The 400 billion’th binary digit of Pi is 0

Simon Plouffe
Associate Member of
the
Centre for Experimental &
Constructive Mathematics, Simon Fraser University
and
Technical Associate at
Wolfram Research Inc.

In collaboration with Peter Borwein (SFU)
and David H. Bailey (Ames Research Center, NASA)

Topics

0) Introduction to transcendental numbers.
1) The computation of 1/n is easy.
2) We can extend this to Polylogarithms.
3) There are some identities for Pi.
4) The complexity of the algorithm is nlog(n).
5) Open questions and conclusion.

What do we know about the digits of transcendental numbers or real numbers... :? (not much).

Transcendental numbers are the numbers that are NOT algebraic (like sqrt(2)).
-They are not solutions of a univariate polynomial with integer coefficient.
-**Almost** all real numbers are transcendental.

A little history,

In1844 Liouville showed this construction of a number to be transcendental.

This number is trans. and very near rational numbers.

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

1873 Hermite : exp(1) or E is also trans.
1882 Lindemann : Pi is trans.
In1934 Gelfond came with this general statement.

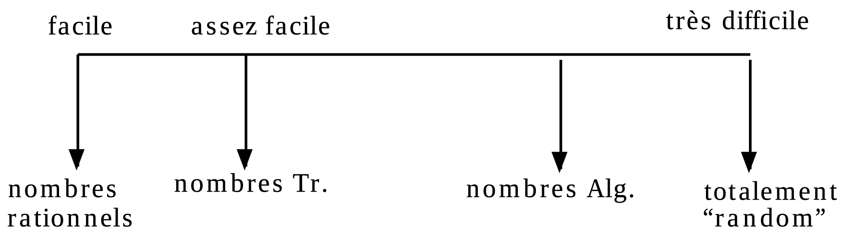
a^b is trans. if a is algebraic not 0 or1 and b algebraic irrational, so 2^sqrt(2) is transcendental, by the way x^(x^x) is 2 when x is sqrt(2) also (27/8)^(9/4) = (9/4)^(27/8). and this number is irrational : (Ramanujan Notebooks, B. Berndt).

Since then, mathematicians have discovered sparse numbers and classes..., sin(1), J0(1), log(Pi), Gamma(1/4) : Chudnovsky.

The theory of transcendental numbers is a difficult thing to study and there are very few results that are not really effective. It takes years to grasp the basics, methods are really deep, etc.

We can do this following diagram about numbers in general.

Les Réels



This diagram could be completely false since there could be transcendental numbers that are very difficult to compute as well. Ordinary intuition about numbers is really misleading.

In a general manner (apart from artificially constructed transcendental numbers).

Irrational numbers mean ---more work (cpu + memory) = digits.

More digits -> More work.

Recently there has been a computation of 2^34 digits of Pi (17.1 billion) and Yasumasa Kanada is preparing 2^35 (coming soon to a theater near you). This amount of digits is impossible to print and needs a fairly large computer just to be on disk! He did it in only 5 hours and 26 minutes (this is quite short compared to the last computation). There is a catch in this : the machine (SR2201) is ranked in the top 5 in the world... and has 1024 processors.

see

http://www.cecm.sfu.ca/projects/ISC/records.html
for the latest records.

not seriously we could say....E= MC^2 = memory * CPU^2 ... haha.

According to all what we know about numbers : this fact is (was) probably true, the more distance you are from the decimal point the more work you have to do. (memory and CPU).

But this is **FALSE**.

We can compute the n’th digit (binary or hexadecimal) of Pi with small work and without having to compute the (n-1) digits before... and this without practically any memory at all.

HOW !?   first, 3 observations

1) The computation of 1/n is easy, very easy.

2) We can extend this to polylogarithms numbers...

3) There are nice formulas that exist for Pi, log(2), log(2)^2 , Pi^2 ,arctan(1/2)...

The magic comes from the computation of 1/n.

Everybody knows what is long division, a computer does it the same way...

1,ØØØØØØØØØØØØØØØØØØØ divided by 17
0,0588235294117647...

This computation is carried with 1 digit at the time.

No need to say that this is very old (at least 2200 years old).

The k’th decimal of 1/n is given by the solution of (we are in base 10 for clarity).

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

the digits are with the computation of r/n from the rank (k+1). More precisely the k’th digit is [10*r/n]. [ ] is the floor function.

