magnetism

MDPI

Article

Numerically Stable and Computationally Efficient Expression for the Magnetic Field of a Current Loop

Michael Ortner *, Peter Leitner and Florian Slanovc

Magnetic Microsystem Technologies, Silicon Austria Labs, 9500 Villach, Austria
* Correspondence: michael.ortner@silicon-austria.com

Abstract: In this work, it is demonstrated that straightforward implementations of the well-known textbook expressions of the off-axis magnetic field of a current loop are numerically unstable in a large region of interest. Specifically, close to the axis of symmetry and at large distances from the loop, complete loss of accuracy happens surprisingly fast. The origin of the instability is catastrophic numerical cancellation, which cannot be avoided with algebraic transformations. All exact expressions found in the literature exhibit similar instabilities. We propose a novel exact analytic expression, based on Bulirsch's complete elliptic integral, which is numerically stable (15–16 significant figures in 64 bit floating point arithmetic) everywhere. Several field approximation methods (dipole, Taylor expansions, Binomial series) are studied in comparison with respect to accuracy, numerical stability and computation performance. In addition to its accuracy and global validity, the proposed method outperforms the classical solution, and even most approximation schemes in terms of computational efficiency.

Keywords: magnetic field; current loop; analytic solution; numerical stability; computation performance

Citation: Ortner, M.; Leitner, P.; Slanovc, F. Numerically Stable and Computationally Efficient Expression for the Magnetic Field of a Current Loop. *Magnetism* **2023**, *3*, 11–31. https://doi.org/10.3390/magnetism3010002

Academic Editors: Roberto Zivieri, Giancarlo Consolo and Israa Medlej

Received: 9 November 2022
Revised: 16 December 2022
Accepted: 23 December 2022
Published: 30 December 2022

1. Introduction and Motivation

Analytic expressions for the magnetic field of current and magnetization problems are widely used in modern science and engineering. They offer much faster field computation than their numerical counterparts, and the superposition principle often makes up for the simple geometries for which solutions exist. User friendly, but much slower numerical alternatives like ANSYS Maxwell [1] or Comsol [2] are often not available due to their substantial financial costs, and implementations through open-source packages like FEniCS [3], or NG-Solve [4] are time consuming and require substantial know-how.

In addition, analytic forms enable extreme computation precision up to 12–14 significant digits in Double (64 bit) floating point arithmetic when they are properly implemented. The downside is that the expressions found in the literature are mostly arranged to look as simple as possible, with little or no thought given to their implementation. A straightforward transfer to the computer code often results in numerical instabilities close to symmetry positions and special cases, surface and edge extensions, and at large distances from the magnetic field sources.

A good example is the classical textbook expression for the radial component of the B-field of a circular current loop [5],

$$B_\rho = \frac{\mu_0 i_0}{2\pi} \frac{z}{((\rho+\rho_0)^2+z^2)^{1/2}} \left[-K(k^2) + \frac{\rho^2+\rho_0^2+z^2}{(\rho-\rho_0)^2+z^2} E(k^2) \right], \quad \text{with} \quad k^2 = \frac{4\rho_0\rho}{(\rho+\rho_0)^2+z^2}. \tag{1}$$

The current loop with radius ρ_0 lies in the $z = 0$ plane, with center in the origin of a cylindrical coordinate system (ρ, φ, z), and carries a current i_0. The vacuum permeability

is denoted by μ_0, and K and E are the complete elliptic integrals of first and second kind, respectively, for which fast and stable numerical algorithms exist [6,7] and implementations are provided by common tools like Mathematica [8], Matlab [9] and Scipy [10].

A straightforward implementation $\hat{B}_\rho$ of expression (1) is numerically troublesome because catastrophic cancellation happens close to the axis, $\rho \ll \rho_0$, and at large distances from the loop, $\rho^2 + z^2 \gg \rho_0^2$. The magnitude of this phenomenon is easily underestimated. For example, evaluations of $\hat{B}_\rho$ in Double floating point format give only four correct significant figures when evaluated at not very distant positions $\boldsymbol{r} = (0.01\rho_0, \varphi, 100\rho_0)$. The extent of the problem is laid out in Figure 1. There, the first quadrant in ρz-space is depicted for an arbitrary φ. The other quadrants follow from the symmetry of the problem.

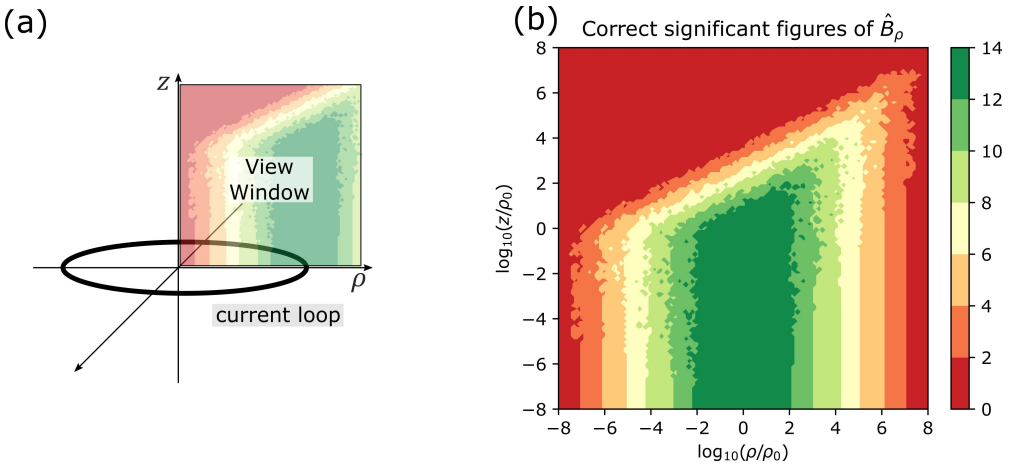

Figure 1. (**a**) Sketch of a current loop and the observed first quadrant. (**b**) The number of correct significant figures of a straightforward implementation of the textbook expression for the radial component of the B-field of a current loop on a log–log scale.

Discussions in the Magpylib [11] forum on Github underline the practical relevance of the issue. Magpylib is an open-source Python package that combines analytic solutions of permanent magnets and current problems with a geometric reference frame API. It enables users to easily compute the magnetic fields of sources at observers with arbitrary relative position and orientation. In the now outdated version 3.0.5, a numerically unstable, straightforward implementation is used for the current loop field, together with a special case for $\rho = 0$. When an unsuspecting user creates such a current loop object and rotates it about an observer located on the loop axis, see Figure 2a, the limited precision of the rotation naturally results in a misalignment between the observer and loop axis by small values of the order of the machine precision. In the reference frame of the current loop, the observer is thus moved from the stable special case, $\rho = 0$, to the numerically unstable position $0 < \rho \ll \rho_0$. The resulting bad computation is shown in Figure 2b for multiple angles α ranging from 0 to 360 degrees. Complete loss of precision can be observed in many cases.

Achieving numerical stability is of critical importance. Instabilities like the one described in this work are often not visible, but can lead to erroneous computation results that are difficult to track down. In this paper, we analyze various expressions for the B-field of a current loop in terms of range of validity, precision, numerical stability and performance. Eventually, a reader can decide by himself which formulation suits his needs best.

The structure of the manuscript is as follows: In Section 2, the classical textbook expression is discussed, the origin of the problem is identified, and several expressions for the B-field are introduced and analyzed in terms of accuracy. In Section 3, all these expressions are tested in terms of computation performance. In Section 4, all results are reviewed and discussed. Finally, in Appendix A, various algorithms used in this work are provided.

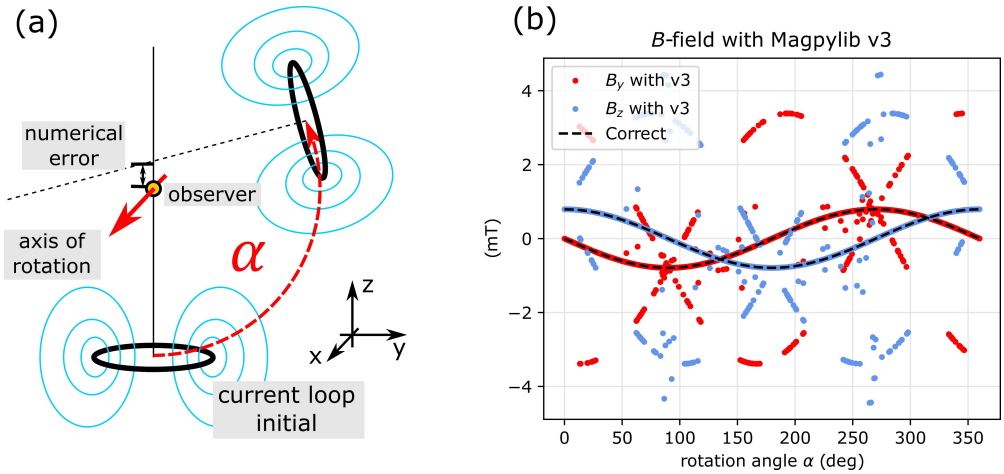

Figure 2. (**a**) Sketch of typical current loop positioning with the package Magpylib. (**b**) Demonstrating the relevance of the numerical instability, which becomes visible when a current loop rotates about an observer.

