A New Implementation of the Mathieu Functions for SciPy

Stuart Brorson
Northeastern University

February 16, 2026

1 Introduction

The Mathieu Functions are a set of special functions of some importance in physics. They naturally emerge when solving the Helmholtz equation in elliptic coordinates through separation of variables. They also appear in celestial mechanics when analyzing perturbed planetary motion. As such, it is important to be able to calculate values for the Mathieu Functions using commonly-available numerical packages. This paper describes a new implementation of the Mathieu functions for SciPy written in C++ and compatible with SciPy's xsf (special functions) library.

The derivation and properties of the Mathieu functions are covered extremely well in the literature, most recently in reviews by Shin [1] and Brimacombe et al. [2]. Properties of the Mathieu functions are also captured by an entire chapter of the DLMF [3]. Many additional papers and books also provide detailed information about methods to calculate them [**?**]. Therefore, I will provide a summary introduction to the Mathieu functions sufficient to understand the new implementation, but direct the reader seeking more details to the literature referenced in the bibliography.

Our point of departure is the Helmholtz equation in elliptical coordinates. The two-dimensional elliptical coordinate system is shown in fig. 1. We consider solving the Helmholtz equation subject to Dirichlet boundary conditions. We denote the solution to this problem $U(u, v)$. This models the motion in the z direction of a drumhead which is fixed along the rim of the drum.

In this coordinate system Helmholtz's equation is

$$\frac{2}{f^2\left[\cosh 2u - \cos 2v\right]}\left\{\frac{\partial^2 U}{\partial u^2} + \frac{\partial^2 U}{\partial v^2}\right\} - k^2 U = 0 \tag{1}$$

with boundary condition $U(u, v) = 0$ on the ellipse's perimeter. The parameter f describes the ellipticity of the coordinate system; k is related to the oscillation frequency of the drum.

We use separation of variables to reduce this PDE to a set of ODEs. We assume the solutions of (1) may be written as a product

$$U(u, v) = R(u)S(v)$$

where $R(u)$ depends upon u only, and $S(v)$ upon v only. The function R describes the behavior of U in the "radial" direction and S in the "angular" direction.

Next we define $q = f^2k^2/4$. Then upon separation of variables we get the equations for the two types of Mathieu functions, angular:

$$\frac{\partial^2 S}{\partial v^2} + (a - 2q\cos 2v)S = 0 \tag{2}$$

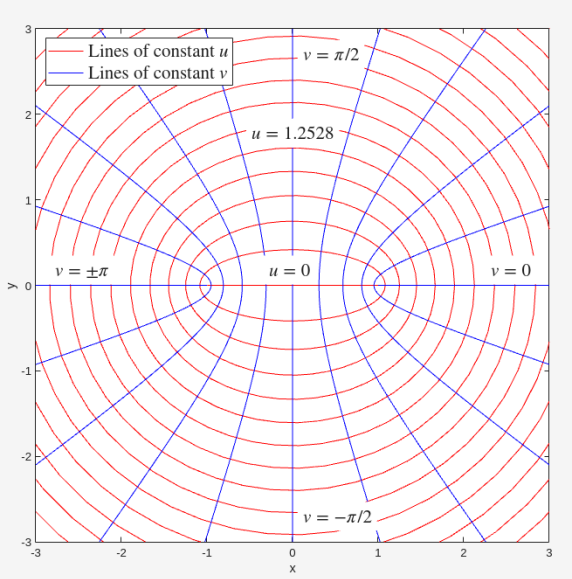


Figure 1: Elliptical coordinate system. Red lines are lines of constant radial coordinate u, blue lines are lines of constant angular coordinate v.

and radial:

$$\frac{\partial^2 R}{\partial u^2} - (a - 2q\cosh 2u)R = 0 \tag{3}$$

The variable a is the separation constant – it corresponds to an eigenvalue to be determined during solution of these equations. It is frequently called the "Mathieu characteristic value". Equation (2) describes the behavior of $U(u, v)$ in the angular direction. The solutions are the ordinary Mathieu functions, commonly written as $ce(v, q)$ and $se(v, q)$. They take two args: The angular coordinate v and the parameter q. Similarly, equation (3) describes the behavior of $U(u, v)$ in the radial direction. The solutions are the modified Mathieu functions, commonly written as $Mc(v, q)$ and $Ms(v, q)$. They also take two args: The radial coordinate u and the parameter q.

Equations (2) and (3) are of Sturm-Liouville form. Therefore, upon imposing boundary conditions we expect to find a set of solutions $ce_m(v, q)$ and $se_m(v, q)$ and corresponding eigenvalues a_m. That is, there is an infinite set of solutions to (2) and (3) indexed by order m; taken all together they form the set of Mathieu Functions.

The original implementation of the Mathieu Functions used by SciPy was written by S. Zhang and J.-M. Jin, [5] and documented in their book "Computation of Special Functions" [5]. That implementation worked fine for low-order functions, but evidenced bugs (incorrect returns) starting with $m = 6$ and continuing with high-order functions [8]. This was brought to my attention by my undergraduate REU students in Summer 2024. My new implementation of the Mathieu Functions is meant to fix the buggy behavior by using an up-to-date computational method as described by many recent papers. In particular, my implementation closely follows the ideas presented in Shin's review article [1]. I also made sure that my function implementations and nomenclature mirrored that presented by the DLMF [3] since the DLMF attempts to provide a standard description of the

functions to help tame the chaos presented by the many different Mathieu descriptions in the open literature [1].

2 Calculating the Mathieu Functions

2.1 Angular Functions

As is traditional, we start with the angular functions. These functions must be periodic since they repeat each time one takes a full elliptical walk around the coordinate system presented in fig. 1. (A second set of non-periodic Mathieu functions also exists, but is of less practical interest and is not computed in either the old or the new SciPy implementation.) Since they are periodic, we may expand them into a Fourier series,

$$ce_{2m}(v,q) = \sum_{k=0}^{\infty} A_{2k}^{(2m)}(q)\cos 2kv \tag{4}$$

$$ce_{2m+1}(v,q) = \sum_{k=0}^{\infty} A_{2k+1}^{(2m+1)}(q)\cos(2k+1)v \tag{5}$$

$$se_{2m+1}(v,q) = \sum_{k=0}^{\infty} B_{2k+1}^{(2m+1)}(q)\sin(2k+1)v \tag{6}$$

$$se_{2m+2} = \sum_{k=0}^{\infty} B_{2k+2}^{(2m+2)}(q)\sin(2k+2)v \tag{7}$$

Note the notation labeling order m. The order index is configured so the valid functions are:

- Even functions of even order: $ce_0, ce_2, ce_4, ce_6, ...$
- Even functions of odd order $ce_1, ce_3, ce_5, ce_7, ...$
- Odd functions of odd order $se_1, se_3, se_5, se_7, ...$
- Odd functions of even order $se_2, se_4, se_6, se_8, ...$

This notation is used so that the index remains $m = 0, 1, 2, 3, ...$ for all cases. This notation is used on the DLMF [3] and I use this notation in the remainder of this paper.

At this point the Fourier coefficients A_n and B_n are unknown – we need to compute them in order to evaluate the Fourier sums. There are two potential methods to calculate these Fourier coefficients. In one, a recurrence relation is derived relating the coefficients to each other. This recurrence relation is then manipulated into a function $T(a, q)$ possessing many zeros as a function of eigenvalue a. For fixed q a rootfinder may then be used to identify the values of a for which $T(a, q) = 0$. Details about this method are presented in Brimacombe et al's review [2] and in the original work by Blanch [7]. This method has the problem that it is very difficult to use a rootfinder to find specific roots of a function with many closely-spaced roots. Presented with closely-spaced roots, most rootfinders can mistakenly jump from one root to a different one while trying to find a solution. This behavior is the root cause of the bugs displayed by the Zhang and Jin implementation [4].

