draft manuscript No.
(will be inserted by the editor)

Rigorous high-precision computation of the Hurwitz zeta function and its derivatives

Fredrik Johansson

Abstract We study the use of the Euler-Maclaurin formula to numerically evaluate the Hurwitz zeta function $\zeta(s, a)$ for $s, a \in \mathbb{C}$, along with an arbitrary number of derivatives with respect to s, to arbitrary precision with rigorous error bounds. Techniques that lead to a fast implementation are discussed. We present new record computations of Stieltjes constants, Keiper-Li coefficients and the first nontrivial zero of the Riemann zeta function, obtained using an open source implementation of the algorithms described in this paper.

Keywords Hurwitz zeta function · Riemann zeta function · arbitrary-precision arithmetic · rigorous numerical evaluation · fast polynomial arithmetic · power series

Mathematics Subject Classification (2000) 65D20 · 68W30 · 33F05 · 11-04 · 11M06 · 11M35

1 Introduction

The Hurwitz zeta function $\zeta(s, a)$ is defined for complex numbers s and a by analytic continuation of the sum

$$\zeta(s, a) = \sum_{k=0}^{\infty} \frac{1}{(k+a)^s}.$$

The usual Riemann zeta function is given by $\zeta(s) = \zeta(s, 1)$.

In this work, we consider numerical computation of $\zeta(s, a)$ by the Euler-Maclaurin formula with rigorous error control. Error bounds for $\zeta(s)$ are classical (see for example [14, 7] and numerous references therein), but previous works have restricted to the case $a = 1$ or have not considered derivatives. Our main contribution is to give an efficiently computable error bound for $\zeta(s, a)$

RISC, Johannes Kepler University, 4040 Linz, Austria
fredrik.johansson@risc.jku.at

valid for any complex s and a and for an arbitrary number of derivatives with respect to s (equivalently, we allow s to be a formal power series).

We also discuss implementation aspects, such as parallelization and use of fast polynomial arithmetic. An open source implementation of $\zeta(s, a)$ based on the algorithms described in this paper is available. In the last part of the paper, we present results from some new record computations done with this implementation.

Our interest is in evaluating $\zeta(s, a)$ to high precision (hundreds or thousands of digits) for a single s of moderate height, say with imaginary part less than 10^6. Investigations of zeros of large height typically use methods based on the Riemann-Siegel formula and fast multi-evaluation techniques such as the Odlyzko-Schönhage algorithm [32] or the recent algorithm of Hiary [22].

This work is motivated by several applications. For example, recent work of Matiyasevich and Beliakov required values of thousands of nontrivial zeros ρ_n of $\zeta(s)$ to a precision of several thousand digits [30,31]. Investigations of quantities such as the Stieltjes constants $\gamma_n(a)$ and the Keiper-Li coefficients λ_n also call for high-precision values [25,28]. The difficulty is not necessarily that the final result needs to be known to very high accuracy, but that intermediate calculations may involve catastrophic cancellation.

More broadly, the Riemann and Hurwitz zeta functions are useful for numerical evaluation of various other special functions such as polygamma functions, polylogarithms, Dirichlet L-functions, generalized hypergeometric functions at singularities [5], and certain number-theoretical constants [16]. High-precision numerical values are of particular interest for guessing algebraic relations among special values of such functions (which subsequently may be proved rigorously by other means) or ruling out the existence of algebraic relations with small norm [2].

2 Evaluation using the Euler-Maclaurin formula

Assume that f is analytic on a domain containing $[N, U]$ where $N, U \in \mathbb{Z}$, and let M be a positive integer. Let B_n denote the n-th Bernoulli number and let $\tilde{B}_n(t) = B_n(t - \lfloor t \rfloor)$ denote the n-th periodic Bernoulli polynomial. The Euler-Maclaurin summation formula (described in numerous works, such as [33]) states that

$$\sum_{k=N}^{U} f(k) = I + T + R \tag{1}$$

where

$$I = \int_N^U f(t)\,dt, \tag{2}$$

$$T = \frac{1}{2}\left(f(N) + f(U)\right) + \sum_{k=1}^{M} \frac{B_{2k}}{(2k)!}\left(f^{(2k-1)}(U) - f^{(2k-1)}(N)\right), \tag{3}$$

$$R = -\int_N^U \frac{\tilde{B}_{2M}(t)}{(2M)!} f^{(2M)}(t)\,dt. \tag{4}$$

If f decreases sufficiently rapidly, (1)–(4) remain valid after letting $U \to \infty$. To evaluate the Hurwitz zeta function, we set

$$f(k) = \frac{1}{(a+k)^s} = \exp(-s\log(a+k))$$

with the conventional logarithm branch cut on $(-\infty, 0)$. The derivatives of $f(k)$ are given by

$$f^{(r)}(k) = \frac{(-1)^r (s)_r}{(a+k)^{s+r}}$$

where $(s)_r = s(s+1)\cdots(s+r-1)$ denotes a rising factorial. The Euler-Maclaurin summation formula now gives, at least for $\Re(s) > 1$ and $a \neq 0, -1, -2, \ldots$,

$$\zeta(s,a) = \sum_{k=0}^{N-1} f(k) + \sum_{k=N}^{\infty} f(k) = S + I + T + R \tag{5}$$

where

$$S = \sum_{k=0}^{N-1} \frac{1}{(a+k)^s}, \tag{6}$$

$$I = \int_N^{\infty} \frac{1}{(a+t)^s} dt = \frac{(a+N)^{1-s}}{s-1}, \tag{7}$$

$$T = \frac{1}{(a+N)^s}\left(\frac{1}{2} + \sum_{k=1}^{M} \frac{B_{2k}}{(2k)!} \frac{(s)_{2k-1}}{(a+N)^{2k-1}}\right), \tag{8}$$

$$R = -\int_N^{\infty} \frac{\tilde{B}_{2M}(t)}{(2M)!} \frac{(s)_{2M}}{(a+t)^{s+2M}} dt. \tag{9}$$

If we choose N and M such that $\Re(a+N) > 0$ and $\Re(s+2M-1) > 0$, the integrals in I and R are well-defined, giving us the analytic continuation of $\zeta(s,a)$ to $s \in \mathbb{C}$ except for the pole at $s = 1$.

