

GSoC 2025: Special Functions in The Julia Programming Language

Ahmed Y. Kadah¹

Oscar D. Smith²

¹ Student, University of Science and Technology in Zewail City

² Mentor, JuliaHub

Abstract

In this project, we will work on implementing high speed and high accuracy implementations of various special functions, as well as optimizing and improving existing implementations in Julia. Our work will cover the **SpecialFunctions.jl** and the **HyperGeometricFunctions.jl** Julia libraries. There are 4 distinct goals which are as follows: (1) implementing the error function variants [erf, erfc, erfcx, erfi, Dawson, Faddeeva] in SpecialFunctions.jl, (2) resolving existing issues in the SpecialFunctions.jl repository, (3) implementing the Appell functions in HypergeometricFunctions.jl, (4) resolving existing issues in HypergeometricFunctions.jl. The current estimate for the total project duration is ≈ 191 hours, which lies within the medium sized project range.

Introduction

About me   

Hi there! I am Ahmed Yasser Kadah, a third year Communications and Information Engineering student at the University of Science and Technology in Zewail City, Giza, Egypt. I recently took an advanced course in partial differential equations and special functions, so when I saw this project I thought it would be interesting to follow along this path and study the topics further when it comes to real world applications and make something that would genuinely benefit others. This is my first time contributing to an open source project and just recently made my first contribution with a **pull request** for a Julia implementation of the Float64 erf(x) (will be explained later).

Background on Special Functions

Special functions are a class of mathematical functions that arise in the solutions of complex physical, engineering, and mathematical problems.[25, 1] These functions are typically defined by specific properties or satisfy particular differential equations, and they include some of the

most well-known functions in mathematics, such as the error function, Gamma function, and Bessel functions.

This class of functions serves a unique purpose in many different applications, as they have properties which can be utilized to solve highly complex problems which otherwise have no analytic solutions, or even assist in numerical computation.

Another interesting class of functions, referred to as functions of the hypergeometric type, are generalized functions, of which special functions and elementary functions are special cases. [25, 2] These functions also have uses in solving highly advanced problems in physics and mathematical analysis.

Due to the nature and importance of this topic in many applications, there already exists a large number of implementations in other languages such as Wolfram Mathematica [10, 11], MATLAB [12, 13], Python [14, 15], C [24, 16, 23, 19, 20, 21, 22], and others. This significance shows that a natural progression would be to implement Julia versions of these functions. This, however, is not the only reason.

One of the things that make Julia a language of interest for such a project is the full integration of AutoDiff methods with Julia native functions. Automatic differentiation (AutoDiff) is an approach to evaluating the derivative of a function through the use of techniques which automate the evaluation of partial derivatives of the elementary arithmetic operations and functions through the repeated usage of the chain rule.[17, 18] In other words, if a function has an algorithm implemented in Julia, AutoDiff can "trace" the code to find the elementary operations and implement its methods. This isn't the case, however, for wrapper functions for example using "ccall" (a call to a C function). Since all Julia would see is the function call, the derivative for this call would need to be manually defined to be used. By implementing functions in Julia, AutoDiff will automatically be compatible.

SpecialFunctions.jl and **HyperGeometricFunctions.jl** are Julia libraries which aim to provide Julia implementations/interfaces for special functions and hypergeometric functions. SpecialFunctions.jl makes use of external libraries like openlibm and openspecfun which adds a layer of overhead and reduces the efficiency. Meanwhile, HyperGeometricFunctions.jl is purely implemented in Julia with several papers cited for the numerical approaches used in the implementations.

Goals

The goals of the project are to (1) implement some of the functions which currently use external library calls in Julia, (2) work on fixing some of the existing issues in the github repositories, (3) optimizing poorly optimized implementations, (4) explore and add missing functions.

The two repositories of interest are SpecialFunctions.jl and HypergeometricFunctions.jl as said previously, and they will be the focus of most/all of our work.

