DOI: 10.21105/joss.09386

Software

- Review
- Repository
- Archive

Editor: Richard Littauer

Reviewers:

- @ariostas
- @joeybernard

Submitted: 17 October 2025
Published: 23 January 2026

Ellip: An Elliptic Integral Library for Rust

Sira Pornsiriprasert [1,2¶]

1 Faculty of Medicine Ramathibodi Hospital, Mahidol University, Thailand **2** Center for Biomedical and Robotics Technology (BART LAB), Department of Biomedical Engineering, Faculty of Engineering, Mahidol University, Thailand ¶ Corresponding author

Summary

Ellip is an elliptic integral library implemented in Rust. Legendre's, Carlson's, and Bulirsch's forms are provided as generic-typed functions, compatible with no-std environments. The library is extensively tested for accuracy against Wolfram Engine with errors within a few machine epsilons. Ellip contributes to the Rust scientific ecosystem by providing fundamental mathematical functions applicable across mathematics, physics, and engineering.

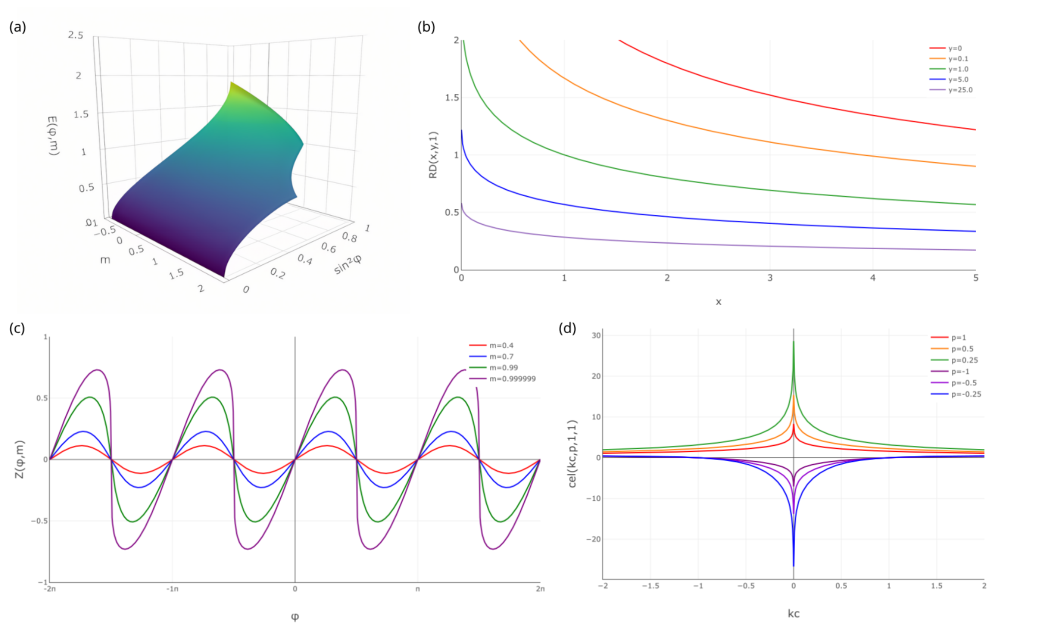

Figure 1: Examples of Elliptic Integral Functions Computed with Ellip and Visualized Using Plotly (Plotly Inc., 2025). **(a)** Legendre's Incomplete Elliptic of the Second Kind **(b)** Carlson's Degenerate Elliptic Integral of the Third Kind **(c)** Jacobi Zeta **(d)** Bulirsch's General Complete Elliptic Integral

Statement of Need

Elliptic integrals are special functions that arise in many areas of mathematics, physics, and engineering. Notably, they are used for computing the lengths of plane curves (Carlson, 2025), calculating magnetic fields (Derby & Olbert, 2010), modeling interactions in string theory (Blümlein et al., 2019), solving nonlinear mechanics (Anakhaev, 2020), and describing two-body scattering dynamics in the field of astrophysics (Bern et al., 2022).

Rust is a relatively young programming language. It features high-level memory safety without compromising performance (Jung et al., 2017), making it particularly suited for embedded scientific computing, where elliptic integrals are increasingly needed. While several programming languages have mature libraries for elliptic integrals, such as SciPy in Python (Virtanen et al., 2020), Boost.Math in C++ (Maddock et al., 2025), and the GNU Scientific Library in C (Galassi, 2009), the Rust ecosystem has lacked a comprehensive, well-tested implementation. Although the Russell Lab (Dorival Pedroso, 2025) in Rust includes Legendre's elliptic integrals, it lacks the commonly used Carlson's symmetric forms.

