Chapter 8
Fast Computation of Bernoulli, Tangent and Secant Numbers

Richard P. Brent and David Harvey

Abstract We consider the computation of Bernoulli, Tangent (zag), and Secant (zig or Euler) numbers. In particular, we give asymptotically fast algorithms for computing the first n such numbers $O(n^2(\log n)^{2+o(1)})$. We also give very short in-place algorithms for computing the first n Tangent or Secant numbers in $O(n^2)$ integer operations. These algorithms are extremely simple and fast for moderate values of n. They are faster and use less space than the algorithms of Atkinson (for Tangent and Secant numbers) and Akiyama and Tanigawa (for Bernoulli numbers).

Key words: Bernoulli number • Tangent number • Secant number • Asymptotically fast algorithm • In-place algorithm

R.P. Brent (✉)
Mathematical Sciences Institute, Australian National University, Canberra, ACT 0200, Australia
e-mail: Tangent@rpbrent.com

D. Harvey
School of Mathematics and Statistics, University of New South Wales, Sydney, NSW 2052, Australia
e-mail: D.Harvey@unsw.edu.au

8.1   Introduction

The *Bernoulli numbers* are rational numbers B_n defined by the generating function

$$\sum_{n\geq 0} B_n \frac{z^n}{n!} = \frac{z}{\exp(z)-1}. \tag{8.1}$$

Bernoulli numbers are of interest in number theory and are related to special values of the Riemann zeta function (see Sect. 8.2). They also occur as coefficients in the Euler–Maclaurin formula, so are relevant to high-precision computation of special functions [7, Sect. 4.5].

It is sometimes convenient to consider *scaled* Bernoulli numbers

$$C_n = \frac{B_{2n}}{(2n)!}, \tag{8.2}$$

with generating function

$$\sum_{n\geq 0} C_n z^{2n} = \frac{z/2}{\tanh(z/2)}. \tag{8.3}$$

The generating functions (8.1) and (8.3) only differ by the single term $B_1 z$, since the other odd terms vanish.

The *Tangent numbers T_n* and *Secant numbers S_n* are defined by

$$\sum_{n>0} T_n \frac{z^{2n-1}}{(2n-1)!} = \tan z, \quad \sum_{n\geq 0} S_n \frac{z^{2n}}{(2n)!} = \sec z. \tag{8.4}$$

In this paper, which is based on an a talk given by the first author at a workshop held to mark Jonathan Borwein's sixtieth birthday, we consider some algorithms for computing Bernoulli, Tangent, and Secant numbers. For background, combinatorial interpretations, and references, see Abramowitz and Stegun [1, Chap. 23] (where the notation differs from ours, e.g. $(-1)^n E_{2n}$ is used for our S_n), and Sloane's [27] sequences A000367, A000182, and A000364.

Let $M(n)$ be the number of bit-operations required for n-bit integer multiplication. The Schönhage–Strassen algorithm [25] gives $M(n) = O(n\log n\log\log n)$, and Fürer [17] has recently given an improved bound $M(n) = O(n(\log n)2^{\log^* n})$. For simplicity we merely assume that $M(n) = O(n(\log n)^{1+o(1)})$, where the $o(1)$ term depends on the precise algorithm used for multiplication. For example, if the Schönhage–Strassen algorithm is used, then the $o(1)$ term can be replaced by $\log\log\log n/\log\log n$.

In Sects. 8.2 and 8.3 we mention some relevant and generally well-known facts concerning Bernoulli, Tangent, and Secant numbers.

Recently, Harvey [20] showed that the *single* number B_n can be computed in $O(n^2(\log n)^{2+o(1)})$ bit-operations using a modular algorithm. In this paper we show

that *all* the Bernoulli numbers $B_0, \ldots, B_n$ can be computed with the same complexity bound (and similarly for Secant and Tangent numbers).

In Sect. 8.4 we give a relatively simple algorithm that achieves the slightly weaker bound $O(n^2(\log n)^{3+o(1)})$. In Sect. 8.5 we describe the improvement to $O(n^2(\log n)^{2+o(1)})$. The idea is similar to that espoused by Steel [29], although we reduce the problem to division rather than multiplication. It is an open question whether the *single* number B_{2n} can be computed in $o(n^2)$ bit-operations.

In Sect. 8.6 we give very short in-place algorithms for computing the first n Secant or Tangent numbers using $O(n^2)$ integer operations. These algorithms are extremely simple and fast for moderate values of n (say $n \leq 1000$), although asymptotically not as fast as the algorithms given in Sects. 8.4 and 8.5. Bernoulli numbers can easily be deduced from the corresponding Tangent numbers using the relation (8.14) below.

8.2   Bernoulli Numbers

From the generating function (8.1) it is easy to see that the B_n are rational numbers, with $B_{2n+1} = 0$ if $n > 0$. The first few nonzero B_n are $B_0 = 1$, $B_1 = -1/2$, $B_2 = 1/6$, $B_4 = -1/30$, $B_6 = 1/42$, $B_8 = -1/30$, $B_{10} = 5/66$, $B_{12} = -691/2730$, and $B_{14} = 7/6$.

