PI, GAMMA, E ET D'AUTRES ... À TRÈS GRANDE VITESSE

SUJET PROPOSÉ PAR BRUNO SALVY
VERSION 1.0
DIFFICULTÉ : MOYENNE

Résumé. Les records récents de calculs de décimales de nombreuses constantes utilisent de belles formules mathématiques, mais qui ne seraient rien sans une panoplie algorithmique efficace (et élégante). L'objectif du projet est d'implanter et comparer certains de ces algorithmes en Java, en se fondant sur des bibliothèques existantes pour l'addition et la multiplication rapide de grands entiers.

Pour calculer quelques millions (voire milliards) de décimales de constantes classiques, l'approche générale qui a été utilisée dans les records récents part d'une représentation en série convergeant rapidement. Par exemple, les records de calculs de π utilisent la formule suivante, découverte en 1989 par les frères Chudnovsky :

$$\frac{1}{\pi} = \frac{12}{C^{3/2}} \sum_{n=0}^{\infty} \frac{(-1)^n (6n)!(A+nB)}{(3n)!n!^3 C^{3n}}$$

où $A = 13591409$, $B = 545140134$ et $C = 640320$. Chaque terme améliore d'environ 14 décimales la précision du résultat.

Pour évaluer efficacement cette formule, il faut savoir calculer efficacement l'évaluation de la série, la racine carrée, l'inverse et le produit. Le coeur de tous ces calculs est fourni par une bonne multiplication rapide. La classe `java.math.BigInteger` permet de manipuler des entiers de taille arbitraire, mais l'algorithme utilisé pour la multiplication est la méthode naïve en complexité quadratique, qui ne permet pas de calculer efficacement au-delà de quelques centaines de décimales. Une bonne implantation d'algorithmes de multiplication rapide constituerait un projet en soi, et ne fait donc pas partie de ce projet, qui reposera sur une version plus moderne de `java.math.BigInteger` [1].

Les méthodes détaillées dans ce sujet sont d'application bien plus vaste que le simple calcul de π. Ainsi, le programme que vous allez écrire pourra calculer efficacement des millions de décimales non seulement de π, mais aussi d'autres constantes un peu plus faciles à calculer comme

- $\ln 2$;
- $e = \exp(1)$ ou ses puissances rationnelles ;
- (plus dur) $\exp(x)$ pour x réel arbitraire ;
- la constante d'Euler γ ;
- la valeur de la fonction ζ de Riemann en 3 ;
- et bien d'autres que vous pourrez imaginer.

Le programme que vous allez écrire demande une précision N à l'utilisateur, propose un choix de constantes à calculer, et écrit les N premières décimales de la constante choisie (en base 10) dans un fichier. Le choix de constantes proposé devra contenir au minimum π, $\exp(p/q)$, $\sqrt{p/q}$ où p et q sont des entiers positifs de petite taille (c'est-à-dite des `int`) demandés à l'utilisateur. La valeur $N = 1\,000\,000$ ne doit pas poser de problème. Lors de la mise au point, vous aurez intérêt à utiliser un système de calcul formel comme Mathematica ou Sage (qui est libre) pour vérifier vos résultats. Vérifiez aussi vos temps de calcul en séparant le calcul lui-même de la conversion finale en base 10.

1. On peut la trouver à l'url https://github.com/tbuktu/bigint.git.

1. Inverse et racine carrée

Les itérations de Newton

$$u_{n+1} = u_n + u_n(1 - au_n) \quad \text{et} \quad v_{n+1} = v_n + \frac{a - v_n^2}{v_n}$$

convergent quadratiquement vers $1/a$ (pour u_n) et $\sqrt{a}$ (pour v_n), pourvu que le point initial ne soit pas trop loin de la valeur cherchée.

L'inconvénient de la seconde itération est qu'elle demande une division à chaque étape. Il peut être plus efficace d'utiliser plutôt

$$w_{n+1} = w_n + \frac{1}{2} w_n(1 - aw_n^2)$$

qui converge vers $1/\sqrt{a}$, et de n'effectuer qu'un seul calcul d'inverse à la fin.

Dans tous les cas, cette itération a une propriété très intéressante : asymptotiquement, non seulement la précision double à chaque itération, mais en plus, les calculs de la k^{e} itération peuvent être effectués à la précision qui y sera atteinte par le résultat.

Implantation. L'implantation sera donc récursive vis-à-vis de la précision, en la divisant par 2 à chaque étape. Pour le point initial, on peut utiliser une évaluation de la fonction en petite précision (un `double` suffit).