The Ellip library provides a suite of elliptic integral forms: Legendre's and Carlson's forms. The implementations of the two modules were derived from Boost.Math (Maddock et al., 2025). However, ellip adopts the elliptic parameter m instead of the modulus k, where $m = k^2$. This choice extends the domain of validity as m remains real when k is purely imaginary. An example application is the computation of the magnetic field of a cylinder magnet with arbitrary uniform magnetization (Caciagli et al., 2018).

Additionally, the library covers Bulirsch's forms, which are often not included in standard mathematical libraries, and miscellaneous functions, such as Heuman lambda and Jacobi zeta. All functions are extensively documented and tested, with parallelization available through the companion library Ellip-Rayon and Python support via EllipPy (Pornsiriprasert, 2025).

Software Implementation

An elliptic integral is an integral of rational function R of t and the square root of polynomial $P(t)$:

$$\int R(t, \sqrt{P(t)})dt,$$

where P is a cubic or quartic polynomial in t (Byrd et al., 1971).

Ellip consists of four modules: `legendre`, `bulirsch`, `carlson`, and `misc`. The functions are implemented exclusively in Rust, accept generic real numbers provided by num-traits's `float` (Stone, 2024), and operate entirely on the stack. The functions are outlined in Tables 1, 2, 3, and 4, respectively.

Performance is optimized by deferring input validation, i.e., assuming inputs are valid and raising errors upon completing the routine. Ellip-Rayon was released as a companion library for parallelizing large inputs. Python is supported via the EllipPy library, using PyO3 (PyO3 Project and Contributors, 2025) for Rust-Python binding. The documentation covers all public functions, encompassing their mathematical definitions, domains, graphical representations, special cases, and related functions.

Table 1: Legendre's Elliptic Integral Functions of the Module `legendre`

Function	Definition (Carlson, 2025)
Legendre's Complete Elliptic Integrals	
ellipk	$K(m) = \int_0^{\pi/2} \frac{d\theta}{\sqrt{1-m\sin^2\theta}}$
ellipe	$E(m) = \int_0^{\pi/2} \sqrt{1-m\sin^2\theta}\,d\theta$
ellippi	$\Pi(n,m) = \int_0^{\pi/2} \frac{d\theta}{(1-n\sin^2\theta)\sqrt{1-m\sin^2\theta}}$
ellipd	$D(m) = \frac{K(m)-E(m)}{m}$

Function	Definition (Carlson, 2025)
Legendre's Incomplete Elliptic Integrals	
ellipf	$F(\phi, m) = \int_0^{\phi} \frac{d\theta}{\sqrt{1-m\sin^2\theta}}$
ellipeinc	$E(\phi, m) = \int_0^{\phi} \sqrt{1-m\sin^2\theta}\, d\theta$
ellippiinc	$\Pi(\phi, n, m) = \int_0^{\phi} \frac{d\theta}{(1-n\sin^2\theta)\sqrt{1-m\sin^2\theta}}$
ellippiinc_bulirsch	Same as $\Pi(\phi, n, m)$
ellipdinc	$D(\phi, m) = \frac{F(\phi,m)-E(\phi,m)}{m}$
Computation using lookup tables (Abramowitz & Stegun, 2013) and relations to Carlson's symmetric integrals (Carlson, 2025).	

Table 2: Bulirsch's Elliptic Integral Functions of the Module `bulirsch`

Function	Definition (Bulirsch, 1969)
cel	$cel(k_c, p, a, b) = \int_0^{\pi/2} \frac{a\cos^2\theta+b\sin^2\theta}{(\cos^2\theta+p\sin^2\theta)} \frac{d\theta}{\sqrt{\cos^2\theta+k_c^2\sin^2\theta}}$
cel1	$cel1(k_c) = \int_0^{\pi/2} \frac{d\theta}{\sqrt{\cos^2\theta+k_c^2\sin^2\theta}}$
cel2	$cel2(k_c, a, b) = \int_0^{\pi/2} \frac{a+b\tan^2\theta}{(1+k_c^2\tan^2\theta)} \frac{d\theta}{\sqrt{1+\tan^2\theta}}$
el1	$el1(x, k_c) = \int_0^{\arctan x} \frac{d\theta}{\sqrt{\cos^2\theta+k_c^2\sin^2\theta}}$
el2	$el2(x, k_c, a, b) = \int_0^{\arctan x} \frac{a+b\tan^2\theta}{(1+k_c^2\tan^2\theta)} \frac{d\theta}{\sqrt{1+\tan^2\theta}}$
el3	$el3(x, k_c, p) = \int_0^{\arctan x} \frac{d\theta}{(\cos^2\theta+p\sin^2\theta)\sqrt{\cos^2\theta+k_c^2\sin^2\theta}}$
Based on the original implementations by Bulirsch (1969).	