In order to evaluate derivatives with respect to s of $\zeta(s,a)$, we substitute $s \to s + x \in \mathbb{C}[[x]]$ and evaluate (5)–(9) with the corresponding arithmetic operations done on formal power series (which may be truncated at some arbitrary finite order in an implementation). For example, the summand in (6)

becomes

$$\frac{1}{(a+k)^{s+x}} = \sum_{i=0}^{\infty} \frac{(-1)^i \log(a+k)^i}{(a+k)^s i!}\, x^i \in \mathbb{C}[[x]]. \tag{10}$$

Note that we can evaluate $\zeta(S, a)$ for any formal power series $S = s + s_1 x + s_2 x^2 + \ldots$ by first evaluating $\zeta(s+x, a)$ and then formally right-composing by $S - s$. We can also easily evaluate derivatives of $\zeta(s, a) - 1/(s-1)$ at $s = 1$. The pole of $\zeta(s, a)$ only appears in the term I on the right hand side of (5), so we can remove the singularity as

$$\begin{aligned}\lim_{s\to 1}\left[I - \frac{1}{(s+x)-1} = \frac{(a+N)^{1-(s+x)}}{(s+x)-1} - \frac{1}{(s+x)-1}\right] \\ = \sum_{i=0}^{\infty} \frac{(-1)^{i+1}\log(a+N)^{i+1}}{(i+1)!}\, x^i \ \in \mathbb{C}[[x]].\end{aligned} \tag{11}$$

For $F = \sum_k f_k x^k \in \mathbb{C}[[x]]$, define $|F| = \sum_k |f_k| x^k \in \mathbb{R}[[x]]$. If it holds for all k that $|f_k| \le |g_k|$, we write $|F| \le |G|$. Clearly $|F + G| \le |F| + |G|$ and $|FG| \le |F||G|$. With this notation, we wish to bound $|R(s+x)|$ where $R(s) = R$ is the remainder integral given in (9).

To express the error bound in a compact form, we introduce the sequence of integrals defined for integers $k \ge 0$ and real parameters $A > 0, B > 1, C \ge 0$ by

$$J_k(A, B, C) \equiv \int_A^{\infty} t^{-B}(C + \log t)^k dt = \frac{L_k}{(B-1)^{k+1} A^{B-1}} \tag{12}$$

where $L_k = k! \sum_{m=0}^{k} D^m/m!$ and $D = (B-1)(C + \log A)$. The list of values $J_0, J_1, \ldots, J_n$ can be computed easily using $O(n)$ arithmetic operations via the recurrence $L_0 = 1$, $L_k = kL_{k-1} + D^k$. To prove the right-hand equality in (12), one can expand the integrand using the binomial theorem. As pointed out by an anonymous referee, (12) can also be evaluated via the incomplete gamma function $\Gamma(\alpha, x) = \int_x^{\infty} e^{-t} t^{\alpha-1} dt$ as

$$\begin{aligned}J_k(A, B, C) &= \int_{\log A}^{\infty} e^{-(B-1)u}(C+u)^k du = e^{C(B-1)} \int_{C+\log A}^{} e^{-(B-1)v} v^k dv \\ &= \frac{e^{C(B-1)}}{(B-1)^{k+1}} \int_D^{\infty} e^{-y} y^k dy = \frac{e^D \Gamma(k+1, D)}{(B-1)^{k+1} A^{B-1}}.\end{aligned}$$

Theorem 1 *Given complex numbers $s = \sigma + \tau i$, $a = \alpha + \beta i$ and positive integers N, M such that $\alpha + N > 1$ and $\sigma + 2M > 1$, the error term (9) in the Euler-Maclaurin summation formula applied to $\zeta(s+x, a) \in \mathbb{C}[[x]]$ satisfies*

$$|R(s+x)| \le \frac{4\,|(s+x)_{2M}|}{(2\pi)^{2M}} \left| \sum_{k=0}^{\infty} R_k x^k \right| \in \mathbb{R}[[x]] \tag{13}$$

where $R_k \le (K/k!)\, J_k(N + \alpha, \sigma + 2M, C)$, with

$$C = \frac{1}{2}\log\left(1 + \frac{\beta^2}{(\alpha+N)^2}\right) + \operatorname{atan}\left(\frac{|\beta|}{\alpha+N}\right) \tag{14}$$

and

$$K = \exp\left(\max\left(0, \tau \operatorname{atan}\left(\frac{\beta}{\alpha+N}\right)\right)\right). \tag{15}$$

Proof We have

$$\begin{aligned}
|R(s+x)| &= \left| \int_N^\infty \frac{\tilde{B}_{2M}(t)}{(2M)!} \frac{(s+x)_{2M}}{(a+t)^{s+x+2M}} dt \right| \\
&\le \int_N^\infty \left| \frac{\tilde{B}_{2M}(t)}{(2M)!} \frac{(s+x)_{2M}}{(a+t)^{s+x+2M}} \right| dt \\
&\le \frac{4\left|(s+x)_{2M}\right|}{(2\pi)^{2M}} \int_N^\infty \left| \frac{1}{(a+t)^{s+x+2M}} \right| dt
\end{aligned}$$

where the last step invokes the fact that

$$|\tilde{B}_{2M}(t)| < \frac{4(2M)!}{(2\pi)^{2M}}.$$

Thus it remains to bound the coefficients R_k satisfying

$$\int_N^\infty \left| \frac{1}{(a+t)^{s+x+2M}} \right| dt = \sum_k R_k x^k, \quad R_k = \int_N^\infty \frac{1}{k!} \left| \frac{\log(a+t)^k}{(a+t)^{s+2M}} \right| dt.$$

By the assumption that $\alpha + t \ge \alpha + N \ge 1$, we have

$$\begin{aligned}
|\log(\alpha+\beta i+t)| &= \left| \log(\alpha+t) + \log\left(1+\frac{\beta i}{\alpha+t}\right) \right| \\
&\le \log(\alpha+t) + \left| \log\left(1+\frac{\beta i}{\alpha+t}\right) \right| \\
&= \log(\alpha+t) + \left| \frac{1}{2}\log\left(1+\frac{\beta^2}{(\alpha+t)^2}\right) + i \operatorname{atan}\left(\frac{\beta}{\alpha+t}\right) \right| \\
&\le \log(\alpha+t) + C
\end{aligned}$$

where C is defined as in (14). By the assumption that $\sigma + 2M > 1$, we have

$$\frac{1}{|(\alpha+\beta i+t)^{\sigma+\tau i+2M}|} = \frac{\exp(\tau \arg(\alpha+\beta i+t))}{|\alpha+\beta i+t|^{\sigma+2M}} \le \frac{K}{(\alpha+t)^{\sigma+2M}}$$

where K is defined as in (15). Bounding the integrand in R_k in terms of the integrand in the definition of J_k now gives the result. □

The bound given in Theorem 1 should generally approximate the exact remainder (9) quite well, even for derivatives of large order, if $|a|$ is not too large. The quantity K is especially crude, however, as it does not decrease when $|a+t|^{-\tau i}$ decreases exponentially as a function of τ. We have made this

simplification in order to obtain a bound that is easy to evaluate for all s, a. In fact, assuming that a is small, we can simplify the bounds a bit further using

$$C \le \frac{\beta^2}{2(\alpha+N)^2} + \frac{|\beta|}{(\alpha+N)}.$$

In practice, the Hurwitz zeta function is usually only considered for $0 < a \le 1$, unless s is an integer greater than 1 in which case it reduces to a polygamma function of a. It is easy to derive error bounds for polygamma functions that are accurate for large $|a|$, and we do not consider this special case further here.

