Sur la nième décimale des nombres transcendants
ou
La 10 milliardième décimale (hex) de Pi est 9

Simon Plouffe
Centre for Experimental &
Constructive Mathematics

Simon Fraser University
BC, CANADA

Avec la collaboration de Peter Borwein (SFU)
et David H. Bailey (Ames Research Center, NASA)

Résumé

0) Introduction aux nombres Tr.
1) Le calcul de 1/n est facile
2) On peut étendre aux polylogarithmes.
3) Il existe des identités pour Pi...
4) la complexité de l'algorithme est n*log(n)
5) Questions ouvertes

Que sait-on des nombres transcendants ?

C'est la classe des nombres NON algébriques.
(pas solution d'équations algébriques à coeff. rat.)
Presque tous les réels sont Tr.

Une brève histoire...

1844 Liouville :

Ce nombre est Tr. et il est très près des rationnels.

$$\sum_{n=0}^{\infty} \frac{1}{10^{n!}} = 0.110001000000000000000001...$$

1873 Hermite : E est Tr.
1882 Lindemann : π est Tr.
1934 Gelfond :

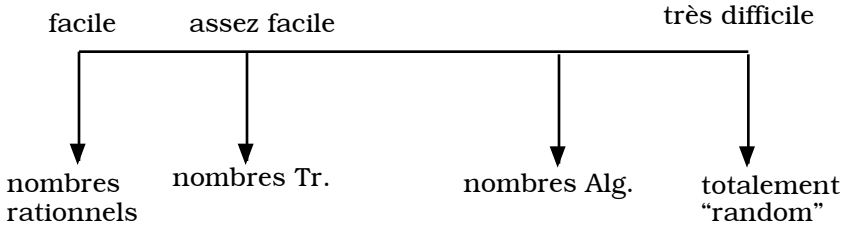
a^b est Tr. si a est algébrique $\neq 0,1$ et b algébrique irrationnel. --> $2^{\sqrt{2}}$ est Tr.

Depuis, quelques résultats éparses, sin(1), J0(1), log(π), $\Gamma(1/4)$: Chudnovsky (récent).

La théorie des nombres Tr. : difficile et peu de résultats : $e^{\sqrt{2}}$ est irrationnel, A. Froda. γ est ?

On pourrait se faire le diagramme suivant...

Les Réels



Cette image pourrait cependant être fausse puisqu'il pourrait exister des exemples (très difficiles) qui seraient Tr.

De façon générale (mis à part les nombres artificiellement construits).

Irrationnels ---> travail (cpu + mémoire) = digits.

+ de travail = + de décimales…

Calcul de Pi à 6.4 milliards de décimales…

==========
Our latest record was established as the followings;

Declared record:
 6,442,450,000 decimal digits
 Two independent calculation based on two different algorithms generated 6,442,450,944 (=3*2^31) decimal digits of pi and comparison of two generated sequences matched 6,442,450,938 decimal digits, e.g., 6 decimal digits difference. Then we are declaring 6,442,450,000 decimal digits as the new world record.

Main program run:
 Job start : 19th September 1995 20:54
 Job end : 24th September 1995 17:32
 Elapsed time : 116:38:12
 Vector CPU : 112:36:06
 Main memory : 1856.75 MB
 ES memory : 32764 MB
 Algorithm : Borwein's 4-th order convergent algorithm

Verification program run:
 Job start : 06th October 1995 09:58
 Job end : 11th October 1995 21:38
 Elapsed time : 131:40:24
 Vector CPU : 135:45:52
 Main memory : 1792.75 MB
 ES memory : 32328 MB
 Algorithm : Gauss-Legendre algorithm

Programs were written by Mr. Daisuke TAKAHASHI, a member of Kanada Lab.
CPU used was HITAC S-3800/480 at the Computer Centre, University of Tokyo.
Two CPU were definitely used through single job parallel processing for total of four programs run.

