.2 2. L'algorithme de Bailey, Borwein et Plouffe (BBP)

1.2.1 2.1 Partie fractionnaire

Notons, pour tout réel $x \geq 0$, $x \bmod 1 = x - \lfloor x \rfloor$ la partie fractionnaire de x.

```
In [8]: def frac(x):
            return x - math.floor(x)
```

Pour $q = 1, 4, 5, 6$, posons

$$S_q = \sum_{i=0}^{\infty} \frac{1}{16^i}\frac{1}{8i+q}$$

Soit $d \geq 0$. On a

$$(16^d S_q) \bmod 1 = \sum_{i=0}^{d} \frac{16^{d-i}}{8i+q} \bmod 1 + \sum_{i=d+1}^{\infty} \frac{16^{d-i}}{8i+q} \bmod 1$$

ou encore

$$(16^d S_q) \bmod 1 = \sum_{i=0}^{d} \frac{16^{d-i} \bmod (8i+q)}{8i+q} \bmod 1 + \sum_{i=d+1}^{\infty} \frac{16^{d-i}}{8i+q} \bmod 1$$

Notons dans ce qui suit

$$S_{dq} = \sum_{i=0}^{d} \frac{16^{d-i} \bmod (8i+q)}{8i+q} \bmod 1$$

et

$$S'_{dq} = \sum_{i=d+1}^{\infty} \frac{16^{d-i}}{8i+q} \bmod 1$$

1.2.2 2.2 Le calcul de S_{dq}

Ce calcul demande l'évaluation de 16^{d-i} modulo $8i+q$. Pour cela, on utilise une exponentiation rapide. La fonction `puissance` prend en paramètres trois entiers x, n, p. Elle renvoie $x^n \bmod p$.

```
In [9]: def puissance(x, n, p):
            z = 1
            m = n
            y = x
            while m != 0:
                if m % 2 == 1:
                    z = (z * y) % p
                m = m // 2
                y = (y * y) % p
            return z
```

```
In [10]: puissance(2, 10000000000, 10000)
```

```
Out[10]: 9376
```

La somme S_{dq} est alors facilement calculable.

```
In [11]: def somme1(d, q):
             s = 0
             for i in range(d + 1):
                 s = frac(s + puissance(16, d - i, 8 * i + q) / (8 * i + q))
             return s
```

```
In [21]: somme1(10 ** 5, 1)
```

```
Out[21]: 0.27179622818372623
```

1.2.3 2.3 Le calcul de S'_{dq}

Pour le calcul de S'_{dq} on se contente d'additionner un certain nombre de ses termes, N disons, en virgule flottante. Bref, on approche S'_{dq} par

$$\sum_{i=d+1}^{d+N} \frac{16^{d-i}}{8i+q}$$

Combien prendre pour N ? Eh bien l'erreur commise est

$$\sum_{i=d+N+1}^{\infty} \frac{16^{d-i}}{8i+q} \leq \sum_{i=d+N+1}^{\infty} 16^{d-i} = \frac{1}{16^{N+1}} \sum_{k=0}^{\infty} \frac{1}{16^k} = \frac{1}{15 \times 16^N}$$

Si l'on veut une erreur inférieure à 10^{-16}, valeur raisonnable pour des calculs sur des flottants, il suffit donc de choisir N tel que $15 \times 16^N \geq 10^{16}$. Passons au logarithme décimal :

$$N \log 16 + \log 15 \geq 16$$

c'est à dire

$$N \geq \frac{16 - \log 15}{\log 16}$$

```
In [13]: (16 - math.log(15, 10)) / math.log(16, 10)
```

```
Out[13]: 12.310989730647321
```

Soyons généreux et prenons 15 termes pour le calcul de la somme en question.

```
In [14]: def somme2(d, q):
             s = 0
             for i in range(d + 1, d + 16):
                 s = frac(s + 16 ** (d - i) / (8 * i + q))
             return s
```

```
In [22]: somme2(10 ** 5, 1)
```

```
Out[22]: 8.333234029020411e-08
```

1.2.4 2.4 BBP, enfin

Tout a été fait, il n'y a plus qu'à additionner. La fonction `somme`, tout d'abord, renvoie $S_{dq} + S'_{dq}$.

```
In [16]: def somme(d, q):
             return somme1(d, q) + somme2(d, q)
```

```
In [18]: somme(10 ** 5, 1)
```

```
Out[18]: 0.27179631151606654
```

La fonction BBP renvoie une valeur approchée de $(16^d \pi)$ mod 1.

```
In [19]: def BBP(d):
             s = frac(4 * somme(d, 1) - 2 *  somme(d, 4) - somme(d, 5) - somme(d, 6))
             return s
```

```
In [20]: BBP(10 ** 5 - 1)
```

```
Out[20]: 0.32566269772397405
```

1.3 3. Fiabilité des résultats

1.3.1 3.1 Comptons les opérations

La valeur renvoyée par BBP est-elle correcte ? Après tout nous faisons des calculs en virgule flottante ! Python utilise pour ces calculs le format IEE754, qui garantit une erreur d'au plus 1 bit sur le dernier chiffre pour chaque opération. Combien avons nous effectué d'opérations en virgule flottante ?

Nous effectuons 4 sommes de d termes. Pour chaque terme on fait une division. Cela nous donne donc $8d - 4$ opérations : $4d$ divisions et $4(d-1)$ additions.

Nous effectuons également 4 sommes de 15 termes : pour chaque terme il y a un calacul de puissance en flottants et une division. Au total $8 \times 15 + 56 = 176$ opérations en virgule flottante.