2. Calcul rapide de $n!$

Même avec une multiplication rapide, le calcul de

$$n! = 1 \times 2 \times \cdots \times n$$

n'est pas immédiat. Cette formule est efficace pour de petites valeurs de n, mais lorsque n devient grand, l'utilisation naïve de cette formule mène à une complexité élevée. En effet, pour bien exploiter la multiplication rapide, il faut essayer de multiplier des entiers de tailles similaires. On obtient donc un algorithme asymptotiquement efficace par une technique de diviser pour régner, en écrivant

$$n! = (1 \times 2 \times \cdots \times \lfloor n/2 \rfloor) \times ((\lfloor n/2 \rfloor + 1) \times \cdots \times n),$$

(où $\lfloor x \rfloor$ désigne le plus grand entier inférieur ou égal à x), en calculant les deux produits récursivement avec la même idée et en les multipliant à la fin. Cette méthode porte le nom de *scindage binaire*. Asymptotiquement, le coût du calcul $n!$ par cette méthode est borné par celui de $\log n$ multiplications d'entiers de la taille du résultat.

Implantation. L'implantation est à peu près immédiate. En dessous d'un certain seuil à déterminer empiriquement, il vaut mieux arrêter la récursion et repasser à la boucle simple. Vous devez pouvoir observer côte à côte le coût de la multiplication et celui de la factorielle et vérifier l'analyse de complexité, au moins pour des grandes tailles.

3. La constante e

La méthode de scindage binaire utilisée pour le calcul de $n!$ s'étend pour le calcul de séries comme

$$e_n = \sum_{k=0}^{n} \frac{1}{k!}.$$

À nouveau, l'utilisation directe de la formule n'est pas suffisante pour obtenir une vitesse satisfaisante. Le point clé est de réorganiser les calculs par diviser pour régner de sorte à tirer parti de la multiplication rapide des entiers. La suite $(e_n)_{n\geq 0}$ vérifie $e_n - e_{n-1} = 1/n!$ et donc

$$n(e_n - e_{n-1}) = e_{n-1} - e_{n-2}.$$

Cette récurrence se récrit sous forme matricielle

$$\begin{pmatrix} e_n \\ e_{n-1} \end{pmatrix} = \frac{1}{n} \underbrace{\begin{pmatrix} n+1 & -1 \\ n & 0 \end{pmatrix}}_{A(n)} \begin{pmatrix} e_{n-1} \\ e_{n-2} \end{pmatrix} = \frac{1}{n!} A(n)A(n-1)\cdots A(2) \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

Le calcul du produit de matrices peut alors être effectué par scindage binaire.

Comme la suite e_n converge vers e, cette méthode permet le calcul de e à précision 10^{-N} dès que l'on sait où tronquer la série. Cet ordre de troncature se déduit de la preuve classique de convergence : le terme de reste dans $e - e_n$ se majore par une série géométrique de sorte que

$$0 \leq e - e_n < \frac{1}{(n+1)!}\left(1 + \frac{1}{n+1} + \frac{1}{(n+1)^2} + \cdots\right) = \frac{1}{n\,n!}.$$

Pour calculer N décimales de e par cette série, il suffit de rendre $\frac{1}{n\,n!}$ inférieur à 10^{-N}. Il s'ensuit qu'il suffit de prendre n de sorte que $n\,n!$ et 10^N soient du même ordre, c'est-à-dire de prendre $n = O(N/\log N)$ termes dans la série.

Implantation. Comme pour la factorielle, en dessous d'un certain seuil à déterminer, il vaut mieux utiliser une boucle. On pourra aussi essayer de calculer e en le récrivant $\exp(1/2^k)^{2^k}$ en utilisant la méthode ci-dessus pour l'exponentielle (en ajustant les coefficients), qui demande moins de termes de la série pour converger, puis une élévation à la puissance 2^k en k multiplications.

4. L'exponentielle

Le raisonnement utilisé pour le calcul de e s'étend au calcul de $e^{p/q}$ pour p et q entiers quelconques. Le coût augmente cependant avec la taille de p et q. Une technique additionnelle permet de calculer efficacement $\exp(x)$ pour x réel quelconque. D'abord en utilisant répétitivement $\exp(-a) = 1/\exp(a)$ et $\exp(2a) = \exp(a)^2$ on peut se ramener dans l'intervalle $[0, 1]$ et ainsi diminuer le nombre de termes à prendre dans le développement en série. Ensuite, si x a un développement binaire de 2^m bits, alors on peut l'écrire (de manière unique) sous la forme

$$x = \sum_{k=0}^{m} \frac{p_k}{q_k},$$

avec $q^k = 2^{2^k}$ et $0 \leq p_k < 2^{2^{k-1}}$. Le calcul de

$$e^x = \prod_{k=0}^{m} e^{p_k/q_k}$$

est efficace parce que l'augmentation de taille de p_k et q_k est compensée par une diminution du nombres de termes de la série à évaluer.