Table 3: Carlson's Symmetric Elliptic Integral Functions of the Module `carlson`

Function	Definition (Carlson, 2025)
elliprf	$R_F(x, y, z) = \frac{1}{2}\int_0^{\infty} \frac{dt}{\sqrt{(t+x)(t+y)(t+z)}}$
elliprg	$R_G(x, y, z) = \frac{1}{4}\int_0^{\infty} \frac{t\left(\frac{x}{t+x}+\frac{y}{t+y}+\frac{z}{t+z}\right)dt}{\sqrt{(t+x)(t+y)(t+z)}}$
elliprj	$R_J(x, y, z, p) = \frac{3}{2}\int_0^{\infty} \frac{dt}{(t+p)\sqrt{(t+x)(t+y)(t+z)}}$
elliprc	$R_C(x, y) = \frac{1}{2}\int_0^{\infty} \frac{dt}{(t+y)\sqrt{t+x}}$
elliprd	$R_D(x, y, z) = \frac{3}{2}\int_0^{\infty} \frac{dt}{(t+z)\sqrt{(t+x)(t+y)(t+z)}}$
Computation using Carlson's duplication theorem (Carlson, 2025).	

The compilation of Ellip is controlled by feature flags. The `no-std` flag enables Ellip to compile in no-std environments. The `unstable` flag exposes internal functions for advanced users, bypassing input validations at the cost of safety guarantees. Lastly, the `test_force_fail` is used for code coverage, where some conditions are unreachable as remnants of defensive programming.

Table 4: Miscellaneous Functions Related to Elliptic Integrals of the Module `misc`

Function	**Definition** (Bulirsch, 1969; Reinhardt & Walker, 2025)
jacobi_zeta	$Z(\phi, m) = E(\phi, m) - \frac{E(m)\,F(\phi,m)}{K(m)}$
heuman_lambda	$\Lambda_0(\phi, m) = \frac{F(\phi,1-m)}{K(1-m)} + \frac{2}{\pi} K(m)\, Z(\phi, 1-m)$
Computation using relations to Carlson's symmetric integrals.	

Results

The library has been extensively tested against Wolfram Engine across the supported domains. The results were reported as symmetric relative errors (δ) in units of machine epsilon, as defined by

$$\delta(a,b) = \begin{cases} 0, & \text{if } |a-b| < \epsilon, \\ \dfrac{|a-b|}{\max(|a|,|b|)}, & \text{otherwise,} \end{cases}$$

where $\epsilon = 2.2204460492503131 * 10^{-16}$.

Test data generation scripts are provided for reproducibility. The error report is automatically generated via continuous integration. The following results were generated on AMD Ryzen 5 4600H with Radeon Graphics @3.0 GHz running x86_64-unknown-linux-gnu rustc 1.90.0 using ellip v0.5.9 at 64-bit precision.

Table 5: Summary of Function Accuracy at 64-bit Precision.

Function	**Median (ε)**	**Max (ε)**	**Variance (ε²)**
Module `legendre`			
ellipk	0.00	108.14	8.39
ellipe	0.00	3.00	0.19
ellippi	0.00	36.35	0.86
ellipd	0.00	2.64	0.27
ellipf	0.00	7.47	0.36
ellipeinc	0.00	24.66	1.87
ellippiinc	0.00	395.31	180.66
ellippiinc_bulirsch	0.00	395.31	180.05
ellipdinc	0.00	8.38	0.46
Module `bulirsch`			
cel	0.62	36.94	7.88
cel1	0.00	8.68	1.56
cel2	0.00	3.47	0.51
el1	0.00	1.70	0.08
el2	0.00	74.60	16.74
el3	0.00	53.21	17.51

Function	Median (ε)	Max (ε)	Variance (ε²)
Module `carlson`			
elliprf	0.00	1.57	0.19
elliprg	0.00	5.25	0.38
elliprj	0.56	136.97	13.93
elliprc	0.00	2.82	0.14
elliprd	0.00	6.25	0.40
Module `misc`			
jacobi_zeta	0.00	8.66	0.34
heuman_lambda	0.00	2.86	0.24
Machine epsilon (ε) $= 2.220446049250313 1 * 10^{-16}$			

Numerical errors are typically below one machine epsilon, and generally within tens to hundreds of machine epsilon. Since the functions are convergent, such errors can be mitigated upon the availability of a 128-bit float type in the Rust stable build.