The denominators of the Bernoulli numbers are given by the *Von Staudt–Clausen Theorem* [12, 28], which states that

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

Here the sum is over all primes p for which $p - 1$ divides $2n$.

Since the "correction" $B'_{2n} - B_{2n}$ is easy to compute, it might be convenient in a program to store the integers B'_{2n} instead of the rational numbers B_{2n} or C_n.

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

$$(-1)^{n-1} \frac{B_{2n}}{(2n)!} = \frac{2\zeta(2n)}{(2\pi)^{2n}}. \tag{8.5}$$

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

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

From Stirling's approximation to $(2n)!$, the number of bits in the integer part of B_{2n} is $2n \lg n + O(n)$ (we write $\lg$ for $\log_2$). Thus, it takes $\Omega(n^2 \log n)$ space to store $B_1, \ldots, B_n$. We cannot expect any algorithm to compute $B_1, \ldots, B_n$ in fewer than $\Omega(n^2 \log n)$ bit-operations.

Another connection between the Bernoulli numbers and the Riemann zeta-function is the identity

$$\frac{B_{n+1}}{n+1} = -\zeta(-n) \tag{8.6}$$

for $n \in \mathbb{Z}$, $n \geq 1$. This follows from (8.5) and the functional equation for the zeta-function or directly from a contour integral representation of the zeta-function [31].

From the generating function (8.1), multiplying both sides by $\exp(z) - 1$ and equating coefficients of z, we obtain the recurrence

$$\sum_{j=0}^{k} \binom{k+1}{j} B_j = 0 \text{ for } k > 0. \tag{8.7}$$

This recurrence has traditionally been used to compute $B_0, \ldots, B_{2n}$ with $O(n^2)$ arithmetic operations, for example, in [22]. However, this is unsatisfactory if floating-point numbers are used, because the recurrence is *numerically unstable*: the relative error in the computed B_{2n} is of order $4^n \varepsilon$ if the floating-point arithmetic has precision ε, i.e., $\lg(1/\varepsilon)$ bits.

Let C_n be defined by (8.2). Then, multiplying each side of (8.3) by $\sinh(z/2)/(z/2)$ and equating coefficients gives the recurrence

$$\sum_{j=0}^{k} \frac{C_j}{(2k+1-2j)! \, 4^{k-j}} = \frac{1}{(2k)! \, 4^k}. \tag{8.8}$$

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

Equation (8.5) can be used in several ways to compute Bernoulli numbers. If we want just one Bernoulli number B_{2n} then $\zeta(2n)$ on the right-hand side of (8.5) can be evaluated to sufficient accuracy using the Euler product: this is the "zeta-function" algorithm for computing Bernoulli numbers mentioned (with several references to earlier work) by Harvey [20]. On the other hand, if we want several Bernoulli numbers, then we can use the generating function

$$\frac{\pi z}{\tanh(\pi z)} = -2 \sum_{k=0}^{\infty} (-1)^k \zeta(2k) z^{2k}, \tag{8.9}$$

computing the coefficients of z^{2k}, $k \leq n$, to sufficient accuracy, as mentioned in [3,8,9]. This is similar to the fast algorithm that we describe in Sect. 8.4. The similarity can be seen more clearly if we replace πz by z in (8.9), giving

$$\frac{z}{\tanh(z)} = -2 \sum_{k=0}^{\infty} (-1)^k \frac{\zeta(2k)}{\pi^{2k}} z^{2k}, \tag{8.10}$$

since it is the rational number $\zeta(2n)/\pi^{2n}$ that we need in order to compute B_{2n} from (8.5). In fact, it is easy to see that (8.10) is equivalent to (8.3).

There is a vast literature on Bernoulli, Tangent, and Secant numbers. For example, the bibliography of Dilcher and Slavutskii [15] contains more than 2,000 items. Thus, we do not attempt to give a complete list of references to related work. However, we briefly mention the problem of computing *irregular primes* [10, 11], which are odd primes p such that p divides the class number of the pth cyclotomic field. The algorithms that we present in Sects. 8.4 and 8.5 below are not suitable for this task because they take too much memory. It is much more space-efficient to use a modular algorithm where the computations are performed modulo a single prime (or maybe the product of a small number of primes), as in [10, 11, 14, 20]. Space can also be saved by the technique of "multisectioning", which is described by Crandall [13, Sect. 3.2] and Hare [19].