2. Numerical Stability

2.1. Fundamentals and Annotation

As this paper is intended mostly for physicists and engineers, we outline a few basic concepts and annotation details common in numerical analysis, following the textbook *Accuracy and Stability of Numerical Algorithms* [12].

Evaluation of an expression in floating point arithmetic is denoted $fl(\cdot)$. A basic arithmetic operation op $\in \{+, -, *, /\}$ satisfies

$$fl(x \text{ op } y) = (x \text{ op } y)(1+\delta), \quad |\delta| \leq u. \tag{2}$$

Here, u is the machine precision, which is $2^{-52} \approx 1.11 \times 10^{-16}$ for the standard 64 bit Double floating point format, which is also used in this work. In contrast to the textbook, we use the common notation that bold symbols denote vectors, and also do not distinguish between accuracy and precision.

Computed quantities wear a hat, i.e., $\hat{f}$ denotes the computed value of an expression f. The absolute error of a computed quantity is defined as, $E_{\text{abs}}(\hat{f}) = |f - \hat{f}|$. In relation to the function value, it gives the important relative error

$$E_{\text{rel}}(\hat{f}) = \frac{|f - \hat{f}|}{|f|}. \tag{3}$$

The intuitive measure of *correct significant figures* of an evaluation is a flawed concept, as explained in the textbook, but is used here nonetheless by rounding the relative error. In most cases, an evaluation of $\hat{f}$ experiences large relative errors about zero crossings of the function f, accompanied by a respective loss of significant figures. However, this is generally not a problem as long as the absolute error is reasonably bounded there, $E_{\text{abs}}(\hat{f}) \ll \epsilon$.

We speak of *numerical stability* when the relative error is small, or at zero crossings, when the absolute error is, respectively, smaller than the mean absolute function value in the vicinity of the crossing. In Double floating point format, proper implementations of analytic expressions are expected to give relative errors below 10^{-10} to 10^{-12}, or in the vicinity of zero-crossings, absolute errors that are at least 10–12 orders of magnitude smaller than the mean absolute function value.

When a vector field $\boldsymbol{g}$ is studied, this is done componentwise with (3), or vectorwise using the Euclidean norm $||\cdot||$,

$$E_{\text{rel}}(\hat{\boldsymbol{g}}) = \frac{||\boldsymbol{g} - \hat{\boldsymbol{g}}||}{||\boldsymbol{g}||}. \tag{4}$$

Such a measure can also mitigate the apparent relative error problem about zero-crossings of individual components.

Rounding happens when an internal operation results in a number that has more decimal places than is provided by the used floating point arithmetic. For example, the input $a = 8$ can be fully represented in Single (32 bit) precision, while the result of $b = a/3 = 2.6666667$ is rounded off. Rounding is by itself not a problem, since the error appears only in the last digit. The relative error according to (3) yields $E_{\text{rel}}(\hat{b}) = 1.25 \times 10^{-8}$, which is in the expected range for Single precision with 7–8 significant digits. However, this rounding error sets the stage for *numerical cancellation*. Consider that a number $c = 2.6666$ is subtracted from the previous result. The outcome is $d = b - c = 6.67 \times 10^{-5}$ and has only three significant figures. The resulting relative error increases to $E_{\text{rel}}(\hat{d}) = 5 \times 10^{-4}$ in this case. Further computation with this value must be treated with great care. Numerical cancellation happens when subtracting rounded values of equal size from each other, and can even lead to a complete loss of precision, i.e., no correct significant figures.

A power series can be assumed to be stable when implementing the sum with falling order (small terms first). The limit of precision of a series expansion is mostly a result of the chosen truncation.

In this work, all quantities are evaluated in Python v3.9.9, and making use of the standard packages Numpy v1.21.5 and Scipy v1.7.3. Many of the used implementations are reproduced in Appendix A.

2.2. The Classical Solution

The exact analytic expressions for the off-axis magnetic field of a circular current loop have been known for a long time. They can either be derived by direct integration over the current density, or by solving a boundary value problem. A few examples included in classical textbooks show expressions derived in cylindrical coordinates [5] and in spherical coordinates [13] via the vector potential. A solution via the boundary value problem is demonstrated in [14]. A detailed summary of analytic expressions for the current loop field and their derivatives in different coordinate systems is given in [15]. More recent developments show a computation through the law of Biot–Savart where the result is expressed in terms of hypergeometric functions [16]. In [17], the field of arc segments is reviewed and expressed in terms of incomplete elliptic integrals, where the special case of a complete current loop is shown to yield the classical results.

Two recent works focus on computationally efficient approximation schemes: Prantner et al. [18] use local Taylor series approximations about supporting points. This has the advantage that the resulting expressions are simple, numerically stable and fast to compute. On the other hand, the validity is limited to small regions about the supporting points, and for each new region, a new series must be constructed. A very elegant solution was proposed by Chapman et al. [19]. They make use of a binomial expansion and reduce the field to a simple series with a high level of accuracy everywhere. Their work aligns strongly with this paper, is reproduced in Section 2.6, and analyzed in terms of performance in Section 3.

Hints towards stability problems of the classical solutions are found in [20], where the authors use a series expansion to "obtain simpler expressions valid at intermediate distances", and in [21], where various analytic representations are derived and studied. Proper numerical implementation, numerical stability and computation performance are hardly discussed in the literature beyond basic algorithms for the complete elliptic integrals and Taylor series expansions.

All works that offer exact analytic forms for the field of the current loop end up with the same expressions in cylindrical coordinates, or with expressions that are easily transformed to the following one

$$B_\rho^{\text{Classic}} = B_0 \frac{k}{\sqrt{\bar\rho}} \frac{\bar z}{\bar\rho} \frac{1}{q^2} \underbrace{\left[(1+q^2)E(k^2) - 2q^2K(k^2)\right]}_{\xi_0(k,q)}, \tag{5a}$$

$$B_z^{\text{Classic}} = B_0 \frac{k}{\sqrt{\bar\rho}} \frac{1}{q^2} \underbrace{\left[2q^2K(k^2) - \left(1+q^2-\frac{k^2}{\bar\rho}\right)E(k^2)\right]}_{\xi_1(\bar\rho,k,q)} \tag{5b}$$

$$\text{with} \quad k^2 = \frac{4\bar\rho}{\bar z^2 + (1+\bar\rho)^2}, \qquad q^2 = \frac{\bar z^2 + (1-\bar\rho)^2}{\bar z^2 + (1+\bar\rho)^2}.$$

The loop lies in the $z = 0$ plane of a cylindrical coordinate system (ρ, φ, z), with the origin at its center. B_ρ and B_z denote radial and axial component of the B-field, respectively. Everything is expressed in dimensionless quantities, $\bar\rho = \rho/\rho_0$ and $\bar z = z/\rho_0$ with the loop radius ρ_0, and $B_0 = \mu_0 i_0/8\pi\rho_0$ with the vacuum permeability μ_0. The functions K and E denote the complete elliptic integrals of first and second kind, respectively, for which fast and stable algorithms exist [6,10].

The formulation is chosen through the two dimensionless quantities k and q that satisfy

$$q^2 = 1 - k^2. \tag{6}$$

While k and q can be expressed through each other, it is important that they are computed individually to avoid numerical cancellation effects. Specifically, a pure k-formulation results in additional problems about the singularity at $\bar\rho = 1, \bar z = 0$, while a pure q-formulation adds to the main instability described in the next section.

The system is independent of the azimuth angle φ, so that the parameter space of interest, covering all of $\mathbb{R}^3$ except the loop itself, is,

$$\left\{\bar\rho \in [0,\infty), \bar z \in (-\infty,\infty) : \bar\rho \neq 1 \vee \bar z \neq 0\right\} \qquad \rightarrow \qquad k, q \in [0,1]. \tag{7}$$

Due to the symmetry, we only study positive values of $\bar z$.

2.3. Numerical Stability of the Classical Solution

By straightforward implementation, we mean naive transfer of a function to the computer script. A code example for such an implementation $\hat{\boldsymbol{B}}^{\text{Classic}}$ of Equation (5) is provided in Appendix A.1. This implementation is component- and vectorwise numerically unstable, which can be seen from the relative errors shown in Figure 3. An increasing loss of precision with distance from the loop ($\bar\rho^2 + \bar z^2 \gg 1$) can be observed in all subplots. In addition, the radial component becomes unstable towards the axis ($\bar\rho \ll 1$). This instability translates partially to the vectorwise error in (c), despite the dominating amplitude of the stable B_z^{Classic}. The troublesome "speckled" region in (c) is the result of the competing error between the two components. Finally, the relative error diverges naturally in a narrow band about the zero crossing of the z-component, outlined with a dotted line in (b). All error computations in this work are achieved by comparison to stable forms that are derived below.

The origin of the observed instabilities is numerical cancellation when evaluating the functions ξ_0 and ξ_1 in (5). Both functions are made up of two summands that tend to π and $-\pi$, respectively, for $k \to 0$, which corresponds to regions close to the axis and far from the loop, as can be seen from Figure 3d.

The cancellation effect is best demonstrated with the function ξ_0. Figure 4a shows a straightforward implementation of ξ_0, and an implementation of its Taylor series for small k including eight terms up to order k^{18}. The fluctuation of $\hat\xi_0$ with decreasing k is

a result of the numerical cancellation, which ends in a complete loss of accuracy below $k^2 \approx 3.5 \times 10^{-8}$. Loss of precision resulting from the truncation of the Taylor expansion, as k approaches 1, is made visible in the inset figure.