Usage Example

The following example shows a function for calculating the perimeter of an ellipse using the formula derived from Chandrupatla & Osler (2010). With $m = 1 - b^2/a^2$, the formula can be expressed using Legendre's elliptic integral of the second kind E or Carlson's symmetric integrals R_G:

$$P(a, b) = 4aE(m) = 8aR_G(0, a^2, b^2).$$

Example 1: Ellipse Perimeter Calculation.

```rust
use ellip::*;

fn ellipse_perimeter(a: f64, b: f64) -> Result<f64, StrErr> {
    Ok(8.0 * elliprg(0.0, a * a, b * b)?)
}

// Example: ellipse with semi-major axis 5, semi-minor axis 3
println!("{}", ellipse_perimeter(5.0, 3.0).unwrap()); // 25.526998863398124
```

References

Abramowitz, M., & Stegun, I. A. (2013). *Handbook of Mathematical Functions: With Formulas, Graphs, and Mathematical Tables* (9. Dover print.; [Nachdr. der Ausg. von 1972]). Dover Publ. ISBN: 978-0-486-61272-0

Anakhaev, K. N. (2020). Elliptic Integrals in Nonlinear Problems of Mechanics. *Doklady Physics*, *65*(4), 142–146. https://doi.org/10.1134/S1028335820040011

Bern, Z., Parra-Martinez, J., Roiban, R., Ruf, M. S., Shen, C.-H., Solon, M. P., & Zeng, M. (2022). Scattering Amplitudes, the Tail Effect, and Conservative Binary Dynamics at O(G^4). *Physical Review Letters*, *128*(16), 161103. https://doi.org/10.1103/PhysRevLett.128.161103

Blümlein, J., Schneider, C., & Paule, P. (2019). *Elliptic Integrals, Elliptic Functions and Modular Forms in Quantum Field Theory*. Springer International Publishing. https://doi.org/10.1007/978-3-030-04480-0

Bulirsch, R. (1969). Numerical Calculation of Elliptic Integrals and Elliptic Functions. III. *Numerische Mathematik*, *13*(4), 305–315. https://doi.org/10.1007/BF02165405

Byrd, P. F., Friedman, M. D., Byrd, P. F., & Friedman, M. D. (1971). *Handbook of Elliptic Integrals for Engineers and Scientists* (2. ed., rev). Springer. ISBN: 978-3-540-05318-7

Caciagli, A., Baars, R. J., Philipse, A. P., & Kuipers, B. W. M. (2018). Exact Expression for the Magnetic Field of a Finite Cylinder with Arbitrary Uniform Magnetization. *Journal of Magnetism and Magnetic Materials*, *456*, 423–432. https://doi.org/10.1016/j.jmmm.2018.02.003

Carlson, B. C. (2025, March). *DLMF: Chapter 19 Elliptic Integrals*. https://dlmf.nist.gov/19

Chandrupatla, T. R., & Osler, T. J. (2010). The Perimeter of an Ellipse. *Mathematical Scientist*, *35*(2), 122–131. https://research.ebsco.com/linkprocessor/plink?id=194f5a76-a788-3459-b2f8-05334647e374

Derby, N., & Olbert, S. (2010). Cylindrical Magnets and Ideal Solenoids. *American Journal of Physics*, *78*(3), 229–235. https://doi.org/10.1119/1.3256157

Dorival Pedroso. (2025, February). *Russell_lab::math - Rust*. https://docs.rs/russell_lab/1.10.0/russell_lab/math/index.html

Galassi, M. (2009). *GNU Scientific Library Reference Manual: For GSL Version 1.12* (3. ed). Network Theory. ISBN: 978-0-9546120-7-8

Jung, R., Jourdan, J.-H., Krebbers, R., & Dreyer, D. (2017). RustBelt: Securing the Foundations of the Rust Programming Language. *Proc. ACM Program. Lang.*, *2*, 66:1–66:34. https://doi.org/10.1145/3158154

Maddock, J., Bristow, P., Holin, H., & Zhang, X. (2025, April). *Boost Math Library - Special Functions - Elliptic Integrals*. https://www.boost.org/doc/libs/1_88_0/libs/math/doc/html/math_toolkit/ellint.html

Plotly Inc. (2025). *Plotly.rs* (Version 0.12.1). Plotly. https://github.com/plotly/plotly.rs

Pornsiriprasert, S. (2025, October 17). *EllipPy Documentation — EllipPy 0.5.6*. https://p-sira.github.io/ellippy/

PyO3 Project and Contributors. (2025). *PyO3* (Version 0.26.0). https://github.com/PyO3/pyo3

Reinhardt, W. P., & Walker, P. L. (2025, March). *DLMF: Chapter 22 Jacobian Elliptic Functions*. https://dlmf.nist.gov/22

Stone, J. (2024). *Num-traits* (Version 0.2.19). https://docs.rs/num-traits/0.2.19/num_traits/index.html

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., … van Mulbregt, P. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*(3), 261–272. https://doi.org/10.1038/s41592-019-0686-2