Au total, nous faisons donc $8d + 176$ opérations en virgule flottante. Rajoutons quelques additions (il y a 8 sommes à ajouter), une multiplication par 4 et une multiplication par 2. Nous arrivons $8d + 185$ opérations (en ai-je oublié ?).

Imaginons qu'à chaque opération l'erreur de 1 bit s'ajoute aux précédentes. Ceci est très pessimiste ! Nous aurons à la fin de nos calcul un nombre de bits erronés de l'ordre de $\log_2(8d + 185)$.

```
In [23]: def bits_erreur(d):
             return math.log(8 * d + 185, 2)

In [24]: bits_erreur(10 ** 6)

Out[24]: 22.93160193126125
```

Python calcule sur 52 bits. Combien de bits exacts de façon sûre ?

```
In [25]: def bits_exacts(d):
             return 52 - bits_erreur(d)

In [26]: bits_exacts(10 ** 6)

Out[26]: 29.06839806873875
```

Sachant qu'un chiffre en hexadécimal est constitué de 4 bits, combien de chiffres hexadécimaux exacts ?

```
In [27]: def hexa_exacts(d):
             return math.floor(bits_exacts(d)) // 4

In [28]: hexa_exacts(10 ** 6)

Out[28]: 7
```

En lançant `BBP(10 ** 6 - 1)` nous aurons donc de façon sûre les chiffres 1000000 à 1000006 du nombre π.

1.3.2 3.2 Convertissons en hexadécimal

La fonction `vers_hexa` prend un réel $x \in [0,1[$ en paramètre et renvoie une chaîne de caractères qui contient les dix premiers chiffres du développement hexadécimal de x. Lisez la fonction, elle ne pose aucune difficulté.

```
In [29]: def vers_hexa(x):
             s = ''
             for k in range(10):
                 y = 16 * x
                 c = math.floor(y)
                 s = s + chiffre_hexa(c)
                 x = frac(y)
             return s
```

La fonction `vers_hexa` utilise `chiffre_hexa` qui prend en paramètre un entier c entre 0 et 15 et renvoie le "chiffre" hexadécimal correspondant.

```
In [30]: def chiffre_hexa(c):
             if c <= 9: return str(c)
             elif c == 10: return 'A'
             elif c == 11: return 'B'
             elif c == 12: return 'C'
             elif c == 13: return 'D'
             elif c == 14: return 'E'
             elif c == 15: return 'F'
```

1.3.3 3.3 Nous y sommes

3.3.1 Cent mille Quelle est le cent-millième chiffre hexadécimal de π ?

```
In [31]: vers_hexa(BBP(10 ** 5 - 1))
```

```
Out[31]: '535EA16C40'
```

C'est un 5. Et les chiffres suivants ? Sont-ils corrects ?

```
In [32]: hexa_exacts(10 ** 5)
```

```
Out[32]: 8
```

On peut donc garantir les 8 chiffres `535EA16C`.

3.3.1 Un million Quelle est le millionième chiffre hexadécimal de π ? Soyez patients, environ une minute de calcul sur ma machine. Python n'est pas fort en boucles `for`, et on ne fait que des boucles `for` :-).

```
In [33]: vers_hexa(BBP(10 ** 6 - 1))
```

```
Out[33]: '26C65E52CB'
```

C'est un 2. Et les chiffres suivants ? Combien sont corrects ?

```
In [34]: hexa_exacts(10 ** 6)
```

```
Out[34]: 7
```

On peut donc garantir les 7 chiffres `26C65E5`. Au cas où vous vous poseriez la question, il s'avère que les chiffres `2CB` sont également corrects (voir l'article BBP). Mais on ne peut pas les **garantir** avec cet algorithme.

3.3.1 Dix millions Quelle est le dix-millionième chiffre hexadécimal de π ? Soyez infiniment patients. Allez prendre un petit café, cela va prendre 10 fois plus de temps que pour un million (donc, dix minutes environ).

```
In [35]: vers_hexa(BBP(10 ** 7 - 1))
```

```
Out[35]: '17AF5863F0'
```

C'est un 1. Et les chiffres suivants ? Sont-ils corrects ?

```
In [36]: hexa_exacts(10 ** 7)
```

```
Out[36]: 6
```

On peut donc garantir les 6 chiffres `17AF58`. Il s'avère que les chiffres `63` sont également corrects (voir l'article BBP). Les deux suivants sont `EF` alors que notre fonction renvoie les chiffres erronés `F0`.

1.4 4. Et en base 10 ?

Bien évidemment, tout le monde meurt d'envie de savoir si une telle formule existe pour la base 10. Borwein, Borwein et Galway ont montré que ce genre de formule ne peut pas exister. Je resterai vague là-dessus, et me contenterai de citer leur article :

" *Of particular interest, we show that π has no Machin-type BBP arctangent formula when $b \neq 2$. To the best of our knowledge, when there is no Machin-type BBP formula for a constant then no BBP formula of any form is known for that constant* ".

La référence de l'article est :

J. Borwein, D. Borwein, W. Galway, *Finding and Excluding b-ary Machin-Type Individual Digit Formulae*, Canadian Journal of Mathematics, Vol. 56 (5), 2004, Pages 897-925.

Il convient pour terminer de remarquer que des algorithmes permettant le calcul des décimales individuelles de π existent mais, à l'heure actuelle, ces algorithmes sont moins efficaces que ceux qui consistent à calculer toutes les décimales.