In the second method, the recurrence relations are used to construct a matrix eigendecomposition problem. Then the eigenvectors of the matrix are the desired Fourier coefficients and its eigenvalues are the Mathieu characteristic values corresponding to each order [**?**], [**?**]. There are four different eigendecompositions corresponding to the four different Mathieu functions. For even ce_{2m}

$$\begin{bmatrix} 0 & \sqrt{2}q & 0 & 0 & 0 & \dots \\ \sqrt{2}q & 4 & q & 0 & 0 & \dots \\ 0 & q & 16 & q & 0 & \dots \\ 0 & 0 & q & 36 & q & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \begin{pmatrix} \sqrt{2}A_0^{(2m)} \\ A_2^{(2m)} \\ A_4^{(2m)} \\ A_6^{(2m)} \\ A_8^{(2m)} \\ \vdots \end{pmatrix} = a_{2m} \begin{pmatrix} \sqrt{2}A_0^{(2m)} \\ A_2^{(2m)} \\ A_4^{(2m)} \\ A_6^{(2m)} \\ A_8^{(2m)} \\ \vdots \end{pmatrix} \tag{8}$$

The appearance of $\sqrt{2}$ is related to constructing a symmetric matrix suitable for accurate numerical eigendecomposition; refer to [1] and [2] for details.
For odd ce_{2m+1}

$$\begin{bmatrix} 1+q & q & 0 & 0 & 0 & \dots \\ q & 9 & q & 0 & 0 & \dots \\ 0 & q & 25 & q & 0 & \dots \\ 0 & 0 & q & 49 & q & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \begin{pmatrix} A_1^{(2m+1)} \\ A_3^{(2m+1)} \\ A_5^{(2m+1)} \\ A_7^{(2m+1)} \\ A_9^{(2m+1)} \\ \vdots \end{pmatrix} = a_{2m+1} \begin{pmatrix} A_1^{(2m+1)} \\ A_3^{(2m+1)} \\ A_5^{(2m+1)} \\ A_7^{(2m+1)} \\ A_9^{(2m+1)} \\ \vdots \end{pmatrix} \tag{9}$$

For se_{2m+1}

$$\begin{bmatrix} 1-q & q & 0 & 0 & 0 & \dots \\ q & 9 & q & 0 & 0 & \dots \\ 0 & q & 25 & q & 0 & \dots \\ 0 & 0 & q & 49 & q & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \begin{pmatrix} B_1^{(2m+1)} \\ B_3^{(2m+1)} \\ B_5^{(2m+1)} \\ B_7^{(2m+1)} \\ B_9^{(2m+1)} \\ \vdots \end{pmatrix} = b_{2m+1} \begin{pmatrix} B_1^{(2m+1)} \\ B_3^{(2m+1)} \\ B_5^{(2m+1)} \\ B_7^{(2m+1)} \\ B_9^{(2m+1)} \\ \vdots \end{pmatrix} \tag{10}$$

and for se_{2m+2}

$$\begin{bmatrix} 4 & q & 0 & 0 & 0 & \dots \\ q & 16 & q & 0 & 0 & \dots \\ 0 & q & 36 & q & 0 & \dots \\ 0 & 0 & q & 64 & q & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \begin{pmatrix} B_2^{(2m+2)} \\ B_4^{(2m+2)} \\ B_6^{(2m+2)} \\ B_8^{(2m+2)} \\ B_{10}^{(2m+2)} \\ \vdots \end{pmatrix} = b_{2m+2} \begin{pmatrix} B_2^{(2m+2)} \\ B_4^{(2m+2)} \\ B_6^{(2m+2)} \\ B_8^{(2m+2)} \\ B_{10}^{(2m+2)} \\ \vdots \end{pmatrix} \tag{11}$$

Solving the eigenvalue equations (8) – (11) provides both the Mathieu characteristic a, b respectively for the even, odd functions as well as the Fourier expansion coefficients A and B used to compute the function value via (4) – (7). This is the method I adopted for my Mathieu implementation.

2.2 Angular Function Derivatives

Knowing the derivatives of the angular functions is useful for solving the Helmholtz equation subject to Neumann boundary conditions, amongst other uses. Fortunately, the derivatives of *ce* and *se* are easily found by differentiating the Fourier series (4) – (7). The same coefficients found from (8) – (11) are used in the derivative sum. Here are the derivatives:

$$\frac{d}{dv}ce_{2m}(v,q) = -\sum_{k=0}^{\infty} A_{2k}^{(2m)}(q)(2k)\sin 2kv \tag{12}$$

$$\frac{d}{dv}ce_{2m+1}(v,q) = -\sum_{k=0}^{\infty} A_{2k+1}^{(2m+1)}(q)(2k+1)\sin(2k+1)v \tag{13}$$

$$\frac{d}{dv}se_{2m+1}(v,q) = \sum_{k=0}^{\infty} B_{2k+1}^{(2m+1)}(q)(2k+1)\cos(2k+1)v \tag{14}$$

$$\frac{d}{dv}se_{2m+2} = \sum_{k=0}^{\infty} B_{2k+2}^{(2m+2)}(q)(2k+2)\cos(2k+2)v \tag{15}$$

2.3 Radial Functions

The radial Mathieu functions come in two variants. They are called the "modified Mathieu function of the first type" and the "modified Mathieu function of the second type" and are denoted by $Mc^{(1)}$, $Ms^{(1)}$, $Mc^{(2)}$, and $Ms^{(2)}$ where the superscript denotes first and second type. Similar to the angular functions, each radial function has an even and odd variant. The modified functions may be computed using the following Fourier-Bessel series:

$$\begin{aligned} Mc_{2m}^{(i)}(q,u) = {} & \frac{(-1)^m}{\varepsilon_c}\sum_{k=0}^{\infty}(-1)^k\frac{A_{2k}^{(2m)}(q)}{A_{2c}^{(2m)}(q)} \\ & \times\Bigg(J_{k-c}(\sqrt{q}e^{-u})Z_{k+c}(\sqrt{q}e^{u}) \\ & \qquad + J_{k+c}(\sqrt{q}e^{-u})Z_{k-c}(\sqrt{q}e^{u})\Bigg) \end{aligned} \tag{16}$$

$$\begin{aligned} Mc_{2m+1}^{(i)}(q,u) = {} & (-1)^m\sum_{k=0}^{\infty}(-1)^k\frac{A_{2k+1}^{(2m+1)}(q)}{A_{2c+1}^{(2m+1)}(q)} \\ & \times\Bigg(J_{k-c}(\sqrt{q}e^{-u})Z_{k+c+1}(\sqrt{q}e^{u}) \\ & \qquad + J_{k+c+1}(\sqrt{q}e^{-u})Z_{k-c}(\sqrt{q}e^{u})\Bigg) \end{aligned} \tag{17}$$

$$Ms_{2m+1}^{(i)}(q,u) = (-1)^m \sum_{k=0}^{\infty} (-1)^k \frac{B_{2k+1}^{(2m+1)}(q)}{B_{2c+1}^{(2m+1)}(q)} \times \Bigg(J_{k-c}(\sqrt{q}e^{-u}) Z_{k+c+1}(\sqrt{q}e^{u}) - J_{k+c+1}(\sqrt{q}e^{-u}) Z_{k-c}(\sqrt{q}e^{u}) \Bigg) \tag{18}$$

$$Ms_{2m+2}^{(i)}(q,u) = (-1)^m \sum_{k=0}^{\infty} (-1)^k \frac{B_{2k+2}^{(2m+2)}(q)}{B_{2c+2}^{(2m+2)}(q)} \times \Bigg(J_{k-c}(\sqrt{q}e^{-u}) Z_{k+c+2}(\sqrt{q}e^{u}) - J_{k+c+2}(\sqrt{q}e^{-u}) Z_{k-c}(\sqrt{q}e^{u}) \Bigg) \tag{19}$$