3 Algorithmic matters

The evaluation of $\zeta(s+x, a)$ can be broken into three stages:

1. Choosing parameters M and N and bounding the remainder R.
2. Evaluating the power sum S.
3. Evaluating the tail T (and the trivial term I).

In this section, we describe some algorithmic techniques that are useful at each stage. We sketch the computational complexities, but do not attempt to prove strict complexity bounds.

We assume that arithmetic on real and complex numbers is done using ball arithmetic [37], which essentially is floating-point arithmetic with the added automatic propagation of error bounds. This is probably the most reasonable approach: *a priori* floating-point error analysis would be overwhelming to do in full generality (an analysis of the floating-point error when evaluating $\zeta(s)$ for real s, with a partial analysis of the complex case, is given in [34]).

Using algorithms based on the Fast Fourier Transform (FFT), arithmetic operations on b-bit approximations of real or complex numbers can be done in time $\tilde{O}(b)$, where the $\tilde{O}$-notation suppresses logarithmic factors. This estimate also holds for division and evaluation of elementary functions.

Likewise, polynomials of degree n can be multiplied in $\tilde{O}(n)$ coefficient operations. Here some care is required: when doing arithmetic with polynomials that have approximate coefficients, the *accuracy* of the result can be much lower than the working precision, depending on the shape of the polynomials and the multiplication algorithm. If the coefficients vary in magnitude as $2^{\pm\tilde{O}(n)}$, we may need $\tilde{O}(n)$ bits of precision to get an accurate result, making the effective complexity $\tilde{O}(n^2)$. This issue is discussed further in [36].

Many operations can be reduced to fast multiplication. In particular, we will need the *binary splitting* algorithm: if a sequence c_n of integers (or polynomials) satisfies a suitable linear recurrence relation and its bit size (or degree) grows as $\tilde{O}(n)$, then we can use a balanced product tree to evaluate c_n using $\tilde{O}(n)$ bit (or coefficient) operations, versus $\tilde{O}(n^2)$ for repeated application of the recurrence relation [3, 19].

3.1 Evaluating the error bound

For a precision of P bits, we should choose $N \sim M \sim P$. A simple strategy is to do a binary search for an N that makes the error bound small enough when $M = cN$ where $c \approx 1$. This is sufficient for our present purposes, but more sophisticated approaches are possible. In particular, for evaluation at large heights in the critical strip, N should be larger than M.

Given complex balls for s and a, and integers N and M, we can evaluate the error bound (13) using ball arithmetic. The output is a power series with ball coefficients. The absolute value of each coefficient in this series should be added to the radius for the corresponding coefficient in $S + I + T \approx \zeta(s+x, a)$ at the end of the whole computation. If the assumptions that $\Re(a) + N > 1$ and $\Re(s) + 2M > 1$ are not satisfied for all points in the balls s and a, we set the error bounds for all coefficients to $+\infty$.

If we are computing D derivatives and D is large, the rising factorial $|(s+x)_{2M}|$ can be computed using binary splitting and the outer power series product in (13) can be done using fast polynomial multiplication, so that only $\tilde{O}(D+M)$ real number operations are required. Or, if D is small and M is large, $|(s+x)_{2M}|$ can be computed via the gamma function in time independent of M.

3.2 Evaluating the power sum

As a power series, the power sum S becomes $\sum_{k=0}^{N-1} (\sum_i c_i(k) x^i)$ where the coefficients $c_i(k)$ are given by (10). For $i \geq 1$, the coefficients can be computed using the recurrence

$$c_{i+1}(k) = -\frac{\log(a+k)}{i+1} c_i(k).$$

If we are computing D derivatives with a working precision of P bits, the complexity of evaluating the power sum is $\tilde{O}(NPD)$, or $\tilde{O}(N^2 D)$ if $N \sim P$. The computation is easy to parallelize by assigning a range of k values to each thread (for large D, a more memory-efficient method is to assign a range of i to each thread).

When evaluating the ordinary Riemann zeta function, i.e. when $a = 1$, and we just want to compute a small number of derivatives, we can speed up the power sum a bit. Writing the sum as $\sum_{k=1}^{N} f(k)$, the terms $f(k) = k^{-(s+x)}$ are completely multiplicative, i.e. $f(k_1 k_2) = f(k_1) f(k_2)$. This means that we only need to evaluate $f(k)$ from scratch when k is prime; when k is composite, a single multiplication is sufficient.

This method has two drawbacks: we have to store previously computed terms, which requires $O(NPD)$ space, and the power series multiplication $f(k_1) f(k_2)$ becomes more expensive than evaluating $f(k_1 k_2)$ from scratch for large D. For both reasons, this method is only useful when D is quite small (say $D \leq 4$).

Algorithm 1 Sieved summation of a completely multiplicative function

Input: A function f such that $f(jk) = f(j)f(k)$ for $j, k \in \mathbb{Z}_{\geq 1}$, and an integer $N \geq 1$
Output: $\sum_{k=1}^{N} f(k)$

$p \leftarrow 2^{\lfloor \log_2 N \rfloor}$ (largest power of two such that $p \leq N$)
$h \leftarrow 1$, $z \leftarrow 0$, $u \leftarrow 0$
$D = [\,]$ ▷ Build table of divisors
for $k \leftarrow 1;\ k \leq N;\ k \leftarrow k + 2$ **do**
    $D[k] \leftarrow 0$
for $k \leftarrow 3;\ k \leq \lfloor \sqrt{N} \rfloor;\ k \leftarrow k + 2$ **do**
    if $D[k] = 0$ **then**
        for $j \leftarrow k^2;\ j \leq N;\ j \leftarrow j + 2k$ **do**
            $D[j] \leftarrow k$
$F = [\,]$ ▷ Create initially empty cache of $f(k)$ values
$F[2] \leftarrow f(2)$
for $k \leftarrow 1;\ k \leq N;\ k \leftarrow k + 2$ **do**
    if $D[k] = 0$ **then** ▷ k is prime (or 1)
        $t \leftarrow f(k)$
    else
        $t \leftarrow F[D[k]]F[k/D[k]]$ ▷ k is composite
    if $3k \leq N$ **then**
        $F[k] \leftarrow t$ ▷ Store $f(k)$ for future use
    $u \leftarrow u + t$
    while $k = h$ and $p \neq 1$ **do** ▷ Horner's rule
        $z \leftarrow u + F[2]z$
        $p \leftarrow p/2$
        $h \leftarrow \lfloor N/p \rfloor$
        if h is even **then**
            $h \leftarrow h - 1$
return $u + F[2]z$

We can avoid some redundant work by collecting multiples of small primes. For example, if we extract all powers of two, $\sum_{k=1}^{10} f(k)$ can be written as

$$\begin{aligned} &[f(1) + f(3) + f(5) + f(7) + f(9)] \\ +f(2)\,&[f(1) + f(3) + f(5)] \\ +f(4)\,&[f(1)] \\ +f(8)\,&[f(1)]. \end{aligned}$$

This is a polynomial in $f(2)$ and can be evaluated from bottom to top using Horner's rule while progressively adding the terms in the brackets. Asymptotically, this reduces the number of multiplications and the size of the tables by half. Algorithm 1 implements this trick, and requires about $\pi(N) \approx N/\log N$ evaluations of $f(k)$ and $N/2$ multiplications, at the expense of having to store about $N/6$ function values plus a table of divisors of about $N/2$ integers. Constructing the table of divisors using the sieve of Eratosthenes requires $O(N \log \log N)$ integer operations, but this cost is negligible when multiplications and $f(k)$ evaluations are expensive. One could also extract other powers besides 2 (for example powers of 3 and 5), but this gives diminishing returns.