8.3   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. 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 $(2n)!(2/\pi)^n$. To obtain more precise estimates, let

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

be the *odd* zeta-function. Then

$$\frac{T_n}{(2n-1)!} = \frac{2^{2n+1}\zeta_0(2n)}{\pi^{2n}} \sim \frac{2^{2n+1}}{\pi^{2n}} \tag{8.11}$$

(this can be proved in the same way as Euler's relation (8.5) for the Bernoulli numbers). We also have [1, (23.2.22)]

$$\frac{S_n}{(2n)!} = \frac{2^{2n+2}\beta(2n+1)}{\pi^{2n+1}} \sim \frac{2^{2n+2}}{\pi^{2n+1}}, \tag{8.12}$$

where

$$\beta(s) = \sum_{j=0}^{\infty} (-1)^j (2j+1)^{-s}. \tag{8.13}$$

From (8.5) and (8.11), we see that

$$T_n = (-1)^{n-1} 2^{2n} (2^{2n} - 1) \frac{B_{2n}}{2n}. \tag{8.14}$$

This can also be proved directly, without involving the zeta-function, by using the identity

$$\tan z = \frac{1}{\tan z} - \frac{2}{\tan(2z)}.$$

Since $T_n \in \mathbb{Z}$, it follows from (8.14) that the odd primes in the denominator of B_{2n} must divide $2^{2n} - 1$. This is compatible with the Von Staudt–Clausen theorem, since $(p-1)|2n$ implies $p|(2^{2n}-1)$ by Fermat's little theorem.

T_n has about $4n$ more bits than $\lceil B_{2n} \rceil$, but both have $2n \lg n + O(n)$ bits, so asymptotically there is not much difference between the sizes of T_n and $\lceil B_{2n} \rceil$. Thus, if our aim is to compute B_{2n}, we do not lose much by first computing T_n, and this may be more convenient since $T_n \in \mathbb{Z}$, $B_{2n} \in \mathbb{Q}$.

8.4 A Fast Algorithm for Bernoulli Numbers

Harvey [20] showed how B_n could be computed exactly, using a modular algorithm and the Chinese remainder theorem, in $O(n^2 (\log n)^{2+o(1)})$ bit-operations. The same complexity can be obtained using (8.5) and the Euler product for the zeta-function (see the discussion in Harvey [20, Sect. 1]).

In this section we show how to compute *all* of $B_0, \ldots, B_n$ with almost the same complexity bound (only larger by a factor $O(\log n)$). In Sect. 8.5 we give an even faster algorithm, which avoids the $O(\log n)$ factor.

Let $A(z) = a_0 + a_1 z + a_2 z^2 + \cdots$ be a power series with coefficients in $\mathbb{R}$, with $a_0 \neq 0$. Let $B(z) = b_0 + b_1 z + \cdots$ be the *reciprocal* power series, so $A(z)B(z) = 1$. Using the FFT, we can multiply polynomials of degree $n - 1$ with $O(n \log n)$ real operations. Using Newton's method [24, 26], we can compute $b_0, \ldots, b_{n-1}$ with the *same* complexity $O(n \log n)$, up to a constant factor.

Taking $A(z) = (\exp(z) - 1)/z$ and working with N-bit floating-point numbers, where $N = n \lg(n) + O(n)$, we get $B_0, \ldots, B_n$ to sufficient accuracy to deduce the exact (rational) result. (Alternatively, use (8.3) to avoid computing the terms with odd subscripts, since these vanish except for B_1.) The work involved is $O(n \log n)$ floating-point operations, each of which can be done with N-bit accuracy in $O(n (\log n)^{2+o(1)})$ bit-operations. Thus, overall we get $B_0, \ldots, B_n$

with $O(n^2(\log n)^{3+o(1)})$ bit-operations. Similarly for Secant and Tangent numbers. We omit a precise specification of N and a detailed error analysis of the algorithm, since it is improved in the following section.

8.5   A Faster Algorithm for Tangent and Bernoulli Numbers

To improve the algorithm of Sect. 8.4 for Bernoulli numbers, we use the "Kronecker–Schönhage trick" [7, Sect. 1.9]. Instead of working with power series $A(z)$ (or polynomials, which can be regarded as truncated power series), we work with binary numbers $A(z)$ where z is a suitable (negative) power of 2.

The idea is to compute a single real number $\mathscr{A}$ which is defined in such a way that the numbers that we want to compute are encoded in the binary representation of $\mathscr{A}$. For example, consider the series

$$\sum_{k>0} k^2 z^k = \frac{z(1+z)}{(1-z)^3}, \quad |z| < 1.$$

The right-hand side is an easily computed rational function of z, say $A(z)$. We use decimal rather than binary for expository purposes. With $z = 10^{-3}$ we easily find

$$A(10^{-3}) = \frac{1001000}{997002999} = 0.00\underline{1}\,00\underline{4}\,00\underline{9}\,0\underline{16}\,0\underline{25}\,0\underline{36}\,0\underline{49}\,0\underline{64}\,0\underline{81}\,\underline{100}\cdots$$

Thus, if we are interested in the finite sequence of squares $(1^2, 2^2, 3^2, \ldots, 10^2)$, it is sufficient to compute $\mathscr{A} = A(10^{-3})$ correctly rounded to 30 decimal places, and we can then "read off" the squares from the decimal representation of $\mathscr{A}$.

