UNIVERSITAT POLITÈCNICA DE CATALUNYA

Department of Computer Science

High-precision Computation of Uniform Asymptotic Expansions for Special Functions

Guillermo Navas-Palencia

Supervisor: Argimiro Arratia Quesada

A dissertation submitted in fulfillment of the requirements for the degree of Doctor of Philosophy in Computing

May 7, 2019

Abstract

In this dissertation, we investigate new methods to obtain uniform asymptotic expansions for the numerical evaluation of special functions to high-precision. We shall first present the theoretical and computational fundamental aspects required for the development and ultimately implementation of such methods. Applying some of these methods, we obtain efficient new convergent and uniform expansions for numerically evaluating the confluent hypergeometric functions ${}_1F_1(a;b;z)$ and $U(a,b,z)$, and the Lerch transcendent $\Phi(z,s,a)$ at high-precision. In addition, we also investigate a new scheme of computation for the generalized exponential integral $E_\nu(z)$, obtaining one of the fastest and most robust implementations in double-precision arithmetic.

In this work, we aim to combine new developments in asymptotic analysis with fast and effective open-source implementations. These implementations are comparable and often faster than current open-source and commercial state-of-the-art software for the evaluation of special functions.

Acknowledgements

First, I would like to express my gratitude to my supervisor Argimiro Arratia for his support, encourage and guidance throughout this work, and for letting me choose my research path with full freedom. I also thank his assistance with administrative matters, especially in periods abroad.

I am grateful to Javier Segura and Amparo Gil from Universidad de Cantabria for inviting me for a research stay and for their inspirational work in special functions, and the Ministerio de Economía, Industria y Competitividad for the financial support, project APCOM (TIN2014-57226-P), during the stay. Special thanks go to my colleges at Numerical Algorithms Group from whom I learned many important aspects in the development of numerical software, and to Fredrik Johansson for fruitful and interesting discussions, and for his remarkable work developing tools for arbitrary-precision arithmetic.

Finally, I would like to thank my mother for her patience and support, and Regina for her encouragement, appreciation and understanding of the effort required to complete this work.

Contents

List of Figures

List of Tables

Introduction

This thesis studies the development of efficient algorithms for the numerical evaluation of special functions to fixed and high-precision. Special functions are a class of well-studied mathematical functions that are formally defined as a solution of differential, integrals or functional equation. Amongst the vast majority of applications, special functions are used for obtaining closed-form expressions with analytical properties for the solution of problems arising in physics, engineering, mathematics, and statistics.

The study of effective algorithms for evaluating special functions has been considered by many authors in the last two centuries. More recently, the use of arbitrary-precision arithmetic has allowed the computation of special functions to very high-precision, awakening special interest in the fields of number theory and combinatorics. On the other hand, on many occasions, applications do not have such requirements in terms of precision but are demanding in terms of speed. Consequently, there is also a need to develop fast and robust algorithms in moderate precision, which turns out to be a challenging problem given the limitations of working in lower precision, requiring the use of techniques to reduce rounding effects and numerical instability issues.

The objective of this thesis is to contribute to the field through the development of robust and efficient algorithms for several special functions. We present these new algorithms from a performance-oriented approach rather than merely theoretical, with the aim to reduce the existing gap between theory and software implementations. This thesis is divided into five chapters.

Chapter 1 serves to introduce notation and revisit fundamental concepts in asymptotic analysis and numerical methods used throughout this work.

Chapter 2 focuses on algorithmic aspects to develop numerical software for the computation of special functions. This chapter reviews standard approaches and includes an extensive treatment of best practices based on the author's experience in the numerical software industry. In addition, a new numerical library for the evaluation of special functions developed while preparing this work is presented.

Chapter 3 explores different numerical methods in order to develop an algorithm for the generalized exponential integral, which will be used for several important special cases and uniform asymptotic expansions for confluent hypergeometric functions. The developed algorithm, unlike most of the available codes, considers real and not just integer parameter. Furthermore, we present various improvements for relevant asymptotic expansions and a new asymptotic expansion for large parameters in terms of Bell polynomials, for which a new uniform upper bound is calculated. Finally, this algorithm is compared with implementations available in the most widely used numerical libraries, showing a superior accuracy and usually running significantly faster.

Chapter 4 starts with an introduction to the generalized hypergeometric function, particularly the confluent hypergeometric functions, the methods of computation and their role in physics, mathematics, and statistics. In particular, we derive two new results for the beta and Gumbel distribution with applications in Bayesian statistics and the assessment of classification models' performance in credit risk

modelling. Next, we present an efficient algorithm for the computation of the confluent hypergeometric functions when the imaginary part of the parameters and argument is large. The algorithm is based on the application of the steepest descent method, applied to a suitable representation of the confluent hypergeometric integrals as a highly oscillatory integral, which is then integrated by using various numerical quadrature methods. The performance of the algorithm is compared with open-source and commercial software solutions with arbitrary precision. Our algorithm clearly outperforms all the available open-source codes in floating-point arithmetic and achieves high-precision (close to machine-precision) in both real and imaginary parts. As stated before, special functions have many applications in different areas and in this work our motivation comes from the need of accurate computation of the characteristic function of the beta distribution, which is required in several financial models, for example, modelling the loss given default in the context of portfolio credit risk. In the third section, we present a method of high-precision computation of the confluent hypergeometric functions using an effective computational approach of what we termed Franklin-Friedman expansions, originated in 1957. These expansions are convergent under mild conditions of the involved amplitude function and for some interesting cases the coefficients can be rapidly computed, thus providing a viable alternative to the conventional dichotomy between series expansion and asymptotic expansions widely used in practice. Our uniform asymptotic expansion is implemented and extensively tested in different regimes of the parameters and compared with recently investigated convergent and uniform asymptotic expansions.

Finally, chapter 5 deals with the evaluation of the Lerch transcendent, which serves as a unified framework for the study of various particular cases of special functions in number theory. After a first section introducing some functions in analytic number theory, the second section examines the use of Euler-Maclaurin formula and new derived uniform asymptotic expansions for the numerical evaluation of the Lerch transcendent for complex parameters and argument to arbitrary-precision. A detailed analysis of these expansions is accompanied by rigorous error bounds and algorithmic details to achieve high performance implementations. Our main contribution is an algorithm for large and small values of the parameters and argument, showing a superior performance compared to current state-of-the-art codes.

Chapters 3, 4 and 5 of this thesis are primarily based on a series of published papers and preprints, which have been adjusted to consolidate the presentation. The list of papers is:

- Chapter 3:

    1. G. Navas-Palencia. *Fast and accurate algorithm for the generalized exponential integral $E_\nu(x)$ for positive real order*. Numerical Algorithms, 77(2):603-630, 2018. [118].

- Chapter 4:

    1. G. Navas-Palencia and A. Arratia. *On the computation of confluent hypergeometric functions for large imaginary part of parameters b and z*. Lecture Notes in Computer Science, 9725:241–248, 2016. [120].

2. G. Navas-Palencia. *High-precision computation of the confluent hypergeometric functions via Franklin-Friedman expansion*. Advances in Computational Mathematics, 44(3):841-859, 2018. [119].

- Chapter 5:

1. G. Navas-Palencia. *Numerical methods and arbitrary-precision computation of the Lerch transcendent*. Preprint 2018, submitted to Numerical Algorithms.

Finally, work from this thesis was presented at the following conferences:

1. 5^{th} International Congress on Mathematical Software, 11-14 July 2016. Zuse Institute Berlin (ZIB), Germany. Talk: *On the computation of confluent hypergeometric functions for large imaginary part of the parameters b and z.*

2. FOCM 2017 - Foundations of Computational Mathematics, 10-19 July 2017. Universitat de Barcelona, Spain. Poster: *Fast and accurate algorithm for the generalized exponential integral $E_\nu(x)$ for positive real order.*

3. International Conference on Computational Science and Engineering 2017, 23-25 October 2017. Oslo, Norway. Talk: *GNSTLIB: a new numerical library for the evaluation of mathematical functions.*

Chapter 1

Analytic and Numerical Methods for Special Functions

This first chapter introduces standard methods for evaluating special functions. Here we merely provide a short discussion about methods used in subsequent chapters. The primary purpose of this introductory chapter is not to give a precise description of these concepts but rather references to the non-versed reader, to whom we refer to the classical books [39, 125] and internet resources [43].

1.1 Analytic methods and asymptotic expansions

1.1.1 Introduction

As customary, we introduce asymptotic estimates presenting the big O-symbol, denoted by $\mathcal{O}$. Let us consider two functions $f(z)$ and $g(z)$ defined on a complex domain $\mathcal{D}$, then

$$f(z) = \mathcal{O}(g(z)), \quad z \in \mathcal{D},$$

means that $f(z)$ is bounded by a constant M multiple of $g(z)$, which can be written as $|f(z)| \leq M|g(z)|$, $\forall z \in \mathcal{D}$. Another standard symbol in asymptotic estimates is the little o-symbol, which is used to express the limit $\lim_{z\to\infty} f(z)/g(z) = 0$ as

$$f(z) = o(g(z)), \quad z \to \infty, \quad z \in \mathcal{D}.$$

We also express that functions are asymptotically equivalent, meaning that the limit $\lim_{z\to\infty} f(z)/g(z) = 1$, using the notation

$$f(z) \sim g(z), \quad z \to \infty, \quad z \in \mathcal{D}.$$

Let us consider a formal series expansion representation

$$f(z) = \sum_{k=0}^{\infty} a_k z^k.$$

For $f(z)$ having singular points the series converges for all $|z| < r$, where r is positive number and coefficients a_k are real or complex numbers. Less formally, the series $f(z)$ converges if the sequence of partial sums $\sum_{k=0}^{n} a_k z^k$ converges as $n \to \infty$. Otherwise, we say that the series diverges. The series converges absolutely if $\sum_{k=0}^{\infty} |a_k z^k|$ converges.

Convergence tests are tools to determine if a given infinite series converges, conditionally converges, absolutely converges or diverges. Two well-known tests are the comparison test and the ratio test. The comparison test compares a given series with a simpler series whose convergence properties are already known. Given a series $\sum_{k=0}^{\infty} b_k z^k$ with $b_k \geq 0$ that converges, if $|a_k| < b_k$, then $\sum_{k=0}^{\infty} a_k z^k$ converges absolutely. The ratio test, also known as D'Alembert criterion, is a test for determining the radius of convergence for power series of the form $\sum_{k=0}^{\infty} a_k$ with positive terms a_k, defined as

$$r = \lim_{k\to\infty} \frac{a_{k+1}}{a_k}.$$

Then, if $r < 1$, the series converges. If $r > 1$ or $r = \infty$, the series diverges. Otherwise, the ratio test is inclusive, the series may converge or diverge.

Let us now consider the Poincaré asymptotic expansion of the form

$$f(z) \sim \sum_{k=0}^{\infty} \frac{a_k}{z^k}, \quad z \to \infty,$$

which is also frequently written with a remainder term, which should hold $\forall K$.

$$f(z) = \sum_{k=0}^{K-1} \frac{a_k}{z^k} + \mathcal{O}(z^{-K}), \quad z \to \infty,$$

for z in an unbounded domain $\mathcal{D}$. The asymptotic series might accurately approximate the true value of $f(z)$ for sufficiently large values of z, but we do not assume convergence as the number of terms increases.

1.1.2 Asymptotic methods for integrals

In what follows we give a brief overview of classical methods usually employed to develop asymptotic expansions of integrals: Watson's lemma, Laplace's method, and the saddle point method. These methods are part of the standard approach to obtain asymptotic expansions for Laplace-type and contour integrals. For other methods such as the method of stationary phase or the Bleistein's method, we refer to [157] and [125], where proofs and detailed analysis of these and other asymptotic methods for integrals are provided.

Watson's lemma

The Watson's lemma is one the most useful results from the theory of asymptotics for integrals, being often the first option for deriving asymptotic expansions for special functions, mainly due to its simplicity and effectiveness. Let us consider a Laplace-type integral of the form

$$F(z, \mu) = \int_0^{\infty} t^{\mu-1} e^{-zt} f(t)\, dt,$$

with $\Re(z) > 0$ large and $\Re(\mu) > 0$ fixed, and $f(t)$ has a finite number of discontinuities. If $F(z, \mu)$ converges for sufficiently large values of $\Re(z)$, then

$$F(z,\mu) \sim \sum_{k=0}^{\infty} a_k \frac{\Gamma(k+\mu)}{z^{k+\mu}}, \quad z \to \infty, \ |\mathrm{ph}\, z| < \frac{\pi}{2},$$

where $\Gamma(z)$ is the gamma function and a_k are the coefficient of the expansion of $f(t)$ at $t = 0$, also known as Maclaurin expansion, given by

$$f(t) \sim \sum_{k=0}^{\infty} a_k t^k.$$

Laplace's method and saddle point method

The Laplace's method is a generalization of Watson's lemma applicable to general integrals of the form

$$F(z) = \int_a^b e^{-zp(t)} q(t)\, dt\,.$$

We assume that $p(t)$ has a simple saddle point, i.e., the point t_0 such that $p'(t_0) = 0$, with $p''(t_0) \neq 0$, thus the local minimum of $p(t)$. For large z the main contribution is located in a neighbourhood of $t - t_0$. Assuming that $p(t)$ and $q(t)$ are continuous in the vicinity of t_0 and independent of z, if $F(z)$ converges absolutely for sufficiently large values of z then

$$F(z) \sim \sqrt{\frac{\pi}{z p_2}} e^{-zp(t_0)} \sum_{k=0}^{\infty} a_{2k} \frac{(1/2)_k}{z^k}, \quad z \to \infty,$$

where $(a)_k$ is the Pochhammer symbol. Formulas for the coefficients a_{2k} require the coefficients of the Taylor expansion of $p(t)$ and $q(t)$ at the saddle point $t = t_0$. Thus,

$$p(t) = \sum_{k=0}^{\infty} p_k (t - t_0)^k, \quad q(t) = \sum_{k=0}^{\infty} q_k (t - t_0)^k.$$

The first two coefficients a_0 and a_2 are

$$a_0 = q_0, \quad a_2 = \frac{1}{p_2}\left(q_2 - \frac{3p_3 q_1}{2p_2} + \left(\frac{15p_3^2}{8p_2^2} - \frac{3p_4}{2p_2}\right) q_0\right).$$

The saddle point method deals with contour integrals in the form

$$F(z) = \int_{\mathcal{C}} e^{-zp(t)} q(t)\, dt,$$

for large z. The integral is taken along a path $\mathcal{C}$ in a domain where $p(t)$ and $q(t)$ are analytic, avoiding the singularities of the integrand. The idea is to modify the contour of integration $\mathcal{C}$ through a saddle point such that the imaginary part of the dominant part of the integral is constant, thus obtaining a steepest descent path. The resulting integral may ultimately be estimated by using one of the previous methods. This technique has been extensively applied to numerical integration of

oscillating integrals, see Section 4.2. Finally, a unified formulation of Laplace's and steepest descent method is presented in [50].

A closely related method is the determination of saddle point bounds, which are commonly derived in analytic combinatorics. An elementary saddle point bound [53, §VIII.2] satisfies that

$$\left|\int_A^B f(t)\,dt\right| \leq |C_0||f'(t_0)|, \quad f'(t_0) = 0,$$

where $|C_0|$ are saddle point paths made of arcs connecting A to B through t_0. Thus, $|C_0| = |t_0 - A| + |B - t_0|$. This method is often used throughout this work, for example in Chapter 3, where we develop an effective bound for the Bell polynomials.

1.1.3 Uniform expansions for Laplace-type integrals

Besides the classical Watson's lemma, the method of Laplace and the saddle point methods, several analytic methods have been devised to obtain uniformly convergent asymptotic expansions. These methods generally require a thorough analysis of the coefficients involved, but they form a powerful set of methods to obtain uniform expansion valid for extended regimes of the parameters. These methods serve to complement the standard dichotomy between series expansion and asymptotic expansion. A detailed analysis of uniform methods applicable to Laplace-type integrals is provided in [157]. Here, we briefly present two of these methods that we shall used throughout this work.

The Franklin-Friedman expansion [55, 119] is a suitable method to obtain convergent uniform asymptotic expansions. This method replaces the Maclaurin expansion in Watson's lemma by a type of interpolation process. See Section 4.3 for an extensive detailed analysis of the coefficients of the expansion.

The vanishing saddle-point method [156] is an alternative method to the direct application of the Watson's lemma for Laplace-type integrals, where parameters and argument can be simultaneously large. The resulting expansion is expressible in terms of Tricomi-Carlitz polynomials [157, §24.3], which satisfy a three-term recursive relation. A comparison between the Franklin-Friedman expansion and the vanishing saddle point method is carried out in Section 4.3, to evaluate one of the confluent hypergeometric functions.

1.2 Numerical Methods

This Section covers standard alternative methods to the direct evaluation of a convergent or asymptotic series expansion for computing special functions. These methods are needed when the coefficients of the expansion cannot be computed efficiently, or the optimal truncation point of the asymptotic expansion is reached before achieving the desired accuracy. The excellent and already classic book *Numerical Methods for Special Functions* [61] deals with numerical methods and algorithms to approximate special functions.

1.2.1 Quadrature methods

Many special functions can be expressed in terms of several integral representations. This favours the development of asymptotic expansions using some of the techniques described in the previous Section. On the other hand, this also permits the use of quadrature methods when those asymptotic expansions have difficult coefficients and cannot be computed efficiently. The use of standard quadrature methods for the computation of special functions is extensively studied in[61, §5], including a discussion about the most suitable quadrature rule considering the analytical properties of the integrand.

Among the variety of numerical integration methods, the most efficient methods used in practice are the extended trapezoidal rule [161], the Gauss quadrature (Gauss-Legendre, Gauss-Laguerre, Gauss-Hermite, Gauss-Kronrod among other variants) and the double-exponential integration, also known as tanh-sinh quadrature, [150, 110]. The double-exponential quadrature is often preferred when evaluating integrals at high-precision since computing the nodes and weights for the integration is cheap and can appropiately handle singularities at endpoints of the interval of integration [6]. The double-exponential quadrature of an integrand $f(x)$ over $[-1,1]$ is given by

$$\int_{-1}^{1} f(x)\,dx = \int_{-\infty}^{\infty} f(g(t))g'(t)\,dt \approx \sum_{j=-N}^{N} w_j f(x_j),$$

for $h > 0$, where abscissas x_j and weights w_j are given by

$$x_j = g(hj) = \tanh\left(\frac{\pi}{2}\sinh(hj)\right)$$
$$w_j = g'(hj) = \frac{\pi}{2}\frac{\cosh(hj)}{\cosh(\pi/2\sinh(hj))^2}$$

Numerical implementation aspects are detailed in [7]. Usually, standard methods struggle to return accurate results in the presence of severe oscillations of the integrand. For this case, one can consider the specialized methods using the double-exponential integration method developed in [127, 126] or the Gaussian quadrature [82, 40], see Section 4.2 for a complete overview of numerical methods for highly oscillatory integrals.

Recent developments aim to overcome the manual choice of the most suitable quadrature method for each integral using a black-box approach [91]. This method combines adaptive bisection with degree-adaptive Gauss-Legendre quadrature for the evaluation of definite integrals. This method is adequate to evaluate integrals at high-precision, say up to 1000 digits, considering rigorous error bounds rather than merely heuristic error estimates like the previous methods.

1.2.2 Continued fractions

Let us define a continued fraction $C = C_n$ when $n \to \infty$ by [43, §1.12]

$$C_n = b_0 + \frac{a_1}{b_1+}\frac{a_2}{b_2+}\cdots\frac{a_n}{b_n} = \frac{A_n}{B_n}, \quad a_n \neq 0,$$

where C_n is called the n-th approximant to C, and A_n and B_n are called the numerator and denominator respectively. A continued fraction converges if the approximants C_n tend to the finite limit as $n \to \infty$. The numerator and denominator can be computed iteratively by means of the recurrence relations for $k \geq 1$

$$A_k = b_k A_{k-1} + a_k A_{k-2}$$
$$B_k = b_k B_{k-1} + a_k B_{k-2},$$

with $A_{-1} = 1$, $A_0 = b_0$, $B_{-1} = 0$ and $B_0 = 1$. A modern and more convenient representation is

$$f(z) = b_0(z) + \operatorname*{K}_{m=1}^{\infty} \frac{a_m(z)}{b_m(z)}.$$

There are several continued fraction representations: Stieltjes fraction (S-fraction), Jacobi fraction (J-fraction), C-fraction, M-fraction and T-fraction. Additionally, approximants modifications to improve convergence of these continued fractions are discussed in [38].

Continued fractions are generally computed by means of one of the following methods: forward recursion using the above three-term recurrence relations, backwards recurrence algorithm, Steed's algorithm and the modified Lentz's algorithm. The latter is the preferred choice due to avoidance of successive rescaling and that prior information about the value of n is not required [61].

Continued fractions are a common technique to evaluate special functions and their quotients. Continued fractions are applied for the computation of confluent hypergeometric functions in [113]. A comparison among several continued fractions types for evaluating the generalized exponential integral is presented in Chapter 3. Lastly, we refer the reader to [38] for an exhaustive treatment of theoretical and numerical aspects for continued fractions.

1.2.3 Sequence acceleration techniques

For some particular methods the resulting series expansion converges very slowly, requiring a large number of terms to yield just a few correct digits. In addition, the cost becomes prohibited for a series expansion with coefficients involving functions hard to compute. In these cases, series acceleration techniques stand out as powerful tools to improve convergence at the expense of performing several operations with partial sums. Therefore, the goal of convergence acceleration algorithms is to accurately estimate the sum of slowly convergent infinite series.

There exist an extensive literature on sequence (series) acceleration, also referred as sequence transformations. For examples and covering of many transformations such as the classic techniques (the Euler's transformation, Shank's transformation and Levin-type transformations among others) we refer the reader to [21, 142]. Nowadays, there is not a universal technique that can efficiently accelerate the convergence of all infinite series, in contrast, there are tailored algorithms for different types of sequences. For example, a novel technique developed to accelerate the convergence of generalized hypergeometric functions is studied in [168].

Other methods often applied to improve convergence properties of a power series are the well-known Euler-Maclaurin formula, requiring derivatives and Bernoulli numbers and polynomials, and the Abel-Plana formula, with no derivatives required, see [43, §2.10].

In the last decades, significant research has been endeavoured to develop efficient algorithms to accelerate the convergence of alternating and divergent series. A simple yet very effective linear method is presented in [35]. This method has been widely applied in number theory for rapid evaluation of alternating sums to very high precision. Nonlinear acceleration methods have been extensively treated by Weniger in [166, 86] among other works, where the transformation of monotone slowly convergent series into an alternating series is investigated. Another interesting method for the summation of divergent asymptotic series is the use of factorial series [167]. Finally, note that the technique in [35] is applied to accelerate convergence are remove cancellations when evaluating the Lerch transcendent for small negative argument, see Chapter 5.

1.2.4 Other methods

The described methods are generally applicable for the computations of special functions in arbitrary-precision. However, there are also some other methods tailored for computation at fixed precision that can be encountered in many numerical libraries.

Chebyshev expansions, rational Chebyshev approximations and Padé approximations [61, §9.2] were methods popularly used during the last part of the past century to develop numerical routines for the evaluation in double-precision floating-point arithmetic. These methods are applicable to approximate a function $f(x)$ by a polynomial $p(x)$ within real intervals $[a, b]$ with uniform accuracy. The minimax approximation is the best option to approximate a continuous function $f(x)$ on an interval $[a, b]$. For details and examples for univariate special functions we refer to the work developed by W. J. Cody [30, 28] and the numerical library developed by the same author, SPECFUN [29], where this type of methods are often used to accurately compute functions for arguments in bounded intervals.

To conclude, we remark that occasionally, methods to approximate solutions of ordinary differential equations such as Runge-Kutta methods may be a viable alternative when no available method returns a satisfactory value for some function's domain, see discussion in [133, §Appendix C].

Chapter 2

Software Development for the Numerical Evaluation of Special Functions

In the second Chapter, we focus on the main aspects considered when developing numerical software for the evaluation of special functions. The first two Sections are devoted to arithmetic in fixed and arbitrary-precision, where we present algorithms implemented in software arithmetic (not to be confused with hardware arithmetic). The remaining Sections discuss fundamental aspects in numerical software development such as the use of standard libraries for compatibility, compilers and best practices for proper testing, profiling, and deployment. The Chapter concludes with an introduction to GNSTLIB, a numerical library written in C++ for the evaluation of special functions developed following the principles previously presented.

2.1 Arbitrary-precision arithmetic

In what follows we briefly summarize some fundamental arbitrary-precision algorithms for performing arithmetic operations, which are implemented in packages for the computation of elementary and special functions. By high-precision, more generally arbitrary-precision, we refer to higher precision than the achievable directly using IEEE 754 standard floating-point hardware, introduced after this subsection. A modern book on the subject is *Modern Computer Arithmetic* [18], complementing the classical reference [99], which we refer to more complete material.

2.1.1 Algorithms

Let us define n as the number of binary digits required to obtain a result with accuracy 2^{-n}. For addition and subtraction of two n-bit integer numbers the cost is $\mathcal{O}(n)$, but in computer algebra the quintessential primitive operation is multiplication. Consider $M(n)$ a bound for the time, measured in word or bit-operations, required to multiply two n-bit integer numbers or polynomials of degree $n-1$. The naive multiplication algorithm (schoolbook multiplication) gives $M(n) = \mathcal{O}(n^2)$ and the subquadratic "divide and conquer" multiplication algorithm by Karatsuba [96] gives $M(n) = O(n^{\log_2 3}) \approx O(n^{1.585})$. For large n, the Schönhage-Strassen algorithm [139], based on the fast Fourier transform gives $M(n) = \mathcal{O}(n \log n \log\log n)$

asymptotic complexity. In general, the naive multiplication algorithm is used for small n, and subquadratic algorithms are used elsewhere except for very large n, where asymptotic complexity algorithms are preferred. A good survey of algorithms for fast multiplication with applications can be found in [9]. The division operation is avoided when possible, and replaced by multiplication since it is a more expensive operation [18, §1.4]. Note that the complexity of the arithmetic operations of p-bit floating-point numbers is the same as the described for integer numbers, up to a constant factor [20, §3].

Several techniques are used for computing elementary transcendental functions to arbitrary-precision. The basic and widely used techniques are: argument reduction, the arithmetic-geometric mean (AGM), Newton's method and fast methods for evaluation of power series. Argument reduction (expansion) is a technique used to decrease (augment) the magnitude of the argument to a domain where the function can be evaluated effectively. This result in the evaluation of fewer terms to achieve the desired accuracy thus improving the convergence rate; for example $\exp(x) = (\exp(x/2^k))^{2^k}$. Other tricks such as combination of power series after argument reduction or the use of symmetries are usually exploited in software implementations. When computations are performed at very large precision p, the AGM algorithm is chosen due to its asymptotic complexity; requires $\mathcal{O}(M(p)\log p)$ bit operations to give p-bit accuracy. On the other hand, Newton's method can be used in the same cases where AGM is applicable. Newton's method has complexity $\mathcal{O}(M(p)\log p)$, but increased with a constant factor.

Now, we mention several techniques to evaluate polynomials of degree $P(x) = \sum_{k=0}^{n-1} a_k x^k$ at p-bit precision, resulting after determination of the optimal truncation point when evaluating a power series. Hereinafter, we assume that coefficients a_{k+j}/a_k are rational functions, which eases a sequential evaluation by means of a linear recurrence (this is the common case of hypergeometric functions, see Chapter 4). Fast algorithms for evaluation of linear recurrence include the binary splitting, rectangular splitting and fast multipoint evaluation. A naive forward recursion algorithm requires $\mathcal{O}(n^2)$ operations and is only suitable at low precision. The binary splitting algorithm has time complexity $\mathcal{O}(M(n)\log n)$ as noted in [19]. In this algorithm the finite sum is computed by "divide and conquer" [18, §4.9]. The fast multipoint evaluation method allows evaluation of the n-th term of a linear recurrence in $O(M(\sqrt{n}))$ operations [17]. The rectangular splitting algorithm [18, §4.4.3] is based on the baby-step giant-step technique in [130] and extended in [145]. The idea of rectangular series splitting is to evaluate a power series with $\mathcal{O}(\sqrt{n})$ non-scalar multiplications and $\mathcal{O}(n)$ scalar operations using $\mathcal{O}(\sqrt{n})$ storage. Finally, a description of the domain of applicability of each algorithm in the context of the evaluation of hypergeometric functions is given in [88, 90].

2.1.2 Libraries

Here we present a list of popular open-source software packages for arbitrary-precision arithmetic.

1. GMP [69]: The GNU MP is a library written in C and the main reference for arbitrary-precision arithmetic. Most of the new developments are built on

this library. GMP is released under the GNU Lesser General Public License (LGPL).

2. MPFR [54]: The MPFR is a library written in C for arbitrary-precision floating-point computations with correct rounding and exceptions, extending the ideas of the IEEE 754 standard. This library is based on the GMP and therefore also distributed under the GNU LGPL.

3. MPC [47]: The GNU MPC library, is a C library for arbitrary-precision complex arithmetic with correct rounding, based on the MPFR and GMP libraries. This library is also distributed under the GNU LGPL.

4. mpmath [92]: This is a Python library for arbitrary-precision real and complex arithmetic. Mpmath can use GMP as a backend, replacing native Python integer operations thus improving performance at high precision. Mpmath is distributed under the Berkeley Software Distribution (BSD) license.

5. Arb [87]: Arb is a C library for rigorous real and complex arithmetic with arbitrary precision. This library differs from the previous ones on the approach to tracking numerical errors, which relies on a form of interval arithmetic based on a midpoint-radius representation called ball arithmetic. Arb is distributed under the GNU LGPL.

2.2 Floating-point arithmetic

In this subsection, we present a brief survey on the extensive topic of floating-point arithmetic. An excellent reference for this subject is the *Handbook of Floating-point Arithmetic* [112] to which we refer the reader for further reading.

After a mandatory introduction to floating-point arithmetic, this subsection is focused on the less known floating-point extensions. Floating-point extensions are used when higher precision than the standard double-precision or binary64 is required, and a software implementing arbitrary-precision arithmetic is not available. This need arises in numerical problems in mathematics, physics and engineering fields when higher precision is necessary to compensate cancellation or ill-conditioned sums (for example, the procedure known as iterative refinement when numerically solving a system of linear equations). Some of the techniques presented are implemented and discussed in detail in Section 3.4. for the numerical evaluation of the generalized exponential integral.

2.2.1 Definitions and basic notation

Let us start by defining a floating-point number in binary representation. A normal binary floating-point number with precision p-bit is a number of the form

$$x = M \cdot 2^{e-p+1}, \tag{2.1}$$

where M is a signed integer denoted as the significant or mantissa of x, which absolute value is bounded $2^{p-1} \leq |M| \leq 2^p - 1$. The integer e is called the exponent of x. The 2 in above formula represents the base or radix for exponentiation, in this

case binary. The commonly used bases to represent floating-point numbers are decimal (the scientific notation with base ten) and binary with base two.

Let us define e_{min} and e_{max} as the minimum and maximum allowed exponent, respectively. Three extreme cases are considered:

1. minimum subnormal number: $2^{e_{min}-p+1}$.
2. minimum normal number: $2^{e_{min}}$.
3. maximum normal number: $2^{e_{max}}(2-2^{1-p})$.

In order to store floating-point number correctly, its significant must be normalized, i.e,. must be represented in the normal form (2.1), assuring the first bit of the significant is set to 1 and adjusting the exponent accordingly. This process is known as normalization. For example, 1001.101 is normalized as $1.001101 \cdot 2^3$. A floating-point number x satisfying that $x \leq 2^{e_{min}}$ is called subnormal or denormalized number. This number is characterized by having the leading bit of the significant set to 0, which result in an exponent that is below e_{min}. In the IEEE 754 standard, described below, an underflow exception is raised when a subnormal number occurs and the operation is inexact.

The IEEE 754 standard, specifies interchange and arithmetic formats and methods for binary and decimal floating-point arithmetic, along with exception conditions and their default handling, in computer programming environments. The active IEEE 754-2008 standard[146] [1] supports several binary formats. The characteristics of the three most common floating-point binary storage formats are listed in Table 2.1. For example, the bits for binary64 or double-precision are arranged as follows, 1 bit for the sign, 11 bits for the exponent, and 52 bits for the significant.

format	word size	sign	exponent	significant	p	e_{min}	e_{max}
Binary32 (Single)	32	1	8	23	24	-126	127
Binary64 (Double)	64	1	11	52	53	-1022	1023
Binary128 (Quad)	128	1	15	112	113	-16382	16383

TABLE 2.1: Main parameters of the basic formats specified by the IEEE 754-2008 standard.

The operations specified by the IEEE 754-2008 are: addition/subtraction, multiplication, division/remainder and squared root. All values from these operations are returned with correct rounding using the current rounding mode.

The IEEE 754 standard considers five exceptions that can be flagged during a calculation. The first three exceptions (invalid operation, division by zero and overflow) are the most common exceptions and can be easily identified. The last two (underflow and inexact) and usually ignored, and careful inspection is required to detect the source of errors.

1. Invalid operation: This exception is raised if the result of a calculation is mathematically undefined. By default returns quite NaN. A few examples: $\sqrt{x}$ where $x < 0$ or $(\pm 0)/(\pm 0)$.

[1] The active IEEE 754-2008 standard was adopted in June 2008. This version superseded two previous standards from 1985 and 1987. Two of the major additions with respect to previous standards are the fused multiply-add (FMA) operator and including quadruple precision.

2. Division by zero: This occurs when computing $x/0$ if x is a nonzero finite number, the division by zero exception is flagged. The result is a correctly signed infinity.

3. Overflow: This exception is raised when the absolute value of the computed result is strictly larger than the maximum normal number, i.e., $|x| > 2^{e_{max}}(2 - 2^{1-p})$, or equivalently, when the exponent of the computed result is strictly larger that e_{max}.

4. Underflow: This exception is raised when the result of computation is a nonzero of absolute value less than $|x| < 2^{e_{min}}$. In this case, the returned value is in the subnormal range, and a significant loss of accuracy may occur. The returned value is subnormal or zero.

5. Inexact: This exception is raised when the rounded result of a valid operation is not exact. The correctly rounded, overflowed or underflowed result is returned.

Floating-point operations are performed internally with extra precision. The calculated values, not exactly representable in the same floating-point system used for the input data, are returned rounded to ensure precision. The IEEE 754 defines four possible rounding modes:

1. Round toward $+\infty$: result is rounded to the smallest representable value which is greater that the result.

2. Round toward $-\infty$: result is rounded to the largest representable value which is less that the result.

3. Round toward zero: truncation, it is similar to the common behaviour of float-to-integer conversions. If the result is negative is rounded up, and contrarily for positive result.

4. Round to nearest: the result is rounded to the nearest representable value. Two modes: the default rounding mode for binary floating-point arithmetic is round to nearest even, which in case of ties, select result with even significant. Alternatively, round to nearest away, where ties round away from zero (used only for decimal floating-point arithmetic).

To conclude, we introduce two instruction sets currently supported by many processors. We refer to [112, §3.5-3.6] for an extended list of instructions supported by modern CPU and GPU.

1. Fused multiply-add (FMA):

   The FMA instruction was introduced to facilitate correctly rounded software division and to make some calculations (especially dot products and polynomial evaluations) faster and more accurate. The new IEEE 754-2008 standard for floating-point arithmetic specifies the FMA instruction. The FMA instruction performs the operation $a \times b + c$ with only one rounding error with respect to the exact result. This is actually a family of instructions that includes useful variations such as fused multiply-subtract. The processors implementing these instructions sets provide hardware FMA operators, with latencies comparable to classical $+$ and $\times$ operators.

2. SIMD instructions:

   Most recent instructions sets also offer single instruction, multiple data (SIMD) instructions. One such instruction applies the same operation to all elements of a vector of data kept in a wide register.

2.2.2 Floating-point expansions and error-free transformation

The theory on floating-point expansions was pioneered by Dekker [42] and extended by Priest [135] and Schewchuck [140]. A floating-point expansion can be defined as an exact finite sum operation of floating-point numbers. More precisely, a floating-point expansion is a multiple-term representation in which a number is expressed as the unevaluated sum of n standard floating-point numbers. This approach offers high performance, exploiting modern hardware capabilities, due to using directly available and optimized hardware implemented floating-point operations. However, among the drawbacks of this approach, we have that those operations performed using floating-point expansions are not compliant with the IEEE 754-2008 standard, the returned results are not correctly rounded and numbers have a limited exponent range.

Most algorithms performing arithmetic operations on floating-point expansions are based on the so-called error-free transformations. An error-free transformation is an algorithm which transforms any arithmetic operation of two floating-point numbers a and b into a sum of two floating-point numbers s and t, being a floating-point approximation and an exact error term, respectively. Therefore, these algorithms rely on native precision operations but keep track of all accumulated rounding errors to avoid a loss of information.

A reference library using this approach is Bailey's library QD [80]. This library provides support to double-double and quad-double numbers, i.e., numbers are represented as the unevaluated sum of two and four double-precision floating-point numbers, respectively.

Basic algorithms

Here we only briefly present a survey of the algorithms developed to cover practically all common mathematical operations. The presented algorithms assume that operations are performed in rounding to nearest mode. We start with addition/-subtraction operation; the algorithms *TwoSum* [99] and *FastTwoSum* [42] are the basic bricks used for more involved operations. These two algorithms take 6 and 3 flops[2], respectively, but *FastTwoSum* requires a branch ($|a| \geq |b|$ must be satisfied), making *TwoSum* usually preferable on modern processors. For multiplying two floating-point numbers, the algorithm *TwoProduct* in [42] takes a total of 17 flops and requires an error-free splitting of each floating-point number into two parts, $x = x_h + x_l$, guaranteeing that no information is lost in the transformation. This is known as Veltkamp's algorithm and requires 4 flops. Another accurate algorithm for the product and exponentiation of floating-point numbers is discussed in [65]. An alternative algorithm taking only 2 flops is *2MultFMA*, which uses a FMA

[2]Flops: native floating-point operations.

instruction [93]. For division, a fast classical algorithm is given in [135], which division is carried out using the long division algorithm. A recent algorithm using truncated Newton-Raphson iterations for reciprocal of a floating-point expansion is studied in [94, 93]. Finally, an algorithm for square root based on adapted Newton-Raphson iteration with a detailed error analysis is detailed in [93].

Several summation techniques [81] have been developed to perform accurate addition of n floating-point numbers. These algorithms replace the traditional method when the problem is ill-conditioned. A sum is ill-conditioned if its condition number $C = |\sum_{i=0}^{n} x_i| / \sum_{i=0}^{n} |x_i|$ is large. Some techniques typically used are: reordering in increasing/decreasing magnitude before addition, compensated summation based on previous algorithms *FastTwoSum* and *TwoSum*, and distillation algorithms, which separate accumulators with exact additions until these are finally added. Among the variations of compensated algorithm, we remark *Vecsum* [121]. Similar algorithms have been devised to perform dot product accurately [121]. For polynomial evaluation, compensated Horner's methods have been extensively studied in [66, 67, 68], including variants using FMA and with complex floating-point arithmetic.

Finally, we note that we have implemented most of these algorithms in the library Fhyperg, introduced in the following subsection.

2.2.3 DD vs MPFR for the evaluation of Riemann zeta function

In this subsection we compare the use of double-double arithmetic against arbitrary-precision libraries in "medium" precision (106-bit). These floating-point expansions are suitable in many applications in engineering and physics when only a small multiple of the double precision is needed, without recurring to arbitrary precision, obtaining significant performance gain. For this benchmark we implement the Riemann zeta function for integer argument using Fhyperg, a library developed in Fortran 90 for the computation of special functions using double-double numbers for real and complex values. Fhyperg is a work in progress (with roughly 7000 lines of code) and can be downloaded from `https://sites.google.com/site/guillermonavaspalencia/software`.

For the purpose of this study, we briefly introduce the Riemann zeta function and the algorithms implemented. More details about the Riemann zeta and other related functions can be found in Chapter 5. The Riemann zeta function is the simplest of all L-functions, which can be defined as a Dirichlet series

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s} \quad \Re(s) > 1. \tag{2.2}$$

and Dirichlet L-series can be expressed in terms of its Euler product, such that

$$\zeta(s) = \prod_{p} \frac{1}{(1 - p^{-s})} \tag{2.3}$$

where p indicates an infinite product over primes. The function has a continuation to the whole complex plane with a simple pole at $z = 1$ with residue 1. For $s = 3$ the

Riemann zeta function is defined as Apéry's constant. This constant can be computed very efficiently using the accelerated formula described in [2] and applied in [73], defined by

$$\zeta(3) = \sum_{k=0}^{\infty} (-1)^k \frac{205k^2 + 250k + 77}{64} \frac{k!^{10}}{(2k+1)!^5}. \tag{2.4}$$

Other identities for the Riemann zeta function at odd integers were discovered by Plouffe[3], for example $\zeta(7)$ is given by

$$\zeta(7) = \frac{19\pi^7}{56700} - 2\sum_{k=1}^{\infty} \frac{1}{k^7(e^{2\pi k} - 1)}. \tag{2.5}$$

For large positive integers we use the defining series of the Riemann zeta function. The required number of terms to obtain p bits of precision is estimated using $n \approx (2^p)^{1/s}$, in practice we found that a heuristic $n = \lceil (2^p)^{1/s} \rceil + 3$ suffices for $p = 106$. As shown in Figure 2.1, this methods exhibits fast convergence as s increases, therefore the L-series is applied for $s \geq 18$.

Borwein's algorithms

P. B. Borwein in [15] describes three algorithms particularly simple compared with other methods, but remarkably efficient. We focus on algorithm 2 and 3. The series expansion in Algorithm 2 is given by

$$\zeta(s) = \frac{-1}{d_n(1 - 2^{1-s})} \sum_{k=0}^{n-1} \frac{(-1)^k (d_k - d_n)}{(k+1)^s} + \varepsilon_n(s), \tag{2.6}$$

where

$$d_k = n \sum_{i=0}^{k} \frac{(n+i-1)!4^i}{(n-i)!(2i)!} \tag{2.7}$$

taking the coefficient $d_0 = 1$ and subsequent coefficients can be computed by using the hypergeometric recurrence $\frac{2(n+i-1)(n-i+1)}{i(2i-1)}$. It can be checked that the coefficients d_k are integers. The error term $\varepsilon_n(s)$ satisfies

$$|\varepsilon_n(s)| \leq \frac{3}{(3+\sqrt{8})^n} \frac{1}{1 - 2^{1-s}}. \tag{2.8}$$

The analysis in [54] specifies that in order to obtain an accuracy of around p bits for $s \geq 2$, a number of terms $n \geq \frac{\log(2)}{\log(3+\sqrt{8})} p$ is sufficient. We tested that $n = 44$ is sufficient for $s < 18$ with $p = 106$.

The series expansion in Algorithm 3 is defined by

$$\zeta(s) = \frac{-1}{2^n(1 - 2^{1-s})} \sum_{j=0}^{2n-1} \frac{e_j}{(j+1)^s} + \varepsilon_n(s), \tag{2.9}$$

[3] http://www.lacim.uqam.ca/~plouffe/identities.html

$$e_j = (-1)^j \left(\sum_{k=0}^{j-n} \frac{n!}{k!(n-k)!} - 2^n \right), \tag{2.10}$$

where the empty sum is zero. The error term $\varepsilon_n(s)$ satisfies

$$|\varepsilon_n(s)| \leq \frac{1}{8^n} \frac{1}{1-2^{1-s}}. \tag{2.11}$$

This algorithm is simpler to implement but it is not quite as fast as Algorithm 2, a comparison between both algorithms is shown in Figure 2.1. Other efficient algorithms exist, for example the Euler-Maclaurin summation formula, which converges much faster than the algorithms discussed. However, Euler-Maclaurin summation requires the computation of Bernoulli numbers, which turns out to be undesired due to the required storage and computational time. Given that precision is fixed, we could pre-compute all required Bernoulli numbers, but the purpose of this test is to provide a fair comparison of the underlying arithmetic.

In summary, our implementation treats special cases for $s \in \{0, 1, 3\}$ and combines Borwein's algorithm 2 and L-series for other values of s.

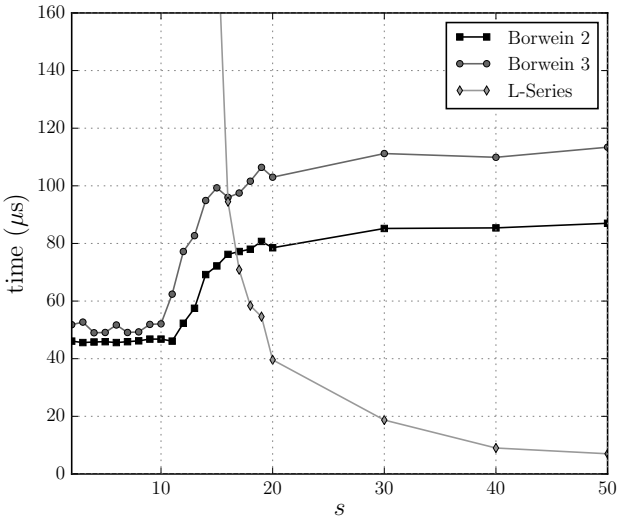

FIGURE 2.1: Timing of the three methods in microseconds.

Implementation and benchmarks

As stated in [80], a precision of 106 bits (around 32 decimal digits) is achievable by using double-double numbers. More precisely, the equivalent machine epsilon[4] is $\epsilon_{dd} = 2^{-104} \approx 4.93038e{-}32$.

As mentioned, we implemented the above algorithms in the Fhyperg library, written in Fortran 90 using some Fortran 2003 features. Fhyperg is compiled with gfortran 5.4.0. with optimization flag -O3. An important optimization considered,

[4]This is the epsilon considered in QD package file `dd_const.cpp`: https://bitbucket.org/njet/qd-library/src/352c372f9c94d957d312e131c804fcac1088561d/src/dd_const.cpp

is to use integer powers and switch to double-double only when necessary in order to speed up the code. The following code illustrates how this is implemented:

LISTING 2.1: Main loop Borwein's algorithm 2

```
! evaluate series
do j = 0, n
  ! (-1)^j (d[j] - d[n])
  if (mod(j, 2) == 0) then
    call dd_negative_copy(dn, v)
    call dd_add_dd_dd_ip(ds(j+1), v)
  else
    call dd_negative_copy(ds(j+1), v)
    call dd_add_dd_dd_ip(dn, v)
  end if

  nr = real(j + 1, kind=wp) ** s
  if (nr < maxint) then
    m = (j + 1) ** s
    call dd_div_dd_d_ip(real(m, kind=wp), v)
  else
    call dd_pow_dd_i(dd_real(real(j + 1, kind=wp)), -s, q, info)
    call dd_mul_dd_dd_ip(q, v)
  end if

  call dd_add_dd_dd_ip(v, t)
end do
```

The benchmarks are performed on an Intel i7-6700HQ at 2.60GHz running Linux 64-bit. Our implementation using double-double extended precision is compared with MPFR 3.1.4 in Sage 7.3 with 106 bits of precision. The measures for MPFR were obtained by using the function `%timeit`, which basically evaluates the function in a loop N times and returns the best average time out of three runs. Fhyperg is tested next using the same number of evaluations in the loop and returning again the best average out of three.

Figure 2.2 shows a notable general speedup for Fhyperg for small s, which is diminished as s increases. We also observe a peak in the transition regions between the Borwein's algorithm and the L-series, due to the increasing numbers of the terms in Borwein's algorithm. For ≥ 18, we observe a steady decay on timing. On the other hand, Figure 2.3 shows the achieved accuracy, which remains below the machine epsilon threshold.

Finally, although floating-point extensions can effectively exploit modern hardware obtaining high-performance, recent improvements on this range of precision for arbitrary-precision arithmetic have been achieved [103], almost closing the performance gap.

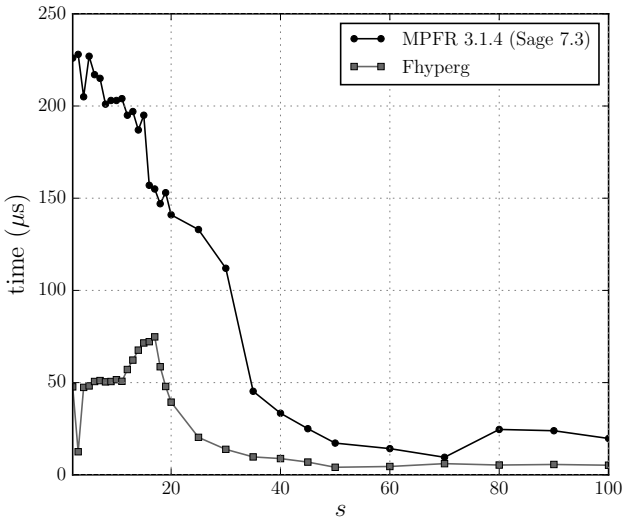

FIGURE 2.2: Timing in microseconds for 106-bit precision vs MPFR 3.1.4. MPFR caches intermediate results for $s \in \{50, 60, 70\}$.

2.3 Development of numerical libraries in floating-point precision

This section deals with the main aspects to take into account when developing numerical software in general, and for the evaluation of special functions, in particular. This includes the use of standard libraries, compilers, and various development techniques and testing methodologies important to guarantee robust and quality software.

2.3.1 Numerical libraries and compilers

Nowadays, there exists a notable number of open source and commercial numerical libraries including implementations for the numerical evaluation of special functions y double-precision floating-point arithmetic, extended precision, and arbitrary-precision arithmetic.