Yasumasa KANADA
Computer Centre, University of Tokyo
Bunkyo-ku Yayoi 2-11-16
Tokyo 113 Japan
Fax : +81-3-3814-7231 (office)
E-mail: kanada@pi.cc.u-tokyo.ac.jp
==========

Donc,...

E= MC^2 = mémoire * CPU^2 ... haha.

Il est donc normal de penser que la notion de distance (du point décimal) va avec la notion de travail.
---------- FAUX.

On peut calculer la nième décimale (bin ou hex) de π avec peu de travail, sans calculer les (n-1) précédentes et avec virtuellement aucune mémoire.

Comment ? d'abord 3 observations.

1) Le calcul de $1/n$ est très facile.

2) On peut étendre aux logarithmes, arctg et polylogarithmes.

3) Il existe de belles formules pour π, $\log(2)$, $\log(2)^2$, π^2 ,$\arctg(1/2)$...

La magie réside dans le calcul de 1/n.

Tout le monde connait la longue division, un ordinateur effectue le calcul de la même façon.

1,ØØØØØØØØØØØØØØØØØØ | 17
0,0588235294117647...

Ce calcul s'effectue avec 1 décimale à la fois.

Très très vieil algorithme... en se bornant à la base 10.
La k'ième décimale de 1/n est donnée par la solution de

$$10^k \equiv r \bmod n$$

les décimales sont avec le calcul de r/n à partir du rang (k+1). Plus exactement la k'ième est [10*r/n].

En d'autres mots,

$$\left\lfloor \frac{10^k}{n} \right\rfloor = \left\lfloor \frac{r}{n} \right\rfloor$$

EST la même chose. Donc en base b on a,

$$b^k \equiv r \bmod n$$

Mais où est le problème ?

Le problème est lorsque K est grand ou arbitraire.
En utilisant l'exponentiation rapide (Knuth, ACP #2),
vieux truc datant de 200 BC appelé aussi The Binary Method.
Par exemple on veut les décimales de 1/257 à partir du rang 1000. Alors il faut calculer,

$$10^{999} = ? \bmod 257$$

Il suffit d'évaluer,

$10^0, 10^1, 10^2, 10^3, 10^6, 10^7, 10^{14}, 10^{15}, 10^{30}, 10^{31},$
$10^{62}, 10^{124}, 10^{248}, 10^{249}, 10^{498}, 10^{499}, 10^{998}$ et enfin $10^{999} \bmod 257 = 96$. Donc $96/257 = 0{,}373540$
sont les décimales à partir de 1000. La 1000ème est 3.

En sachant que, si on a (analyse numérique I)

$$b^m \equiv r \bmod n$$

alors

$$(b^m)^2 \equiv r^2 \bmod n$$

Donc en multipliant par b et en mettant au carré successivement on arrive **en peu d'étapes** à 10^{999}.

En fait le développement binaire de 999 nous dit quand il faut effectuer l'une et/ou l'autre opération, d'où le nom de la méthode.

2 remarques importantes.
1) Pour un k donné on peut sauver des étapes, ici (1.5*log(k)) ---> (1.29*log(k)) en choisissant de façon judicieuse les fois où il faut mettre au carré plutôt que multiplier par b. Voir Knuth, Brlek : Addition Chains.

2) En utilisant le petit théorème de Fermat, favorable si k >>n mais pas beaucoup si k~n. (th. des nombres).
Sauve 30 % du calcul en moyenne MAIS codage + compliqué.
Il faut se rendre compte que le calcul de cet algorithme est TRÈS serré, en BASIC le programme a 432 caractères de longueur (!). Donc toute addition de code ralentit la boucle principale.