Of course, this example is purely for illustrative purposes, because it is easy to compute the sequence of squares directly. However, we use the same idea to compute Tangent numbers. Suppose we want the first n Tangent numbers $(T_1, T_2, \ldots, T_n)$. The generating function

$$\tan z = \sum_{k \geq 1} T_k \frac{z^{2k-1}}{(2k-1)!}$$

gives us almost what we need, but not quite, because the coefficients are rationals, not integers. Instead, consider

$$(2n-1)!\tan z = \sum_{k=1}^{n} T'_{k,n} z^{2k-1} + R_n(z), \tag{8.15}$$

where

$$T'_{k,n} = \frac{(2n-1)!}{(2k-1)!} T_k \tag{8.16}$$

is an integer for $1 \leq k \leq n$, and

$$R_n(z) = \sum_{k=n+1}^{\infty} T'_{k,n} z^{2k-1} = (2n-1)! \sum_{k=n+1}^{\infty} T_k \frac{z^{2k-1}}{(2k-1)!} \tag{8.17}$$

is a remainder term which is small if z is sufficiently small. Thus, choosing $z = 2^{-p}$ with p sufficiently large, the first $2np$ binary places of $(2n-1)! \tan z$ define $T'_{1,n}, T'_{2,n}, \ldots, T'_{n,n}$. Once we have computed $T'_{1,n}, T'_{2,n}, \ldots, T'_{n,n}$ it is easy to deduce $T_1, T_2, \ldots, T_n$ from

$$T_k = \frac{T'_{k,n}}{(2n-1)!/(2k-1)!}.$$

For this idea to work, two conditions must be satisfied. First, we need

$$0 \leq T'_{k,n} < 1/z^2 = 2^{2p}, \ 1 \leq k \leq n, \tag{8.18}$$

so we can read off the $T'_{k,n}$ from the binary representation of $(2n-1)! \tan z$. Since we have a good asymptotic estimate for T_k, it is not hard to choose p sufficiently large for this condition to hold.

Second, we need the remainder term $R_n(z)$ to be sufficiently small that it does not influence the estimation of $T'_{n,n}$. A sufficient condition is

$$0 \leq R_n(z) < z^{2n-1}. \tag{8.19}$$

Choosing z sufficiently small (i.e., p sufficiently large) guarantees that condition (8.19) holds, since $R_n(z)$ is $O(z^{2n+1})$ as $z \to 0$ with n fixed.

Lemmas 8.3 and 8.4 below give sufficient conditions for (8.18) and (8.19) to hold.

Lemma 8.1.

$$\frac{T_k}{(2k-1)!} \leq \left(\frac{2}{\pi}\right)^{2(k-1)} \quad \text{for } k \geq 1.$$

Proof. From (8.11),

$$\frac{T_k}{(2k-1)!} = 2\left(\frac{2}{\pi}\right)^{2k} \zeta_0(2k) \leq 2\left(\frac{2}{\pi}\right)^{2k} \zeta_0(2) \leq \left(\frac{2}{\pi}\right)^{2k} \frac{\pi^2}{4} = \left(\frac{2}{\pi}\right)^{2(k-1)}.$$

∎

Lemma 8.2. $(2n-1)! \leq n^{2n-1}$ *for* $n \geq 1$.

Proof.

$$(2n-1)! = n \prod_{j=1}^{n-1} (n-j)(n+j) = n \prod_{j=1}^{n-1} (n^2 - j^2) \leq n^{2n-1}$$

with equality iff $n = 1$. ∎

Lemma 8.3. *If* $k \geq 1$, $n \geq 2$, $p = \lceil n \lg(n) \rceil$, $z = 2^{-p}$, *and* $T'_{k,n}$ *is as in* (8.16), *then* $z \leq n^{-n}$ *and* $T'_{k,n} < 1/z^2$.

Proof. We have $z = 2^{-p} = 2^{-\lceil n \lg(n) \rceil} \leq 2^{-n \lg(n)} = n^{-n}$, which proves the first part of the Lemma.

Assume $k \geq 1$ and $n \geq 2$. From Lemma 8.1, we have

$$T'_{k,n} \leq (2n-1)! \left(\frac{2}{\pi}\right)^{2(k-1)} \leq (2n-1)!,$$

and from Lemma 8.2 it follows that

$$T'_{k,n} \leq n^{2n-1} < n^{2n}.$$

From the first part of the Lemma, $n^{2n} \leq 1/z^2$, so the second part follows. ∎

Lemma 8.4. *If* $n \geq 2$, $p = \lceil n \lg(n) \rceil$, $z = 2^{-p}$, *and* $R_n(z)$ *is as defined in* (8.17), *then* $0 < R_n(z) < 0.1 z^{2n-1}$.