Historically, these numerical libraries were developed using low-level programming languages such as Fortran or C/C++. More recently, these numerical libraries have been ported to other programming languages by means of wrappers, to provide stable, well-tested implementations to newer influential languages without much work. However, there are a few exceptions like the Julia package SpecialFunctions.jl, which combines implementations from Fortran and C/C++ libraries and others entirely developed in Julia, a modern high-performance programming language very well-suited for scientific computing. Another example is the Python module `scipy`[95], which also combines Fortran and C implementations with new developments in Python and Cython (C extensions for Python) to obtain C performance. The classical open source numerical libraries including special functions are the Fortran (AMOS[3], SLATEC[163], SPECFUN[29]), and C

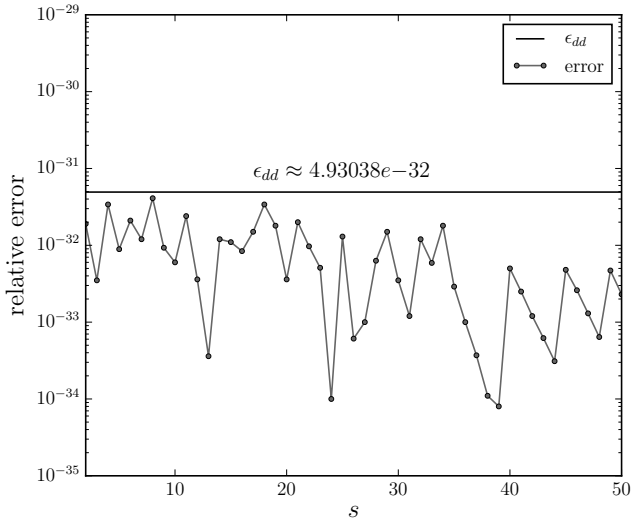

FIGURE 2.3: Relative errors checked with Mathematica. Maximum relative error $\approx 0.8\epsilon_{dd}$.

(Cephes[111] and GSL[56]). On the other side, two classical commercial libraries including special functions from their very first release are IMSL[147] and NAG[114]. Recent alternatives are Boost[11] and AMath[46]. In addition, some scientific journal such as TOMS stores the collected algorithm of the ACM, published by the journal ACM Transactions on Mathematical Software. Other public repositories are Netlib or John Burkardt webpage `https://people.sc.fsu.edu/~jburkardt/`, where one can find a great deal of TOMS codes written or translated to C++.

New numerical libraries for the evaluations of special functions should make use of high-performance libraries with parallelization capabilities on the CPU, and ideally GPU. The Intel Math Kernel Library (MKL) is a good example, including functions to accelerate and optimize routines in linear algebra, statistics and common special functions. This is a high-optimize and easy-to-use library including vectorized mathematical functions. In particular, includes an extensive list of vectorized special functions[5]. Additionally, several programming languages include many common and special functions as a part of their standards; for example, C++17 `<cmath>` incorporates new special functions such as elliptic integrals or exponential integrals. Finally, as mentioned later, most of the previous numerical libraries do not cover correctly all functions' domain, especially for those functions with several real and complex parameters and arguments. Nevertheless, some of the functions correctly implemented can be used as building blocks for new implementation of more involved functions.

For the development of numerical libraries in Fortran or C/C++, we required the use of compilers. The following list includes the most widely used compilers by programming language:

[5] `https://software.intel.com/en-us/mkl-developer-reference-c-vm-mathematical-functions`

- C/C++: GNU (gcc and g++), Clang C++ and Intel C++ (icc). Vendors benchmarks[6].
- Fortran: GNU (gfortran), Intel Fortran (ifort), NAG Fortran compiler, PGI compilers and absoft Fortran compiler. Vendors benchmarks[7].

During the development, it is convenient to use different compilers with high checking capabilities and detailed error reporting, rather than the compilers producing the best performance results. This extensive checking is useful to write cleaner code (some compilers include source file polishers) leading to fewer mistakes. For deployment, the fastest compiler across all supported platforms is used. Note that compilers' performance is strongly influenced by the use of adequate optimization flags; careful and systematic testing of these flags is required, as noted below. Finally, we remark that apart from automatic optimization and vectorization performed by the compiler, it is recommended to write algorithms using techniques allowing optimizations. For example, writing power series or piecewise approximation polynomials using the Horner scheme with a fixed number of terms permits loop-unrolling optimizations available in all modern compilers.

2.3.2 Design of software for computing special functions

The implementation of special functions in double-precision floating-point arithmetic presents some major difficulties when most of a the function's domain needs to be correctly covered. For arbitrary-precision arithmetic, implementation of special functions usually includes only a few algorithms, and working precision is adequately incremented to compensate numerical instabilities. However, this approach is not suitable for fixed-point arithmetic.

Therefore, the goal is to combine various methods in order to efficiently and accurately compute a wide region of the function's domain. These methods range from series expansions and continued fractions to numerical quadrature. In cases when there are various valid methods for the same function's domain, the obvious choice is the fastest and more accurate method. For these cases, it is also a good practice to enable various execution modes for the same function, allowing the choice of the best suitable method for each application. This is a common practice to control central compiler behaviours, for instance, optimization flag `-ffast-math` on GCC compiler might yield faster code, but do not guarantee the strict IEEE compliance.

For some special functions, the achievable accuracy of each method can be effectively estimated by developing rigorous error bounds. Generally, these developments require hard theoretical error analysis, especially when dealing with complex uniform asymptotic expansion with various parameters involved. Therefore, unfortunately, only a few rigorous bounds are known for elementary functions on restricted domains. Availability of such rigorous limits eases the correct determination of the region of applicability for each algorithm and permits to identify regions where none of the considered algorithms behaves adequately. Besides, these bounds might be used to determine the required number of iterations to achieve

[6] https://software.intel.com/en-us/c-compilers/features/benchmarks
[7] https://www.fortran.uk/fortran-compiler-comparisons/

certain accuracy; for asymptotic expansions this helps to detect if it is applicable with given parameters and argument values. However, when rigorous bounds are not available, it is unavoidable to resort to using heuristic error estimates, giving up rigour and being exposed to incorrect results. Therefore, if used, these must be meticulously tested to have strong empirical results about their usability.

Implementations of special functions with several parameters and argument tend to require several methods. If various methods cover the same regions of the function's domain, the process of manually choosing the most suitable method for each region becomes a time-consuming task. One can treat this problem using classification algorithms such as decision trees, widely employ in predictive modelling. Given a large set of samples with features (parameters and argument) and a multi-class target (the most accurate and/or fast method for each sample), we aim to devise the best split points for each feature in order to predict the class (the most suitable algorithm) and the probability of each class. This method does not substitute a numerical analysis of error bounds but generates a first set of rules useful to develop heuristics. As an advantage, decision trees are data structures easy to visualize and interpret, which rules can be straightforwardly translated into code. The main disadvantages include over-complex trees producing over-fitting and the need to have implementations for several methods, some of which may be discarded in the final algorithm. Figure 2.4 show a decision tree example generated with several methods to compute the generalized exponential integral. On the other hand, Figure 2.5 presents an alternative plot with regions of applicability of each method for the same functions. Note that these type of plots are difficult to interpret as the number of parameters increases, even using pairwise plots and for these cases, therefore, binary trees improve visualization.

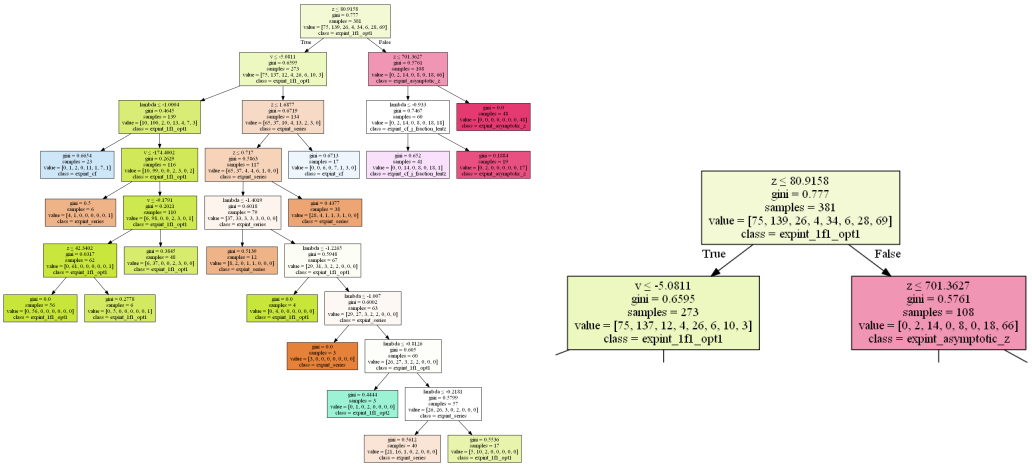

FIGURE 2.4: Decision tree generated with several methods to compute the generalized exponential integral for real order and argument. Detail with the first two split points, where class indicates the method exhibiting superior accuracy (right).

Besides considering several methods, the use of connection formulas with simpler formulas is recommended to facilitate code maintenance and reduce development times. Furthermore, the implementation of many special cases, expressible in

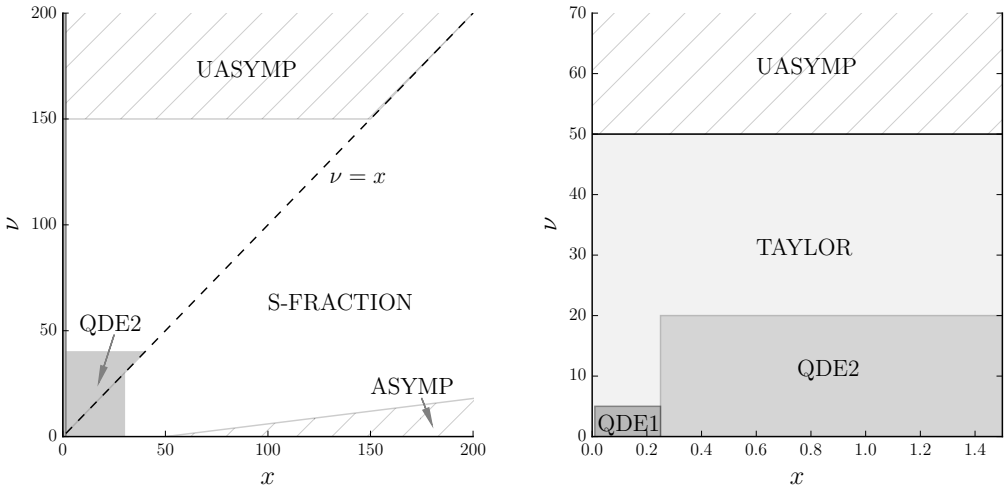

FIGURE 2.5: Regions of applicability of each method for the numerical evaluation of the generalized exponential integral for real order and argument. Detail for small argument x (right).

terms of simpler formulas, also reduces sources of numerical instability improving consistency across the same family of functions.

In some applications we need scaled versions of special functions to widen the region of calculation and avoid overflow problems, for instance, when evaluating ratios of functions with an exponential terms. Additionally, some specific regions such as the vicinity of function zeros suffer from accuracy loss and require algorithms implemented in extended precision to mitigate catastrophic cancellation and other numerical issues, see Section 3.2.2. Many open-source and commercial libraries exhibit similar accuracy degradation when evaluating functions in close vicinity of zeros. However, not many include efficient algorithms for these regions. Any new numerical library should be designed taking into account this severe problem; for instance, see efficient algorithm for Bessel functions in [74].

In addition, some challenging regions of a function's domain might present numerical difficulties for all algorithms. These cases can be effectively treated by the use of efficient analytic continuation along a path, from a proximal point correctly evaluated to the point of interest by connecting local solutions. Given a function defined by a second order ordinary differential equation (ODE) with known derivatives at an arbitrary point z, it is possible to build generalized power series (local Taylor series expansion) which can be used to compute values in a neighbourhood of the point z. Although this computation method is primarily used to cover difficult corner cases, it can also find applications in sensibility analysis, when we need to evaluate a function to many similar points. Therefore, we can develop specific algorithms which can be effectively vectorized, as shown in Section 2.4.2. The main inconvenience of this approach is that computation of generalized power series for some functions can be trickier than the function in question.

Another important aspect missing in many numerical libraries implementing

special functions is the availability of vectorized versions exploiting multi-core architectures and SIMD (single instruction multiple data) instructions of modern processors. Vectorization refers to the evaluation of a function on an array of values at once avoiding looping over each array element. These functions are generally implemented in low-level programming languages (C/C++ or Fortran) to reduce overhead when looping in languages such as Python[8] or R, and can easily interface with OpenMP for developing parallel implementations.

Finally, it is important to consider development tools to locate and fix performance issues. Checking why a software is slow is an extremely time-consuming task, therefore fast code analysis to detect bottlenecks is critical. There are broadly used tools such as Callgrind (front-end for Valgrind[9]), and its accompanying visualization tool KCachegrind, to profiling and generating call-graphs. Furthermore, Intel® Parallel Studio XE is a great tool to analyse performance and memory access, and report possible optimizations for single and multi-threading applications, being useful to assure that the executed code effectively exploits multi-core architectures and detect which parts of the code are blocking vectorization.

2.3.3 Testing methodologies

There are several methods to test the quality of the implementation of special functions. These methods use distinct well-established metrics, such as achievable absolute and relative accuracy and efficiency or computation time among others.

A reliable numerical library, usable in production environments, must show stable performance when used in various infrastructures with different specifications. In general, automatic testing tools are employed to generate various configuration settings combining different processors, compilers and optimization flags. Some companies developing commercial multi-platform numerical libraries, consider robustness when the maximum difference in terms of precision for a given function across different configurations is about $10^4\epsilon$ and $10^2\epsilon$, where ϵ is the machine precision. In situations where higher differences or inconsistent performance is observed across some supported platforms, a careful inspection of the implemented algorithm is required. Eventually, the algorithm should be changed in order to maintain homogeneous accuracy levels across all supported environments, guaranteeing portability. In case, no suitable algorithmic substitution exist, these performance issues must be adequately documented.

The standard and widely used automation testing tool in the software development industry is Jenkins[10]. Jenkins is an open-source tool used to automate processes with continuous integration (CI) to facilitate monitoring tasks during development, ultimately reducing time-to-market delivery. Other popular CI tool is Travis[11].

Several methods exist to assess the achievable accuracy of a function implemented in double-precision floating-point arithmetic. The most widely used method is to compute relative and absolute errors with respect to the same function evaluated at higher precision, say quadruple, octuple-precision or superior. For instance,

[8] Note that vectorization in numpy arrays are fast because ufunc is implemented in C.

[9] Valgrind is used to detect memory management bugs (memory leaks) and profile performance.

[10] https://jenkins.io/

[11] https://travis-ci.com/

various values are computed using a software package of arbitrary-precision arithmetic to ascending levels of precision, assuring sufficient certainty about the precision of the result used as a reference. Note that, this method can only be trusted if reliable computer algebra systems (CAS) are employed to perform validations. Basic testing of special functions include the evaluations of special values, which provides a first estimation of the attainable accuracy. Other methods for verification involve the use of functions relations via evaluation of Wronskians [61], evaluations of functional relationships and other special cases such as the zeros of the function. Naturally, the functional relationships used to assess accuracy must be well-conditioned to be utilized for verification purposes.

The other main metric to evaluate the performance of an implementation is computational time. Obtainment of accurate and reliable timing measurements might be tricky under certain settings, ultimately generating misleading conclusions. For implementations in double-precision, it is common that functions calls take more than the execution time of the function itself, thus difficulting the determination of the actual performance. In order to circumvent the problem, it is recommended to measure the time required to evaluate a function many times, diminishing the impact of function calls. Finally, the average and deviation of CPU times is reported. In other cases, a given function is evaluated during an elapsed-time, returning the number of evaluations. These precautions are needed to perform a fair comparison among different software implemented in various programming languages.

During the development of new algorithms, it is recommended to evaluate their consistency with respect to other methods used to evaluate a function in the same regions. This allows a comparison in terms of accuracy and CPU time per iteration, facilitating the selection of the most efficient algorithm.

An alternative method for univariate and bivariate functions is accuracy evaluation by visual inspection. The idea is simple, evaluate a function randomly within its parameters regions and plot the attained accuracy to detect wherever loss of accuracy is exhibited. Figure 2.6 compares the accuracy of the exponential integral for complex argument between the implementation in the numerical libraries `chypergeo`[12] and `scipy`. We observe that the implementation in `scipy` shows a significant loss of accuracy around $\Re(z) \leq 5$, due to the use of the ascending series. In some particular cases, the affected regions are well determined, permitting the study of specific algorithms to improve the accuracy in that particular region. Figure 2.7 shows a concerning loss of accuracy in the sector $arg(z) \in [1.7, 1.9]$; the first image shows the performance of the initial implementation, whereas the second image shows how a new algorithm for that particular region reduces this loss. Additionally, note that this type of tests requires the use of appropriate legends to correctly identify regions of study.

Nevertheless, performance assessment by visual inspections is difficult to undertake when evaluating multivariate functions, therefore, is only a viable approach for univariate and bivariate functions.

Finally, a robust testing environment should include tools for reporting and monitoring tasks, along with a warning system to ease the detection of those cases

[12] https://sites.google.com/site/guillermonavaspalencia/software/chypergeo

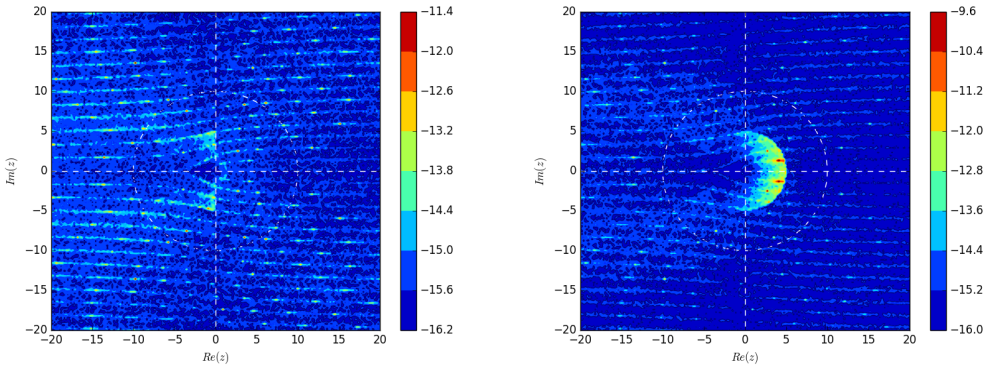

FIGURE 2.6: Accuracy plot for the exponential integral using the numerical library `chypergeo` (left) and `scipy` (right).

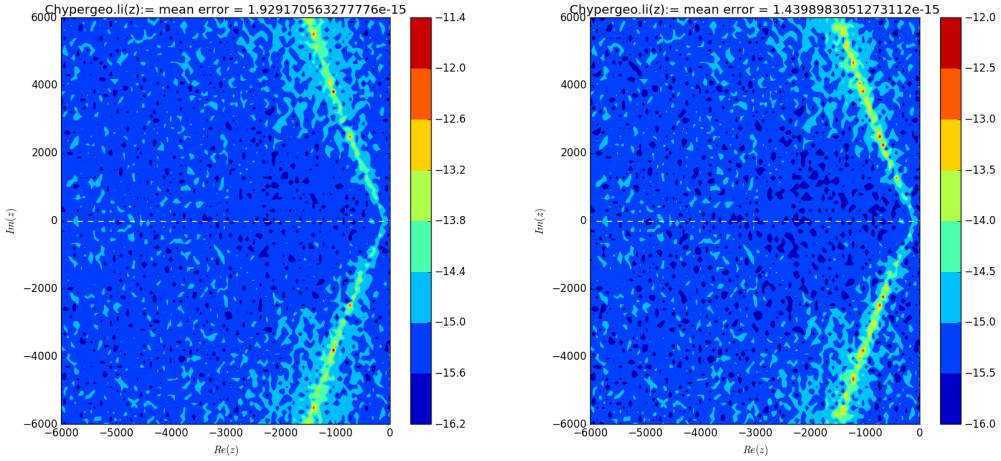

FIGURE 2.7: Attained accuracy using the default algorithm for the logarithmic integral (left) and accuracy plot when using the new algorithm applied in the region exhibiting loss of accuracy (right).

requiring revision. Usually, automatic building of code into the version control system and testing is performed overnight, this is usually called a *nightly build*. Automated builds are generally run on dedicated servers with stable CPU workloads to avoid altering performance metrics.

2.3.4 Benchmarking methodologies

The amount of information gathered during testing of different mathematical software packages is analysed for posterior comparisons. In order to compare and benchmark the mathematical software, the performance profiles were developed in [44]. This method was devised to assess the performance of optimization packages and sparse linear algebra, as a way of providing objective information, but has become a popular tool in other mathematical fields. Many metrics can be compared by means of performance profiles, CPU times, number of iterations and attained accuracy are some commonly used for benchmarking. This methodology has been

subject to criticism by several authors, due to being a biased method when ranking after excluding one particular library from the comparison. Therefore, performance profiles are appropriate when comparing two solvers, but comparison among more than two solvers leads to misleading conclusions regarding the performance of one solver relative to another. Recent work in [76] propose a reliable algorithm to bypass the negative side effect incurred by performance profiles. When comparing mathematical software in terms of CPU time, one approach is to use the shifted geometric mean, as used in well-known and publicly accepted optimization software benchmarks http://plato.asu.edu/bench.html.

For benchmarking special functions, besides CPU times, we consider statistical analysis of the relative and absolute errors with respect to reference values. These statistics permit comparison by parameters regions, building a ranking with the best numerical library for each region of interest. An extensive example is presented in Section 3.5, comparing different numerical libraries implementing the generalized exponential integral.

2.4 GNSTLIB project

2.4.1 Introduction

GNSTLIB is a numerical library currently developed in C++11 for fast and accurate computation of special functions in double-precision floating-point arithmetic. The aim of GNSTLIB is to be used as stand-alone C++ library and provide wrappers for the major programming languages used in scientific computing, such as Fortran, C and Python. This library includes vectorized versions of all special functions implemented. These routines include an option to trigger OpenMP for parallelization, thus taking advantage of multi-core processors. Furthermore, several algorithms for multi-evaluation of univariate special functions via analytic continuation/generalized power series are implemented. These routines are suitable for the evaluation of large samples with values within a small range, as previously mentioned.

This library is developed with the following considerations:

1. Quality: most of the algorithms implemented have been published in peer-review journals (verified software such as ACM TOMS). An extensive framework for testing has been developed to guarantee the highest quality standards.

2. One of the goals is to fill several gaps in the available software for the evaluations of special functions. This is achieved by developing new algorithms and improving those widely used showing a loss of accuracy in some regions.

3. Flexibility: new software must have interfaces for most programming languages.

4. Parallelization and vectorization are a must.

The GNSTLIB project was initiated by the present author in 2016 with the development of Chypergeo[13], which was the basis for the development of GNSTLIB.

[13] https://sites.google.com/site/guillermonavaspalencia/software/chypergeo

During the author's stay at Universidad de Cantabria in May 2017, Amparo Gil, Javier Segura and Nico M. Temme joined the project. GNSTLIB 0.1 was released in November 2017 and was made freely available at https://sites.google.com/site/guillermonavaspalencia/software/gnstlib. Version 0.1 includes

1. Elementary functions (real and complex)
2. Gamma functions (real and complex)
3. Exponential, logarithmic and trigonometric integrals (real and complex)
4. Error functions, Dawson's and Fresnel integrals (real and complex)
5. Incomplete Gamma and generalized exponential integral (real)

In addition, GNSTLIB 0.1 was presented in the International Conference on Computational Science and Engineering, Oslo 2017. Future releases will include Airy, Bessel and confluent hypergeometric functions among others. The addition of statistical distributions and routines for fast computation of Gauss quadratures is also on the roadmap. Furthermore, we plan to include GPU implementation of special functions, extending the CUDA Math Library to provide high-performance when running large simulations.

In the next subsections, we describe the algorithmic details of the multi-evaluation vectorized implementation of the exponential integral and show various benchmarks results comparing against open source and commercial libraries.

2.4.2 Efficient vectorization via generalized power series

In this subsection we present a methodology for multi-evaluation of univariate special functions via generalized power series, using as an illustrative example the exponential integral. The use of generalized power series can benefit the evaluation of an array of arguments with small variations, which might occur when performing sensitivity analysis in some applications in engineering.

Let us define the exponential integral for real argument as a Cauchy principal value

$$\mathrm{Ei}(x) = \mathcal{P}\int_{-\infty}^{x} \frac{e^t}{t}\,dt, \quad x \in \mathbb{R}. \tag{2.12}$$

The exponential integral series representation is given by

$$\mathrm{Ei}(x) = \gamma + \log(x) + \sum_{k=1}^{\infty} \frac{x^k}{kk!} - \begin{cases} i\pi, & x < 0 \\ 0, & \text{otherwise} \end{cases}, \tag{2.13}$$

and the generalized power series at $x = x_0$ is given by

$$\mathrm{Ei}(x) = \mathrm{Ei}(x_0) - \sum_{k=1}^{\infty} \frac{(-1)^k}{k! x_0^k} \Gamma(k, -x_0)(x - x_0)^k, \tag{2.14}$$

where $\Gamma(k, -x_0)$ is the upper incomplete gamma defined as

$$\Gamma(k, -x_0) = (k-1)! e^{x_0} \sum_{m=0}^{k-1} \frac{(-x_0)^m}{m!}, \quad k \in \mathbb{N}. \tag{2.15}$$

Let us define the coefficients $A_k(x_0, x)$ as follows

$$A_k(x_0, x) = (-1)^k (x - x_0)^k \left(\frac{e^{x_0}}{k x_0^k} \sum_{m=0}^{k-1} \frac{(-x_0)^m}{m!} \right), \tag{2.16}$$

thus, we obtain

$$\mathrm{Ei}(x) = \mathrm{Ei}(x_0) - \sum_{k=1}^{\infty} A_k(x_0, x), \tag{2.17}$$

The basic usage of generalized power series is to perform analytic continuation steps $|x_i - x_0|$ for $i = 1, 2, \ldots$. Note that there exists a trade-off in choosing the steps lengths $|x_i - x_0|$, since larger steps yield slower convergence, i.e., more terms are needed. In order to determine the required numbers of terms given $|x - x_0|$ we calculate two rigorous upper bounds for the regions $x_0 > 0$ and $x_0 \leq -18$. The uncovered region $x_0 \in (-18, 0)$ utilizes the first bound, since it is found that performs reasonably well based on numerical experiments.

Proposition 2.4.1 *For $x_0 > 0$ and $(x, x_0) \in \mathbb{R}$*

$$\frac{|x - x_0|^k}{k} \left(\frac{1}{x_0^k} - \frac{e^{x_0}}{k!} \right) < 2^{-p} |\mathrm{Ei}(x_0)|, \tag{2.18}$$

where p indicates the precision in bits.

Proof: First, by observing that

$$\sum_{m=0}^{k-1} \frac{(-x_0)^m}{m!} = e^{-x_0} - \sum_{m=k}^{\infty} \frac{(-x_0)^m}{m!}. \tag{2.19}$$

The problem is reduced to find k such that $|A_k(x_0, x)|$ is below a given threshold. For $x_0 > 0$ the series is alternating, therefore the following inequality is satisfied

$$e^{-x_0} - \sum_{m=k}^{\infty} \frac{(-x_0)^m}{m!} \leq e^{-x_0} - \frac{|x_0|^k}{k!}, \tag{2.20}$$

plugging into (2.16) and after performing basic manipulations we obtain the result. □

Proposition 2.4.2 *For $x_0 \leq -18$ and $(x, x_0) \in \mathbb{R}$*

$$|A_k(x_0, x)| \leq \left| \frac{x - x_0}{x_0} \right|^k \left(\frac{e^{x_0}}{k!(1 + (k-1)/x_0)} \right) < 2^{-p} |\mathrm{Ei}(x_0)|. \tag{2.21}$$

Proof: We use a majorant $\bar{\Gamma}(k, x_0) \geq \Gamma(k, x_0)$ described in [10] valid when the following two conditions are satisfied

$$x_0 + 1 \geq k \quad \text{and} \quad 1 \geq \frac{1}{\sqrt{1 + \pi(k-1)/2}} + \frac{k-1}{x_0}. \tag{2.22}$$

Based on extensive testing, we restrict the maximum number of terms to $k = 18$, therefore the majorant can be effectively used in this sector. In fact, fixing $k = 18$ is easy to see that above inequalities are satisfied for $x_0 < -17.637$. Hence,

$$\bar{\Gamma}(k, x_0) = \frac{e^{-x_0} x_0^{k-1}}{\left(1 - \frac{(k-1)}{x_0}\right)}. \tag{2.23}$$

□

For both cases we use the relative error with respect to $\mathrm{Ei}(x_0)$. We select the number of terms k by using a simple linear search. A complete algorithm with several optimizations is implemented in `gnstlib.e1_vec_ac`, see benchmarks.

2.4.3 Benchmarks

We show how GNSTLIB compares to other open-source and commercial software for mathematical functions. For benchmarking we use the Python wrapper generated using SWIG (Simplified Wrapper and Interface Generator). Tests were run on an Intel i5-3317U CPU @ 1.70GHz running Linux-Lubuntu 16.10 x86_64 with 4GB RAM. GNSTLIB was compiled with GNU g++ 6.2.0 with the following optimization flags and linked to Intel MKL.

```
-O3 -DMKL_ILP64 -m64 -msse2 -mfpmath=sse -fopenmp -lquadmath
-W -fpic -std=c++11
```

```
MKLROOT="/opt/intel/mkl/"
-Wl,--start-group ${MKLROOT}/lib/intel64/libmkl_intel_ilp64.a
\${MKLROOT}/lib/intel64/libmkl_gnu_thread.a
${MKLROOT}/lib/intel64/libmkl_core.a \
-Wl,--end-group -lquadmath -lgomp -lpthread -lm -ldl
```

Vectorized exponential integral $\mathrm{E}_1(x)$

This benchmark tests the speed of computation of $\mathrm{E}_1(\bar{x}) = -\mathrm{Ei}(-\bar{x})$ for large vectors $\bar{x}$ with values within a reduced range. We test different versions of `gnstlib.e1_vec` and Intel MKL Vector Mathematics functions (VM), which provides optimized vector implementation of several mathematical functions taking advantage of modern SIMD instructions. Timing in (ms) is measured with `%time` function in IPython.

```
>> import gnstlib
>> import numpy as np
>> samples = 10**7 # example
>> v = np.linspace(10.0, 10.2, samples)
>> r = np.empty_like(v)
>> %time gnstlib.e1_vec(v, r, 0)  # sequential
>> %time gnstlib.e1_vec(v, r, 1)  # parallel - 4 threads
>> %time gnstlib.e1_vec_ac(v, r)  # analytic continuation
>> %time gnstlib.e1_vec_mkl(v, r) # Intel MKL vdExpInt1
```

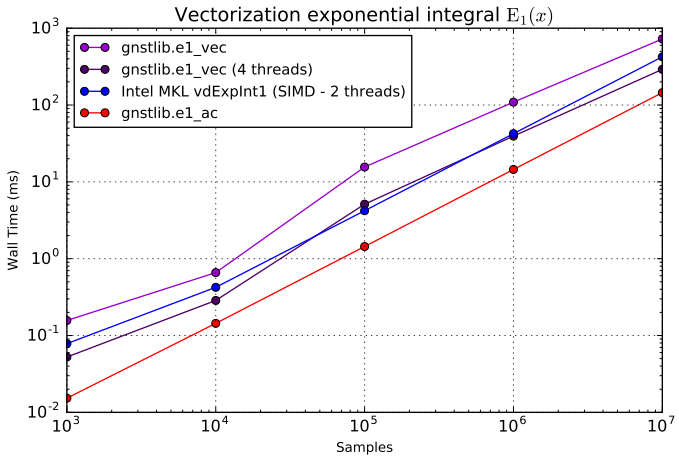

FIGURE 2.8: Comparison `gnstlib.e1_vec` methods to Intel `vdExpInt1`.

N	gnstlib.e1_vec(0)	gnstlib.e1_vec(1)	gnstlib.e1_vec_ac	Intel vdExpInt1
10^3	0.16	0.05	0.02	0.08
10^4	0.66	0.29	0.14	0.42
10^5	15.6	5.1	1.4	4.2
10^6	109	39	15	42
10^7	724	291	145	424

TABLE 2.2: Time (ms) to compute the exponential integral of vector of size N element-wise with values within a reduced range.

`gnstlib.ei_vec_ac` was run with a single thread, so there is significant room for improvement. A parallel implementation can be achieved by splitting the total vector $\bar{x}$ into K blocks, K being the total number of available threads. In this way, the new steps $|x_0^K - x_i^K|$ are smaller and faster local convergence is yielded. On the other hand, a simple approach such as a direct OpenMP instruction `#pragma omp parallel for` on the main loop does not provide a significant speedup.

Exponential integral $\mathrm{E}_1(x)$

Time (ms) to compute $\mathrm{E}_1(x)$ for large vectors $\bar{x} : x_i \in \{1\mathrm{e}{-5}, 700\}$. Comparison to MATLAB R2016b setting the maximum number of threads with `maxNumCompThreads(N)`. Timing in MATLAB is measured using `tic; toc;`. For `gnstlib.e1_vec` timing is measured as in the previous benchmark.

```
>> samples = 10^4
>> v = linspace(1e-5, 700, samples)
>> tic; expint(v); toc;
```

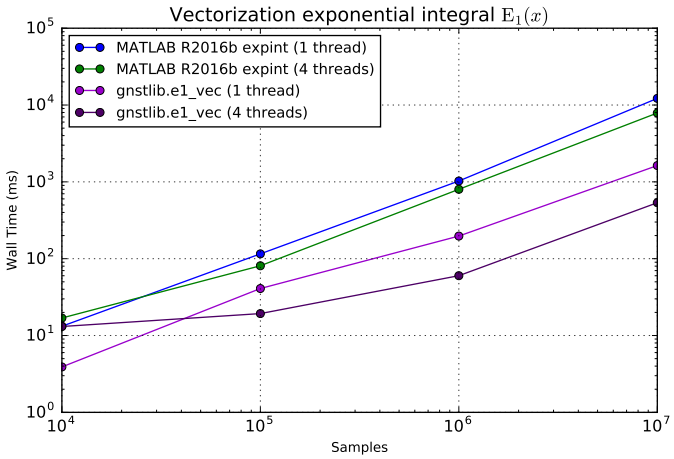

FIGURE 2.9: Comparison `gnstlib.e1_vec` methods to MATLAB R2016b `expint`.

N	gnstlib.e1_vec(0)	gnstlib.e1_vec(1)	matlab.expint(0)	matlab.expint(1)
10^4	3.9	13.1	13.2	16.9
10^5	40.9	19.3	115.6	80.9
10^6	197	60.1	1026	800
10^7	1630	535	12203	7875

TABLE 2.3: Time (ms) to compute the exponential integral of vector of size N element-wise with values in $[0.0001, 700]$.

As usual, we clearly observe that triggering parallelization in `gnstlib.e1_vec` is only worthwhile for large arrays.

Exponential integral Ei(x)

Time (ms) to compute Ei(x) large vectors $\bar{x} : x_i \in \{-670, 670\}$. Comparison to Scipy [95] function `scipy.special.expi`, which calls Cephes C source [111]. Timings in IPython is measured as in the previous benchmarks. Note that Scipy computes Ei(x) sequentially, so a fair comparison is against `gnstlib.ei_vec(0)`.

```
>> import scipy.special
>> samples = 10**4 + 1
>> v = linspace(-670, 670, samples)
>> %time r = scipy.special.expi(v);
```

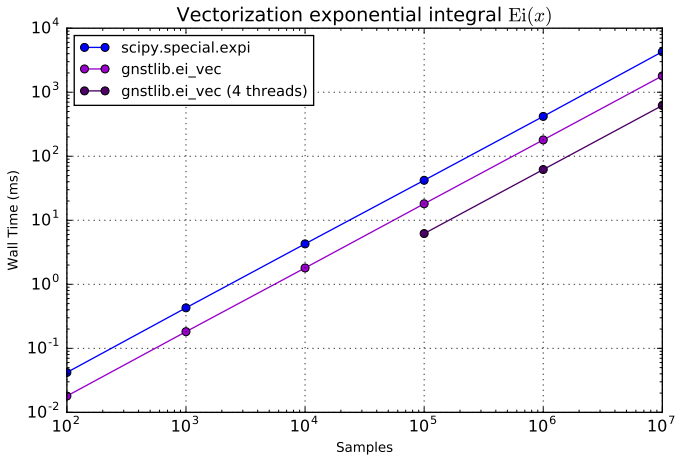

FIGURE 2.10: Comparison `gnstlib.ei_vec` methods to Scipy `scipy.special.expi`.

N	**scipy.special.expi**	**gnstlib.ei_vec(0)**	**gnstlib.ei_vec(1)**
10^2	0.04	0.02	-
10^3	0.43	0.18	-
10^4	4.28	1.80	-
10^4	42.1	18.1	6.20
10^5	419	180	62
10^6	4290	1800	620

TABLE 2.4: Time (ms) to compute the exponential integral of vector of size N element-wise with values in [-670, 670]. Cases where parallelization was slower than single-thread were omitted.

Chapter 3

Fast and Accurate Algorithm for the Generalized Exponential Integral for positive real order

3.1 Introduction

The generalized exponential integral is defined by [43, §8.19.3]

$$E_\nu(x) = \int_1^\infty e^{-xt} t^{-\nu}\, dt, \quad \nu \in \mathbb{R},\ x > 0. \tag{3.1}$$

Another similar integral representation is given by

$$E_\nu(x) = x^{\nu-1} \int_x^\infty e^{-t} t^{-\nu}\, dt, \tag{3.2}$$

or [43, §8.19.4]

$$E_\nu(x) = \frac{x^{\nu-1} e^{-x}}{\Gamma(\nu)} \int_0^\infty \frac{e^{-xt} t^{\nu-1}}{1+t}\, dt, \quad \nu > 0,\ x > 0. \tag{3.3}$$

The generalized exponential integral can also be expressed in terms of the upper and lower incomplete gamma functions ($\Gamma(a,x)$ and $\gamma(a,x)$, respectively) by means of the following functional relations,

$$E_\nu(x) = x^{\nu-1}\Gamma(1-\nu, x), \tag{3.4}$$

$$E_\nu(x) = x^{\nu-1}(\Gamma(1-\nu) - \gamma(1-\nu, x)). \tag{3.5}$$

Recently, numerical methods for the evaluation of incomplete gamma functions have been extensively investigated in [60, 62], therefore those proposed algorithms could be used when $\nu < 0$. Albeit Equation (3.4) is being used in several software packages, its direct application may lead to unsatisfactory results, as will be shown throughout this work. The main contribution of this work is a detailed algorithm for the computation of $E_\nu(x)$ for real and integer order ν, which avoids recursive calculations and includes new numerical methods not present in other existing algorithms. These new computation schema are more efficient and return more accurate results than available software packages in double-precision floating-point arithmetic.

The function $E_\nu(x)$, with $\nu > 0$, appears in many fields of physics and engineering, in particular is of interest its connection with transport theory and radiative equilibrium [1]. For other values of ν, $E_\nu(x)$ is encountered in the computation of molecular electronic integrals in quantum chemistry and wave acoustics of overlapping sound beams. Some well-known examples are the Schwarzschild-Milne integral equation [23] or the generalization of Chandrasekhar's integrals [27]. Besides the applications in physics, the generalized exponential integral arises in several special cases for more difficult special functions, such as the *confluent hypergeometric functions* ${}_1F_1(a;b;x)$ and $U(a,b,x)$. For instance, $E_\nu(x)$ can be defined in terms of $U(a,b,x)$

$$E_\nu(x) = x^{\nu-1}e^{-x}U(\nu,\nu,x) \tag{3.6}$$

and using Kummer's transformation $U(a,b,x) = x^{1-b}U(a-b+1,2-b,x)$ we can write (3.6) in the form

$$E_\nu(x) = e^{-x}U(1,2-\nu,x). \tag{3.7}$$

Moreover, the generalized exponential integral plays an important role in some exponentially improved asymptotic expansions (also known as hyperasymptotic expansions) for the confluent hypergeometric function $U(a,b,x)$ (U-Kummer function) [122] and [43, §13.7(iii)]

$$U(a,b,x) = x^{-a}\sum_{k=0}^{N-1}\frac{(a)_k(a-b+1)_k}{k!}(-x)^{-k} + R_N(a,b,x) \tag{3.8}$$

and

$$R_N(a,b,x) = \frac{(-1)^N 2\pi x^{a-b}}{\Gamma(a)\Gamma(a-b+1)}\left(\sum_{k=0}^{M-1}\frac{(1-a)_k(b-a)_k}{k!}(-x)^{-k}G_{N+2a-b-k}(x) + (1-a)_M(b-a)_M R_{M,N}(a,b,x)\right), \tag{3.9}$$

where M is an arbitrary non-negative integer, and

$$G_p(x) = \frac{e^x\Gamma(p)}{2\pi x^{p-1}}E_p(x), \tag{3.10}$$

$G_p(x)$ being the so-called *terminant function*. Then as $|x| \to \infty$ with a, b and M fixed

$$R_{M,N}(a,b,x) = \begin{cases} O(e^{-x}x^{-M}), & x > 0 \\ O(e^{x}x^{-M}), & x < 0 \end{cases} \tag{3.11}$$

Finally, for a collection of integrals involving $E_\nu(x)$, refer to [27, 102] and [43, §8.19(x)].

An extensive study of different methods for the computation of the generalized exponential integral has been carried out by Chiccoli, Lorenzutta and Maino in [24, 25, 26]. The main method described by the authors, valid for real positive ν and x, is essentially based on recursive calculations starting from a different series

representations for the cases $x < 1$ and $x \geq 1$. This algorithm applies the recurrence

$$E_\nu(x) = \frac{1}{x}(e^{-x} - \nu E_{\nu+1}(x)), \quad (x > 0,\ \nu \in \mathbb{R}) \tag{3.12}$$

and combines Taylor series expansion, series expansion in terms of Tricomi functions and uniform asymptotic expansion for large ν.

The outline of the work is the following: First, we study some of the main numerical methods considered for the evaluation of $E_\nu(x)$, including error bounds, and we describe a new asymptotic expansion along with several improvements for difficult regions of computation. Then, we review other methods of computation implemented in several software packages, focusing in quadrature methods and suitable integral representations. Subsequently, we present our algorithm and a detailed description of its implementation. After that, we assess the performance of our algorithm – in terms of relative error and computation time – and we compare to other publicly available codes. Finally, we present our conclusions.

3.2 Methods of computation

In this section we provide a detailed description of the methods of computation implemented in the algorithm, both for the special case $\nu \equiv n, n \in \mathbb{N}$ and the general case $\nu \in \mathbb{R}^+$.

3.2.1 Special values

Special cases are typically treated separately, here we list some of the most relevant cases:

$$E_0(x) = \frac{e^{-x}}{x}, \quad x \neq 0, \tag{3.13}$$

which can be used as starting point for recursive evaluation of $E_n(x)$. For $x = 0$

$$E_\nu(0) = \begin{cases} \frac{1}{\nu-1}, & \nu > 1 \\ \infty, & \nu \in (-\infty, 1] \end{cases} \tag{3.14}$$

For $\nu = 1/2$ we have the following value, which is also used for recursive evaluation of half-integers

$$E_{\frac{1}{2}}(x) = \frac{\sqrt{\pi}}{\sqrt{x}}\operatorname{erfc}(\sqrt{x}). \tag{3.15}$$

Finally, $\nu = 1$, the so-called *exponential integral*

$$E_1(x) = -\operatorname{Ei}(-x), \quad x > 0. \tag{3.16}$$

3.2.2 Series expansions

Series expansions for the generalized exponential integral are commonly used in the domain $|x| \lesssim 1$. We consider two different series expansions valid when the

parameter $\nu \in \mathbb{R} \setminus \mathbb{N}$. The first expansion [43, §8.19.10] is given by

$$E_\nu(x) = \Gamma(1-\nu)x^{\nu-1} - \sum_{k=0}^{\infty} \frac{(-1)^k x^k}{(1-\nu+k)k!}, \quad \nu \in \mathbb{R} \setminus \mathbb{N},\ x \in \mathbb{R} \setminus \{0\}. \tag{3.17}$$

The terms of the series expansion at the origin decrease for $|x| < 1$, and in practice the best performance is observed in this region, where fast convergence is experimented. The series expansion is alternating and hence the computation of such a sum leads to catastrophic cancellation issues for $x > 1$, especially in finite precision arithmetic. For $x < 0$, ν not being a negative integer, the result is complex. The principal branch of the generalized exponential integral is defined by taking the principal branch of the natural logarithm in $\exp((\nu-1)\log(x)) = x^{\nu-1}$ (with the logarithm branch cut $(-\infty, 0)$). Along the negative real axis, the cancellation errors on the power series are removed, although the required number of terms increases due to slower convergence. For its evaluation, the series is truncated after a level of precision 2^{-p} (p number of bits) is obtained, therefore it can be written as follows,

$$E_\nu(x) = \Gamma(1-\nu)x^{\nu-1} - \left(\sum_{k=0}^{N-1} \frac{(-x)^k}{(1-\nu+k)k!} + \sum_{k=N}^{\infty} \frac{(-x)^k}{(1-\nu+k)k!} \right). \tag{3.18}$$

For $x > 0$, Equation (3.17) is an alternating series and thus the remainder is easily bounded by the first neglected term, therefore we choose N such that $x^N/(|1-\nu+N|N!) < 2^{-p}$. For small values of x, the first neglected term tends rapidly to 0 as $k \to \infty$. However, the required number of terms can grow considerably when $x > 1$, leading to initial ascending terms before the series starts to show convergence, which originates a loss of digits due to roundoff errors.

Series in terms of the confluent hypergeometric function

As previously stated, the generalized exponential integral can be defined in terms of the confluent hypergeometric function of the first kind, ${}_1F_1(a;b;x)$,

$$E_\nu(x) = \Gamma(1-\nu)x^{\nu-1} + \frac{{}_1F_1(1-\nu;2-\nu;-x)}{\nu-1}, \tag{3.19}$$

where ${}_1F_1(a;b;z)$ is defined as

$${}_1F_1(a;b;z) = \sum_{k=0}^{\infty} \frac{(a)_k z^k}{(b)_k k!} \tag{3.20}$$

for $a \in \mathbb{C}$, $b \in \mathbb{C} \setminus \mathbb{Z}_0^-$ and $z \in \mathbb{C}$. Series (3.19) is equivalent to series (3.17), note that $(1-\nu)_k/(2-\nu)_k = (1-\nu)/(1-\nu+k)$. In order to reduce significant cancellation issues, we apply Kummer's transformation ${}_1F_1(a;b;z) = e^z {}_1F_1(b-a;b;-z)$, thus

Equation (3.19) can be written in this form, see [43, §8.19.11]

$$\begin{aligned} E_\nu(x) &= \Gamma(1-\nu)x^{\nu-1} + \frac{e^{-x}}{\nu-1}\,{}_1F_1(1;2-\nu;x) \\ &= \Gamma(1-\nu)x^{\nu-1} + \frac{e^{-x}}{\nu-1}\sum_{k=0}^{\infty}\frac{x^k}{(2-\nu)_k}. \end{aligned} \tag{3.21}$$

This series expansion turns out to be more numerically stable, especially for small values of ν, (for $\nu < 2$ all the terms of the expansion are positive), but the convergence is slightly slower. Similarly to the previous series representation, the number of terms diminish for $|x| < 1$. A rigorous error bound for the evaluation of convergent generalized hypergeometric series ${}_pF_q(a_1,\ldots,a_p;b_1,\ldots,b_q;z)$ in devised in [90].

Expansion (3.21) can be effectively used as an asymptotic expansion for large ν and fixed x. In fact, note that when $\nu \gg x$ the term $\Gamma(1-\nu)x^{\nu-1}$ is not significant compared to the terms in the finite sum and it can be neglected when less than ν terms are used. When $\nu = n$, n being a large integer, and not many terms are considered, we can neglect the contribution of $\Gamma(1-\nu)x^{\nu-1}$ because it should be combined with the $k = n-1$ term, which gives the term shown in series expansion (3.39).

Laguerre series

A globally convergent Laguerre series for the incomplete gamma function $\Gamma(a,x)$ is described in [77]

$$\Gamma(a,x) = x^{a-1}e^{-x}\sum_{k=0}^{\infty}\frac{(1-a)_k}{(k+1)!L_k^{-a}(-x)L_{k+1}^{-a}(-x)}, \tag{3.22}$$

where $L_k^{-a}(-x)$ is a generalized Laguerre polynomial. The region of validity of the Laguerre series is outside the zeros of Laguerre polynomials. Zeros of $L_k^a(x)$ occur in the interval $x \in (0, 4\kappa)$ where $\kappa = n + \frac{1}{2}(a+1)$, so for our region of interest ($x > 0$ and $a = 1-\nu$) $L_k^{-a}(-x)$ lies in the monotonic region. As remarked in [13], the Laguerre series for (3.22) is closely related to the continued fraction for the incomplete gamma function; see §3.3.2 for a comparison to several continued fractions.

Applying the functional relation (3.4) in (3.22) we obtain the Laguerre series for $E_\nu(x)$,

$$E_\nu(x) = e^{-x}\sum_{k=0}^{\infty}\frac{(\nu)_k}{(k+1)!L_k^{(\nu-1)}(-x)L_{k+1}^{(\nu-1)}(-x)}. \tag{3.23}$$

This series has two properties that make it very suitable for our regime of parameters: it does not exhibit cancellations and the error can be made arbitrarily small by increasing the number of terms (in contrast to asymptotic expansions, which have an optimal value).