La prochaine étape était de remarquer qu'il existe de belles séries qui sont simples et élégantes. Par exemple,

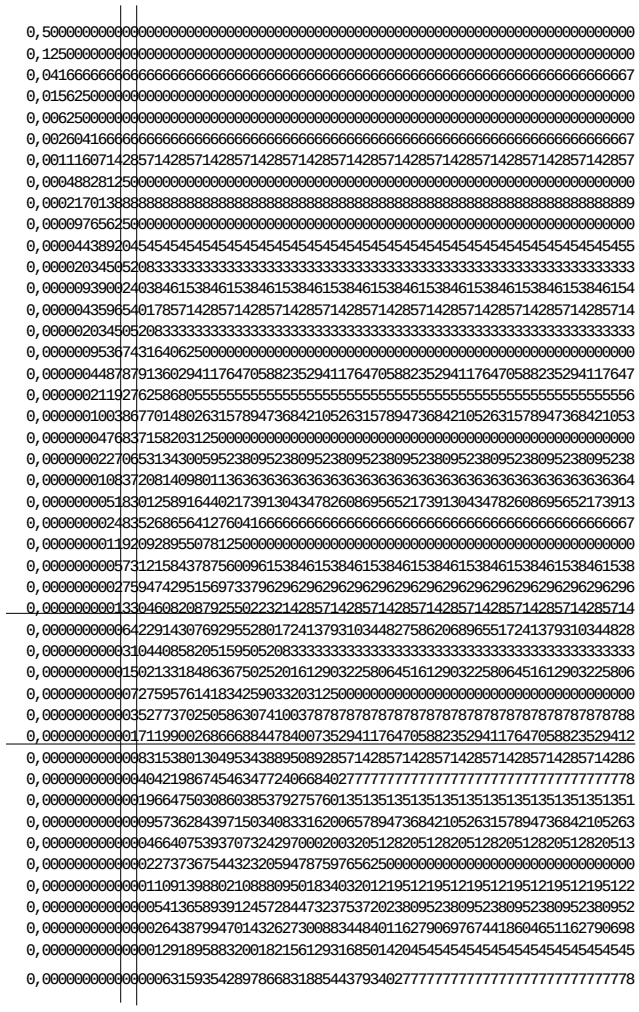
$$\sum_{n=1}^{\infty} \frac{1}{n2^n} = -\log(1/2) = \log(2).$$

Cette série se prête particulièrement bien au calcul puisque le terme 2^n n'est qu'un **DÉPLACEMENT** vers la droite du point 'décimal', il n'y a essentiellement que 1/n à calculer.

Si on veut la 10^6 position en base 2, on tasse de 1 position à chaque étape + le calcul de 1/n. Il suffira de convertir n en binaire, on peut effectuer la conversion sans vraiment calculer ce développement.

La série converge si on pose $k=10^6$, oui, 1/(n*2**n) est aussi petit qu'on veut.

Visuellement, on a le triangle de nombres suivants à additionner.



La partie à gauche de | est inutile puisqu'on peut sauter directement à la position k. Il n'y a qu'une seule colonne à additionner de largeur (1 ou 2 mots). Un calcul final qund m>k permet d'avoir quelques décimales (binaires).

Donc on peut calculer en base b, si on peut représenter le nombre par une série dans cette base (évident).

p.e. $\sum_{n=1}^{\infty} \frac{1}{n4^n} = -\log(3/4) = \log(3) - 2\log(2)$ permet de calculer log(3) en base 2, ... --> log(5), log(7),...

avec des variantes de séries connues pour le log.

Comme, $\log(1-x), \log\left(\frac{1-x}{1+x}\right)$ permet d'avoir la plupart des petits entiers. Mais pas TOUS, log(23) est impossible à obtenir de cette façon, puisque 2047=23*89, on peut l'avoir avec 89 mais pas isolément.

Aussi, trivialement arctg(1/2) est calculable en base 2.

Alors ??!!, π ?, on a (Euler),

$$\frac{\pi}{4} = arctg(\frac{1}{2}) + arctg(\frac{1}{3})$$

en base (2+3) ?, en base (2*3) ?, en base 2/3 ?...

Does not compute !...

Quelle est cette classe de nombres en fait ?