Another trick that can save time at high precision is to avoid computing the logarithms of integers from scratch. If q and p are nearby integers (such as

two consecutive primes) and we already know $\log(p)$, we can use the identity

$$\log(q) = \log(p) + 2\,\mathrm{atanh}\left(\frac{q-p}{q+p}\right)$$

and evaluate the inverse hyperbolic tangent by applying binary splitting to its Taylor series. This is not an asymptotic improvement over the best known algorithm for computing the logarithm (which uses the arithmetic-geometric mean), but likely faster in practice.

If $D \sim N$, we can improve the asymptotic complexity of computing S to $\tilde{O}(PD)$, which is softly optimal in the bit size of the output (the author thanks David Harvey for making this observation). The vector of coefficients $((-1)^k k![x^k]S)_{k=0}^{D-1}$ is given by $V^T Y$ where

$$V = \begin{bmatrix} 1 & \log a & \cdots & \log^{D-1} a \\ 1 & \log(a+1) & \cdots & \log^{D-1}(a+1) \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \log(a+N-1) & \cdots & \log^{D-1}(a+N-1) \end{bmatrix}, \quad Y = \begin{bmatrix} a^{-s} \\ (a+1)^{-s} \\ \vdots \\ (a+N-1)^{-s} \end{bmatrix}.$$

It is well known that multiplying a vector from the left by the Vandermonde matrix V can be done in $\tilde{O}(N)$ coefficent operations in what amounts to *fast multipoint evaluation*. Multiplying a vector from the left by V^T when $D \sim N$ then has essentially the same complexity according to the transposition principle (this problem is discussed, for example, in [15]).

3.3 Evaluating the tail

Except for the multiplication by Bernoulli numbers, the terms of the tail sum T satisfy a simple (hypergeometric) recurrence relation. If we are computing D derivatives with a working precision of P bits, the complexity of evaluating the tail by repeated application of the recurrence relation is $\tilde{O}(MPD)$, or $\tilde{O}(P^2 D)$ if $M \sim P$. We can do better if D is large, using binary splitting (Algorithm 2).

If $D \sim M$, the complexity with binary splitting is only $\tilde{O}(PD)$, or softly optimal in the bit size of the output. A drawback is that the intermediate products increase the memory consumption.

The Bernoulli numbers can of course be cached for repeated evaluation of the zeta function, but computing them the first time can be a bottleneck at high precision, at least if done naively. The first $2M$ Bernoulli numbers can be computed in quasi-optimal time $\tilde{O}(M^2)$, for example by using Newton iteration and fast polynomial multiplication to invert the power series $(e^x - 1)/x$. For most practical purposes, simpler algorithms with a bit complexity of $\tilde{O}(M^3)$ are adequate, however. Various algorithms are discussed in [21]. An interesting alternative, used in unpublished work of Bloemen [4], is to compute B_n via $\zeta(n)$ by direct approximation of the sum $\sum_{k=0}^{\infty} k^{-n}$, recycling the powers to process several n simultaneously.

Algorithm 2 Evaluation of the tail T using binary splitting

Input: $s, a \in \mathbb{C}$ and $N, M, D \in \mathbb{Z}_{\geq 1}$

Output: $T = \dfrac{1}{(a+N)^{s+x}} \left(\dfrac{1}{2} + \displaystyle\sum_{k=1}^{M} \dfrac{B_{2k}}{(2k)!} \dfrac{(s+x)_{2k-1}}{(a+N)^{2k-1}} \right) \in \mathbb{C}[[x]]/\langle x^D \rangle$

Let x denote the generator of $\mathbb{C}[[x]]/\langle x^D \rangle$
function BinSplit(j, k)
    if $j + 1 = k$ **then**
        if $j = 0$ **then**
            $P \leftarrow (s+x)/(2(a+N))$
        else
            $P \leftarrow \dfrac{(s+2j-1+x)(s+2j+x)}{(2j+1)(2j+2)(a+N)^2}$
        return $(P,\ B_{2j+2}P)$
    else
        $(P_1,\ R_1) \leftarrow$ BinSplit(j, $\lfloor (j+k)/2 \rfloor$)
        $(P_2,\ R_2) \leftarrow$ BinSplit($\lfloor (j+k)/2 \rfloor$, k)
        return $(P_1 P_2,\ R_1 + P_1 R_2)$ ▷ Polynomial multiplications mod x^D
$(P,\ R) \leftarrow$ BinSplit$(0, M)$
$T \leftarrow (a+N)^{-(s+x)}(1/2 + R)$ ▷ Polynomial multiplication mod x^D
return T

4 Implementation and benchmarks

We have implemented the Hurwitz zeta function for $s \in \mathbb{C}[[x]]$ and $a \in \mathbb{C}$ with rigorous error bounds as part of the Arb library [24] [1]. This library is written in C and is freely available under version 2 or later of the GNU General Public License. It uses the MPFR [17] library for evaluation of some elementary functions, GMP [12] or MPIR [13] for integer arithmetic, and FLINT [20] for polynomial arithmetic.

Our implementation incorporates most of the techniques discussed in the previous section, including optional parallelization of the power sum (we have not implemented the fast algorithm based on transposed multiplication by a transposed Vandermonde matrix). Bernoulli numbers are computed using the algorithm of Bloemen. Fast and numerically stable multiplication in $\mathbb{R}[x]$ and $\mathbb{C}[x]$ is implemented by rescaling polynomials and breaking them into segments with similarly-sized coefficients and computing the subproducts exactly in $\mathbb{Z}[x]$ (a simplified version of van der Hoeven's block multiplication algorithm [36]). Polynomial multiplication in $\mathbb{Z}[x]$ is done via FLINT which for large polynomials uses a Schönhage-Strassen FFT implementation by William Hart.

In the remainder of this section, we present the results of some computations done with our implementation. The computed data can be downloaded from the author's website [2].