The generalized Laguerre polynomials satisfy the three-term recurrence relation

$$L_{k+1}^{(\nu-1)}(-x) = \frac{x+2k+\nu}{k+1}L_k^{(\nu-1)}(-x) - \frac{k+\nu-1}{k+1}L_{k-1}^{(\nu-1)}(-x), \tag{3.24}$$

which is not ill-conditioned in both backward and forward direction, consequently it is used with initial values $L_0^{(\nu-1)}(-x) = 1$ and $L_1^{(\nu-1)}(-x) = x + \nu$. Moreover, given the reduced number of terms needed, the loss of precision is almost negligible. In [13] a sub-exponential error bound valid for non-negative real x is given for $L_k^{-a}(-x)$,

$$L_k^{(-a)}(-x) \sim S_k(a,x)\left(1 + O\left(\frac{1}{m^{1/2}}\right)\right), \tag{3.25}$$

where $m = k + 1$ and the sub-exponential[1] term $S_k(a,x)$ is

$$S_k(a,x) = \frac{e^{-x/2}e^{2\sqrt{mx}}}{2\sqrt{\pi}x^{1/4-a/2}m^{1/4+a/2}}. \tag{3.26}$$

This asymptotic estimate results to be particularly effective for large n. For moderate and large values of ν and/or x in a "medium-precision" range[2] the Laguerre series converges after a small number of terms and thus the error term of the sub-exponential estimate is significant. We now proceed to calculate approximations for the coefficients of the Laguerre series define as

$$E_\nu(x) = e^{-x}\sum_{k=0}^{\infty} a_k, \quad \text{and} \quad a_k = \frac{(\nu)_k}{(k+1)!L_k^{(\nu-1)}(-x)L_{k+1}^{(\nu-1)}(-x)}. \tag{3.27}$$

For $k \geq 1$,

$$\sqrt{2\pi}k^{k+1/2}e^{-k} \leq k! \leq ek^{k+1/2}e^{-k},$$

and given $(\nu)_k = \Gamma(\nu+k)/\Gamma(\nu)$ the following inequality [97, Theorem 1] holds for $\Gamma(b)/\Gamma(a)$ and $b > a > 1$,

$$\frac{b^{b-1}}{a^{a-1}}e^{a-b} < \frac{\Gamma(b)}{\Gamma(a)} < \frac{b^{b-1/2}}{a^{a-1/2}}e^{a-b}, \tag{3.28}$$

so $(\nu)_k/(k+1)!$ satisfy

$$\frac{(\nu)_k}{(k+1)!} < C_k(\nu,x) = \left(\frac{\nu}{2\pi(\nu+k)}\right)^{1/2}\frac{(\nu+k)^{\nu+k}e^{k+1}}{\nu^\nu(k+1)^{k+3/2}}. \tag{3.29}$$

For $x \gg \nu$ we shall use the relation $L_k^{\nu-1}(-x) = \frac{(-1)^k}{k!}U(-k,\nu,-x)$ and the property $U(a,b,z) \sim z^{-a}$, $z \to \infty$, $|\mathrm{ph}\, z| < \frac{3}{2}\pi$, therefore

$$L_k^{(\nu-1)}(-x) \sim B_k(x) = \frac{(x)^k}{k!}, \quad x \to \infty. \tag{3.30}$$

Combining (3.29) and (3.30) the coefficients a_k satisfy

$$a_k \approx a_k^B = \frac{C_k(\nu,x)}{B_k(x)B_{k+1}(x)}. \tag{3.31}$$

[1] In [13], sub-exponential growth is defined as $\log\log|f(n)| \sim \delta\log n$, for some $0 < \delta < 1$.

[2] We consider medium-precision the range from double precision to up to a few hundred bits of precision.

When k is small and $x \lesssim \nu$, a first order approximation [13, §1.3] is given by

$$L_k^{(\nu-1)}(-x) \approx A_k(\nu, x) = \binom{k+\nu-1}{k}\left(1+\frac{x}{\nu}\right)^k. \tag{3.32}$$

Again, combining (3.29) and (3.32) we obtain the following approximation for a_k

$$a_k \approx a_k^A = \frac{C_k(\nu, x)}{A_k(x)A_{k+1}(x)}. \tag{3.33}$$

Note that approximation $A_k(\nu, x)$ indicates that for small values of x, $a_k \sim k!/(\nu)_{k+1}$ and the rate of convergence is generally slow, depending entirely on ν. Therefore, if ν is not sufficiently large, the method is not fast enough for efficient numerical evaluation.

Table 3.1 shows the first neglected term $a_N < 2^{-53}$ and the corresponding values for a_N^A and a_N^B. Observe that approximation a_N^A acts as an upper bound for large ν, whereas tends to overestimate a_N for smaller values. Nevertheless, one could easily devise a heuristic in order to compensate that overestimation by adding q extra bits such that $a_N < 2^{-p-q}$, $q < p$. For large x, a_N^B bounds a_N adequately, but is too conservative when $\nu > x$.

ν	x	N	a_N	a_N^A	a_N^B
10	10	17	2.84e−17	3.48e−19	3.99e+00
100	10	10	2.9e−17	2.15e−16	5.66e+06
10	100	6	4.14e−18	3.67e−19	2.63e−17
100	100	7	1.32e−17	1.18e−17	6.26e−13
500	500	5	3.69e−18	3.69e−18	7.94e−15
500	100	6	8.28e−18	8.94e−17	1.17e−07
100	500	5	3.38e−19	2.89e−19	2.75e−18
10000	10	4	2.38e−19	2.44e−19	2.44e+08
10	10000	2	2.19e−18	1.55e−18	2.26e−18

TABLE 3.1: Approximation terms a_N.

In practice, these approximations along with double-precision arithmetic are used to select N using linear search or by inverting these approximations in function of N. This approach is particularly useful in arbitrary-precision interval arithmetic.

Taylor series for $1 \leq x < 2$

As previously stated, the series expansions for $x > 1$ exhibit cancellation issues, therefore other methods need to be used, especially if the working precision cannot be increased to compensate the bad condition number of the series. Henceforth we fix the working precision at 53-bit. For values of x such that $x \in [1, 2)$, we consider the following Taylor series described in [24]

$$E_\nu(x-y) = \sum_{k=0}^{\infty} \frac{y^k}{k!} E_{\nu-k}(x), \quad x > 0,\ (\nu, y) \in \mathbb{R}, \tag{3.34}$$

which is obtained from the Taylor series [24, (10)]

$$E_\nu(x-y) = \sum_{k=0}^{\infty} \frac{(-y)^k}{k!} \left[\frac{\mathrm{d}^k}{\mathrm{d}x^k} E_{\nu-k}(x) \right],$$

making use of the following differential formula [24, (11)]

$$\frac{\mathrm{d}^k}{\mathrm{d}x^k} E_\nu(x) = (-1)^k E_{\nu-k}(x).$$

The Taylor series truncated at $k = N$ is given by

$$E_\nu(x-y) = \sum_{k=0}^{N-1} \frac{y^k}{k!} E_{\nu-k}(x) + \sum_{k=N}^{\infty} \frac{y^k}{k!} E_{\nu-k}(x). \tag{3.35}$$

Proposition 3.2.1 *Given $\nu \geq 1$, $x > 0$ and positive integer N, such that $\lfloor \nu + x - 1 \rfloor > N$, the remainder of the Taylor series in (3.35) for $|y| \leq 1$ satisfies*

$$\sum_{k=N}^{\infty} \frac{y^k}{k!} E_{\nu-k}(x) < \frac{4e^{-x} \lfloor \nu + x - 1 \rfloor}{x} \frac{|y|^N}{N!}. \tag{3.36}$$

Proof: Starting with the integral definition (3.2), it immediately results that the generalized exponential integrals is monotonic increasing as $\nu \to -\infty$, satisfying the inequality $E_\nu(x) > E_{\nu+1}(x)$. This implies that

$$E_r(x) \leq E_0(x) = \frac{e^{-x}}{x}, \quad r \in [0, \infty).$$

With the assumption $\lfloor \nu + x - 1 \rfloor > N$ and using the well-known upper bound for the generalized exponential integral in this domain, the following inequality holds

$$E_{\nu-k}(x) \leq \frac{e^{-x}}{\nu + x - k - 1} < \frac{e^{-x}}{x}.$$

Hence,

$$\sum_{k=N}^{\infty} \frac{y^k}{k!} E_{\nu-k}(x) < \frac{\lfloor \nu + x - 1 \rfloor e^{-x}}{x} \sum_{k=N}^{\infty} \frac{y^k}{k!}.$$

Thus it remains to bound the series expansion. By observing that $\sum_{k=N}^{\infty} y^k/k!$ is equivalent to the remainder term after truncating the Taylor series of e^y, we can use the well-known upper bound for $|y| \leq 1$

$$\sum_{k=N}^{\infty} \frac{y^k}{k!} \leq 4 \frac{y^N}{N!}.$$

Finally, combining both bounds the upper bound for the remainder is obtained. □

By using bound (3.36) we can determine the required number of terms N in order to target a level of precision 2^{-53}. For $x \to 1$ and $y \to -1$, $N = 20$ terms suffice, meaning that $\nu \gtrsim 21$; see Figure (3.1) (left). This constraint demands the use of recurrence relations for smaller values of ν. For example, we can use the following recursion after increasing ν taking care of possible cancellation issues for

$\nu > 1$.

$$E_{\nu-n}(x) = \frac{(1-\nu)_n}{x^n}\left(E_\nu(x) + e^{-x}\sum_{k=0}^{n-1}\frac{x^k}{(1-\nu)_{k+1}}\right), \quad n \in \mathbb{N}. \tag{3.37}$$

Furthermore, by using recursion (3.37), this method can be applied for $\nu < 0$. Regarding the finite series in (3.37), for $\nu - n > 0$ the minimum value occurs for $k \sim \nu$; see Figure (3.1) (right). In order to evaluate $E_{\nu-k}(x)$ we make use of the

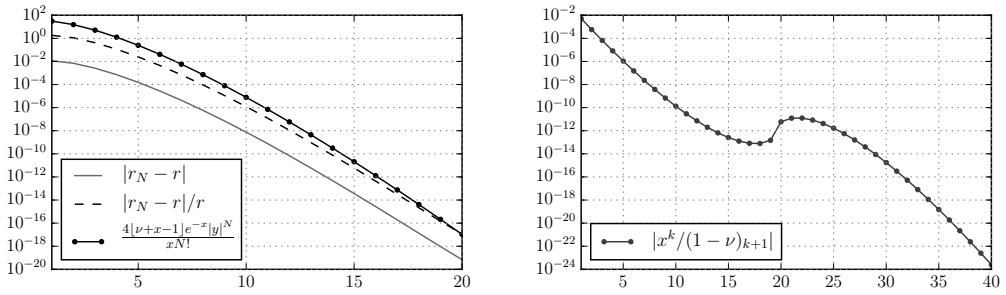

FIGURE 3.1: Plot of the absolute and relative errors of $E_{21.05}(1.98)$ and error bound (3.36) for $N \in [1, 20]$ (left). Plot of $|x^k/(1 - 21.05)_{k+1}|$, $x = 2$ for $k = [1, 40]$ (right).

recurrence relation [43, §8.19.12]

$$E_\nu(x) = \frac{1}{x}(e^{-x} - \nu E_{\nu+1}(x)). \tag{3.38}$$

Hence, we just need to compute the first $E_\nu(x)$ ($k = 0$), in N_1 terms, and for $x \to 1$ and $\nu \approx 21$ we require $N_2 = 18$. Thus, when $x - y \approx 2$, this algorithm would require $N_1 + N_2 + N_3$ terms, where N_3 is the number of recursions in (3.37). For implementation purposes, a simple choice $x = -y = (x - y)/2$ works well, although an iterative procedure could optimize[3] these parameters.

Series expansions: special cases

Previous series expansions, excepting Laguerre series, cannot be used for integer ν. For this special case we introduce two series expansions, see [43, §8.19]:

Case: $n \in \mathbb{N}$

$$E_n(x) = \frac{(-x)^{n-1}}{(n-1)!}(\psi_0(n) - \log(x)) - \sum_{k=0, k\neq n-1}^{\infty}\frac{(-x)^k}{k!(1-n+k)}, \tag{3.39}$$

where $\psi_0(t)$ is the digamma function, which for integer $t = n$ is denoted as

$$\psi_0(n) = -\gamma + \sum_{k=1}^{n-1}\frac{1}{k} = -\gamma + H_{n-1},$$

[3]The optimal choice arises as a solution of a non-convex mix integer nonlinear programming problem, which is prohibitively expensive to compute.

where γ is the Euler-Mascheroni constant and H_n is a harmonic number. Alike the series expansions (3.17) and (3.21), this series performs better for $x < 1$. Other series expansion in terms of the exponential integral $E_1(x)$ (also known as Theis well function) is given by

$$\begin{aligned} E_n(x) &= \frac{(-x)^{n-1}}{(n-1)!}E_1(x) + \frac{e^{-x}}{(n-1)!}\sum_{k=0}^{n-2}(n-k-2)!(-x)^k \\ &= \frac{(-x)^{n-1}}{(n-1)!}E_1(x) + e^{-x}\sum_{k=0}^{n-2}(n)_{-k-1}(-x)^k. \end{aligned} \tag{3.40}$$

Case: $n + \frac{1}{2},\ n \in \mathbb{N}$

$$E_{n+\frac{1}{2}}(x) = \frac{(-1)^n\sqrt{\pi}\,x^{n-\frac{1}{2}}}{(1/2)_n}\operatorname{erfc}(\sqrt{\pi}) - e^{-x}\sum_{k=0}^{n-1}\frac{x^k}{(1/2-n)_{k+1}}. \tag{3.41}$$

This series expansion for half-integers is used for moderate values of n and small x in order to reduce the effect of cancellation.

Case: $n + \epsilon,\ n \in \mathbb{N}$**:** A difficult case arises when $\nu = n + \epsilon$, $|\epsilon| \ll 1$, for small values of x, say $x \lesssim 2$. A direct evaluation of series expansion (3.17) leads to significant loss of precision both in the series expansion and $\Gamma(1-n-\epsilon)x^{n-1+\epsilon}$. Moreover, final subtraction of both terms incurs in catastrophic cancellation, since both are of large magnitude. To deal with this issue, series expansion (3.17) is also implemented in quadruple precision (128-bit) using the libquadmath[4] library. Our implementation includes two interfaces: `expint(const int vi, const double vf, const double x)` and `expint(const double v, const double x)`, where `vi` and `vf` ($|\texttt{vf}| < 0.5$) denote the integral and decimal fractional part of ν, respectively. A unique interface passing $\nu = n + \epsilon$ produces a similar loss of precision, even splitting ν using functions such as `std::modf` in C++. Two relevant examples using a double precision (53-bit) implementation: $\nu = 2 + 1\text{e}{-14}$ and $x = 1\text{e}{-10}$ returns a result with relative error 1.6e-12, whereas $\nu = 1 - 1\text{e}{-13}$ and $x = 1\text{e}{-01}$ has a relative error 6.5e-06. An implementation in quadruple precision returns an relative error below 2^{-53}.

In terms of performance, quadruple precision is about $10 \sim 15$ times slower. Although other approaches are possible, in our experience, quadruple precision is indispensable to return reliable results in this regime of parameters.

[4]There seems to be a bug in libquadmath for `tgammaq` and `lgammaq`. It returns incorrect signs when $\epsilon < 1\text{e} - 6$. This was fixed in our implementation.

3.2.3 Asymptotic expansions

Large x and fixed ν

Asymptotic expansions for $E_\nu(x)$ as $x \to \infty$ can be derived from the integral representation in Equation (3.2)

$$E_\nu(x) = x^{\nu-1} \int_x^\infty e^{-t} t^{-\nu}\, dt = e^{-x} \int_0^\infty e^{-xt} (1+t)^{-\nu}\, dt. \tag{3.42}$$

The transformation gives a Laplace integral and Watson's lemma [165] can be applied, obtaining the following asymptotic expansion

$$E_\nu(x) \sim e^{-x} \sum_{k=0}^{\infty} \frac{(-1)^k (\nu)_k}{x^{k+1}}, \quad x \in \mathbb{R}. \tag{3.43}$$

The remainder $\varepsilon_N(x)$ of the expansion after truncation can be written as follows,

$$E_\nu(x) = e^{-x} \left(\sum_{k=0}^{N-1} \frac{(-1)^k (\nu)_k}{x^{k+1}} + \varepsilon_N(x) \right). \tag{3.44}$$

For $x > 0$, the asymptotic series is alternating, and as previously stated, the remainder can be bounded by the absolute value of the first neglected term,

$$|\varepsilon_N(x)| \leq \left| \frac{(\nu)_N}{x^{N+1}} \right|. \tag{3.45}$$

As pointed out in §3.2.2, in asymptotic expansions the remainder cannot be reduced arbitrarily as $N \to \infty$, in fact, given ν and x, bound (3.45) first decreases until an optimal value of terms $N_{max} = \lceil x - \nu \rceil$ is reached. Subsequent $N > N_{max}$ increase the bound. Thus, linear search is performed up to a limit $N \leq N_{max}$, without guarantee of finding N such that $\varepsilon_N(x) < 2^{-p}$. Hence, when x $(x > 1)$ is not large enough with respect to ν and the required precision bits is moderate/high, this asymptotic expansion cannot be used effectively. For an exponentially-improved asymptotic expansion see [43, §2.11(iii)].

Large ν

A uniform asymptotic expansion when both ν and x are large is introduced in [58]. Using the definition in [43, §8.20(ii)], the expansion is given by

$$E_\nu(x) \sim \frac{e^{-x}}{x+\nu} \sum_{k=0}^{\infty} \frac{A_k(\lambda)}{(\lambda+1)^{2k} \nu^k}, \tag{3.46}$$

where $\lambda = x/\nu$. The coefficients $A_k(\lambda)$, starting with $A_0(\lambda) = 1$, can be computed using the recursion

$$A_{k+1}(\lambda) = (1 - 2k\lambda) A_k(\lambda) + \lambda(\lambda+1) \frac{dA_k(\lambda)}{d\lambda}, \quad k = 0, 1, 2, \ldots \tag{3.47}$$

and the degree of $A_k(\lambda)$ is $k-1$ when $k \geq 1$. In particular, the first 8 coefficients are

$$\begin{aligned}
A_0(\lambda) &= 1,\\
A_1(\lambda) &= 1,\\
A_2(\lambda) &= 1-2\lambda,\\
A_3(\lambda) &= 1-8\lambda+6\lambda^2,\\
A_4(\lambda) &= 1-22\lambda+58\lambda^2-24\lambda^3,\\
A_5(\lambda) &= 1-52\lambda+328\lambda^2-444\lambda^3+120\lambda^4,\\
A_6(\lambda) &= 1-114\lambda+1452\lambda^2-4400\lambda^3+3708\lambda^4-720\lambda^5,\\
A_7(\lambda) &= 1-240\lambda+5610\lambda^2-32120\lambda^3+58140\lambda^4-33984\lambda^5+5040\lambda^6.
\end{aligned}$$

Remark 3.2.2 *It is not hard to observe that polynomials $A_{k+1}(\lambda)$ computed via recursion in (3.47) can be obtained for $k \geq 2$ using the series*

$$A_{k+1}(\lambda) = 1 + \sum_{j=1}^{k-1}(a_j + b_{j-1} + b_j - 2ka_{j-1})\lambda^j + (b_{k-1} - 2ka_{k-1})\lambda^k, \quad k \geq 2$$

where $a_0 = 1$, $b_0 = 0$ and $a_1 = b_1 = -2$. a_j are the coefficients of each polynomial $A_k(\lambda)$ and b_j are the coefficients of their corresponding derivatives. Given the relation $b_j = ja_j$, the above series can be simplified

$$A_{k+1}(\lambda) = 1 + \sum_{j=1}^{k-1}\{a_j(j+1) + (j-1-2k)a_{j-1}\}\lambda^j + \{(k-1-2)ka_{k-1}\}\lambda^k, \quad k \geq 2 \tag{3.48}$$

Remark 3.2.3 *$A_k(\lambda)$ is an Eulerian polynomial of second kind defined by*

$$A_k(\lambda) = \sum_{m=0}^{k}(-1)^m \left\langle\!\left\langle {k \atop m} \right\rangle\!\right\rangle \lambda^m, \tag{3.49}$$

where $\left\langle\!\left\langle {k \atop m} \right\rangle\!\right\rangle$ are second-order Eulerian numbers[5]*, defined by the recursion equation*

$$\left\langle\!\left\langle {k \atop m} \right\rangle\!\right\rangle = (m+1)\left\langle\!\left\langle {k-1 \atop m} \right\rangle\!\right\rangle + (2k-m-1)\left\langle\!\left\langle {k-1 \atop m-1} \right\rangle\!\right\rangle, \tag{3.50}$$

with $\left\langle\!\left\langle {k \atop 0} \right\rangle\!\right\rangle = 1$ and $\left\langle\!\left\langle {k \atop m} \right\rangle\!\right\rangle = 0$ for $m \geq k$.

The remainder after truncating the series $\varepsilon_k(\nu, x)$ satisfies

$$\varepsilon_k(\nu, x) \leq C_k\left(1 + \frac{1}{x+\nu-1}\right)\frac{1}{\nu^k}. \tag{3.51}$$

Gautschi in [58] provides rigorous bounds for $\varepsilon_k(\nu, x)$, $k \leq 7$. This uniform asymptotic expansion proves to be very powerful in a wide domain, $\lambda \in [0, \infty)$. However, each term $A_k(\lambda)$ requires the construction of a polynomial and its evaluation,

[5] https://oeis.org/A008517

which increase the computational time substantially and requires to keep $k-1$ temporary coefficients in cache, which may make it unattractive for arbitrary-precision arithmetic. Nonetheless, for a fixed precision (e.g., 53-bit or 113-bit) one can precalculate as many polynomials as needed[6]. We use Horner's scheme for evaluating the polynomials $A_k(\lambda)$ for $k \geq 3$ to reduce the number of multiplications. Furthermore, we use compensated summation algorithms to minimize roundoff errors.

Large ν and fixed x

We introduce an asymptotic expansion which can be used effectively for $\nu \gg x$. The coefficients of the asymptotic expansion have a relatively simple representation, in contrast to the previous uniform asymptotic expansion. We start with the integral representation $\int_1^\infty e^{-xu}t^{-\nu}\,du$ applying a change of variable $e^t = \phi(u) = 1+u$ to obtain

$$E_\nu(x) = e^{-x}\int_0^\infty e^{-\nu t} f(t)\,dt, \quad f(t) = e^{t-x(e^t-1)}.$$

$f(t)$ is analytic in $[0,\infty)$ and for $x > 0$ the integral is convergent, therefore we can apply Watson's lemma, obtaining the following asymptotic expansion

$$E_\nu(x) \sim e^{-x}\sum_{k=0}^{\infty} c_k \frac{k!}{\nu^{k+1}}, \quad \nu \to \infty, \quad f(t) = \sum_{k=0}^{\infty} c_k(x)t^k, \tag{3.52}$$

where c_k denote the coefficients of the Maclaurin expansion. The function $f(t)$ is a product of two exponential functions, which exponential generating functions are defined by

$$e^t = \sum_{k=0}^{\infty}\frac{t^k}{k!}, \quad e^{-x(e^t-1)} = \sum_{k=0}^{\infty}\frac{B_k(-x)}{k!}t^k, \tag{3.53}$$

where $B_k(x)$ are the Bell polynomials [137]. The Bell polynomials have an explicit formula in terms of Stirling numbers of the second kind denoted as $S(k,j)$ and an infinite series known as Dobiński's formula, respectively

$$B_k(x) = \sum_{j=0}^{k} S(k,j)x^j, \quad B_k(x) = e^{-x}\sum_{j=0}^{\infty}\frac{j^k}{j!}x^j. \tag{3.54}$$

The coefficients c_k are combinations of the coefficients in both exponential generating functions in Equation (3.53), thus we obtain

$$c_k = \sum_{j=0}^{k}\frac{1}{j!}\frac{B_{k-j}(-x)}{(k-j)!} = \frac{1}{k!}\sum_{j=0}^{k}\binom{k}{j}B_{k-j}(-x). \tag{3.55}$$

Let us define the coefficients $d_k = c_k k!$. We observe that d_k is a recurrence formula for Bell polynomials given by $B_n(x) = x\sum_{k=1}^{n}\binom{n-1}{k-1}B_{k-1}(x)$, with $B_0(x) = 1$. Consequently, the coefficients d_k have the following representation, $d_k = -B_{k+1}(-x)/x$. Substituting these coefficients in Equation (3.52), the asymptotic expansion may be

[6] Auxiliary data can potentially compromise thread safety in a multiple processor configuration, therefore it is convenient to avoid its usage.

written in this form

$$E_\nu(x) \sim \frac{e^{-x}}{\nu} \sum_{k=0}^{\infty} \frac{d_k}{\nu^k} = -\frac{e^{-x}}{x} \sum_{k=0}^{\infty} \frac{B_{k+1}(-x)}{\nu^{k+1}}, \quad \nu \to \infty. \tag{3.56}$$

Note that the computation of Bell polynomials with negative argument x leads to substantial cancellation due to the evaluation of large magnitude alternating terms. In order to guarantee the required accuracy, the working precision needs to be increased to at least the exponent of the largest term involved.

Table 3.2 shows the required number of terms k to satisfy $|B_{k+1}(-x)|/\nu^{k+1} < 2^{-53}$ for several values of ν and x when $\nu \gg x$. Numerical experiments reveal that for moderate x not more than roughly 30 terms are needed when $\nu/x \gtrsim 3$.

ν	x	terms	ν	x	terms
20	2	28	1000	200	22
50	10	20	2000	40	10
100	30	21	5000	600	18
500	50	15	5000	10	6

TABLE 3.2: Minimum number of terms k to satisfy $|B_{k+1}(-x)|/\nu^{k+1} < 2^{-53}$ for the asymptotic expansion (3.56).

To determine the number of terms k needed to achieve a required precision 2^{-p}, it is practical to have an upper bound of the truncated term $|B_{k+1}(-x)|/\nu^{k+1}$, particularly to decide whether the asymptotic expansion can be used. In what follows, we proceed to derive an effective upper bound for Bell polynomials for $x \in \mathbb{R}$. In fact, by using Dobiński's formula, the computation of $B_k(x)$ generalizes to $k, x \in \mathbb{C}$, and so thus our upper bound.

An integral representation for Bell polynomials is obtained by direct application of Cauchy's integral formula to the exponential generating function with a parametrization $z(t) = e^{it}$, $t \in [0, 2\pi]$

$$B_n(x) = \frac{n!}{2\pi i} \int_C \frac{e^{x(e^z-1)}}{z^{n+1}}\, dz = \frac{n!}{2\pi} \int_0^{2\pi} e^{x(e^{e^{it}}-1)} e^{-int}\, dt\,. \tag{3.57}$$

Equivalent formulas are given by

$$\begin{aligned} B_n(x) &= \frac{n!}{\pi} \Re\left(\int_0^{\pi} e^{x(e^{e^{it}}-1)} e^{-int}\, dt \right) \\ &= \frac{n!}{\pi e^x} \int_0^{\pi} e^{x(e^{\cos(t)}\cos(\sin(t)))} \cos(nt - x\sin(\sin(t))e^{\cos(t)})\, dt, \end{aligned}$$

where the latter integrand is the real part of $e^{x(e^{e^{it}})}e^{-int}$. In order to compute an effective upper bound for $B_n(x)$ we develop a saddle-point bound[7] [39]. Let us define the function $g(t)$ representing the integrand in Equation (3.57) and its derivative

[7]Saddle-point bound: $|\int_A^B f(t)\, dt\,| \le |C_0||f(t_0)|$, $f'(t_0) = 0$. $|C_0|$ are saddle-point paths made of arcs connecting A and B through the saddle-point t_0.

with respect to t,

$$g(t) = e^{x(e^{e^{it}}-1)-int}, \quad g'(t) = e^{x(e^{e^{it}}-1)-int}(ixe^{it+e^{it}} - in). \tag{3.58}$$

The saddle-point t_0 is the point such that $g'(t_0) = 0$, which is given by

$$t_0 = -i(\log(n/x) - W(n/x)), \tag{3.59}$$

where $W(x)$ is the Lambert-W function which solves $W(x)e^{W(x)} = x$. Substituting t_0 in $g(t)$ we obtain the principal contribution of the bound. It remains to compute the term λ representing the length of the path of the contour joining $[0, 2\pi]$ through t_0 that minimizes $|g(t_0)|$

$$|g(t_0)| = \left| \frac{n!}{2\pi e^x} \frac{e^{xe^{W(n/x)}}}{W(n/x)^n} \right|, \quad \lambda = |0 - t_0| + |2\pi - t_0|. \tag{3.60}$$

Thus, the resulting upper bound for the Bell polynomials is given by

$$|B_n(x)| \leq \lambda \left| \frac{n!}{2\pi e^x} \frac{e^{xe^{W(n/x)}}}{W(n/x)^n} \right|, \tag{3.61}$$

which as aforementioned can be generalized replacing the factorial by the gamma function. Table 3.3 shows the closeness of the upper bound (3.3) for moderate and large values of $n \in \mathbb{N}$ and $x \in \mathbb{R}$.

n	x	$\lvert B_n(x)\rvert$	**Bound (3.61)**	n	x	$\lvert B_n(x)\rvert$	**Bound (3.61)**
30	20	4.01e+44	7.65e+45	30	-20	1.38e+33	1.69e+35
10	200	1.27e+23	1.66e+24	100	-200	8.12e+22	8.66e+23
500	1000/3	1.53e+1356	1.19e+1358	500	-1000/3	1.16e+1179	5.36e+1180

TABLE 3.3: Upper bound for Bell polynomials $B_n(x)$ for $x \in \mathbb{R}$.

For $x = 1$, $B_n(1) = B_n$ is the nth Bell number[8]. As shown in Table 3.4, in turns out that bound (3.61) is sharper than other upper bounds for Bell numbers recently established in [8], especially for moderate and large n, and given by

$$B_n < \left(\frac{0.792n}{\log(n+1)} \right)^n \tag{3.62}$$

n	B_n	**Bound (3.62)**	**Bound (3.61)**	n	B_n	**Bound (3.62)**	**Bound (3.61)**
50	1.9e+47	1.4e+50	7.7e+48	1000	3.0e+1927	2.1e+2059	7.7e+1929
100	4.8e+115	2.9e+123	3.0e+117	10000	1.6e+27664	2.8e+29344	1.6e+27667
500	1.6e+843	1.2e+902	2.7e+845	100000	1.0e+364471	8.2e+383753	3.8e+364474

TABLE 3.4: Upper bound for Bell numbers B_n.

[8]Bell numbers represent the number of class-partitions of a finite set with n elements.

3.3 Other numerical methods

This section presents other numerical methods for the evaluation of the generalized exponential integral. Several of these numerical methods are used in other available implementations, even though we do not employ them in our proposed algorithm in Section 4, due to the existence of more robust and/or faster methods in the same domains of computation introduced in Section 2.

3.3.1 Factorial series

Factorial series are considered an alternative for the summation of divergent inverse power series. The method is a useful numerical tool that can be used for functions defined in terms of Laplace integral, for example integral (3.42), with which we proceed by applying a change of variable $e^{-t} = w$ and $\varphi(w) = (1 - \log(w))^{-\nu}$ to transform into a convergent expansion

$$E_\nu(x) = e^{-x} \sum_{k=0}^{\infty} \frac{a_k k!}{(x)_{k+1}}, \quad \varphi(w) = \sum_{k=0}^{\infty} a_k (1-w)^k, \tag{3.63}$$

where a_k are the coefficients of the Maclaurin series of $\varphi(w)$ at $w = 1$. In particular, the first 7 coefficients are

$$a_0 = 1, \quad a_1 = -\nu, \quad a_2 = \frac{1}{2}\nu^2, \quad a_3 = -\frac{1}{6}\nu(\nu^2+1), \quad a_4 = \frac{1}{24}\nu(\nu^3+4\nu-1),$$
$$a_5 = -\frac{1}{120}\nu(\nu^4+10\nu^2-5\nu+8), \quad a_6 = \frac{1}{720}\nu(\nu^5+20\nu^3-15\nu^2+58\nu-26).$$

This factorial series exhibits fast convergence for moderate values of ν and x when $x > \nu$, and it generally outperforms the asymptotic expansion. However, Laguerre series (3.23) tends to converge more rapidly, whereby factorial series was not included in the proposed algorithm. For an introduction to factorial series we refer to [61, §2.4.4].

3.3.2 Continued fractions

The Stieltjes fraction (S-fraction) [38, §14.1.16] is given by

$$E_\nu(x) = e^{-x}\left(\frac{1/x}{1+} \underset{m=2}{\overset{\infty}{\mathrm{K}}} \left(\frac{a_m/x}{1}\right)\right), \quad x > 0,\ \nu \in \mathbb{R}^+, \tag{3.64}$$

where $a_{2j} = j + n - 1$, $a_{2j+1} = j, \quad j \geq 1$. Since $\lim_{m\to\infty} a_m = +\infty$, the modification [38, §7.7]

$$w_{2k}(x) = \frac{-1+\sqrt{4kx^{-1}+1}}{2}, \quad w_{2k+1}(x) = \frac{-1+\sqrt{4(n+k)x^{-1}+1}}{2}. \tag{3.65}$$

The S-fraction is evaluated using a forward recursion algorithm based on the three-term recurrence relations. This algorithm requires successive rescaling to avoid numerical difficulties. Cephes library implementation includes the S-fraction for the domain $x > 1$ and $\nu \leq 5000$, without modification.

The C-fraction [38, §14.1.19] is given by

$$E_\nu(x) = e^{-x} \mathop{\mathrm{K}}_{m=1}^{\infty} \left(\frac{a_m(\nu) x^{-1}}{1} \right), \quad x > 0,\ \nu \in \mathbb{C}, \tag{3.66}$$

where the coefficients are given by

$$a_1(\nu) = 1, \quad a_{2j}(\nu) = j + \nu - 1, \quad a_{2j+1}(\nu) = j, \quad j \in \mathbb{N}. \tag{3.67}$$

The Jacobi fraction (J-fraction) [38, §14.1.23], obtained by taking the even part of the C-fraction is given by

$$E_\nu(x) = e^{-x} \left(\frac{1}{\nu + x + \mathop{\mathrm{K}}_{m=2}^{\infty} \left(\frac{(1-m)(\nu+m-2)}{\nu + 2m - 2 + x} \right)} \right), \quad x > 0,\ \nu \in \mathbb{C}. \tag{3.68}$$

The C-fraction and J-fraction are evaluated using the modified Lentz algorithm, which is implemented taking into account the suggestions in [61, §6.6.2] to improve numerical robustness. Table 3.5 shows the number of terms and relative error for each continued fraction for regions where asymptotic expansions do not apply due to the amount of terms required. The last column corresponds to the Laguerre series.

ν	x	(3.66)	(3.68)	(3.64)	(3.23)
2.3	1.6	142 (0)	63 (9e−16)	117 (0)	67 (0)
0.3	5.6	52 (9e−16)	23 (9e−16)	41 (3e−16)	21 (0)
10.3	15.6	32 (4e−16)	16 (1e−16)	28 (1e−16)	14 (1e−16)
100.3	15.6	26 (7e−16)	13 (3e−16)	25 (1e−16)	10 (4e−16)
10.3	150.6	14 (7e−16)	8 (0)	13 (2e−16)	6 (2e−16)
100.3	150.6	18 (1e−16)	10 (1e−16)	17 (1e−16)	7 (1e−16)

TABLE 3.5: Comparison of different continued fractions and Laguerre series, number of terms and relative errors. Precision is set to 53-bit.

The results confirm the significant superiority of the Laguerre series with respect to both C-fraction and S-fraction. On the other hand, the J-fraction exhibits rapid convergence, but some loss of precision is observed for small values of ν and x. We refer the reader, e.g., to Cuyt *et al.* [38] for a theoretical background and numerical methods for continued fractions.

3.3.3 Numerical integration

From the definition of the generalized exponential integral (3.2), we apply a change of variable $t = x + q$ to obtain

$$E_\nu(x) = x^{\nu-1} e^{-x} \int_0^\infty e^{-q} (x+q)^{-\nu}\, dq, \tag{3.69}$$

which can be directly evaluated by means of Gauss-Laguerre quadrature. Furthermore, the integrand has the decay property as a single exponential function $q \to \infty$,

therefore we can exploit this by applying a double-exponential transformation (DE-transformation) to an integral over a half-infinite interval. Then,

$$E_\nu(x) = x^{\nu-1}e^{-x}\int_{-\infty}^{\infty} e^{-\phi(t)}(x+\phi(t))^{-\nu}\phi'(t)\,dt, \tag{3.70}$$

where $q = \phi(t) = e^{-t-e^{-t}}$, and $\phi'(t) = (1+e^{-t})\phi(t)$. Another possible change of variable is $\phi(t) = \pi/2\sinh(t)$ and $\phi'(t) = \pi/2\cosh(t)\phi(t)$, although the latter does not provide better results in our experiments. We truncate the infinite summation at $k = -n$ and $k = n$, where the total number of function evaluations is $N = 2n+1$ using the trapezoidal rule with equal mesh size,

$$E_\nu(x)^{(n,h)} = x^{\nu-1}e^{-x}h\sum_{k=-n}^{n} e^{-\phi(kh)}(x+\phi(kh))^{-\nu}\phi'(kh). \tag{3.71}$$

Two methods of computation are used for the evaluation of above integrals; the extended trapezoidal rule and Ooura's implementation for DE-transformation over half-infinite interval [127]. For the extended trapezoidal rule we use symmetric truncations at ± 6, which performs well for moderate values of ν and x.

Other integrals

The integral in §3.2.3, can be computed effectively using numerical quadrature methods, since the integrand decays exponentially for ν and double exponentially for z. If we start with $\lambda = x/\nu$, where $\lambda > 0$, we can write the integral as follows

$$E_\nu(x) = e^{-x}\int_0^\infty e^{-\nu\lambda t}(1+t)^{-\nu}\,dt\,. \tag{3.72}$$

Now by applying a change of variable $e^u/\lambda = \phi(t) = 1+t$, $e^u/\lambda\,du = dt$ we obtain

$$E_\nu(x) = \lambda^{\nu-1}\int_{\log(\lambda)}^{\infty} e^{-\nu e^u}e^{-u(\nu-1)}\,du, \tag{3.73}$$

where the integrand only depends on ν and it is entirely non-increasing. Our experimental results showed that computing these integrals by using the extended trapezoidal rule was 1.5-4 times slower than other available methods in the same domain of computation, consequently these were discarded.

3.4 Algorithm and implementation

We have devised an accurate algorithm along with an efficient implementation in double-precision floating-point arithmetic of $E_\nu(x)$ for integer and real order ν. The program[9] written in C++ is about 800 lines of code and includes python bindings. It is released under MIT license.

Our implementation allows the use of internal computations using higher precision arithmetic implemented in software, in particular we use the so-called *error*

[9] https://sites.google.com/site/guillermonavaspalencia/software/expint.zip

free transformations and double-double numbers, for difficult regions prone to numerical instability, thus diminishing the effect of round off errors. This approach is particularly intended for the evaluation of series expansions, where cancellation errors in the regime $x \in [1, 1.5)$ occur.

A double-double (DD) number is a *multiple-term representation* in which a number is expressed as the unevaluated sum of two standard floating-point (FP) numbers. The DD number is capable of representing at least 106-bit of significant, roughly 31 digits of accuracy and is, therefore, similar to IEEE 754 quadruple-precision. A reference library using this approach is Bailey's library[10] QD [79]. There are several reasons for using DD numbers instead of quadruple-precision (e.g., using libquadmath included in GCC): operations with DD numbers use highly optimized hardware implementation of floating-point operations, and quadruple-precision is still not available for all compilers and programming languages, which would limit the implementation of the algorithm.

An error free transformation (EFT) is an algorithm which transforms any arithmetic operation of two FP numbers a and b into a sum of two FP numbers s and t, a floating-point approximation and an exact error term, respectively. Therefore, these algorithms keep track of all accumulated rounding errors, avoiding the lost of information. The basic brick for our implementation is Algorithm 3. This algorithm requires Algorithm 1 [99], which computes the exact sum of two FP numbers and returns the result under s and t. It requires 6 native FP operations (*flops*).

Algorithm 1 Error-free transformation of the sum of two floating-point numbers.

Input: a, b
Output: s, t
1: **function** TWOSUM(a, b)
2: $\quad s \leftarrow \mathrm{RN}(a + b)$ ▷ RN: Rounding to nearest mode.
3: $\quad c \leftarrow \mathrm{RN}(s - a)$
4: $\quad t \leftarrow \mathrm{RN}(\mathrm{RN}(a - \mathrm{RN}(s - c)) + \mathrm{RN}(b - c))$
5: **end function**

Furthermore, it also uses Algorithm 2 [42], requiring 3 flops. This algorithm is applicable when the exponent of a is larger or equal to that of b.

Algorithm 2 Error-free transformation of the sum of two floating-point numbers ($|a| \geq |b|$).

Input: a, b
Output: s, t
1: **function** FASTTWOSUM(a, b)
2: $\quad s \leftarrow \mathrm{RN}(a + b)$
3: $\quad z \leftarrow \mathrm{RN}(s - a)$
4: $\quad t \leftarrow \mathrm{RN}(b - z)$
5: **end function**

Algorithm 3 computes the exact sum of a DD number and a FP number, storing the resulting operation into the DD number, performing an operation in-place. This

[10] The operations implemented in this library are not compliant with the IEEE 754-2008 standard.

algorithm is used to accumulate the intermediate summation of terms in series expansions (3.17), (3.21) and (3.39) and for the final subtraction or addition operation. Although the use of EFTs for every single operation would definitely enhance the accuracy of the algorithm, we aim to provide a good balance between achievable accuracy and computational time, so that our implementation is competitive with other software packages only using FP operations. Other alternatives to reduce cancellations consist of grouping two consecutive terms in descending order, so the subtraction of the second term does not produce cancellation. However, given the satisfactory results obtained with EFTs, we discarded those methods.

Algorithm 3 Addition of a double-double number and a double number in-place.

Input: sh, sl, a
Output: sh, sl
1: **function** ADD_DD_D_IP(sh, sl, a)
2: $(th, tl) \leftarrow \text{TWOSUM}(sh, a)$
3: $tl \leftarrow \text{RN}(th + tl)$
4: $(sh, sl) \leftarrow \text{FASTTWOSUM}(th, tl)$
5: **end function**

Finally, our algorithm uses some mathematical functions from the standard library defined in `<cmath>`, for instance `tgamma` and `lgamma`, which compute the gamma function and the natural logarithm of the absolute value of the gamma function, respectively.

3.4.1 Algorithm for integer order

The algorithm of integer order $\nu \equiv n, n \in \mathbb{N}$ combines asymptotic expansions, Laguerre series, series expansions and Chebyshev approximations. Laguerre series is the dominant method for $x > 2$ and it is used as a backup method wherever asymptotic expansions are not applicable. For small x series expansions are employed, since they show faster convergence and return more accurate results than Laguerre series. For the special case $n = 1$ we use the Chebyshev approximations in [30], which require fewer terms and provide more accurate results for moderate values of x. Note that we avoid the evaluation of $\text{Ei}(-x)$ in the vicinity of $x_0 \approx -0.372507$, which corresponds to a single zero.

Algorithm 4 Algorithm for $E_n(x)$, $n \in \mathbb{N}$ and $x > 0$

Input: $n \in \mathbb{N}$, $x \in \mathbb{R}$
Output: $E_n(x)$

if $n == 1$ **and** $x \in (0.9, 10)$ **then**
    compute $E_1(x) = -\mathrm{Ei}(-x)$ using Chebyshev approximations [30]
else if $x \leq 1.5$ **and** $n < 20$ **then**
    series expansion (3.39)
else if $x \leq 2.0$ and $n \leq 10$ **then**
    series expansion (3.40) ▷ Faster than Laguerre series
else if $n \geq x$ **then**
    if $x < 5$ **then**
        check if asymptotic expansion (3.56) can be used, otherwise Laguerre series (3.23)
    end if
    Laguerre series (3.23) ▷ Backup method
else
    if $x/\nu > 100$ **then**
        check if asymptotic expansion (3.44) can be used, otherwise Laguerre series (3.23)
    end if
    Laguerre series (3.23) ▷ Backup method
end if

3.4.2 Algorithm for real order

The algorithm for real order differs on the computational methods applied for small x. The alternating series (3.17) converges faster and it is used for large ν/x and as a backup method. Series (3.21) is used when the value ν guarantees that all terms of the expansion are positive. Finally, Laguerre series (3.23) is employed in regions where results returned by series expansions are not sufficiently accurate.

3.5 Benchmarks

Publicly available implementations of the generalized exponential integral in double-precision arithmetic are Cephes [111], Boost [11] and GNU Scientific Library (GSL) [56]. These libraries provide implementations for the special case $\nu \equiv n, n \in \mathbb{N}$. To our knowledge, there are no numerical libraries in double-precision arithmetic implementing $E_\nu(x)$ for real values of ν. We compare our implementation to the aforementioned software for ν integer and to mpmath [92] and Arb [87], both supporting arbitrary-precision arithmetic, for real ν.

To compare our implementation to other software packages we use performance profiles. Performance profiles [44] are widely used tools for benchmarking and evaluating the performance of several solvers, particularly in the fields of optimization and linear algebra, when run on a large test set. Performance profiles provide a convenient procedure of assessing the performance of a code relative to the best code of the set.

Algorithm 5 Algorithm for $E_\nu(x)$, $\nu \in \mathbb{R}^+$ and $x > 0$

Input: $\nu \in \mathbb{R}^+$, $x > 0$
Output: $E_\nu(x)$

if $x \leq 1$ **and** $x < 20$ **then**
  if $\nu/x > 10$ **then**
    series expansion (3.17) ▷ Fast convergence
  end if
  if $\nu > 1.5$ and $x > 0.5$ **then**
    Laguerre series (3.23) ▷ Slow but more accurate
  else if $\nu < 0.9$ **then**
    series expansion (3.21) ▷ All terms of expansion are positive
  else
    series expansion (3.17) ▷ Backup method
  end if
else if $\nu \geq x$ **then**
  if $x < 5$ **then**
    check if asymptotic expansion (3.56) can be used, otherwise Laguerre series (3.23)
  end if
  Laguerre series (3.23) ▷ Backup method
else
  if $x/\nu > 100$ **then**
    check if asymptotic expansion (3.44) can be used, otherwise Laguerre series (3.23)
  end if
  Laguerre series (3.23) ▷ Backup method
end if

Regarding the other codes, Cephes and Boost use similar algorithms, both use continued fractions as a main method and the power series for small x. In addition, Cephes includes the uniform asymptotic expansion in (3.46) for $n > 5000$. The implementation in GSL is purely based on the applications of the functional relation with the incomplete gamma function (3.4). As shown in Figure 3.2 and Table 3.6, this simplistic approach in double-precision has several limitations, especially for large n, as the number of failures indicate.

Figure 3.2 compares all four codes in terms of the relative error using as a reference the value computed by mpmath with 1000 digits of precision. If relevant discrepancies arise using different levels of precision, we use the result from Arb or Mathematica, which tend to be more reliable. The test samples are generated non-uniformly, in fact we select certain input values around regions of transition between computational methods, in order to test the worst cases. Figure 3.3 shows a comparison in terms of computation time. All measurements were obtained by evaluating each test sample 100 times are returning the average time.

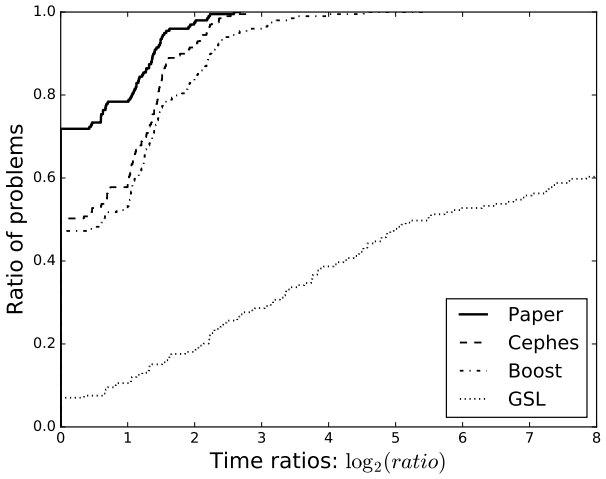

FIGURE 3.2: Accuracy profiles case $n \in \mathbb{N}$ and $x > 0$.

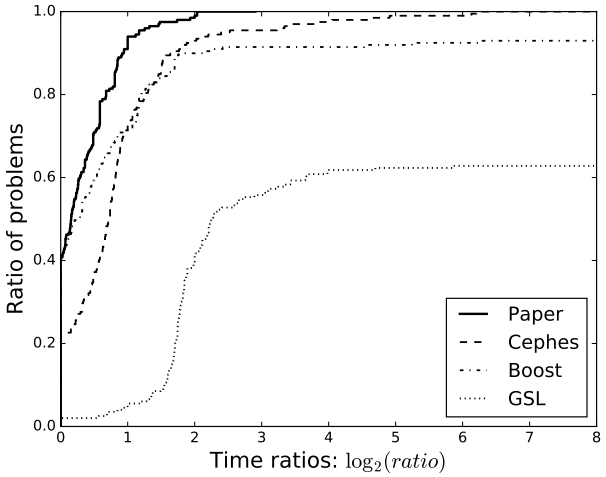

FIGURE 3.3: Performance profiles case $n \in \mathbb{N}$ and $x > 0$.

Observing the performance profiles and the statistics in Table 3.6, it seems safe to claim that our algorithm outperforms the other available codes. In terms of computation time, Boost comes very close in median, however, for some large values of n the computation time is ridiculously high. Cephes does not include an asymptotic expansion for large $n < 5000$, so cases where n is large and $x < 1$ require the computation of n terms, being prohibitively expensive. Nevertheless, Cephes implements a more effective computation scheme than Boost, the former being possibly improved by replacing the S-fraction by the J-fraction, as shown in Section 3.

Library	Max. error	Avg. error	Avg. time (μs)	Stdev. time (μs)	fails
Paper	**9.7e−16**	**1.3e−16**	**0.25**	**0.21**	**0/200**
Cephes	1.4e−15	2.0e−16	0.73	2.47	**0/200**
Boost-1.61.0	4.8e−15	3.3e−16	63.76	558.36	**0/200**
GSL-2.2.1	5.2e−14	6.1e−15	1.34	1.19	75/200

TABLE 3.6: Error statistics for each library. gcc-5.4.0 compiler running Cygwin. Time in microseconds. Fails: returns Incorrect/-NaN/Inf. Intel(R) Core(TM) i5-3317 CPU at 1.70GHz.