In (16) above, $\varepsilon_0 = 2$ and $\varepsilon_c = 1$ for $c = 1, 2, 3, ...$ We also use the shorthand notation $i = 1$ or 2 so that $Z_i^{(1)}(z) = J_i^{(1)}(z)$ and $Z_i^{(2)}(z) = Y_i^{(2)}(z)$. This denotes that the Bessel J function is used in the sum for modified functions of the first kind while Y is used for modified functions of the second kind.

Brimacombe et al. comment that these expansions are "preposterous" and "alien" but yield the best computational results [2]. Of importance to note is the order offset c, an integer which may be chosen at will. That is, the series expansions (16) – (19) are theoretically valid for any integer c, but some c values provide faster convergence than others. Therefore, choosing an optimal value for c is part of the computational strategy required to achieve good accuracy [1].

2.4 Radial Function Derivatives

Similar to the angular function derivatives, the radial function derivatives are computed using the derivative of the Fourier series. Upon differentiating (16) – (19) we have

$$\frac{d}{du} Mc_{2m}^{(i)}(q,u) = \frac{(-1)^m}{\varepsilon_c} \sqrt{q} \sum_{k=0}^{\infty} (-1)^k \frac{A_{2k}^{(2m)}(q)}{A_{2c}^{(2m)}(q)} \times \Bigg(e^{u} \Big(J_{k-c}(\sqrt{q}e^{-u}) Z'_{k+c}(\sqrt{q}e^{u}) + J_{k+c}(\sqrt{q}e^{-u}) Z'_{k-c}(\sqrt{q}e^{u}) \Big) - e^{-u} \Big(J'_{k-c}(\sqrt{q}e^{-u}) Z_{k+c}(\sqrt{q}e^{u}) + J'_{k+c}(\sqrt{q}e^{-u}) Z_{k-c}(\sqrt{q}e^{u}) \Big) \Bigg) \tag{20}$$

$$\frac{d}{du}Mc^{(i)}_{2m+1}(q,u) = (-1)^m\sqrt{q}\sum_{k=0}^{\infty}(-1)^k\frac{A^{(2m+1)}_{2k+1}(q)}{A^{(2m+1)}_{2c+1}(q)}$$
$$\times\left(e^u\Big(J_{k-c}(\sqrt{q}e^{-u})Z'_{k+c+1}(\sqrt{q}e^u) + J_{k+c+1}(\sqrt{q}e^{-u})Z'_{k-c}(\sqrt{q}e^u)\Big)\right.$$
$$\left. - e^{-u}\Big(J'_{k-c}(\sqrt{q}e^{-u})Z_{k+c+1}(\sqrt{q}e^u) + J'_{k+c+1}(\sqrt{q}e^{-u})Z_{k-c}(\sqrt{q}e^u)\Big)\right) \quad (21)$$

$$\frac{d}{du}Ms^{(i)}_{2m+1}(q,u) = (-1)^m\sqrt{q}\sum_{k=0}^{\infty}(-1)^k\frac{B^{(2m+1)}_{2k+1}(q)}{B^{(2m+1)}_{2c+1}(q)}$$
$$\left(e^u\Big(J_{k-c}(\sqrt{q}e^{-u})Z'_{k+c+1}(\sqrt{q}e^u) - J_{k+c+1}(\sqrt{q}e^{-u})Z'_{k-c}(\sqrt{q}e^u)\Big)\right.$$
$$\left. - e^{-u}\Big(J'_{k-c}(\sqrt{q}e^{-u})Z_{k+c+1}(\sqrt{q}e^u) - J'_{k+c+1}(\sqrt{q}e^{-u})Z_{k-c}(\sqrt{q}e^u)\Big)\right) \quad (22)$$

$$\frac{d}{du}Ms^{(i)}_{2m+2}(q,u) = (-1)^m\sqrt{q}\sum_{k=0}^{\infty}(-1)^k\frac{B^{(2m+2)}_{2k+2}(q)}{B^{(2m+2)}_{2c+2}(q)}$$
$$\left(e^u\Big(J_{k-c}(\sqrt{q}e^{-u})Z'_{k+c+2}(\sqrt{q}e^u) + J_{k+c+2}(\sqrt{q}e^{-u})Z'_{k-c}(\sqrt{q}e^u)\Big)\right.$$
$$\left. - e^{-u}\Big(J'_{k-c}(\sqrt{q}e^{-u})Z_{k+c+2}(\sqrt{q}e^u) - J'_{k+c+2}(\sqrt{q}e^{-u})Z_{k-c}(\sqrt{q}e^u)\Big)\right) \quad (23)$$

3 Implementation Details

The new Mathieu implementation uses the above sums to compute the desired functions. Each function takes three input args: order m, parameter q and spatial coordinate u or v (here denoted x for simplicity). The Mathieu characteristic values $a_m(q)$ and $b_m(q)$ are also available. The calling API and some details of the internal function names are shown in the table. Valid values of m are the positive integers starting from 0 for ce, $Mc^{(1)}$, $Mc^{(2)}$ and starting from 1 for se, $Ms^{(1)}$, $Ms^{(2)}$. For $m > 500$ the implementation returns NaN since extremely high orders may suffer from accuracy loss. Valid values of q are the real numbers. For $|q| > 1000$ the implementation returns a computed result, but also sets a SciPy flag indicating possible degraded accuracy.

To compute the functions, the implementation proceeds by first building the appropriate matrix, then solving the eigenvalue decomposition appropriate for the input (one of 8 – 11). It then uses the corresponding series to compute the desired function. The characteristic values are computed using the same matrix and eigenvalue decomposition and then returning the eigenvalue.

Here are some details about the new implementation:

Mathematical function name	**SciPy call/return** (scipy.special)	**Internal C++ call/return**
$a_m(q)$	a = mathieu_a(m,q)	int mathieu_a(m, q, *a)
$b_m(q)$	b = mathieu_b(m,q)	int mathieu_b(m, q, *a)
$ce_m(q,x)$	ce,cederiv = mathieu_cem(m,q,x)	int mathieu_ce(m,q,x,*ce,*ced)
$se_m(q,x)$	se,sederiv = mathieu_sem(m,q,x)	int mathieu_se(m,q,x,*se,*sed)
$Mc_m^{(1)}(q,x)$	cem1,cem1deriv = mathieu_modcem1(m,q,x)	int mathieu_modmc1(m,q,x,*mc1,*mc1d)
$Ms_m^{(1)}(q,x)$	sem1,sem1deriv = mathieu_modsem1(m,q,x)	int mathieu_modms1(m,q,x,*ms1,*ms1d)
$Mc_m^{(2)}(q,x)$	cem2,cem2deriv = mathieu_modcem2(m,q,x)	int mathieu_modmc2(m,q,x,*mc2,*mc2d)
$Ms_m^{(2)}(q,x)$	sem2,sem2deriv = mathieu_modsem2(m,q,x)	int mathieu_modms2(m,q,x,*ms2,*ms2d)

Table 1: The new implementation's API. Note that the interface to SciPy does not change from the old implementation. The internal C++ call returns an integer error code; function values are returned by reference.

