# GSoC 2025: Error Functions and Appell Functions in The Julia Programming Language

Ahmed Y. Kadah[1]
Oscar D. Smith[2]

[1]Student, University of Science and Technology in Zewail City
[2]Mentor, JuliaHub

**Abstract**

Special functions serve a unique purpose in many applications by abstracting otherwise complicated series or equations and allow for careful manipulation through utilizing their well studied properties. Having numerically and computationally stable implementations is important for clear and intuitive abstraction of the functions. The compatibility of the implementations with the surrounding environment's methods and tools is also necessary to ensure coherence. Our work will cover the **SpecialFunctions.jl** and the **HyperGeometricFunctions.jl** Julia libraries which provide implementations/interfaces for special functions and hypergeometric functions. We will work on Julia implementations for the error function and it's variants (erf, erfc, erfcx, erfi, Dawson, Fadeeva), which currently use wrapper function calls to external libraries, as well as a Julia implementation for 2 kinds of the Appell hypergeometric function, which lack implementations in most programming languages.

# Introduction

## About me

Name: Ahmed Yasser Kadah
Github: AhmedYKadah
Email: ahmadyassermo@gmail.com
Project duration: 175 hours (Medium)

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.[1, 2] 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.[1, 3] These functions also have uses in solving highly advanced problems in physics and mathematical analysis.

The **Error Function**, defined as $\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} \mathrm{d}t$. To those with some background in statistics, this integral would look very familiar as it is identical to the integral of a normalized gaussian distribution, scaled by a factor of 2. This function is of high importance in many fields due to it's relation to the gaussian distribution and all applications which rely on the central limit theorem.

The **Appell Hypergeometric Functions** are a set of four series which generalize the Gaussian hypergeometric series. They constitute solutions to a set of defined partial differential equations, and have applications in solving problems in both pure and applied mathematics. Some of these applications include those in quantum field theory, particularly in the evaluation of some Feynman integrals. [22]

## Benefits to Community

Due to the nature and importance of this topic in many applications, there already exists a large number of implementations of special functions (including the error functions and it's variants) in other languages such as Wolfram Mathematica [4, 5], MATLAB [6, 7], Python [8, 9], C [11, 12, 13, 14, 15, 16, 17], 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.

**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, openspecfun, and mpfr through wrapper functions. Meanwhile, HyperGeometricFunctions.jl is purely implemented in julia with several papers cited for the numerical approaches used in the implementations.

One of the things that make this project of interest for a language like Julia is the full compatibility 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 derivates of the elementary arithmetic operations and functions through the repeated usage of the chain rule.[18, 19] In other words, if a function has an algorithm implemented in julia, AutoDiff can "trace" the code to find the elementary operations and implement it's 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.

Further, as we will see later on, the Appell functions have very few implementations. Many languages only have implementations of the Appell function of the first kind $F_1$. The only languages according to our findings which have implementation of all the Appell functions are Python in the mpmath[10] library and Wolfram[5].

## Goals

The goals of the project are to

1. Implement all variants of the error function in Julia
2. Implement the Appell hypergeometric functions in Julia.

The two repositories of interest are SpecialFunctions.jl and HypergeometricFunctions.jl.

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 already implemented in **Bessels.jl**
- All BigFloat implementations make use of the MPFR library [11], but quality of execution is high and there would be no merit behind implementing Julia versions considering the difficulty of implementation
- The majority of the function implementations in SpecialFunctions.jl/erf.jl (the error function variants source code file) use wrapper function calls to the C standard math library [13] so this will be one of our main focuses

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

- Function implementations are already Julia implementations
- 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.[30] 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.

*Expected difficulties and risks:* Part of this project will involve an in-depth literature review to find appropriate methods of computing these special functions and the quality and extent of documentation of existing methods will affect the speed of progress.

# Deliverables and Timeline

## Deliverables

- Full implementation of the **Error Function** and it's varieties [20]
  - Error Function: $\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} \mathrm{d}t$ for Float32 and Float64 (Float64 for real values has been implemented as said previously), for real as well as complex inputs.
  - Complementary Error Function: $\operatorname{erfc}(x) = 1 - \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} \mathrm{d}t$ 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.
- Full implementation of two of the **Appell Hypergeometric Functions** [21, 28, 29]
  - $F_1$ 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 $(q)_n$, the Pochhammer symbol, is defined as

$$(x)_0 = 1$$

$$(x)_n = \underbrace{x(x+1)(x+2)\dots(x+n-1)}_{n \text{ factors}} = \frac{\Gamma(x+n)}{\Gamma(x)}$$

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

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

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

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

  - (Optional) $F_4$ 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

Estimated total time spent ≈ 175 hours
Average time per week ≈ 15 hours

### June '25

**W 1**

| | |
|---|---|
| Literature review on polynomial and rational approximations | ≈ 12 hours |
| erf($x$) for real values for Float32 | ≈ 6 hours |

**W 2**

| | |
|---|---|
| erfc($x$) for real values for Float32, and Float64 | ≈ 6 hours |
| erfi($x$) for real values for Float 32, Float 64 | ≈ 6 hours |
| Literature review on evaluation of complex input special functions | ≈ 8 hours |