[1] `http://fredrikj.net/arb`
[2] `http://fredrikj.net/math/hurwitz_zeta.html`

4.1 Computing zeros to high precision

For $n \geq 1$, let ρ_n denote the n-th smallest zero of $\zeta(s)$ with positive imaginary part. We assume that ρ_n is simple and has real part $1/2$. Using Newton's method, we can evaluate ρ_n to high precision nearly as fast as we can evaluate $\zeta(s)$ for s near ρ_n.

It is convenient to work with real numbers. The ordinate $t_n = \Im(\rho_n)$ is a simple zero of the real-valued function $Z(t) = e^{i\theta(t)}\zeta(1/2 + it)$ where

$$\theta(t) = \frac{\log \Gamma\left(\frac{2it+1}{4}\right) - \log \Gamma\left(\frac{-2it+1}{4}\right)}{2i} - \frac{\log \pi}{2} t.$$

We assume that we are given an isolating ball $B_0 = [m_0 - \varepsilon_0, m_0 + \varepsilon_0]$ such that $t_n \in B_0$ and $t_m \notin B_0, m \neq n$, and wish to compute t_n to high precision (finding such a ball for a given n is an interesting problem, but we do not consider it here).

Newton's method maps an approximation z_n of a root of a real analytic function $f(z)$ to a new approximation z_{n+1} via $z_{n+1} = z_n - f(z_n)/f'(z_n)$. Using Taylor's theorem, the error can be shown to satisfy

$$|\epsilon_{n+1}| = \frac{|f''(\xi_n)|}{2\,|f'(z_n)|}\,|\epsilon_n|^2$$

for some ξ_n between z_n and the root.

As a setup step, we evaluate $Z(s), Z'(s), Z''(s)$ (simultaneously using power series arithmetic) at $s = B_0$, and compute

$$C = \frac{\max |Z''(B_0)|}{2 \min |Z'(B_0)|}.$$

This only needs to be done at low precision.

Starting from an input ball $B_k = [m_k - \varepsilon_k, m_k + \varepsilon_k]$, one step with Newton's method gives an output ball $B_{k+1} = [m_{k+1} - \varepsilon_{k+1}, m_{k+1} + \varepsilon_{k+1}]$. The updated midpoint is given by

$$m_{k+1} = m_k - \frac{Z(m_k)}{Z'(m_k)} \tag{16}$$

where we evaluate $Z(m_k)$ and $Z'(m_k)$ simultaneously using power series arithmetic. The updated radius is given by $\varepsilon_{k+1} = \varepsilon'_{k+1} + C\varepsilon_k^2$ where ε'_{k+1} is the numerical error (or a bound thereof) resulting from evaluating (16) using finite-precision arithmetic. The new ball is valid as long as $B_{k+1} \subseteq B_k$ (if this does not hold, the algorithm fails and we need to start with a better B_0 or increase the working precision).

For best performance, the evaluation precision should be chosen so that $\varepsilon'_{k+1} \approx C\varepsilon_k^2$. In other words, for a target accuracy of p bits, the evaluations should be done at $\ldots, p/4, p/2, p$ bits, plus some guard bits.

As a benchmark problem, we compute an approximation $\tilde{\rho}_1$ of the first nontrivial zero $\rho_1 \approx 1/2 + 14.1347251417i$ and then evaluate $\zeta(\tilde{\rho}_1)$ to the same precision. We compare our implementation of the zeta function and the

Digits	mpmath		Mathematica		Arb	
	$\tilde{\rho}_1$	$\zeta(\tilde{\rho}_1)$	$\tilde{\rho}_1$	$\zeta(\tilde{\rho}_1)$	$\tilde{\rho}_1$	$\zeta(\tilde{\rho}_1)$
100	0.080	0.0031	0.044	0.012	0.012	0.0011
1000	7.1	0.24	11	1.6	0.18	0.05
10000	7035	252	5127	779	29	15
100000	-	-	-	-	6930	3476
303000	-	-	-	-	73225	31772

Table 1 Time in seconds to compute an approximation $\tilde{\rho}_1$ of the first nontrivial zero ρ_1 accurate to the specified number of decimal digits, and then to evaluate $\zeta(\tilde{\rho}_1)$ at the same precision. Computations were done on a 64-bit Intel Xeon E5-2650 2.00 GHz CPU.

root-refinement algorithm described above (starting from a double-precision isolating ball) with the `zetazero` and `zeta` functions provided in mpmath version 0.17 in Sage 5.10 [35] and the `ZetaZero` and `Zeta` functions provided in Mathematica 9.0. The results of this benchmark are shown in Table 1. At 10000 digits, our code for computing the zero is about two orders of magnitude faster than the other systems, and the subsequent single zeta evaluation is about one order of magnitude faster.

We have computed ρ_1 to 303000 digits, or slightly more than one million bits, which appears to be a record (a 20000-digit value is given in [31]). The computation used up to 62 GiB of memory for the sieved power sum and the storage of Bernoulli numbers up to B_{325328} (to attain even higher precision, the memory usage could be reduced by evaluating the power sum without sieving, perhaps using several CPUs in parallel, and not caching Bernoulli numbers).

4.2 Computing the Keiper-Li coefficients

Riemann's function $\xi(s) = \frac{1}{2}s(s-1)\pi^{-s/2}\Gamma(s/2)\zeta(s)$ satisfies the symmetric functional equation $\xi(s) = \xi(1-s)$. The coefficients $\{\lambda_n\}_{n=1}^{\infty}$ defined by

$$\log \xi\left(\frac{1}{1-x}\right) = \log \xi\left(\frac{x}{x-1}\right) = -\log 2 + \sum_{n=1}^{\infty} \lambda_n x^n$$

were introduced by Keiper [25], who noted that the truth of the Riemann hypothesis would imply that $\lambda_n > 0$ for all $n > 0$. In fact, Keiper observed that if one makes an assumption about the distribution of the zeros of $\zeta(s)$ that is even stronger than the Riemann hypothesis, the coefficients λ_n should behave as

$$\lambda_n \approx (1/2)\left(\log n - \log(2\pi) + \gamma - 1\right). \tag{17}$$

Keiper presented numerical evidence for this conjecture by computing λ_n up to $n = 7000$, showing that the approximation error appears to fluctuate increasingly close to zero. Some years later, Li proved [29] that the Riemann hypothesis actually is equivalent to the positivity of λ_n for all $n > 0$ (this reformulation of the Riemann hypothesis is known as Li's criterion). Recently, Arias de Reyna has proved that a certain precise statement of (17) also is equivalent to the Riemann hypothesis [11].