With regards to SpecialFunctions.jl, there are a few points of interest:

- SpecialFunctions.jl/bessel.jl and SpecialFunctions.jl/gamma.jl would appear to need Julia implementations, however they are both implemented in **Bessels.jl**
- All BigFloat implementations make use of the MPFR library [24], but quality of execution is high and there would be no merit behind implementing Julia versions considering the difficulty of implementation
- The majority of SpecialFunctions.jl/erf.jl uses wrapper function calls to the C standard math library [23] so this will be one of our main focuses

- 64 open issues, many of which are redundant and won't be counted after a round of filtration

With regards to `HypergeometricFunctions.jl`, there are a few points of interest:

- Most other function implementations are already Julia implementations
- 19 open issues, which again will be filtered for those worth working on
- It lacks implementations for the Appell functions, so this will be our main focus

The standard approaches for implementing special functions tend to make use of different methods of numerical approximation methods like polynomial/rational approximation. In particular, an algorithm of high significance is the **Remez algorithm**. Some of the important tools which utilize this algorithm include:

1. **Remez.jl**, a Julia library for constructing minimax polynomials and rational functions
2. **Sollya**, another powerful tool for constructing polynomials but with a less convenient interface. Will be used as the reference for the performance of `Remez.jl` since it is still being developed.

Some of the standard approaches for implementing hypergeometric functions are Taylor and asymptotic series computations, Gauss–Jacobi quadrature, numerical solution of differential equations, recurrence relations, and others.[27] So, a lot of effort needs to be put into researching the appropriate approach for an implementation of the Appell functions. To best approach these implementations, researchers of interest will be contacted to give their guidance or support.

Deliverables and Timeline

Deliverables

- Full implementation of the **Error Function** and it's varieties [3]
 - Error Function: $\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$ for Float32 and Float64 (Float64 for real values has been implemented as said previously), for real as well as complex inputs.
 - Complementary Error Function: $\operatorname{erfc}(x) = 1 - \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$ for Float32, and Float64, for real as well as complex inputs.
 - Scaled Complementary Error Function: $\operatorname{erfcx}(x) = e^{x^2} \operatorname{erfc}(x)$ for Float32, and Float64, for real as well as complex inputs.
 - Imaginary Error Function: $\operatorname{erfi}(x) = -i \operatorname{erf}(ix)$ for Float32, and Float64, for real as well as complex inputs.
 - Dawson Function: $D(x) = \frac{\sqrt{\pi}}{2} e^{-x^2} \operatorname{erfi}(x)$ for Float32, and Float64, for real as well as complex inputs.
 - Faddeeva Function: $w(z) = e^{-z^2} \operatorname{erfi}(z) = \operatorname{erfcx}(-iz)$ for Float32, and Float64, for complex inputs.
- At least 15 issues resolved on **SpecialFunctions.jl**, which will be filtered for quality

- At least 5 issues resolved on **HyperGeometricFunctions.jl**, which will be filtered for quality

- Full implementation of the **Appell Hypergeometric Functions** [4]

- F_1 is defined for $|x| < 1$, $|y| < 1$ by the series

$$F_1(a, b_1, b_2; c; x, y) = \sum_{m,n=0}^{\infty} \frac{(a)_{m+n}(b_1)_m(b_2)_n}{(c)_{m+n}m!n!} x^m y^n$$

where $(q)_n$ is the Pochhammer symbol.

- F_2 is defined for $|x| + |y| < 1$ by the series

$$F_2(a, b_1, b_2; c_1, c_2; x, y) = \sum_{m,n=0}^{\infty} \frac{(a)_{m+n}(b_1)_m(b_2)_n}{(c_1)_m(c_2)_n m!n!} x^m y^n$$

- F_3 is defined for $|x| < 1$, $|y| < 1$ by the series


$$F_3(a_1, a_2, b_1, b_2; c; x, y) = \sum_{m,n=0}^{\infty} \frac{(a_1)_m(a_2)_n(b_1)_m(b_2)_n}{(c)_{m+n}m!n!} x^m y^n$$