- The entry point to the implementation occurs in the function mathieu.h. The API presented in this function matches that required by SciPy's existing implementation which supports ufuncs. This function takes care of casting the calling args to types acceptable to the new implementation, performs some argument checks, then calls the new C++ implementation which lives in a subdirectory of xsf. I purposely did not modify the API presented in this file for backwards compatibility with the existing SciPy functionality.

- The recursion matrix size N is tied to the input order m and parameter q. For $q \leq 1$ we use $N = m + 25$. The reason is that for small q the Fourier coefficient magnitudes are sharply peaked at elements whose index lies around the order m. Therefore, the eigenvalue equation must be solved using a matrix of at least that size. For $q > 1$ we use $N = m + 25 + 10log_{10}(q)$. The reason is that for larger q the Fourier coefficients become less sharply peaked around m, so more terms must be brought into the sum. The exact expression used here was chosen empirically based upon numerical experimentation and evaluation of the resulting accuracy of the computed function. Other approaches to computing N have been suggested in the literature; we found that those methods gave rise to unnecessarily long runtimes.

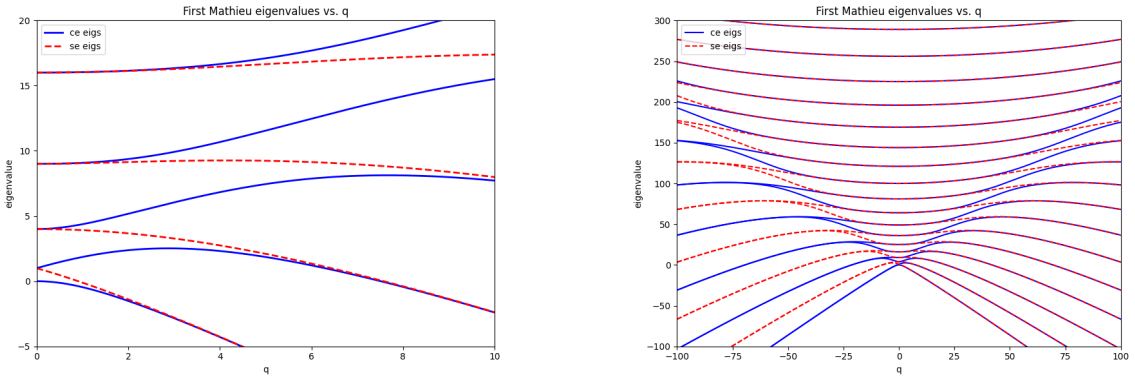
- Solving the eigenvalue equation is performed using Lapack's dstevd algorithm, which is optimized for solving tridiagonal eigenvalue equations. I tried other Lapack functions appropriate for dense matrices, but they consumed too much time and also placed limits on the usable

matrix size. One might think Arpack would be a good choice for this task since it is targeted to sparse eigensystems. However, Arpack uses Arnoldi iteration which produces accurate eigenvalues for the largest and smallest eigenvalues, but less accurate results for intermediate eigenvalues. I found through numerical experimentation that Arpack's results were not sufficiently accurate to yield good results for the Mathieu computation.

- In the C++ implementation the Fourier series terms are placed into partial sums depending upon whether the term is positive or negative. Then the positive and negative partial sums are summed at the end of the loop. The idea is to reduce any round-off error accumulated during the process of evaluating the series.
- For the modified functions, the so-called s-max method is used to determine the offset factor c for the Bessel orders. According to Shin [1] use of the s-max method is key to achieving good accuracy for the modified Mathieu functions. The idea of the s-max is that the Bessel offset c should track the index of the Fourier coefficient with the largest magnitude.
- The Bessel functions J and Y appearing in eq. (16) – eq. (19) are computed using the Cephes library which is already present in SciPy's special function library.

4 Results

The Mathieu characteristic values a and b computed using the new implementation are plotted in fig. 2 vs. parameter q. These are the eigenvalues of the Sturm-Liouville problems eq. (2) and eq. (3). a is the eigenvalue corresponding to the even eigenfunctions ce and b is the eigenvalue corresponding to the odd eigenfunctions se.

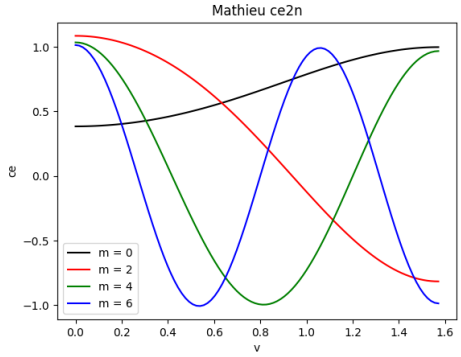


(a) Closeup.

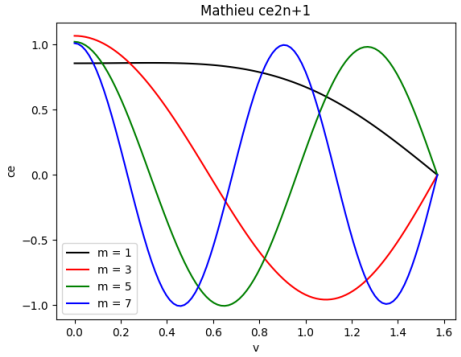
(b) Expanded view.

Figure 2: Mathieu characteristic values (eigenvalues) a and b vs. q

Plots of the Mathieu functions and derivatives computed using the new implementation are shown in fig. 3 – fig. 8. Different orders m are plotted; for each plot we have $q = 1.0$.

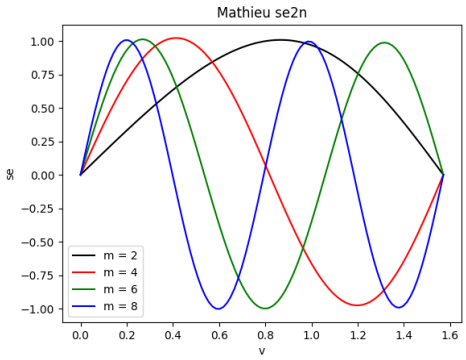


(a) Mathieu ce_{2m}

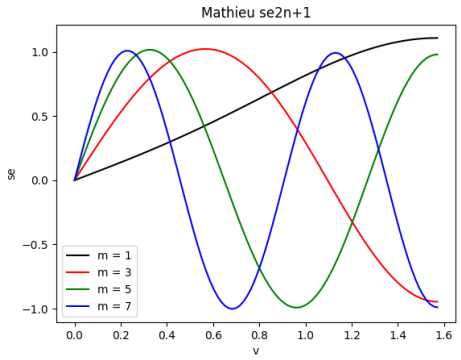


(b) Mathieu ce_{2m+1}

Figure 3: Even angular Mathieu functions ce.