For real order ν we have generated two sample sets with the following characteristics:

- Large set: $\nu \in [0.0, 10000]$ and $x \in [10^{-9}, 1000]$
- Small set: $\nu \in [0.04, 70]$ and $x \in [0.00075, 1.5]$

The large set was generated to test the accuracy of asymptotic expansions and Laguerre series, whereas the small set is testing the region in x where a loss of significant digits is expected. As shown in Table 3.7, the maximum relative error determines that about 5 bits of precision might be lost. These results suggest that the computation in this region could be improved by re-implementing both series expansions using DD operations, at the cost of worsening the computation time.

Library	Max. error	Avg. error	Avg. time (μs)	Stdev. time (μs)	fails
Large set	9.8e−16	1.1e−16	0.14	0.10	0/1500
Small set	3.1e−15	1.7e−16	0.52	0.37	0/500

TABLE 3.7: Error statistics for each library. gcc-5.4.0 compiler running Cygwin. Time in microseconds. Fails: returns Incorrect/-NaN/Inf. Intel(R) Core(TM) i5-3317 CPU at 1.70GHz.

3.5.1 Arbitrary-precision floating-point libraries

We evaluate the implementation of the generalized exponential integral in the arbitrary-precision packages Arb 2.8 and mpmath 0.19. Test were run on an Intel(R) Core(TM) i7-6700HQ CPU at 2.60GHz. For testing Arb we use Sage 7.3. Both software packages implement the generalized exponential integral using the functional relation with the incomplete gamma function, which is implemented using the series expansion of the confluent hypergeometric function for small x and the asymptotic expansion of the U-Kummer function for large x, for more details refer to [90]. Table 3.8 summarizes the results obtained by mpmath for all three test sets using 53-bit of precision. Mpmath automatically chooses guard bits to achieve the requested accuracy, however it is usually unable to return a correct results for large parameters. Additionally, mpmath doest not return a flag indicating that the result is incorrect.

Set	Max. error	Avg. error	Avg. time (μs)	Stdev. time (μs)	fails
Integer	1.0	4.4e−13	125	243	5/200
Large real	1.2e+163	2.2e−15	16397	128024	75/1500
Small real	0.0	0.0	672	187	0/500

TABLE 3.8: Error statistics `mpmath` library. Average error is computed after excluding relative errors $\geq$ 1e−10. A result is considered wrong if relative error is $>$ 1e−14.

Arb uses interval arithmetic and efficiently tracks errors. The usage of the functional relation works reasonably well in most of the cases, but as shown below for large parameters one needs to increase the working precision considerably to obtain a solution with 16 significant digits. For example, evaluating $E_{500.25}(400)$ at 53, 1000 and 1210 bits with Arb produces:

```
sage: CBF = ComplexBallField(53)
sage: CBF(400).exp_integral_e(CBF(500+1/4))
nan + nan*I
sage: CBF = ComplexBallField(1000)
sage: CBF(400).exp_integral_e(CBF(500+1/4))
[+/- 5.05e-130]
sage: CBF = ComplexBallField(1210)
sage: CBF(400).exp_integral_e(CBF(500+1/4))
[2.128687916150507e-177 +/- 6.64e-194]

sage: %timeit CBF(400).exp_integral_e(CBF(500+1/4))
1000 loops, best of 3: 1.15 ms per loop
```

Hence, it is necessary to systematically increase the precision in order to obtain results at the desired accuracy. On the other hand, our implementation uses the Laguerre series in this domain reporting fast convergence (5 terms), see below:

```
In [1]: import expint
In [2]: expint.expint_v(500+1/4,400)
Out[2]: 2.128687916150507e-177

In [3]: %timeit expint.expint_v(500+1/4,400)
1000000 loops, best of 3: 475 ns per loop
```

Finally, we encourage the use of more sophisticated computation scheme and addition of specific numerical methods for the computation of the generalized exponential integral in arbitrary-precision arithmetic.

3.6 Conclusions

In this work, we proposed an efficient algorithm for the computation of the generalized exponential integral. The algorithm includes a new asymptotic expansion for large order ν and other methods not implemented in existing software. Numerical experiments confirmed the benefits of using internal higher precision arithmetic for

regions where numerical instability appears, thus obtaining more reliable results. This resulted in an implementation capable of outperforming available software packages in terms of accuracy and computation time.

We believe the improvements and suggested numerical methods in this work should be considered for inclusion in arbitrary-precision arithmetic software packages, which in general implement simplistic computation schema and rely on continuously increasing the working precision to obtain a solution satisfying the user's precision. Finally, our implementation in C++ was made freely available.

Chapter 4

Confluent Hypergeometric Functions

4.1 Background and Previous Work

Confluent hypergeometric functions (also known in the literature as Kummer functions) and Gauss hypergeometric function compound a set of general functions covering a majority portion of the most commonly used special functions. Even though this chapter is focused on confluent hypergeometric functions of the first kind ${}_1F_1$ and the confluent hypergeometric function of the second kind or Kummer U-function U, we consider instructive to provide a minimum background on the *generalized hypergeometric function*. The generalized hypergeometric function of order (p, q), p and q being nonnegative integers representing the number of numerator and denominator parameters, respectively, is usually represented as

$$ {}_pF_q(a_1, \ldots, a_p; b_1, \ldots, b_q; z) = \sum_{k=0}^{\infty} \frac{(a_1)_k \cdots (a_p)_k}{(b_1)_k \cdots (b_q)_k} \frac{z^k}{k!}, \tag{4.1} $$

where $(a)_k$ is the Pochhammer symbol or rising factorial, $a_i,\ \forall i \in \{1, \ldots, p\}$ and $b_i,\ \forall i \in \{1, \ldots, q\}$ are the parameters and z is called the argument. The generalized hypergeometric function is given by a *hypergeometric function*. A hypergeometric function defined as $F(z) = \sum_{k=0}^{\infty} c_k z^k$, is a power series for which coefficients c_k satisfy a first-order recurrence relation, $c_{k+1} = \frac{P(k)}{Q(k)} c_k$, where the ratio of polynomials $P(k)$ and $Q(k)$ form a rational function of k. For numerical computation reasons when b_j is close to a non-positive integer, it is sometimes preferable to use the *regularized generalized hypergeometric function* given by

$$ {}_p\tilde{F}_q(a_1, \ldots, a_p; b_1, \ldots, b_q; z) = \frac{{}_pF_q(a_1, \ldots, a_p; b_1, \ldots, b_q; z)}{\Gamma(b_1) \cdots \Gamma(b_q)}. $$

For certain values of a_i and b_i two important cases are considered: if any $a_i \in \mathbb{Z}_0^-$, the series terminates and the generalized hypergeometric function is a polynomial in z. On the other hand, if any $b_j \in \mathbb{Z}_0^-$, the series is undefined due to zero denominator, except if $a_i > b_j$. The radius of convergence of the series (4.1) is described by three cases.

1. Case $p \leq q$: the series converges for all finite values of z and defines an entire function. This is the case for confluent hypergeometric functions ${}_0F_1$, called confluent hypergeometric limit function, and ${}_1F_1$.

2. Case $p = q + 1$: For $a_i \notin \mathbb{Z}_0^-$, $\forall i \in \{1, \ldots, p\}$, the series has a radius of 1 for $|z| < 1$ and is defined by analytic continuation with respect to z elsewhere.

3. Case $p > q + 1$: the series diverges for all nonzero values of z, except when values of the parameters satisfy some of the aforementioned terminating conditions. The function ${}_2F_0$, which is equivalent to U is the most used hypergeometric function of this kind.

Several infinite power series and partial sums can be written as generalized hypergeometric functions by inspection of the ratios of consecutive terms. There is a large number of identities in terms of ${}_pF_q$ for large orders (p, q) [43, §16.4]. In the field of discrete mathematics, the generalized hypergeometric function arise in many combinatorial identities and random graph theory [84].

4.1.1 Confluent hypergeometric function of the first and second kind

The confluent hypergeometric function of the first kind $M(a, b, z) = {}_1F_1(a; b; z)$ arises as one of the solutions of the limiting form of the hypergeometric differential equation

$$z\frac{d^2w}{dz^2} + (b - z)\frac{dw}{dz} - aw = 0, \tag{4.2}$$

for $b \notin \mathbb{Z}_0^-$, see [43, §13.2]. ${}_1F_1(a; b; z)$ is entire in z and a and is a meromorphic function of b, however, its regularized form $\Gamma(b){}_1\tilde{F}_1(a; b; z) = {}_1F_1(a; b; z)$ is entire in b since its limiting form is well defined

$$\lim_{b\to -m} \frac{{}_1F_1(a; b; z)}{\Gamma(b)} = \frac{(a)_{m+1}z^{m+1}}{(m+1)!}\,{}_1F_1(a + m + 1; m + 2; z).$$

Another standard solution is the confluent hypergeometric function of the second kind (also called Kummer U-function or Tricomi confluent hypergeometric function, indistinctly) $U(a, b, z)$, which is defined by the property $U(a, b, z) \sim z^{-a}$, $z \to \infty$, $|\mathrm{ph}\, z| \leq (3/2)\pi - \delta$ where δ is an arbitrary small positive constant such that $0 < \delta \ll 1$.

For certain regimes of the parameters, these functions are expressible as polynomials in z. In particular, when $a \in \mathbb{Z}_0^-$, $U(a, b, z)$ is a polynomial of degree m

$$U(-m, b, z) = (-1)^m(b)_m\,{}_1F_1(-m, b, z) = (-1)^m \sum_{k=0}^{m} \binom{m}{k} (b + k)_{m-k}(-z)^k.$$

Similarly, when $a - b + 1 \in \mathbb{Z}_0^-$,

$$U(a, a + m + 1, z) = (-1)^m(1 - a - m)_m z^{-a-m}\,{}_1F_1(-m, 1 - a - m, z).$$

For ${}_1F_1$ the previous polynomials can be used after application of connection formulas as shown later on. However, we mention a few terminating series in terms of the lower incomplete gamma function, see http://functions.wolfram.com/07.20.03.0106.01

$${}_1F_1(a; a + m; z) = \frac{\Gamma(a + m)(-z)^{-a}}{\Gamma(a)\Gamma(m)} \sum_{k=0}^{m} \binom{m-1}{k} \gamma(a + k, -z)z^{-k}, \quad m \in \mathbb{N}^+,$$

and http://functions.wolfram.com/07.20.03.0115.01

$$ {}_1F_1(m,b,z) = \frac{\Gamma(b)e^z z^{m-b}}{\Gamma(b-m)\Gamma(m)} \sum_{k=0}^{m} \binom{m-1}{k} \gamma(b+k-m,z)(-z)^{-k}, \quad m \in \mathbb{N}^+. $$

We refer to [60] for numerical algorithms to compute the incomplete gamma function for real a and negative argument z, where the function well-behaved $\gamma^*(a,z) = z^{-a}\gamma(a,z)/\Gamma(a)$ real for real values of a and z is computed instead.

Kummer's transformations [43, §13.2] are particularly useful to handle regimes of parameters prone to numerical instabilities, such as the case $z < 0$ for ${}_1F_1$, or in situations where the parameters are not directly valid for some methods,

$$ {}_1F_1(a;b;z) = e^z \, {}_1F_1(b-a;b;-z) \tag{4.3} $$

and

$$ U(a,b,z) = z^{1-b}U(a-b+1,2-b,z). \tag{4.4} $$

Connection formulas for confluent hypergeometric functions for $(a,b,z) \in \mathbb{C}^3 : z \neq 0$ are

$$ {}_1\tilde{F}_1(a;b;z) = \frac{e^{\mp\pi i a}}{\Gamma(b-a)}U(a,b,z) + \frac{e^{\pm i(b-a)}}{\Gamma(a)}e^z U(b-a,b,ze^{\pm\pi i}), \tag{4.5} $$

or

$$ U(a,b,z) = \frac{\pi}{\sin(\pi b)} \frac{{}_1\tilde{F}_1(a;b;z)}{\Gamma(a-b+1)} - \frac{z^{1-b} \, {}_1\tilde{F}_1(a-b+1;2-b;z)}{\Gamma(a)} \tag{4.6} $$

and when $b \notin \mathbb{Z}$

$$ U(a,b,z) = \frac{\Gamma(1-b)}{\Gamma(a-b+1)} \, {}_1F_1(a;b;z) + \frac{z^{1-b}\Gamma(b-1)}{\Gamma(a)} \, {}_1F_1(a-b+1;2-b;z). \tag{4.7} $$

Connection formulas (4.6) and (4.7) are implemented in some numerical libraries to handle cases for $U(a,b,z)$ when $|z|$ is not sufficiently large to use asymptotic expansions. On the contrary, formula (4.5) is an alternative to compute ${}_1F_1$ for large values of $|z|$. In general, most of the methods for computing confluent hypergeometric functions are interchangeable by making use of connection formulas and Kummer transformations. However, specific methods usually lead to more stable algorithms and therefore more accurate results.

4.1.2 Computational methods and available software

Most numerical libraries implementing confluent hypergeometric functions in double-precision arithmetic combine various numerical methods to cover, in many cases, the functions' domains encountered in many applications. In this section, we survey the most common computational methods implemented in open-source and commercial software. More advanced methods shall be discussed in subsequent sections.

Series representations. The most standard definition of the confluent hypergeometric function of the first kind is given by the Maclaurin series expansion

$$ {}_1F_1(a;b;z) = \sum_{k=0}^{\infty} \frac{(a)_k}{(b)_k} \frac{z^k}{k!}, \tag{4.8} $$

convergent for all finite values of z. The convergent Maclaurin series expansion is the building block of most of the available implementations for ${}_1F_1(a;b;z)$ and $U(a,b,z)$. As previously discussed, series expansion are prone to suffer cancellation as the argument z increases. In particular, for $z \in \mathbb{R},\ z < 0$, Kummer transformation (4.5) should be used. Several methods for computing the Taylor series are investigated in [133] such as direct hypergeometric recursion, three-term recurrence equation or single fraction computation, all of them exhibiting similar accuracy and performance. For numerical computation in double-precision arithmetic, compensated summation methods (see Chapter 2) are usually implemented to extend the region of z where series (4.8) can be evaluated effectively. By examining the coefficients of the Maclaurin series we observe that (4.8) is indeed an asymptotic expansion of b when a and z are fixed.

For large $|a|$ or real a and z such that $sign(a) \neq sign(z)$, alternatives series representations in terms of Bessel functions of the first kind, $J_\nu(z)$, are available in [162] and [22] . The expansion in terms of Buchholz polynomials, $p_k(b,z)$, is given by

$$ {}_1F_1(a;b;z) = \Gamma(b) e^{z/2} 2^{b-1} \sum_{k=0}^{\infty} p_k(b,z) \frac{J_{b-1+k}(\sqrt{z(2b-4a)})}{(\sqrt{z(2b-4a)})^{b-1+k}}, \tag{4.9} $$

where Buchholz polynomials are defined as

$$ p_k(b,z) = \frac{(iz)^k}{k!} \sum_{j=0}^{\lfloor k/2 \rfloor} \binom{k}{2j} f_j(b) g_{k-2j}(z), $$

and polynomials $f_j(b)$ and $g_j(z)$ are defined by the following recurrence equations

$$ f_j(b) = -(b/2-1) \sum_{r=0}^{j-1} \binom{2j-1}{2r} \frac{4^{j-r} |B_{2(j-r)}|}{j-r} f_r(b), \quad f_0(b) = 1, $$

$$ g_j(z) = -\frac{iz}{4} \sum_{r=0}^{\lfloor (j-1)/2 \rfloor} \binom{j-1}{2r} \frac{4^{r+1} |B_{2(r+1)}|}{r+1} g_{j-2r-1}(z), \quad g_0(z) = 1, $$

and B_r denote the Bernoulli numbers. The coefficients of Buchholz's expansion, unlike the ones from Tricomi's expansion, are independent of a, being preferable even though their computation is more involved.

To compute the Kummer function $U(a,b,z)$ for small values of z, the usual approach is to employ connection formulas for this function in terms of ${}_1F_1(a;b;z)$, for example (4.6) or (4.7), which is not defined for integer values of b, although the limit exists for $b \to 0$. Additionally, a recent method for computing the Kummer function $U(a,b,z)$ for small values of $|a|$, $|b|$ and $|z|$ is described in [63].

Asymptotic expansions for large argument z**.** Ascending series expansion (4.8), even though convergent in nature, is not adequate for large values of $|z|$ when

performing computations using double-precision arithmetic, and it is costly at arbitrary-precision due to the necessity of increasing the working precision to compensate cancellation. In what follows, we introduce the basic asymptotic formulas for computing the confluent hypergeometric functions when $|z| \to \infty$.

The asymptotic expansion of $U(a,b,z)$ results from the application of Watson's lemma to the Laplace integral representation in (4.13), which gives us [43, §13.7.3]

$$U(a,b,z) \sim z^{-a} \sum_{k=0}^{\infty} \frac{(a)_k (a-b+1)_k}{k!(-z)^k}, \tag{4.10}$$

$|\mathrm{ph}\, z| < 3/2\pi$. Note that the Kummer U-function is related to the hypergeometric function ${}_2F_0$ via $U(a,b,z) = z^{-a}\, {}_2F_0(a, a-b+1, -1/z)$, which exhibits its divergent behavior. An effective bound is detailed in [43, §13.7]. For $\Re(z) > 0$, asymptotic series (4.10) is alternating and thus the remainder is bounded by the absolute value of the first neglected term. As previously discussed, the remainder cannot be reduced arbitrarily, hence when z is not sufficiently large with respect to a and b (not made rigorous here) and the required precision bits is moderate, this expansion cannot be used effectively. Evaluation of $U(a,b,z)$ outside the sector $|\mathrm{ph}\, z| < \frac{1}{2}\pi$ can be achieved by use of the continuation formula (4.5).

Similarly, one can obtain the asymptotic expansion for ${}_1F_1(a;b;z)$ using (4.12) resulting in

$${}_1F_1(a;b;z) \sim \frac{\Gamma(b) e^z z^{a-b}}{\Gamma(a)} \sum_{k=0}^{\infty} \frac{(b-a)_k (1-a)_k}{k! z^k}, \tag{4.11}$$

given $a \notin \mathbb{Z}_0^-$ and $|\mathrm{ph}\, z| < \pi/2$. The region of validity of the expansion can be easily extended by making a compound expansion using the connection formula (4.5) and asymptotic expansion (4.10)

$${}_1F_1(a;b;z) \sim \frac{e^z z^{a-b}\Gamma(b)}{\Gamma(a)} \sum_{k=0}^{\infty} \frac{(b-a)_k (1-a)_k}{k! z^k} + \frac{e^{\pm\pi i a} z^{-a}\Gamma(b)}{\Gamma(a)} \sum_{k=0}^{\infty} \frac{(a)_k (a-b+1)_k}{k!(-z)^k},$$

which is valid in sectors $-\pi/2 < \pm\mathrm{ph}\, z < 3/2\pi$.

An alternative asymptotic expansion of $U(a,b,z)$ in the form of a convergent expansion involving a series of Kummer U-functions described in [157, §17.2] is given by

$$U(a,b,z) = z^{-a} \sum_{k=0}^{\infty} \frac{(a)_k (b-a-1)_k}{k!} U(k, 1-a, z),$$

suitable for fixed a and $|z| \to \infty$ when $|\mathrm{ph}\, z| < \pi/2$. An exhaustive analysis of this kind of expansion is conducted in [41], where numerical examples and a recursive scheme of computation using continued fractions in backward direction is included.

Asymptotic expansions for large parameters. An excellent survey of asymptotic expansions for the confluent hypergeometric functions for large parameters can be found in [157, §10]. In this section, we briefly comment those more relevant results and refer to the previous book for details.

Asymptotic expansions for large values of a or b have been extensively studied in the last 50 years. First asymptotic expansions for large a for ${}_1F_1(a;b;z)$ and $U(a,b,z)$ are given in [143], where expansions are derived from the Kummer differential equation (4.2). Extensions for $a \to \pm\infty$ in terms of modified Bessel functions are derived in [153] and [157, §10.3, §27.4].

A direct asymptotic expansion of ${}_1F_1(a;b;z)$ for large b follows the series expansion in (4.8). Uniform asymptotic expansions for large b and z are derived in [152], in terms of parabolic cylinder functions, and [107].

The coefficients of the aforementioned asymptotic expansions and some uniform variants are generally involved and require expensive computations which make them unattractive to be implemented in numerical libraries. As a constructive criticism, many classic works in asymptotic analysis only provide a few coefficients computed using computer algebra systems rather than a general expression. Even though this might be sufficient from a theoretical perspective, it is limiting for its successful implementation. To the author's knowledge the algorithm in [154] for the numerical evaluation of $U(a,b,z)$ for $(a,b,z) \in \mathbb{R}^3$ and $z > 0$, is one of the few exceptions, given that it is the primary method implemented in the AMath special functions library [1] written in Pascal.

Integral representations. As stated in [43, §13.4.1, §13.4.4], the functions ${}_1F_1(a;b;z)$ and $U(a,b,z)$ have the following integral representations, respectively

$$ {}_1F_1(a;b;z) = \frac{\Gamma(b)}{\Gamma(a)\Gamma(b-a)} \int_0^1 e^{zt} t^{a-1} (1-t)^{b-a-1}\, dt, \quad \Re(b) > \Re(a) > 0 \tag{4.12} $$

$$ U(a,b,z) = \frac{1}{\Gamma(a)} \int_0^\infty t^{a-1} e^{-zt} (1+t)^{b-a-1}\, dt, \tag{4.13} $$

valid for $\Re(a) > 0$ and $\Re(z) > 0$. As we shall see in subsection 4.3, Laplace-type integral (4.13) is a convenient starting point for developing methods for $U(a,b,z)$ exploiting the amplitude function $f(t) = (1+t)^{b-a-1}$. Contour integrals also play an important role on the derivation of powerful asymptotic expansions, see [43, §13.4] for a complete list.

The two integrals along the real line are generally included in implementations in double-precision arithmetic taking advantage of available quadrature methods. The most common method is the use of Gauss-Jacobi quadrature applied to integral (4.12). In order to obtain an integral in the standard form of Gauss-Jacobi quadrature

$$ I(\alpha,\beta) = \int_{-1}^1 f(x)(1-x)^\alpha (1+b)^\beta, $$

we apply the transformation $t \to s/2 + 1/2$ and taking $\alpha = b-a-1$ and $\beta = a-1$ yields the standard form

$$ {}_1F_1(a;b;z) = \frac{2^{1-b}\Gamma(b)}{\Gamma(a)\Gamma(b-a)} \int_{-1}^1 e^{s/z+1/2} (1-s)^{b-a-1} (1+s)^{a-1}\, ds\,. \tag{4.14} $$

[1] http://www.wolfgang-ehrhardt.de/amath_functions.html

See [59] for more details about effectiveness of the Gauss-Jacobi quadrature. Other more general quadrature methods such as Gauss-Legendre, double-exponential or Clenshaw-Curtis quadrature are also amenable for the evaluation of both integrals after proper transformations. In general, as long as the imaginary part of the parameters and/or argument is bounded and the integrand decays smoothly, the aforementioned methods are suitable. An extensive treatment of quadrature methods for large imaginary part of parameters and argument is given in Section 4.2 of this thesis, where the behaviour of the integrals is highly oscillatory.

Other methods. Several numerical methods and techniques not previously discussed have been investigated to compute confluent hypergeometric functions. Probably, the most helpful technique is the use of recurrence relations (using either Miller's or Olver's algorithm) to move parameters to regimes where previous methods are usable. Besides, a rational approximation is generally applied as a backup method [108].

Moreover, for many special values of the parameters, confluent hypergeometric functions are expressible in terms of less complex special functions such as Bessel function and generalized exponential integrals (see Chapter 3), to mention a few. Efficient software implementations handle these special cases effectively.

Software. In this section we list open-source and commercial software packages including implementations for the confluent hypergeometric functions. We note that, specially for double-precision arithmetic, implementations for complex parameters and/or argument are rarely available. On the other hand, arbitrary-precision arithmetic libraries tend to cover most of regimes of the parameters and argument. An extensive benchmarking of open-source and commercial software is conducted in [90].

Double-precision arithmetic software:

- GNU Scientific Library (C/C++) [56]: provides implementations for real parameters and argument of ${}_0F_1$, ${}_1F_1$, U, ${}_2F_1$ and ${}_2F_0$. In particular, for ${}_1F_1(a;b;x)$ and $U(a,b,x)$ we have `gsl_sf_hyperg_1F1` and `gsl_sf_hyperg_U`, respectively. Furthermore, this library provides special routines returning a result with extended range, which aim to avoid numerical issues with overflow and/or underflow.

- Cephes (C) [111]: File `hyperg.c` implements ${}_1F_1$ and ${}_2F_0$ and `hyp2f1.c` implements ${}_2F_1$, both routines for real parameters and argument. The implementation for ${}_1F_1(a;b;x)$ combines the Maclaurin series expansion (4.8) and the asymptotic expansion in terms of (4.10) along with Kummer's transformation.

- SLATEC (Fortran 77) [163]: Function `chu.f/dchu.f` implements $U(a,b,x)$ (referenced as logarithmic confluent hypergeometric function) for real parameters and argument combining the particular cases, Macluarin series (4.8) with connection formula (4.6) and Luke rational approximation [108] in the asymptotic region.

- Scipy (Python/Cython) [95]: The module `scipy.special` includes implementations for ${}_0F_1$, ${}_1F_1$, U, ${}_1F_2$, ${}_2F_0$, ${}_2F_1$ and ${}_3F_0$. Functions `hyp1f1` and `hyperu` are implemented for real parameters and argument. As experienced in Github[2], implementation of confluent hypergeometric functions in double-precision arithmetic is a rather challenging assignment. We note that some of the previous functions are wrapped from the Cephes library and Fortran implementations available at [171].

- AMath (Pascal/Delphi) [46]: This library implements functions ${}_0F_1$, ${}_1F_1$, U, ${}_2F_0$, ${}_2F_1$ and regularized versions in double-precision and extended-precision (80 bit) arithmetic for real parameters and argument. Many of the previously discussed methods and several special cases are considered, being one of the most advanced open-source implementations.

- NAG (Fortran) [114]: This commercial library includes implementations of ${}_1F_1$ and ${}_2F_1$ for real parameters and argument. Routines `s22ba/s22bb` for ${}_1F_1$ and `s22be/s22bf` for ${}_2F_1$ compute the value of the function in standard and scaled form, respectively. Routines `s22bb` and `s22bf` accept parameters split into an integral and decimal fractional component, which can improve the precision in the final result.

  To the author's knowledge, NAG and AMath implementations stand out by being the most robust and reliable. In general, old implementations vaguely include a few methods which might successfully cover a decent amount of the regimes of parameters. However, these cannot be safely included in applications with very varying parameters. AMath library would be more noticeable by including comprehensive interfaces to more popular programming languages such as Python or Java, for instance.

Arbitrary-precision arithmetic software:

- mpmath (Python/Cython)[92]: provides implementations for real and complex parameters and argument of ${}_0F_1$, ${}_1F_1$, U, ${}_1F_2$, ${}_2F_0$, ${}_2F_1$, ${}_2F_2$, ${}_2F_3$ and ${}_3F_2$. In particular, for ${}_1F_1(a;b;z)$ and $U(a,b,z)$ we have `hyp1f1` and `hyperu`.

- Arb (C) [87]: This highly-optimized library includes implementations for real and complex parameters and argument of ${}_0F_1$, ${}_1F_1$, U and ${}_2F_1$. Besides hypergeometric auxiliary functions, the main routines are: `acb_hypgeom_0f1`, `acb_hypgeom_m_1f1/acb_hypgeom_m`, `acb_hypgeom_u` and `acb_hypgeom_2f1`.

- Matlab (C/C++) [158]: This software provides arbitrary-precision and symbolic computation of the generalized hypergeometric function ${}_pF_q$ and its special cases. In particular, routine `hypergeom` is used to compute ${}_0F_1$ and ${}_1F_1$, whereas `kummerU` computes U.

- Mathematica (C/C++) [169]: This software provides arbitrary-precision and symbolic computation of the generalized hypergeometric function ${}_pF_q$, corresponding regularizations and many special cases. Function `HypergeometricPFQ` covers all cases, but functions `Hypergeometric0F1`, `Hypergeometric1F1`,

[2] https://github.com/scipy/scipy/issues?utf8=%E2%9C%93&q=is%3Aopen+label%3Ascipy.special+hypergeometric

`HypergeometricU` and `Hypergeometric2F1` compute ${}_0F_1$, ${}_1F_1$, U and ${}_2F_1$ as well.

- Maple (C) [109]: This software provides the function `hypergeom` to compute a floating-point approximate value of ${}_pFq$ and `Hypergeom` to evaluate ${}_pFq$ symbolically.

  To the author's knowledge/experience Arb and Mathematica are generally faster and more reliable than mpmath, Matlab or Maple. Matlab is observed to be significantly slower than other software libraries. Finally, we note that Arb and Mathematica are chosen for benchmarking throughout this work.

4.1.3 Applications

In this section, we mention several applications where hypergeometric functions are encountered. A complete and up to date survey of their applications would hardly be achievable. Therefore, we aim to introduce applications that might be less well-known or that are of interest on current relevant research topics. Other specific applications, related to numerical methods developed in subsequent sections, will be discussed further on.

Special functions. There are many special functions appearing in engineering, physics and mathematics which are particular cases of the hypergeometric function. The following list presents the most common derived cases of confluent hypergeometric functions and Gauss hypergeometric function.

- Confluent hypergeometric functions:
  - Elementary functions.
  - Exponential, logarithmic and trigonometric integrals.
  - Error functions, Faddeeva function, Fresnel integrals and Dawson's and Goodwin-Staton integral.
  - Incomplete gamma functions and generalized exponential integral.
  - Airy and Scorer functions.
  - Bessel, Kelvin, Struve and Lommel functions.
  - Parabolic cylinder functions.
  - Whittaker functions.
  - Hermite and generalized Laguerre orthogonal polynomials.
- Gauss hypergeometric function:
  - Incomplete beta functions.
  - Chebyshev, Gegenbauer, Jacobi and Legendre orthogonal polynomials.
  - Spherical and spheroidal harmonics.
  - Complete elliptic integrals.

Physics. Confluent hypergeometric functions, their special cases and especially the closely related Whittaker functions are ubiquitous in physics. They appear in the study of potentials in quantum mechanics, the solution of eigenvalue problems, dynamics of many-body systems and relativistic cosmology to mention a few. As recent application, we note that confluent hypergeometric functions are often used to describe first passage of Ornstein-Ulhenbeck processes [70]. It is worthwhile to mention that the confluent hypergeometric function ${}_1F_1$ is used for the solution of time-dependent Schrödinger equation for an atom in the laser field, where fast multi-evaluation over a grid is required during simulation processes[3].

Other cases of the hypergeometric function such as ${}_3F_2$ arise in considering coupled angular momenta in two quantum systems, where Wigner 3 j-symbols are used [141]. Last, single variable hypergeometric functions ${}_pF_q$ arise in applications in quantum and statistical physics and chemistry as described in [31].

Mathematics. In particular, Gauss hypergeometric function ${}_2F_1$ arises in several applications in number theory; complete elliptic integrals of the first and second kind are expressible in terms of ${}_2F_1$, $K(k) = \pi/2\,{}_2F_1(1/2, 1/2; 1; k^2)$ and $E(k) = \pi/2\,{}_2F_1(-1/2, 1/2; 1; k^2)$, respectively. Furthermore ${}_2F_1$ is used to compute the inverse of the j-invariant in parametrization of elliptic curves over $\mathbb{C}$. Additionally, the hypergeometric function arise as matrix coefficients of representations of Lie groups.

We mention the connection between hypergeometric functions and algebraic equations, for example the roots of the general quintic equation are expressible in terms of ${}_4F_3$, which can be found in [98]. Combinatoric interpretations of the hypergeometric function and general hypergeometric sums, together with computational aspects about their identification, are studied in depth in the already classic book [134]. Finally, an extensive survey of classic applications of basic hypergeometric functions to partitions, number theory, finite vector spaces and combinatorial identities is presented in [4].

Statistics. The hypergeometric function and several special cases are regularly encountered in statistical applications. Numerous probability distribution functions (PDF) and cumulative distribution functions (CDF) of discrete and continuous distributions are expressible in terms of the hypergeometric function. A few examples are:

1. Hyperbolic distribution: probability distribution function in terms of the modified Bessel function of the second kind,

$$F(x; \alpha, \beta, \delta, \mu) = \frac{\sqrt{\alpha^2 - \beta^2}}{2\alpha\delta K_1(\delta\sqrt{\alpha^2 - \beta^2})} e^{-\alpha\sqrt{\delta^2 + (x-\mu)^2} + \beta(x-\mu)}, \quad (\alpha, \beta, \delta, \mu) \in \mathbb{R}^4,$$

[3] Personal communication from Professor Liangyou Peng. `http://www.phy.pku.edu.cn/~lypeng/`.

where the modified Bessel function $K_1(z)$ is expressible in terms of the Kummer U-function as follows

$$K_1(z) = 2z\sqrt{\pi}e^{-z}U\left(\frac{3}{2}, 3, 2z\right).$$

2. Hypergeometric distribution: cumulative distribution function in terms of hypergeometric function ${}_3F_2$ given by

$$F(k; N, K, n) = 1 - \frac{\binom{n}{k+1}\binom{N-n}{K-k-1}}{\binom{N}{K}} {}_3F_2(1, A, B; C, D; 1), \tag{4.15}$$

where $(k, N, K, n) \in \mathbb{N}^4$ and

$$A = k + 1 - K, \quad B = k + 1 - n, \quad C = k + 2, \quad D = N + k + 2 - K - n.$$

3. Non-central Student's t-distribution: probability distribution function is expressed in terms of the confluent hypergeometric function ${}_1F_1$ as follows

$$f(x; \nu, \mu) = \frac{\nu^{\nu/2}\Gamma(\nu+1)e^{-\mu^2/2}}{2^{\nu}(\nu+x^2)^{\nu/2}\Gamma(\nu/2)}\left(\sqrt{2}\mu x\frac{{}_1F_1(\nu/2+1; 3/2; \gamma)}{(\nu+x^2)\Gamma((\nu+1)/2)}\right.$$
$$\left. + \frac{{}_1F_1((\nu+1)/2; 1/2; \gamma)}{\sqrt{\nu+x^2}\Gamma(\nu/2+1)}\right), \quad \nu \in \mathbb{R}^+, \ \mu \in \mathbb{R},$$

where $\gamma = \frac{(\mu x)^2}{2(\nu+x^2)}$.

Other well-known statistical distributions such as the non-central Chi distribution or the Pearson type IV distribution have PDF and CDF expressible in terms of the incomplete gamma function, incomplete beta function and Marcum functions, see [157, §36 – 39] for references and further examples. Furthermore, several characteristic functions and moment generating functions might be represented in terms of the hypergeometric function as detailed in the next section, where numerical methods for the computation of various characteristic functions are presented. Moreover, we refer to [138] for recent applications in the field of directional statistics in Machine Learning, where Von Mises-Fisher distribution and particular cases of the Kent distribution (also known as Fisher-Bingham distribution) in terms of the hypergeometric functions are considered.

Despite the fact that matrix functions are not studied in this work, we note that the hypergeometric function of a matrix argument have applications in multivariate statistical analysis, random matrix theory and wireless communications, see [100] for numerical aspects and experiments.

The hypergeometric function occasionally arise in the computation of the total probability that a given distribution is greater than another distribution, which is required for evaluating A/B tests in a Bayesian context[4], for instance. Another related problem is the computation of the accuracy ratio (AR) or Gini coefficient, which are widely used performance metrics in classification models, the latest being the standard metric to assess the performance of credit risk default models. The

[4] http://www.evanmiller.org/bayesian-ab-testing.html

remaining of this section is devoted to the computation of the Gini coefficient for two well-known statistical distributions; the Beta and the Gumbel distribution.

Let us introduce a standard definition of the Gini coefficient [151]: given two distribution X_0 and X_1 the Gini coefficient is defined as

$$Gini = AR = |2Pr[X_0 < X_1] + Pr[X_0 = X_1] - 1| = |Pr[X_0 < X_1] - Pr[X_0 > X_1]|,$$

and considering the assumption that parameters of both distributions are different, $Pr[X_0 = X_1] = 0$, we take the definition using the area under the ROC curve (AUC) given by

$$Gini = AR = |2AUC - 1|. \tag{4.16}$$

As previously stated, we will focus on the Beta distribution and the Gumbel distribution. Their respective probability density functions are given by

$$f_{Beta}(x;\alpha,\beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)}, \qquad (\alpha,\beta) \in \mathbb{R}^2_+,$$
$$f_{Gumbel}(x;\mu,\beta) = \frac{e^{-(x-\mu)/\beta - e^{-(x-\mu)/\beta}}}{\beta}, \qquad \mu \in \mathbb{R},\ \beta \in \mathbb{R}^+.$$

Having introduced the main requirements, we proceed to present our contributions in the form of the following two theorems.

Theorem 4.1.1 *Given two standard Beta distributions $X_0 \sim Beta(\alpha_0,\beta_0)$ and $X_1 \sim Beta(\alpha_1,\beta_1)$ such that $(\alpha_0,\beta_0,\alpha_1,\beta_1) \in \mathbb{R}^4_+$, the Gini or Accuracy Ratio (AR) between distributions is given by*

$$AR_{Beta} = |2AUC - 1|, \tag{4.17}$$

where AUC is the area under the ROC curve defined as

$$AUC = \frac{B(\alpha_0+\alpha_1,\beta_0)}{\alpha_1 B(\alpha_0,\beta_0)B(\alpha_1,\beta_1)}\, {}_3F_2(\alpha_1,\alpha_0+\alpha_1,1-\beta_1;1+\alpha_1,\alpha_0+\alpha_1+\beta_0;1). \tag{4.18}$$

Proof: Integrating the joint distribution, under the assumption of independence, over all values of $X_1 > X_0$ we obtain the following integral

$$\begin{aligned} Pr[X_1 > X_0] &= \int_0^1 \int_{x_0}^1 \frac{x_0^{\alpha_0-1}(1-x_0)^{\beta_0-1}}{B(\alpha_0,\beta_0)} \frac{x_1^{\alpha_1-1}(1-x_1)^{\beta_1-1}}{B(\alpha_1,\beta_1)}\, dx_1\, dx_0 \\ &= 1 - \int_0^1 \frac{x_0^{\alpha_0-1}(1-x_0)^{\beta_0-1}}{B(\alpha_0,\beta_0)} I_{x_0}(\alpha_1,\beta_1)\, dx_0, \end{aligned}$$

where $I_x(a,b)$ is the regularized incomplete Beta function defined by

$$I_x(a,b) = \frac{x^a}{B(a,b)} \sum_{k=0}^{\infty} \frac{(1-b)_k}{(a+k)} \frac{x^k}{k!}, \quad |x| < 1.$$

Let us focus on the above integral I. By formally interchanging integration and summation, which is justified by the absolute convergence of the hypergeometric

series, we obtain

$$\begin{aligned} I &= \int_0^1 \frac{x^{\alpha_0-1}(1-x)^{\beta_0-1}x^{\alpha_1}}{B(\alpha_0,\beta_0)B(\alpha_1,\beta_1)} \sum_{k=0}^{\infty} \frac{(1-\beta_1)_k}{(\alpha_1+k)} \frac{x^k}{k!}\, dx \\ &= \frac{1}{B(\alpha_0,\beta_0)B(\alpha_1,\beta_1)} \sum_{k=0}^{\infty} \frac{(1-\beta_1)_k}{(\alpha_1+k)k!} \int_0^1 x^{\alpha_0+\alpha_1+k-1}(1-x)^{\beta_0-1}\, dx \\ &= \frac{\Gamma(\beta_0)}{B(\alpha_0,\beta_0)B(\alpha_1,\beta_1)} \sum_{k=0}^{\infty} \frac{(1-\beta_1)_k}{(\alpha_1+k)k!} \frac{\Gamma(\alpha_0+\alpha_1+k)}{\Gamma(\alpha_0+\alpha_1+\beta_0+k)}. \end{aligned}$$

Finally, note that the resulting series is hypergeometric and expressible in terms of ${}_3F_2$ which yields

$$\sum_{k=0}^{\infty} \frac{(1-\beta_1)_k}{(\alpha_1+k)k!} \frac{\Gamma(\alpha_0+\alpha_1+k)}{\Gamma(\alpha_0+\alpha_1+\beta_0+k)} = \frac{{}_3F_2(\alpha_1,\alpha_0+\alpha_1,1-\beta_1;1+\alpha_1,\alpha_0+\alpha_1+\beta_0;1)}{\alpha_1(\alpha_0+\alpha_1)_{\beta_0}}.$$

Rearranging terms the proof is completed. □

Note that the hypergeometric function in (4.18) is terminating for $\beta_1 \in \mathbb{N}$, although is numerically unstable due to the amount of cancellation when adding large alternating terms. A stable convergent solution is given by applying the transformation in http://functions.wolfram.com/07.27.17.0035.01 yielding

$$\frac{B(\alpha_0+\alpha_1,\beta_0+\beta_1)}{\beta_0 B(\alpha_0,\beta_0)B(\alpha_1,\beta_1)}\, {}_3F_2(1,\alpha_0+\beta_0,\beta_0+\beta_1;\beta_0+1,\alpha_0+\alpha_1+\beta_0+\beta_1;1). \quad (4.19)$$

The hypergeometric function is reduced to

$$\sum_{k=0}^{\infty} \frac{(\alpha_0+\beta_0)_k(\beta_0+\beta_1)_k}{(\beta_0+1)_k(\alpha_0+\alpha_1+\beta_0+\beta_1)_k},$$

where all terms are positive. For some combinations of parameters, this hypergeometric series is slowly convergent, therefore series acceleration techniques come into play.

Finally, we can derive an asymptotic approximation for large parameters to estimate $Pr[X_1 > X_0]$. First, notice the resemblance of expression (4.19) with the Hypergeometric distribution in (4.15). Matching parameters we have

$$K = -\alpha_0, \quad n = -\beta_1, \quad k = \beta_0 - 1, \quad N = \alpha_1 - 1.$$

By the Law of Large Numbers we can approximate $Pr[X_1 > X_0]$ as the probability of $Pr[X \leq k]$, $X \sim \text{Hypergeometric}(K, N, n)$. Hence,

$$\mu = \frac{\alpha_0\beta_1}{\alpha_1 - 1}, \quad \sigma^2 = \left(\frac{\alpha_0\beta_1}{\alpha_1-1}\right)\left(\frac{\alpha_0+\alpha_1-1}{\alpha_1-1}\right)\left(\frac{\alpha_1\beta_1-1}{\alpha_1-2}\right)$$

and

$$Pr[X_1 > X_0] \approx \Phi\left(\frac{\beta_0 - 1 - \frac{\alpha_0\beta_1}{\alpha_1-1}}{\sqrt{\left(\frac{\alpha_0\beta_1}{\alpha_1-1}\right)\left(\frac{\alpha_0+\alpha_1-1}{\alpha_1-1}\right)\left(\frac{\alpha_1\beta_1-1}{\alpha_1-2}\right)}}\right), \quad (4.20)$$

where $\Phi(x)$ is the cumulative normal distribution function. Table 4.1 shows the relative error of the asymptotic estimate for several parameters of the Beta distribution. We observe the effectiveness of the asymptotic estimate as $\alpha_0, \beta_0, \alpha_1, \beta_1 \to \infty$, exhibiting a linear decay rate.

α_0	β_0	α_1	β_1	rel. error
50	70	100	120	4e-2
500	700	1000	1200	3e-3
5000	7000	100	160	3e-2
300	610	20000	40000	2e-2
3000	6100	20000	40000	3e-3
10000	20100	20000	40000	4e-3
100000	201000	200000	400000	9e-5

TABLE 4.1: Relative errors of the asymptotic estimate for $Pr[X_1 > X_0]$ for large parameters of two Beta distributions.

Theorem 4.1.2 *Given two Gumbel distributions defined as $X_0 \sim Gumbel(\mu_0, \beta_0)$ and $X_1 \sim Gumbel(\mu_1, \beta_1)$ such that $(\mu_0, \mu_1) \in \mathbb{R}^2$ and $(\beta_0, \beta_1) \in \mathbb{R}^2_+$, the Gini or Accuracy Ratio (AR) between distributions is given by*

$$AR_{Gumbel} = |2AUC - 1|, \tag{4.21}$$

where AUC is the area under the ROC curve defined as

$$AUC = \sum_{k=0}^{\infty} \frac{(-1)^k \Gamma(Ak+1)}{k! e^{Bk}}, \quad A = \frac{\beta_0}{\beta_1} \quad \text{and} \quad B = \frac{(\mu_0 - \mu_1)}{\beta_1},$$

where $|A| < 1$.

Proof: Following the steps presented previously, we compute the integral I given by

$$I = \frac{1}{\beta_0} \int_{-\infty}^{\infty} e^{\frac{-(x-\mu_0)}{\beta_0}} e^{-e^{\frac{-(x-\mu_0)}{\beta_0}}} e^{-e^{\frac{-(x-\mu_1)}{\beta_1}}} \, dx \, .$$

After performing the change of variable $z = (x - \mu_0)/\beta_0 \to dz\, \beta_0 = dx$ we get

$$I = \int_{-\infty}^{\infty} e^{-z} e^{-e^{-z}} e^{-e^{-(zA+B)}} \, dz,$$

where $A = \beta_0/\beta_1$ and $B = (\mu_0 - \mu_1)/\beta_1$. Unfortunately, this integral does not seem to have a closed-form for arbitrary values of A and B, therefore we easily compute a series expansion by expanding the third exponential function in the integrand as follows

$$e^{-e^{-(zA+B)}} = \sum_{k=0}^{\infty} \frac{(-1)^k (e^{-(zA+B)})^k}{k!},$$

and plugging the resulting series expansion into the integral

$$\begin{aligned} I &= \int_{-\infty}^{\infty} e^{-e^{-z}} e^{-z} \sum_{k=0}^{\infty} \frac{(-1)^k e^{-z(Ak+1)}}{k!} e^{-Bk}\, dz, \\ &= \sum_{k=0}^{\infty} \frac{(-1)^k}{k!} e^{-Bk} \int_{-\infty}^{\infty} e^{-e^{-z}} e^{-z(Ak+1)}\, dz = \sum_{k=0}^{\infty} \frac{(-1)^k \Gamma(Ak+1)}{k! e^{Bk}}. \end{aligned}$$

Note that the series expansion is convergent for $|A| < 1$. □

A simple observation of the Gumbel case shows that given the absolute value in the calculation of the Gini coefficient, we can interchange distributions X_0 and X_1 when $A \geq 1$, satisfying

$$AR_{Gumbel}\left(A = \frac{\beta_0}{\beta_1}, B = \frac{(\mu_0 - \mu_1)}{\beta_1}\right) = AR_{Gumbel}\left(A = \frac{\beta_1}{\beta_0}, B = \frac{(\mu_1 - \mu_0)}{\beta_0}\right)$$

Finally, it is possible to obtain a closed-form expression for AR_{Gumbel} for fractional values of A and B, expressing the series expansion as a combination of hypergeometric functions. For example $A = 1/3$ and $B = 2$ yields

$$AR_{Gumbel} = {}_1F_2\left(1; \frac{1}{3}, \frac{2}{3}; -\frac{1}{27e^6}\right) - \frac{2\pi}{3} \frac{1}{\sqrt[3]{3}e^2} \mathrm{Bi}\left(\frac{1}{\sqrt[3]{3}e^2}\right),$$

where $\mathrm{Bi}(z)$ is one of the standard solutions of the Airy's equation [43, §9], expressible in terms of ${}_0F_1$.

4.2 On the Computation of Confluent Hypergeometric Functions for Large Imaginary Part of Parameters b and z

4.2.1 Introduction

Many different methods have been devised for evaluating the confluent hypergeometric function of the first and second kind, ${}_1F_1(a;b;z)$ and $U(a,b,z)$ respectively. In this work we are mainly interested in numerical methods involving some of their integral representations for complex values of the parameters and argument.

As mentioned in previous sections of this Chapter, the basic approach for computing the confluent hypergeometric functions is based on the combination of power series and asymptotic series expansions. In addition, the Kummer's transformations are applied for cases where the parameters are not valid for some methods, or when the regime of parameters causes numerical instability. In particular, as noted in [90], the Kummer's transformation (4.3) is used to compute ${}_1F_1$ for $\Re(z) < 0$, so that the worst-case cancellation occurs around the oscillatory region $z = iy$, $y \in \mathbb{R}$. Besides the described disadvantages commonly occurring by using series expansions, numerical integration methods based on polynomial interpolation or Gaussian quadrature for large complex parameters also suffer severe difficulties to return accurate values. In order to compensate the inaccuracies in the presence of large oscillations, these methods require a prohibited number of quadrature points, consequently being computationally inefficient. Therefore, specific analytic and numerical methods are required to potentially diminish these difficulties.

The most relevant analytic method to handle this problem is the so-called method of stationary phase. The method of stationary phase [157, §14] is an analytic approach to handle high oscillatory integrals commonly encountered in the form of Fourier-type integrals, $F(\omega) = \int_a^b f(t)e^{i\omega g(t)}\,dt$; more details in subsequent sections. The principle of the method establishes that the main contributions to the asymptotic expansion of the integral are located in the vicinity of the critical points: stationary points in interval $[a,b]$, the finite endpoints a and b and singular points of the integrand. An application of the method to the confluent hypergeometric function ${}_1F_1$ for pure imaginary argument z was derived by Erdélyi in 1955, see [157, §14.6],

$$
\begin{aligned}
{}_1F_1(a;a+b;i\omega) \sim{}& \frac{\Gamma(a+b)}{\Gamma(a)}e^{i\omega}\sum_{k=0}^{\infty}e^{-\frac{1}{2}\pi i(k+b)}\frac{(b)_k(1-a)_k}{k!\omega^{k+b}} \\
&+\frac{\Gamma(a+b)}{\Gamma(b)}e^{i\omega}\sum_{k=0}^{\infty}e^{\frac{1}{2}\pi i(k+a)}\frac{(a)_k(1-b)_k}{k!\omega^{k+a}}, \quad \omega\to+\infty,
\end{aligned}
$$

where $\Re(a) > 0$ and $\Re(b) > 0$. We note that these type of asymptotic expansions are generally divergent when ω is not sufficiently large. Therefore, one must switch to alternative methods such as numerical integration for fixed values of ω or whenever high-precision results are needed.

