Computing Bernoulli and Tangent Numbers

Richard P. Brent
MSI, ANU

17/18 May 2011

Dedicated to Jon Borwein
on the occasion of his 60th birthday

Summary

Bernoulli numbers are rational numbers B_n defined by the generating function

$$\sum_{n\geq 0} B_n \frac{z^n}{n!} = \frac{z}{\exp(z) - 1}.$$

They are of interest in number theory and are related to special values of the Riemann zeta function. They also occur as coefficients in the Euler-Maclaurin formula.

The closely related *Tangent numbers* T_n, and *Secant numbers* S_n, defined by

$$\sum_{n>0} T_n \frac{z^{2n-1}}{(2n-1)!} = \tan z, \quad \sum_{n\geq 0} S_n \frac{z^{2n}}{(2n)!} = \sec z,$$

are more convenient for computation because they are integers.

Summary continued

In this talk I will consider some algorithms for computing Bernoulli, Secant and Tangent numbers.

Recently, David Harvey [*Math. Comp.* 2010] showed that the *single* number B_n can be computed in

$$O(n^2(\log n)^{2+o(1)})$$

bit-operations. In fact, the Bernoulli numbers $B_0, \ldots, B_n$ can all be computed with the same complexity bound (and similarly for Secant and Tangent numbers).

Summary continued

I will first give a relatively simple algorithm that achieves the slightly weaker bound $O(n^2(\log n)^{3+o(1)})$. If time permits, I will sketch the improvement to $O(n^2(\log n)^{2+o(1)})$ bit-operations.

I will also give very simple in-place algorithms for computing the first n Secant or Tangent numbers using $O(n^2)$ integer operations. Although they are not the asymptotically fastest algorithms, they are extremely simple and convenient for moderate values of n.

Advertisement

Much of the material for this talk is drawn from my recent book:

Richard P. Brent and Paul Zimmermann,
Modern Computer Arithmetic,
Cambridge University Press, 2010, 237 pp.
(online version available from my website).

In particular, see §4.7.2 and exercises 4.35–4.41.

Equation numbers such as "(4.58)" are as in the book.

Bernoulli numbers

From the generating function

$$\sum_{n\geq 0} B_n \frac{z^n}{n!} = \frac{z}{\exp(z) - 1}$$

it's easy to see that the B_n are rational numbers, $B_{2n+1} = 0$ if $n > 0$, and they satisfy the recurrence

$$B_0 = 1, \ \sum_{j=0}^{k} \binom{k+1}{j} B_j = 0 \text{ for } k > 0. \qquad (4.58)$$

It's sometimes convenient to consider *scaled* Bernoulli numbers $C_n = B_{2n}/(2n)!$, with generating function

$$\sum_{n\geq 0} C_n z^{2n} = \frac{z/2}{\tanh(z/2)}.$$

The Von Staudt – Clausen theorem

Computing a few B_n, we find

$$\begin{array}{llll} B_0 = 1, & B_1 = -1/2, & B_2 = 1/6, & B_4 = -1/30, \\ B_6 = 1/42, & B_8 = -1/30, & B_{10} = 5/66, & B_{12} = -691/2730, \\ B_{14} = 7/6, & \text{etc.} & & \end{array}$$

What are the denominators? This is answered by the *Von Staudt – Clausen Theorem* (1840), which says that

$$B'_{2n} := B_{2n} + \sum_{(p-1)|2n} \frac{1}{p} \in \mathbb{Z},$$

(where the sum is over primes p for which $p - 1$ divides $2n$).

Thus, in a program it might be more convenient to store B'_{2n} than B_{2n}.

Connection with the Riemann zeta-function

Euler found that the Riemann zeta-function for even non-negative integer arguments can be expressed in terms of Bernoulli numbers – the relation is

$$(-1)^{k-1}\frac{B_{2k}}{(2k)!} = \frac{2\zeta(2k)}{(2\pi)^{2k}}.$$

Since $\zeta(2k) = 1 + O(4^{-k})$ as $k \to +\infty$, we see that

$$|B_{2k}| \sim \frac{2\,(2k)!}{(2\pi)^{2k}}.$$

From Stirling's approximation to $(2k)!$ we see that the number of bits in the integer part of B_{2k} is $2k \lg k + O(k)$.