	$n = 1000$	$n = 10000$	$n = 100000$
1: Error bound	0.017	1.0	97
1: Power sum	0.048	47	65402
(1: Power sum, CPU time)	(0.65)	(693)	(1042210)
1: Bernoulli numbers	0.0020	0.19	59
1: Tail	0.058	11	1972
2: Series logarithm	0.047	8.5	1126
3: log $\Gamma(1+x)$ series	0.019	3.0	1610
4: Composition	0.022	4.1	593
Total wall time	0.23	84	71051
Peak RAM usage (MiB)	8	730	48700

Table 2 Elapsed time in seconds to evaluate the Keiper-Li coefficients $\lambda_0 \ldots \lambda_n$ with a working precision of $1.1n + 50$ bits, giving roughly $0.1n$ accurate bits. The computations were done on a multicore system with 64-bit Intel Xeon E7-8837 2.67 GHz CPUs (16 threads were used for the power sum, and all other parts were computed serially on a single core).

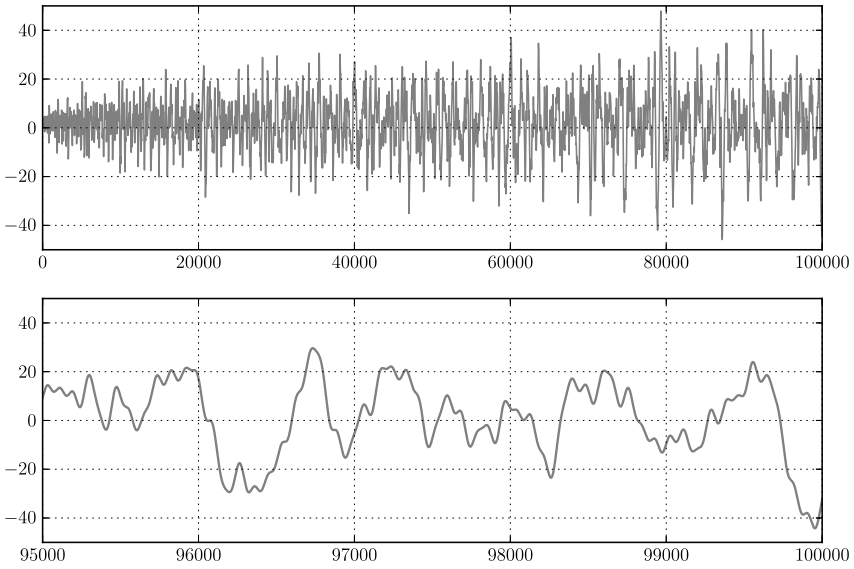

Fig. 1 Plot of $n\,(\lambda_n - (\log n - \log(2\pi) + \gamma - 1)/2)$.

A computation of the Keiper-Li coefficients up to $n = 100000$ shows agreement with Keiper's conjecture (and the Riemann hypothesis), as illustrated in Figure 1. We obtain $\lambda_{100000} = 4.625807824069022314094160 38\ldots$ (plus about 2900 more accurate digits), whereas (17) gives $\lambda_{100000} \approx 4.626132$. Empirically, we need a working precision of about n bits to determine λ_n accurately. A breakdown of the computation time to determine the signs of λ_n up to $n = 1000$, 10000 and 100000 is shown in Table 2.

Our computation of the Keiper-Li coefficients uses the formula

$$\log \xi(s) = \log(-\zeta(s)) + \log \Gamma\left(1 + \frac{s}{2}\right) + \log(1 - s) - \frac{s \log \pi}{2}$$

which we evaluate at $s = x \in \mathbb{R}[[x]]$. This arrangement of the terms avoids singularities and branch cuts at the expansion point. We carry out the following steps (plus some more trivial operations):

1. Computing the series expansion of $\zeta(s)$ at $s = 0$.
2. Computing the logarithm of a power series, i.e. $\log f(x) = \int f'(x)/f(x)dx$.
3. Computing the series expansion of $\log \Gamma(s)$ at $s = 1$, i.e. computing the sequence of values $\gamma, \zeta(2), \zeta(3), \zeta(4), \ldots$.
4. Finally, right-composing by $x/(x-1)$ to obtain the Keiper-Li coefficients.

Step 2 requires $O(M(n))$ arithmetic operations on real numbers, where $M(n)$ denotes the complexity of polynomial multiplication. We use a hybrid algorithm to compute the integer zeta values in step 3; the details are beyond the scope of the present paper.

There is a very fast way to perform step 4. For $f = \sum_{k=0}^{\infty} a_k x^k \in \mathbb{C}[[x]]$, the binomial (or Euler) transform $T\colon \mathbb{C}[[x]] \to \mathbb{C}[[x]]$ is defined by

$$T[f(x)] = \frac{1}{1-x} f\left(\frac{x}{x-1}\right) = \sum_{n=0}^{\infty} \left(\sum_{k=0}^{n} (-1)^k \binom{n}{k} a_k \right) x^n.$$

We have

$$f\left(\frac{x}{x-1}\right) = a_0 + xT\left[\frac{a_0 - f}{x}\right].$$

If $B\colon \mathbb{C}[[x]] \to \mathbb{C}[[x]]$ denotes the Borel transform

$$B\left[\sum_{k=0}^{\infty} a_k x^k\right] = \sum_{k=0}^{\infty} \frac{a_k}{k!} x^k,$$

then (see [18]) $T[f(x)] = B^{-1}[e^x B[f(-x)]]$. This identity gives an algorithm for evaluating the composition which requires only $M(n) + O(n)$ coefficient operations. Moreover, this algorithm is numerically stable (in the sense that it does not significantly increase errors from the input when using ball arithmetic), provided that a numerically stable polynomial multiplication algorithm is used.

The composition could also be carried out using various generic algorithms for composition of power series. We tested three other algorithms, and found them to perform much worse:

- Horner's rule is slow (requiring about $nM(n)$ operations) and is numerically unsatisfactory in the sense that it gives extremely poor error bounds with ball arithmetic.
- The Brent-Kung algorithm based on matrix multiplication [9] turns out to give adequate error bounds, but uses about $O(n^{1/2}M(n) + n^2)$ operations which still is expensive for large n.

– We also tried binary splitting: to evaluate $f(p/q)$ where f is a power series and p and q are polynomials, we recursively split the evaluation in half and keep numerator and denominator polynomials separated. In the end, we perform a single power series division. This only costs $O(M(n)\log n)$ operations, but turns out to be numerically unstable. It would be of independent interest to investigate whether this algorithm can be modified to avoid the stability problem.

4.3 Computing the Stieltjes constants

The generalized Stieltjes constants $\gamma_n(a)$ are defined by

$$\zeta(s,a) = \frac{1}{s-1} + \sum_{n=0}^{\infty} \frac{(-1)^n}{n!}\gamma_n(a)\,(s-1)^n.$$

The "usual" Stieltjes constants are $\gamma_n(1) = \gamma_n$, and $\gamma_0 = \gamma \approx 0.577216$ is Euler's constant. The Stieltjes constants were first studied over a century ago. Some historical notes and numerical values of γ_n for $n \leq 20$ are given in [6]. Keiper [25] provides a method for computing the Stieltjes constants based on numerical integration and recurrence relations, and lists various γ_n up to $n = 150$. Keiper's algorithm is implemented in Mathematica [23].