Implantation. Vous pouvez par exemple tester des nombres comme $\exp(\sqrt{2}/2)$ ou le fameux $\exp(\pi\sqrt{163})$ (quand votre évaluation de π fonctionnera).

5. Sommation de séries hypergéométriques

Les idées utilisées pour l'exponentielle s'étendent. On dit qu'une suite (f_k) est *hypergéométrique* s'il existe une fraction rationnelle R telle que $f_{k+1}/f_k = R(k)$ pour tout $k \in \mathbb{N}$. Par exemple $(1/k!)$ est une suite hypergéométrique, mais aussi la suite des sommants dans la formule des Chudnovsky. Dans ces conditions, comme pour e, le calcul de

$$E_n = \sum_{k=0}^{n} f_k$$

se ramène à deux calculs de scindage binaire : un pour un produit de matrices 2×2 au numérateur, et un pour un produit de valeurs d'un polynôme au dénominateur. Toujours comme pour e, l'utilisation de cette méthode pour une série infinie demande de tronquer la sommation. Ceci peut être fait en majorant la queue de la série par une série géométrique.

À ce stade, vous avez tous les éléments pour calculer π efficacement. Les autres exemples d'application potentielle de cette méthode sont très nombreux. En voici quelques uns :

$$\operatorname{arctanh} x = \sum_{k=0}^{\infty} \frac{x^{2k+1}}{2k+1}, \qquad |x| < 1,$$

$$\zeta(3) = \frac{1}{64} \sum_{k=0}^{\infty} (205k^2 + 250k + 77) \frac{(-1)^k k!^{10}}{(2k+1)!^5},$$

$$\ln(2) = 18 \operatorname{arctanh} \frac{1}{26} - 2 \operatorname{arctanh} \frac{1}{4801} + 8 \operatorname{arctanh} \frac{1}{8749}.$$

Cette formule pour $\zeta(3)$ date de 1997. Elle converge beaucoup plus vite que la définition $\zeta(s) = \sum 1/n^s$. La formule pour $\ln(2)$ date quant à elle de 2010. C'est une variante de la fameuse formule de Machin, choisie parmi de nombreuses autres pour ses bonnes qualités vis-à-vis de l'évaluation rapide.

Implantation. Il faut éviter la duplication de code entre les évaluations de ces séries, en organisant bien les calculs.

6. La constante d'Euler

La constante d'Euler est classiquement définie par l'équation

$$\gamma = \lim_{n\to\infty} H_n - \log n,$$

où H_n est le nombre harmonique $1 + 1/2 + \cdots + 1/n$. Cette série converge très lentement, l'erreur après n termes étant en $1/n$. L'irrationalité de cette constante reste conjecturée. Voici la formule découverte en 1980 qui est utilisée pour son calcul :

$$\gamma = \frac{A_n}{B_n} - \frac{C_n}{B_n}^2 - \log n + O(e^{-8n}),$$

où

$$A_n = \sum_{k=0}^{\infty} \left(\frac{n^k}{k!}\right)^2 H_k, \quad B_n = \sum_{k=0}^{\infty} \left(\frac{n^k}{k!}\right)^2, \quad C_n = \frac{1}{4n} \sum_{k=0}^{2n} \frac{(2k!)^3}{k!^4 (16n)^{2k}}.$$

Implantation. Pour ne pas avoir de difficulté à évaluer le logarithme, il suffit de prendre pour n une puissance de 2 et d'utiliser la formule de la section précédente pour $\ln(2)$. Les sommations pour A_n et B_n peuvent être tronquées à $5n$. Pour le calcul de A_n, il faut de plus augmenter à 3 la taille des matrices à utiliser pour le scindage binaire.

Références

[1] Amdeberhan, T., and Zeilberger, D. Hypergeometric series acceleration via the WZ method. *Electron. J. Combin. 4*, 2 (1997), Research Paper 3, approx. 4 pp. (electronic). The Wilf Festschrift (Philadelphia, PA, 1996). http://www.combinatorics.org/Volume_4/Abstracts/v4i2r3.html.

[2] Bellard, F. Computation of 2700 billion decimal digits of Pi using a desktop computer. 4th revision. http://bellard.org/pi/pi2700e9/, 2010.

[3] Borwein, J. M., and Borwein, P. B. *Pi and the AGM*. John Wiley, 1987.

[4] Brent, R. P., and McMillan, E. M. Some new algorithms for high-precision computation of Euler's constant. *Math. Comp. 34*, 149 (1980), 305–312. http://maths-people.anu.edu.au/~brent/pub/pub049.html.

[5] Gourdon, X., and Sebah, P. Numbers, constants and computation, 1999. http://numbers.computation.free.fr/Constants/constants.html.

[6] Yee, A. J. y-cruncher - a multi-threaded pi-program, 2014. http://www.numberworld.org/y-cruncher/.