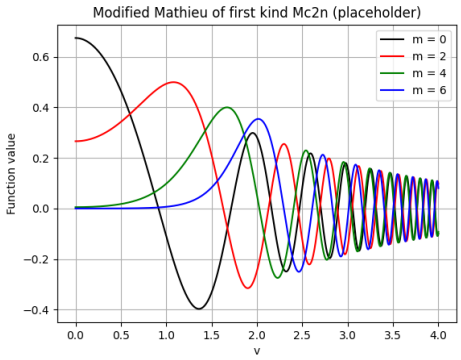


(a) Mathieu se_{2m}

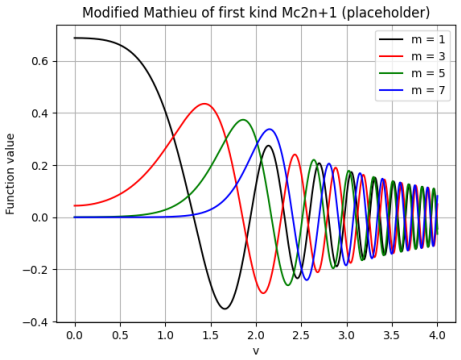


(b) Mathieu se_{2m+1}

Figure 4: Odd angular Mathieu functions se.

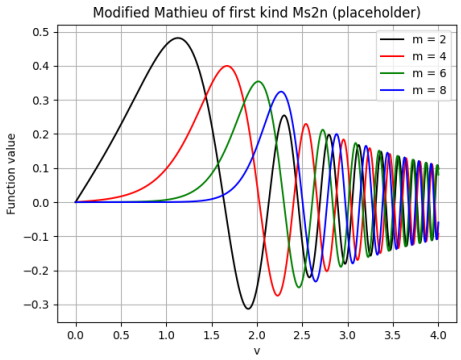


(a) Mathieu $Mc_{2m}^{(1)}$

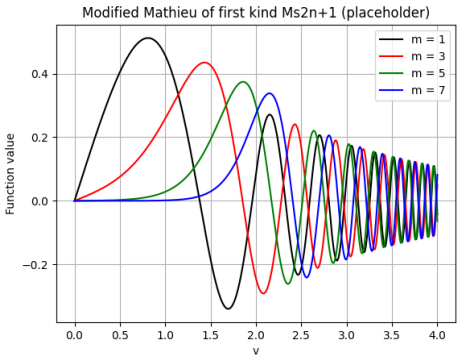


(b) Mathieu $Mc_{2m+1}^{(1)}$

Figure 5: Even radial Mathieu functions of the first kind $Mc^{(1)}$.

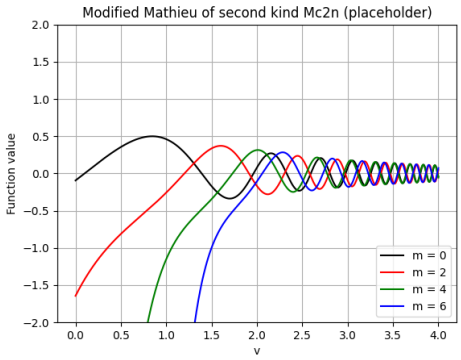


(a) Mathieu $Ms_{2m}^{(1)}$

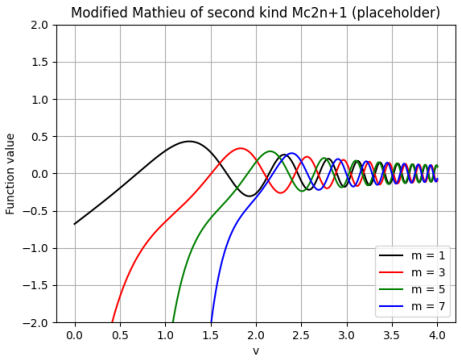


(b) Mathieu $Ms_{2m+1}^{(1)}$

Figure 6: Odd radial Mathieu functions of the first kind $Ms^{(1)}$.

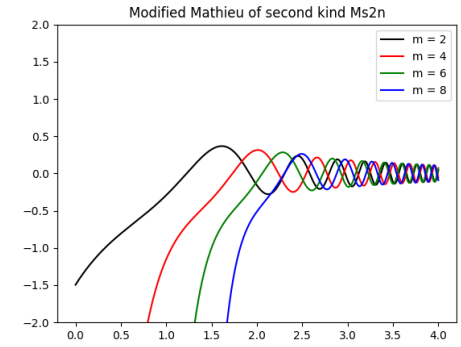


(a) Mathieu $Mc_{2m}^{(2)}$

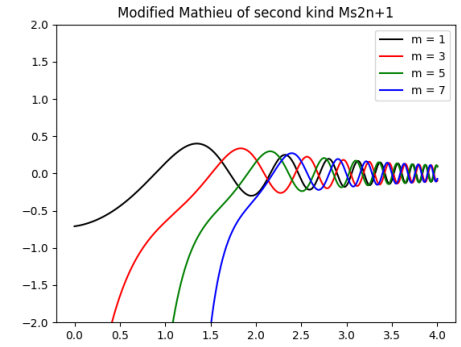


(b) Mathieu $Mc_{2m+1}^{(2)}$

Figure 7: Even radial Mathieu functions of the second kind $Mc^{(2)}$.



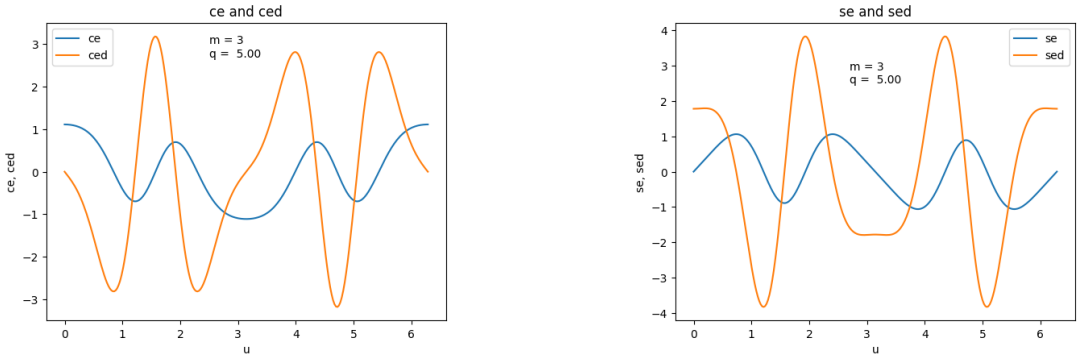
(a) Mathieu $Ms_{2m}^{(2)}$



(b) Mathieu $Ms_{2m+1}^{(2)}$

Figure 8: Odd radial Mathieu functions of the first kind $Ms^{(2)}$.

The implementation returns both the Mathieu function and its derivative with respect to the coordinate v or u. Plots of the angular functions and their derivatives for arbitrarily chosen m and q are shown in fig. 9. Analogous plots may be obtained from the modified functions.



(a) Mathieu *ce* and derivative

(b) Mathieu *se* and derivative

Figure 9: Even and odd angular functions and derivatives plotted over the entire $[0, 2\pi]$ domain. Order m and parameter q were chosen arbitrarily.

5 Testing and Validation

Validating the correctness of a library of math functions is equally important as producing the library in the first place. Therefore, validation played a big role in the creation of this new Mathieu implementation. Besides making plots and comparing to the DLMF [3] we used the tests discussed below to check the results returned by the new Mathieu implementation. The tests are available for download from GitHub [9].