More recently, Kreminski [28] has given an algorithm for the Stieltjes constants, also based on numerical integration but different from Keiper's. He reports having computed γ_n to a few thousand digits for all $n \leq 10000$, and provides further isolated values up to γ_{50000} (accurate to 1000 digits) as well as tables of $\gamma_n(a)$ with various $a \neq 1$.

The best proven bounds for the Stieltjes constants appear to be very pessimistic (see for example [1]). In a recent paper, Knessl and Coffey [27] give an asymptotic approximation formula that seems to be very accurate even for small n, having the form

$$\gamma_n \approx Bn^{-1/2}e^{An}\cos(an+b) \tag{18}$$

where A, B, a, b are functions that depend weakly on n. Notably, this formula captures both the asymptotic growth and the oscillation pattern of γ_n. Based on numerical computations done with Mathematica, Knessl and Coffey observe that (18) correctly predicts the sign of γ_n up to at least $n = 35000$ with the single exception of $n = 137$.

Our implementation immediately gives the generalized Stieltjes constants by computing the series expansion of $\zeta(s,a) - 1/(s-1)$ at $s = 1$ using (11). The costs are similar to those for computing the Keiper-Li coefficients: due to ill-conditioning, it appears that we need about $n+p$ bits of precision to determine γ_n with p bits of accuracy. This makes our method somewhat unattractive for computing just a few digits of γ_n when n is large, but reasonably good if we want a large number of digits. Our method is also useful if we want to compute a table of all the values $\gamma_0, \ldots, \gamma_n$ simultaneously.

For example, we can compute γ_n for all $n \leq 1000$ to 1000-digit accuracy in just over 10 seconds on a single CPU. Computing the single coefficient γ_{1000} to 1000-digit accuracy with Mathematica 9.0 takes 80 seconds, with an estimated 20 hours required for all $n \leq 1000$. Thus our implementation is nearly four orders of magnitude faster. We can compute a table of accurate values of γ_n for all $n \leq 10000$ in a few minutes on an ordinary workstation with around one GiB of memory.

We have computed all γ_n up to $n = 100000$ using a working precision of 125050 bits, resulting in an accuracy from about 37640 decimal digits for γ_0 to about 10860 accurate digits for γ_{100000}. The computation took 26 hours on a multicore system with 16 threads utilized for the power sum, with a peak memory consumption of about 80 GiB during the binary splitting evaluation of the tail.

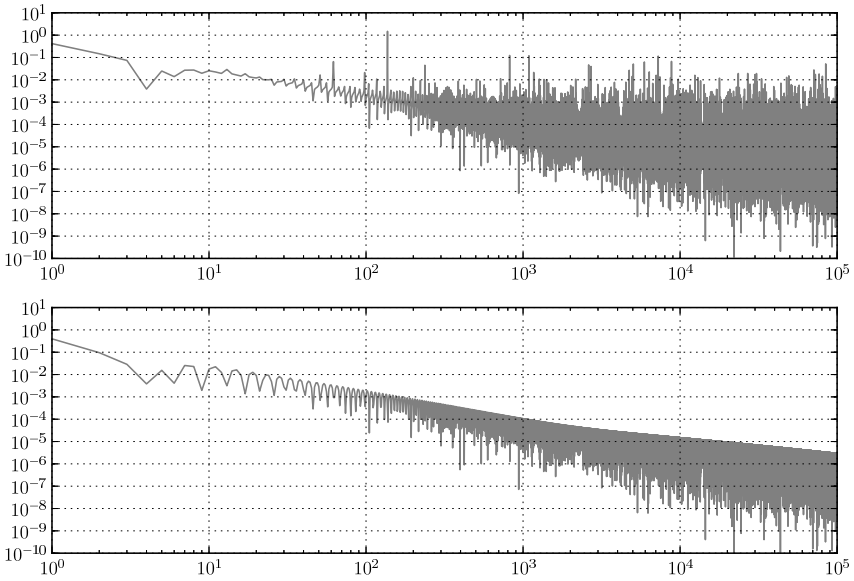

Fig. 2 Top: plot of the relative error $|\gamma_n - \tilde{\gamma}_n|/|\gamma_n|$ of the Knessl-Coffey approximation $\tilde{\gamma}_n = Bn^{-1/2}e^{An}\cos(an+b)$ for the Stieltjes constants. Bottom: plot of the normalized error $|\cos(an+b)||\gamma_n - \tilde{\gamma}_n|/|\gamma_n|$.

As shown in Figure 2, the Knessl-Coffey approximation (18) is very accurate in practice, approaching six correct digits on average when $n \approx 10^5$. Our computation gives $\gamma_{100000} = 1.991927306312541095658\ldots \times 10^{83432}$, while (18) gives $\gamma_n \approx 1.9919333 \times 10^{83432}$.

The relative error of (18) can be large when the oscillatory factor $\cos(an+b)$ is small. For example, this factor is about 0.000027 when $n = 84589$, and in this case (18) only gives one correct digit (our computation confirms that

$n = 137$ is the only instance for $n \leq 100000$ where (18) gives the wrong sign). Nonetheless, the worst-case error of (18) appears to decay smoothly when adjusted for the oscillation, as illustrated in the bottom of Figure 2.

In [26], Knessl and Coffey extend their asymptotic approximation formula to $a \neq 1$. Their approximation gives, for example, the estimate

$$\gamma_{50000}(1+i) \approx (1.0324943 - 1.4419586i) \times 10^{39732}$$

while we compute (rounded to 15 decimal digits)

$$\gamma_{50000}(1+i) = (1.03250208743188 - 1.44196255284053i) \times 10^{39732}.$$

We emphasize that our implementation computes $\gamma_n(a)$ with proved error bounds, while the other cited works and implementations (to our knowledge) depend on heuristic error estimates.

We have not yet implemented a function for computing isolated Stieltjes constants of large index; this would have roughly the same running time as the evaluation of the tail (since only a single derivative of the power sum would have to be computed). The memory consumption is highest when evaluating the tail, and would therefore remain the same.

5 Discussion

One direction for further work would be to improve the error bounds for large $|a|$ and to investigate strategies for selecting N and M optimally, particularly when the number of derivatives is large. It would also be interesting to investigate parallelization of the tail sum, or look for ways to evaluate a single derivative of high order of the tail in a memory-efficient way. Further constant-factor improvements are possible in an implementation, for example by reducing the precision of terms that have small magnitude (rather than naively performing all operations at the same precision). It would also be interesting to implement the asymptotically fast algorithm for the power sum when computing many derivatives.

Finally, it would be interesting to compare the efficiency of the Euler-Maclaurin formula with other approaches to evaluating the Hurwitz zeta function such as the algorithms of Borwein [8], Vepštas [38] and Coffey [10].