Thus, it takes $\Omega(n^2 \log n)$ space to store $B_1, \ldots, B_n$.

Another connection with the zeta-function

From the functional equation for the Riemann zeta-function, we also have

$$\zeta(-n) = -\left(\frac{B_{n+1}}{n+1}\right)$$

for $n \in \mathbb{Z}$, $n \geq 1$.

Computing Bernoulli numbers

From the generating function $z/(\exp(z) - 1)$ we obtain the recurrence

$$B_0 = 1, \ \sum_{j=0}^{k} \binom{k+1}{j} B_j = 0 \text{ for } k > 0. \qquad (4.58)$$

This recurrence has traditionally been used to compute $B_0, \ldots, B_{2k}$ with $O(k^2)$ arithmetic operations.

This is unsatisfactory if floating-point numbers are used, because the recurrence is *numerically unstable*: the relative error in the computed B_{2k} is of order $4^k 2^{-n}$ if the floating-point arithmetic has a precision of n bits.

A Numerically Stable Recurrence

As before, let $C_k = B_{2k}/(2k)!$. Then

$$\frac{\sinh(z/2)}{z/2} \sum_{k \geq 0} C_k z^{2k} = \cosh(z/2),$$

and equating coefficients gives the recurrence

$$\sum_{j=0}^{k} \frac{C_j}{(2k+1-2j)!\,4^{k-j}} = \frac{1}{(2k)!\,4^k}. \qquad (4.60)$$

Using this recurrence to evaluate $C_0, C_1, \ldots, C_k$, the relative error in the computed C_k is only $O(k^2 2^{-n})$, which is satisfactory from a numerical point of view.

An Asymptotically Fast Algorithm

Harvey (2010) showed how B_n could be computed exactly, using a modular algorithm and the Chinese remainder theorem, in $O(n^2(\log n)^{2+\varepsilon})$ bit-operations. (We write ε for terms that are $o(1)$ as $n \to \infty$.)

We'll show how to compute all of $B_0, \ldots, B_n$ with almost the same complexity bound (only larger by a factor $O(\log n)$).

Digression – Reciprocals of Power Series

Let $A(z) = a_0 + a_1 z + a_2 z^2 + \cdots$ be a power series with coefficients in some field F (e.g. $F = \mathbb{Q}$ or $\mathbb{R}$), with $a_0 \neq 0$. Let $B(z) = b_0 + b_1 z + \cdots$ be the *reciprocal* power series, so $A(z)B(z) = 1$.

Suppose we can multiply polynomials of degree $n - 1$ in $F[z]$ using $M(n) = O(n(\log n)^{1+\varepsilon})$ field operations. Then, using Newton's method [Kung and Sieveking], we can compute $b_0, \ldots, b_{n-1}$ with the *same* complexity $O(n(\log n)^{1+\varepsilon})$, up to a constant factor.

Application – an Asymptotically Fast Algorithm

Taking

$$A(z) = (\exp(z) - 1)/z$$

and working with $(n \lg(n) + O(n))$-bit floating-point numbers, we get $B_0, \ldots, B_n$ to sufficient accuracy to deduce the exact (rational) result using $O(n(\log n)^{1+\varepsilon})$ floating-point operations, each of which can be done with

$$O(n(\log n)^2 \log\log n)$$

bit-operations by Schönhage–Strassen. Thus, overall we get $B_0, \ldots, B_n$ with

$$O(n^2(\log n)^{3+\varepsilon})$$

bit-operations. Similarly for Secant and Tangent numbers.

Tangent and Secant numbers

The *Tangent numbers* T_n $(n > 0)$ (also called *Zag* numbers) are defined by

$$\sum_{n>0} T_n \frac{z^{2n-1}}{(2n-1)!} = \tan z = \frac{\sin z}{\cos z}.$$

Similarly, the *Secant numbers* S_n $(n \geq 0)$ (also called *Euler* or *Zig* numbers) are defined by

$$\sum_{n\geq 0} S_n \frac{z^{2n}}{(2n)!} = \sec z = \frac{1}{\cos z}.$$

Unlike the Bernoulli numbers, the Tangent and Secant numbers are positive integers.

Asymptotics

Because $\tan z$ and $\sec z$ have poles at $z = \pi/2$, we expect T_n to grow roughly like $(2n-1)!(2/\pi)^n$ and S_n like $(2n)!(2/\pi)^n$.

