N-th digit computation

Xavier Gourdon and Pascal Sebah
February 12, 2003[1]

Is it possible to compute directly the n-th digit of π without computing all its first n digits ? At first sight, the problem seems of the same complexity. Recently [1], a formula which permits to find directly the n-th bit of π with very little memory and in quasi-linear time was exhibited. In other words, the question has a positive answer in base 2.

1 Computing the n-th bit of π

In [1], David Bailey, Peter Borwein and Simon Plouffe give the following formula

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

and use that to compute the n-th bit of π without computing all its first n bits. More precisely, their method permits to obtain the n-th bit of π in time $O(n \log^3 n)$ and space $O(\log(n))$.

1.1 Principle of the algorithm

For convenience, we call the k-th bit of a number the k-th bit of its fractional part.

Computation of the N-th bit of a number

We make use of the notation

$$\{x\} = x - [x] \qquad \text{(fractionalpartofarealnumberx)}.$$

The basic idea depends on the following easy result :

> *The $N + n$-th bit of a real number α is obtained by computing the n-th bit of the fractional part of $2^N \alpha$.*

Proof : This property is obvious since the binary expansion of α

$$\alpha = \sum_k \frac{\alpha_k}{2^k}, \qquad (\alpha_k \in \{0, 1\})$$

entails

$$2^N \alpha = \sum_k \frac{\alpha_k}{2^{k-N}} = A + B,$$

[1] This pages are from *//numbers.computation.free.fr/Constants/constants.html*

where

$$A = \sum_{k \leq N} \alpha_k 2^{N-k} \qquad \text{and} \qquad B = \sum_{k>N} \frac{\alpha_k}{2^{k-N}} = \sum_{k>0} \frac{\alpha_{N+k}}{2^k}.$$

Thus A is an integer and we have $0 \leq B < 1$. In other words, B is the fractional part of $2^N\alpha$ (which we denote by $\{2^N\alpha\}$), and the k-th bit of B is the $N+k$-th bit of α. •

Thus from a sufficiently accurate value of $\{2^N\alpha\}$, this result permits to know the first term of its binary expansion. We illustrate this simple idea on π. One has

$$2^{12}\pi = 12867.9635\ldots,$$

so that $\{2^{12}\pi\} = 0.9635\ldots$. The 4-digits precision of this fractional part permits to have the first term of its binary expansion

$$0.9635\ldots = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{0}{32} + \frac{1}{64} + \frac{1}{128} + \cdots = [0.1111011\ldots]_{\text{base2}}$$

Thus the 13-th bit of π is 1, the 14-th bit of π is 1, the 17-th bit of π is 0.

Powering

We are thus lead to compute a sufficiently accurate value of the fractional part $\{2^N\pi\}$ of $2^N\pi$. Formula (1) entails that $\{2^N\pi\}$ is the fractional part of the series

$$\pi_N = \sum_{n=0}^{\infty} \left(\left\{ \frac{4 \cdot 2^{N-4n}}{8n+1} \right\} - \left\{ \frac{2 \cdot 2^{N-4n}}{8n+4} \right\} - \left\{ \frac{2^{N-4n}}{8n+5} \right\} - \left\{ \frac{2^{N-4n}}{8n+6} \right\} \right) = A_N + B_N,$$

where A_N and B_N are the summations

$$A_N = \sum_{0 \leq n \leq N/4}, \qquad B_N = \sum_{n>N/4}.$$

The first few digits of B_N are obtained easily. We thus concentrate on A_N. The general term of the series in A_N has the form

$$\left\{ \frac{p \cdot 2^n}{m} \right\}$$

where p, n and m are non negative integers. If $p \cdot 2^n = qm + r$ is the euclidian division of $p \cdot 2^n$ by m, we have $\{p \cdot 2^n/m\} = r/m$. In other words

$$\left\{ \frac{p \cdot 2^n}{m} \right\} = \frac{p \cdot 2^n \pmod m}{m}.$$

The computation of 2^n modulo m is obtained thanks to a *binary powering* method, using only $O(\log(n))$ operations modulo m. It consists in writing n in base 2

$$n = \sum_{k=0}^{K} n_k 2^k,$$

so that

$$2^n \pmod m = \prod_{0\le k\le K, n_k=1} 2^{2^k} \pmod m.$$

The values $P_k = (2^{2^k} \pmod m)$ are computed by successive squaring modulo m. Since $K \simeq \log_2(n)$, only $O(\log(n))$ operations modulo m are finally needed to compute $2^n \pmod m$.