Acknowledgements The author was supported by the Austrian Science Fund (FWF) grant Y464-N18. The author thanks Paul Zimmermann, Ricky Farr, and the anonymous reviewer for pointing out errors and suggesting improvements. A version of this article also appears as a section in the author's PhD thesis.

References

1. J. A. Adell. Estimates of generalized Stieltjes constants with a quasi-geometric rate of decay. *Proceedings of the Royal Society A*, 468:1356–1370, 2012.

2. D. H. Bailey and J. M. Borwein. Experimental mathematics: recent developments and future outlook. In B. Engquist, W. Schmid, and P. W. Michor, editors, *Mathematics Unlimited – 2001 and Beyond*, pages 51–66. Springer, 2000.
3. D. J. Bernstein. Fast multiplication and its applications. *Algorithmic Number Theory*, 44:325–384, 2008.
4. R. Bloemen. Even faster $\zeta(2n)$ calculation!, 2009. `http://remcobloemen.nl/2009/11/even-faster-zeta-calculation.html`.
5. A. I. Bogolubsky and S. L. Skorokhodov. Fast evaluation of the hypergeometric function ${}_pF_{p-1}(a;b;z)$ at the singular point $z = 1$ by means of the Hurwitz zeta function $\zeta(\alpha, s)$. *Programming and Computer Software*, 32(3):145–153, 2006.
6. J. Bohman and C-E. Fröberg. The Stieltjes function – definition and properties. *Mathematics of Computation*, 51(183):281–289, 1988.
7. J. M. Borwein, D. M. Bradley, and R. E. Crandall. Computational strategies for the Riemann zeta function. *Journal of Computational and Applied Mathematics*, 121:247–296, 2000.
8. P. Borwein. An efficient algorithm for the Riemann zeta function. *Canadian Mathematical Society Conference Proceedings*, 27:29–34, 2000.
9. R. P. Brent and H. T. Kung. Fast algorithms for manipulating formal power series. *Journal of the ACM*, 25(4):581–595, 1978.
10. M. W. Coffey. An efficient algorithm for the Hurwitz zeta and related functions. *Journal of Computational and Applied Mathematics*, 225(2):338–346, 2009.
11. J. Arias de Reyna. Asymptotics of Keiper-Li coefficients. *Functiones et Approximatio Commentarii Mathematici*, 45(1):7–21, 2011.
12. The GMP development team. GMP: The GNU multiple precision arithmetic library. `http://www.gmplib.org`.
13. The MPIR development team. MPIR: Multiple Precision Integers and Rationals. `http://www.mpir.org`.
14. H. M. Edwards. *Riemann's zeta function*. Academic Press, 1974.
15. T. Finck, G. Heinig, and K. Rost. An inversion formula and fast algorithms for Cauchy-Vandermonde matrices. *Linear algebra and its applications*, 183:179–191, 1993.
16. P. Flajolet and I. Vardi. Zeta function expansions of classical constants. Unpublished manuscript, `http://algo.inria.fr/flajolet/Publications/landau.ps`, 1996.
17. L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Transactions on Mathematical Software*, 33(2):13:1–13:15, June 2007. `http://mpfr.org`.
18. H. Gould. Series transformations for finding recurrences for sequences. *Fibonacci Quarterly*, 28:166–171, 1990.
19. B. Haible and T. Papanikolaou. Fast multiprecision evaluation of series of rational numbers. In J. P. Buhler, editor, *Algorithmic Number Theory: Third International Symposium*, volume 1423, pages 338–350. Springer, 1998.
20. W. B. Hart. Fast Library for Number Theory: An Introduction. In *Proceedings of the Third international congress conference on Mathematical software*, ICMS'10, pages 88–91, Berlin, Heidelberg, 2010. Springer-Verlag. `http://flintlib.org`.
21. D. Harvey and R. P. Brent. Fast computation of Bernoulli, tangent and secant numbers. *Springer Proceedings in Mathematics & Statistics*, 50:127–142, 2013. `http://arxiv.org/abs/1108.0286`.
22. G. Hiary. Fast methods to compute the Riemann zeta function. *Annals of mathematics*, 174:891–946, 2011.
23. Wofram Research Inc. Some notes on internal implementation (section of the online documentation for Mathematica 9.0). `http://reference.wolfram.com/mathematica/tutorial/SomeNotesOnInternalImplementation.html`, 2013.
24. F. Johansson. Arb: a C library for ball arithmetic. *ACM Communications in Computer Algebra*, 47:166–169, December 2013.
25. J. B. Keiper. Power series expansions of Riemann's ξ function. *Mathematics of Computation*, 58(198):765–773, 1992.
26. C. Knessl and M. Coffey. An asymptotic form for the Stieltjes constants $\gamma_k(a)$ and for a sum $S_\gamma(n)$ appearing under the Li criterion. *Mathematics of Computation*, 80(276):2197–2217, 2011.

27. C. Knessl and M. Coffey. An effective asymptotic formula for the Stieltjes constants. *Mathematics of Computation*, 80(273):379–386, 2011.
28. R. Kreminski. Newton-Cotes integration for approximating Stieltjes (generalized Euler) constants. *Mathematics of Computation*, 72(243):1379–1397, 2003.
29. X-J. Li. The positivity of a sequence of numbers and the Riemann Hypothesis. *Journal of Number Theory*, 65(2):325–333, 1997.
30. Y. Matiyasevich. An artless method for calculating approximate values of zeros of Riemann's zeta function, 2012. `http://logic.pdmi.ras.ru/~yumat/personaljournal/artlessmethod/`.
31. Y. Matiyasevich and G. Beliakov. Zeroes of Riemann's zeta function on the critical line with 20000 decimal digits accuracy, 2011. `http://dro.deakin.edu.au/view/DU:30051725?print_friendly=true`.
32. A. M. Odlyzko and A. Schönhage. Fast algorithms for multiple evaluations of the Riemann zeta function. *Transactions of the American Mathematical Society*, 309(2):797–809, 1988.
33. F. W. J. Olver. *Asymptotics and Special Functions*. A K Peters, Wellesley, MA, 1997.
34. Y.-F.S Pétermann and J-L. Rémy. Arbitrary precision error analysis for computing $\zeta(s)$ with the Cohen-Olivier algorithm: complete description of the real case and preliminary report on the general case. Rapport de recherche RR-5852, INRIA, 2006.
35. W. A. Stein et al. *Sage Mathematics Software*. The Sage Development Team, 2013. `http://www.sagemath.org`.
36. J. van der Hoeven. Making fast multiplication of polynomials numerically stable. Technical Report 2008-02, Université Paris-Sud, Orsay, France, 2008.
37. J. van der Hoeven. Ball arithmetic. Technical report, HAL, 2009. `http://hal.archives-ouvertes.fr/hal-00432152/fr/`.
38. L. Vepštas. An efficient algorithm for accelerating the convergence of oscillatory series, useful for computing the polylogarithm and Hurwitz zeta functions. *Numerical Algorithms*, 47(3):211–252, 2008.