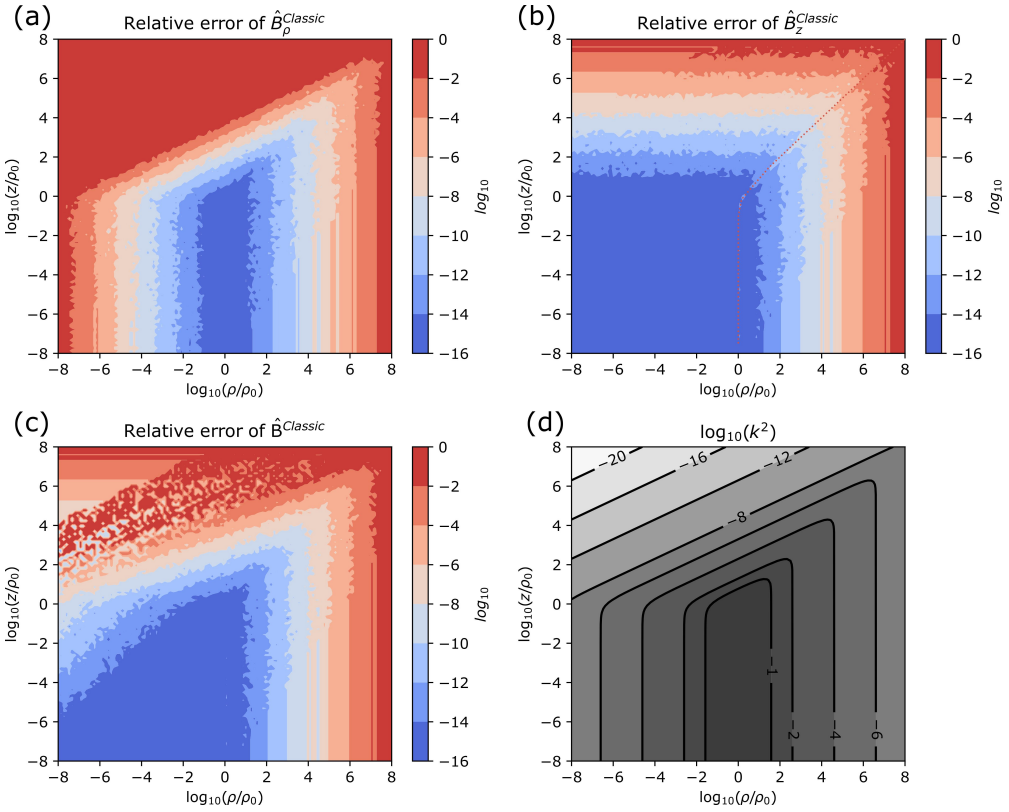

Figure 3. Relative error of a straightforward implementation of the textbook expressions for the B-field of a current loop. (**a**) Radial and (**b**) axial components, as well as (**c**) vectorwise analysis reveal a high level of numerical instability. (**d**) The relation between the cylindrical coordinates and the important quantity k.

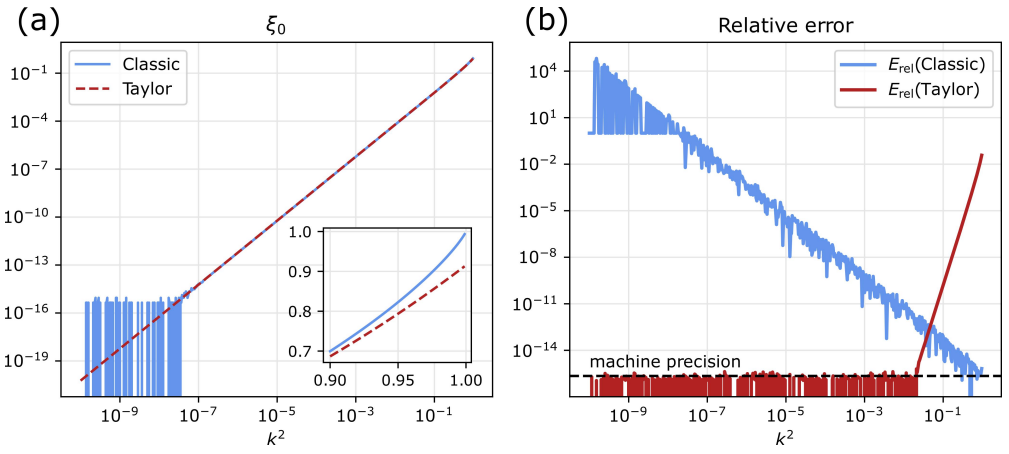

Figure 4. (**a**) Two different implementations of the function ξ_0 and (**b**) the relative difference when comparing them to each other.

Relative errors of the two implementations are shown in Figure 4b. With decreasing k the straightforward implementation loses precision. Below $k^2 \approx 3.5 \times 10^{-8}$ the relative error exceeds 1 which is understood as a complete loss of precision. On the other hand,

the accuracy of the Taylor series increases with decreasing k until the truncation error undercuts the machine precision. For small values of k, this Taylor series can be used as an excellent numerical reference.

All exact expressions for the magnetic field of a current loop that are provided in the literature seem to exhibit similar instabilities. This includes the tempting form in spherical coordinates [13,15]. Rearrangement of the problematic summands can partially reduce some cancellation effects, but the main problem remains.

2.4. Dipole Approximation

Dipole models are commonly used to describe the magnetic field at large distances from the sources. They are numerically stable, easy to implement and fast to compute. Specifically for current loops, dipole models are used when coupling distant coils [22], or in some geomagnetic works [23,24]. The relation between the magnetic dipole moment and the current loop parameters is $\boldsymbol{m} = (0, 0, i_0 \rho_0^2 \pi)$. The resulting dipole field in cylindrical coordinates is given by

$$\boldsymbol{B}^{\text{Dipole}} = B_0 \frac{2\pi}{(\bar{\rho}^2 + \bar{z}^2)^{5/2}} \begin{pmatrix} 3\bar{\rho}\bar{z} \\ 0 \\ 2\bar{z}^2 - \bar{\rho}^2 \end{pmatrix}. \tag{8}$$

Straightforward implementations of (8) are numerically stable, so that the precision of the dipole model is limited only by the approximation error. The vectorwise relative error is shown in Figure 5a. It can be observed that the accuracy of the dipole approximation increases only slowly with distance from the loop. At $\bar{\rho} = 100$, the relative error is still above 10^{-4}. The large difference between current loop and dipole model is also noted in [20].

The colorful contour lines in the figure correspond to the vectorwise relative error of $\hat{\boldsymbol{B}}^{\text{Classic}}$, and are taken from Figure 3c. While the dipole approximation saves the day at large distances, the figure also reveals that both computations fail to give correct results in a large region of interest close to the axis.

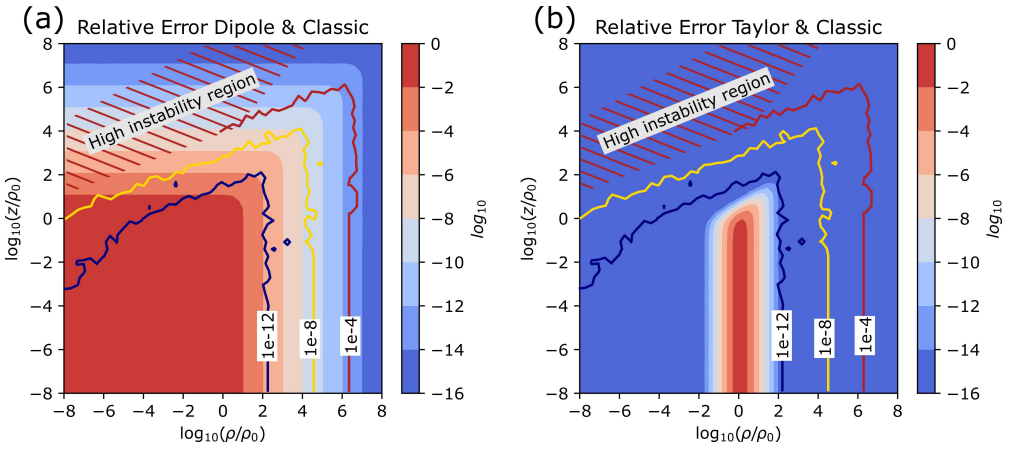

Figure 5. Vectorwise relative error of the dipole approximation (**a**) and a Taylor approximation (**b**) of the B-field of a current loop. The colored contour lines show the numerical error from a straightforward implementation of the classical textbook expressions.

2.5. Taylor Approximation

In Section 2.3, it is explained that the classical textbook expressions suffer from the numerical instabilities of $\xi_0(k,q)$ and $\xi_1(\bar{\rho},k,q)$ for small values of k. A series expansion seems like a natural choice. Taylor approximations of (5a) and (5b) about $k=0$ calculate as

$$B_\rho^{\text{Taylor-k}} = B_0 \frac{\bar{z}}{\bar{\rho}^{3/2}} \frac{3\pi}{16q^2}\left[k^5 + \frac{1}{4}k^7 + \frac{15}{128}k^9 + \mathcal{O}(k^{11})\right], \tag{9a}$$

$$B_z^{\text{Taylor-k}} = B_0 \frac{1}{\bar{\rho}^{3/2}} \frac{\pi}{2q^2}\left[k^3 - \frac{2+3\bar{\rho}}{8}k^5 - 3\frac{1+3\bar{\rho}}{64}k^7 + \mathcal{O}(k^9)\right]. \tag{9b}$$

The vectorwise error of such a series expansion with up to order k^{21} (10 terms) is shown in Figure 5b. Such an expansion with a simple cutoff criterion $k_c = 0.26$ can be combined with the classical solution (5) to yield at least 10 significant digits everywhere.