5.1 Round trip equation using finite differences

The angular differential equation eq. (2) may be rewritten

$$r = \frac{\partial^2 S}{\partial v^2} + (a - 2q\cos 2v)S \tag{24}$$

where r is a residual. Ideally $r = 0$, but numerical errors usually prevent this ideal case. To characterize the error we discretize this equation using a finite-difference formula for the second derivative. We used a 6th order approximation for the second derivative operator,

$$\frac{d^2S(v)}{dv^2} \approx \Delta^2 S_n = \frac{(1/90)S_{n+3} - (3/20)S_{n+2} + (3/2)S_{n+1} - (49/18)S_n + (3/2)S_{n-1} - (3/20)S_{n-2} + (1/90)S_{n-3}}{h^2} \tag{25}$$

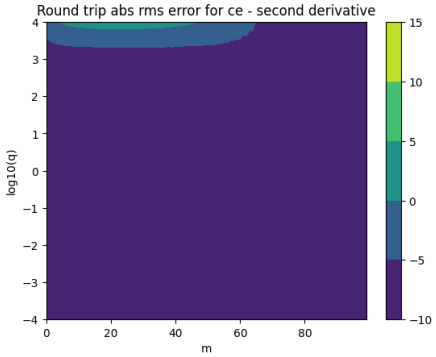
where Δ^2 denotes the discretized second-second derivative operator and h is the stepsize. Using Δ^2 we have an approximate Mathieu angular equation,

$$r_n = \Delta^2 S_n + (a - 2q\cos 2v_n)S_n \tag{26}$$

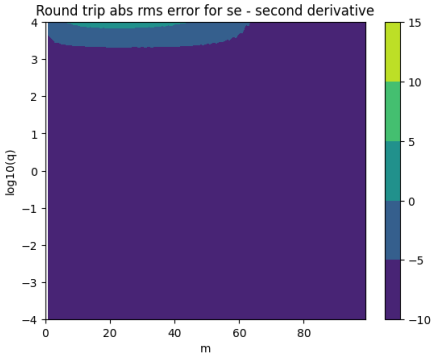
Testing using this formula involves selecting a Mathieu order m and parameter q, then computing the corresponding eigenvalue $a = a_m(q)$. We then evaluate (eq. (26)) to obtain the r_n for varying m and q. Then we compute the absolute RMS error of this residual,

$$err = \sqrt{\frac{1}{N}\sum_{n=1}^{N} r_n^2} \tag{27}$$

where N is the number of points computed. For the angular Mathieus we sample the functions at 100 points in the domain $v_n = [-\pi, \pi]$. Plots of this err are shown in fig. 10.



(a) Round trip RMS error for *ce* computed using stepsize $h = 3e - 4$.



(b) Round trip RMS error for *se* computed using stepsize $h = 3e - 4$.

Figure 10: Round trip error for the angular functions computed using finite-difference scheme for second derivative.

A similar check may be performed for the modified functions by using the approximate Mathieu radial equation,

$$r = \Delta^2 S_n - (a - 2q\cosh 2u_n)S_n \tag{28}$$

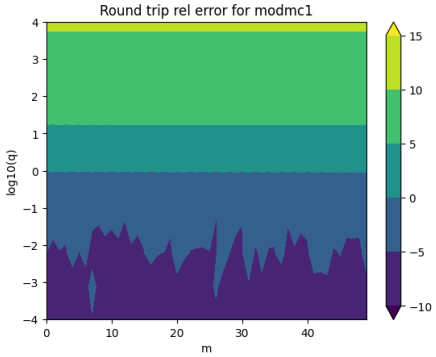
However, in this case the functions are singular for $u \to 0$, and the breakpoint where the singularity ends increases with increasing m. This is evident in the plots fig. 5 – fig. 8. Therefore, instead of plotting absolute error (which is highly sensitive to the singularity) we plot the relative RMS error

$$err = \sqrt{\frac{1}{N}\sum_{n=1}^{N} (r_n/S_n)^2} \tag{29}$$

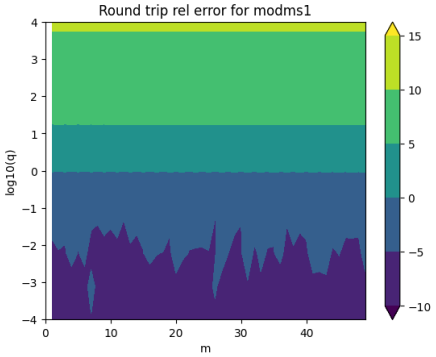
For the radial Mathieus we sample the functions at 100 points in the domain $u_n = [1, 8]$. This domain is chosen because the functions oscillate increasingly wildly as $u_n \to \infty$, so a finite-difference

approximation becomes increasingly unreliable. Moreover, the modified functions of the second kind go to infinity for $u_n \to 0$, so we need to cut off the sample domain at the low end.

One could certainly quibble that plotting relative error hides inaccuracy. However, the goal of these plots is to show that the implementation's output remains sane while the function itself is going singular. Demonstrating the relative error remains small compared to the function itself which is zooming down to $-\infty$ is a valid characterization of sanity. The relative error plots are shown in fig. 11 – fig. 12.

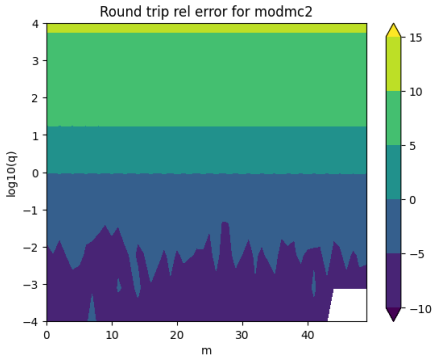


(a) Round trip error for $Mc^{(1)}$.

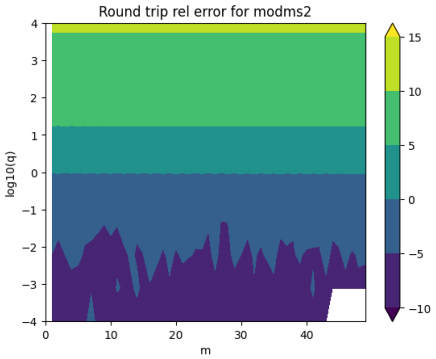


(b) Round trip error for $Ms^{(1)}$.

Figure 11: Round trip error for the modified functions of the first kind.



(a) Round trip error for $Mc^{(2)}$.



(b) Round trip error for $Ms^{(2)}$.

Figure 12: Round trip error for the modified functions of the second kind.

5.2 Wronskian check of modified fcns

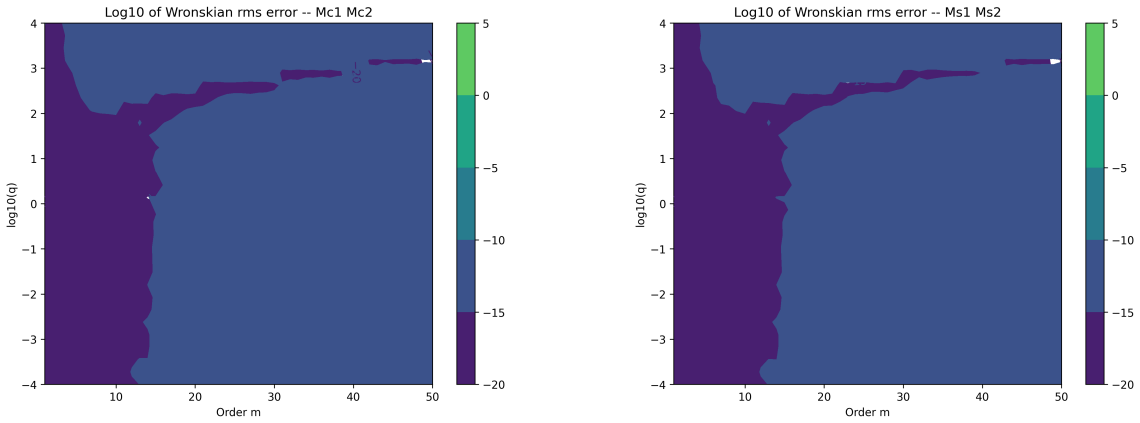
The modified Mathieu functions obey a Wronskian of the form

$$\mathcal{W}(Mx_m^{(1)}, Mx_m^{(2)}) = \frac{2}{\pi} \tag{30}$$

where $Mx = Mc, Ms$. Since the Mathieu implementation returns both the modified function and its derivative, computing the Wronskian is easy. In fig. 13 – fig. 14 we plot the absolute RMS error,

$$err = \sqrt{\frac{1}{N}\sum_{n=1}^{N}\left(\mathcal{W}(Mx_m^{(1)}, Mx_m^{(2)}) - \frac{2}{\pi}\right)^2} \tag{31}$$