Proof. Since all the terms in the sum defining $R_n(z)$ are positive, it is immediate that $R_n(z) > 0$. Since $n \geq 2$, we have $p \geq 2$ and $z \leq 1/4$. Now, using Lemma 8.1,

$$R_n(z) = \sum_{k=n+1}^{\infty} T'_{k,n} z^{2k-1}$$

$$\leq (2n-1)! \sum_{k=n+1}^{\infty} \left(\frac{2}{\pi}\right)^{2(k-1)} z^{2k-1}$$

$$\leq (2n-1)! \left(\frac{2}{\pi}\right)^{2n} z^{2n+1} \left(1 + \left(\frac{2z}{\pi}\right)^2 + \left(\frac{2z}{\pi}\right)^4 + \cdots\right)$$

$$\leq (2n-1)! \left(\frac{2}{\pi}\right)^{2n} z^{2n+1} \bigg/ \left(1 - \left(\frac{2z}{\pi}\right)^2\right).$$

Since $z \leq 1/4$, we have $1/(1 - (2z/\pi)^2) < 1.026$. Also, from Lemma 8.2, $(2n-1)! \leq n^{2n-1}$. Thus, we have

Input: integer $n \geq 2$
Output: Tangent numbers $T_1, \ldots, T_n$ and (optional) Bernoulli numbers $B_2, B_4, \ldots, B_{2n}$
    $p \leftarrow \lceil n \lg(n) \rceil$
    $z \leftarrow 2^{-p}$
    $S \leftarrow \sum_{0 \leq k < n} (-1)^k z^{2k+1} \times (2n)!/(2k+1)!$
    $C \leftarrow \sum_{0 \leq k < n} (-1)^k z^{2k} \times (2n)!/(2k)!$
    $V \leftarrow \lfloor z^{1-2n} \times (2n-1)! \times S/C \rceil$ (here $\lfloor x \rceil$ means round x to nearest integer)
    Extract $T'_{k,n} = T_k(2n-1)!/(2k-1)!$, $1 \leq k \leq n$, from the binary representation of V
    $T_k \leftarrow T'_{k,n} \times (2k-1)!/(2n-1)!$, $k = n, n-1, \ldots, 1$
    $B_{2k} \leftarrow (-1)^{k-1}(k \times T_k/2^{2k-1})/(2^{2k}-1)$, $k = 1, 2, \ldots, n$ (optional)
    return $T_1, T_2, \ldots, T_n$ and (optional) $B_2, B_4, \ldots, B_{2n}$

Fig. 8.1 Algorithm FastTangentNumbers (also optionally computes Bernoulli numbers)

$$\frac{R_n(z)}{z^{2n-1}} < 1.026 n^{2n-1} \left(\frac{2}{\pi}\right)^{2n} z^2.$$

Now $z^2 \leq n^{-2n}$ from the first part of Lemma 8.3, so

$$\frac{R_n(z)}{z^{2n-1}} < \frac{1.026}{n} \left(\frac{2}{\pi}\right)^{2n}. \tag{8.20}$$

The right-hand side is a monotonic decreasing function of n, so is bounded above by its value when $n = 2$, giving $R_n(z)/z^{2n-1} < 0.1$. ∎

A high-level description of the resulting Algorithm FastTangentNumbers is given in Fig. 8.1. The algorithm computes the Tangent numbers $T_1, T_2, \ldots, T_n$ using the Kronecker–Schönhage trick as described above, and deduces the Bernoulli numbers $B_2, B_4, \ldots, B_{2n}$ from the relation (8.14).

In order to achieve the best complexity, the algorithm must be implemented carefully using binary arithmetic. The computations of S (an approximation to $(2n)! \sin z$) and C (an approximation to $(2n)! \cos z$) involve computing ratios of factorials such as $(2n)!/(2k)!$, where $0 \leq k \leq n$. This can be done in time $O(n^2 (\log n)^2)$ by a straightforward algorithm. The N-bit division to compute S/C (an approximation to $\tan z$) can be done in time $O(N \log(N) \log \log(N))$ by the Schönhage–Strassen algorithm combined with Newton's method [7, Sect. 4.2.2]. Here it is sufficient to take $N = 2np + 2 = 2n^2 \lg(n) + O(n)$. Note that

$$V = \sum_{k=1}^{n} 2^{2(n-k)p} T'_{k,n} \tag{8.21}$$

is just the finite sum in (8.15) scaled by z^{1-2n} (a power of two), and the integers $T'_{k,n}$ can simply be "read off" from the binary representation of V in n blocks of $2p$ consecutive bits. The $T'_{k,n}$ can then be scaled by ratios of factorials in time $O(n^2 (\log n)^{2+o(1)})$ to give the Tangent numbers $T_1, T_2, \ldots, T_n$.

The correctness of the computed Tangent numbers follows from Lemmas 8.3 and 8.4, apart from possible errors introduced by S/C being only an approximation to $\tan(z)$. Lemma 8.5 shows that this error is sufficiently small.