In the last two decades, substantial progress has been accomplished in numerical quadratures methods for highly oscillatory integrals. Most of these methods use, as a building block, analytic tools like the method of stationary phase just mentioned or the steepest descent method hereafter described. Some of these methods

are: the Filon-type method [83], the Levin method [105], numerical steepest descent [82] and complex-valued Gaussian distribution [40]. Other recent numerical integration methods, conceptually different, are based on exploitation of the trapezoidal rule properties after performing effective variable transformations; see [127, 126].

Finally, we consider a recent proposed method applied to the evaluation of confluent hypergeometric function ${}_1F_1$ described in [132]. The numerical quadrature technique is described in [131] and given by

$$\int_{-1}^{1} f(x)e^{i\omega x}\,dx \approx \sum_{k=1}^{N} i^{k-1}(2k-1)\sqrt{\frac{\pi}{2\omega}}J_{k-\frac{1}{2}}(\omega)\sum_{j=1}^{N} w_{j-1}P_{k-1}(x_{j-1})f(x_{j-1}),$$

where $J_k(z)$ is the Bessel function of the first kind, $P_k(x)$ are the Legendre polynomials and w_k and x_k are the weights and nodes for Gauss quadrature. The method is directly applicable to the integral transformation (4.14)

$$\int_{-1}^{1} e^{z(t/2+1/2)}(1-t)^{b-a-1}t^{a-1}\,dt = e^{z/2}\int_{-1}^{1} e^{\Re(z)t/2}(1-t)^{b-a-1}t^{a-1}e^{i\Im(z)t/2}\,dt\,.$$

In this work we present an efficient algorithm for the computation of the confluent hypergeometric functions when the imaginary part of the parameter b or argument z is large. The outline of the Section is the following: we briefly review the theory behind the steepest descent method and present the integral representations for four particular cases. Then, we discuss different numerical quadrature schemes amenable for the resulting integrals. After that, we perform numerical examples and a benchmarking against available software. Then, we present several applications where this method is applicable. Finally, some remarks and prospective enhancements to the presented method are given.

4.2.2 Algorithm

The presented method for the computation of the confluent hypergeometric functions is based on the application of suitable transformations to highly oscillatory integrals and posterior numerical evaluation by means of quadrature methods. Some direct methods for ${}_1F_1(a;b;iz)$ can be applied for moderate values of $|\Im(z)|$, however a more general approach is the use of the numerical steepest descent method, which turns out to be very effective for the regime of parameters of interest. First, we briefly explain the path of steepest descent. Subsequently, we introduce the steepest descent integrals for those cases where $|\Im(b)|$, $|\Im(z)| \to \infty$.

Path of steepest descent

For this work we consider the ideal case for analytic integrand with no stationary points. We follow closely the theory developed in [82]. Let us consider the oscillatory integral

$$I = \int_{\alpha}^{\beta} f(x)e^{i\omega g(x)}\,dx \tag{4.22}$$

where $f(x)$ and $g(x)$ are smooth functions. By applying the steepest descent method, the interval of integration is substituted by a union of contours on the complex plane, such that along these contours the integrand is non-oscillatory and exponentially decaying. Given a point $x \in [\alpha, \beta]$, we define the path of steepest descent $h_x(p)$, parametrized by $p \in [0, \infty)$, such that the real part of the phase function $g(x)$ remains constant along the path. This is achieved by solving the equation $g(h_x(p)) = g(x) + ip$. If $g(x)$ is easily invertible, then $h_x(p) = g^{-1}(g(x) + ip)$, otherwise root-finding methods are employed, see [82, §5.2]. Along this path of steepest descent, integral (4.22) is transformed to

$$\begin{aligned} I[f; h_x] &= e^{i\omega g(x)} \int_0^\infty f(h_x(p)) h_x'(p) e^{-\omega p}\, dp \\ &= \frac{e^{i\omega g(x)}}{\omega} \int_0^\infty f\left(h_x\left(\frac{q}{\omega}\right)\right) h_x'\left(\frac{q}{\omega}\right) e^{-q}\, dq \end{aligned} \tag{4.23}$$

and $I = I[f; h_\alpha] - I[f; h_\beta]$ with both integrals well behaved. In the cases where $\beta = \infty$, this parametrization gives $I = I[f; h_\alpha] - 0$.

A particular case of interest is when $g(x) = x$. Then the path of steepest descent can be taken as $h_x(p) = x + ip$, and along this path (4.22) is written as

$$\int_\alpha^\beta f(x) e^{i\omega x}\, dx = \frac{ie^{i\omega\alpha}}{\omega} \int_0^\infty f\left(\alpha + i\frac{q}{\omega}\right) e^{-q}\, dq - \frac{ie^{i\omega\beta}}{\omega} \int_0^\infty f\left(\beta + i\frac{q}{\omega}\right) e^{-q}\, dq\,. \tag{4.24}$$

Case $U(a, b, z)$, $\Im(z) \to \infty$

Integral representation (4.13) can be transformed into a highly oscillatory integral

$$U(a, b, z) = \frac{1}{\Gamma(a)} \int_0^\infty e^{-\Re(z)t} t^{a-1} (1+t)^{b-a-1} e^{-i\Im(z)t}\, dt\,. \tag{4.25}$$

Although the integral transformation is not in standard form of the oscillatory integral, one can amend it after application of the mirror symmetry property $U(\overline{a}, \overline{b}, \overline{z}) = \overline{U(a, b, z)}$ valid when $z \notin (-\infty, 0)$. Taking $g(t) = t$, $g'(t) = 1 \neq 0$ and there are no stationary points. Therefore, in this case we only have one endpoint and the steepest descent integral obtained by (4.24) is reduced to a single line integral,

$$U(a, b, z) = \frac{i}{\omega\Gamma(a)} \int_0^\infty e^{-\Re(z) i\frac{q}{\omega}} \left(i\frac{q}{\omega}\right)^{a-1} \left(1 + i\frac{q}{\omega}\right)^{b-a-1} e^{-q}\, dq, \tag{4.26}$$

where $\omega = -\Im(z)$.

Case $U(a, b, z)$, $\Im(b) \to \infty$

In this case, the path of integration is modified to avoid the singularity at $t = 0$, as can be observed after performing the transformation to a highly oscillatory integral,

$$\begin{aligned} U(a, b, z) &= \frac{1}{\Gamma(a)} \int_0^\infty e^{-zt} t^{a-1} (1+t)^{b-a-1}\, dt \\ &= \frac{e^z}{\Gamma(a)} \int_1^\infty e^{-zt} (t-1)^{a-1} t^{\Re(b)-a-1} e^{i\Im(b)\log(t)}\, dt\,. \end{aligned} \tag{4.27}$$

Now, we solve the path of steepest descent at $t = 1$ with $g(t) = \log(t)$, which in this case results trivial, $h_1(p) = e^{\log(1)+ip} = e^{ip}$ and $h_1'(p) = ie^{ip}$. Likewise, no stationary points besides $t = \infty$ are present, and therefore there are no further contributions. The steepest descent integral is given by

$$U(a,b,z) = \frac{ie^z}{\omega\Gamma(a)} \int_0^\infty e^{\phi(q,\omega)} (e^{\mu(q,\omega)} - 1)^{a-1} \left(e^{\mu(q,\omega)}\right)^{\Re(b)-a-1} e^{-q}\, dq, \tag{4.28}$$

where $\omega = \Im(b)$, $\mu(q,\omega) = i\frac{q}{\omega}$ and $\phi(q,\omega) = -ze^{\mu(q,\omega)} + \mu(q,\omega)$.

Case ${}_1F_1(a,b,z)$, $\Im(z) \to \infty$

Similarly, we transform integral (4.12) into a highly oscillatory integral

$${}_1F_1(a;b;z) = \frac{\Gamma(b)}{\Gamma(a)\Gamma(b-a)} \int_0^1 e^{\Re(z)t} t^{a-1} (1-t)^{b-a-1} e^{i\Im(z)t}\, dt, \tag{4.29}$$

valid for $\Re(b) > \Re(a) > 0$. Again with $g(t) = t$ and the transformation stated in (4.24), we obtain, after some calculations, the steepest descent integrals given by

$$\begin{aligned} {}_1F_1(a;b;z) \quad &= \tfrac{\Gamma(b)}{\Gamma(a)\Gamma(b-a)} \tfrac{i}{\omega} \Bigg[\int_0^\infty e^{\Re(z) i\frac{q}{\omega}} \left(i\tfrac{q}{\omega}\right)^{a-1} \left(1 - i\tfrac{q}{\omega}\right)^{b-a-1} e^{-q}\, dq \\ &\quad - e^{i\omega} \int_0^\infty e^{\Re(z)(1+i\frac{q}{\omega})} \left(1 + i\tfrac{q}{\omega}\right)^{a-1} \left(-i\tfrac{q}{\omega}\right)^{b-a-1} e^{-q}\, dq \Bigg], \end{aligned} \tag{4.30}$$

where $\omega = \Im(z)$.

Case ${}_1F_1(a,b,z)$, $\Im(b) \to \infty$

For this case we can use the connection formula (4.5), valid for all $z \neq 0$. Alternatively, integral representation (4.12) can be rewritten as a Laplace-type integral for large b as follows

$$\begin{aligned} {}_1F_1(a;b;z) &= \frac{\Gamma(b)}{\Gamma(a)\Gamma(b-a)} \int_0^1 e^{zt} t^{a-1} (1-t)^{b-a-1}\, dt \\ &= \frac{\Gamma(b)}{\Gamma(a)\Gamma(b-a)} \int_0^\infty e^{-bt} \left(1 - e^{-t}\right)^{a-1} e^{at + z(1-e^{-t})}\, dt \\ &= \frac{\Gamma(b)}{\Gamma(a)\Gamma(b-a)} \int_0^\infty e^{\Re(b)t} (1-e^{-t})^{a-1} e^{at + z(1-e^{-t})} e^{-i\Im(b)t}\, dt\,. \end{aligned} \tag{4.31}$$

The resulting transformation is in turn a suitable integral representation to develop an asymptotic expansion for b, as described in [157, §10.4.1]. Note that this case is similar to $U(a,b,z)$ when $\Im(z) \to \infty$, hence we refer to (4.26).

4.2.3 Numerical quadrature schemes

Adaptive quadrature for oscillatory integrals

The integrand in (4.30) can be rewritten in terms of its real and imaginary parts to obtain two separate integrals with trigonometric weight functions, the oscillatory

factor, given the property,

$$\int_0^1 f(t)e^{i\omega t}\,dt = \int_0^1 f(t)\cos(\omega t)\,dt + i\int_0^1 \sin(\omega t)\,dt\,. \tag{4.32}$$

Thus, we obtain the following integral representation for ${}_1F_1(a;b;z)$ for large imaginary z,

$${}_1F_1(a;b;z) = \frac{\Gamma(b)}{\Gamma(a)\Gamma(b-a)} \quad \left[\int_0^1 e^{\Re(z)t}t^{a-1}(1-t)^{b-a-1}\cos(\Im(z)t)\,dt \right. \\ \left. + i\int_0^1 e^{\Re(z)t}t^{a-1}(1-t)^{b-a-1}\sin(\Im(z)t)\,dt\right] \tag{4.33}$$

These type of integrals can be solved using specialized adaptive numerical integration routines, such as the routine `gsl_integration_qawo` from the GNU Scientific Library [56]. This routine combines Clenshaw-Curtis quadrature with Gauss-Kronrod integration. Numerical examples can be found in [117], which show that this method works reasonably well for moderate values of $|\Im(z)|$. Unfortunately, this method cannot be directly applied to $U(a,b,z)$, and Kummer's transformation [43, §13.2.42], valid for $b \notin \mathbb{Z}$, is needed. Finally, Figure 4.1 illustrates the behaviour of an integrand evaluated with this method and the numerical steepest descent. Observe that the reformulation of the integral by the steepest descent method removes the oscillatory nature on the real and imaginary part, a fact that facilitates the use of numerical integration techniques.

Gauss-Laguerre quadrature

An efficient approach for infinite integrals with an exponentially decaying integrand is classical Gauss-Laguerre quadrature. Laguerre polynomials are orthogonal with respect to e^{-x} on $[0,\infty)$. Hence, using n-point quadrature yields an approximation,

$$I[f;h_x] \approx Q[f;h_x] := \frac{e^{i\omega g(x)}}{\omega}\sum_{k=1}^{n} w_k f\left(h_x\left(\frac{x_k}{\omega}\right)\right)h_x'\left(\frac{x_k}{\omega}\right) \tag{4.34}$$

As stated in [82], the approximation error by the quadrature rule behaves asymptotically as $\mathcal{O}(\omega^{-2n-1})$ as $\omega \to \infty$. As an illustrative example, let us consider the asymptotic expansion for $U(a,b,z)$ when $|z| \to \infty$ given in (4.10). The error behaves asymptotically as $\mathcal{O}(z^{-n-1})$, as notice by truncating the asymptotic expansion after n terms. Therefore, the asymptotic order of the Gauss-Laguerre quadrature is practically double using the same number of terms. A well-known formula for the error of the n-point quadrature approximation (4.34) is

$$E = \frac{(n!)^2}{(2n)!}f^{2n}(\zeta), \quad 0 < \zeta < \infty \tag{4.35}$$

According to this formula and under the general assumption that $a, b \in \mathbb{R}\setminus\mathbb{N}$, f is infinitely differentiable on $[0,\infty)$, we can use the general Leibniz rule for the higher

derivatives of a product of m factors to obtain the derivative of order $2n$,

$$((f\cdot g)\cdot h)^{(2n)} = \sum_{j=0}^{2n}\sum_{k=0}^{2n-j} \frac{(2n)!\cdot f^{(j)}g^{(k)}h^{(2n-k-j)}}{j!k!(2n-k-j)!}, \tag{4.36}$$

where

$$f(x) = e^{-\Re(z)ix/\omega}, \quad g(x) = \left(1+i\frac{x}{\omega}\right)^{b-a-1}, \quad h(x) = \left(i\frac{x}{\omega}\right)^{a-1} \tag{4.37}$$

and the $2n$ derivatives are given by

$$\sum_{j=0}^{2n}\sum_{k=0}^{2n-j} \frac{(2n)!(-1)^j}{j!k!(2n-k-j)!}\left(\frac{\Re(z)i}{\omega}\right)^j e^{-\Re(z)ix/\omega}\frac{\left(\frac{i}{\omega}\right)^k}{(b-a-1)_{-k}}\left(1+i\frac{x}{\omega}\right)^{b-a-1-k} \times \frac{\left(\frac{i}{\omega}\right)^{2n-k-j}}{(a-1)_{-2n+k+j}} x^{a-1-2n+k+j}, \tag{4.38}$$

where $(a)_n$ is the Pochhammer symbol or rising factorial. An error bound in terms of a, b and z might be obtained from (4.38). Ideally, the error bound shall be tight enough without increasing the total computation time excessively. However, as can be seen below, numerical experiments indicate that the number of terms n rarely exceeds 50 for moderate values of the remaining parameters, typically if $|a|, |b|\cdot 10 < |\omega|$, for the case $U(a,b,iz)$ or ${}_1F_1(a,b,iz)$. Finally, for large parameters we apply logarithmic properties to the integrand in order to avoid overflow and underflow errors.

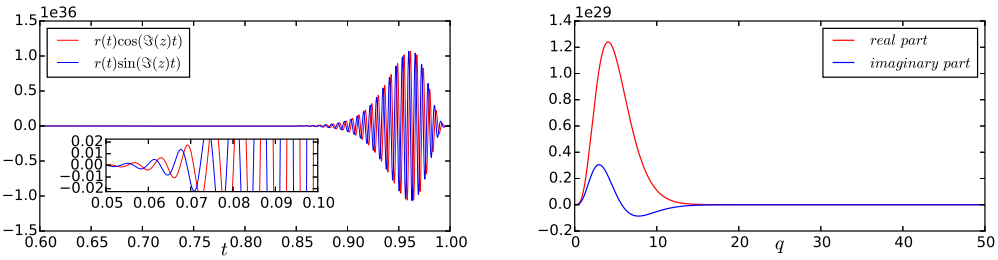

FIGURE 4.1: Real and imaginary part of integrand for ${}_1F_1(5,10,100-1000i)$ before and after applying steepest descent method.

4.2.4 Numerical examples

In this section, we compare our algorithm (NSD) with other routines in double precision floating-point arithmetic in terms of accuracy and computation time[5]. Note that just a few packages in double precision allow the evaluation of the confluent hypergeometric function with complex argument. For this study we use Algorithm 707: CONHYP, described in [115, 116] and Zhang and Jin implementation (ZJ) in [171]. Both codes are written in Fortran 90 and were compiled using `gfortran`

[5]Intel(R) Core(TM) i5-3317U CPU at 1.70GHz.

`4.9.3` without optimization flags. We implemented a simple prototype of the described methods using `Python 3.5.1` and the package `SciPy` [95], therefore there is plenty of room for improvement, and is part of ongoing work. Nevertheless, as shown in Table 4.2, our algorithm clearly outperforms aforementioned codes, being more noticeable as z increases. In order to test the accuracy, we use `mpmath` [92] with 20 digits of precision to compute the relative errors.

${}_1F_1(a,b,z)$	CONHYP	ZJ	NSD	N
$(1,4,50i)$	3.96e−13/4.29e−18i	1.50e−15/4.28e−18i	1.15e−16/1.11e−16i	2
$(3,10,30+100i)$	1.27e−13/1.28e−13i	6.83e−17/1.07e−14i	2.48e−17/1.24e−14i	25
$(15,20,200i)$	9.20e−13/9.20e−13i	E	8.43e−16/7.93e−16i	25
$(400,450,1000i)$	8.32e−12/1.00e−11i	–	1.37e−12/1.02e−13i	50
$(2,20,50-2500i)$	1.35e−11/1.35e−11i	7.30e−11/2.10e−09i	4.75e−16/6.41e−16i	20
$(500,510,100-1000i)$	4.10e−13/3.68e−12i	–	4.71e−13/3.11e−16i	50
$(2,20,-20000i)$	–	5.79e−10/7.99e−07i	5.92e−16/3.62e−14i	10
$(900,930,-10^{10}i)$	–	–	6.78e−13/6.77e−13i	20
$(4000,4200,50000i)^*$	–	–	6.04e−12/5.99e−12i	80

TABLE 4.2: Relative errors for routines computing the confluent hypergeometric function for complex argument. N: number of Gauss-Laguerre quadratures. $(*)$: precision in `mpmath` increased to 30 digits. (E): convergence to incorrect value. $(-)$: overflow.

Table 4.3 and Figure 4.2 summarize the testing results and general performance of the algorithm for $U(a,b,z)$. As can be observed, 13-14 digits of precision in real and imaginary part are typically achieved. A similar precision for ${}_1F_1(a;ib;z)$ is expected. In terms of computational time, we compare our implementation in Python with MATLAB R2013a. As shown in Table 4.4, the MATLAB routine `hypergeom` is significantly slow for large imaginary parameters.

Function	Min	Max	Mean
$U(a,b,iz)$	1.97e−18/2.04e−17i	9.97e−13/2.50e−11i	1.34e−14/6.94e−14i
$U(a,ib,z)$	6.57e−18/6.17e−18i	1.49e−11/8.55e−12i	1.38e−13/1.43e−13i

TABLE 4.3: Error statistics for $U(a,b,iz)$ and $U(a,ib,z)$ using $N=100$ quadratures.

${}_1F_1(a;b;z)$	MATLAB	NSD
$(2,20,-20000i)$	1.509 (0.068)	0.033
$(900,930,-10^{10}i)$	5.594 (0.739)	0.035
$(4000,4200,50000i)$	488.384(18.127)	0.043

TABLE 4.4: Comparison in terms of cpu time. MATLAB second evaluation in parenthesis.

4.2.5 Applications

Besides the necessity of accurate and reliable methods for the regime of parameters and argument considered, confluent hypergeometric functions can be encountered

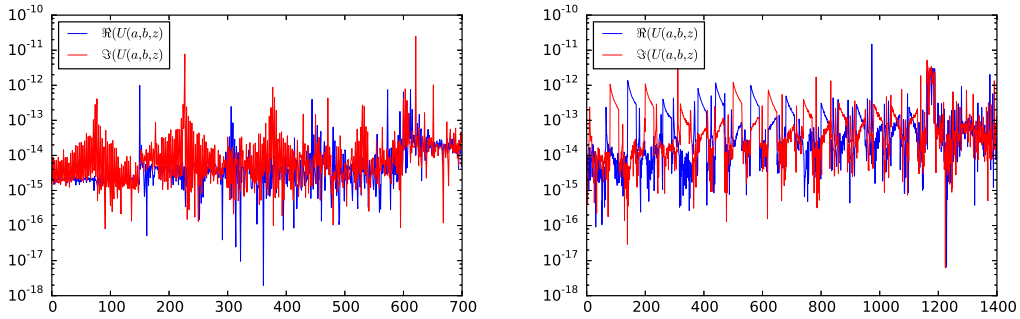

FIGURE 4.2: Relative error in computing $U(a,b,z)$. Error in $U(a,b,iz)$ for $a \in [2,400], b \in [-500,500], z \in [10^3,10^6]$ (left) and $U(a,ib,z)$ for $a \in [10,100], b \in [10^3,10^4], z \in [10,100]$ (right). 700 and 1400 tests, respectively.

in several scientific applications. In this work, we focus on applications in statistics, more precisely on the evaluation of characteristic functions, which can be defined in terms of confluent hypergeometric functions. Let us consider three statistical distributions:

- Characteristic function of the Beta distribution.

$$\phi_X(t) = {}_1F_1(\alpha;\alpha+\beta;it) \tag{4.39}$$

  where $\alpha,\beta > 0$. Thereby, the regime of parameters holds for the integral representation in (4.30).

- The standard Arcsine distribution is a special case of the Beta distribution with $\alpha = \beta = 1/2$, therefore we obtain a similar characteristic function, which can be identically computed.

$$\phi_X(t) = {}_1F_1\left(\frac{1}{2};1;it\right) \tag{4.40}$$

- The characteristic function for the F–distribution is defined in terms of the confluent hypergeometric function of the second kind,

$$\phi_X(t) = \frac{\Gamma((p+q)/2)}{\Gamma(q/2)} U\left(\frac{p}{2},1-\frac{q}{2},-\frac{q}{p}it\right) \tag{4.41}$$

  where $p,q > 0$, are the degrees of freedom. In this case we can use the integral representation in (4.26).

Several numerical libraries include implementations of aforementioned characteristic functions, for example the R package `prob` and MATLAB code `CharFunTool`[6] (`https://github.com/witkovsky/CharFunTool`). Package `prob` relies on the R package `fAsianOptions` which computes the confluent hypergeometric functions using algorithm CONHYP. On the other hand, CharFunTool only implements the power

[6] I was contacted by the author and future collaboration is planned.

series expansion. Therefore, these and other libraries could benefit by adding the described algorithm.

Characteristic functions appear in many financial econometric models, for example modelling a beta-distributed loss given default in portfolio credit risk models (see [117, §4.4.2]). In particular, the computation of the portfolio Fourier transform with beta-distributed loss given default at time t is given by

$$\hat{f}(t) = \int_{-\infty}^{\infty} \prod_{n=1}^{N} \left(1 - p_n(y) + p_n(y)\, {}_1F_1(\alpha; \alpha+\beta; -itE_n)\right) f(y)\, dy, \tag{4.42}$$

where N is total number of assets in the portfolio, $p_n(y)$ is the default probability for each asset, E_n is the exposure at default for each asset and $f(y)$ represents the probability distribution of the credit risk factor. We stress that only a few digits of precision are generally required in financial models, says less than $\sqrt{\epsilon}$, which is achievable by the presented numerical method.

4.2.6 Conclusions

We have presented an efficient algorithm for computing the confluent hypergeometric functions with large imaginary parameter and argument, which emerges as an alternative to asymptotic expansions. The numerical experiments show promising results and fast convergence as the imaginary part increases. Throughout this work we have been considering real values for the remaining parameters, otherwise the function f becomes oscillatory. The numerical steepest descent method is not insensitive to oscillations in f, although in some cases this can be treated by applying other transformations. In cases where that is not possible, other methods have to be considered. Finally, a suitable integral representation for $|\Im(a)| \to \infty$ carry more complications and is part of future work.

4.3 High-precision Computation of the Confluent Hypergeometric Functions via Franklin-Friedman Expansion

4.3.1 Introduction

The current standard method of evaluation of many special functions to high-precision consists of employing the ascending series for small argument z, i.e. the direct series expansion at $z = 0$, combined with the usage of an asymptotic expansion for large values of the argument z. It is well known that the behaviour of asymptotic series of Poincaré type for large $|z|$ is characterized by having initial terms that decrease in magnitude until a minimum is attained, also known as optimal truncation, and thereafter subsequent terms start to increase. This limitation on the achievable accuracy forces a switch from the asymptotic series to the ascending series depending on the desired level of precision. However, the evaluation of ascending series for large z requires to increase the working precision in order to compensate the large amount of cancellation, which in turn increases the computational cost.

Normally, the scheme of computation implemented in high-precision software is a relatively simplistic choice between the ascending series and the asymptotic series based on the magnitude of the argument z and the desired level of precision or other heuristics that generally exclude other parameters involved. This type of scheme, although asymptotically valid, may lead to incorrect results in the vicinity of the transition region.

Alternative computational methods have been devised to complement the described dichotomy between series expansion and asymptotic expansion. These methods, such as *exponentially-improved* expansions [124] or their extension called *hyperasymptotic* expansions [123], are focused on extending the region of validity of the asymptotic series by iteratively re-expanding the remainder terms at optimal truncation into another asymptotic series, each exponentially smaller than its predecessor. This procedure increases the attainable accuracy of the asymptotic expansion at the expense of the computational cost of evaluating substantial complicated terms at each level of the hyperasymptotic expansion.

A remarkable method to obtain geometrically convergent series for the evaluation of special functions consists of using variants of Hadamard series, which have been extensively investigated by R. B. Paris, for example in [129]. These series involving the normalized incomplete gamma function exhibit a rapid decay after the optimal truncation term, being comparable with the behaviour of the first terms of the asymptotic expansion. A similar geometrically convergent series for the evaluation of Bessel functions was developed by D. Borwein, J. Borwein and O. Chan in [12] through the evaluation of the so-called “exp-arc" integrals.

On the other hand, it is essential to mention the role played by uniform asymptotic expansions to obtain powerful expansions valid for extended regimes of the parameters. We shall remark the so-called *vanishing saddle point* method developed by N. M. Temme in [155, 156]. This method is applicable to Laplace-type integrals of the form

$$F_\lambda(z) = \frac{1}{\Gamma(\lambda)} \int_0^\infty t^{\lambda-1} e^{-zt} f(t)\, dt, \tag{4.43}$$

with $\Re(\lambda) > 0$ and z large, in which λ may also be large. Essentially, this method expands the amplitude function $f(t)$ at $t = \mu$, $\mu = \lambda/z \geq 0$ being a uniformity parameter corresponding to the saddle point of the dominant part of the integral (4.43).

In this work, we revisit the theory originally developed by J. Franklin and B. Friedman in [55], which henceforth we shall call *Franklin-Friedman* expansions. Their method was developed with the aim to overcome the disadvantages of the direct application of Watson's lemma to Laplace-type integrals [157, §2]. Historically, this method has not received significant attention, presumably due to the inherent difficulty of evaluating the coefficients of the expansion. We shall show, through the study of an important amplitude function occurring in many integral representations of special functions, how the coefficients of the Franklin-Friedman expansion can be efficiently evaluated, resulting in a convergent method capable of out-performing aforementioned methods.

The rest of the Section is outlined as follows. First, we briefly revisit the theory corresponding to the Franklin-Friedman expansion and we show an illustrative example. Then we compute the coefficients for the amplitude function corresponding to the confluent hypergeometric function and we provide an analysis of the obtained coefficients. Subsequently, we present an effective recursive algorithm for the computation of the coefficients and we provide numerical calculations and compare the present method with the conventional ascending-asymptotic series and previously discussed convergent and uniform asymptotic expansions. Finally, we discuss possible enhancements and present our conclusions.

4.3.2 The Franklin-Friedman expansion

J. Franklin and B. Friedman developed in [55] a method for obtaining convergent asymptotic representations for Laplace-type integrals of the form

$$F_\lambda(z) = \frac{1}{\Gamma(\lambda)} \int_0^\infty t^{\lambda-1} e^{-zt} f(t)\, dt, \quad \Re(z) > 0,\ \Re(\lambda) > 0, \tag{4.44}$$

for large values of z with suitable assumptions on the amplitude function $f(t)$. This method is based on the application of a type of interpolation process to the function $f(t)$, differing from Watson's lemma, in which the amplitude function is expanded in a power series at $t = 0$ and integrated term by term. The first interpolation point $t_0 = \lambda/z$ corresponds to the saddle point of the dominant part $t^\lambda e^{-zt}$, and by substituting in (4.44) we obtain

$$F_\lambda(z) = f(t_0) z^{-\lambda} + \frac{1}{\Gamma(\lambda)} \int_0^\infty t^{\lambda-1} e^{-zt} (f(t) - f(t_0))\, dt\,. \tag{4.45}$$

After integrating by parts we obtain

$$F_\lambda(z) = f(t_0) z^{-\lambda} + \frac{1}{z\Gamma(\lambda)} \int_0^\infty t^{\lambda} e^{-zt} f_1(t)\, dt, \tag{4.46}$$

where the new amplitude function $f_1(t)$ is defined by

$$f_1(t) = \frac{d}{dt}\frac{f(t) - f(t_0)}{t - t_0}. \tag{4.47}$$

One can observe that integral (4.46) has the same form as the integral (4.44), with λ replaced by $\lambda + 1$ and f by f_1. The interpolation point for the next iteration is $t_1 = (\lambda + 1)/z$. This process can be continued iteratively obtaining the following series expansion

$$F_\lambda(z) = \sum_{k=0}^{n-1} f_k(t_k)\frac{(\lambda)_k}{z^{\lambda+2k}} + \frac{1}{z^n\Gamma(\lambda)}\int_0^\infty t^{\lambda+n-1}e^{-zt}f_n(t)\,dt, \tag{4.48}$$

where $n = 0, 1, 2, \ldots$, $f_0(t) = f(t)$ and

$$f_{k+1}(t) = \frac{d}{dt}\frac{f_k(t) - f_k(t_k)}{t - t_k}, \quad t_k = \frac{\lambda + k}{z}, \quad k = 0, 1, 2, \ldots. \tag{4.49}$$

Sufficient conditions on the amplitude function $f(t)$ for the convergent behaviour of the series expansion are stated in the following two theorems. We refer to [55] for proofs.

Theorem 4.3.1 (J. Franklin and B. Friedman [55]) *Take $\lambda = \alpha + i\beta$, where α is real and positive and β is real. For $\beta \neq 0$, suppose that $f(t)$ is analytic for $\Re(t) > 0$ and that $f \in C^{2n}[0, \infty)$, such that the derivatives satisfy*

$$|f^{(m)}(t)| \leq Me^{\mu t}, \quad m = 0, 1, \ldots, 2n,$$

where M and μ are non-negative constants. Under these conditions, expansion (4.48) has an asymptotic behaviour as $z \to \infty$, with remainder term of order $O(z^{-2n-\lambda})$. If $\lambda = \alpha$, the assumption that $f(t)$ is analytic for $\Re(t) > 0$ may be replaced by the assumption that $f \in C^{2n}[0, \infty)$ for $t > 0$. Therefore, the series expansion is convergent.

Theorem 4.3.2 (J. Franklin and B. Friedman [55]) *Suppose $f(q) = f(x + iy)$ can be represented in the form*

$$f(q) = \int_0^\infty e^{-qt}\,d\Psi(t), \quad x > 0,$$

where $\Psi(t)$ is a complex-value function which is of bounded variation in each finite interval $t \in [0, T]$ and which satisfies the inequality

$$|\Psi(t)| \leq M, \quad t \geq 0.$$

Then for $z > 0$ and $\Re(\lambda) > 0$, series expansion (4.48) converges to (4.44).

In [157, §17.4], Temme gives the first five coefficients $f_k(t_k)$ for the incomplete gamma function $e^z\Gamma(1 - \lambda, z) = \frac{1}{\Gamma(\lambda)}\int_0^\infty t^{\lambda-1}e^{-zt}(1 + t)^{-1}\,dt$, amplitude function

$f(t) = (1+t)^{-1}$, by using computer algebra

$$
\begin{aligned}
f_0 &= \frac{z}{\zeta}, \quad f_1 = \frac{z^3}{\zeta(\zeta+1)^2}, \quad f_2 = \frac{z^5(3\zeta+4)}{\zeta(\zeta+1)^2(\zeta+2)^3}, \\
f_3 &= \frac{z^7(15\zeta^3+90\zeta^2+175\zeta+108)}{\zeta(\zeta+1)^2(\zeta+2)^3(\zeta+3)^4}, \\
f_4 &= \frac{z^9(105\zeta^6+1680\zeta^5+11025\zeta^4+37870\zeta^3+71540\zeta^2+70120\zeta+27648)}{\zeta(\zeta+1)^2(\zeta+2)^3(\zeta+3)^4(\zeta+4)^5},
\end{aligned}
\tag{4.50}
$$

where $\zeta = z + \lambda$. Despite the relative ease of computing the coefficients by means of computer algebra systems, it is usually difficult to obtain explicit representations such that these become usable for numerical evaluation purposes. In practice, one generates a few coefficients of the expansions for a bounded domain of the parameters and incorporate them into a routine, this procedure being solely valid for fixed precision.

4.3.3 The expansion for $U(a,b,z)$

The *confluent hypergeometric function of the first kind* ${}_1F_1(a;b;z)$ and the *Kummer function* $U(a,b,z)$ arise as linearly independent solutions of the *Kummer's differential equation* [43, §13.2]

$$
z\frac{d^2w}{dz^2} + (b-z)\frac{dw}{dz} - aw = 0, \tag{4.51}
$$

for $b \notin \mathbb{Z}^- \cup \{0\}$. Confluent hypergeometric functions appear in a wide range of applications in mathematical physics and applied mathematics. Many special functions are expressible in terms of specific forms of the confluent hypergeometric functions such as, for example, Bessel functions, incomplete Gamma functions and Laguerre polynomials amongst others.

A convenient starting point for $U(a,b,z)$ is the integral representation [157, §10.1.5]

$$
U(a,b,z) = \frac{1}{\Gamma(a)}\int_0^\infty t^{a-1}e^{-zt}(1+t)^{b-a-1}\,dt, \tag{4.52}
$$

valid for $\Re(a) > 0$ and $\Re(z) > 0$. Laplace-type integral (4.13) includes the amplitude function $f(t) = (1+t)^{b-a-1}$, which we shall investigate further on. An asymptotic expansion valid for $|z| \to \infty$ can be derived by application of Watson's lemma to the integral representation (4.13). We obtain [43, §13.7.3]

$$
U(a,b,z) \sim z^{-a}\sum_{k=0}^{\infty}(-1)^k\frac{(a)_k(a-b+1)_k}{k!z^k}, \quad |\mathrm{ph}\, z| \leq \frac{3}{2}\pi - \delta, \tag{4.53}
$$

where δ is an arbitrary small positive constant such that $0 < \delta \ll 1$ and $(a)_k = a(a+1)\cdots(a+k-1)$ denotes a rising factorial or Pochhammer symbol. For $\Re(z) > 0$, asymptotic series (4.53) is alternating and thus the remainder is bounded by the absolute value of the first neglected term. As previously discussed, the remainder cannot be reduced arbitrarily, hence when z is not sufficiently large with respect to

a and b (not made rigorous here) and the required precision bits is moderate, this expansion cannot be used effectively.

Evaluation of $U(a,b,z)$ outside the sector $|\mathrm{ph}\, z| < \frac{1}{2}\pi$ can be achieved by use of the continuation formula [157, §10.1.11]

$$\begin{aligned} e^{-z}U(a,b,z) = &\frac{e^{\mp\pi i a}\Gamma(b-a)}{\Gamma(b)}\,{}_1F_1(b-a;b;-z) \\ &- \frac{e^{\mp\pi i b}\Gamma(b-a)}{\Gamma(a)}U(b-a,b,ze^{\mp\pi i}), \end{aligned} \tag{4.54}$$

where ${}_1F_1(a;b;z)$ is an entire function with series expansion given by [157, §10.1.2]

$${}_1F_1(a;b;z) = \sum_{k=0}^{\infty} \frac{(a)_k}{(b)_k k!} z^k. \tag{4.55}$$

To compute the Kummer function $U(a,b,z)$ for small values of z, the usual approach is to employ connection formulas for this function in terms of ${}_1F_1(a;b;z)$, for example [157, §10.1.12]

$$U(a,b,z) = \frac{\Gamma(1-b)}{\Gamma(a-b+1)}\,{}_1F_1(a;b;z) + \frac{\Gamma(b-1)}{\Gamma(a)} z^{1-b}\,{}_1F_1(a-b+1;2-b;z), \tag{4.56}$$

which is not defined for integer values of b, although the limit exists for $b \to 0$. Additionally, a recent method for computing the Kummer function $U(a,b,z)$ for small values of $|a|$, $|b|$ and $|z|$ is described in [63].

4.3.4 The Franklin-Friedman expansion coefficients

We compute the coefficients of the Franklin-Friedman expansion for the amplitude function $f(t) = (1+t)^{b-a-1}$ appearing in the Laplace-type integral for $U(a,b,z)$ in (4.13). We start computing a few coefficients f_k in expansion (4.48) using Mathematica 10 [169] and employing the `Expand` and `FullSimplify` options to perform algebraic simplifications and transformations. We only show the first two coefficients since their size grows considerably with k

$$f_0 = (1+a/z)^{b-a-1} \quad \text{and} \quad f_1 = z^3\left(\frac{(\frac{a+z}{z})^{b-a}}{a+z} - \frac{(\frac{1+a+z}{z})^{b-a}(2+2a-b+z)}{(1+a+z)^2}\right).$$

Note that above coefficients coincide with those in (4.50) when $\lambda = a = b$. Unfortunately, Mathematica was unable to produce more simplified expressions of f_k. Hereinafter, we proceed to generate tractable explicit representations of the coefficients f_k. Let us first define the coefficients $A_s^q := (1+(a+s)/z)^q$. Subsequently, we factorize the previously obtained coefficients and rearrange terms such that coefficients A_j^{b-a-i} appear in ascending order (i,j). The first four coefficients f_k are

now given by

$$
\begin{aligned}
f_0 &= A_0^{b-a-1},\\
f_1 &= (A_1^{b-a-2}(b-a-1) + (A_0^{b-a-1} - A_1^{b-a-1})z)z,\\
f_2 &= \Big(A_2^{b-a-3}(b-a-1)(b-a-2) + 2(A_1^{b-a-2} - A_2^{b-a-2})(b-a-1)z\\
&\quad + (A_0^{b-a-1} - 2A_1^{b-a-1} + A_2^{b-a-1})z^2\Big)\frac{z^2}{2},\\
f_3 &= \Big(A_3^{b-a-4}(b-a-1)(b-a-2)(b-a-3)\\
&\quad + 3(b-a-1)(b-a-2)(A_2^{b-a-3} - A_3^{b-a-3})z\\
&\quad + 3(b-a-1)(A_1^{b-a-2} - 2A_2^{b-a-2} + A_3^{b-a-2})z^2\\
&\quad + (A_0^{b-a-1} - 3A_1^{b-a-1} + 3A_2^{b-a-1} - A_3^{b-a-1})z^3\Big)\frac{z^3}{6}.
\end{aligned}
$$

We observe that the terms multiplying A_j^{b-a-i} in f_k correspond to rows of Pascal's triangle, with alternating sign for the inner terms in $(\cdots)z^{k-j}$. Furthermore, a multiplicative factor $z^k/k!$ is present. After performing a few more algebraic manipulations we obtain the explicit representation of f_k given by

$$
f_k = c_k(z)\frac{z^k}{k!}, \tag{4.57}
$$

where

$$
c_k(z) = \sum_{j=0}^{k}\binom{k}{j}z^{k-j}d_j\sum_{s=j}^{k}(-1)^{s-j}\binom{k-j}{k-s}A_s^{b-a-1-j}, \quad k = 0,1,2,\ldots \tag{4.58}
$$

and d_j is defined by

$$
d_j = \frac{\Gamma(b-a)}{\Gamma(b-a-j)} = \begin{cases} 1, & j = 0\\ d_{j-1}(b-a-j), & j > 0 \end{cases} \tag{4.59}
$$

We remark that equations (4.58)-(4.59) add the internal coefficients backwards. In order to calculate them forward, the coefficients $c_k(z)$ are written equivalently as

$$
c_k(z) = \sum_{j=0}^{k}\binom{k}{j}z^j l_j\sum_{s=0}^{j}(-1)^s\binom{j}{s}A_{k-j+s}^{b-a-1-k+j}, \tag{4.60}
$$

where

$$
l_j = \frac{\Gamma(b-a)}{\Gamma(b-a+j-k)} = \begin{cases} \prod_{i=1}^{k}(b-a-i), & j = 0\\ \frac{l_{j-1}}{(b-a-k+j)}, & j > 0 \end{cases} \tag{4.61}
$$

Note that the previous double finite summation is over the triangle $0 \leq s \leq j \leq k$. Furthermore, $c_k(z)$ can be written as

$$c_k(z) = z^{k-q} k! \sum_{j=0}^{k} \binom{q}{k-j} \frac{1}{j!} \sum_{s=0}^{j} (-1)^s \binom{j}{s} \frac{1}{(p+k-j+s)^{k-q-j}}, \tag{4.62}$$

where $q = b - a - 1$ and $p = z + a$. We shall see that the explicit representation in (4.62) will lead to our main result (Theorem 4.3.5), where we prove that the following expression for $U(a, b, z)$ holds

$$U(a, b, z) = \sum_{k=0}^{\infty} c_k(z) \frac{(a)_k}{k! z^{a+k}}. \tag{4.63}$$

Analysis of the coefficients $c_k(z)$

In this subsection we examine the coefficients $c_k(z)$ defined in (4.62). Let us define the coefficients $r_j := \sum_{s=0}^{j} (-1)^s \binom{j}{s} (p+k-j+s)^{q+j-k}$ corresponding to the inner summation in (4.62). These coefficients are defined by the binomial transform of the sequence $\{(p+k-j+s)^{q+j-k}\}_{s \geq 0}$, which can be represented by means of the Nörlund-Rice integrals [52].

In what follows, we proceed to derive asymptotic expansions and upper bounds for the coefficients $c_k(z)$. Let us consider the alternating binomial sum defined by

$$F(z, N, m) = \sum_{k=0}^{N} \binom{N}{k} (-1)^k \frac{1}{(z+k)^m}. \tag{4.64}$$

In [32], Coffey calculates the alternating binomial sum (4.64) for $(N, m) \in \mathbb{N}$ and $z \in \mathbb{C} \setminus \mathbb{Z}_0^-$. The main result of [32] is an analytic relation in terms of Bell polynomials with generalized harmonic number arguments. We present some of his results along with other relations that will be needed further on

$$F(z, N, m) = \frac{1}{z^m} \, {}_{m+1}F_m(z, \ldots, z, -N; z+1, \ldots, z+1; 1) \tag{4.65}$$

$$= \frac{1}{(m-1)!} \int_0^{\infty} t^{m-1} e^{-zt} (1 - e^{-t})^N \, dt \tag{4.66}$$

where ${}_pF_q(a_1, \ldots, a_p; b_1, \ldots, b_q; z)$ is the generalized hypergeometric function.

The Stirling numbers of the second kind $S(n, k)$ may be defined by the following generating function

$$\sum_{k=1}^{n} S(n, k)(x - k + 1)_k = x^n, \tag{4.67}$$

and $S(n, k) = 0$ for $n < k$. We introduce the following proposition for the coefficients $c_k(z)$.

Proposition 4.3.3 *Given $a, b, z \in \mathbb{C}$, $\Re(z) > 0$ and $j, k \in \mathbb{N}$, coefficients $c_k(z)$ in (4.62) can be represented by the following asymptotic expansion as $(p+k) \to \infty$*

$$c_k(z) \sim z^{k-q} k! \sum_{j=0}^{k} \binom{q}{k-j} (j+1) \sum_{i=j+1}^{\infty} \frac{S(i,j+1)(k-q-j)_i}{i!} \zeta(k-q-j+i, p+k+1). \tag{4.68}$$

Proof: We use the integral representation (4.66) for r_j

$$r_j = \frac{1}{\Gamma(k-q-j)} \int_0^{\infty} t^{k-q-j-1} e^{-(p+k-j)t} (1-e^{-t})^j \, dt \,. \tag{4.69}$$

This Laplace-type integral is defined for $\Re(k-q-j) > 0$, where a direct use of Watson's lemma applies, see (4.72). Alternatively, note that this integral can be written as

$$\int_0^{\infty} t^{k-q-j-1} e^{-(p+k-j)t} (1-e^{-t})^j \, dt = \int_0^{\infty} t^{k-q-j-1} e^{-(p+k+1)t} \frac{(e^t-1)^{j+1}}{(1-e^{-t})} \, dt \,.$$

An asymptotic expansion for r_j can be obtained if we expand $(e^t-1)^{j+1}$ at $t=0$, which corresponds to the following generating function of the Stirling coefficients of the second kind (4.67)

$$(e^t-1)^{j+1} = (j+1)! \sum_{i=j+1}^{\infty} S(i,j+1) \frac{t^i}{i!}, \tag{4.70}$$

and interchanging the previous summation and integration we obtain a divergent asymptotic series

$$r_j \sim \frac{(j+1)!}{\Gamma(k-q-j)} \sum_{i=j+1}^{\infty} \frac{S(i,j+1)}{i!} \int_0^{\infty} t^{k-q-j+i-1} \frac{e^{-(p+k+1)t}}{(1-e^{-t})} \, dt \,. \tag{4.71}$$

The resulting integral has an explicit representation in terms of the Hurwitz zeta function [43, §25.11]. Hence, replacing the integral by the Hurwitz zeta function and substituting the ratio of gamma functions by a Pochhammer symbol gives the result. □

An equivalent asymptotic expansion representation for $(p+k) \to \infty$ is obtained by application of Watson's lemma to integral (4.69)

$$r_j \sim \frac{j!}{\Gamma(k-q-j)} \sum_{i=j}^{\infty} \frac{S(i,j)}{i!} \frac{\Gamma(k-q-j+i)}{(p+k)^{i+k-q-1}} = j! \sum_{i=j}^{\infty} \frac{S(i,j)}{i!} \frac{(k-q-j)_i}{(p+k)^{i+k-q-1}}, \tag{4.72}$$

where the use of Pochhammer symbol permits the evaluation out of the domain $\Re(k-q-j) > 0$. In [32] a similar asymptotic expansion restricted to $k-q-j \in \mathbb{N}$ is obtained after a change of variable and using the generating function for Stirling numbers of the first kind and solving the beta function.

Remark 4.3.4 *Interestingly, coefficients $c_k(z)$ have a remarkable property for (a,b) in a domain $\mathcal{D} := \{(a,b) \in \mathbb{C}^2 : b-a-1 = n, n \in \mathbb{N}\}$. For this particular case, $U(a,b,z)$*

reduces to a polynomial in z of degree n given by

$$U(a, a+n+1, z) = z^{-a} \sum_{j=0}^{n} \binom{n}{j} \frac{(a)_j}{z^j}. \tag{4.73}$$

We note, by expanding the double binomial sum in (4.62), that coefficients $c_k(z)$ vanish by symmetry for $k > \lfloor n/2 \rfloor$. This implies that expansion (4.63) terminates in almost half of terms required by (4.73).

We now prove the Franklin-Friedman expansion for $U(a, b, z)$ in (4.63).