Final result

Finally, formula (1) permits to obtain bits of π between position $N+1$ and $N+K$ by computing the K first bits of the number

$$\pi_N = A_N + B_N$$

where

$$A_N = \sum_{0\le n\le N/4} \left(\frac{4\cdot 2^{N-4n} \pmod{8n+1}}{8n+1} - \frac{2\cdot 2^{N-4n} \pmod{8n+4}}{8n+4} - \frac{2^{N-4n} \pmod{8n+5}}{8n+5} - \frac{2^{N-4n}}{}\right.$$

and

$$B_N = \sum_{N/4<n\le 2+(N+K)/4} \left(\frac{4}{8n+1} - \frac{2}{8n+4} - \frac{1}{8n+5} - \frac{1}{8n+6} \right) \frac{1}{2^{4n-N}}.$$

The computation must be carried with an *absolute* precision of at least 2^{-K}. Using a binary powering method, this formula easily permits to obtain the given bits with $O(N\log(N))$ operations on numbers of size $\log(N)$.

1.2 Other BBP formulas for π

Other formulas of this type have been found for π. The fastest known is due to Fabrice Bellard and is approximately 43% faster than (1) to compute the n-th bit of π :

$$\pi = \frac{1}{2^6} \sum_{n=0}^{\infty} \frac{(-1)^n}{2^{10n}} \left(-\frac{2^5}{4n+1} - \frac{1}{4n+3} + \frac{2^8}{10n+1} - \frac{2^6}{10n+3} - \frac{2^2}{10n+5} - \frac{2^2}{10n+7} + \frac{1}{10n+9} \right)$$

(see *here* for a proof).

2 Computing of the n-th bit of other constants

In [1], David Bailey, Peter Borwein and Simon Plouffe give similar kind of series to compute directy the n-th bit of the constants $\log(2)$, π^2 and $\log^2(2)$. Later [2], D. J. Broadhurst found other similar series for a wider class of constants, like $\zeta(3)$, $\zeta(5)$, the Catalan constant C, π^3, π^4, $\log^3(2)$, $\log^4(2)$, $\log^5(2)$. More recently, Gery Huvent [4] found other formula of this kind.

Here are a sample of those formulas :

$$\log(2) = \sum_{k>0} \frac{1}{k2^k}$$

$$\pi^2 = \sum_{k\geq 0} \left(\frac{16}{(8k+1)^2} - \frac{16}{(8k+2)^2} - \frac{8}{(8k+3)^2} - \frac{16}{(8k+4)^2} - \frac{4}{(8k+5)^2} - \frac{4}{(8k+6)^2} + \frac{2}{(8k+7)^2} \right) \frac{1}{16^k}$$

$$\begin{aligned} \frac{7}{8}\zeta(3) &= 6\sum_{k\geq 0} \left(\frac{1}{2(8k+1)^3} - \frac{7}{2(8k+2)^3} - \frac{1}{4(8k+3)^3} + \frac{10}{4(8k+4)^3} - \frac{1}{8(8k+5)^3} - \frac{7}{8(8k+6)^3} + \frac{}{16(8} \right. \\ &+ 4\sum_{k\geq 0} \left(\frac{1}{2^3(8k+1)^3} + \frac{1}{2^4(8k+2)^3} - \frac{1}{2^6(8k+3)^3} - \frac{2}{2^7(8k+4)^3} - \frac{1}{2^9(8k+5)^3} + \frac{1}{2^{10}(8k+6)^3} \right. \end{aligned}$$

As for the Catalan constant $G = \sum_{k\geq 0}(-1)^k/(2k+1)^2$, we have the concise form, shown in [4]

$$\begin{aligned} G &= \frac{3}{4}\sum_{k\geq 0} \left(\frac{2}{(4k+1)^2} - \frac{2}{(4k+2)^2} + \frac{1}{(4k+3)^2} \right) \frac{(-1)^k}{4^k} \\ &- \frac{1}{32}\sum_{k\geq 0} \left(\frac{8}{(4k+1)^2}\frac{4}{(4k+2)^2} + \frac{1}{(4k+3)^2} \right) \frac{(-1)^k}{64^k}. \end{aligned}$$

The direct computation of the n-th bit of the classical constants e, $\sqrt{2}$ and γ with a similar complexity remains an open problem.

3 Computing the n-th decimal digit

Unhappily, no formula of the same kind to compute decimal digits of π have been found yet (or more generally, in any base that is not a power of two). In other words, no series of the form

$$\sum_n \frac{P(n)}{Q(n)} \frac{1}{10^n}, \qquad P, Q \text{ polynomials}$$

is known for π. No such series either is known for other classical constant like $\log(2)$, π^2, ...

The first progress in this direction is due to Simon Plouffe in 1997, who found an algorithm with time $O(n^3 \log^3 n)$ and very little memory to compute the n-th digit of π (see the *Plouffe page* for a description of the method). Unhappily, the complexity is so high that his technique does not reasonably permit to reach decimal digits at position n higher than several thousands.