C'est la classe des nombres qui sont calculables en TEMPS : polynomial et en ESPACE : log-polynomial. On l'appelle la classe **SC** (Steven Class), de Steven Cook, mentionné dans Knuth. Cet 'espace' est limité, on ne peut pas utiliser les moyens techniques modernes de calcul comme la FFT (produit de grands nombres), on ne sait même pas si on peut convertir un nombre d'une base à l'autre. La conversion de base est un problème qui consomme du temps et de l'espace...

Il suffit d'avoir un mot de largeur et de descendre. la colonne + le dernier triangle pour avoir suffisamment de précision (largeur du mot). On arrête lorsque m=k.

Si on se restreint à n avec quelques mots mémoires de largeur (mise au carré) on peut donc calculer avec une grande précision (en base b), les nombres,

$$\sum_{n=1}^{\infty} \frac{1}{p(n)b^n}$$

et (conséquemment) on peut étendre à d'autres bases comme,

$$\sum_{n=1}^{\infty} \frac{1}{n10^n} = \log\left(\frac{9}{10}\right)$$

qui est 2log(3)-log(2)-log(5).

Aussi lorsque $b=10^{96}$ à la 5,000,000,000 ième décimale (SFU) et log(9/10) à la 100,000,000 ième décimale.

On peut faire les arctg(x), en base b=x, comme arctg(1/2) en base 2. (ok, ok).

Finalement les polylogarithmes, en s'inpirant des belles identités du livre de Lewin. Oui mais pas directes. Li2(1/2) s'exprime avec log(2)*π et π^2.

Tout de même, avec arctg et un peu de finesse, on a,

$$\frac{\pi}{2} = 2arctg(1/\sqrt{2}) + arctg(1/\sqrt{8})$$

en envoyant Mr. √2 de l'autre còté, on trouve facilement,

$$\pi\sqrt{2} = 4f(1/2) + f(1/8)$$

avec

$$f(x) = \sum_{i=0}^{\infty} \frac{(-1)^i x^i}{2i+1}$$

D'autres comme ça ? oui avec π√3 en base 2.

$$-9f(1/8) = \pi\sqrt{3} + 3\log(3)$$

avec

$$f(x) = \sum_{i=0}^{\infty} \frac{(-1)^i x^i}{3i+1}$$

On revient encore avec le livre de Lewin, quelques autres avec des combinaisons de $\zeta(3)$, $\pi^2 \log(2)$ et $\log(2)^3$, mais aucune $\zeta(3)$ avec pris isolément.

Il ne reste alors qu'à utiliser PSLQ de Bailey la version américaine de l'algorithme LLL.

On se calme… ! Tout d'abord on a les séries du type,

les arctg avec $\sum_{n=0}^{\infty} \frac{(-1)^n x^n}{2n+1}$, les logs avec $\sum_{n=1}^{\infty} \frac{x^n}{n}$ ou des variantes de ces séries.

Une partie des difficultés tient au fait que Maple a beaucoup de difficultés à manipuler les logs avec des valeurs complexes.

On a ceci,

si c'est (n) ----> logarithmes.
si c'est (2n+1) ----> π√2.
si c'est (+/-)(2n+1) ----> arctan.
si c'est 3n+1 ----> π√3.
et pour 4n+1 ----> **surprise**.

Toutes ces formules sont en fait sont des formes variées de **LOGARITHMES** seulement. Après tout π est un log !, c'est le log(-1)/i.Arctg(x) est un log également,…

On a les identités suivantes en fouillant systématiquement le Dilogarithme,

$$\pi^2 = 36L_2(1/2) - 36L_2(1/4) - 12L_2(1/8) + 6L_2(1/64)$$

et aussi,

$$\log(2)^2 = 4L_2(1/2) - 6L_2(1/4) - 2L_2(1/8) + L_2(1/64)$$

Mais qui encore une fois n'apparaissent pas directement, seulement ave l'équation fonctionnelle une fois travaillée.