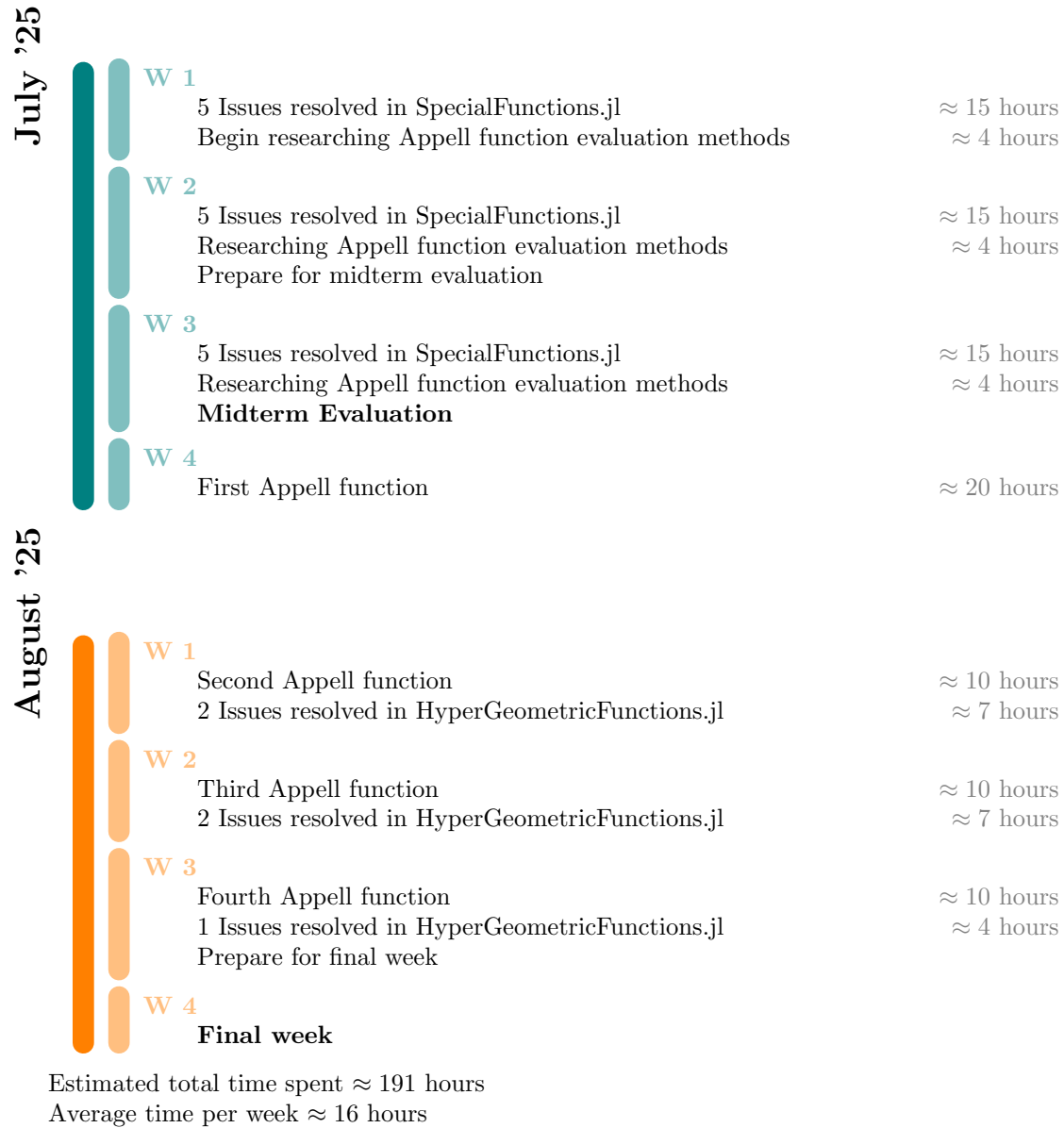
- F_4 is defined for $|x|^{1/2} + |y|^{1/2} < 1$ by the series

$$F_4(a, b; c_1, c_2; x, y) = \sum_{m,n=0}^{\infty} \frac{(a)_{m+n}(b)_{m+n}}{(c_1)_m(c_2)_n m!n!} x^m y^n$$