In other words,

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

ARE the same thing. So in base b we have,

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

But what is the problem in this ?

The problem is when k is really big or arbitrary large.

By using fast exponentiation (Knuth, The Art of Computer Programming, Vol. 2),
this is an old trick that goes back to 220 BC also called the Binary Method. Well known to computer scientists.

For example, if we want the digits of 1/257 from the rank1000, then we have to compute.

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

We only need to follow these steps (we square and multiply by 10 or we multiply by 10 successively).

$10^0, 10^1, 10^2, 10^3, 10^6, 10^7, 10^{14}, 10^{15}, 10^{30}, 10^{31}, 10^{62}, 10^{124}, 10^{248}, 10^{249}, 10^{498}, 10^{499}, 10^{998}$ and finally 10^{999} mod 257 = 96. So 96/257 = 0,373540... (each time we take the residue).
are the digits from the 1000’th position, the 1000`th is 3.

According to what we know (this is introductory numerical analysis, basic 101 stuff).

$$b^m \equiv r \bmod \text{n}$$

$$\text{alors}$$

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

So by multiplying by b and by squaring successively we arrive in **very few steps** at 10^{999}.

In fact the binary expansion of 999 tells us when to do one or the other operation, this is where the name comes from.

2 important remarks.

1) For a given k, we can save a little from (1.5*log(k)) ---> (1.29*log(k)) by choosing the adequate way to do the squaring and the multiplication by b. Knuth analyzed this and called this : **Addition Chains**.(see Brlek and Knuth).

2) By using the little Fermat theorem we can save a little also, it is favourable when k much larger than n but not much when k is about the size of n (basic number theory). It saves about 30% in average, yes but coding is more complicated then so it might be just the same.

We have to take into account that this algorithm is **very tightly** written, one more step, one more instruction makes it much slower. In BASIC the first program I had was 432 characters long. So any addition of code makes it slower.

The next step is to notice that there are simple and elegant series for natural constants such as

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

This series is particularly well suited since the term 2^n is just a **SHIFT** from the right of the decimal point (in base 2), the computation is only 1/n in fact. But we know how to compute 1/n fast.

If we want the 10^6 position in base 2, we simply shift 1 position àat each step + the computation of 1/n. We would have then only to convert the final result in base 2, in all this we never really fully computed 1/n.

Of course all this is convergent since k=10^6, yes , 1/(n*2**n) is as small as we want. This is obvious, but for the skeptics, it satisfies the usual convergence tests.

Visually we have this big triangle of numbers to add to get log(2).

```
0,50000000000000000000000000000000000000000000000000000000000000000000000000
0,12500000000000000000000000000000000000000000000000000000000000000000000000
0,04166666666666666666666666666666666666666666666666666666666666666666666667
0,01562500000000000000000000000000000000000000000000000000000000000000000000
0,00625000000000000000000000000000000000000000000000000000000000000000000000
0,00260416666666666666666666666666666666666666666666666666666666666666666667
0,00111607142857142857142857142857142857142857142857142857142857142857142857
0,00048828125000000000000000000000000000000000000000000000000000000000000000
0,00021701388888888888888888888888888888888888888888888888888888888888888889
0,00009765625000000000000000000000000000000000000000000000000000000000000000
0,00004438920454545454545454545454545454545454545454545454545454545454545455
0,00002034505208333333333333333333333333333333333333333333333333333333333333
0,00000939002403846153846153846153846153846153846153846153846153846153846154
0,00000435965401785714285714285714285714285714285714285714285714285714285714
0,00000203450520833333333333333333333333333333333333333333333333333333333333
0,00000095367431640625000000000000000000000000000000000000000000000000000000
0,00000044878791360294117647058823529411764705882352941176470588235294117647
0,00000021192762586805555555555555555555555555555555555555555555555555555556
0,00000010038677014802631578947368421052631578947368421052631578947368421053
0,00000004768371582031250000000000000000000000000000000000000000000000000000
0,00000002270653134300595238095238095238095238095238095238095238095238095238
0,00000001083720814098011363636363636363636363636363636363636363636363636364
0,00000000518301258916440217391304347826086956521739130434782608695652173913
0,00000000248352686564127604166666666666666666666666666666666666666666666667
0,00000000119209289550781250000000000000000000000000000000000000000000000000
0,00000000057312158437875600961538461538461538461538461538461538461538461538
0,00000000027594742951569733796296296296296296296296296296296296296296296296
0,00000000013304608208792550223214285714285714285714285714285714285714285714
0,00000000006422914307692955280172413793103448275862068965517241379310344828
0,00000000003104408582051595052083333333333333333333333333333333333333333333
0,00000000001502133184863675025201612903225806451612903225806451612903225806
0,00000000000727595761418342590332031250000000000000000000000000000000000000
0,00000000000352773702505863074100378787878787878787878787878787878787878788
0,00000000000171199002686668844784007352941176470588235294117647058823529412
0,00000000000083153801304953438895089285714285714285714285714285714285714286
0,00000000000040421986745463477240668402777777777777777777777777777777777778
0,00000000000019664750308603853792757601351351351351351351351351351351351351
0,00000000000009573628439715034083316200657894736842105263157894736842105263
0,00000000000004664075393707324297000200320512820512820512820512820512820513
0,00000000000002273736754432320594787597656250000000000000000000000000000000
0,00000000000001109139880210888095018340320121951219512195121951219512195122
0,00000000000000541365893912457284473237537202380952380952380952380952380952
0,00000000000000264387994701432627300883448401162790697674418604651162790698
0,00000000000000129189588320018215612931685014204545454545454545454545454545

0,00000000000000063159354289786683188544379340277777777777777777777777777778
```