Moreover, it is possible to provide a second Taylor series about $q=0$, which is accurate when the k-series fails,

$$B_\rho^{\text{Taylor-q}} = B_0 \frac{k\bar{z}}{\bar{\rho}^{3/2}}\left[\frac{1}{q^2} + \frac{3}{4}\ln\left(\frac{eq^2}{16}\right) + \frac{3}{64}\ln\left(\frac{e256}{q^4}\right)q^2 + \mathcal{O}(q^4)\right], \tag{10a}$$

$$B_z^{\text{Taylor-q}} = B_0 \frac{k}{\bar{\rho}^{3/2}}\left[(1-\bar{\rho})\frac{1}{q^2} - \frac{1}{4}\left(4 + \ln\left(\frac{eq^2}{16}\right)(1+3\bar{\rho})\right) + \mathcal{O}(q^2)\right]. \tag{10b}$$

Here e denotes Euler's number. It is shown in Figure 6 that it is possible to cover all of $\mathbb{R}^3$ by overlapping a k-series and a q-series. Specifically, two such series that include terms up to order q^{60} and k^{61} with cutoff criterion $k_c = 0.7$ globally undercut a componentwise relative error of 10^{-10}. The respective $k_c = 0.7$ contour is shown in Figure 6a,b.

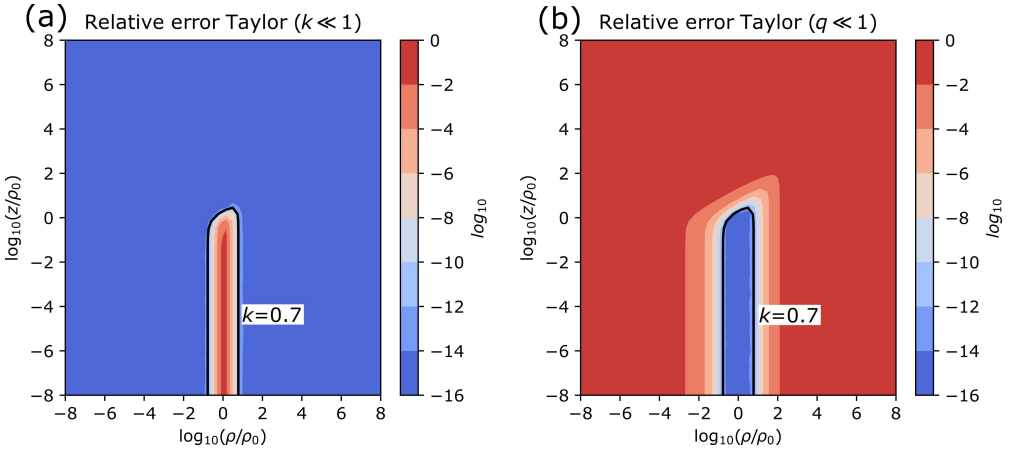

Figure 6. Vectorwise relative error of Taylor approximations of the B-field of a current loop. (**a**) Expansion for small k. (**b**) Expansion for small q.

Proper implementations of these series are provided in Appendix A.2, together with detailed information about the series precision with respect to the chosen truncation. Computation performance is discussed in Section 3.

2.6. Binomial Expansion

It is proposed in [19] to make use of a binomial expansion to generate the following series approximation for the B-field of a current loop,

$$B_\rho^{\text{Binom}} = B_0 \frac{2\pi\bar{z}\bar{\rho}}{(\bar{\rho}^2+\bar{z}^2+1)^{5/2}} \sum_i A_i F_i, \tag{11a}$$

$$B_z^{\text{Binom}} = B_0 \frac{2\pi\bar{\rho}}{(\bar{\rho}^2+\bar{z}^2+1)^{5/2}} \sum_i \left(\frac{4}{W^{1/2}} B_i - \bar{\rho} C_i \right) F_i. \tag{11b}$$

The quantity W is closely related to k^4,

$$W = \frac{4\bar{\rho}^2}{(\bar{\rho}^2+\bar{z}^2+1)^2}, \tag{12}$$

and the functions F_i and coefficients A_i for the lowest five orders are given by

$$\begin{aligned} F_0 &= 1, & A_0 &= 3, \\ F_1 &= W/(1-W), & A_1 &= 3.601265264628424, \\ F_2 &= (\ln(1-W)+W)/W, & A_2 &= 7.270215646065023 \times 10^{-1}, \\ F_3 &= \ln(1-W), & A_3 &= -5.22255035327797 \times 10^{-2}, \\ F_4 &= W, & A_4 &= -8.69531084533186 \times 10^{-3}, \\ F_5 &= W^2, & A_5 &= -1.4574683941872 \times 10^{-3}. \end{aligned}$$

The corresponding coefficients B_i and C_i are

$$\begin{aligned} B_0 &= 1, & C_0 &= 3, \\ B_1 &= 9.003162495383443 \times 10^{-1}, & C_1 &= 3.601264998153377, \\ B_2 &= -3.044435633189330 \times 10^{-2}, & C_2 &= 7.276276696916009 \times 10^{-1}, \\ B_3 &= -2.5946696478408 \times 10^{-2}, & C_3 &= -5.27599282256922 \times 10^{-2}, \\ B_4 &= -3.9595895609185077 \times 10^{-3}, & C_4 &= -8.92318142802500 \times 10^{-3}, \\ B_5 &= -7.177536438 \times 10^{-4}, & C_5 &= -1.5500658123898 \times 10^{-3}. \end{aligned}$$

Here, we have simply reproduced the results from Chapman et al. in a format that aligns with this work. Coefficient C_3 has a different sign due to a typo in the original publication.

When implementing the above expressions, great care must be taken with the functions F_2 and F_3, as they are numerically unstable for small values of W. For this purpose, special implementations, commonly termed $log1p$ and $log1pmx$, can be used. The resulting accuracy is displayed in Figure 7, and confirms the claims made in [19], that more than five significant digits are achieved everywhere with this approximation.

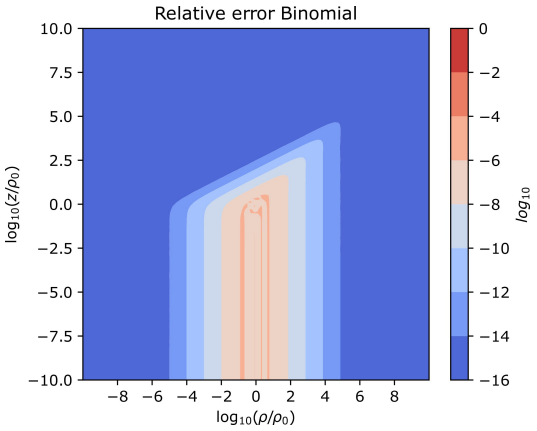

Figure 7. Vectorwise relative error of the binomial expansion for the B-field of a current loop.

2.7. An Exact and Stable Representation

Bulirsch's general complete elliptic integral [25] is defined as

$$cel(k_c, p, c, s) = \int_0^{\pi/2} \frac{c\cos^2\varphi + s\sin^2\varphi}{(\cos^2\varphi + p\sin^2\varphi)\sqrt{\cos^2\varphi + k_c^2\sin^2\varphi}} d\varphi. \tag{13}$$

An efficient algorithm exists for the computation of *cel*. Its original implementation by Bulirsch [25] is reviewed in [26] and is reproduced here in Appendix A.3. A slightly faster, but more verbose implementation by Fukushima can be found in [27]. Algorithm convergence and performance is discussed in Section 3.

It is pointed out in [28] that sums of complete elliptic integrals can be expressed in terms of *cel*,

$$\lambda K(m) + \mu E(m) = cel(\sqrt{1-m}, 1, \lambda + \mu, \lambda + \mu(1-m)). \tag{14}$$

This identity enables us to re-express the troublesome functions ξ_0 and ξ_1 in the form of

$$\xi_0(k, q) = cel(q, 1, k^2, -k^2q^2), \tag{15}$$

$$\xi_1(\bar{\rho}, k, q) = cel(q, 1, k^2(1 - 1/\bar{\rho}), -k^2q^2(1 + 1/\bar{\rho})). \tag{16}$$

Again, we have chosen a favorable argument representation through k and q to avoid instabilities about the singularity at $\bar{z} = 0$ and $\bar{\rho} = 1$.

In principle, the transformation from E and K to *cel* eliminates the numerical cancellation problems of ξ_0 and ξ_1. However, the implementations by Bulirsch and by Fukushima become themselves numerically unstable for small values of k when evaluating (15) and (16).