Lemma 8.5. *Suppose that $n \geq 2$, z, S and C as in Algorithm FastTangentNumbers. Then*

$$z^{1-2n}(2n-1)! \left| \frac{S}{C} - \tan z \right| < 0.02 \,. \tag{8.22}$$

Proof. We use the inequality

$$\left| \frac{A}{B} - \frac{A'}{B'} \right| \leq \frac{|A| \cdot |B - B'| + |B| \cdot |A - A'|}{|B| \cdot |B'|} \,. \tag{8.23}$$

Take $A = \sin z$, $B = \cos z$, $A' = S/(2n)!$, $B' = C/(2n)!$ in (8.23). Since $n \geq 2$ we have $0 < z \leq 1/4$. Then $|A| = |\sin z| < z$. Also, $|B| = |\cos z| > 31/32$ from the Taylor series $\cos z = 1 - z^2/2 + \cdots$, which has terms of alternating sign and decreasing magnitude. By similar arguments, $|B'| \geq 31/32$, $|B - B'| < z^{2n}/(2n)!$, and $|A - A'| < z^{2n+1}/(2n+1)!$. Combining these inequalities and using (8.23), we obtain

$$\left| \frac{S}{C} - \tan z \right| < \frac{6 \cdot 32 \cdot 32}{5 \cdot 31 \cdot 31} \frac{z^{2n+1}}{(2n)!} < \frac{1.28 \, z^{2n+1}}{(2n)!} \,.$$

Multiplying both sides by $z^{1-2n}(2n-1)!$ and using $1.28 z^2/(2n) \leq 0.02$, we obtain the inequality (8.22). This completes the proof of Lemma 8.5. ∎

In view of the constant 0.02 in (8.22) and the constant 0.1 in Lemma 8.4, the effect of all sources of error in computing $z^{1-2n}(2n-1)! \tan z$ is at most $0.12 < 1/2$, which is too small to change the computed integer V, that is to say, the computed V is indeed given by (8.21).

The computation of the Bernoulli numbers $B_2, B_4, \ldots, B_{2n}$ from $T_1, \ldots, T_n$, is straightforward (details depending on exactly how rational numbers are to be represented). The entire computation takes time

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

Thus, we have proved:

Theorem 8.6. *The Tangent numbers $T_1, \ldots, T_n$ and Bernoulli numbers $B_2, B_4, \ldots, B_{2n}$ can be computed in $O(n^2(\log n)^{2+o(1)})$ bit-operations using $O(n^2 \log n)$ space.*

A small modification of the above can be used to compute the Secant numbers $S_0, S_1, \ldots, S_n$ in $O(n^2(\log n)^{2+o(1)})$ bit-operations and $O(n^2 \log n)$ space. The bound on Tangent numbers given by Lemma 8.1 can be replaced by the bound

$$\frac{S_n}{(2n)!} \leq 2\left(\frac{2}{\pi}\right)^{2n+1}$$

which follows from (8.12) since $\beta(2n+1) < 1$.

We remark that an efficient implementation of Algorithm FastTangentNumbers in a high-level language such as Sage [30] or Magma [5] is nontrivial, because it requires access to the internal binary representation of high-precision integers. Everything can be done using (implicitly scaled) integer arithmetic—there is no need for floating-point—but for the sake of clarity we did not include the scaling in Fig. 8.1. If floating-point arithmetic is used, a precision of N bits is sufficient, where $N = 2np + 2$.

Comparing our Algorithm FastTangentNumbers with Harvey's modular algorithm [20], we see that there is a space-time trade-off: Harvey's algorithm uses less space (by a factor of order n) to compute a single B_n, but more time (again by a factor of order n) to compute all of $B_1, \ldots, B_n$. Harvey's algorithm has better locality and is readily parallelizable.

In the following section we give much simpler algorithms which are fast enough for most practical purposes and are based on three-term recurrence relations.

8.6 Algorithms Based on Three-Term Recurrences

Akiyama and Tanigawa [21] gave an algorithm for computing Bernoulli numbers based on a three-term recurrence. However, it is only useful for exact computations, since it is numerically unstable if applied using floating-point arithmetic. It is faster to use a stable recurrence for computing Tangent numbers and then deduce the Bernoulli numbers from (8.14).

8.6.1 Bernoulli and Tangent Numbers

We now give a stable three-term recurrence and corresponding in-place algorithm for computing Tangent numbers. The algorithm 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. This is partly because the operations are on integers rather than rationals and partly because there are fewer operations since we take advantage of zeros.

Bernoulli numbers can be computed using Algorithm TangentNumbers and the relation (8.14). The time required for the application of (8.14) is negligible.
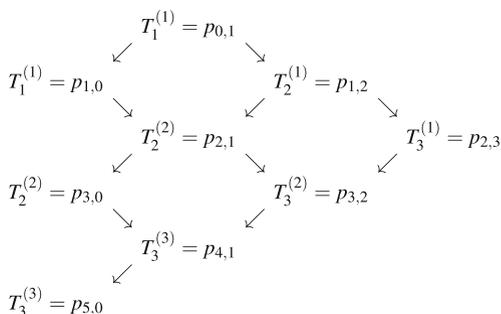
The recurrence (8.24) that we use was given by Buckholtz and Knuth [23], but they did not give our in-place Algorithm TangentNumbers explicitly. Related recurrences with applications to parallel computation were considered by Hare [19].

Fig. 8.2 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 $T_1, T_2, \ldots, T_n$.