The Simon Plouffe method has been improved by Fabrice Bellard with a $O(n^2)$ algorithm (complexity in terms of elementary operations on numbers of

Index of decimal digit	*pidec* time	F. Bellard program time
5,000	0.96 sec	4.85 sec
10,000	3.13 sec	18.10 sec
20,000	10.34 sec	68.29 sec
40,000	35.96 sec	259.8 sec
100,000	185.1 sec	1520 sec
200,000	628.7 sec	5703 sec
500,000	3525 sec	34730 sec
1,000,000	15869 sec	not ran
2,000,000	42316 sec	not ran
4,000,000	168191 sec	not ran

Figure 1: Comparison of timings on the n-th decimal digit computation, between the pidec program by Xavier Gourdon and Fabrice Bellard program, on a Pentium III 900 Mhz. (Fabrice Bellard program has been compiled using the `long long` option, which is faster).

size $O(\log n)$). The Fabrice Bellard *page* describes the corresponding algorithm, and a *C program* is also available. The millionth decimal digit can be reached with that technique. Nethertheless, even using parallelization, the complexity remains too high to give a practical alternative to techniques in quasi-linear time which computes all the decimal digits.

A new algorithm for the n-th decimal digit computation

Recently, Xavier Gourdon found a better algorithm for the n-th decimal digit computation of π, that runs in time $O(n^2 \log\log n/\log^2 n)$ in terms of elementary operations. The improvement is nearly a factor $\log^2 n$ compared to Fabrice Bellard algorithm. The corresponding algorithm is described in an unpublished paper [3] that can be dowloaded *here* in pdf format.

The program *pidec*

An implementation of the algorithm has been made with the program *pidec*. The *source code* and the windows *executable* are available for download. Notice that this implementation has not been fully optimized for clarity.

Timings comparisons have been made between this program and the *program* of Fabrice Bellard.

Principle of the algorithm

The algorithm relies on a different technique compared to the one of Simon Plouffe and Fabrice Bellard. Our starting point was the arctan formula

$$\frac{\pi}{4} = \arctan(1) = \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1}. \tag{2}$$

In this form, the formula is well suited to n-th decimal digit computation, but its convergence is too slow. This problem is overcomed with the use of a general alternating series acceleration technique, from Cohen, Villegas and Zagier, described in the page *Acceleration of the convergence of series*. Choosing families of polynomials of the form $P_m(x) = (x^M(1-x))^N$ permits to obtain the following result. Setting

$$M = 2\left\lceil \frac{n}{\log^3 n} \right\rceil \quad \text{and} \quad N = \left\lceil (n + n0 + 1)\frac{\log(10)}{\log(2eM)} \right\rceil.$$

the fractional part of $10^n\pi$ is approximated with an error $< 10^{-n0}$ by the fractional part of $B - C$, where

$$B = \sum_{k=0}^{(M+1)N-1} (-1)^k \, \frac{4 \times 10^n \bmod (2k+1)}{2k+1}, \tag{3}$$

$$C = \sum_{k=0}^{N-1} (-1)^k \, \frac{5^{N-2} 10^{n-N+2} \, s_k \bmod (2MN + 2k + 1)}{2MN + 2k + 1}, \qquad s_k = \sum_{j=0}^{k} \binom{N}{j} \tag{4}$$

The problem is thus essentially reduced to computing powers and sum of binomial coefficients modulo small numbers. *Details* are fulfilled in [3].

Generalization for intermediate memory size

On the one hand, we have an algorithm to compute directly the n-th decimal digit of π in nearly quadratic time and using very little memory; on the other hand, computing all the first n digits of π is possible in quasi-linear time and with memory $O(n)$. It is natural to ask if it were possible to find intermediate algorithms, that use a limited amount of memory $O(m)$ (for example memory $O(\sqrt{n})$) and obtain an intermediate cost between linear and quadratic. In fact, the question has a positive answer by the use of formula (**??**) together with the so-called *binary splitting* technique on numbers of size $O(m)$. More precisely, it is possible to obtain an algorithm which uses $O(m)$ memory with running time

$$O\left(\frac{n^2 \log^3 n \, \log\log n}{m \log^2(n/m)}\right).$$

(the value of m should be not too close to 0). For example, it is possible to obtain the n-th decimal digit of π in time $O(n^{3/2} \log n \, \log\log n)$ and memory $O(n^{1/2})$.

This technique uses small memory and it is likely that in the followings years, distributed computations on home computers with algorithms of this kind will be used to increase the reachable decimal digit of π with home computers, even if no quasi-linear complexity technique is found. Thousands of home computers could go higher than super computers ?

Details of this algorithm will added soon in [3].

References

[1] D.H. Bailey, P.B. Borwein and S. Plouffe, *On the Rapid Computation of Various Polylogarithmic Constants*, Mathematics of Computation, (1997), vol. 66, p. 903-913

[2] D. J. Broadhurst, *Polylogarithmic ladders, hypergeometric series and the ten millionth digits of* $\zeta(3)$ *and* $\zeta(5)$, preprint (1998).

[3] X. Gourdon, *Computation of the n-th decimal digit of* π *with low memory*, preprint (2003) *nthdecimaldigit.pdf*

[4] G. Huvent, *Formules BBP*, Preprint (2001)