A closer look at the algorithm in Appendix A.3 reveals the problem: At first (line 7–34), a set of parameters is prepared from the input arguments, thereon (line 35–44) the solution is computed iteratively. For the input arguments in (15), it happens that line 28 becomes $cc = k^2 - k^2q^2$, which exhibits cancellation for $k \ll 1$. This problem is easily fixed by rewriting the expression as $cc = k^4$ in that line. Similar problems, which can be solved by algebraic transformations in the first part of the algorithm, happen also when computing ξ_1 in (16).

In Appendixes A.4 and A.5, we show modified *cel* algorithms, termed $cel^*(k, q)$ and $cel^{**}(\bar{\rho}, k, q)$ for computing ξ_0 and ξ_1, respectively, that are stable and fast. For improved performance, the first part of Bulirsch's original algorithm is shortened down by assigning all parameters their respective values, and the improvements proposed by Fukushima [27] are adapted. Comparisons to overlapping Taylor series expansions, introduced in Section 2.5 confirm the global numerical stability of the two algorithms.

As all other terms in (5) are numerically stable, the following expressions will give the components of the B-field of a current loop with machine precision everywhere,

$$B_\rho^{\text{Stable}} = B_0 \frac{\bar{z}}{\bar{\rho}} \frac{k}{q^2\sqrt{\bar{\rho}}} cel^*(k, q), \tag{17a}$$

$$B_z^{\text{Stable}} = B_0 \frac{k}{q^2\sqrt{\bar{\rho}}} cel^{**}(\bar{\rho}, k, q). \tag{17b}$$

2.8. Loss of Precision at Sign Change

The z-component of the B-field exhibits a sign change when passing from small to large values of $\bar{\rho}$. In (17b), the sign change is revealed by the following relation:

$$cel^{**}(\bar{\rho}, k, q) = \frac{k^2}{\bar{\rho}} E(k^2) - cel^*(k, q). \tag{18}$$

Both functions E and cel^* are positive. Towards the coordinate axis, the first term dominates, while for large values of $\bar{\rho}$, the second term does. The zero-crossing in-between corresponds to the change in sign.

The relative error of the z-component naturally diverges around the zero-crossing. As explained in Section 2.1, we do not speak of numerical instability in this case, however, a loss of significant digits of the z-component still occurs. How fast the loss of significant digits occurs depends on the gradient of the field. In this case, a helpful criterion is found empirically

$$\text{lost digits} \approx \log_{10} |B_\rho / 2B_z|, \tag{19}$$

which corresponds only to a very narrow band about the crossing.

3. Performance

Analytic solutions are mostly used for their computation efficiency, so that the performance of the proposed algorithms is critical. For a performance comparison on equal footing, all algorithms are implemented for scalar inputs in native Python, and making use of the standard *math* package. This means that only sequential execution is considered, the potential for code vectorization is not analyzed here. Algorithms for the complete elliptic integrals, that can also be found in the Scipy library, have been reproduced. All tests were run on CentOS Linux 7.9.2009 operating system on a AMD EPYC 7513 2.6 GHz CPU architecture with 32 cores.

The performance test is designed as follows: Each B-field expression is evaluated for input parameters $(\bar{\rho}_0, \bar{z}_0)$. This is repeated for a set of 200 different points, that all correspond to a similar value of k. The points are equally distributed on the respective k^2 contour lines that are shown in Figure 3d. This procedure is repeated in a loop 1000 times. For each value of k, every expression is thus evaluated 200,000 times. The order of evaluation is mixed up by design, to avoid distortions from computation performance spikes resulting from Python garbage collection and hardware issues. For each k, the computation times of every tested expression are summed up.

The results of this test are shown in Figure 8a for different values of $k^2 \in [10^{-12}, 1)$. Computation times are displayed with respect to the fastest one, which is always the dipole expression. Most algorithms evaluate the same expression independent of k, and show similar timings for all inputs. The proposed stable algorithm is of an iterative nature, and requires less iterations for smaller values of k. The steps from four iterations close to the loop filament down to zero iterations are clearly visible in the figure. The binomial approximation exhibits a step at about $k^2 \approx 0.2$, which results from the special implementation of $log1pmx$ that was used here. This function is not available in Python *math*, and makes use of a 4-term power series for small W.

In Figure 8b, we show the maximal vectorwise relative computation errors of the different methods. Interestingly, convergence and divergence of most methods can be observed well as functions of k^2. For the dipole approximation only positions with $\bar{\rho} \geq 1$ on the k^2 contour lines are included, because it does not converge close to the origin. The stable solution is not visible in this figure, because it is used as a reference. It has machine precision everywhere.

The combination of the two figures enables users to choose the scheme that is most suitable for their purpose. When only computation time and accuracy are taken as measures, Figure 8 shows that the proposed method is extremely competitive. The only solutions that are faster and have similar precision are the Taylor k-series and the dipole approximation, both with limited validity. However, in many cases users might not require machine precision, and can be compelled by the simple analytic expressions offered by the approximation schemes, which also enable effortless computation of the derivatives.

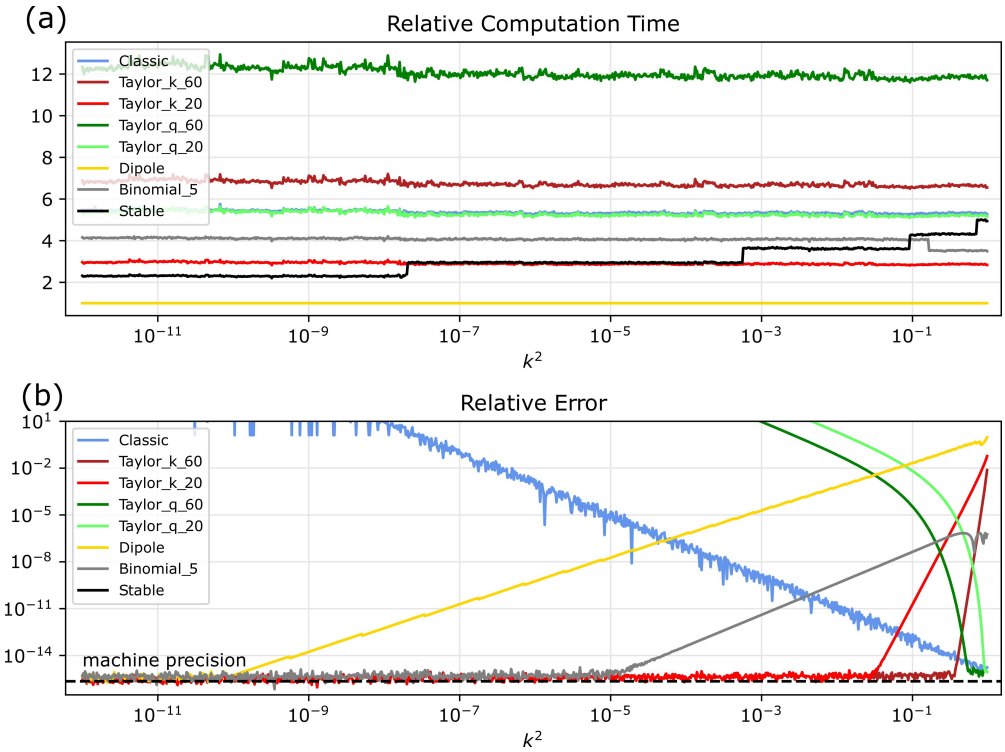

Figure 8. (**a**) Computation times of various methods with respect to the fastest one (dipole) as a function of k^2. (**b**) The respective vectorwise relative errors.

4. Discussion and Conclusions

We have studied straightforward implementations of the textbook expressions for the B-field of a current loop (5), and have found that these commonly used forms are numerically troublesome at large distances from the loop and close to the axis. The computationally fast dipole approximation (8) appears to be an unsatisfactory option, as it converges very slowly and solves the problem only at large distances. Instead, a well-chosen Taylor series approximation (9) with a simple cutoff criterion can mitigate the instability globally. It is also possible to cover all of $\mathbb{R}^3$ with two overlapping Taylor series (9) and (10). However, to achieve good precision, the series must include a high order, which makes them computationally slow. A recently proposed binomial expansion (11) offers simple expressions and excellent computation performance. The only downside is the limited accuracy (more than five significant figures) in the vicinity of the loop. The real advancement of this paper is a novel, exact, numerically stable representation in terms of modified Bulirsch *cel* functions (17). Implementations based thereon offer machine precision everywhere and much faster computation times than the classical solution.

For achieving numerical stability, series approximations are always an interesting option. However, there are several arguments that speak for the proposed *cel* approach: The approximation error of the series is a result of truncation and cannot improve unless more terms are included, while the machine precision limit of the proposed *cel* algorithm depends only on the chosen floating point arithmetic. This means that a 128 bit implementation will give twice as many correct significant figures as a 64 bit implementation when using *cel*, while the series will not improve. Of course, this comes at the cost of more automatic iterations. Secondly, most approximations have a limited region of validity which requires cut-off criteria and treating multiple cases. This is problematic because in addition to programming complexity, case splitting hinders code vectorization, which can be a serious performance issue. Moreover, a *cel*-based implementation can further reduce computation time by combining the time consuming iterative parts of both field components in a single vectorized operation.

The biggest advantage of the approximation schemes is their simple form, which makes it easy to find analytic expressions for their derivatives. While it is also possible to determine derivatives of the proposed *cel* expressions, making them numerically stable seems quite laborious.