Fig. 8.3 Dataflow in
algorithm TangentNumbers
for $n = 3$

$$T_1^{(1)} = p_{0,1}$$

$$T_1^{(1)} = p_{1,0} \qquad\qquad T_2^{(1)} = p_{1,2}$$

$$T_2^{(2)} = p_{2,1} \qquad\qquad T_3^{(1)} = p_{2,3}$$

$$T_2^{(2)} = p_{3,0} \qquad\qquad T_3^{(2)} = p_{3,2}$$

$$T_3^{(3)} = p_{4,1}$$

$$T_3^{(3)} = p_{5,0}$$

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 $D(t^n)$,

$$p_{n,j} = (j-1)p_{n-1,j-1} + (j+1)p_{n-1,j+1}. \tag{8.24}$$

We are interested in $T_k = (\mathrm{d}/\mathrm{d}x)^{2k-1} \tan x|_{x=0} = P_{2k-1}(0) = p_{2k-1,0}$, which can be computed from the recurrence (8.24) in $O(k^2)$ operations using the obvious boundary conditions. We save work by noticing that $p_{n,j} = 0$ if $n + j$ is even. The resulting algorithm is given in Fig. 8.2.

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, $T_k = p_{2k-1,0}$, as expected.

The process in the case $n = 3$ is illustrated in Fig. 8.3, where $T_k^{(m)}$ denotes the value of the variable T_k at successive iterations $m = 1, 2, \ldots, n$. It is instructive to compare a similar figure for the Akiyama–Tanigawa algorithm in [21].

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 $O(n^3 (\log n)^{1+o(1)})$ 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 such as those described in Sects. 8.4 and 8.5. For example, using a straightforward Magma implementation of Algorithm TangentNumbers, we computed the first $1,000$ Tangent numbers in 1.50 s

Fig. 8.4 Algorithm
SecantNumbers

Input: positive integer n
Output: Secant numbers $S_0, S_1 \ldots, S_n$
   $S_0 \leftarrow 1$
  for k **from** 1 **to** n
    $S_k \leftarrow kS_{k-1}$
  for k **from** 1 **to** n
    for j **from** $k+1$ **to** n
      $S_j \leftarrow (j-k)S_{j-1} + (j-k+1)S_j$
  return $S_0, S_1, \ldots, S_n$.

on a 2.26 GHz Intel Core 2 Duo. For comparison, it takes 1.92 s for a single N-bit division computing T in Algorithm FastTangentNumbers (where $N = 19,931,568$ corresponds to $n = 1,000$). Thus, we expect the crossover point where Algorithm FastTangentNumbers actually becomes faster to be slightly larger than $n = 1,000$ (but dependent on implementation details).

8.6.2 Secant Numbers

A similar algorithm may be used to compute Secant numbers. Let $s = \sec x, t = \tan x$, and $D = d/dx$. Then $Ds = st$, $D^2 s = s(1 + 2t^2)$, and in general $D^n s = sQ_n(t)$, where $Q_n(t)$ is a polynomial of degree n in t. The Secant numbers are given by $S_k = Q_{2k}(0)$. Let $Q_n(t) = \sum_{k \geq 0} q_{n,k} t^k$. From

$$D(st^k) = st^{k+1} + kst^{k-1}(1 + t^2)$$

we obtain the three-term recurrence

$$q_{n+1,k} = kq_{n,k-1} + (k+1)q_{n,k+1} \text{ for } 1 \leq k \leq n. \tag{8.25}$$

By avoiding the computation of terms $q_{n,k}$ that are known to be zero ($n + k$ odd), and ordering the computation in a manner analogous to that used for Algorithm TangentNumbers, we obtain Algorithm SecantNumbers (see Fig. 8.4), which computes the Secant numbers in place using non-negative integer arithmetic.

8.6.3 Comparison with Atkinson's Algorithm

Atkinson [2] gave an elegant algorithm for computing both the Tangent numbers $T_1, T_2, \ldots, T_n$ and the Secant numbers $S_0, S_1, \ldots, S_n$ using a "Pascal's triangle" style of algorithm that only involves additions of non-negative integers. Since a triangle with $2n + 1$ rows in involved, Atkinson's algorithm requires $2n^2 + O(n)$

integer additions. This can be compared with $n^2/2 + O(n)$ additions and $n^2 + O(n)$ multiplications (by small integers) for our Algorithm TangentNumbers, and similarly for Algorithm SecantNumbers.

Thus, we might expect Atkinson's algorithm to be slower than Algorithm TangentNumbers. Computational experiments confirm this. With $n = 1,000$, Algorithm TangentNumbers programmed in Magma takes 1.50 s on a 2.26 GHz Intel Core 2 Duo, algorithm SecantNumbers also takes 1.50 s, and Atkinson's algorithm takes 4.51 s. Thus, even if both Tangent and Secant numbers are required, Atkinson's algorithm is slightly slower. It also requires about twice as much memory.

Note added in proof Recently the second author [A subquadratic algorithm for computing the n-th Bernoulli number, arXiv:1209.0533, to appear in Mathematics of Computation] has given an improved algorithm for the computation of a single Bernoulli number. The new algorithm reduces the exponent from $2 + o(1)$ to $4/3 + o(1)$.