Theorem 4.3.5 *For $(a, b, z) \in \mathbb{C}^3$ and $\Re(z) > 0$ the Franklin-Friedman expansion for the confluent hypergeometric function $U(a, b, z)$ is given by*

$$U(a, b, z) = \sum_{k=0}^{\infty} c_k(z) \frac{(a)_k}{k! z^{a+k}}, \tag{4.74}$$

where

$$c_k(z) = \sum_{j=0}^{k} \binom{k}{j} z^{k-j} \frac{\Gamma(b-a)}{\Gamma(b-a-j)} \sum_{s=j}^{k} (-1)^{s-j} \binom{k-j}{k-s} A_s^{b-a-1-j} \tag{4.75}$$

$$= \sum_{j=0}^{k} \binom{k}{j} z^{j} \frac{\Gamma(b-a)}{\Gamma(b-a+j-k)} \sum_{s=0}^{j} (-1)^{s} \binom{j}{s} A_{k-j+s}^{b-a-1-k+j}, \quad k = 0, 1, 2, \ldots, \tag{4.76}$$

and

$$A_s^q = \left(1 + \frac{a+s}{z}\right)^q. \tag{4.77}$$

Proof: It follows from (4.66) that coefficients $c_k(z)$ can be written as

$$c_k(z) = \frac{k!}{z^{q-k}} \sum_{j=0}^{k} \binom{q}{k-j} \frac{1}{j!} \sum_{s=0}^{j} (-1)^s \binom{j}{s} \frac{1}{(p+k-j+s)^{k-q-j}}$$

$$= \frac{k!}{z^{q-k}} \sum_{j=0}^{k} \binom{q}{k-j} \frac{1}{j!\Gamma(k-j-q)} \int_0^{\infty} t^{k-q-j-1} e^{-(p+k-j)t} (1-e^{-t})^j \, dt, \tag{4.78}$$

where $q = b - a - 1$ and $p = z + a$ with the integral representation being valid for $\Re(k-q-j) > 0$ and $\Re(p+k-j) > 0$. Next, we rearrange $U := \binom{q}{k-j} \frac{1}{j!\Gamma(k-j-q)}$, to obtain $U = u(q, k, j) f(k, j)$, where $f(k, j) = 1/(j!(k-j)!)$. The following identity is established for $u(q, k, j)$

$$u(q, k, j) = \frac{\Gamma(q+1)}{\Gamma(q-k+j+1)\Gamma(k-j-q)} = -\frac{\Gamma(q+1)\sin(\pi(j-k+q))}{\pi}. \tag{4.79}$$

Now replacing (4.79) into (4.78) and by formally interchanging integration and summation we obtain the following integral representation for $c_k(z)$

$$c_k(z) = -\frac{\Gamma(q+1)k!}{z^{q-k}\pi}\int_0^\infty \left(\sum_{j=0}^{k}\frac{\sin(\pi(j-k+q))e^{jt}(1-e^{-t})^j}{j!(k-j)!t^j}\right)t^{k-q-1}e^{-(p+k)t}\,dt$$
$$= \frac{\Gamma(q+1)\sin(\pi(k-q))}{z^{q-k}\pi}\int_0^\infty t^{-q-1}e^{-(p+k)t}(1-e^t+t)^k\,dt, \tag{4.80}$$

valid for $\Re(q) < 0$ and $\Re(p) > 0$. Given the explicit integral representation of $c_k(z)$ in (4.80), the proof consists of developing the sum after replacing (4.80) into (4.74). Thus, we obtain after interchanging summation and integration

$$\sum_{k=0}^{\infty}\frac{c_k(z)(a)_k}{k!z^{a+k}} = \frac{\Gamma(q+1)}{\pi z^{a+q}}\int_0^\infty\left(\sum_{k=0}^{\infty}\frac{(a)_k\sin(\pi(k-q))(1-e^t+t)^k}{e^{kt}k!}\right)t^{-q-1}e^{-pt}\,dt\,.$$

This interchange is justified since the series converges absolutely. Now we consider the following identity

$$\sum_{k=0}^{\infty}\frac{(a)_k\sin(\pi(k-q))(1-e^t+t)^k}{e^{kt}k!} = -\frac{\sin(\pi q)e^{at}}{(1+t)^a}. \tag{4.81}$$

Using (4.81), it follows that the Franklin-Friedman expansion in (4.74) is expressible in terms of the Laplace-type integral given by

$$\sum_{k=0}^{\infty}c_k(z)\frac{(a)_k}{k!z^{a+k}} = -\frac{\Gamma(q+1)\sin(\pi q)}{\pi z^{a+q}}\int_0^\infty\frac{t^{-q-1}e^{-(p-a)t}}{(1+t)^a}\,dt$$
$$= -\frac{\Gamma(b-a)\sin(\pi(b-a-1))}{\pi z^{b-1}}\int_0^\infty\frac{t^{a-b}e^{-zt}}{(1+t)^a}\,dt\,.$$

By observing that the resulting integral is indeed the integral representation of $U(a,b,z)$ in (4.13) after application of Kummer's transformation $U(a,b,z) = z^{1-b}U(1+a-b,2-b,z)$, we obtain

$$U(a,b,z) = -\frac{\Gamma(b-a)\sin(\pi(b-a-1))z^{1-b}}{\pi}\int_0^\infty\frac{t^{a-b}e^{-zt}}{(1+t)^a}\,dt \tag{4.82}$$
$$= \frac{1}{\Gamma(a)}\int_0^\infty t^{a-1}e^{-zt}(1+t)^{b-a-1}\,dt, \tag{4.83}$$

and the proof of the theorem is completed. □

Remark 4.3.6 *A relation between contiguous coefficients can be obtained by performing integration by parts on (4.80), which yields*

$$c_{k+1}(p,q;z) = z(c_k(p,q;z) - c_k(p+1,q;z)) + qc_k(p+1,q-1;z). \tag{4.84}$$

Despite the interest of the latter result, this recurrence is not the preferred choice to compute consecutive coefficients. An efficient method to compute a set of coefficients $c_k(z)$ is described in Section 4.

To conclude the analysis of coefficients $c_k(z)$, we derive an upper bound for the domain of parameters $\mathcal{D} := \{(a,b,z) \in \mathbb{C}^3 : \Re(a) > \Re(b) - 1 \wedge \Re(z+a) > 0\}$.

Proposition 4.3.7 *For $\Re(q) < 0$ and $\Re(p) > 0$, we have*

$$|c_k(z)| \leq |z^{k-q} p^q|. \tag{4.85}$$

Proof: Let us consider the integral representation of $c_k(z)$ in (4.80)

$$c_k(z) = \varphi(k,p,q) \int_0^\infty t^{-q-1} e^{-(p+k)t} (1 - e^t + t)^k \, dt,$$

where

$$\varphi(k,p,q) = \frac{z^{k-q}\Gamma(q+1)\sin(\pi(k-q))}{\pi} = (-1)^k \frac{z^{k-q}}{\Gamma(-q)}.$$

We have that the amplitude function $(1 - e^t + t)^k$ is bounded in absolute value by $|1 - e^t + t|^k \leq (1+t)^k + e^{kt}$. Hence,

$$|c_k(z)| \leq |\varphi(k,p,q)| \left(\int_0^\infty t^{-q-1} e^{-(p+k)t} (1+t)^k \, dt + \int_0^\infty t^{-q-1} e^{-pt} \, dt \right). \tag{4.86}$$

We note that the first integral is expressible in terms of Charlier polynomials of degree q [43, §13.6.20]

$$\frac{1}{\Gamma(-q)} \int_0^\infty t^{-q-1} e^{-(p+k)t} (1+t)^k \, dt = (k+p)^q C_q(k; k+p), \tag{4.87}$$

whereas the second integral is simply

$$\int_0^\infty t^{-q-1} e^{-pt} \, dt = \Gamma(-q) p^q. \tag{4.88}$$

Substituting (4.87) and (4.88) into (4.86) gives an upper bound for $c_k(z)$. A simpler bound can be easily obtained by means of the equivalent integral representation

$$c_k(z) = \frac{z^{k-q}}{\Gamma(-q)} \int_0^\infty t^{-q-1} e^{-pt} (1 - (1+t)e^{-t})^k \, dt,$$

where the amplitude function is bounded in absolute value by $|1 - (1+t)e^{-t}| \leq 1$. Thus, using the result in (4.88) gives the bound. □

Table 4.5 shows effectiveness of the bound for several values $(a,b,z) \in \mathcal{D}$ and k. Such a bound can be used to determine the truncation level N due to the convergent behaviour of the expansion.

Furthermore, application of Watson's lemma to integral representation (4.80) gives a first-order asymptotic approximation for $(p+k) \to \infty$ given by

$$c_k(z) \sim \frac{z^{k-q}\Gamma(q+1)\sin(\pi(k-q))}{\pi} \frac{\Gamma(2k-q)}{2^k (k+p)^{2k-q}} = (-1)^k \frac{z^{k-q}(-q)_{2k}}{2^k (k+p)^{2k-q}}. \tag{4.89}$$

Table 4.6 shows a few examples of the above asymptotic approximation of $c_k(z)$ for large argument.

a	b	z	k	$\lvert c_k \rvert$	**(4.85)**
-500.1	-602.4	770	10	1.1e+64	7.9e+75
710.2	72.5	1500	15	2.7e−82	1.3e−60
-50.1	-62.4	70	20	2.8e+34	1.5e+44
-10.1	-52.4	80	30	1.3e+40	4.3e+59
-5.1	-62.4	40	50	5.4e+73	3.6e+83

TABLE 4.5: Effectiveness of bound on $c_k(z)$ in (4.85) for $q < 0$ and $p > 0$.

a	b	z	k	$c_k(z)$	**(4.89)**
12.1	10.4	100	10	5.7e−05	4.6e−05
22.1	11.4	100	50	1.7e+36	1.8e+38
20.1	12.4	280	50	3.9e+23	3.6e+25

TABLE 4.6: Asymptotic approximation on $c_k(z)$ in (4.89) for $q < 0$ as $p + k \to \infty$.

Finally, we note that the terms of the Franklin-Friedman expansion (4.63) satisfy the order estimate

$$c_k(z)\frac{(a)_k}{k!z^{k+a}} = O(2^k k^{a-3/2} e^{-2k} h(z)), \quad k \to \infty, \quad p \to \infty, \tag{4.90}$$

where $h(z) = (1 + p/k)^{-2k+q}$. This result is obtained combining the asymptotic estimate in (4.89) and the usual estimates for the factorial and Pochhammer symbol given by

$$k! = O\left(k^{k+1/2}e^{-k}\right), \quad (a)_k = O\left(k^{a+k-1/2}e^{-k}\right), \quad k \to \infty.$$

4.3.5 Efficient computation of $U(a, b, z)$

Equations (4.58), (4.60) and (4.62) are explicit representations that can be used to compute $c_k(z)$ directly, but a double binomial sum turns out to be significantly expensive as k grows. Furthermore, from a numerical perspective, the evaluation of alternating binomial sums are prone to suffer from substantial cancellation. As we shall see, the direct computation of $c_k(z)$ can be avoided by constructing a recurrence equation for generating a set of $c_k(z)$, $k \in \{0, \ldots, N\}$. This idea leads to the following proposition and Algorithm 6.

Proposition 4.3.8 *The coefficients $c_k(z)$ of the Franklin-Friedman expansion for the amplitude function $f(t) = (1 + t)^{b-a-1}$ satisfy the recurrence equation*

$$c_k(z) = u_k + \sum_{i=0}^{k-1} \binom{k}{i} z^{k-i} u_i, \tag{4.91}$$

where

$$u_k = A_k^{b-a-1-k} k! L_k^{b-a-1-k}(z + a + k) \quad \text{and} \quad c_0 = u_0 = A_0^{b-a-1}, \tag{4.92}$$

$L_k^{\lambda}(z)$ being generalized Laguerre polynomials.

Proof: We start defining u_k as the leading coefficient $f(A_k)$, resulting from the iteration $j = 0$ in (4.60). After grouping the remaining coefficients $f(A_j)$, $j < k$ in ascending order j, we identify that each $f(A_j)$ is a non-linear combination of the previous leading terms u_j, $j < k$. These non-linear terms in the recurrence equation are indeed binomial coefficients $\binom{k}{j}$ times z^{k-j}. This gives the following expression for u_k, which can be expressed in terms of generalized Laguerre polynomials as follows

$$\begin{aligned} u_k &= \sum_{j=0}^{k} (-1)^{k-j} \binom{k}{j} z^{k-j} d_j A_k^{b-a-1-j} \\ &= A_k^{b-a-1-k} \sum_{j=0}^{k} \binom{k}{j} (-zA_k)^{k-j} \frac{\Gamma(b-a)}{\Gamma(b-a-j)} \\ &= A_k^{b-a-1-k} k! L_k^{b-a-1-k}(zA_k) \end{aligned} \tag{4.93}$$

and d_j is defined as in (4.59). Taking $zA_k = z + a + k$ gives the final closed form for u_k. □

Recently, several variants of asymptotic expansions for large order k have been extensively studied for generalized Laguerre polynomials. For example, the paper [13] provides a treatment for the region of sub-exponential behaviour and the recent paper [45] studies uniform asymptotic expansions for a larger domain of the parameters. Although the computation of $c_k(z)$ by means of computing Laguerre polynomials would certainly reduce cancellation effects, there is a substantial computational cost involved.

In order to bypass the computation of generalized Laguerre polynomials and the direct computation in (4.62), we introduce a fast algorithm for computing $c_k(z)$, see Algorithm 6. This algorithm combines both binomial expansion sums to reuse the binomial coefficients, so in practice $c_k(z)$ at u_k are computed at once. In terms of time complexity, computing the first N coefficients $c_k(z)$ using Algorithm 6 has complexity $O(N^2)$, whereas clearly a direct computation has complexity $O(N^3)$. Additionally, note that coefficients d_j in (4.59) do not need to be computed for each j but for k, thus avoiding redundant operations. In terms of algorithmic aspects, given a suitable truncation level N, the complete Pascal's triangle until row N can be pre-computed with complexity $O(N^2)$, obviously computing only half rows. On the other hand, in terms of space complexity, computing the first N coefficients with Algorithm 6 has complexity $O(2N)$, due to the storage of successive u_k and d_k. As we shall see later, this is not an issue due to the small number of terms needed to obtain high accuracy. In general, several parts of the algorithm can be easily pre-computed in parallel, for example a block of $k : k \leq N$ coefficients $A_k^{b-a-1-j}$ can be distributed to each thread. However, although pre-computations improve efficiency, they require substantial space, especially for high-precision computations. Concerning working precision, higher precision arithmetic is normally needed to satisfy the requested accuracy, especially for large values of the parameters and argument. Based on experiments, we found that to guarantee good performance a working precision of p bits must satisfy $p \gtrsim 2N$, N being the number of terms in expansion (4.63). For this same reason, approaches based on performing linear search to obtain an optimal number of terms N using floating-point arithmetic are

less likely to succeed.

Algorithm 6 Fast computation of coefficients $c_k(z)$

Input: $a, b, z \in \mathbb{C}$, $\Re(z) > 0$, $N \in \mathbb{N}$
Output: $c_k(z)$, $k \in \{0, \ldots, N\}$

Pre-compute Pascal's triangle, $PascalRow(k)$, $k \in \{0, \ldots, N\}$
$U = [\,], C = [\,], D = [\,]$ ▷ Empty cache of coefficients c, u and d
$U[0] \leftarrow C[0] \leftarrow (1 + a/z)^{b-a-1}$ and $D[0, 1] \leftarrow [1, b - a - 1]$
for $k = 1; k = N; k \leftarrow k + 1$ **do**
    $r \leftarrow 0, t \leftarrow 0$
    $f \leftarrow z^k$
    $h \leftarrow 1/z$
    $m \leftarrow 1 + (a + k)/z$
    $n \leftarrow 1/m$
    $q \leftarrow m^{b-a-1}$
    $R = PascalRow(k)$ ▷ k-th row of pre-computed Pascal's triangle
    for $j = 0; j = k; j \leftarrow j + 1$ **do**
        $u \leftarrow f \cdot R[j]$
        $p \leftarrow (-1)^{k-j} \cdot u \cdot D[j] \cdot q$
        $t \leftarrow t + p$
        $f \leftarrow f \cdot h$
        $q \leftarrow q \cdot n$
        if $j < k$ **then**
            $r \leftarrow r + U[j] \cdot u$
        else
            $D[j + 1] \leftarrow D[j] \cdot (b - a - 1 - j)$ ▷ Store next d
        end if
    end for
    $U[k] \leftarrow t$ ▷ Store block u_k to posterior use
    $r \leftarrow r + t$
    $C[k] \leftarrow r$ ▷ Store new coefficient $c_k(z)$
end for

4.3.6 Numerical experiments

In this Section we compare expansion (4.63) with other asymptotic and convergent series previously mentioned. The described algorithms has been implemented in Python using the Mpmath library for multi-precision floating-point arithmetic [92]. Firstly, we compare (4.63) with the convergent expansion for $U(a, b, z)$ in [129], which we briefly summarize:

$$U(a, a + b, z) = z^{-a} \sum_{k=0}^{\infty} \frac{(-1)^k (a)_k (1 - b)_k}{k! z^k} P(k + a, \vartheta x) + \frac{e^{-ia\theta}}{\Gamma(a)} \sum_{n=1}^{\infty} e^{-\Omega_n x} S_n(x; \theta), \tag{4.94}$$

valid in $|\arg z| < \pi$ with $z = xe^{i\theta}$ and $x = |z|$. The second series is defined as

$$S_n(x;\theta) = \sum_{k=0}^{\infty} \frac{c_{k,n}(\theta)}{x^{k+1}} \Delta P\left(k+1, \frac{\omega_n x}{2}\right), \quad \Delta P(m+1,z) = P(m+1,z) - P(m+1,-z).$$

Coefficients $c_{k,n}(\theta)$ can be computed via recurrence relation. This method subdivides the integration path in (4.13) into intervals of length $\frac{1}{2}\omega_n$, where $\frac{1}{2}\omega_n = \vartheta$, $\vartheta \in (0,1]$ and Ω_n denote the mid-points of these intervals, being both freely chosen. As shown in [129, §4], expansion (4.94) converges geometrically without restriction on the parameters a and b when $\vartheta < 1$. It is observed that evaluation of the second series may be unavoidably expensive, even though the incomplete gamma function can be computed via recursion.

We consider the example in [129, §5] to compare both expansions. This example evaluates the modified Bessel function $K_\nu(z)$ given by

$$e^z K_\nu(z) = \sqrt{\pi}(2z)^\nu U\left(\nu + \frac{1}{2}, 2\nu + 1, 2z\right). \tag{4.95}$$

We reproduce Table 3 in [129, §5] for different values of θ with convergence rate $\vartheta = \frac{1}{2}$ and $n = 2$, i.e. using the first two terms of the second expansion in (4.94). We skip the improved case $\vartheta = \frac{1}{3}$ and $n = 4$ in [129, §5] due to the notable computational effort. Results reveal that for $z \in \mathbb{R}$, expansion (4.63) gives more correct digits than Hadamard expansion, but for $\Re(z) \leq \Im(z)$ expansion (4.63) is affected by a progressive loss of accuracy due to the omission of path rotation arguments like those employed in [129].

(θ/π)	Hadamard (4.94)	Expansion (4.63)
0	4.3e−43	1.3e−50
0.125	8.3e−43	2.4e−49
0.250	9.8e−43	3.0e−45
0.375	5.3e−43	1.3e−38
0.500	6.4e−43	2.3e−28

TABLE 4.7: Comparison of the absolute error values when $z = 15e^{i\theta}$ and $\nu = \frac{3}{4}$. Series truncated at $N = 100$ terms.

In the following tables we compare the performance of the ascending series, asymptotic series and the vanishing saddle point series. The vanishing saddle point series for $U(a,b,z)$ is given by [157, §25.4]

$$U(a,b,z) \sim \sum_{k=0}^{\infty} \frac{(1+\mu)^{b-a-1-k}\binom{b-a-1}{k} P_k(a)}{z^{a+k}}, \tag{4.96}$$

where $\mu = a/z$ is the saddle point of the dominant part of the integral (4.13). Coefficients $P_k(a)$ are expressible in terms of generalized Laguerre polynomials defined by

$$P_k(a) = k! L_k^{-k-a}(-a), \tag{4.97}$$

and satisfy the following recursion relation

$$P_0(a) = 1, \quad P_1(a) = 0, \quad \text{and} \quad P_{k+1}(a) = k(P_k(a) + aP_{k-1}(a)), \quad k = 1,2,\ldots.$$

For comparison, absolute relative errors are computed using as a reference Arb [87] `hypergeometric_U` evaluated at 5000-10000 bits of precision. Symbol (-) indicates an absolute error > 1. Table 4.8 shows the absolute relative errors for large parameters and argument when truncated at N terms. For these cases each term of expansion (4.63) adds about 1 digit of accuracy every term. In particular, the first case in Table 4.8 corresponds to the generalized exponential integral $E_\nu(z) = z^{\nu-1}U(\nu,\nu,z)$. We remark that this special case can be computed with faster methods, for example the Laguerre expansion described in [118] returns an absolute relative error of magnitude 1.4e$-$294 with $N = 200$ terms.

(a,b,z)	N	Asymptotic (4.53)	Vanishing (4.96)	Expansion (4.63)
$(600,600,500)$	10	-	3e$-$14	6e$-$25
	30	-	7e$-$34	1e$-$59
	50	-	7e$-$50	6e$-$88
	100	-	4e$-$82	5e$-$146
	200	-	3e$-$128	2e$-$234
$(100,1,1000)$	10	-	1e$-$04	1e$-$10
	30	-	4e$-$15	8e$-$38
	50	4e$-$02	2e$-$26	7e$-$66
	100	1e$-$27	2e$-$54	4e$-$132
	200	9e$-$60	1e$-$102	2e$-$245
$(1000,500,5000)$	10	-	3e$-$01	3e$-$03
	30	-	2e$-$05	2e$-$17
	50	-	1e$-$11	2e$-$36
	100	-	2e$-$31	8e$-$92
	200	-	4e$-$78	3e$-$213

TABLE 4.8: Comparison between various methods for $U(a,b,z)$. Large parameters and argument.

Table 4.9 shows the performance of expansion (4.63) for small and moderate positive values of parameters and argument. For these cases, we incorporate the ascending series in (4.56) into the first column, where the value within parentheses indicates the required number of terms to obtain that absolute relative error. We notice that for sufficiently small argument z the ascending series out-performs the Franklin-Friedman expansion, although for moderate values the latter substantially improves both the ascending and asymptotic expansion.

Finally, Table 4.10 shows cases with moderate negative parameters and positive argument. For these cases expansion (4.63) exhibits fast convergence giving about 1.7 digits of accuracy every term. We remark that, as described in [129], Hadamard series accuracy deteriorates when a is large and negative due to the oscillatory behaviour of the incomplete gamma function.

4.3.7 Discussion

The Franklin-Friedman expansion developed provides a uniform approach to evaluating confluent hypergeometric functions to arbitrary-precision for sufficiently large $\Re(z) > 0$ and outside this sector via connection formulas. This expansion is generally convergent and especially useful for the transition region between the ascending and asymptotic series. It is found that the expansion developed systematically out-performs the vanishing saddle point series and asymptotic expansion

(a,b,z)	N	Ascending (4.56)	Asymptotic (4.53)	Vanishing (4.96)	Expansion (4.63)
$(30, 81/4, 300)$	10	-	1e−04	6e−10	8e−20
	30	-	1e−16	8e−26	1e−52
	50	-	1e−27	8e−39	3e−79
	100	-	5e−48	6e−63	5e−131
	200	1e−13 $(N = 1000)$	4e−68	3e−88	4e−202
$(123/4, 101/5, 50)$	10	-	-	8e−04	6e−08
	30	-	-	3e−07	7e−20
	50	-	-	7e−08	2e−28
	100	-	-	-	4e−43
	300	2e−48	-	-	9e−71
$(5/4, 10/4, 30)$	10	-	1e−10	2e−11	8e−19
	30	-	5e−15	4e−16	4e−32
	50	-	7e−13	3e−14	2e−39
	100	6e−11	-	-	4e−50
	200	6e−80	-	-	4e−61
$(401/2, 211/6, 300)$	10	-	-	-	7e−01
	30	-	-	-	1e−04
	50	-	-	-	5e−11
	100	-	-	-	6e−30
	200	4e−53 $(N = 1600)$	-	-	3e−66

TABLE 4.9: Comparison between various methods for $U(a,b,z)$. Small and moderate values of parameters and argument.

(a,b,z)	N	Asymptotic (4.53)	Vanishing (4.96)	Expansion (4.63)
$(-241/2, 20, 400)$	10	-	-	-
	30	-	-	-
	50	-	-	2e−19
	100	1e−11	3e−16	2e−170
	200	1e−226	4e−138	2e−375
$(-500/6, -21/6, 300)$	10	-	-	-
	30	-	-	3e−21
	50	-	9e−11	2e−87
	100	9e−118	7e−78	9e−212
	200	6e−225	1e−143	3e−334

TABLE 4.10: Comparison between various methods for $U(a,b,z)$. Moderate negative parameters and argument.

in their respective regions of validity and is remarkably useful to compute the confluent hypergeometric function for large parameters and argument, providing a clear advantage over direct computation using the ascending series. Furthermore, it is observed that for small argument the expansion developed converges at approximately geometric rate 2^{-N}. However, for sufficiently small argument the ascending series still exhibits faster convergence.

The presented approach results adequate in a “medium/high-precision" range, say 100 - 1000 digits, given the considerable computation complexity as the number of terms increases. Therefore, a complete arbitrary-precision implementation should combine the Franklin-Friedman expansion with other low-complexity methods described in this work.

Finally, further work is needed to compute uniform bounds for the coefficients of the expansion in a wider regions of the parameters a and b, and to enhance the algorithm to accelerate the computation of coefficients via efficient parallelization.

Chapter 5

The Lerch Transcendent and Other Special Functions in Analytic Number Theory

5.1 Background

5.1.1 Special number and polynomials

We introduce some basic properties of special numbers and polynomials essential in the field of analytic number theory and asymptotic expansions for special functions. Most of the section shall be devoted to the Bernoulli numbers and polynomials given their number of appearances throughout this work and their relevance in the field. The number of formulae relating special numbers and polynomials between each other is considerable and practically infeasible to cover in detail.

Bernoulli numbers and polynomials

The Bernoulli polynomials $B_n(z)$, for $n \in \mathbb{N}_0$ and $z \in \mathbb{C}$, are defined by their exponential generating function (EGF)

$$\frac{te^{zt}}{e^t - 1} = \sum_{n=0}^{\infty} B_n(z)\frac{t^n}{n!}, \quad |t| < 2\pi, \tag{5.1}$$

and the Bernoulli numbers B_n by $B_n(0) = B_n$. Some basic properties of Bernoulli numbers and polynomials are the following

- Bernoulli numbers are rational numbers. B_n is defined for $n \in \{0, 1\}$ and, $n \geq 2$ and n even, otherwise $B_n = 0$. On the other hand, the even-indexed B_n alternate in sign, thus $(-1)^{n+1}B_{2n} > 0$, $n \geq 1$.
- The exact denominator of B_n is obtained by the Von Staudt-Clausen theorem. The sequence of the numerator and denominators of the rational B_n are detailed in [144, A027641] and [144, A027642], respectively.
- Bernoulli polynomials are expressible in terms of Bernoulli numbers

$$B_n(z) = \sum_{k=0}^{n} \binom{n}{k} B_k z^{n-k}. \tag{5.2}$$

- They satisfy the difference equation: $B_n(z+1) - B_n(z) = nz^{n-1}$.
- The derivative w.r.t. argument z: $\frac{d}{dz}B_n(z) = nB_{n-1}(z)$.
- Multiplicative formula is given by the finite series [136],

$$B_n(mx) = m^{n-1}\sum_{k=0}^{m-1} B_n\left(x + \frac{k}{m}\right), \quad m \geq 2 \text{ and } m \text{ even}.$$

A generalization of Bernoulli polynomials denoted by $B_n^{(\mu)}(z)$ for complex order and argument, and integer degree is described in [157, §15.6]. The generalized Bernoulli polynomials, like previous particular cases, are defined by an exponential generating function given by [43, §24.16.1]

$$\frac{t^\mu e^{zt}}{(e^t-1)^\mu} = \sum_{n=0}^{\infty} B_n^{(\mu)}(z)\frac{t^n}{n!}, \quad |t| < 2\pi. \tag{5.3}$$

Note that when $\mu = 1$, they reduce to the conventional Bernoulli polynomials and numbers $B_n^{(1)}(z) = B_n(z)$ and $B_n^{(1)}(0) = B_n$. As previously stated, generalized Bernoulli polynomials can be computed recursively with the following formula

$$B_n^{(\mu)}(z) = \sum_{k=0}^{n}\binom{n}{k}B_{n-k}^{(\mu)}z^k = z^n\sum_{k=0}^{n}\binom{n}{k}B_k^{(\mu)}z^{-k}. \tag{5.4}$$

Therefore, these polynomials are expressible in terms of a finite sum of generalized Bernoulli numbers, also called Nörlund numbers, $B_n^{(\mu)} = B_n^{(\mu)}(0)$. Nörlund numbers are, therefore, easily defined by the generating function [43, §24.16.9]

$$\left(\frac{t}{e^t-1}\right)^\mu = \sum_{n=0}^{\infty} B_n^{(\mu)}\frac{t^n}{n!}, \quad |t| < 2\pi. \tag{5.5}$$

Several methods have been developed to perform the efficient computation of many Bernoulli numbers. In what follows, we aim to summarize some common approaches implemented in arbitrary-precision software packages. An exhaustive chronology of earlier work with several references is included in [75]. The most widely used algorithm for computing a single B_n is based on the application of the relationship between B_n and the Riemann zeta function $\zeta(s)$, given by

$$B_n = (-1)^{n/2+1}\frac{2\zeta(n)n!}{(2\pi)^n}, \quad n \geq 2 \text{ and } n \text{ even}. \tag{5.6}$$

A state-of-the-art parallel implementation is the software CalcBn V3.0 (C++ making use of the GMP library) `https://www.bernoulli.org/`. A faster method developed in [75] computes B_n modulo p for many small primes p and then reconstruct B_n via the Chinese Remainder Theorem. This is the default algorithm in Sage [160] and was used to achieve a new record (2008), B_n for $n = 10^8$. On the other hand, the recurrent formula from Ramanujan's congruences is the method of choice in mpmath [92] for $n < 3000$, whereas the Riemann zeta function relationship is exploited for

larger values. Ramanujan congruences are

$$\binom{n+3}{n} B_n = \begin{cases} \frac{n+3}{3} - \sum_{k=1}^{n/6} \binom{n+3}{n-6k} B_{n-6k}, & \text{if } n \equiv 0 \pmod 6 \\ \frac{n+3}{3} - \sum_{k=1}^{(n-2)/6} \binom{n+3}{n-6k} B_{n-6k}, & \text{if } n \equiv 2 \pmod 6 \\ -\frac{n+3}{3} - \sum_{k=1}^{(n-4)/6} \binom{n+3}{n-6k} B_{n-6k}, & \text{if } n \equiv 4 \pmod 6 \end{cases}, \tag{5.7}$$

where $a \equiv b \pmod c$ indicates that a and b are congruent modulo c, i.e. $a - b$ is integrally divisible by c, thus $(a-b)/c \in \mathbb{Z}$.

The Arb library [87] implementation also exploits the relationship between B_n and the Riemann zeta function, but instead of using the Euler product for $\zeta(n)$, as in various aforementioned implementations, a reversing computation of the L-series to favour the recycling of powers is used. Finally, it is worth mentioning that most of the libraries systematically catch Bernoulli numbers for subsequent calculations, since the computation of a large number of B_n at high precision is time-consuming.

The number of available methods of computation of Bernoulli polynomials and generalized Bernoulli polynomials is limited if compared with Bernoulli numbers. To the author knowledge the main methods implemented are based on the recurrences (5.2) and (5.4). However, this recurrence is numerically unstable if applied using floating-point arithmetic. Alternatively, $B_n(z)$ are expressible in terms of the Hurwitz zeta function $\zeta(s,a)$ via the functional relation

$$B_n(z) = -n\zeta(1-n, z). \tag{5.8}$$

Note that the Hurwitz zeta function generalizes the Bernoulli polynomials to $n \in \mathbb{C}$. Furthermore, some explicit formulas have been developed such as

$$B_n(x) = \sum_{k=0}^{n} \frac{1}{k+1} \sum_{j=0}^{k} (-1)^j \binom{k}{j} (x+j)^n, \tag{5.9}$$

which might also cause numerical instability due to the binomial alternating sum. Finally, for $z \in [0,1]$, the Fourier series in [43, §24.8] can be applied.

For the generalized Bernoulli polynomials, extensions to complex z, μ and n and uniform asymptotic expansion for large argument z and degree n are studied in [157, §15.6]. Explicit formulas for the generalized Bernoulli polynomials and Nörlund polynomials are studied in [148], these are respectively

$$B_n^{(\mu)} = \sum_{k=0}^{n} (-1)^k \binom{\mu+n}{n-k} \binom{\mu+k-1}{k} \frac{S(n+k,k)}{\binom{n+k}{k}}, \tag{5.10}$$

where $S(n,k)$ are Stirling numbers of the second kind, described later on, and

$$B_n^{(\mu)}(z) = \sum_{k=0}^{n} \binom{n}{k} \binom{\mu+k-1}{k} \frac{k!}{(2k)!} \sum_{j=0}^{k} (-1)^j \binom{k}{j} j^{2k} (z+j)^{n-k} \times {}_2F_1\left(k-n, k-\mu; 2k+1; \frac{j}{z+j}\right), \tag{5.11}$$

where ${}_2F_1$ is the Gauss hypergeometric function, see Chapter 4.

Euler numbers and polynomials

Analogous to the generalized Bernoulli polynomials, the generalized Euler polynomials are defined by means of the exponential generating function [43, §24.16.2]

$$\frac{2^{\mu}}{(e^t+1)^{\mu}}e^{zt}=\sum_{n=0}^{\infty}E_n^{(\mu)}(z)\frac{t^n}{n!},\quad |t|<\pi. \tag{5.12}$$

The Euler polynomials are defined by $E_n(z)=E_n^{(1)}(z)$, and the classical Euler numbers by $E_n=2^nE_n(1/2)$, which leads to the generating function for Euler numbers given by

$$\operatorname{sech}(t)=\frac{1}{\cosh(t)}=\frac{2}{e^t+e^{-t}}=\sum_{n=0}^{\infty}E_n\frac{t^n}{n!},\quad |t|<\frac{\pi}{2}. \tag{5.13}$$

Euler polynomials satisfy the recurrence formula

$$E_n(z)=\sum_{k=0}^{n}\binom{n}{k}E_k\frac{(z-1/2)^{n-k}}{2^k}. \tag{5.14}$$

The even-indexed Euler numbers are expressible in terms of Bernoulli polynomials as follows

$$E_{2n}=-4^{2n+1}\frac{B_{2n+1}(1/4)}{2n+1}, \tag{5.15}$$

and Euler polynomials are related to the Bernoulli polynomials by

$$E_{n-1}(z)=\frac{2}{n}\left(B_n(x)-2^nB_n(x/2)\right). \tag{5.16}$$

Finally, we refer to [34, §9] and [43, §24] for further properties. For uniform asymptotic expansion of the generalized Euler polynomials see [157, §15.7], which derivation follow the same steps as those of the generalized Bernoulli polynomials.

Stirling numbers and polynomials

The Stirling polynomials are defined by the exponential generating function

$$\left(\frac{t}{1-e^{-t}}\right)^{z+1}=\sum_{n=0}^{\infty}S_n(z)\frac{t^n}{n!},\quad |t|\le 2\pi. \tag{5.17}$$

Note that Stirling polynomials represent a special case of the generalized Bernoulli polynomials in (5.3). This can be observed by rewriting $(t/(1-e^{-t}))^{z+1}=(te^t/(e^t-1))^{z+1}$ hence $S_n(z)=B_n^{(z+1)}(z+1)$ follows.

Stirling numbers of the first kind, denoted as $s(n,m)$, are related to the Stirling polynomials by

$$S_n(m)=\frac{(-1)^n}{\binom{m}{n}}s(m+1,m+1-n). \tag{5.18}$$

The Stirling numbers of the second kind, denoted as $S(n,m)$, are also related by

$$S_n(-m) = \frac{(-1)^n}{\binom{n+m-1}{m-1}} S(n+m-1, m-1). \tag{5.19}$$

Given $m, n \in \mathbb{Z}$ and $m \leq n$, both Stirling numbers share the following properties

$$s(n,n) = S(n,n) = 1,\ n \geq 0 \quad \text{and} \quad s(n,0) = S(n,0) = 0,\ n \geq 1, \tag{5.20}$$

and $s(n,m) = S(n,m) = 0$ if $m > n$.

The Stirling numbers of the first kind, also known as ordinary or signed Stirling numbers of the first kind, represent the number of permutations of n elements with m disjoint cycles. Several generating functions exist

$$(t-n+1)_n = \sum_{m=1}^{n} s(n,m) t^m \tag{5.21}$$

$$\frac{(\ln(t+1))^m}{m!} = \sum_{n=m}^{\infty} s(n,m) \frac{t^n}{n!}, \tag{5.22}$$

for $t < 1$. These numbers satisfy the recurrence relation

$$s(n,m) = s(n-1, m-1) - (n-1)s(n-1,m), \quad 1 \leq m \leq n, \tag{5.23}$$

and can be computed iteratively by means of the finite summation

$$s(n,m) = \sum_{k=m}^{n} n^{k-m} s(n+1, k+1), \quad m \geq 1. \tag{5.24}$$

The Stirling numbers of the second kind are non-negative integers representing the number of partitions of n elements into exactly m non-empty subsets. These are defined by the generating functions

$$t^n = \sum_{m=1}^{n} S(n,m)(t-m+1)_m \tag{5.25}$$

$$\frac{(e^t-1)^m}{m!} = \sum_{n=m}^{\infty} S(n,m) \frac{t^n}{n!}. \tag{5.26}$$

The Stirling numbers of the second kind have an explicit representation

$$S(n,m) = \frac{1}{m!} \sum_{k=0}^{m} (-1)^{m+k} \binom{m}{k} k^n. \tag{5.27}$$

Equivalently to the Stirling numbers of the first kind, they satisfy the recurrence relations

$$S(n,m) = mS(n-1,m) + S(n-1,m-1) \tag{5.28}$$

$$S(n,m) = \sum_{k=m}^{n} m^{n-k} S(k-1, m-1). \tag{5.29}$$

Interestingly, the Stirling numbers are related to other special numbers; for example the Bernoulli numbers are expressible in terms of the Stirling numbers of the second kind as follows

$$B_n = \sum_{k=0}^{n} (-1)^k \frac{k! S(n,k)}{k+1} = \sum_{k=0}^{n} (-1)^k \binom{n+1}{k+1} \frac{S(n+k,k)}{\binom{n+k}{n}}. \tag{5.30}$$

Although the general recurrence and direct summation is usually implemented in numerical libraries such as Sage [160], we mention that several methods for obtaining asymptotic expansion for large order n and uniform asymptotic expansions when $n \sim m$ for both Stirling numbers are described in [157, §24].

Other special numbers and polynomials

For completeness we give a non-exhaustive short overview of other combinatorial numbers and polynomials of interest in the field of number theory, some of them being used in various asymptotic formulas derived throughout this work. We start with some less-known list of special numbers expressible in terms of Bernoulli numbers

- Genocchi numbers are defined by the exponential generating function

$$\frac{2t}{e^t - 1} = \sum_{n=1}^{\infty} G_n \frac{t^n}{n!}, \tag{5.31}$$

  and are related to the Bernoulli numbers through the relation

$$G_n = 2(1 - 2^n) B_n. \tag{5.32}$$

- Tangent numbers are defined by the exponential generating function

$$\tan(t) = \sum_{n=0}^{\infty} T_n \frac{t^n}{n!}. \tag{5.33}$$

  These are related to the Bernoulli numbers by means of the functional relation

$$T_{2n-1} = (-1)^{n-1} \frac{2^{2n}(2^{2n} - 1)}{2n} B_{2n}, \quad T_{2n} = 0, \quad n \geq 1, \tag{5.34}$$

  and $T_0 = 0$. The first n secant numbers S_n, also known as zig numbers and expressible in terms of Euler numbers, $S_n = |E_{2n}|$, and Tangent numbers can be computed very efficiently using asymptotically fast algorithms in $O(n^2 \log(n)^{2+o(1)})$ bit-operations, as described [20].

- Bernoulli numbers of the second kind are defined by the generating function

$$\frac{t}{\log(1+t)} = \sum_{n=0}^{\infty} b_n t^n, \quad |t| < 1, \tag{5.35}$$

and are expressible in terms of Nörlund numbers as follows

$$n!b_n = -\frac{1}{n-1}B_n^{(n-1)}, \quad n \geq 2. \tag{5.36}$$

The Eulerian polynomials are defined by the exponential generating function

$$\frac{x-1}{x-e^{(x-1)t}} = \sum_{n=0}^{\infty} A_n(x)\frac{t^n}{n!}. \tag{5.37}$$

Properties of the Eulerian polynomials are described in the next Section. The Eulerian numbers, usually denoted as $A(n,m)$ or $\left\langle {n \atop m-1} \right\rangle$, give the number of permutations of set of n elements having m permutation ascents, i.e. m elements are greater than the previous element. They are the coefficient of the Eulerian polynomials

$$A_n(x) = \sum_{m=0}^{n} A(n,m)x^m. \tag{5.38}$$

The Eulerian numbers satisfy the recurrence relation

$$A(n,m) = (n-m+1)A(n-1,m-1) + mA(n-1,m), \tag{5.39}$$

and have the explicit formulas

$$\left\langle {n \atop k} \right\rangle = \sum_{j=0}^{k}(-1)^j \binom{n+1}{j}(k+1-j)^n \tag{5.40}$$

$$\left\langle {n \atop k} \right\rangle = \sum_{j=0}^{n-k}(-1)^{n-k-j}j!\binom{n-j}{k}S(n,j), \tag{5.41}$$

the latter in terms of the Stirling numbers of the second kind. The Eulerian numbers are also connected to Bernoulli numbers [170]

$$B_n = \sum_{k=0}^{n} \left\langle {n \atop k} \right\rangle \frac{(-1)^k}{k+1}, \quad n \geq 0. \tag{5.42}$$

The Bell polynomials $B_n(x)$ (not be confused with Bernoulli polynomials) have the exponential generating function

$$e^{(e^t-1)x} = \sum_{n=0} B_n(x)\frac{t^n}{n!}, \tag{5.43}$$

and they are connected with Stirling numbers of the second kind through the explicit formula

$$B_n(x) = \sum_{k=0}^{n} x^k S(n,k). \tag{5.44}$$

The computation of Bell polynomials can be extended to complex order and argument by using the Dobiínski's formula given by

$$B_n(x) = e^{-x} \sum_{j=0}^{\infty} \frac{j^n}{j!} x^j. \tag{5.45}$$

For $x = 1$, $B_n(1) = B_n$ is the n-th Bell number, which represent the number of class-partitions of a finite set with n elements. A well-known asymptotic formulas for B_n is given in [39]. A recent discussion about methods for efficient computation of large Bell numbers with arbitrary-precision can be found in http://fredrikj.net/blog/2015/08/computing-bell-numbers/. Last, we recall that uniform upper bounds for the Bell polynomials and Bell numbers are derived in Chapter 3.

5.1.2 The Lerch transcendent and related functions

The most prominent and complete work aiming to present a unified framework for the Lerch transcendent and other related functions was carried out by Crandall in [37]. In this subsection, we present some well-known functional relations involving the Lerch transcendent. We briefly introduce the Lerch transcendent notation, and we shall refer to the next section for extended treatment and presentation of numerical methods for its computation. Finally, a quite complete list of zeta functions is given in https://en.wikipedia.org/wiki/List_of_zeta_functions.

The Lerch transcendent is defined by the L-series

$$\Phi(z, s, a) = \sum_{k=0}^{\infty} \frac{z^k}{(k+a)^s}, \tag{5.46}$$

which converges for $|z| \leq 1$, $a \notin \mathbb{Z}_0^-$ or $\Re(s) > 1$, $|z| = 1$ and is defined by analytic continuation elsewhere. Relevant special cases include the Riemann zeta function

$$\zeta(s) = \sum_{k=1}^{\infty} \frac{1}{k^s} = \Phi(1, s, 1). \tag{5.47}$$

Furthermore, by analytic continuation the following alternating series valid for $\Re(s) > 0$ is obtained

$$\zeta(s) = \frac{1}{1 - 2^{1-s}} \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n^s}, \quad \Re(s) > 0 \tag{5.48}$$

where the sum on the right-hand side is exactly the Dirichlet eta function $\eta(s)$ or alternating zeta function given by

$$\eta(s) = \sum_{k=1}^{\infty} \frac{(-1)^{k-1}}{k^s} = (1 - 2^{1-s})\zeta(s) = \Phi(-1, s, 1). \tag{5.49}$$

The Riemann zeta function is also expressible in terms of Stieltjes constants γ_k by means of the Laurent expansion

$$\zeta(s) = \frac{1}{s-1} + \sum_{k=0}^{\infty} (-1)^n \gamma_n \frac{(s-1)^n}{n!}. \tag{5.50}$$

The Hurwitz zeta function is defined by analytic continuation of the sum

$$\zeta(s,a) = \sum_{k=0}^{\infty} \frac{1}{(a+k)^s} = \Phi(1,s,a), \tag{5.51}$$

and has a Laurent expansion in terms of generalized Stieltjes constants $\gamma_k(a)$ given by

$$\zeta(s,a) = \frac{1}{s-1} + \sum_{k=0}^{\infty} (-1)^k \gamma_k(a) \frac{(s-1)^k}{k!}. \tag{5.52}$$

Other related functions representable in terms of the Lerch transcendent are the Dirichlet beta function

$$\beta(s) = \sum_{k=0}^{\infty} \frac{(-1)^k}{(2k+1)^s} = 4^{-s}\left[\zeta\left(s,\frac{1}{4}\right) - \zeta\left(s,\frac{3}{4}\right)\right] = 2^{-s}\Phi\left(-1,s,\frac{1}{2}\right), \tag{5.53}$$

the Legendre chi function

$$\chi_s(z) = \sum_{k=0}^{\infty} \frac{z^{2k+1}}{(2k+1)^s} = 2^{-s} z \Phi\left(z^2,s,\frac{1}{2}\right). \tag{5.54}$$

and the Polylogarithm

$$\mathrm{Li}_n(z) = \sum_{k=1}^{\infty} \frac{z^k}{k^n} = z\Phi(z,n,1). \tag{5.55}$$

The Polylogarithm is generally computed by means of its functional relation with the Hurwitz zeta function

$$\mathrm{Li}_s(z) = \frac{\Gamma(1-s)}{(2\pi)^{1-s}}\left[i^{1-s}\zeta\left(1-s,\frac{1}{2}+\frac{\log(-z)}{2\pi i}\right) + i^{s-1}\zeta\left(1-s,\frac{1}{2}-\frac{\log(-z)}{2\pi i}\right)\right].$$

Additionally, some special polynomials such as the Bernoulli polynomials are also special cases of the Lerch transcendent

$$B_n(x) = \sum_{k=0}^{n} \frac{1}{k+1} \sum_{j=0}^{k} (-1)^j \binom{k}{j} (x+j)^n = -n\Phi(1,1-n,x). \tag{5.56}$$

There is a vast literature on numerical methods for the computation of the Riemann zeta function and the Hurwitz zeta function. Besides the numerical methods for the Riemann zeta function in [15], previously discussed in Chapter 2, other methods are included in [14] and [78]. A complete algorithmic description of the implementation in [54] for real and integer argument is included in `https://www.mpfr.org/algorithms.pdf`.

Regarding the Hurwitz zeta function, a recent study on the numerical evaluation using the Euler-Maclaurin formula is described in [89], other algorithms avoiding the use of Bernoulli numbers are described in [33, 164]. For uniform asymptotic expansions in the regime of parameters $s \sim a$ we refer to the usage of the vanishing saddle point explained in [157, §25.5].

5.1.3 Software

- mpmath (Python/Cython)[92]: includes the Lerch transcendent and related functions in `http://mpmath.org/doc/1.1.0/functions/zeta.html` and several special numbers in `http://mpmath.org/doc/1.1.0/functions/numtheory.html`. In particular for $\zeta(s)$, $\zeta(s,a)$ and $\Phi(z,s,a)$ we have `zeta` and `lerchphi`.
- Arb (C) [87]: provides implementations for the functions $\zeta(s)$, $\eta(s)$, $\zeta(s,a)$, $B_n(x)$, $\mathrm{Li}_s(z)$ and other Dirichlet L-functions, and special numbers B_n, E_n among others. Algorithms for the Hurwitz zeta function are described in `http://arblib.org/hurwitz.html` and for polylogarithms in `http://arblib.org/polylogarithms.html`.
- Pari/GP (C) [159]: support several L-functions (Dirichlet L-functions, Hecke L-functions and Artin L-functions) and theta functions described the documentation `https://pari.math.u-bordeaux.fr/dochtml/html-stable/_L_minusfunctions.html`. The unified function to compute several L-functions is `lfun(L, s, {D = 0})`. Transcendental functions are also directly available, for example `zeta` or `zetahurwitz`. See documentation `https://pari.math.u-bordeaux.fr/dochtml/html/Transcendental_functions.html`.
- Mathematica (C/C++) [169]: includes one of the most completed collections of transcendent functions. Some the available functions are `Zeta[s]`, `Zeta[s,a]` and `LerchPhi[z,s,a]`. Other related zeta functions can be found in `http://functions.wolfram.com/ZetaFunctionsandPolylogarithms/`, special integer numbers in `http://functions.wolfram.com/IntegerFunctions/` and generalized Bernoulli and Bell polynomials in `http://functions.wolfram.com/Polynomials/`.
- Maple (C) [109]: as other software packages, it includes most of the presented functions, for example `Zeta` and `LerchPhi` and `polylog` and special polynomials such as `bernoulli`, `euler` and `BellB`.

Other specialized software packages implementing a great deal of these functions are GAP [57], Magma [16] and some special-purpose programs such as the Dokchitser's L-function calculator `https://people.maths.bris.ac.uk/~matyd/computel/index.html`.

5.1.4 Applications

The generalized Bernoulli polynomials and their special cases arise in many applications in mathematics. They often appear in numerical methods for interpolation, differentiation and integration, for example the Euler-Maclaurin summation formula. They also occur in power series expansions and asymptotic expansions of

various classical special functions and are prevalent in the field of number theory and mathematical analysis. Bernoulli and Euler polynomials are used to express several sum of powers [43, §24.4].

The Bell number and polynomials appear in combinatorics and various asymptotic expansions. We recall the asymptotic expansion in terms of Bell polynomials for the generalized exponential integral for large order, developed in Chapter 3. The Stirling numbers play an important role in combinatorics and probability theory; for instance, they are intimately connected with Bell polynomials and the Poisson distribution. In particular, Stirling numbers of the second kind also appear in asymptotic expansions; see application for the coefficients of the Franklin-Friedman expansion in Chapter 4.

The Lerch transcendent and their special cases and other closely related functions are essential in physics applications and mathematics, playing a crucial role in fundamental conjectures (the renowned Riemann hypothesis, the Selberg conjecture and Lehmer conjecture, among others). Additionally, they serve to express several sums of reciprocal in closed-form and occur in some Taylor expansions; in particular, the Riemann zeta function with integer values appears as coefficients of some power series expansions for the gamma function,

$$\frac{1}{\Gamma(x)} = x \exp\left[\gamma x - \sum_{k=2}^{\infty} \frac{(-1)^k \zeta(k) x^k}{k}\right], \quad \log\Gamma(1+x) = -\gamma x + \sum_{k=2}^{\infty} \frac{(-1)^k \zeta(k) x^k}{k}.$$

5.2 Numerical Methods and Arbitrary-Precision Computation of the Lerch Transcendent

5.2.1 Introduction

The Lerch transcendent, also called Hurwitz-Lerch zeta function, which is named after the Czech mathematician Mathias Lerch (1860 - 1922) is defined by means of the Dirichlet series [5]

$$\Phi(z,s,a) = \sum_{k=0}^{\infty} \frac{z^k}{(k+a)^s}, \tag{5.57}$$

where $\Phi(z,s,a)$ is absolutely convergent for $|z| \leq 1,\ a \notin \mathbb{Z}_0^-$ or $\Re(s) > 1,\ |z| = 1$ and is defined elsewhere by analytic continuation. The Lerch transcendent serves as a unified framework for the study of various particular cases of special functions in number theory such as polygamma functions, polylogarithms, Dirichlet L-functions and certain number-theoretical constants. The Lerch transcendent is related to Lipschitz-Lerch zeta function by the functional equation

$$\mathcal{R}(a,x,s) = \Phi(e^{2\pi i x}, s, a).$$

This function was introduced and investigated by Lerch [104] and Lipschitz [106], where the latter studied general Euler integrals including the Lerch zeta function. Subsequently, many authors have studied properties of these functions. Among the recent investigations on the analytic properties of Lerch zeta function, we remark the work conducted by Laurinčikas and Garunkštis in [101].