Similar stability problems of analytic forms can be found throughout the literature when systems of cylindrical symmetry are involved. Specific examples are expressions found for the axially magnetized cylinder [26], the diametrically magnetized cylinder [29], and homogeneously magnetized ring segments [30]. It is very likely that the arc solutions [17] also suffer from this problem. A similar treatment should be attempted for the extremely useful expressions in these works.

Finally, we are happy to announce that the proposed implementation of the magnetic field of a current loop, based on the cel^* and cel^{**} algorithms, was adopted into the open-access Python package Magpylib version 4.2.0 and is already available to the general public.

Author Contributions: Conceptualization, methodology, validation and investigation were performed by M.O. and P.L.; original draft preparation, visualization, administration and funding were conducted by M.O.; writing, review and editing, formal analysis were done by P.L. and F.S. All authors have read and agreed to the published version of the manuscript.

Funding: This project has been supported by the COMET K1 centre ASSIC Austrian Smart Systems Integration 100 Research Center.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: This project has been supported by the COMET K1 centre ASSIC Austrian Smart Systems Integration 100 Research Center. The COMET—Competence Centers for Excellent Technologies—Program is supported by BMVIT, 101 BMDW and the federal provinces of Carinthia and Styria. Feedback and discussions with Christoph Ortner on numerical analysis were highly appreciated. Thanks to Github users @fwessel and @Blue-Spartan for pointing out numerical stability issues in the Magpylib computations. Special thanks to Alexandre Boisellet for his support of the Magpylib project.

Conflicts of Interest: The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Appendix A. Algorithms

All algorithms in this section are written in Python 3.9. They are presented in a scalar form for readability.

Appendix A.1. Straightforward Implementation

The following code shows a naive implementation of the classical expressions (5) of the B-field of a current loop in cylindrical coordinates.

```
import math as m
from scipy.special import ellipe, ellipk

def B(rho, z):
    """
    B-field of current loop in cylindrical coordinates in units of (mT).
    """
    z2 = z**2
    x0 = (z2 + (rho+1)**2)

    k2 = 4*rho/x0
    q2 = (z2 + (rho-1)**2) / x0

    k = m.sqrt(k2)
    E = ellipe(k2)
```

```
    K = ellipk(k2)

    x1 = k/m.sqrt(rho)/q2/20
    x2 = (1+q2)*E - 2*q2*K

    Br = x1*z/rho * x2
    Bz = - x1*(x2 - k2/rho*E)

    return (Br, 0, Bz)
```

Appendix A.2. Taylor Series Implementations

The following code is an implementation of the Taylor series expansion (9a) with orders up to k^{61}.

```
import math as m

def Br_taylor_k(rho, z):
    """
    Taylor series of the radial component of the B-field of a current loop for
        small k up to order k**61.
    Inputs: Radial and axial positions, rho = rho/rho0 and z = z/rho_0.
    Output: B-field in units of millitesla
    """
    z2 = z**2
    x0 = (z2 + (rho+1)**2)
    k2 = 4*rho/x0
    q2 = (z2+(rho-1)**2)/x0
    k = m.sqrt(k2)

    P = (
        4.348319233471567e-05, # k**61
        4.660399561232684e-05,
        5.007331535430794e-05,
        5.394505893886228e-05,
        5.828397724486921e-05,
        6.316839218549437e-05, # k**51
        6.869375982346906e-05,
        7.497737506313522e-05,
        8.216465605330530e-05,
        9.043764518687763e-05,
        1.000266678574197e-04, # k**41
        1.112265650306444e-04,
        1.244196727442359e-04,
        1.401089569905765e-04,
        1.589667810126675e-04,
        1.819083471102810e-04, # k**31
        2.102052011052137e-04,
        2.456659045960062e-04,
        2.909335350495146e-04,
        3.499952301347544e-04,
        4.290963192983367e-04, # k**21
        5.384738124528147e-04,
        6.958738499390220e-04,
        9.343201341838618e-04,
        1.321260795815562e-03,
        2.013349784099904e-03, # k**11
        3.451456772742693e-03,
        7.363107781851078e-03,
        2.945243112740431e-02, # k**5
    )

    result = P[0]
    for p in P[1:]:
        result *= k2
        result += p
    result *= k2*k2*k*z/rho**(3/2)/q2

```

```
    return result
```

The following code is an implementation of the Taylor series expansion (10a) with orders up to q^{60}.

```
import math as m

def Br_taylor_q(rho, z):
    """
    Taylor series of the radial component of the B-field of a current loop for
        small q up to order q**60.
    Inputs: Radial and axial positions, rho = rho/rho0 and z = z/rho_0.
    Output: B-field in units of millitesla
    """
    z2 = z**2
    x0 = (z2 + (rho+1)**2)
    k2 = 4*rho/x0
    q2 = (z2+(rho-1)**2)/x0
    k = m.sqrt(k2)
    log_q2 = m.log(q2)

    P = (
        2.140383169962052e-08 - 2.157384139784586e-07 * log_q2, # q**60
        2.448605439302092e-08 - 2.386401724650211e-07 * log_q2,
        2.814368061953632e-08 - 2.649020096262435e-07 * log_q2,
        3.251058034393473e-08 - 2.951635428013855e-07 * log_q2,
        3.775841577054470e-08 - 3.302162609809064e-07 * log_q2,
        4.410934326064394e-08 - 3.710473232630592e-07 * log_q2, # q**50
        5.185372706900618e-08 - 4.188984109787034e-07 * log_q2,
        6.137516212651031e-08 - 4.753457145857628e-07 * log_q2,
        7.318631654988898e-08 - 5.424099936978626e-07 * log_q2,
        8.798105223936001e-08 - 6.227100551585218e-07 * log_q2,
        1.067114704413582e-07 - 7.196799136541265e-07 * log_q2, # q**40
        1.307038624714910e-07 - 8.378809805536608e-07 * log_q2,
        1.618366787740300e-07 - 9.834587570977332e-07 * log_q2,
        2.028196779163440e-07 - 1.164823878536536e-06 * log_q2,
        2.576424314628448e-07 - 1.393689567281251e-06 * log_q2,
        3.323144988153871e-07 - 1.686689932371525e-06 * log_q2, # q**30
        4.361242251684836e-07 - 2.067972330876966e-06 * log_q2,
        5.838535308384648e-07 - 2.573476678424669e-06 * log_q2,
        7.998285803570591e-07 - 3.258245255466363e-06 * log_q2,
        1.125665818448870e-06 - 4.209410019484494e-06 * log_q2,
        1.635873678758604e-06 - 5.570347093453165e-06 * log_q2, # q**20
        2.471262120850365e-06 - 7.588088919874281e-06 * log_q2,
        3.915944146725158e-06 - 1.071259612217546e-05 * log_q2,
        6.591535804299384e-06 - 1.582168042659760e-05 * log_q2,
        1.200571004791862e-05 - 2.478361129760742e-05 * log_q2,
        2.434841408803401e-05 - 4.205703735351563e-05 * log_q2, # q**10
        5.769486681760943e-05 - 8.010864257812500e-05 * log_q2,
        1.750345560741787e-04 - 1.831054687500000e-04 * log_q2,
        8.433137044373718e-04 - 5.859375000000000e-04 * log_q2,
        1.534025963549897e-02 - 4.687500000000000e-03 * log_q2,
       -6.647207708399180e-02 + 3.750000000000000e-02 * log_q2, # q**0
        5.000000000000000e-02,
    )

    result = P[0]
    for p in P[1:]:
        result *= q2
        result += p
    result *= k*z/rho**(3/2)/q2

    return result
```

The following code is an implementation of the Taylor series expansion (9b) with orders up to k^{61}.

```
import math as m

def Bz_taylor_k(rho, z):
```

```
    """
    Taylor series of the axial component of the B-field of a current loop for
        small k up to order k**61.
    Inputs: Radial and axial positions, rho = rho/rho0 and z = z/rho_0.
    Output: B-field in units of millitesla
    """
    z2 = z**2
    x0 = (z2 + (rho+1)**2)
    k2 = 4*rho/x0
    q2 = (z2+(rho-1)**2)/x0
    k = m.sqrt(k2)

    P = (
        4.348319233471567e-05 * rho + 1.499420425335023e-05, # k**61
        4.660399561232684e-05 * rho + 1.608947467568427e-05,
        5.007331535430794e-05 * rho + 1.730929419655089e-05,
        5.394505893886228e-05 * rho + 1.867328963268310e-05,
        5.828397724486921e-05 * rho + 2.020511211155466e-05,
        6.316839218549437e-05 * rho + 2.193346950885221e-05, # k**51
        6.869375982346906e-05 * rho + 2.389348167772837e-05,
        7.497737506313522e-05 * rho + 2.612847918866833e-05,
        8.216465605330530e-05 * rho + 2.869241957417010e-05,
        9.043764518687763e-05 * rho + 3.165317581540717e-05,
        1.000266678574197e-04 * rho + 3.509707644119991e-05, # k**41
        1.112265650306444e-04 * rho + 3.913527288115264e-05,
        1.244196727442359e-04 * rho + 4.391282567443621e-05,
        1.401089569905765e-04 * rho + 4.962192226749586e-05,
        1.589667810126675e-04 * rho + 5.652152213783733e-05,
        1.819083471102810e-04 * rho + 6.496726682510037e-05, # k**31
        2.102052011052137e-04 * rho + 7.545827731982029e-05,
        2.456659045960062e-04 * rho + 8.871268777078002e-05,
        2.909335350495146e-04 * rho + 1.057940127452780e-04,
        3.499952301347544e-04 * rho + 1.283315843827433e-04,
        4.290963192983367e-04 * rho + 1.589245627030877e-04, # k**21
        5.384738124528147e-04 * rho + 2.019276796698055e-04,
        6.958738499390220e-04 * rho + 2.650947999767703e-04,
        9.343201341838618e-04 * rho + 3.633467188492796e-04,
        1.321260795815562e-03 * rho + 5.285043183262248e-04,
        2.013349784099904e-03 * rho + 8.388957433749600e-04, # k**11
        3.451456772742693e-03 * rho + 1.533980787885641e-03,
        7.363107781851078e-03 * rho + 3.681553890925539e-03,
        2.945243112740431e-02 * rho + 1.963495408493621e-02,
                                    - 7.853981633974483e-02, # k**3
    )

    result = P[0]
    for p in P[1:]:
        result *= k2
        result += p
    result *= -k/rho**(3/2)/q2*k2

    return result
```