References

1. Abramowitz, M., Stegun, I.A.: Handbook of Mathematical Functions. Dover (1973)
2. Atkinson, M.D.: How to compute the series expansions of sec x and tan x. Am. Math. Monthly **93**, 387–389 (1986)
3. Bailey, D.H., Borwein, J.M., Crandall, R.E.: On the Khintchine constant. Math. Comput. **66**, 417–431 (1997)
4. Borwein, J.M., Corless, R.M.: Emerging tools for experimental mathematics. Am. Math. Mon. **106**, 899–909 (1999)
5. Bosma, W., Cannon, J., Playoust, C.: The Magma algebra system. I: the user language. J. Symbolic Comput. **24**, 235–265 (1997)
6. Brent, R.P.: Unrestricted algorithms for elementary and special functions. Information Processing, vol. 80, pp. 613–619. North-Holland, Amsterdam (1980). arXiv:1004.3621v1
7. Brent, R.P., Zimmermann, P.: Modern Computer Arithmetic. Cambridge University Press, Cambridge (2010). arXiv:1004.4710v1
8. Buhler, J., Crandall, R., Sompolski, R.: Irregular primes to one million. Math. Comput. **59**, 717–722 (1992)
9. Buhler, J., Crandall, R., Ernvall, R., Metsänkylä, T.: Irregular primes to four million. Math. Comput. **61**, 151–153 (1993)
10. Buhler, J., Crandall, R., Ernvall, R., Metsänkylä, T., Shokrollahi, M.A.: Irregular primes and cyclotomic invariants to twelve million. J. Symbolic Comput. **31**, 89–96 (2001)

11. Buhler, J., Harvey, D.: Irregular primes to 163 million. Math. Comput. **80**, 2435–2444 (2011)
12. Clausen, T.: Theorem. Astron. Nachr. **17**, 351–352 (1840)
13. Crandall, R.E.: Topics in Advanced Scientific Computation. Springer, New York (1996)
14. Crandall, R.E., Pomerance, C.: Prime Numbers: A Computational Perspective. Springer, New York (2001)
15. Dilcher, K., Slavutskii, I.Sh.: A Bibliography of Bernoulli Numbers (last updated March 3, 2007). http://www.mscs.dal.ca/%7Edilcher/bernoulli.html.
16. Ferguson, H.R.P., Bailey, D.H., Arno, S.: Analysis of PSLQ, an integer relation finding algorithm. Math. Comput. **68**, 351–369 (1999)
17. Fürer, M.: Faster integer multiplication. Proceedings of 39th Annual ACM Symposium on Theory of Computing (STOC), pp. 57–66. ACM, San Diego (2007)
18. Graham, R.L., Knuth, D.E., Patashnik, O.: Concrete Mathematics, 3rd edn. Addison-Wesley, Reading (1994)
19. Hare, K.: Multisectioning, rational poly-exponential functions and parallel computation. M.Sc. thesis, Department of Mathematics and Statistics, Simon Fraser University, Canada (2002)
20. Harvey, D.: A multimodular algorithm for computing Bernoulli numbers. Math. Comput. **79**, 2361–2370 (2010)
21. Kaneko, M.: The Akiyama–Tanigawa algorithm for Bernoulli numbers. J. Integer Seq. **3**, Article 00.2.9, 6 (2000). http://www.cs.uwaterloo.ca/journals/JIS/
22. Knuth, D.E.: Euler's constant to 1271 places. Math. Comput. **16**, 275–281 (1962)
23. Knuth, D.E., Buckholtz, T.J.: Computation of Tangent, Euler, and Bernoulli numbers. Math. Comput. **21**, 663–688 (1967)
24. Kung, H.T.: On computing reciprocals of power series. Numer. Math. **22**, 341–348 (1974)
25. Schönhage, A., Strassen, V.: Schnelle Multiplikation großer Zahlen. Computing **7**, 281–292 (1971)
26. Sieveking, M.: An algorithm for division of power series. Computing **10**, 153–156 (1972)
27. Sloane, N.J.A.: The on-line encyclopedia of integer sequences. http://oeis.org
28. Von Staudt, K.G.C.: Beweis eines Lehrsatzes, die Bernoullischen Zahlen betreffend. J. Reine Angew. Math. **21**, 372–374 (1840). http://gdz.sub.uni-goettingen.de
29. Steel, A.: Reduce everything to multiplication. Presented at Computing by the Numbers: Algorithms, Precision and Complexity. Workshop for Richard Brent's 60th Birthday, Berlin, 2006. http://www.mathematik.hu-berlin.de/%7Egaggle/EVENTS/2006/BRENT60/
30. Stein, W. et al.: Sage. http://www.sagemath.org/
31. Titchmarsh, E.C.: The Theory of the Riemann Zeta-Function, 2nd edn (revised by D. R. Heath-Brown). Clarendon Press, Oxford (1986)