Timeline

June '25

	W 1	erf(x) for real values for Float32	≈ 4 hours
		erfc(x) for real values for Float32, and Float64	≈ 6 hours
		erf(x) for complex values for Float 32, Float 64	≈ 8 hours
	W 2	erfcx(x) for real values for Float 32, Float 64	≈ 6 hours
		erfi(x) for real values for Float 32, Float 64	≈ 6 hours
	W 3	erfc(x) for complex values for Float 32, Float 64	≈ 6 hours
		erfi(x) for complex values for Float 32, Float 64	≈ 6 hours
		erfcx(x) for complex values for Float 32, Float 64	≈ 6 hours
	W 4	$D(x)$ for real values for Float 32, Float 64	≈ 6 hours
		$D(x)$ for complex values for Float 32, Float 64	≈ 6 hours
		$w(x)$ for complex values for Float 32, Float 64	≈ 6 hours



Expected Results

Complete, functional, high efficiency and accuracy, julia implementations of the deliverables with proper documentation for future work.

Expected difficulties and risks: Poor estimation of difficulty and time duration of individual tasks may lead to late or incomplete results. This effect may snowball in the middle of the coding period and force a restructuring of the project.

References

- [1] Special Functions https://en.wikipedia.org/wiki/Special_functions
- [2] Hypergeometric Functions https://en.wikipedia.org/wiki/Hypergeometric_function
- [3] The Error Function https://en.wikipedia.org/wiki/Error_function
- [4] The Appell Functions https://en.wikipedia.org/wiki/Appell_series
- [5] The SpecialFunctions.jl Julia Library <https://github.com/JuliaMath/SpecialFunctions.jl>
- [6] The HyperGeometricFunctions.jl Julia Library <https://github.com/JuliaMath/HypergeometricFunctions.jl>
- [7] The Remez Algorithm https://en.wikipedia.org/wiki/Remez_algorithm
- [8] The Remez.jl Julia Library <https://github.com/simonbyrne/Remez.jl>
- [9] Sollya: an environment for the development of numerical codes <https://www.sollya.org/>
- [10] Wolfram list of special functions <https://reference.wolfram.com/language/guide/SpecialFunctions.html.en>
- [11] Wolfram list of hypergeometric functions <https://reference.wolfram.com/language/guide/HypergeometricFunctions.html>
- [12] MATLAB list of special functions <https://www.mathworks.com/help/matlab/special-functions.html>
- [13] MATLAB list of hypergeometric functions <https://www.mathworks.com/help/symbolic/sym.hypergeom.html>
- [14] SciPy: Fundamental Algorithms for Scientific Computing in Python <https://docs.scipy.org/doc/scipy/reference/special.html>
- [15] mpmath: a Python library for arbitrary-precision floating-point arithmetic <https://mpmath.org/>
- [16] GNU Scientific Library <https://www.gnu.org/software/gsl/>
- [17] Automatic Differentiation https://en.wikipedia.org/wiki/Automatic_differentiation
- [18] Forward-Mode Automatic Differentiation in Julia <https://github.com/JuliaDiff/ForwardDiff.jl>
- [19] Cephes Mathematical Library <https://www.netlib.org/cephes/>
- [20] ARM Optimized Routines <https://github.com/ARM-software/optimized-routines/tree/master/math>
- [21] AMOS A Portable Package for Bessel Functions of a Complex Argument and Nonnegative Order <https://www.netlib.org/amos/>

- [22] ACM Collected Algorithms <https://www.netlib.org/toms-2014-06-10/>
- [23] The GNU Standard C library <https://sourceware.org/git/?p=glibc.git>
- [24] MPFR: a C library for multiple-precision floating-point computations with correct rounding <https://www.mpfr.org/>
- [25] Larry C. Andrews, "*Special Functions of Mathematics for Engineers*", 2nd Edition
- [26] Michael J. Schlosser, "*Multiple Hypergeometric Series – Appell Series and Beyond*", <https://www.mat.univie.ac.at/~schlosse/pdf/appell-survey.pdf>
- [27] Pearson, J.W., Olver, S. & Porter, M.A. Numerical methods for the computation of the confluent and Gauss hypergeometric functions. Numer Algor 74, 821–866 (2017). <https://doi.org/10.1007/s11075-016-0173-0>