The following code is an implementation of the Taylor series expansion (10b) with orders up to q^{60}.

```
import math as m

def Bz_taylor_q(rho, z):
    """
    Taylor series of the axial component of the B-field of a current loop for
        small q up to order q**60.
    Inputs: Radial and axial positions, rho = rho/rho0 and z = z/rho_0.
    Output: B-field in units of millitesla
    """
    z2 = z**2
    x0 = (z2 + (rho+1)**2)
    k2 = 4*rho/x0
    q2 = (z2+(rho-1)**2)/x0
```

```
    k = m.sqrt(k2)
    log_q2 = m.log(q2)

    P = (
         2.140383169962052e-08 * rho + 5.756366599134164e-07, # q**60
         2.448605439302092e-08 * rho + 6.367692247077877e-07,
         2.814368061953632e-08 * rho + 7.068759571675434e-07,
         3.251058034393473e-08 * rho + 7.876663620944477e-07,
         3.775841577054470e-08 * rho + 8.812562039945226e-07,
         4.410934326064394e-08 * rho + 9.902847920909337e-07, # q**50
         5.185372706900618e-08 * rho + 1.118072627259595e-06,
         6.137516212651031e-08 * rho + 1.268835739807469e-06,
         7.318631654988898e-08 * rho + 1.447980732716223e-06,
         8.798105223936001e-08 * rho + 1.662516406570504e-06,
         1.067114704413582e-07 * rho + 1.921636483711170e-06, # q**40
         1.307038624714910e-07 * rho + 2.237557829363388e-06,
         1.618366787740300e-07 * rho + 2.626747507371085e-06,
         2.028196779163440e-07 * rho + 3.111754087360531e-06,
         2.576424314628448e-07 * rho + 3.723999925319969e-06,
         3.323144988153871e-07 * rho + 4.508141566084171e-06, # q**30
         4.361242251684836e-07 * rho + 5.529063837031901e-06,
         5.838535308384648e-07 * rho + 6.883443473579986e-06,
         7.998285803570591e-07 * rho + 8.719539805210148e-06,
         1.125665818448870e-06 * rho + 1.127244058408705e-05,
         1.635873678758604e-06 * rho + 1.492981081843003e-05, # q**20
         2.471262120850365e-06 * rho + 2.036144759732213e-05,
         3.915944146725158e-06 * rho + 2.879192411774280e-05,
         6.591535804299384e-06 * rho + 4.262260553943059e-05,
         1.200571004791862e-05 * rho + 6.700276866917860e-05,
         2.434841408803401e-05 * rho + 1.143628488115506e-04, # q**10
         5.769486681760943e-05 * rho + 2.201260551956201e-04,
         1.750345560741787e-04 * rho + 5.143424513214410e-04,
         8.433137044373718e-04 * rho + 1.748691113312115e-03,
         1.534025963549897e-02 * rho + 1.931709939249829e-02,
        -6.647207708399180e-02 * rho + 2.784264097200273e-02, # q**0
         5.000000000000000e-02 * rho - 5.000000000000000e-02, # q**-2
    )
    Q = (
         2.157384139784586e-07 * rho + 8.701449363797829e-06, # q**60
         2.386401724650211e-07 * rho + 9.306966726135822e-06,
         2.649020096262435e-07 * rho + 9.977975695921838e-06,
         2.951635428013855e-07 * rho + 1.072427538845034e-05,
         3.302162609809064e-07 * rho + 1.155756913433172e-05,
         3.710473232630592e-07 * rho + 1.249192654985633e-05, # q**50
         4.188984109787034e-07 * rho + 1.354438195497808e-05,
         4.753457145857628e-07 * rho + 1.473571715215865e-05,
         5.424099936978626e-07 * rho + 1.609149647970326e-05,
         6.227100551585218e-07 * rho + 1.764345156282479e-05,
         7.196799136541265e-07 * rho + 1.943135766866142e-05, # q**40
         8.378809805536608e-07 * rho + 2.150561183421063e-05,
         9.834587570977332e-07 * rho + 2.393082975604484e-05,
         1.164823878536536e-06 * rho + 2.679094920634033e-05,
         1.393689567281251e-06 * rho + 3.019660729109378e-05,
         1.686689932371525e-06 * rho + 3.429602862488768e-05, # q**30
         2.067972330876966e-06 * rho + 3.929147428666235e-05,
         2.573476678424669e-06 * rho + 4.546475465216915e-05,
         3.258245255466363e-06 * rho + 5.321800583928393e-05,
         4.209410019484494e-06 * rho + 6.314115029226741e-05,
         5.570347093453165e-06 * rho + 7.612807694385992e-05, # q**20
         7.588088919874281e-06 * rho + 9.358643001178280e-05,
         1.071259612217546e-05 * rho + 1.178385573439300e-04,
         1.582168042659760e-05 * rho + 1.529429107904434e-04,
         2.478361129760742e-05 * rho + 2.065300941467285e-04,
         4.205703735351563e-05 * rho + 2.943992614746094e-04, # q**10
         8.010864257812500e-05 * rho + 4.539489746093750e-04,
         1.831054687500000e-04 * rho + 7.934570312500000e-04,
         5.859375000000000e-04 * rho + 1.757812500000000e-03,
         4.687500000000000e-03 * rho + 7.812500000000000e-03,
        -3.750000000000000e-02 * rho - 1.250000000000000e-02, # q**0
```

```
        0.,
    )

    result = P[0] - Q[0]*log_q2
    for p,q in zip(P[1:], Q[1:]):
        result *= q2
        result += p - q*log_q2
    result *= -k/rho**(3/2)/q2

    return result
```

In most cases, the 30 provided terms will not be necessary. A relation between the number of terms included and precision of the Taylor series is shown in Table A1. The table gives values of k below (k-series) or above (q-series) which a series has a precision below the indicated number of significant figures when the indicated order is included in the sum. For example, an implementation of Br_taylor_k that includes up to order k^{21} has at least 8 digits of precision when $k \leq 0.46$. An implementation of Br_taylor_q that includes up to order q^{40} has at least 10 digits of precision when $k \geq 0.69$.

Table A1. Precision of various Taylor implementations in relation to the number of terms included in the respective sums.

Br_taylor_k ($k \leq$ value)						**Br_taylor_q** ($k \geq$ value)					
sig.figs	k-order included in series					sig.figs	q-order included in series				
	21	31	41	51	61		20	30	40	50	60
8	0.46	0.62	0.71	0.77	0.81	8	0.77	0.67	0.60	0.55	0.50
10	0.35	0.53	0.63	0.70	0.75	10	0.85	0.76	0.69	0.63	0.58
12	0.27	0.45	0.56	0.64	0.69	12	0.90	0.82	0.75	0.70	0.65
Bz_taylor_k ($k \leq$ value)						**Bz_taylor_q** ($k \geq$ value)					
sig.figs	k-order included in series					sig.figs	q-order included in series				
	21	31	41	51	61		20	30	40	50	60
8	0.43	0.59	0.68	0.74	0.78	8	0.86	0.76	0.70	0.64	0.59
10	0.33	0.50	0.61	0.68	0.73	10	0.91	0.83	0.76	0.71	0.66
12	0.26	0.43	0.54	0.62	0.68	12	0.94	0.87	0.81	0.76	0.72

Appendix A.3. Original Implementation of Bulirsch's cel Algorithm

The original implementation of Bulirsch's *cel* algorithm [25] is reproduced here. This implementation is also found in the first edition of the popular Numerical Recipes [31] and is commented on in [26]. Advantages of the algorithm over representations in terms of Carlson's functions are discussed in [32]. A slightly faster implementation by Fukushima is found in [27].

```
import math as m

def cel(kc, p, c, s):
    """
    Bulirsch's cel algorithm taken from Derby2010
    """
    if kc == 0:
        raise RuntimeError("FAIL")
    errtol = 0.000001
    k = abs(kc)
    pp = p
    cc = c
    ss = s
    em = 1.0
    if p > 0:
        pp = m.sqrt(p)
        ss = s / pp
```

```
    else:
        f = kc * kc
        q = 1.0 - f
        g = 1.0 - pp
        f = f - pp
        q = q * (ss - c * pp)
        pp = m.sqrt(f / g)
        cc = (c - ss) / g
        ss = -q / (g * g * pp) + cc * pp
    f = cc
    cc = cc + ss / pp
    g = k / pp
    ss = 2 * (ss + f * g)
    pp = g + pp
    g = em
    em = k + em
    kk = k
    while abs(g - k) > g * errtol:
        k = 2 * m.sqrt(kk)
        kk = k * em
        f = cc
        cc = cc + ss / pp
        g = kk / pp
        ss = 2 * (ss + f * g)
        pp = g + pp
        g = em
        em = k + em
    return (m.pi / 2) * (ss + cc * em) / (em * (em + pp))
```