More precisely, let

$$\zeta_0(s) = (1 - 2^{-s})\zeta(s) = 1 + 3^{-s} + 5^{-s} + \cdots$$

be the *odd* zeta-function. Then

$$\frac{T_k}{(2k-1)!} = \frac{2^{2k+1}\zeta_0(2k)}{\pi^{2k}} \sim \frac{2^{2k+1}}{\pi^{2k}}.$$

We also have

$$\frac{S_k}{(2k)!} \sim \frac{2^{2k+2}}{\pi^{2k+1}}.$$

Bernoulli numbers via Tangent numbers

From the formulas for T_k and B_{2k} in terms of $\zeta(2k)$, we see that

$$T_k = (-1)^{k-1} 2^{2k} (2^{2k} - 1) \frac{B_{2k}}{2k}$$

(this can be proved directly, without involving the zeta-function).

Since $T_k \in \mathbb{Z}$, the odd primes in the denominator of B_{2k} must divide $2^{2k} - 1$. This is compatible with the Von Staudt–Clausen theorem, since $(p-1)|2k \implies p|(2^{2k} - 1)$ by Fermat's little theorem.

T_k has about $4k$ more bits than $\lceil B_{2k} \rceil$, but both have $2k \lg k + O(k)$ bits, so asymptotically not much difference.

Thus, to compute B_{2k} we don't lose much by first computing T_k, and this may be more convenient since $T_k \in \mathbb{Z}$, $B_{2k} \in \mathbb{Q}$.

Getting Rid of a "log" Factor in the Time Bound

To improve the algorithm for Bernoulli numbers, we use the Kronecker–Schönhage trick. Here is an outline.

Fix $n > 1$, choose $p = \lceil n \lg(n) \rceil$, $N = 2np$, $z = 2^{-p}$.

Write down N-bit approximations to $(2n)! \sin(z)$ and $(2n)! \cos(z)$ from the truncated Taylor series.

Perform an N-bit division (using Newton's method) to get an N-bit approximation to $\tan(z)$ in time $O(N \log(N) \log\log(N))$.

Multiply by $(2n-1)!$ and read off the integers $T'_k = T_k (2n-1)!/(2k-1)!$ from the binary representation.
Now deduce the T_k and B_{2k}, $k \le n$. The overhead involving factorials can be handled within the overall time bound, which is

$$O(N \log(N) \log\log(N)) = O(n^2 (\log n)^2 \log\log n).$$

Algorithms based on 3-term Recurrences

Akiyama and Tanigawa gave an algorithm for computing Bernoulli numbers based on a 3-term recurrence. However, it is only useful for exact computations, since it is numerically unstable if applied using floating-point arithmetic.

We'll give a stable 3-term recurrence and corresponding in-place algorithm for computing Tangent numbers. It is perfectly stable since all operations are on positive integers and there is no cancellation. Also, it involves less arithmetic than the Akiyama-Tanigawa algorithm.

The three-term recurrence was given by Buckholtz and Knuth (1967), but they did not give the in-place algorithm explicitly.

There is a similar 3-term recurrence and stable in-place algorithm for computing Secant numbers.

A 3-term Recurrence for Computing Tangent Numbers

Write $t = \tan x$, $D = \mathrm{d}/\mathrm{d}x$, so $Dt = 1 + t^2$ and $D(t^n) = nt^{n-1}(1 + t^2)$ for $n \geq 1$.

It is clear that $D^n t$ is a polynomial in t, say $P_n(t)$.

Write $P_n(t) = \sum_{j\geq 0} p_{n,j}t^j$. Then $\deg(P_n) = n + 1$ and, from the formula for $D(t^n)$,

$$p_{n,j} = (j - 1)p_{n-1,j-1} + (j + 1)p_{n-1,j+1}. \qquad (4.63)$$

We are interested in $T_k = p_{2k-1,0}$, which can be computed from the 3-term recurrence in $O(k^2)$ operations using the obvious boundary conditions.

We can save work by noticing that $p_{n,j} = 0$ if $n + j$ is even.