**W 3**

| | |
|---|---|
| erfi($x$) for complex values for Float 32, Float 64 | ≈ 12 hours |
| erf($x$) for complex values for Float 32, Float 64 | ≈ 1 hours |

**W 4**

| | |
|---|---|
| erfc($x$) for complex values for Float 32, Float 64 | ≈ 12 hours |

### July '25

**W 1**

| | |
|---|---|
| erfcx($x$) for real values for Float 32, Float 64 | ≈ 6 hours |
| erfcx($x$) for complex values for Float 32, Float 64 | ≈ 12 hours |

**W 2**

| | |
|---|---|
| $D(x)$ for real values for Float 32, Float 64 | ≈ 6 hours |
| $D(x)$ for complex values for Float 32, Float 64 | ≈ 12 hours |

**W 3**

| | |
|---|---|
| Literature review on methods of evaluating Appell functions | ≈ 6 hours |
| $w(x)$ for complex values for Float 32, Float 64 | ≈ 12 hours |
| **Midterm Evaluation** | |

**W 4**

| | |
|---|---|
| First Appell function | ≈ 16 hours |

### August '25

**W 1**

| | |
|---|---|
| First Appell function | ≈ 16 hours |

**W 2**

| | |
|---|---|
| Second Appell function | ≈ 16 hours |

**W 3**

| | |
|---|---|
| Second Appell function | ≈ 16 hours |

**W 4**

**Final week**

# Related Work

### Special Functions

Most notable languages have some form of implementation of the error function and most/all of its varients because it is simply too important to leave out. There are also various C math libraries with Julia interfaces which include special function calls like GSL, openlibm, and openspecfun.

### Hypergeometric Functions

The resources for hypergeometric are a bit less available than special functions. Implementations for the Appell functions are somewhat rare, and mostly restricted to the first kind $F_1$. Wolfram is likely the only language which has all Appell functions built in. Also, python's mpmath library (open source and available for use under BSD license) does have all the variant and could be used as a reference.

# References


[1] Larry C. Andrews, *"Special Functions of Mathematics for Engineers"*, 2nd Edition

[2] Special Functions `https://en.wikipedia.org/wiki/Special_functions`

[3] Hypergeometric Functions `https://en.wikipedia.org/wiki/Hypergeometric_function`

[4] Wolfram list of special functions `https://reference.wolfram.com/language/guide/SpecialFunctions.html.en`

[5] Wolfram list of hypergeometric functions `https://reference.wolfram.com/language/guide/HypergeometricFunctions.html`

[6] MATLAB list of special functions `https://www.mathworks.com/help/matlab/special-functions.html`

[7] MATLAB list of hypergeometric functions `https://www.mathworks.com/help/symbolic/sym.hypergeom.html`

[8] SciPy: Fundamental Algorithms for Scientific Computing in Python `https://docs.scipy.org/doc/scipy/reference/special.html`

[9] mpmath: a Python library for arbitrary-precision floating-point arithmetic `https://mpmath.org/`

[10] mpmath list of hypergeometric functions `https://mpmath.org/doc/current/functions/hypergeometric.html`

[11] MPFR: a C library for multiple-precision floating-point computations with correct rounding `https://www.mpfr.org/`

[12] GNU Scientific Library `https://www.gnu.org/software/gsl/`

[13] The GNU Standard C library `https://sourceware.org/git/?p=glibc.git`

[14] Cephes Mathematical Library https://www.netlib.org/cephes/

[15] ARM Optimized Routines https://github.com/ARM-software/optimized-routines/tree/master/math

[16] AMOS A Portable Package for Bessel Functions of a Complex Argument and Nonnegative Orderhttps://www.netlib.org/amos/

[17] ACM Collected Algorithms https://www.netlib.org/toms-2014-06-10/

[18] Automatic Differentiation https://en.wikipedia.org/wiki/Automatic_differentiation

[19] Forward-Mode Automatic Differentiation in Julia https://github.com/JuliaDiff/ForwardDiff.jl

[20] The Error Function https://en.wikipedia.org/wiki/Error_function

[21] The Appell Functions https://en.wikipedia.org/wiki/Appell_series

[22] Kim, I., & Rathie, A. K. (2023). "*A Note on Certain General Transformation Formulas for the Appell and the Horn Functions*," Symmetry, 15(3), 696. https://doi.org/10.3390/sym15030696

[23] The SpecialFunctions.jl Julia Library https://github.com/JuliaMath/SpecialFunctions.jl

[24] The HyperGeometricFunctions.jl Julia Library https://github.com/JuliaMath/HypergeometricFunctions.jl

[25] The Remez Algorithm https://en.wikipedia.org/wiki/Remez_algorithm

[26] The Remez.jl Julia Library https://github.com/simonbyrne/Remez.jl

[27] Sollya: an environment for the development of numerical codes https://www.sollya.org/

[28] Appell Functions, NIST https://dlmf.nist.gov/16.13

[29] Michael J. Schlosser, *"Multiple Hypergeometric Series – Appell Series and Beyond"*, https://www.mat.univie.ac.at/~schlosse/pdf/appell-survey.pdf

[30] 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