*Appendix A.4. Implementation of cel**

Here, we show a modification of Bulirsch's *cel* algorithm for computing the function *cel**, described in Section 2.7. Input arguments are k^2 and q^2. A vectorized version of this algorithm is provided by the open-source library Magpylib [11].

```
import math as m

def celx(k2, q2):
    """
    Modified Bulirsch cel algorithm for computing
    xi0 = (1+q2)*E(k2)-2*q2*K(k2)
        = cel(q, 1, k2, -k2*q2)
    with 0 < k2 < 1, and q2 = 1-k2
    """
    qc = m.sqrt(q2)
    p  = 1 + qc
    g  = 1
    cc = k2 * k2
    ss = 2 * cc * (qc / (qc + 1) )
    em = p
    kk = qc

    while m.fabs(g - qc) >= g * 1e-8:
        qc = 2 * m.sqrt(kk)
        kk = qc * em
        f  = cc
        cc = cc + ss / p
        g  = kk / p
        ss = 2 * (ss + f * g)
        p  = p + g
        g  = em
        em = em + qc
    return 1.5707963267948966 * (ss + cc * em) / (em * (em + p))
```

*Appendix A.5. Implementation of cel***

Here, we show a modification of Bulirsch's *cel* algorithm for computing the function *cel***, described in Section 2.7. Input arguments are $\bar{\rho}$, k^2 and q^2. A vectorized version of this algorithm is provided by the open-source library Magpylib [11].

```
import math as m

def celxx(rho, k2, q2):
    """
    Modified Bulirsch cel algorithm for computing
    xi1 = (1+q2-k2/rho)*E(k2) - 2*q2*K(k2)
        = cel(q, 1, k2*(1-1/rho), -k2*q2*(1+1/rho))
    with 0 < k2 < 1, and q2 = 1-k2
    """
    qc = m.sqrt(q2)
    p = 1 + qc
    g = 1
    cc = k2*(k2-(q2+1)/rho)
    ss = 2 * k2 * qc * (k2/(1+qc) - (qc+1)/rho)
    em = p
    kk = qc

    while m.fabs(g - qc) >= g * 1e-8:
        qc = 2 * m.sqrt(kk)
        kk = qc * em
        f  = cc
        cc = cc + ss / p
        g  = kk / p
        ss = 2 * (ss + f * g)
        p  = p + g
        g  = em
        em = em + qc
    return 1.5707963267948966 * (ss + cc * em) / (em * (em + p))
```

References

1. Madenci, E.; Guven, I. *The Finite Element Method and Applications in Engineering Using ANSYS®*; Springer: Berlin/Heidelberg, Germany, 2015.
2. Pryor, R.W. *Multiphysics Modeling Using COMSOL®: A First Principles Approach*; Jones & Bartlett Publishers: Burlington, MA, USA, 2009. 2009.
3. Alnæs, M.; Blechta, J.; Hake, J.; Johansson, A.; Kehlet, B.; Logg, A.; Richardson, C.; Ring, J.; Rognes, M.E.; Wells, G.N. The FEniCS project version 1.5. *Arch. Numer. Softw.* **2015**, *3*, 9–23.
4. Schöberl, J. *C++ 11 Implementation of Finite Elements in NGSolve;* Institute for Analysis and Scientific Computing, Vienna University of Technology: Vienna, Austria 2014; Volume 30.
5. Smythe, W.B. *Static and Dynamic Electricity*; Hemisphere Publishing: New York, NY, USA, 1988.
6. Moshier, S.L.B. *Methods and Programs for Mathematical Functions*; Ellis Horwood Ltd Publisher: Chichester, UK, 1989.
7. ALGLIB. Available online: https://www.alglib.net/download.php (accessed on 9 June 2022).
8. Wolfram, S. *The Mathematica Book*; Wolfram Research, Inc.: Champaign, IL, USA 2003; Volume 1.
9. MATLAB. *Version 7.10.0 (R2010a)*; The MathWorks Inc.: Natick, MA, USA, 2010.
10. Virtanen, P.; Gommers, R.; Oliphant, T.E.; Haberland, M.; Reddy, T.; Cournapeau, D.; Burovski, E.; Peterson, P.; Weckesser, W.; Bright, J.; et al. SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nat. Methods* **2020**, *17*, 261–272.
11. Ortner, M.; Bandeira, L.G.C. Magpylib: A free Python package for magnetic field computation. *SoftwareX* **2020**, *11*, 100466.
12. Higham, N.J. *Accuracy and Stability of Numerical Algorithms*; SIAM: Philadelphia, PA, USA, 2002.
13. Jackson, J.D. Classical electrodynamics. *Am. J. Phys.* **1999**, *67*, 841.
14. Ortner, M.; Filipitsch, B. Feedback of Eddy Currents in Layered Materials for Magnetic Speed Sensing. *IEEE Trans. Magn.* **2017**, *53*, 1–11.
15. Simpson, J.C.; Lane, J.E.; Immer, C.D.; Youngquist, R.C. *Simple Analytic Expressions for the Magnetic Field of a Circular Current Loop*; Technical Report; NASA, Kennedy Space Center: Merritt Island, FL, USA, 2001.
16. Behtouei, M.; Faillace, L.; Spataro, B.; Variola, A.; Migliorati, M. A novel exact analytical expression for the magnetic field of a solenoid. *Waves Random Complex Media* **2020**, *32*, 1977–1991.
17. González, M.A.; Cárdenas, D.E. Analytical Expressions for the Magnetic Field Generated by a Circular Arc Filament Carrying a Direct Current. *IEEE Access* **2020**, *9*, 7483–7495.

18. Prantner, M.; Parspour, N. Analytic multi Taylor approximation (MTA) for the magnetic field of a filamentary circular current loop. *J. Magn. Magn. Mater.* **2021**, *517*, 167365.
19. Chapman, G.H.; Carleton, D.E.; Sahota, D.G. Current Loop Off Axis Field Approximations with Excellent Accuracy and Low Computational Cost. *IEEE Trans. Magn.* **2022**, *58*, 1–6.
20. Seleznyova, K.; Strugatsky, M.; Kliava, J. Modelling the magnetic dipole. *Eur. J. Phys.* **2016**, *37*, 025203.
21. Schill, R.A. General relation for the vector magnetic field of a circular current loop: A closer look. *IEEE Trans. Magn.* **2003**, *39*, 961–967.
22. Urzhumov, Y.; Smith, D.R. Metamaterial-enhanced coupling between magnetic dipoles for efficient wireless power transfer. *Phys. Rev. B* **2011**, *83*, 205114.
23. Rong, Z.; Wei, Y.; Klinger, L.; Yamauchi, M.; Xu, W.; Kong, D.; Cui, J.; Shen, C.; Yang, Y.; Zhu, R.; et al. A New Technique to Diagnose the Geomagnetic Field Based on a Single Circular Current Loop Model. *J. Geophys. Res. Solid Earth* **2021**, *126*, e2021JB022778.
24. Alldredge, L.R. Circular current loops, magnetic dipoles and spherical harmonic analyses. *J. Geomagn. Geoelectr.* **1980**, *32*, 357–364.
25. Bulirsch, R. Numerical calculation of elliptic integrals and elliptic functions. III. *Numer. Math.* **1969**, *13*, 305–315.
26. Derby, N.; Olbert, S. Cylindrical magnets and ideal solenoids. *Am. J. Phys.* **2010**, *78*, 229–235.
27. Fukushima, T.; Kopeikin, S. Elliptic functions and elliptic integrals for celestial mechanics and dynamical astronomy. *Front. Relativ. Celest. Mech.* **2014**, *2*, 189–228.
28. Fukushima, T.; Ishizaki, H. Numerical computation of incomplete elliptic integrals of a general form. *Celest. Mech. Dyn. Astron.* **1994**, *59*, 237–251.
29. Caciagli, A.; Baars, R.J.; Philipse, A.P.; Kuipers, B.W. Exact expression for the magnetic field of a finite cylinder with arbitrary uniform magnetization. *J. Magn. Magn. Mater.* **2018**, *456*, 423–432.
30. Slanovc, F.; Ortner, M.; Moridi, M.; Abert, C.; Suess, D. Full analytical solution for the magnetic field of uniformly magnetized cylinder tiles. *J. Magn. Magn. Mater.* **2022**, *559*, 169482.
31. Press, W.H.; Teukolsky, S.A.; Vetterling, W.T.; Flannery, B.P. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*; Cambridge University Press: Cambridge, UK, 2007.
32. Reinsch, K.D.; Raab, W. Elliptic Integrals of the First and Second Kind—Comparison of Bulirsch's and Carlson's Algorithms for Numerical Calculation. In *Special Functions*; World Scientific: Singapore 2000; pp. 293–308.