Algorithm TangentNumbers

Input: positive integer n
Output: Tangent numbers $T_1, \ldots, T_n$

$T_1 \leftarrow 1$
for k **from** 2 **to** n
  $T_k \leftarrow (k-1)T_{k-1}$
for k **from** 2 **to** n
  for j **from** k **to** n
    $T_j \leftarrow (j-k)T_{j-1} + (j-k+2)T_j$
return $T_1, T_2, \ldots, T_n$.

The first **for** loop initializes $T_k = p_{k-1,k} = (k-1)!$. The variable T_k is then used to store $p_{k,k-1}$, $p_{k+1,k-2}$, ..., $p_{2k-2,1}$, $p_{2k-1,0}$ at successive iterations of the second **for** loop. Thus, when the algorithm terminates, $T_k = p_{2k-1,0}$ (correct).

Illustration

The process in the case $n = 3$ is illustrated in the following diagram, where $T_k^{(m)}$ denotes the value of the variable T_k at successive iterations $m = 1, 2, \ldots, n$:

$$
\begin{array}{ccccccc}
 & & T_1^{(1)} = p_{0,1} & & & & \\
 & \swarrow & & \searrow & & & \\
T_1^{(1)} = p_{1,0} & & & & T_2^{(1)} = p_{1,2} & & \\
 & \searrow & & \swarrow & & \searrow & \\
 & & T_2^{(2)} = p_{2,1} & & & & T_3^{(1)} = p_{2,3} \\
 & \swarrow & & \searrow & & \swarrow & \\
T_2^{(2)} = p_{3,0} & & & & T_3^{(2)} = p_{3,2} & & \\
 & \searrow & & \swarrow & & & \\
 & & T_3^{(3)} = p_{4,1} & & & & \\
 & \swarrow & & & & & \\
T_3^{(3)} = p_{5,0} & & & & & &
\end{array}
$$

Complexity of Algorithm TangentNumbers

Algorithm TangentNumbers takes $\Theta(n^2)$ operations on positive integers. The integers T_n have $O(n \log n)$ bits, other integers have $O(\log n)$ bits.

Thus the overall complexity is $O(n^3 (\log n)^{1+\varepsilon})$ bit-operations, or $O(n^3 \log n)$ word-operations if n fits in a single word.

The algorithm is not optimal, but it is good in practice for moderate values of n, and much simpler than asymptotically faster algorithms.

Acknowledgements

- Jon Borwein, Kurt Mahler and George Szekeres for encouraging my belief that high-precision computations are useful in "experimental" mathematics.
- David Harvey for discussions on the complexity of algorithms for computing Bernoulli numbers.
- Donald Knuth and Thomas Buckholtz for giving the three-term recurrence (4.63) for Tangent numbers.
- Ben F. "Tex" Logan, Jr., for suggesting the use of Tangent numbers to compute Bernoulli numbers.
- Christian Reinsch for pointing out the numerical instability of the recurrence (4.58) and suggesting the use of the numerically stable recurrence (4.60).
- Paul Zimmermann for coauthoring our book *Modern Computer Arithmetic*.

References

1. Jonathan M. Borwein and David H. Bailey, *Mathematics by Experiment: Plausible Reasoning in the 21st Century*, second edition, A. K. Peters, 2008.

2. Richard P. Brent, Unrestricted algorithms for elementary and special functions, in *Information Processing 80*, North-Holland, Amsterdam, 1980, 613–619. arXiv:1004.3621v1

3. Richard P. Brent and Paul Zimmermann, *Modern Computer Arithmetic*, Cambridge University Press, 2010, 237 pp.

4. Ronald L. Graham, Donald E. Knuth, and Oren Patashnik, *Concrete Mathematics*, third edition, Addison-Wesley, 1994.

5. David Harvey, A multimodular algorithm for computing Bernoulli numbers, *Mathematics of Computation* **79** (2010), 2361–2370.

References continued

6. Masanobu Kaneko, The Akiyama-Tanigawa algorithm for Bernoulli numbers, *Journal of Integer Sequences* **3** (2000). Article 00.2.9, 6 pages.

7. Donald E. Knuth, Euler's constant to 1271 places, *Mathematics of Computation*, 16 (1962), 275–281.

8. Donald E. Knuth and Thomas J. Buckholtz, Computation of Tangent, Euler, and Bernoulli numbers, *Mathematics of Computation* **21** (1967), 663–688.

9. Christian Reinsch, personal communication, about 1979.