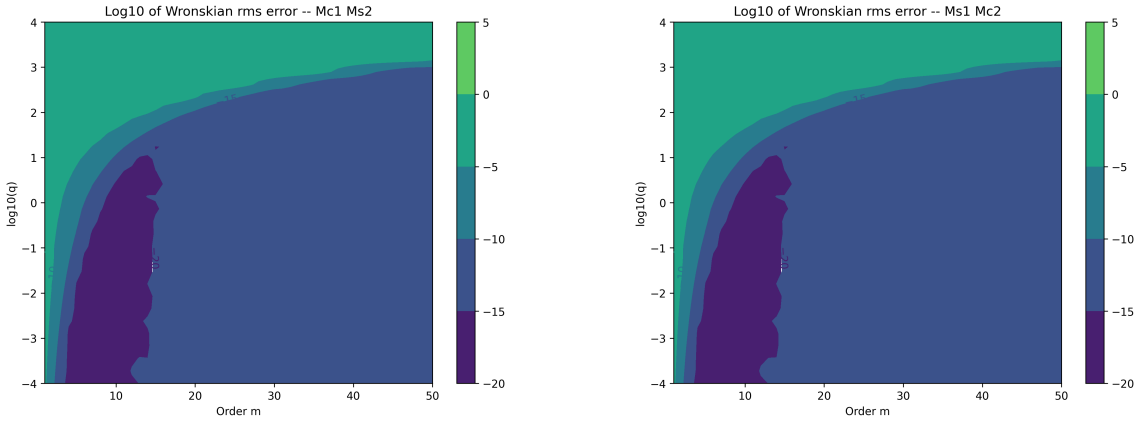
computed for different m and q values.



(a) Wronskian err $\mathcal{W}(Mc_m^{(1)}, Mc_m^{(2)}) - 2/\pi$.

(b) Wronskian err $\mathcal{W}(Ms_m^{(1)}, Ms_m^{(2)}) - 2/\pi$.

Figure 13: Absolute Wronskian errors between modified functions of the same handedness (even, odd).



(a) Wronskian err $\mathcal{W}(Mc_m^{(1)}, Ms_m^{(2)}) - 2/\pi$.

(b) Wronskian err $\mathcal{W}(Ms_m^{(1)}, Mc_m^{(2)}) - 2/\pi$.

Figure 14: Absolute Wronskian errors between modified functions of opposite handedness.

5.3 Check of first deriv using finite differences

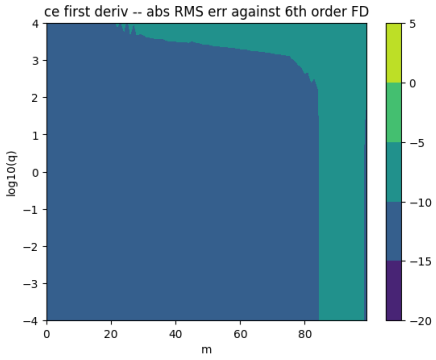
This implementation of the Mathieu functions returns both the function value and its first derivative. As a sanity check, we computed a first derivative approximation from the function value using a first-order finite difference formula,

$$\frac{dS(v)}{dv} \approx \Delta S_n = \frac{-(1/60)S_{n+3} + (3/20)S_{n+2} - (3/4)S_{n+1} + (3/4)S_{n-1} - (3/20)S_{n-2} + (1/60)S_{n-3}}{h} \tag{32}$$

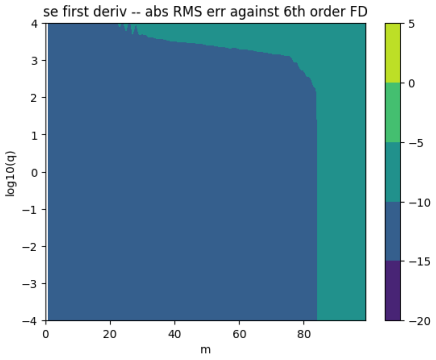
The residual is

$$r = \Delta S_n - S'_n \tag{33}$$

where S' is the first derivative returned from the implementation. For the angular functions we plot the absolute RMS error of the residual per eq. (27) in fig. 15.



(a) Mathieu *ce*.



(b) Mathieu *se*

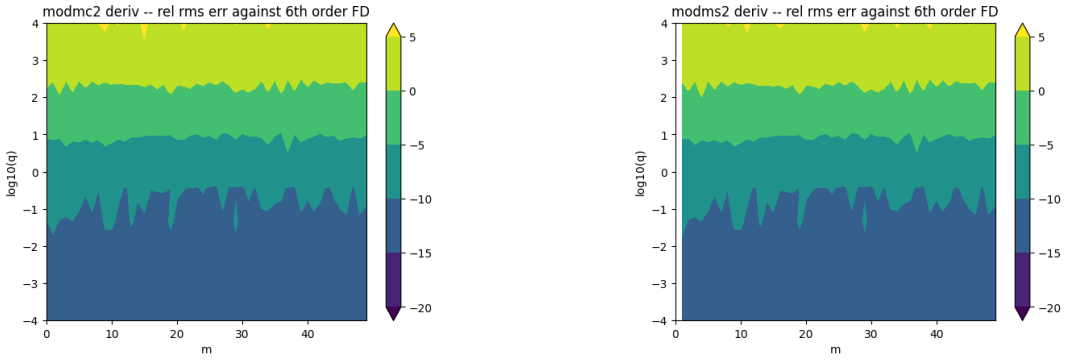
Figure 15: Absolute RMS difference between the first derivative returned by the implementation and a 6th order finite-difference approximation.

For the radial functions we plot the relative RMS error of the residual per eq. (29) in fig. 16 – fig. 17.

(a) Mathieu $Mc^{(1)}$.

(b) Mathieu $Ms^{(1)}$

Figure 16: Relative RMS difference between the first derivative returned by the implementation and a 6th order finite-difference approximation.



(a) Mathieu $Mc^{(2)}$.

(b) Mathieu $Ms^{(2)}$

Figure 17: Relative RMS difference between the first derivative returned by the implementation and a 6th order finite-difference approximation.