The left side of the vertical bars is useless in this, we can jump directly at position k in log(k) steps. There is only one column to add of a width of 1 or 2 memory words. |We can compute that number in base b IF we can represent it with a suitable series (like this one for log(2)).

for example $\sum_{n=1}^{\infty} \frac{1}{n4^n} = -\log(3/4) = \log(3) - 2\log(2)$ is suitable for log(3) in base 2, by simply spliting the log and simple arithmetic we can get log(5), log(7).

Since, $\log(1-x), \log\left(\frac{1-x}{1+x}\right)$ can be used to get the log of small integers.

But not all of them, log(23) is still impossible, it has to do with the factorization of 2047 = 23*89, the factors do not appears separatly. We can have log(2047) but not log(89) or log(23).

Also arctan(1/2) is trivially obtainable in base 2.

But then we have Pi, isn`t?, (not so simple), here we have an identity of Euler but it is with something that needs base 2 and/or base 3.

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

in base (2+3) ?, then ? or in base (2*3) ? perhaps..., in base 2/3 ?..., NO.

Does not compute !...

What class of numbers is this anyway?

It is the class of numbers that can be computed in POLYNOMIAL time and log-polynomial SPACE. We call that class: $\mathbf{SC}_2$, (2 for base 2), as mentionned in Knuth. This space is limited, we can't use FFT algorithms for large multiplication obvisouly and base conversion is (not proved) but suspected to be impossible. We are talking here of high precision computations, which is needed to compute classical constants with classical algorithms. FFT (fast fourier transform) and the Karatsuba method are critically needed to compute numbers with more than a few thousands digits. Naive computation of n*n takes n^2 steps with the `school boy` method and about $n^{1.585}$ if Karatsuba method is used, to convince yourself of this : try to evaluate how many operations are needed to compute n*n when n is 1 billion digits wide, the best computer in the world has a raw speed of 10^{12} operations÷/sec. This means forever for ordinary humans.

So, in this , it suffices to have a word size width and to be able to add a column of numbers. The column of numbers + the last triangle (word width). We top when m=k.

If we restrict ourselves to a few word widths (needed to square) largeur (mise au carré) we then can compute (in base b) the numbers of the form,

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

and consequently this number is fast-computable in base 10 (Now called Bailey-Borwein-Plouffe computable).

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

which is 2log(3)-log(2)-log(5).

For the sake of it (in the beginning of all this), I computed b=10^{96} à the 5,000,000,000`th digit of that number using an ordinary SUN/SPARC at SFU and the 100,000,000`th digit of log(9/10) with another computer in Montreal.(UQAM).

We can do arctan(x), in base b b=x, like arctan(1/2) en base 2. (ok, ok).

Finally most of polylogarithms, we have to dig indentities in the beautiful book of Lewin on polylogarithms. Like Li2(1/2) and such numbers.