The Lerch transcendent and their special cases are ubiquitous in theoretical physics. They play a relevant role in particle physics, thermodynamics and statistical mechanics, being present, for instance, in Bose-Einstein condensation distribution [71] and integrals of the Fermi-Dirac distribution. They also occur in quantum field theory, in particular in quantum electrodynamic bound state calculations [86]. Regarding mathematical applications, the Lerch zeta function can be used to evaluate Dirichlet L-series of the form

$$L(s,\chi) = \sum_{k=1}^{\infty} \frac{\chi(k)}{k^s},$$

where $\chi : (\mathbb{Z}/q\mathbb{Z})^* \to \mathbb{C}$ is a Dirichlet character and q a natural number, thus the above summation is also expressible as a combination of Hurwitz zeta functions $\zeta(s,a)$ or polygamma functions for $s \in \mathbb{N}$

$$L(s,\chi) = \sum_{r=1}^{q} \chi(r) \sum_{n=0}^{\infty} \frac{1}{(r+nq)^s} = q^{-s} \sum_{r=1}^{q} \chi(r)\zeta(s, r/q).$$

The Lerch transcendent occasionally occurs in statistics, for instance, it provides an analytic expression for the central moments of the geometric distribution.

Over the last two decades several authors have devised new series representations to extend the regime of computation of the Lerch transcendent. Complete asymptotic expansions including error bounds of $\Phi(z,s,a)$ for large a and large z are derived in [49]. More recently, an *exponentially-improved* expansion for the Lerch

zeta function in large a asymptotic was examined in [128]. A remarkable and extensive review of properties, identities and numerical methods for the computation of the Lerch transcendent and their special cases was carried out by R. Crandall in [37]. In addition, we mention two important convergent series: the Hasse's convergent series expansion in [72] given by

$$(1-z)\Phi(z,s,a) = \sum_{n=0}^{\infty}\left(\frac{-z}{1-z}\right)^n \sum_{k=0}^{n}(-1)^k\binom{n}{k}(a+k)^{-s},$$

which holds for $s,z\in\mathbb{C}$ with $\Re(z)<1/2$ and Erdélyi-series representation [48]

$$z^a\Phi(z,s,a) = \sum_{k=0}^{\infty}\zeta(s-k,a)\frac{\log^k(z)}{k!} + \Gamma(1-s)(-\log(z))^{s-1},$$

where s is not a positive integer, and for parameter $a\in(0,1]$, $|\log(z)|<2\pi$, the series representation is linearly convergent.

Finally, the Hermite-type integral representation is given by

$$\begin{aligned}\Phi(z,s,a) = \frac{1}{2a^s} + \frac{(-\log(z))^{s-1}}{z^a}\Gamma(1-s,-a\log(z)) \\ + 2\int_0^{\infty}\frac{\sin(s\arctan(t/a)-t\log(z))}{(a^2+t^2)^{s/2}(e^{2\pi t}-1)}\,dt, \quad \Re(a)>0. \end{aligned} \tag{5.58}$$

In this work, we derive complete new uniform asymptotic expansions of $\Phi(z,s,a)$ for large order of the parameters a, s and argument z, with special emphasis on the less investigated case $\Re(z)\gg 0$. The starting point for our asymptotic expansions is the integral representation in (5.58). Additionally, a careful treatment of the Euler-Maclaurin formula is considered along with the calculation of a rigorous error bound. A significant effort have been made to develop uniform asymptotic expansions with tractable coefficients in terms of known entities and amenable to arbitrary-precision computations. An extensive discussion on algorithmic aspects for their successful implementation is also provided.

The outline of the work is the following: in Section 2 we study the main numerical methods considered for the numerical evaluation of $\Phi(z,s,a)$, including error bounds. Then, in Section 3, we discuss in detail implementation aspects, several heuristics and performance issues. We also devise an effective algorithm that permits computation to arbitrary-precision in an extensive region of the function's domain. In Section 4, we provide numerical calculations and compare the present implementation with open source and commercial state-of-the-art libraries. Finally, in Section 5, we discuss possible enhancements and present our conclusions.

5.2.2 Numerical methods

Euler-Maclaurin formula

We briefly summarized the Euler-Maclaurin formula and refer to [34] for a formal proof. We closely follow the expository style in [89]. Let us suppose that f is an analytic function on a closed domain $[N,U]$ where $N,U\in\mathbb{Z}$, and let M be a positive integer. Let B_n denote the n-th Bernoulli number and $\tilde{B}_{2M}(t) = B_n(t-\lfloor t\rfloor)$ denote

the n-th periodic Bernoulli polynomials. The Euler-Maclaurin summation formula states that

$$\sum_{k=N}^{U} f(k) = I + T + R \tag{5.59}$$

where

$$I = \int_N^U f(t)\,dt \tag{5.60}$$

$$T = \frac{1}{2}(f(N) + f(U)) + \sum_{k=1}^{M} \frac{B_{2k}}{(2k)!}\left(f^{(2k-1)}(U) - f^{(2k-1)}(N)\right) \tag{5.61}$$

$$R = -\int_N^U \frac{\tilde{B}_{2M}(t)}{(2M)!} f^{(2M)}(t)\,dt\,. \tag{5.62}$$

If f decreases sufficiently rapid, letting $U \to \infty$ the above equations remain valid.

Proposition 5.2.1 *The Euler-Maclaurin summation formula for the Lerch transcendent is given by*

$$\Phi(z, s, a) = S + I + T + R, \tag{5.63}$$

where

$$S = \sum_{k=0}^{N-1} \frac{z^k}{(k+a)^s}, \tag{5.64}$$

$$I = \frac{(-\log(z))^{s-1}}{z^a}\Gamma(1-s, -(a+N)\log(z)), \tag{5.65}$$

$$T = \frac{z^N}{(a+N)^s}\left(\frac{1}{2} + \sum_{k=1}^{M} \frac{B_{2k}}{(2k)!}\frac{U(-2k+1, -2k+2-s, -(a+N)\log(z))}{(a+N)^{2k-1}}\right), \tag{5.66}$$

$$R = -\int_N^\infty \frac{\tilde{B}_{2M}(t)}{(2M)!}\frac{z^t}{(a+t)^{s+2M}} U(-2M, -2M+1-s, -(a+t)\log(z))\,dt, \tag{5.67}$$

for $a, s, z \in \mathbb{C}$ with $|\log(z)| < 2\pi$ and $N, M \in \mathbb{N}$ such that $\Re(a) + N > 0$ and $\Re(s) + 2M > 1$.

Proof: Let us first consider the Hermite-type integral in (5.58)

$$I := \int_0^\infty \frac{\sin(s\arctan(t/a) - t\log(z))}{(a^2+t^2)^{s/2}(e^{2\pi t} - 1)}\,dt\,. \tag{5.68}$$

For $z, s, a \in \mathbb{R}$, $z > 0$ and $a > 0$, the above integral can be written in the form

$$I = \frac{1}{a^s}\Im\left(\int_0^\infty \frac{z^{-it}}{(1-it/a)^s}\frac{dt}{e^{2\pi t} - 1}\right). \tag{5.69}$$

The domain delimited by previous constraints shall be extended by analytic continuation. Now we express the integrand in (5.69) in terms of the confluent hypergeometric function $U(a,b,z)$ which yields

$$I = \frac{1}{a^s}\Im\left((-a\log(z))^s\int_0^\infty \frac{e^{-i\log(t)}U(s,s+1,(it-a)\log(z))}{e^{2\pi t}-1}\,dt\right), \tag{5.70}$$

By applying the addition theorem for $U(a,b,z)$ [43, §13.13] given by

$$U(a,b,x+y) = e^y\sum_{n=0}^\infty \frac{(-y)^n}{n!}U(a,b+n,x), \quad |y|<|x|. \tag{5.71}$$

the integrand can be written as a summation defined by

$$e^{-i\log(t)}U(s,s+1,(it-a)\log(z)) = \sum_{k=0}^\infty \frac{(-it\log(z))^k U(s,s+k+1,-a\log(z))}{k!}. \tag{5.72}$$

Substituting (5.72) into (5.70) and formally interchanging summation and integration we obtain

$$I = \frac{1}{a^s}\Im\left((-a\log(z))^s\sum_{k=0}^\infty \frac{(-i\log(z))^k U(s,s+k+1,-a\log(z))}{k!}\int_0^\infty \frac{t^k}{e^{2\pi t}-1}\,dt\right),$$

where the integral can be directly evaluated in closed form by

$$\int_0^\infty \frac{t^k}{e^{2\pi t}-1}\,dt = \frac{k!}{(2\pi)^{k+1}}\zeta(k+1).$$

We use Kummer's transformation $U(s,s+k+1,-a\log(z))(-a\log(z))^{k+s} = U(-k,1-k-s,-a\log(z))$ to rewrite I in the form

$$I = \frac{1}{a^s}\Im\left(\sum_{k=1}^\infty \frac{i^k}{a^k}\frac{U(-k,1-k-s,-a\log(z))}{(2\pi)^{k+1}}\zeta(k+1)\right). \tag{5.73}$$

Note that the same summation formula can be derived by expanding $f(t) = z^{-it}(1-it/a)^{-s}$, which gives

$$f(t) = \sum_{k=0}^\infty \left(\frac{-it}{a}\right)^k \frac{1}{k!}\sum_{j=0}^\infty \binom{k}{j}(-1)^j(a\log(z))^{k-j}\sum_{m=0}^j (-1)^{j-m}s(j,m)s^m, \tag{5.74}$$

where $s(j,m)$ are Stirling numbers of the first kind. The inner summation in (5.74) is expressible in terms of rising factorial or Pochhammer's symbol $(s)_j$ using the well-known identities

$$\sum_{m=0}^j (-1)^{j-m}s(j,m)s^m = (-1)^j(-s)^{(j)} = (s)_j \tag{5.75}$$

and

$$\sum_{j=0}^{\infty}\binom{k}{j}(a\log(z))^{k-j}(-s)^{(j)} = (-1)^k U(-k, 1-k-s, -a\log(z)). \tag{5.76}$$

Finally, taking the imaginary part of (5.73) yields

$$\begin{aligned} I &= \frac{1}{a^s}\sum_{k=0}^{\infty}\frac{(-1)^k}{a^{2k+1}}\frac{U(-2k-1,-2k-s,-a\log(z))}{(2\pi)^{2k+2}}\zeta(2k+2) \\ &= \frac{1}{2a^s}\sum_{k=1}^{\infty}\frac{B_{2k}}{(2k)!}\frac{U(-2k+1,-2k+2-s,-a\log(z))}{a^{2k-1}}, \end{aligned} \tag{5.77}$$

where the relationship between Bernoulli numbers B_{2k} and the Riemann zeta function is applied. □

Note that the expansion is convergent for $|\log(z)| < 2\pi$. This can be observed by taking the asymptotic estimate of the k-th term in (5.77)

$$\begin{aligned} |t_k| &= \left|\frac{B_{2k}}{(2k)!}\frac{U(-2k+1,-2k+2-s,-(a+N)\log(z))}{(a+N)^{2k-1}}\right| \\ &\sim \frac{2}{(2\pi)^{2k}}|\log(z)|^{2k-1}, \end{aligned} \tag{5.78}$$

as $N, M \to \infty$, where we consider the usual asymptotic estimates for $U(a,b,z) \sim z^{-a}$ as $|z| \to \infty$ and $|B_{2k}|/(2k)!$ using the fact that $\zeta(2k) \sim 1$ as $k \to \infty$. To assess the domain of convergence for z we use the ratio test (d'Alembert ratio test)

$$\lim_{k\to\infty}\left|\frac{t_{k+1}}{t_k}\right| \sim \frac{|\log(z)^2|}{4\pi^2} < 1 \iff |\log(z)| < 2\pi.$$

Finally, taking M such that $\Re(s) + 2M - 1 > 0$, the remainder term (5.67) in the Euler-Maclaurin summation formula is well defined, giving its analytic continuation to $s \in \mathbb{C} \setminus \{1\}$.

Theorem 5.2.2 *Given $a, s, z \in \mathbb{C}$ with $|\log(z)| < 2\pi$ and $N, M \in \mathbb{N}$ such that $\Re(a) + N > 0$ and $\Re(s) + 2M > 1$, the error term (5.67) in the Euler-Maclaurin summation formula satisfies*

$$|R| \leq \frac{4}{(2\pi)^{2M}}\left|C\sum_{k=0}^{2M}\binom{2M}{k}Q(k+1-2M-s,W)\frac{(-\log(z))^{-k-1+2M+s}}{\log^{-k}(z)}\right|, \tag{5.79}$$

where $C = \Gamma(1-s)/z^a$, $W = -(a+N)\log(z)$ and $Q(a,z) = \Gamma(a,z)/\Gamma(a)$ is the regularized incomplete Gamma function.

Proof: We have

$$\begin{aligned}|R| &= \left|\int_N^\infty \frac{\tilde{B}_{2M}(t)}{(2M)!}\frac{z^t}{(a+t)^{s+2M}}U(-2M,-2M+1-s,-(a+t)\log(z))\,dt\right| \\ &\leq \frac{\left|\tilde{B}_{2M}(t)\right|}{(2M)!}\left|\int_N^\infty \frac{z^t}{(a+t)^{s+2M}}U(-2M,-2M+1-s,-(a+t)\log(z))\,dt\right| \\ &\leq \frac{4}{(2\pi)^{2M}}\left|\sum_{k=0}^{2M}\binom{2M}{k}B_k^M(s)\int_N^\infty \frac{z^t}{(a+t)^{s+2M}}((a+t)\log(z))^k\,dt\right|\end{aligned}$$

with $B_k^M(s) = (k+1-2M-s)_{2M-k}$. We apply the usual upper bound for $|\tilde{B}_n(t)| < 4n!/(2\pi)^n$ and formally interchange integration and the expansion of $U(-2M,-2M+1-s,-(a+N)\log(z))$ given by

$$U(-2M,-2M+1-s,-(a+N)\log(z)) = \sum_{k=0}^{2M}\binom{2M}{k}B_k^M(s)((a+t)\log(z))^k.$$

The integral above can be expressed in terms of the incomplete Gamma function $\Gamma(a,z)$ as follows (similar to (5.65))

$$\begin{aligned}\int_N^\infty \frac{z^t}{(a+t)^{s+2M}}((a+t)\log(z))^k\,dt = \frac{\log^k(z)}{z^a}(-\log(z))^{-k-1+2M+s} \\ \times\,\Gamma(k+1-2M-s,-(a+N)\log(z)) \quad (5.80)\end{aligned}$$

Thus, we have

$$\begin{aligned}&\sum_{k=0}^{2M}\binom{2M}{k}B_k^M(s)\int_N^\infty \frac{z^t}{(a+t)^{s+2M}}((a+t)\log(z))^k\,dt \\ &= \frac{\Gamma(1-s)}{z^a}\sum_{k=0}^{2M}\binom{2M}{k}Q(k+1-2M-s,-(a+N)\log(z))\frac{(-\log(z))^{-k-1+2M+s}}{\log^{-k}(z)},\end{aligned}$$

where we use $B_k^M(s) = \Gamma(1-s)/\Gamma(k+1-2M-s)$. Note that for $s \in \mathbb{N}$ we take equation (5.80) to avoid the pole. □

The bound given in Theorem 5.2.2 give us a notably tight approximation of remainder (5.67). However, for large M the direct evaluation of the terminating series in (5.79) might be substantially expensive, being a not negligible part of the total computation time, therefore approximations for large order will be considered in Section 5.2.3.

Uniform asymptotic expansion for $\Phi(z,s,a)$

A suitable Laplace-type integral representation of $\Phi(z,s,a)$ amenable to derive multiple asymptotic expansions [37], is given by

$$\Phi(z,s,a) = \frac{1}{\Gamma(s)}\int_0^\infty \frac{t^{s-1}e^{-at}}{1-ze^{-t}}\,dt, \quad \Re(s)>0,\ \Re(a)>0,\ z\notin[1,\infty), \qquad (5.81)$$

which serves to define the analytic continuation of the Lerch-series to $z \in \mathbb{C} \setminus [1, \infty)$. This integral has been chosen as starting point to derive asymptotic expansions for either large or small a (assuming that s and z are fixed) or for large z in [49], and for a Bernoulli-series representation as in [37]. The aim of this subsection is to extend the domain of computation of the Poincaré type asymptotic expansion for large a defined in [49] by constructing a uniform asymptotic expansion for large a, s and z.

We proceed to construct that expansion by using the vanishing saddle point method described in [157]. This method is fundamentally a modification of Laplace's method applicable to integrals of the form

$$F_\lambda(z) = \frac{1}{\Gamma(\lambda)} \int_0^\infty t^{\lambda-1} e^{-zt} f(t)\, dt, \tag{5.82}$$

with $\Re(\lambda) > 0$ and z large, in which λ might also be large. The resulting expansion is given by

$$F_\lambda(z) \sim \sum_{k=0}^{\infty} \frac{a_k(\mu) P_k(\lambda)}{z^{k+\lambda}},$$

where $a_k(\mu)$ are the coefficients of the expansion of $f(t)$ at the saddle point $\mu = \lambda/z$ and coefficients $P_k(\lambda)$ are expressible in terms of generalized Laguerre polynomials defined by

$$P_k(\lambda) = k! L_k^{-k-\lambda}(-\lambda). \tag{5.83}$$

At this point, we briefly recall the definition of the Eulerian polynomial and its connection with the polylogarithm function before stating the next proposition.

The Eulerian polynomial is defined as

$$A_k(z) = \sum_{j=0}^{k-1} \left\langle {k \atop j} \right\rangle z^j, \tag{5.84}$$

where $\left\langle {k \atop j} \right\rangle$ are the Eulerian numbers [64]. The Eulerian polynomials satisfy the recurrence equation

$$A_0(z) = 1, \qquad A_k(z) = \sum_{j=0}^{k-1} \binom{k}{j} A_j(z)(z-1)^{k-1-j}, \quad k \geq 1. \tag{5.85}$$

The Eulerian polynomial and polylogarithm are related by the functional equation

$$A_k(z) = \frac{(1-z)^{k+1}}{z} \mathrm{Li}_{-k}(z) = \frac{(1-z)^{k+1}}{z} \sum_{j=1}^{\infty} j^k z^j, \quad |z| < 1, \tag{5.86}$$

and if $|z| > 1$ then $A_k(z) = (-1)^{k+1} A_k\left(\frac{1}{z}\right)$.

Proposition 5.2.3 *For $a, s, z \in \mathbb{C}$, $\Re(a) > 0$ and $z \notin [1, \infty)$ we have the following uniform asymptotic expansion for $\Phi(z, s, a)$*

$$\Phi(z,s,a) = \frac{e^\mu}{e^\mu - z} \left(\frac{1}{a^s} + \sum_{k=2}^{K-1} (-1)^k \frac{P_k(s)}{k! a^{k+s}} r^k A_k\left(\frac{e^\mu}{z}\right) \right) + \varepsilon_K(z,s,a), \tag{5.87}$$

where $r = z/(e^{\mu} - z)$.

Proof: We take $f(t) = (1 - ze^{-t})^{-1}$ and $\mu = s/a$ in (5.82), where μ is the saddle point of the dominant part of the integral. Following closely the derivation in [49], we expand $f(t)$ at $t = \mu$ to obtain the Taylor expansion

$$f(t) = \sum_{k=0}^{\infty} a_k(\mu)(t-\mu)^k, \quad a_k(\mu) = (-1)^k \frac{e^{\mu} z^k}{k!(e^{\mu} - z)^{k+1}} \sum_{j=0}^{k-1} \left\langle {k \atop j} \right\rangle \left(\frac{e^{\mu}}{z}\right)^j. \tag{5.88}$$

After performing a few algebraic manipulations we obtain the final representation for the vanishing point expansion for $\Phi(z, s, a)$. □

We can clearly observe that for large values of $\Re(s)$ and $|z|$, the asymptotic convergence of the expansion improves. Furthermore, from a numerical perspective, moderate to large values of $\Re(s) < 0$ permit the evaluation of $A_k(e^{\mu}/z)$ via the convergent series (5.86).

It remains to bound the error term in the expansion after truncation at $k = K - 1$. Let us consider the k-th term of expansion (5.87) defined as

$$|t_k| = \left| \frac{P_k(s)}{k! a^{k+s}} \left(\frac{z}{e^{\mu} - z}\right)^k A_k\left(\frac{e^{\mu}}{z}\right) \right| \leq \left| \frac{1}{k! a^{k+s}} \left(\frac{z}{e^{\mu} - z}\right)^k \right| |P_k(s)| \left| A_k\left(\frac{e^{\mu}}{z}\right) \right|.$$

A bound for the error term by comparison with a geometric series yields

$$\left| \sum_{k=K}^{\infty} t_k \right| \leq \frac{|t_K|}{1 - C}, \quad C = \left| \frac{t_{K+1}}{t_K} \right|, \tag{5.89}$$

iff $C < 1$, where t_K is the first omitted term in the expansion and

$$|\varepsilon_K(z, s, a)| \leq \left| \frac{e^{\mu}}{e^{\mu} - z} \right| \frac{|t_K|}{1 - C}.$$

In order to provide an effective upper bound for $|t_k|$, we compute two saddle point bounds for polynomials $P_k(z)$ and $A_k(z)$.

Proposition 5.2.4 *For* $k > 1$ *and* $z \in \mathbb{C} \setminus \{1\}$ *the Eulerian polynomials satisfy the following bound*

$$|A_k(z)| \leq k! \left| (z-1) e^{\phi(t_0)} \right|, \tag{5.90}$$

where

$$t_0 = \frac{W(e^k k z) - k}{z - 1} \quad \textit{and} \quad \phi(t_0) = -k \log t_0 - \log\left(z - e^{(z-1)t_0}\right),$$

and $W(x)$ *is the Lambert-W function which solves* $W(x)e^{W(x)} = x$.

Proof: An integral representation for the Eulerian polynomials is obtained after applying Cauchy's integral formula to the exponential generating function given by

$$\sum_{k=0}^{\infty} A_k(z) \frac{t^k}{k!} = \frac{z-1}{z - e^{(z-1)t}} \Longrightarrow A_k(z) = \frac{k!(z-1)}{2\pi i} \oint \frac{t^{-k-1}}{z - e^{(z-1)t}} \, dt,$$

which can be written in the form

$$A_k(z) = \frac{k!(z-1)}{2\pi i} \oint \frac{e^{\phi(t)}}{t}\, dt, \quad \phi(t) = -k\log t - \log\left(z - e^{(z-1)t}\right).$$

We compute the saddle point of the integrand by solving the following equation

$$\phi'(t) = \frac{(z-1)e^{(z-1)t}}{z - e^{(z-1)t}} - \frac{k}{t}, \quad t_0 = \frac{W(e^k kz) - k}{z-1}. \tag{5.91}$$

The principal contribution of the saddle point bound is obtained by substituting t_0 into the integrand

$$|A_k(z)| \leq \frac{k!}{2\pi i}\left|(z-1)e^{\phi(t_0)}\right| \oint \frac{dt}{t} = k!\left|(z-1)e^{\phi(t_0)}\right|.$$

Finally, by the residue theorem we obtain the result. □

A similar analysis is carried out for polynomials $P_k(z)$. The use of the generating function for generalized Laguerre polynomials gives the Cauchy-type integral representation

$$P_k(s) = k!L_k^{-k-s}(-s) = \frac{k!}{2\pi i}\int_{\mathcal{C}} (1-t)^{k+s-1} e^{ts/(1-t)} \frac{dt}{t^{k+1}}, \tag{5.92}$$

where $\mathcal{C}$ is a circle around the origin with a radius less than unity.

Proposition 5.2.5 *For $k > 1$ and $s \in \mathbb{C} \setminus \{0, 1\}$ the polynomials $P_k(s)$ satisfy the following bound*

$$|P_k(s)| \leq k!\left|e^{\phi(t_0)}\right|, \tag{5.93}$$

where

$$t_0 = \frac{\sqrt{k^2 + 4ks - 2k + 1} + k + 1}{2(1-s)}$$

and

$$\phi(t_0) = s\frac{t_0}{1-t_0} + (k+s-1)\log(1-t_0) - k\log t_0.$$

Proof: A proof follows the steps presented previously. □

Combination of both bounds (5.90) and (5.93) gives the final form for the error bound. The selection of the appropriate truncation point K to achieve a desired level of precision is detailed in Section 5.2.3.

Asymptotic expansion for large z

A careful reader shall have noticed that none of the previous series expansions are suitable for arbitrarily large $\Re(z) > 0$. The expansion in this subsection complements the asymptotic expansion described in [49] for $z \in \mathbb{C} \setminus [0, \infty)$, $\Re(a) > 0$ and $\Re(s) > 0$ for large z and fixed a and s

Theorem 5.2.6 *For $a, s, z \in \mathbb{C}$ and $\Re(a) > 0$ we have an asymptotic expansion for large a and z, and fixed s given by*

$$\begin{aligned}\Phi(z,s,a) &\sim \frac{1}{2a^s} + \frac{(-\log(z))^{s-1}}{z^a}\Gamma(1-s,-a\log(z)) \\ &+ \frac{1}{2a^s}\left(\frac{2}{\log(z)} - \coth\left(\frac{\log(z)}{2}\right)\right) \\ &+ \frac{1}{a^s}\sum_{k=1}^{\infty}\frac{(s)_k}{a^k(2\pi)^{k+1}}\left(\frac{1}{u^{k+1}} - \frac{\pi^{k+1}}{k!}\coth(\pi u)^{k-1}\operatorname{csch}(\pi u)^2\mathcal{P}_k(\operatorname{sech}(\pi u)^2)\right),\end{aligned} \tag{5.94}$$

where $u = \frac{\log(z)}{2\pi}$ and $\mathcal{P}_k(x)$ are peak polynomials [149].

Proof: We start from the integral representation (5.69). Application of the binomial theorem yields

$$I = \Im\left(\int_0^\infty \frac{z^{-it}}{(1-it/a)^s}\frac{dt}{e^{2\pi t}-1}\right) = \Im\left(\sum_{k=0}^{\infty}\frac{(s)_k}{k!}\left(\frac{i}{a}\right)^k\int_0^\infty \frac{z^{-it}t^k}{e^{2\pi t}-1}\,dt\right).$$

Let us focus on the inner integral I_k defined as

$$I_k = \int_0^\infty \frac{z^{-it}t^k}{e^{2\pi t}-1}\,dt = \int_0^\infty \frac{z^{-it}t^k(1-e^{-t})}{(1-e^{-t})(e^{2\pi t}-1)}\,dt.$$

Noting that $(1-e^{-t})/(e^{2\pi t}-1) = \frac{1}{2}(\coth(\pi t)-1)(\sinh(t)-\cosh(t)+1)$, we split the integral obtaining a closed form in terms of the Hurwitz zeta function

$$I_k = k!\left(\frac{\zeta(k+1, i\log(z)/(2\pi))}{(2\pi)^{k+1}} - \frac{1}{(i\log(z))^{k+1}}\right) = \frac{k!}{(2\pi)^{k+1}}\zeta\left(k+1, 1+i\frac{\log(z)}{2\pi}\right).$$

For $k=0$, $\zeta\left(k+1, 1+i\frac{\log(z)}{2\pi}\right)$ has a pole, so we proceed as follows

$$\Im\left(\int_0^\infty \frac{z^{-it}}{e^{2\pi t}-1}\,dt\right) = -\int_0^\infty \frac{\sin(\log(z)t)}{e^{2\pi t}-1}\,dt = \frac{1}{4}\left(\frac{2}{\log(z)} - \coth\left(\frac{\log(z)}{2}\right)\right).$$

Combining terms give us the asymptotic expansion for integral (5.69)

$$I \sim \frac{1}{4}\left(\frac{2}{\log(z)} - \coth\left(\frac{\log(z)}{2}\right)\right) + \sum_{k=1}^{\infty}\frac{(s)_k}{a^k(2\pi)^{k+1}}\Im\left(i^k\zeta\left(k+1, 1+i\frac{\log(z)}{2\pi}\right)\right).$$

Hereinafter we use $u = \frac{\log(z)}{2\pi}$ to simplify notation. Let us define the terms $C_k(u)$ as

$$C_k(u) = \Im\left(i^k\zeta\left(k+1, 1+iu\right)\right) = \frac{i^{k+1}}{2}\left((-1)^k\zeta\left(k+1, 1-iu\right) - \zeta\left(k+1, 1+iu\right)\right),$$

where we remove the imaginary part. In order to eliminate the computations on the complex plane for real z, we expand[1] $C_k(u)$ reducing compound arguments.

[1] We employ `FunctionExpand` in Mathematica [169].

The first five coefficients $c_k(u) = 2C_k(u)$ are

$$c_1(u) = \frac{1}{u^2} - \pi^2 \operatorname{csch}(\pi u)^2,$$
$$c_2(u) = \frac{1}{u^3} - \pi^3 \coth(\pi u) \operatorname{csch}(\pi u)^2,$$
$$c_3(u) = \frac{1}{u^4} - \frac{\pi^4}{6}\left(4\coth(\pi u)^2 \operatorname{csch}(\pi u)^2 + 2\operatorname{csch}(\pi u)^4\right),$$
$$c_4(u) = \frac{1}{u^5} - \frac{\pi^5}{24}\left(8\coth(\pi u)^3 \operatorname{csch}(\pi u)^2 + 16\coth(\pi u) \operatorname{csch}(\pi u)^4\right),$$
$$c_5(u) = \frac{1}{u^6} - \frac{\pi^6}{120}\left(16\coth(\pi u)^4 \operatorname{csch}(\pi u)^2 + 88\coth(\pi u)^2 \operatorname{csch}(\pi u)^4 + 16\operatorname{csch}(\pi u)^4\right).$$

From the observation of previous coefficients, we state the following identity, which proof follows by induction

$$\begin{aligned} C_k(u) &= \frac{1}{2u^{k+1}} - \frac{\pi^{k+1}}{2k!} \sum_{j=0}^{\lceil k/2\rceil - 1} P(k,j) \coth(\pi u)^{k-1-2j} \operatorname{csch}(\pi u)^{2(j+1)} \\ &= \frac{1}{2u^{k+1}} - \frac{\pi^{k+1}}{2k!} \coth(\pi u)^{k-1} \operatorname{csch}(\pi u)^2 \sum_{j=0}^{\lceil k/2\rceil - 1} P(k,j) \operatorname{sech}(\pi u)^{2j} \\ &= \frac{1}{2u^{k+1}} - \frac{\pi^{k+1}}{2k!} \coth(\pi u)^{k-1} \operatorname{csch}(\pi u)^2 \mathcal{P}_k(\operatorname{sech}(\pi u)^2). \end{aligned}$$

where $P(k,j)$ denotes the number of permutations of k numbers with j peaks, also known as *peak number* or *pk-number*, and $\mathcal{P}_k(x)$ a *pk-polynomial*. □

Peak numbers $P(k,j)$ give the sequence A008303 of the OEIS [144]. For $k \geq 1$ and $0 \leq j \leq k$, we have a functional recursion generating a triangular array

$$P(k,j) = 2(j+1)P(k-1,j) + (k-2j)P(k-1,j-1). \tag{5.95}$$

Note that $P(k,j) = 0$ for $j \geq k/2$ and therefore $\deg P_k(x) = \lceil k/2 \rceil - 1$. Peak polynomials are given by the generating function for peak numbers $P(k,j)$.

$$\mathcal{P}_k(x) = \sum_{j=0}^{\lceil k/2\rceil - 1} P(k,j) x^j.$$

Note that for values of $|x| \to 1$ we can estimate its magnitude by the finite sum of peak numbers, since $\sum_{j=0}^{\lceil k/2\rceil - 1} P(k,j) = k!$, hence

$$|\mathcal{P}_k(x)| \sim k!, \quad |x| \to 1. \tag{5.96}$$

The bivariate exponential generating function can be defined as in [51], [172]

$$\begin{aligned}\sum_{k=0}^{\infty}\mathcal{P}_k(p)\frac{x^k}{k!} &= 1-\frac{1}{p}+\frac{\sqrt{p-1}}{p}\tan\left(x\sqrt{p-1}+\arctan(1/\sqrt{p-1})\right)\\ &= \frac{\sqrt{1-p}\cosh(x\sqrt{1-p})}{\sqrt{1-p}\cosh(x\sqrt{1-p})-\sinh(x\sqrt{1-p})}\\ &= \frac{1}{\sqrt{1-p}\coth(x\sqrt{1-p})-1}\end{aligned}$$

As customary in analytic combinatorics, application of Cauchy's integral formula to the bivariate exponential generating function gives

$$\mathcal{P}_k(p)=\frac{k!}{2\pi}\int_0^{2\pi}\frac{e^{-ikt}}{\sqrt{1-p}\coth(e^{it}\sqrt{1-p})-1}\,dt\,.$$

A remarkable result from the theory of enriched P-partitions is the functional relation between peak polynomials and Eulerian polynomials stated in [149]

$$\mathcal{P}_k\left(\frac{4x}{(1+x)^2}\right)=\frac{2^{k-1}}{(1+x)^{k-1}}A_k(x), \tag{5.97}$$

which allows us to use the upper bound in (5.90) to estimate the truncation point in (5.94). Furthermore, a good asymptotic estimate of $P_k(x)$ for large order k can be derived from a Mittag-Leffler type decomposition of Eulerian polynomials [36]:

$$A_k(z)=C(k,z)\left(\frac{1}{\log(z)^{k+1}}+\sum_{j=1}^{\infty}\frac{1}{(\log(z)+2\pi ij)^{k+1}}+\frac{1}{(\log(z)-2\pi jk)^{k+1}}\right),$$

where

$$C(k,z)=\frac{e^{\pi i(k-1)}(1-z)^{k+1}k!}{z}.$$

Taking the prefactor of the expansion and applying the functional relation (5.97) we have

$$\mathcal{P}_k(x)\sim\frac{2^{k-1}k!e^{\pi i(k-1)}(1-u)^{k+1}}{(1+u)^{k-1}u\log(u)^{k+1}},\quad u=\frac{2-x-2\sqrt{1-x}}{x},\quad k\to\infty.$$

5.2.3 Algorithmic details and implementation

In this section we discuss in detail the implementation aspects and several proposed heuristics easy to evaluate while being effective in practice. All algorithms described are implemented in Python[2] using the mpmath library for arbitray-precision floating-point arithmetic [92] with GMPY2, which supports integer and rational arithmetic via the GMP library [69] and real and complex arithmetic by the MPFR [54] and MPC [47] libraries.

[2] https://sites.google.com/site/guillermonavaspalencia/software/lerch.py

As it is well-known, numerical evaluation of special functions requires the use of several methods of computation to cover the whole regime of the parameters. We aim to sketch the building blocks of a basic algorithm, which have been tested to work reasonable well for most cases, but we do not dare to claim that it will cover the whole function's domain optimally. For those cases either not covered by current series expansions or prone to numerical instability, we select numerical complex integration, which serves as a backup method.

Evaluation of L-series

The L-series of the form (5.57) are in general difficult to accelerate due to the non recursive scheme of computation. In order to employ common acceleration techniques such as parallelization, the determination of the optimal truncation level is crucial. As described in the previous section, a bound for the remainder term of the L-series can be constructed as follows

$$\left|\sum_{k=K}^{\infty} \frac{z^k}{(k+a)^s}\right| \leq \frac{|z|^K}{|(K+a)^s|(1-C_K(z,s,a))} \leq \frac{|z|^K}{|(K+a)^s|(1-|z|)}, \tag{5.98}$$

where

$$C_K(z,s,a) = \left|\frac{z(K+a)^s}{(K+1+a)^s}\right|, \quad \lim_{K\to\infty} C_K(z,s,a) = |z|. \tag{5.99}$$

The required number of terms K to obtain a result with P-bit precision can be obtained by performing a simple linear search, which is generally sufficient to target an absolute error of about 2^{-P}. However, a more efficient approximation of K is yielded by solving the following equation with the first omitted term, $z^K(K+a)^{-s} = 2^{-P}$ for K. The first solution in closed-form is given by

$$K = -\frac{sW(\phi(z,s,a,P)) + a\log(z)}{\log(z)}, \quad \phi(z,s,a,P) = -\frac{(2^{-P}z^a)^{-1/s}\log(z)}{s}.$$

We distinguish two different approximations for K, denoted as $\hat{K}$, depending on $\Re(s)$. For $\Re(s) > 0$

$$\hat{K} = \left\lceil \left| \frac{\Re(s)W_0(\phi_1(z,s,a,P)) + |a|\log(|z|)}{\log(|z|)} \right| \right\rceil,$$

where $[x]$ denotes the nearest integer function and

$$\phi_1(z,s,a,P) = -\frac{(2^{-P}|a|^{-\Re(s)-1}|z|^{|a|})^{-1/(\Re(s)+1)}\log(|z|)}{\Re(s)+1}.$$

For $\Re(s) < 0$

$$\hat{K} = \left\lceil \left| \frac{\Re(s)W_{-1}(\phi_2(z,s,a,P)) + |a|\log(|z|)}{\log(|z|)} \right| \right\rceil,$$

where

$$\phi_2(z,s,a,P) = -\frac{(2^{-P}|a|^{-\Re(s)}|z|^{|a|})^{-1/\Re(s)}\log(|z|)}{\Re(s)}.$$

Given that ϕ_1, $\phi_2 \in \mathbb{R}$, we use the principal branch $W_0(z) = W(z)$ when $\Re(s) > 0$, since $\phi_1 > 0$ and the branch $W_{-1}(z)$ for $\Re(s) < 0$, since $\phi_2 \in (-1/e, 0)$. Remember that $W(x)$ is two-valued for $-1/e \leq x < 0$. Numerical tests suggest that these approximations for choosing K are sufficient to obtain good estimates of the required number of terms. We note that $\hat{K}$ can be computed using 53-bit machine floating-point arithmetic.

Several heuristics are implemented to compensate catastrophic cancellation for cases when $\Re(z) < 0$ and/or $\Re(s) < 0$. In particular, for $\Re(s) < 0$ we increase the working precision $P^W = P + \lfloor P/3 \rfloor + [-\Re(s)]$. On the other hand, for the case $\Re(s) > 1$ and $z \in \mathbb{R}_{<0}$ we employ the linear acceleration methods for alternating series described in [35]. This method is used when $\hat{K} > 1.2[1.31D]$, where D is the precision digits. For all other cases, we add up to 20 guard bits to the working precision.

The computation of the L-series is particularly simple to parallelize by assigning a block of size $k : k \leq N$ to each thread. This parallelization scheme is implemented using the `multiprocessing` module in Python. Based on experiments, parallelization provides a significant speedup factor for $\hat{K} > 1024$ or $P \geq 1024$ bits.

We remark that L-series converges rather slowly when $|z| \to 1$. It is possible to employ convergence acceleration techniques to obtain an efficient evaluation of the Lerch transcendent; see the application of combined nonlinear-condensation transformation in [85]. Alternatively, the Euler-Maclaurin formula is also convenient for those cases, as shown later.

Evaluation of the Euler-Maclaurin error bound

For a precision of D digits, we choose $N = \lfloor D/3 \rfloor$. For large $\Re(a) > 0$ we choose $N = 0$ if the following condition is satisfied

$$\Re(a) > |\Re(s)| + |\Re(z)| + D.$$

The number of terms M can be effectively approximated by solving $t_k = 2^{-P}$, where P is the precision in bits and t_k is the asymptotic estimate in (5.78), which yields

$$M \sim \left\lceil \frac{1}{2} \left| \frac{\log(2^{-P-1}\log(z))}{\log(2\pi) - \log(\log(z))} \right| \right\rceil.$$

This is a near-optimal approximation at high-precision. In practice, the asymptotic estimate of M is used for $P \geq 500$, otherwise we use the heuristic $M = N + \lfloor P/3 \rfloor$. There is a unavoidable trade-off when choosing N and M, large values results in catastrophic cancellation since the L-series might be unstable, especially for $z \in \mathbb{C} \setminus \mathbb{R}$, but reduces the number of terms M, therefore the time spent computing Bernoulli numbers, which represents a significant amount of the total computation time.

We can evaluate the coefficient in the error bound (5.79) using a recurrence. Computation of $Q(k+1-2M-s, -(a+N)\log(z))$ only requires the initial value $Q(1-s, -(a+N)\log(z))$, which can be computed re-using $\Gamma(1-s, -(a+N)\log(z))$

in (5.66). Subsequent terms can be computed at lower precision via the recurrence

$$Q(a,z) = Q(a-1,z) + \frac{e^{-z}z^{a-1}}{\Gamma(a)}, \quad a \in \mathbb{C} \setminus \mathbb{Z}^-,$$

or

$$\Gamma(a,z) = (a-1)\Gamma(a-1,z) + e^{-z}z^{a-1}, \quad a \in \mathbb{C}.$$

A recurrence for the rest of terms in the coefficients is trivial. The direct evaluation of the recurrence requires $\mathcal{O}(M)$ arithmetic operations, therefore the associated computational cost is not negligible for very large M, as previously mentioned.

For large M we might use asymptotic estimates to reduce the complexity, for example the case $M > |(a+N)\log(z)|$

$$\begin{aligned} |R| &\sim \frac{4}{(2\pi)^{2M}} \left| (-2M+1-s)_{2M} E_{2M+s}(-(a+N)\log(z)) \frac{(a+N)^{-2M-s+1}}{z^{a+N}} \right| \\ &\sim \frac{4}{(2\pi)^{2M}} \left| \frac{(-2M+1-s)_{2M}}{2M+s} \frac{(\log(z)/z)^{a+N}}{(a+N)^{2M+s-1}} \right|, \end{aligned} \tag{5.100}$$

where $E_\nu(z)$ is the generalized exponential integral [118] and we take the asymptotic estimate $E_\nu(z) \sim e^{-z}/\nu$ as $\nu \to \infty$.

For $M \sim |(a+N)\log(z)|$ and $|s| \ll M$, we use the first order estimate of the Franklin-Friedman expansion for $U(a,b,z)$ in [119] given by

$$U(-2M, -2M+1-s, -(a+N)\log(z)) \sim \left(1 + \frac{2M}{(a+N)\log(z)}\right)^{-s} ((a+N)\log(z))^{2M},$$

replacing it in (5.79) and after observation that the remaining integral is expressible in terms of the incomplete gamma function we obtain

$$|R| \sim \frac{4}{(2\pi)^{2M}} \left| \frac{(-\log(z))^{2M+s-1}}{3^{\min(\Re(s),0)} z^a} \Gamma(1-s, -(a+N)\log(z)) \right|. \tag{5.101}$$

Table 5.1 shows a few examples when $|z| < 1$, otherwise the integral (5.67) is not well defined. Approximations (5.100) and (5.101), although being quite simple, might be used to compute a crude estimate of the magnitude of the remainder in reasonable time.

z	s	a	N	M	(5.67)	(5.79)	(5.100)	(5.101)
0.8	3.2	10.5	6	15	$7.5e{-30}$	$1.5e{-28}$	$1.3e{-38}$	$1.1e{-47}$
0.5 + 0.2i	-30.2 -i	10.5 +5i	10	40	$4.8e{-24}$	$1.0e{-22}$	$8.1e{-37}$	$3.2e{-20}$
0.5 + 0.2i	-30.2 -i	100.5 +5i	100	2000	$2.5e{+302}$	$5.6e{+302}$	$3.9e{+282}$	–
0.5 + 0.7i	-3.2 +10i	10.5 + 5i	250	300	$1.6e{-503}$	$4.8e{-502}$	$1.5e{-503}$	$5.4e{-496}$
0.5 + 0.7i	30.2 +10i	10.5 + 10.5i	600	700	$9.9e{-1240}$	$1.2e{-1238}$	$7.5e{-1216}$	$3.8e{-1264}$

TABLE 5.1: Effectiveness of bound (5.79) in error term of the Euler-Maclaurin formula.

Evaluation of the Euler-Maclaurin tail

A more interesting form of the tail (5.66) is obtained by applying Kummer's transformation to $U(-2k+1,-2k+2-s,-(a+N)\log(z))$, thus

$$T = z^N\left(\frac{1}{2(a+N)^s} + (-\log(z))^s\sum_{k=1}^{M}\frac{B_{2k}}{(2k)!}(-\log(z))^{2k-1}U(s,s+2k,-(a+N)\log(z))\right).$$

For this particular case, $U(a,b,z)$ reduces to a polynomial in $-(a+N)\log(z)$ of degree $2k-1$, indeed expressible in terms of generalized Laguerre polynomials, given by

$$U(s,s+2k,-(a+N)\log(z))) = (-(a+N)\log(z))^{-s}\underbrace{\sum_{j=0}^{2k-1}\binom{2k-1}{j}\frac{(s)_j}{(-(a+N)\log(z))^j}}_{T_2^{(k)}}.$$

Terms $T_2^{(k)}$ can be constructed using a linear holonomic recurrence equation. Let us define the constants expressions p and q

$$p = s-(a+N)\log(z)),\quad q = -\frac{1}{(a+N)\log(z)}.$$

The sequence of terms $T_2^{(k)}$ satisfy the recurrence equation

$$\begin{aligned}T_1^{(k)} &= [pT_2^{(k-1)} + (2k-3)(T_2^{(k-1)} - T_1^{(k-1)})]q\\ T_2^{(k)} &= [pT_1^{(k)} + (2k-2)(T_1^{(k)} - T_2^{(k-1)})]q\end{aligned}$$

for $k \geq 2$, with initial values

$$T_1^{(1)} = 1,\quad T_2^{(1)} = 1 - \frac{s}{(a+N)\log(z)}.$$

A matrix form for $k \geq 2$ is defined as

$$\begin{pmatrix}T_1^{(k)}\\ T_2^{(k)}\\ T_2^{(k-1)}\end{pmatrix} = \begin{pmatrix}q(p+2k-3) & q(3-2k) & 0\\ q(2-2k) & 0 & q(p+2k-2)\\ 1 & 0 & 0\end{pmatrix}\begin{pmatrix}T_2^{(k-1)}\\ T_1^{(k-1)}\\ T_1^{(k)}\end{pmatrix}$$

or simply

$$\begin{pmatrix}T_2^{(k)}\\ T_1^{(k)}\end{pmatrix} = q\begin{pmatrix}k(4k+4p-12)+(p-5)p+8 & k(-4k-2p+10)+3p-6\\ 3-2k & p+2k-3\end{pmatrix}\begin{pmatrix}T_2^{(k-1)}\\ T_1^{(k-1)}\end{pmatrix}.$$

The complexity of the recurrence scheme is $\mathcal{O}(MP)$ and requires a small temporary storage. Note that a matrix recurrence for the sequence of coefficients $T_*^{(k)}$ is suitable in a binary splitting scheme. The previous analysis results in a more

tractable expression for the tail T

$$T = \frac{z^N}{(a+N)^s}\left(\frac{1}{2} + \sum_{k=1}^{M} \frac{B_{2k}}{(2k)!}(-\log(z))^{2k-1} T_2^{(k)}\right). \tag{5.102}$$

Now the terms of tail sum T satisfy a recurrence equation except for the multiplication by Bernoulli numbers. The Bernoulli numbers are cached for repeated evaluation, but computing them the first time at very high precision is time-consuming. We do not attempt to improve current implementations but rather rely on the algorithm implemented in mpmath, which automatically caches Bernoulli numbers B_n when $n < 3000$ for multiple evaluations. For larger values of n the connection to Riemann zeta function $\zeta(n)$ is used. Many recursive algorithms for computing $B_0, \ldots, B_n$ such as $B_n = -\sum_{k=0}^{n-1} \frac{B_k}{k!(n-k+1)}$ require $\mathcal{O}(n^2)$ arithmetic operations. As an alternative, an algorithm based on recycling terms in the Riemann zeta function series expansion, which also have cubic complexity, is implemented in [87].

Evaluation of asymptotic expansions

The main drawback of the asymptotic expansions in (5.87) and (5.94) is the difficulty of computing a large number of Eulerian and peak polynomials efficiently. Computing the first k Eulerian polynomials simultaneously can be performed by using the recursion in (5.85). Thus, given $A_0(z), \ldots, A_{k-1}(z)$, we can compute $A_k(z)$ in $\mathcal{O}(k)$ arithmetic operations, and noting that $A_k(z)$ has $\mathcal{O}(k \log k)$ bits from (5.90), the algorithm needs $\mathcal{O}(k^{3+o(1)})$ bit operations and requires $\mathcal{O}(k^2 \log k)$ space to store previous $A_j(z), j < k$. For example, using a straightforward implementation, we compute $A_0(2), \ldots, A_{1000}(2)$ at 333-bit precision in 1.51 seconds on a 2.6 GHz Intel i7 processor.