5.4 Mathieu sum formula for Hankel function

Ho-Chul Shin's review article [1] provides a handy series expression for the Hankel function $H_0^{(1)}(\mathbf{r})$ as a sum over Mathieu angular and radial functions. His expression is

$$\begin{aligned}\frac{1}{2}H_0^{(1)}(k|\mathbf{r}-\mathbf{r}_0|) = \\ \sum_{n=0}^{\infty} ce_n(v_0,q)ce_n(v,q)\cdot\begin{cases} Mc_n^{(1)}(q,u_0)Mc_n^{(3)}(q,u), & u>u_0, \\ Mc_n^{(1)}(q,u)Mc_n^{(3)}(q,u_0), & u<u_0, \end{cases} \\ +\sum_{n=1}^{\infty} se_n(v_0,q)se_n(v,q)\cdot\begin{cases} Ms_n^{(1)}(q,u_0)Ms_n^{(3)}(q,u), & u>u_0, \\ Ms_n^{(1)}(q,u)Ms_n^{(3)}(q,u_0), & u<u_0 \end{cases}\end{aligned} \tag{34}$$

where the coordinate is $\mathbf{r}=(u,v)$ and $q=k^2f^2/4$ using the notation in this paper. The left-hand side Hankel function describes an outgoing cylindrical wave emanating from a line source located at position $\mathbf{r}_0$. On the right-hand side we have $Mx_m^{(3)}(q,x) = Mx_m^{(1)}(q,x) + \mathrm{i}\,Mx_m^{(2)}(q,x)$ where Mx represents either Mc or Ms. The right-hand side decomposes the Hankel function into a sum of elliptical waves. The idea is that the Hankel function is well-known and is implemented in several function libraries and so is readily available for testing the Mathieu functions. Moreover, any errors in one or more of the Mathieu functions in the sum will yield incorrect values for $H_0^{(1)}$. Accordingly, eq. (34) tests the accuracy of all Mathieu functions at once.

In his test, Shin places a point source of waves at a location and then samples the waves at many different locations indexed by j. He then computes the relative difference between the reference $H_0^{(1)}$ (ref) and that returned by the implementation under test (iut),

$$\text{Error (dB)} = 10\log_{10}\frac{\sum_{j=1}^{N_\mathrm{R}}\left[H_0^{(1)}(kr_j)_\mathrm{iut} - H_0^{(1)}(kr_j)_\mathrm{ref}\right]^2}{\sum_{j=1}^{N_\mathrm{R}}\left[H_0^{(1)}(kr_j)_\mathrm{ref}\right]^2} \tag{35}$$

Note that Shin plots the error in dB units.

Dr. Shin kindly provided a plot produced when running his test function on my Mathieu implementation; his plot is shown in **??** below.

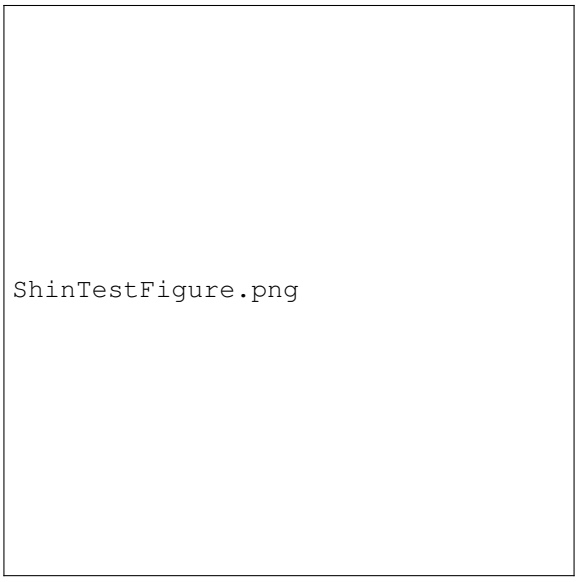


Figure 18: Error plot produced using eq. (34) and eq. (35)

6 Acknowledgements

I wish to thank my collaborators:

- The original SciPy bug was brought to my attention by Yuzhi Liu and Aidan Tillman (undergrads) whose 2024 REU involved counting the eigenvalues of the Helmholtz equation on simple shapes in 2- and 3-dimensions [10]. They attempted to count eigenvalues for an elliptic drum but were stopped by SciPy's Mathieu bugs. Their work (without the elliptic drum) was later published in [11].
- In 2025 I mentored another REU whose participants were Elodie Yecko, Quella Wang, and Katherine Glenn (undergrads) and Alia Yusaini (graduate mentor). Their project involved investigating methods to reliably compute the Mathieu functions; they implemented the first draft of the algorithms presented in this paper [12].
- Following the REU, graduate students Alia Yusaini and Tanishq Bhatia focused on developing tests for the C++ Mathieu implementations which I wrote and linked into a private Scipy build used for development.
- In 2026 Kushala Rani Talakad Manjunath worked with me on further testing the Mathieu implementations in the private SciPy build. She contributed Python code to the test suite and provided many of the plots presented in section 4 and section 5.

- Finally, during the final stages of this work the authoritative paper by Shin [1] appeared in the SIAM Journal. His paper ties together the different calculational approaches used for the Mathieu functions, identifies the best method, clarifies the different nomenclatures and function normalizations present in the literature, and finally compares the different available implementations using a test of his devising. After his paper appeared I immediately reached out to him and recruited his help with testing my implementation. He gracefully accepted my invitation; the results of his testing are shown in section 5.4. He also provided several valuable suggestions which improved my implementation.

References

[1] H.-C. Shin, "Numerical review of Mathieu function programs for integer orders and real parameters," *SIAM Review*, vol. 67, no. 4, pp. 661–733, 2025.

[2] C. Brimacombe, R. M. Corless, and M. Zamir, "Computation and applications of Mathieu functions: A historical perspective," arXiv preprint arXiv:2008.01812, 2021. Available: `https://arxiv.org/abs/2008.01812`

[3] NIST Digital Library of Mathematical Functions, `https://dlmf.nist.gov/28` (Mathieu Functions and Hill's Equation).

[4] S. Zhang and J.-M. Jin, "Special functions library (special_functions.f), ported to C++ by SciPy implementers as specfun.h," part of XSF: Extended Special Functions library. [Online]. Available: `https://github.com/scipy/xsf/tree/main/include/xsf/specfun`

[5] S. Zhang and J.-M. Jin, *Computation of Special Functions*. New York: Wiley, 1996.

[6] E. Cojocaru, "Mathieu functions computational toolbox implemented in Matlab," arXiv preprint arXiv:0811.1970, 2008. Available: `https://arxiv.org/abs/0811.1970`

[7] G. Blanch, "On the computation of Mathieu functions," *Journal of Mathematics and Physics*, vol. 25, pp. 1–20, 1946.

[8] S. Brorson, "Issue #47: [Bugs in Mathieu functions]," GitHub repository scipy/xsf, 2024. [Online]. Available: `https://github.com/scipy/xsf/issues/47` (This bug report provides links to several other bug reports observing problems with the Mathieu function implementation.)

[9] S. Brorson, "ScipyMathieuTesting: Python programs to test Mathieu functions in Scipy," 2025–2026. [Online]. Available: `https://github.com/brorson/ScipyMathieuTesting`

[10] RTG @ Northeastern, *Summer Math Research Program – REU 2024*, `https://sites.google.com/view/rtg-northeastern/undergraduate/independent-research-experience/reu-2024`

[11] A. Tillman and Y. Liu, *Counting the Eigenvalues of the Laplace Operator on Some Convex Domains*, SIAM Undergraduate Research Online, **18** (2025), pp. 106–124. `https://doi.org/10.1137/24S1681069`

[12] RTG @ Northeastern, *Northeastern Summer Math Research Program – Independent Research Experience 2025*, https://sites.google.com/view/rtg-northeastern/undergraduate/independent-research-experience