With a little inguinity and simple series manipulation we have,

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

by sending Mr Sqrt(2) on the other side, we easily find,

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

with,

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

Others like that? in base 2?, yes,

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

with,

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

We go back then to the Lewin’`s book to get (with LLL) identities for Pi^2 , Pi$^2{}_*$sqrt(2) and others with Zeta(3) and log(2)^3 but none with Zeta(3) alone, unfortunately.

There is one last resort : PSLQ of Bailey, the american version of the LLL algorithm.

Ok, let’`s calm down and write what we have so far. We have the series of the type :

arctan series are $\sum_{n=0}^{\infty} \frac{(-1)^n x^n}{2n+1}$, the logs are of the type $\sum_{n=1}^{\infty} \frac{x^n}{n}$ and variants of this with (-1) or not at the denominator.

Part of the difficulty in this resides in the fact that ALL of the symbolic computation packages are having big trouble dealing with complex values of logarithms (multiples of Pi).

We have this (we know that).

if the denominator is (n) then we have series of logarithms.
if the denominator is (2n+1) then we have series with arctan`s and logs.
if it is (+/-)(2n+1) then it is arctan also.
if it is (3n+1) we have Pi*sqrt(3).
and what about (4n+1)? big **surprise**.

This is not the good model, all those series in fact are all **LOGARITHMS** and only logarithms. After all : Pi is a log!, it is log(-1)/i. Arctan(x) is also a log (in disguise). We have the following identities by scanning systematically the Dilogarithm.

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

and also,

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

But again, all of this is not direct, we had to fool around with functional equations to get them.

Those same formulas can be rewritten as,

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

and,

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

We notice here that the computation of log(2), log(3), log(5) in this manner can all be computed at the same TIME. With 15 memory cells, we can compute about a hundred of them all in once.

All those formulas can be re-written in the same shape, more suitable for computer implementation.

Finally after 1-2 months of trial and errors, tests, sweat and etc, at 0h29 on september 19, 1995 in about 2 seconds of CPU, I had a mix of programs : PSLQ, LLL (Pari-Gp), Maple and Unix scripts all tighted in the same Maple program, I got this.

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

By collecting terms, or if we split the arctan series we can rewrite this in a more comprehensive manner to be,

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

With 4 terms that are proportional to 1/n and a shift of 1 hexadecimal digit for each 4 terms.

The next step was to tight up a SGI Power Challenge, R8000 for 96 hours to obtain the 40,000,000,000`th digit (binairy) of Pi which is 1, followed by 0,0,1,0,0,1,0,... about 30 bits long.

We could had found that same formula in a variety of ways, one of them is,

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

Maple finds this to be valid and once expanded into an hypergeomtric series it makes a mistake in translating an identity and comes out false.

All this creates a series of interesting thoughts and conclusions :

1) We could probably reach the 10^{15} position in binary of Pi with a super-computer, it took only 1 year after that when F. Bellard reached the 400,000,000,000`th position (it is 1).

2) The algorithm is embarassingly parallel (Rob Corless).

3) Binary digits can be computed in any order.

4) Is Pi normal ?, can we prove that log(2) is normal ?

5) Is there a formula for Pi in base 10 using this? (Yes but it takes O(n*n*n) steps). see http://www.cecm.sfu.ca/~plouffe/Simon/articlepi.html and Pour La Science, janvier 1997.

6) Are there any patterns in the binary expansion of log(2) ?

7) Now, if we apply modern tools in addition to the BBP algorithm and a super-computer, can we reach 10^20?

8) 2 big hardware and software flaws have been discovered in IBM 590 et R8000 in the process of computation of Pi.

Daniel Shanks said in 1962 (He computed Pi to 100265 deécimals),
“We will NEVER reach the billion’th digit of Pi.”

Borwein & Borwein (1988) :

“We will never reach the 10**1000 digit of π and probably that the computation of the n’th is as difficult as the computation itself”

Mr. Spock, in Star Trek, 1968 ... (The wolf in the fold).

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

References and URL's

LLL and PSLQ

1) Type ?lattice in a Maple session or LatticeReduce in Mathematica.
2) Go to Altavista and type Ferguson-Forcade, Helaman Ferguson or David H. Bailey. or Simon Plouffe. (hum).

Also : MPFUN, PSLQ will lead to 'something'.

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

articles on the net. :

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

my home page contains a good deal of infos about all this at
http://www.cecm.sfu.ca/~plouffe

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

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/