To compute k Eulerian polynomials in time complexity $\mathcal{O}(k^{2+o(1)})$ we might apply a *multisectioning* scheme to the bivariate exponential generating function. Alternatively, it is possible to recycle terms of the sum (5.86) to speedup multievaluation, considering that terms $j^k z^j$ can be optimized to only compute binary exponentiation when j is prime and multiplication otherwise. The required number of terms is approximated by solving $J^k z^J = 2^{-P}$,

$$J^* \approx \left\lceil \left| \Re\left(-\frac{k W_{-1}\left(-\frac{(2^{-P})^{1/k}\log(|z|)}{k}\right)}{\log(|z|)} \right) \right| \right\rceil. \tag{5.103}$$

For large k the size of J^* growth rapidly, therefore it is convenient to apply asymptotic faster methods such as the Mittag-Leffler type decomposition previously introduced, which acts as an asymptotic expansion. For $z \in \mathbb{R}$, two optimizations can be implemented:

$$A_k(z) = \frac{(z-1)^{k+1}k!}{z} \Re\left(\frac{1}{\log(z)^{k+1}} + 2\sum_{j=1}^{\infty} \frac{1}{(\log(z) + 2\pi i j)^{k+1}} \right), \quad z \in \mathbb{R}^+,$$

$$A_k(z) = \frac{(z-1)^{k+1}k!}{z} \Re\left(\frac{1}{\log(z)^{k+1}} + \sum_{j=1}^{\infty} \frac{1}{(\log(z) - 2\pi i j)^{k+1}} \right), \quad z \in \mathbb{R}^-.$$

To compute a single $A_{1000}(1/5)$ at 333-bit, the power series requires $J^* = 5545$ whereas the Mittag-Leffler decomposition only needs $J^* = 59$ terms, hence a complete algorithm shall combine the iterative computation via the three-term recurrence and the asymptotic expansion as $k \to \infty$.

Like other orthogonal polynomials, polynomials $P_k(\lambda)$ in (5.83), which are strongly related to Tricomi-Carlitz polynomials, satisfy three-term recurrence,

$$P_0(\lambda) = 1, \quad P_1(\lambda) = 0, \quad \text{and} \quad P_{k+1}(\lambda) = k(P_k(\lambda) + \lambda P_{k-1}(\lambda)),\ k > 1.$$

Given the complexity of computing a large number of Eulerian polynomials, the cost of the three-term recurrence is almost negligible.

The truncation level K in (5.89) is computed at lower precision via linear search and C is estimated as

$$C = \left|\frac{t_{K+1}}{t_K}\right| \sim \left|\frac{z}{z - e^{\mu}} \frac{(s+K)(z-1)}{a\log(z)}\right|, \quad k \to \infty,$$

from which we obtain an estimate of the number of terms

$$K_{max} \approx \left|\frac{a\log(z)(z - e^{\mu})}{z(z-1)} - s\right|.$$

Computation of peak polynomials is carried out using the generating function for peak numbers for moderate k, which evaluation only involves roughly half of the terms $k/2$ compared to the Eulerian polynomials. For example, computing the triangular array for the first 1000 peak numbers using recurrence (5.95) takes 1.55 seconds. For larger k the functional relation with the Eulerian polynomial is applied. A trickier aspect of the asymptotic expansion (5.94) is to determine the optimal truncation K. The coefficients $c_k(u)$ behave as

$$|c_k(u)| = \left|\frac{1}{u^{k+1}} - \frac{\pi^{k+1}}{k!}\coth(\pi u)^{k-1}\operatorname{csch}(\pi u)^2 \mathcal{P}_k(\operatorname{sech}(\pi u)^2)\right| \sim \frac{1}{|1 - iu|^k},$$

as $k \to \infty$. Given the ratio of convergence of the asymptotic expansion, we can estimate the maximum number of terms K_{max}, thus the maximum attainable accuracy as follows

$$\left|\frac{t_{K+1}}{t_K}\right| \sim \left|\frac{(s+K)}{a2\pi(1-iu)}\right|, \quad k \to \infty, \quad K_{max} \approx |a(2\pi - i\log(z)) - s|.$$

It remains to estimate the required numbers of terms K to target P-bit accuracy, which is approximated heuristically and subsequently refined via linear search using

$$K \approx \left\lceil \left| \frac{\varphi(s)}{W_{-1}(\frac{\varphi(s)}{a2\pi\log(z)})} \right| \right\rceil,$$

where $\varphi(s) = -\log(2)P - \log(1+s)$. In fact, we slightly increase K by a factor ≈ 1.2, which works well in practice. Hence, we can evaluate the asymptotic expansion as long as $K \leq K_{max}$ to target an absolute error of 2^{-P}.

Numerical integration

The current implementation in mpmath computes the Lerch transcendent via numerical integration using the double-exponential method for the integral representation (5.58) employing the `quad` function. We choose numerical integration for $\Re(a) > 0$ as a backup method when computation by aforementioned methods is not satisfactory. Note that a few optimizations are possible for real parameters by rewriting the integrand, for example, the integral representation (5.69) when $z > 0$ and $a > 0$ or

$$I = -\Im\left(\int_0^\infty \frac{(1-it/a)^s z^{it}}{(a^2+t^2)^{s/2}(1+t^2/a^2)^{s/2}} \frac{dt}{(e^{2\pi t}-1)}\right), \quad z, s, a \in \mathbb{R},\, z > 0,$$

both integral representations avoiding evaluation of trigonometric functions. The computation at high-precision, say 1000 digits onwards, is generally costly compared to asymptotic methods, therefore this is the method of choice when only strictly indispensable.

5.2.4 Benchmark

In this Section, we benchmark our implementation to current state-of-the-art software supporting evaluation of the Lerch transcendent function to arbitrary-precision. Tests were conducted on an Intel(R) Core(TM) i7-6700HQ CPU at 2.60GHz, using up to 4 cores for parallel mode, running Linux Ubuntu. We compare the computing times of Mathematica 10.4 and mpmath 1.0.0 using functions `Timing[]` and `time.perf_counter()`, respectively. For mpmath we set the precision in bits p using `mpmath.mp.prec = p`, whereas for Mathematica the desired level of precision in digits d is set with `N[..., d]`, applying the conversion factor $d = \lfloor 0.301p \rfloor$. To assess the correctness of our implementation, we compare to Mathematica at higher precision since it is frequently faster and more reliable than mpmath. Note, however, that Mathematica attempts to achieve d digits of precision might fail unexpectedly, therefore we check the consistency of results at increasing levels of precision.

The following tables show timing results to compute the Lerch transcendent for various regimes of the parameters and argument, varying the level of precision. We remark that Mathematica and mpmath use GMP internally, so timing measurements are directly comparable.

Table 5.2 shows the performance of the Euler-Maclaurin formula (5.77) for small z and moderate values of s and a. The Euler-Maclaurin formula is implemented in a loop manner checking the level of cancellation at each iteration and increasing the working precision accordingly to correct it. Hence, a better estimation of the total amount of cancellation would reduce the computation time considerably. However, as we see for these cases, both Mathematica and mpmath are regularly an order of magnitude slower. Furthermore, as previously noted, larger values of $|a|$ improve the convergence of the series, reducing significantly the number of terms N and M; Table 5.3 shows the metrics corresponding to the last iteration.

$\Phi(z,s,a)$	bits	mpmath	Mathematica	Euler-Maclaurin	Parallel
	64	0.096 (0.139)	0.313	0.008 (0.013)	-
$z = 2.5 + 1.5i$	333	1.09 (1.21)	0.672	0.041 (0.089)	-
$s = 1.25 + 2i$	1024	8.39 (9.38)	3.14	0.128 (0.233)	-
$a = 3.5 + 5i$	3333	154.4 (161.1)	27	1.64 (2.37)	-
	10000	1564.6	438	25.04 (33.33)	19.06 (27.32)
	64	0.263 (0.295)	0.047	0.016 (0.039)	-
$z = 2.5 + 7.5i$	333	1.79 (1.98)	0.250	0.110 (0.250)	-
$s = -50.25 + 10i$	1024	6.59 (7.05)	1.58	0.72 (1.49)	-
$a = 1.5 - i$	3333	133.6 (135.1)	21.66	11.83 (16.64)	8.72 (13.22)
	10000	1453	409.1	190.3 (224.5)	153.2 (196.5)
	64	0.253 (0.324)	0.031	0.013 (0.022)	-
$z = 2.5 + 0.5i$	333	1.79 (1.92)	0.265	0.058 (0.104)	-
$s = -100.25 + 10i$	1024	12.66 (13.94)	7.72	0.298 (0.685)	-
$a = 100.5 - 10i$	3333	120.7 (127.1)	83.14	2.97 (4.19)	-
	10000	1500.6	> 1800	24.65 (32.89)	18.88 (28.34)

TABLE 5.2: Time (in seconds) to compute $\Phi(z,s,a)$ with moderate values of z, s and a to 64, 333, 1024, 3333 and 10000 bits of precision. First evaluation pre-computing Bernoulli numbers within parentheses. Maximum time 1800 seconds.

Current implementation does not incorporate complexity-reducing methods for the evaluation of the tail but simply uses the recurrence scheme and only includes optional parallelization of the truncated L-series, thus limiting the observable improvement by activating the parallel mode.

$\Phi(z,s,a)$	bits	N	M	P^W(bits)
	64	7	32 (H)	86
$z = 2.5 + 1.5i$	333	39	172 (H)	400
$s = 1.25 + 2i$	1024	123	247 (A)	1229
$a = 3.5 + 5i$	3333	401	805 (A)	4000
	10000	1203	2416 (A)	12000
	64	28	122 (H)	84
$z = 2.5 + 7.5i$	333	127	402 (A)	383
$s = -50.25 + 10i$	1024	364	1146 (A)	1094
$a = 1.5 - i$	3333	1115	3503 (A)	3346
	10000	3193	10032 (A)	10419
	64	10	46 (H)	108
$z = 2.5 + 0.5i$	333	52	97 (A)	528
$s = -100.25 + 10i$	1024	156	285 (A)	1561
$a = 100.5 - 10i$	3333	480	876 (A)	4790
	10000	1214	2216 (A)	12108

TABLE 5.3: Number of terms N, M in the Euler-Maclaurin expansion and working precision P^W for Euler-Maclaurin cases. (A) and (H) indicate the method used to estimate M, asymptotic and heuristic, respectively.

Table 5.4 assesses the performance of the L-series implementation and its particular cases for $|z| < 1$ and small values of s and a. Due to the performance gap

between Mathematica and mpmath (mpmath only implements numerical integration), only the former is used for benchmarking on subsequent tests. Results show that our implementation is comparable to Mathematica (presumably evaluating the same L-series) at lower precision and it is found to be surprisingly faster at higher precision[3]. Moreover, we observe that our parallelization scheme achieves speedup ratios close to theoretical maximum, which apparently is not implemented in Mathematica. Interestingly, the Euler-Maclaurin formula should be the preferred algorithm at low-medium precision when $|z| \sim 1$.

$\Phi(z,s,a)$	bits	Mathematica	L-series	Parallel	Euler-Maclaurin
	64	0.0011	0.0011	-	-
$z = 1/4$	333	0.0067	0.0055	-	-
$s = 10/4$	1024	0.0339	0.0155	-	-
$a = 20/7$	3333	0.7313	0.0660	0.0431	-
	10000	21.406	0.4885	0.1513	-
	64	0.0015	0.0069	-	0.0031 (0.0053)
$z = 9/10$	333	0.0109	0.0607	0.0429	0.0153 (0.0255)
$s = 10/4$	1024	0.0984	0.2165	0.1052	0.0166 (0.0222)
$a = 20/7$	3333	1.8843	1.0422	0.3281	0.0953 (0.1197)
	10000	39.937	8.6969	2.4881	2.7331 (3.1578)
	64	0.0031	0.0020	-	-
$z = -6/10$	333	0.0156	0.0083	-	-
$s = 10/4$	1024	0.1422	0.0243	-	-
$a = 20/7$	3333	3.0922	0.0983	-	-
	10000	21.2969	0.7500	-	-

TABLE 5.4: Time (in seconds) to compute $\Phi(z,s,a)$ for small argument $|z|$.

The third example assesses the performance of the series acceleration technique for alternating series, which while it is hardly parallelizable, it is consistently faster than Mathematica for all tested instances.

Table 5.5 shows the time to compute the Lerch transcendent using the asymptotic expansion (5.94) for large z and a. The optimal truncation of the first test is $K_{max} = 1598$, limiting the evaluation at 3333 bits of precision, which would require $K = 2767$. The optimal truncation for the second test is $K_{max} = 22297$ requiring up to 1952 terms at 10000 bit of precision. As noted, the time spent on the computation of a large number of peak polynomials accounts for a significant amount of the total time, therefore a more sophisticated and efficient algorithm would be needed at higher precision. On the other hand, for multiple evaluations, peak numbers can be cached same as Bernoulli numbers.

Numerical experiments show a performance deterioration of the Euler-Maclaurin formula as z increases due to catastrophic cancellation, therefore its use should be restricted to low precision calculations. Our implementation of the asymptotic expansion exhibits fast convergence for large parameters, but the limitation on the achievable accuracy forces a switch to numerical integration depending on the desired level of precision.

[3]We guess the poor performance is due to incorrect error tracking, which overestimates the required working precision.

$\Phi(z,s,a)$	bits	mpmath	Mathematica	Euler-Maclaurin	Asymptotic	K	peak time
$z = 140$	64	0.0684	0.0154	0.0027	0.0022	9	6.3%
$s = 1/4$	333	0.7859	0.1219	2.2342	0.0134	59	10.9%
$a = 200$	1024	5.0361	0.7297	-	0.1931	254	10.5%
	64	0.1238	0.0661	-	0.0017	6	4.6%
$z = 10000$	333	1.5456	0.2078	-	0.0066	34	11.5%
$s = 10/4$	1024	9.9478	1.0406	-	0.0481	122	10.0%
$a = 2000$	3333	94.362	15.141	-	0.9498	493	8.4%
	10000	1978.2	283.21	-	39.779	1952	6.0%

TABLE 5.5: Time (in seconds) to compute $\Phi(z,s,a)$ for large parameter a and argument z. Comparison to Euler-Maclaurin at low precision. The rightmost column shows the percentage of the total time devoted to computation of K peak numbers.

Finally, Table 5.6 compares the uniform asymptotic expansion (5.87) to the asymptotic expansion (5.94). Results show that the former expansion should be the preferred choice at low-medium precision for sufficiently large parameters and argument, otherwise the previous methods generally show superior performance. Note that the computation of Eulerian polynomials accounts for the majority of the total time, consequently any improvement on this respect will directly reduce the reported timings.

$\Phi(z,s,a)$	bits	mpmath	Mathematica	Asymptotic	Uniform	Eulerian time
$z = -200.65$	64	0.0228*	0.0469	0.0031 (16)	0.0173 (20)	97.0%
$s = 100.25$	333	0.0149*	0.1563	0.0412 (104)	0.0592 (60)	97.5%
$a = 501.5$	1024	1.0219**	0.8438	0.7512 (421)	0.5151 (229)	98.8%
$z = -20000$	64	0.0101*	0.0312	0.0027 (15)	0.0051 (9)	93.2%
$s = 100.25$	333	0.0168*	0.1875	0.0362 (93)	0.0436 (53)	96.9%
$a = 501.5$	1024	1.7736	1.0313	0.5424 (365)	0.2964 (196)	98.3%

TABLE 5.6: Time (in seconds) to compute $\Phi(z,s,a)$ for large parameter a and s, and argument z. Number of terms for each expansion within parentheses. For mpmath: (*) and (**) indicate no answer and inaccurate answer, respectively.

5.2.5 Discussion

The algorithms presented in this work are an important step towards a complete arbitrary-precision implementation of the Lerch transcendent using asymptotically fast methods. A fundamental improvement to our implementation is to devise a more intelligent strategy to address cancellation issues for the Euler-Maclaurin formula, which should yield a significant reduction of the current overhead factor.

Further work is needed to develop an efficient multithreaded implementation of the asymptotic expansions. More importantly, it remains an open problem whether there is a fast memory-efficient algorithm for computing a large number of Eulerian and peak polynomials.

5.2.6 Appendix - Algorithms and implementations

L-series

Algorithm 7 Alternating L-series for $z < 0$

Input: $a, s, z \in \mathbb{C}$ and $(|z| \leq 1, a \notin \mathbb{Z}_0^-) \vee (\Re(s) > 1, |z| = 1)$

Output: $S = \sum_{k=0}^{\infty} \frac{z^k}{(k+a)^s}$ with D digits of precision initialization

$N \leftarrow \lceil 1.31D \rceil$
$d \leftarrow (3+\sqrt{8})^N$
$b \leftarrow -1$
$c \leftarrow -d$
$u \leftarrow 1$
$S \leftarrow 0$
for $k = 0; k = N; k \leftarrow k+1$ **do**
    $t \leftarrow u/(k+a)^s$
    $c \leftarrow c - b$
    if $k \mod 2$ **then**
        $S \leftarrow S - c \cdot t$
    else
        $S \leftarrow S + c \cdot t$
    end if
    $b \leftarrow b\frac{2(k+N)(k-N)}{(2k+1)(k+1)}$
    $u \leftarrow u \cdot z$
end for
return S/d

LISTING 5.1: mpmath implementation of truncated block L-series

```
def _lerch_lseries_trunc_block(z, s, a, n1, n2, queue):
    """Truncated L-series block for parallel implementation."""
    u = z**n1
    lsum = u / (n1 + a)**s
    for k in range(n1+1, n2):
        u *= z
        lsum += u / (k + a)**s
    queue.put(lsum)
```

LISTING 5.2: mpmath parallel implementation of truncated L-series

```
def _lerch_lseries_trunc_parallel(z, s, a, N):
    """Truncated L-series, parallel implementation."""
    if N == 0: return 0

    threads = multiprocessing.cpu_count()
    block_size = N // threads
    last_block = N % threads

    queue = multiprocessing.Queue()
    processes = [multiprocessing.Process(
        target=_lerch_lseries_trunc_block, args=(z, s, a,
```

```
            block_size*thread, block_size*(thread+1),
                    queue)) for thread in range(threads)]

    for p in processes: p.start()
    for p in processes: p.join()

    series_sum = sum([queue.get() for p in processes])

    if last_block:
        _lerch_lseries_trunc_block(z, s, a, block_size*(threads),
            block_size*(threads) + last_block, queue)
        series_sum += queue.get()

    return series_sum
```

Euler-Maclaurin formula

Algorithm 8 Evaluation of the tail T in (5.66)

Input: $a, s, z \in \mathbb{C}$ and $N, M \in \mathbb{N}$

Output: $T = \sum_{k=1}^{M} \frac{B_{2k}}{(2k)!} \frac{U(-2k+1,-2k+2-s,-(a+N)\log(z))}{(a+N)^{2k-1}}$

$a \leftarrow a + N$
$b \leftarrow -\log(z)$
$r \leftarrow a \cdot b$
$q \leftarrow 1/r$
$p \leftarrow s + r$
$c \leftarrow b^2$
$m \leftarrow 2$
$n \leftarrow b$
$t_1 \leftarrow 1$
$t_2 \leftarrow 1 + s \cdot q$
$T \leftarrow t_2 \cdot b/12$
for $k = 2; k = M; k \leftarrow k + 1$ **do**
    $h \leftarrow 2k + 3$
    $p_1 \leftarrow p + h$
    $p_2 \leftarrow -h$
    $t_1 \leftarrow (t_2 \cdot p_1 + t_1 \cdot p_2)q$
    $t_2 \leftarrow (t_1 \cdot p_1 + t_2 \cdot p_2 + t_1 - t_2)q$
    $m \leftarrow m \cdot (2k-1)(2k)$
    $n \leftarrow n \cdot b$
    $u \leftarrow t_2 \cdot n \cdot B_{2k}/m$
    $T \leftarrow T + u$
end for

Asymptotic expansions

Algorithm 9 Compute first N Eulerian polynomials using recurrence

Input: $z \in \mathbb{C}, N \in \mathbb{N}$
Output: $A_k(z)\ k \in \{0, \ldots, N\}$

$A = [1]$ ▷ Cache of polynomials A with initialization
$y \leftarrow z - 1$
$r \leftarrow 1/y$
$t \leftarrow r$
for $k = 1; k = N; k \leftarrow k + 1$ **do**
    $s \leftarrow 0$
    $t \leftarrow t \cdot y$
    $u \leftarrow 1$
    for $j = 0; j = k; j \leftarrow j + 1$ **do**
        $s \leftarrow s + A[k] \cdot u$
        $u \leftarrow u \cdot (k - j) \cdot r/(j + 1)$
    end for
    $s \leftarrow s \cdot t$
    $A[k] \leftarrow s$ ▷ Store new polynomial s
end for

Algorithm 10 Compute first N rows of peak numbers triangle

Input: $N \in \mathbb{N}$
Output: N rows of $P(k, j), k \in \{0, \ldots, N\}$

$P = [[1], [2]]$ ▷ Cache of coefficients P with initialization
for $k = 2; k = N; k \leftarrow k + 1$ **do**
    $v = [\,]$
    $w \leftarrow \lceil k/2 \rceil$
    $q \leftarrow \lfloor k/2 \rfloor$
    for $j = 0; j = q + 1; j \leftarrow j + 1$ **do**
        if $j < w$ **then**
            $t_1 \leftarrow 2(j + 1)P[k - 1, j]$
        else
            $t_1 \leftarrow 0$
        end if
        if $j > 0$ **then**
            $t_2 \leftarrow (k + 1 - 2j)P[k - 1, j - 1]$
        else
            $t_2 \leftarrow 0$
        end if
        $v[j] \leftarrow t_1 + t_2$
    end for
    $P[k] \leftarrow v$ ▷ Store row v
end for

Algorithm 11 Evaluation of power series in asymptotic expansion (5.94)

Input: $a, s, z \in \mathbb{C}, \Re(a) > 0$ and $N \in \mathbb{N}$
Output:

$$S(N) = \frac{1}{2a^s}\left(\frac{2}{\log(z)} - \coth\left(\frac{\log(z)}{2}\right)\right) + \frac{1}{a^s}\sum_{k=1}^{N}\frac{(s)_k}{a^k(2\pi)^{k+1}}\left(\frac{1}{u^{k+1}} - \frac{\pi^{k+1}}{k!}\coth(\pi u)^{k-1}\operatorname{csch}(\pi u)^2\mathcal{P}_k(\operatorname{sech}(\pi u)^2)\right)$$

$logz \leftarrow \log(z)$
$logz2 \leftarrow logz/2$
$q \leftarrow 1/(2\pi)$
$y \leftarrow 1 + i \cdot logz \cdot q$
$ia \leftarrow 1/a \cdot q$
$n \leftarrow q$
$sech2 \leftarrow \operatorname{sech}(logz2)^2$
$cothk \leftarrow \coth(logz2)$
$iu \leftarrow 1/(logz2/\pi)$
$t \leftarrow iu$
$d \leftarrow \operatorname{csch}(logz2)^2\pi/cothk$
$c \leftarrow (2/logz - cothk)/2$
$P = GeneratePeakNumbers(N)$ ▷ pre-compute N of peak numbers triangle
$S \leftarrow 0$
for $k = 1; k = N; k \leftarrow k + 1$ **do**
    $n \leftarrow n \cdot ia(s + k - 1)$
    $t \leftarrow t \cdot iu$
    $d \leftarrow d \cdot (\pi cothk/k)$
    $pN \leftarrow \lceil k/2 \rceil$
    $ps \leftarrow 0$
    $sc \leftarrow 1$
    for $j = 0; j = pN; j \leftarrow j + 1$ **do**
        $ps \leftarrow ps + P[k-1, j] \cdot sc$
        $sc \leftarrow sc \cdot sech2$
    end for
    $h \leftarrow t - d \cdot ps$
    $S \leftarrow S + n \cdot h$
end for
return $(c + S)/a^s$

Bibliography

[1] Z. Altaç. Integrals involving Bickley and Bessel functions in radiative transfer, and generalized exponential integral functions. *J. Heat Transfer*, 118(3):789–792, 1996.

[2] T. Amdeberhan and D. Zeilberger. Hypergeometric series acceleration via the WZ method. *Electron. J. Combin.*, 4(3), 1996.

[3] Donald E. Amos. Algorithms 683: A portable FORTRAN subroutine for exponential integrals of a complex argument. *ACM Trans. Math. Softw.*, 16(2):178–182, 1990.

[4] E. G. Andrews. Applications of basic hypergeometric functions. *SIAM Review*, 16(4):441–484, 1974.

[5] T. M. Apostol. On the lerch zeta function. *Pacific J. Math.*, 1(2):161–167, 1951.

[6] D. H. Bailey and J. M. Borwein. High-precision numerical integration: Progress and challenges. *Journal of Symbolic Computation*, 46(7):741–754, 2011.

[7] D. H. Bailey, K. Jeyabalan, and X. S. Li. A comparison of three high-precision quadrature schemes. *Experimental Mathematics*, 14(3):317–329, 2005.

[8] D. Berend and T. Tassa. Improved bounds on bell numbers and on moments of sums of random variables. *Probability and Mathematical Statistics*, 30(2):185–205, 2010.

[9] D. J. Bernstein. Fast multiplication and its applications. *Algorithmic Number Theory*, 44:325–384, 2008.

[10] G. Boese. Eine majorante asymptotische Abschätzung für die unvollständige Gammafunktion. *Z. Angew. Math. Mech.*, (52):552–553, 1972.

[11] Boost. Boost C++ Libraries. http://www.boost.org/, 2016. Last accessed 2016-12-29.

[12] D. Borwein, J. M. Borwein, and O. Chan. The evaluation of Bessel functions via exp-arc integrals. *Journal of Mathematical Analysis and Applications*, 341(1):478–500, 2008.

[13] D. Borwein, J. M. Borwein, and R. E. Crandall. Effective laguerre asymptotics. *SIAM J. Numerical Analysis*, 46(6):3285–3312, 2008.

[14] 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.

[15] P. B. Borwein. An efficient algorithm for the Riemann zeta function. *Canadian Mathematical Society Conference Proceedings*, 27:29–32, 2000.

[16] W. Bosma, J. Cannon, and C. Playoust. The Magma algebra system. I. The user language. *J. Symbolic Comput.*, 24(3-4):235–265, 1997. Computational algebra and number theory (London, 1993).

[17] A. Bostan, P. Gaudry, and É. Schost. Linear recurrences with polynomial coefficients and application to integer factorization and Cartier-Manin operator. *SIAM Journal on Computing*, 36(6):1777–1806, 2007.

[18] R. Brent and P. Zimmermann. *Modern Computer Arithmetic*. Cambridge University Press, New York, NY, USA, 2010.

[19] R. P. Brent. The complexity of multiple-precision arithmetic. *The Complexity of Computatioanl Problem Solving*, pages 126–165, 1976.

[20] R. P. Brent and D. Harvey. Fast computation of Bernoulli, Tangent and Secant numbers. In *Computational and Analytical Mathematics*, pages 127–142, New York, NY, 2013. Springer New York.

[21] C. Brezinski and M. Redivo Zaglia. *Extrapolation Methods. Theory and Practice*, volume 2 of *Studies in Computational Mathematics*. North-Holland Publishing Co., Amsterdam, 1991. With 1 IBM-PC floppy disk (5.25 inch).

[22] H. Buchholz. *The Confluent Hypergeometric Function with Special Emphasis on Its Applications*. Springer-Verlag, New York, 1969. Translated from the German by H. Lichtbau and K. Wetzel.

[23] S. Chandrasekhar. *Radiative Transfer*. Dover Books on Intermediate and Advanced Mathematics. Dover Publications, 1960.

[24] C. Chiccoli, S. Lorenzutta, and G. Maino. A numerical method for generalized exponential integrals. *Comput. Math. Appl.*, 14(4):261–268, 1987.

[25] C. Chiccoli, S. Lorenzutta, and G. Maino. An algorithm for exponential integrals of real order. *Computing*, 45(3):269–276, 1990.

[26] C. Chiccoli, S. Lorenzutta, and G. Maino. Recent results for generalized exponential integrals. *Comput. Math. Appl.*, 19(5):21–29, 1990.

[27] C. Chiccoli, S. Lorenzutta, and G. Maino. Concerning some integrals of the generalized exponential-integral function. *Comput. Math. Appl.*, 23(11):13–21, 1992.

[28] W. J. Cody. A survey of practical rational and polynomial approximation of functions. *SIAM Review*, 12(3):400–423, 1970.

[29] W. J. Cody. Algorithm 715: SPECFUN–a portable FORTRAN package of special function routines and test drivers. *ACM Trans. Math. Softw.*, 19(1):22–30, 1993.

[30] W. J. Cody and H. C. Thacher, Jr. Chebyshev approximations for the exponential integral $Ei(x)$. *Math. Comp.*, 23(106):289–303, 1969.

[31] M. W. Coffey. On hypergeometric series reductions from integral representations, the Kampe de Feriet function, and elsewhere. *Int. J. Mod. Phys.*, B(19):4483–4493, 2006.

[32] M. W. Coffey. A set of identities for a class of alternating binomial sums arising in computing applications. *Util. Math.*, 2007.

[33] M. W. Coffey. An efficient algorithm for the Hurwitz zeta and related functions. *Journal of Computational and Applied Mathematics*, 225(2):338–346, 2009.

[34] H. Cohen. *Number Theory - Vol II: Analytic and Modern Tools*. Springer-Verlag - Graduate Texts in Mathematics 240, 2007.

[35] H. Cohen, F. Rodriguez Villegas, and D. Zagier. Convergence acceleration of alternating series. *Experiment. Math.*, 9(1):3–12, 2000.

[36] O. Costin and S. Garoufalidis. Resurgence of the fractional polylogarithms. 2009.

[37] R. E. Crandall. Unified algorithms for polylogarithms, *L*-series, and zeta variants. 2012.

[38] A. Cuyt, V. Petersen, B. Verdonk, H. Waadeland, W. B. Jones, and C. Bonan-Hamada. *Handbook of Continued Fractions for Special Functions*. Kluwer Academic Publishers Group, Dordrecht, 2007.

[39] N.G. de Bruijn. *Asymptotic Methods in Analysis*. Bibliotheca mathematica. Dover Publications, 1970.

[40] A. Deaño and D. Huybrechs. Complex gaussian quadrature of oscillatory integrals. *Numerische Mathematik*, 112(2):197–219, 2009.

[41] A. Deaño and N. M. Temme. On modified asymptotic series involving confluent hypergeometric functions. *Electronic Transactions on Numerical Analysis*, 35:88–103, 2009.

[42] T. J. Dekker. A floating-point technique for extending the available precision. *Numer. Math.*, 18(3):224–242, 1971.

[43] *NIST Digital Library of Mathematical Functions*. http://dlmf.nist.gov/, Release 1.0.14 of 2016-12-21. F. W. J. Olver, A. B. Olde Daalhuis, D. W. Lozier, B. I. Schneider, R. F. Boisvert, C. W. Clark, B. R. Miller and B. V. Saunders, eds.

[44] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

[45] T.M. Dunster, A. Gil, and J. Segura. Uniform asymptotic expansions for laguerre polynomials and related confluent hypergeometric functions. 2017.

[46] W. Ehrhardt. *The AMath and DAMath Special Functions (version 2.23)*, 2018. http://www.wolfgang-ehrhardt.de/amath_functions.html.

[47] A. Enge, M. Gastineau, P. Théveny, and P. Zimmermann. *mpc — A library for multiprecision complex arithmetic with exact rounding*. INRIA, 1.0.3 edition, February 2015. http://mpc.multiprecision.org/.

[48] A. Erdélyi, W. Magnus, F. Oberhettinger, and F. G. Tricomi. *Higher Transcendental Functions. Vol. I.* McGraw-Hill Book Company, Inc., New York-Toronto-London, 1953.

[49] C. Ferreira and J. L. López. Asymptotic expansions of the Hurwitz—Lerch zeta function. *Journal of Mathematical Analysis and Applications*, 298(1):210–224, 2004.

[50] C. Ferreira, J. L. López, P. Pagola, and E. Pérez Sinusía. The Laplace's and steepest descents methods revisited. *Int. Math. Forum*, 2:297–314, 2007.

[51] C. J. Fewster and D. Siemssen. Enumerating permutations by their run structure. *The electronic journal of combinatorics*, 21, 10 2014.

[52] P. Flajolet and R. Sedgewick. Mellin transforms and asymptotics: Finite differences and Rice's integrals. *Theoretical Computer Science*, (144):101–124, 1995.

[53] P. Flajolet and R. Sedgewick. *Analytic Combinatorics*. Cambridge University Press, New York, NY, USA, 1 edition, 2009.

[54] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Trans. Math. Softw.*, 33(2), June 2007.

[55] J. Franklin and B. Friedman. A convergent asymptotic representation for integrals. *Mathematical Proceedings of the Cambridge Philosophical Society*, 53(3):612–619, 1957.

[56] Galassi et al. *Gnu Scientific Library: Reference Manual (3rd Ed.)*. Network Theory Ltd., 2003.

[57] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.10.0*, 2018.

[58] W. Gautschi. Exponential integral $\int_1^\infty e^{-xt}t^{-n}dt$ for large values of n. *J. Res. Nat. Bur. Standards*, 62:123–125, 1959.

[59] W. Gautschi. Gauss quadrature approximations to hypergeometric and confluent hypergeometric functions. *Journal of Computational and Applied Mathematics*, 139:173–187, 2002.

[60] A. Gil, D. Ruiz-Antolín, J. Segura, and N. M. Temme. Algorithm 969: Computation of the incomplete gamma function for negative values of the argument. *ACM Trans. Math. Softw.*, 43(3):26:1–26:9, November 2016.

[61] A. Gil, J. Segura, and N. M. Temme. *Numerical methods for special functions*. SIAM, 2007.

[62] A. Gil, J. Segura, and N. M. Temme. Efficient and accurate algorithms for the computation and inversion of the incomplete gamma function ratios. *SIAM J. Scientific Computing*, 34(6), 2012.

[63] A. Gil, J. Segura, and N. M. Temme. Computing the kummer function $U(a, b, z)$ for small values of the arguments. *Applied Mathematics and Computation*, 271:532 – 539, 2015.

[64] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 1994.

[65] S. Graillat. Accurate floating-point product and exponentiation. *IEEE Transactions on Computers*, 58(7):994–1000, 2009.

[66] S. Graillat, P. Langlois, and N. Louvet. Compensated horner scheme. 2005.

[67] S. Graillat, P. Langlois, and N. Louvet. Improving the compensated Horner scheme with a fused multiply and add. In *Proceedings of the 2006 ACM Symposium on Applied Computing*, SAC '06, pages 1323–1327, New York, NY, USA, 2006.

[68] S. Graillat and V. Ménissier-Morain. Compensated horner scheme in complex floating point arithmetic. 2008.

[69] T. Granlund and the GMP development team. *GNU MP: The GNU Multiple Precision Arithmetic Library*. http://gmplib.org/.

[70] D. S. Grebenkov. First exit times of harmonically trapped particles: a didactic review. *Journal of Physics A: Mathematical and Theoretical*, 48(1):013001, 2015.

[71] A. Griffin, Wen-Chin Wu, and S. Stringari. Hydrodynamic modes in a trapped Bose gas above the Bose-Einstein transition. *Phys. Rev. Lett.*, 78:1838–1841, Mar 1997.

[72] J. Guillera and J. Sondow. Double integrals and infinite products for some classical constants via analytic continuations of lerch's transcendent. *The Ramanujan Journal*, 16(3):247–270, Aug 2008.

[73] B. Haible and T. Papanilolaou. Fast evaluation of series of rational numbers. *J.P. Buhler (Ed.), Algorithmic Number Theory, Third International Symposium, ANTS-III, Lecture Notes in Computer Science*, 1423, 1998.

[74] J. Harrison. Fast and Accurate Bessel Function Computation. In *2009 19th IEEE Symposium on Computer Arithmetic*, pages 104–113, 2009.

[75] D. Harvey. A multimodular algorithm for computing Bernoulli numbers. *Mathematics of Computation*, 79(272):2361–2370, 2010.

[76] R. Hekmati and H. Mirhajianmoghadam. Nested performance profiles for benchmarking software. *preprint*, 2018.

[77] P. Henrici. *Applied and Computational Complex Analysis. Vol. 2: Special Functions—Integral Transforms—Asymptotics—Continued Fractions*. Wiley-Interscience [John Wiley & Sons], New York, 1977. Reprinted in 1991.

[78] G. Hiary. Fast methods to compute the Riemann zeta function. *Annals of mathematics*, 174:891–946, 2011.

[79] Y. Hida, X. S. Li, and D. H. Bailey. Algorithms for quad-double precision floating point arithmetic. In *Proceedings 15th IEEE Symposium on Computer Arithmetic. ARITH-15 2001*, pages 155–162, 2001.

[80] Y. Hida, X. S. Li, and D. H. Bailey. Library for double-double and quad-double arithmeti. *NERSC Division, Lawrence Berkeley National Laboratory*, 2007.

[81] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2002.

[82] D. Huybrechs and S. Vandewalle. On the evaluation of highly oscillatory integrals by analytic continuation. *SIAM J. Numerical Analysis*, 44(3):1026–1048, 2006.

[83] A. Iserles. On the numerical quadrature of highly-oscillating integrals I: Fourier transforms. *IMA Journal of Numerical Analysis*, 24(3):365–391, 2004.

[84] S. Janson, D. E. Knuth, T. Łuczak, and B. Pittel. The birth of the giant component. *Random Structures Algorithms*, 4(3):231–358, 1993. With an introduction by the editors.

[85] U. D. Jentschura, P. J. Mohr, G. Soff, and E. J. Weniger. Convergence acceleration via combined nonlinear-condensation transformations. *Computer Physics Communications*, 116(1):28 – 54, 1999.

[86] U. D. Jentschura, G. Soff, and P. J. Mohr. Lamb shift of $3p$ and $4p$ states and the determination of α. *Phys. Rev. A*, 56:1739–1755, Sep 1997.

[87] F. Johansson. Arb: a C library for ball arithmetic. *ACM Communications in Computer Algebra*, 47(4):166–169, 2013.

[88] F. Johansson. Evaluating parametric holonomic sequences using rectangular splitting. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, ISSAC '14, pages 256–263, 2014.

[89] F. Johansson. Rigorous high-precision computation of the hurwitz zeta function and its derivatives. *Numerical Algorithms*, 69(2):253–270, Jun 2015.

[90] F. Johansson. Computing hypergeometric functions rigorously. working paper or preprint, 2016.

[91] F. Johansson. Numerical integration in arbitrary-precision ball arithmetic. In *Mathematical Software – ICMS 2018*, pages 255–263, Cham, 2018. Springer International Publishing.

[92] F. Johansson et al. *mpmath: a Python library for arbitrary-precision floating-point arithmetic (version 0.19)*, December 2014. http://mpmath.org/.

[93] M. Joldeş, O. Marty, J. Muller, and V. Popescu. Arithmetic algorithms for extended precision using floating-point expansions. *IEEE Transactions on Computers*, 65(4):1197–1210, 2016.

[94] M. Joldeş, J.-M. Muller, and V. Popescu. On the computation of the reciprocal of floating point expansions using an adapted Newton-Raphson iteration. *2014 IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors*, pages 63–67, 2014.

[95] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, 2001–.

[96] A. Karatsuba and Y. Ofman. Multiplication of multidigit numbers on automata (in russian). *Doklady Akad. Nauk SSSR*, 145:293–294, 1962.

[97] J. D. Kečlić and P. M. Vasić. Some inequalities for the gamma function. *Publ. Inst. Math. (Beograd) (N. S.)*, 11:107–114, 1971.

[98] F. Klein. *Lectures of the Icosahedron*. Dover Publications, 2003.

[99] D. E. Knuth. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[100] P. Koev and A. Edelman. The efficient evaluation of the hypergeometric function of a matrix argument. *Mathematics of Computation*, 75(254):833–846, 2006.

[101] A. Laurincikas and R. Garunkstis. *The Lerch zeta-function*. Springer, 2002.

[102] J. LeCaine. *A Table of Integrals Involving the Functions $En(x)$*. N.R.C. (National Research Council). National Research Council Canada, Division of Atomic Energy, 1949.

[103] V. Lefèvre and P. Zimmermann. Optimized Binary64 and Binary128 Arithmetic with GNU MPFR. In *24th IEEE Symposium on Computer Arithmetic (ARITH 24)*, pages 18–26, 2017.

[104] M. Lerch. Note sur la fonction $\aleph(w,x,s) = \sum_{k=0}^{\infty} \frac{e^{2k\pi ix}}{(w+k)^s}$. *Acta Math.*, 11(1-4):19–24, 1887.

[105] D. Levin. Fast integration of rapidly oscillatory functions. *Journal of Computational and Applied Mathematics*, 67(1):95 – 101, 1996.

[106] R. Lipschitz. Untersuchung einer aus vier Elementen gebildeten Reihe. *J. Reine Angew. Math.*, 54:313–328, 1857.

[107] J. L. López and P. J. Pagola. The confluent hypergeometric functions $M(a,b;z)$ and $U(a,b;z)$ for large b and z. *J. Comput. Appl. Math.*, 233(6):1570–1576, 2010.

[108] Y. L. Luke. *Algorithms for the Computation of Mathematical Functions*. Academic Press, 1977.

[109] Maplesoft, a division of the Waterloo Maple Inc. Maple 18.

[110] M. Mori and M. Sugihara. The double-exponential transformation in numerical analysis. *Journal of Computational and Applied Mathematics*, 127:287–296, 2001.

[111] S. L. Moshier. Cephes mathematical function library, 2000.

[112] J.-M. Muller, N. Brisebarre, F. de Dinechin, V. Jeannerod C.-P.and Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres. *Handbook of Floating-Point Arithmetic*. Birkhäuser, 2010.

[113] K. E. Muller. Computing the confluent hypergeometric function, $M(a, b, x)$. *Numerische Mathematik*, 90(1):179–196, 2001.

[114] The Numerical Algorithms Group (NAG). *The NAG Library*. Oxford, United Kingdom, 2018. www.nag.com.

[115] W. F. Nardin, M. Perger and A. Bhalla. Algorithm 707. conhyp : A numerical evaluator of the confluent hypergeometric function for complex arguments of large magnitudes. *ACM Trans. Math. Software*, 18:345–349, 1992.

[116] W. F. Nardin, M. Perger and A. Bhalla. Numerical evaluation of the confluent hypergeometric function for complex arguments of large magnitudes. *J. Comput. Appl. Math.*, 39:193–200, 1992.

[117] G. Navas-Palencia. Portfolio credit risk: Models and numerical methods. Master's thesis, Universitat Politècnica de Catalunya, 2016.

[118] G. Navas-Palencia. Fast and accurate algorithm for the generalized exponential integral $E_\nu(x)$ for positive real order. *Numerical Algorithms*, 77(2):603–630, 2018.

[119] G. Navas-Palencia. High-precision computation of the confluent hypergeometric functions via Franklin-Friedman expansion. *Advances in Computational Mathematics*, 44(3):841–859, 2018.

[120] G. Navas-Palencia and A. Arratia. On the computation of confluent hypergeometric functions for large imaginary part of parameters b and z. *Lecture Notes in Computer Science*, 9725:241–248, 2016.

[121] T. Ogita, S. M. Rump, and S. Oishi. Accurate sum and dot product. *SIAM J. Sci. Comput.*, 26(6):1955–1988, 2005.

[122] A. B. Olde Daalhuis. Hyperasymptotic expansions of confluent hypergeometric functions. *IMA Journal of Applied Mathematics*, 49:203–216, 1992.

[123] A. B. Olde Daalhuis and F. W. J. Olver. Hyperasymptotic solutions of second-order linear differential equations. I. *Methods Appl. Anal.*, 2(2):173–197, 1995.

[124] F. W. J. Olver. Uniform, exponentially improved, asymptotic expansions for the confluent hypergeometric function and other integral transforms. *SIAM J. Math. Anal.*, 22(5):1475–1489, 1991.

[125] F. W. J. Olver. *Asymptotics and Special Functions*. A. K. Peters, Wellesley, MA, 1997. Reprint, with corrections, of original Academic Press edition, 1974.

[126] T. Ooura. A double exponential formula for the Fourier transforms. *Research Institute for Mathematical Sciences*, 41:971–977, 2005.

[127] T. Ooura and M. Mori. The double exponential formula for oscillatory functions over the half infinite interval. *Journal of Computational and Applied Mathematics*, 38(1):353–360, 1991.

[128] R. B. Paris. The Stokes phenomenon and the Lerch zeta function. *Mathematica Aeterna*, 6(2):165–179, 2016.

[129] R.B. Paris. High-precision evaluation of the Bessel functions via Hadamard series. *Journal of Computational and Applied Mathematics*, 224(1):84 – 100, 2009.

[130] M. Paterson and L. Stockmeyer. On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM Journal on Computing*, 2(1):60–66, 1973.

[131] T. N. L. Patterson. On high precision methods for the evaluation of Fourier integrals with finite and infinite limits. *Numerische Mathematik*, 27:41–52, 1976.

[132] J. W. Pearson. Computation of hypergeometric functions. Master's thesis, University of Oxford, 2009.

[133] J. W. Pearson, S. Olver, and M. A. Porter. Numerical methods for the computation of the confluent and Gauss hypergeometric functions. *Numerical Algorithms*, 74(3):821–866, 2017.

[134] M. Petkovsek, H. Wilf, and D. Zeilberger. *A=B*. A K Peters, Ltd., 1997.

[135] D. M. Priest. Algorithms for arbitrary precision floating point arithmetic. In *Proceedings 10th IEEE Symposium on Computer Arithmetic*, pages 132–143, 1991.

[136] J. L. Raabe. Zurückführung einiger Summen und bestimmten Integrale auf die Jakob Bernoullische Function. *J. reine angew. Math.*, 42:348–376, 1851.

[137] S. Roman. *The Umbral Calculus*. Pure and Applied Mathematics. Elsevier Science, 1984.

[138] Sra S. Directional statistics in machine learning: A brief review. *Applied Directional Statistics*, 2016.

[139] A. Schönhage and V. Strassen. Schnelle multiplikation grosser zahlen. *Computing*, 7:281–292, 1971.

[140] J.R. Shewchuk. Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete & Computational Geometry*, 18(3):305–368, 1997.

[141] B.W. Shore and D.H. Menzel. *Principles of atomic spectra*. Wiley series in pure and applied spectroscopy. Wiley, 1968.

[142] A. Sidi. *Practical Extrapolation Methods: Theory and Applications*, volume 10 of *Cambridge Monographs on Applied and Computational Mathematics*. Cambridge University Press, Cambridge, 2003.

[143] L. J. Slater. *Confluent Hypergeometric Functions*. Cambridge University Press, Cambridge-New York, 1960. Table errata: Math. Comp. v. 30 (1976), no. 135, 677–678.

[144] N. J. A. Sloane. The On-Line Encyclopedia of Integer Sequences.

[145] D. M. Smith. Efficient multiple-precision evaluation of elementary functions. *Mathematics of Computation*, 52:131–131, 1989.

[146] IEEE Computer Society. IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2008*, pages 1–70, 2008. Available from https://ieeexplore.ieee.org/document/4610935.

[147] RogueWave Software. *IMSL Numerical Libraries*. Louisville, USA, 2018. https://www.roguewave.com/help-support/documentation/imsl-numerical-libraries.

[148] H.M Srivastava and P. G. Todorov. An explicit formula for the generalized Bernoulli polynomials. *Journal of Mathematical Analysis and Applications*, 130(2):509–513, 1988.

[149] J. R. Stembridge. Enriched p-partitions. *Transactions of the American Mathematical Society*, 349(2):763–788, 1997.

[150] H. Takahasi and M. Mori. Double exponential formulas for numerical integration. *Publications of the Research Institute for Mathematical Sciences*, 9:721–741, 1974.

[151] D. Tasche. Estimating discriminatory power and PD curves when the number of defaults is small. *Lloyds Banking Group*, 2009.

[152] N. M. Temme. Uniform asymptotic expansions of confluent hypergeometric functions. *J. Inst. Math. Appl.*, 22(2):215–223, 1978.

[153] N. M. Temme. On the expansion of confluent hypergeometric functions in terms of Bessel functions. *Journal of Computational and Applied Mathematics*, 7:27–32, 1981.

[154] N. M. Temme. The numerical computation of the confluent hypergeometric function $U(a, b, z)$. *Numer. Math.*, 41(1):63–82, 1983. Algol 60 variable-precision procedures are included.

[155] N. M. Temme. Uniform asymptotic expansions for Laplace integrals. *Analysis*, 3:221–249, 1983.

[156] N. M. Temme. Laplace type integrals: Transformations to standard form and uniform asymptotic expansions. *Quarterly of Applied Mathematics*, 43(1):103–123, 1985.

[157] N. M. Temme. *Asymptotic Methods for Integrals*, volume 6 of *Series in Analysis*. World Scientific Publishing Co. Pte. Ltd., Hackensack, NJ, 2015.

[158] The MathWorks, Inc. Matlab release 2017b.

[159] The PARI Group, Univ. Bordeaux. *PARI/GP version* `2.11.0`, 2018. Available from `http://pari.math.u-bordeaux.fr/`.

[160] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 8.4.0)*, 2018. `http://www.sagemath.org`.

[161] L. N. Trefethen and J. A. C. Weideman. The exponentially convergent trapezoidal rule. *SIAM Review*, 56(3):385–458, 2014.

[162] F. G. Tricomi. Fonctions hypergéométriques confluentes. *Mémorial des sciences mathématiques*, 140:1–86, 1960.

[163] W.H. Vandevender and K. H. Haskell. The SLATEC mathematical subprogram library. *ACM SIGNUM Newsletters*, 17(3), 1982.

[164] 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.

[165] G. N. Watson. The harmonic functions associated with the parabolic cylinder. *Proceedings of the London Mathematical Society*, s2-17(1):116–148, 1918.

[166] E. J. Weniger. Nonlinear sequence transformations for the acceleration of convergence and the summation of divergent series. *Computer Physics Reports*, 10(5):189 – 371, 1989.

[167] E. J. Weniger. Summation of divergent power series by means of factorial series. *Applied Numerical Mathematics*, 60(12):1429 – 1441, 2010. Approximation and extrapolation of convergent and divergent sequences and series (CIRM, Luminy - France, 2009).

[168] J. L. Willis. Acceleration of generalized hypergeometric functions through precise remainder asymptotics. *Numerical Algorithms*, 59(3):447–485, 2012.

[169] Wolfram Research, Inc. Mathematica 10.

[170] J. Worpitzky. Studien über die Bernoullischen und Eulerschen zahlen. *J. reine angew. Math.*, 94:203–233, 1883.

[171] S. Zhang and J. Jin. *Computation of special functions*. John Wiley & Sons Inc., New York, 1996.

[172] Y. Zhuang. Counting permutations by runs. *Journal of Combinatorial Theory, Series A*, 142:147 – 176, 2016.