Ces mêmes formules se réécrivent,

$$\frac{\pi^2}{36} = \sum_{i=1}^{\infty} \frac{a_i}{2^i i^2}, \text{ avec } a_i = \left[\overline{1,-3,-2,-3,1,0}\right]$$

et aussi ,

$$\log(2)^2 = \sum_{i=1}^{\infty} \frac{b_i}{2^i i^2}, \text{ avec } b_i = \left[\overline{-2,-10,-7,-10,2,-1}\right]$$

On peut remarquer que le calcul de log(2), log(3), log(5) mis sous cette forme permet (en prenant 15 mémoires) de calculer une centaine de logarithmes en MÊME temps.

Toutes les formules obtenues peuvent se mettre sous cette forme, plus pratique pour le calcul.

Finalement après 1 mois à tâtons, le programme PSLQ et 2 secondes de temps machine à 00h29 le 19 septembre 1995...

$$\pi = 4\,{}_2F_1\left(\begin{matrix}1, 1/4\\ 5/4\end{matrix}; -\frac{1}{4}\right) + 2\,arctg(1/2) - \log(5)$$

En collectant les termes, le premier est une bissection de l'arctg, ... on a,

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

Avec 4 termes proportionnels à 1/n et un déplacement de 4 positions à chaque n (binaire).

On a donc lancé un SGI Power Challenge, R8000 pour 48 heures et obtenu la 40,000,000,000ième décimale (binaire) de π qui est 1, suivi de 0,0,1,0,0,1,0,... une trentaine de bits.

IL y a plusieurs moyens qui auraient pu arriver à la même formule, par exemple,

$$\pi = \int_0^1 \frac{16x - 16}{x^4 - 2x^3 + 4x - 4}\,dx$$

Maple arrive à trouver π avec cette dernière mais pas celle sous forme hypergéométrique, FAUX selon Maple. On a trouvé au moins une autre forme resemblante et 2 autres pour π*π et log(2)**3.

Tout ceci soulève d'intéressantes questions et remarques...

1) On peut rejoindre la 10^{15} position binaire de π au moins avec cet algorithme, peut-être plus, un CRAY YMP effectue 29,5 milliards d'opérations/seconde sur 64 bits.

2) L'algorithme est parallélisable et c'en est même embarassant (Rob Corless).

3) La distance du point 0. à la nième décimale n'est pas ce que l'on croyait, la 57ème position est plus loin que la 512ème.

4) π est-il normal en base 2 ?, a-t-on les moyens de le prouver maintenant ?

5) Une formule existe-t-elle pour π en base 10 ?

6) Y-a-t-il un motif dans le développement binaire de log(2) ?

7) Si (en plus de l'algorithme) on applique les outils modernes, FFT et calcul parallèle jusqu'où peut-on se rendre ?, 10**20 ?,

8) 2 erreurs 'graves' ont été détectées dans les machines IBM 590 et R8000 en calculannt π...

Daniel Shanks a dit en 1962 (π à 100265 décimales),
“We will NEVER reach the billion’th digit of π”

Borwein & Borwein (1988) :

“We will never reach the 10**1000 digit of π”

Spock , Star Trek, 1968 ...

“Computer !, compute π to the LAST digit”.

Références

LLL et PSLQ
1) Faire ?lattice dans une session Maple
2) Chercher les noms Ferguson-Forcade,
Helaman Ferguson et David H. Bailey.

Aussi : MPFUN, PSLQ dans LYCOS

http://www.nas.nasa.gov/NAS/TechReports/RNRreports/dbailey/RNR-91-032/RNR-91-032.html

article :

http://www.cecm.sfu.ca/personal/pborwein/PISTUFF/Apistuff.html

http://www.mathsoft.com/asolve/plouffe/plouffe.html

The Globe and Mail, 18 octobre 1995, pp. A1-A5.
Science News : récent.

Simon Plouffe:
http://www.cecm.sfu.ca/personal/~plouffe

Peter B. Borwein :
http://www.cecm.sfu.ca/personal/~pborwein

David H. Bailey :
http://www.nas.nasa.gov/