Polynomial Approximation of the Boys Function Optimized for High Performance Computing

A Thesis
Presented to
The Academic Faculty

by

Cory Brzycki

In Partial Fulfillment
of the Requirements for the Degree
Bachelor's in Computer Science
with the Research Option
in the College of Computing

Georgia Institute of Technology
May 2016

Polynomial and Rational Approximation of the Boys Function Optimized for High Performance Computing

Approved by:

Dr. Edmond Chow, Advisor
School of Computational Science and Engineering
Georgia Institute of Technology

Dr. David Sherrill
School of Chemistry and Biochemistry
Georgia Institute of Technology

Date Approved: [Date Approved by Committee]

ACKNOWLEDGEMENTS

I would like to thank my research advisor, Professor Edmond Chow, without whose mentorship and guidance this research would not have been possible.

TABLE OF CONTENTS

	Page
ACKNOWLEDGEMENTS	iii
LIST OF TABLES	v
LIST OF FIGURES	vi
LIST OF SYMBOLS AND ABBREVIATIONS	vii
SUMMARY	viii
CHAPTER	
1 Introduction and Literature Review	1-6
2 Methods	7-9
3 Results	10-14
4 Discussion	15
5 Future Work	16
APPENDIX A: Matlab Program Used to Interface With Chebfun	17
REFERENCES	18

LIST OF TABLES

Page

Table 1: Table of Coefficient Values 13-14

LIST OF FIGURES

Page

Figure 1: Implementation in python for approximate *Boys* Values 8

Figure 2: Implementation in Matlab for approximate *Boys* Values 8

Figure 3: Implementation in Matlab for Chebyshev Nodes 9

Figure 4: Implementation in Matlab for value of *Boys Function* 9

Figure 5: *Boys Function* in a form that can be used with Chebfun Library 10

Figure 6: Approximation of *Boys Function* using Chebfun Library 11

Figure 7: Degree 25 Polynomial of 1st *Boys Function* 11

Figure 8: Degree 25 Polynomial of 32nd *Boys Function* 12

Figure 9: Matlab code outlining how data was collected 12

LIST OF SYMBOLS AND ABBREVIATIONS

HPC	High Performance Computing

SUMMARY

This study provides a method for finding a polynomial approximation of the *Boys Function* on a given arbitrary domain to a given arbitrary degree. Such an approximation would allow computer conducted that involves the *Boys Function* to be sped up through code parallelization. Current methods for evaluating the *Boys Function* rely on branching through division of domain that prevents parallelization. *Remez* algorithm is used to provide an approximation with coefficients for a degree 20 polynomial approximation listed for the first 32 *Boys Functions*. Matlab code is provided with directions on use and links to installation of libraries to allow other coefficients to be determined.

CHAPTER 1

INTRODUCTION AND LITERATURE REVIEW

Introduction

High performance computing is the development of software intended to speed up slow running processes. Processes are usually sped up by creating code optimized to run effectively on specific hardware and through the development of numerical approximations for complex functions. Often, high performance computing research is undertaken with the intent to aid in modeling research. Complex systems require complex models that often have associated difficult-to-evaluate functions. These calculations can take days.

When working with models on the molecular scale, molecular integrals must be evaluated. Evaluation of these molecular integrals is very costly and leads to long computation times. Numerical methods have been applied to convert molecular integrals into forms that are more apt to being evaluated on a computer; however, evaluation of these numerical approximations is still costly. Calculations of molecular integral approximation functions tend to rely heavily on interval-based divisions, which lead to branching in computer programs. The presence of these branching conditions prevents the code from being optimized for parallel computation. Since modeling relies on iterative processes, being unable to parallelize computations causes a large hit to performance.

One such type of molecular integral is the coulomb integrals that are used in calculations associated with two-electron interactions. These integrals are typically evaluated through the use of an incomplete gamma function (the *Boys Function)* [5]. It is

not practical to provide an exact evaluation of the incomplete gamma functions when modeling interactions. The cost of its evaluation is extremely high, and in molecular models many two-electron interactions must be considered. Fruchtl and Otto investigated optimization of the *Boys Function* for evaluation on vector computers. Most importantly, they provide a rational form of the *Boys Function*. The relevance of their research has declined as modern modeling relies on the development of parallelized code that can be run on graphics processing units. Schaad and Morrell have also conducted research on the *Boys Function* and provided detailed analysis of the function as well as tables of calculated values (up to the 16th degree of the function) [4].

Previous analysis of the *Boys Function* and its optimization for computer use fail to provide methods that are favorable for parallelized computing. Until a numerical approximation of the function is developed that removes the dependency on interval divided calculations, the *Boys Function* remains unable to be optimized for parallelized computation. This research intends to provide and analyze a polynomial and rational approximation of the function that eliminates the need for an interval division. Such approximations would allow code for a numerical approximation of the *Boys Function* to be developed and optimized for parallelized computing. This would decrease the computation time of molecular models that rely on the evaluation of many two-electron interactions. This study intends to apply classical numerical methods, such as *Remez Algorithm*, to the approximation functions provided by Fruchtl and Otto to develop a polynomial and rational approximation of the *Boys Function* that eliminates the need for an interval division.

Literature Review

The *Boys Function* is used to evaluate coulomb integrals that come up in the calculations associated with two-electron interactions [5]. In the modeling of molecular systems, such integrals are often evaluated, leading to heavy use of the *Boys Function*. It is impractical to provide an exact evaluation of this function since it would slow down the computations necessary for such models. Instead, these models use approximations of the *Boys Function* for their calculations. Previous researchers have investigated and provided approximations for the function that are applicable to their areas of research and available computer hardware.

Fruchtl and Otto investigated the *Boys Function* and worked towards optimization for evaluation on vector computers. They provide derivation of the *Boys Function* in a rational form, which serves a good foundation for the investigation of a rational approximation of the function. The relevance of their research has declined as modern modeling relies on the development of parallelized code that can be run on GPUs. Although their research was able to provide an approximation for the function, it relies heavily on interval-based divisions. Additionally, the rational form provided has the peculiar property that the numerator and denominator are not fully independent of each other [5]. Schaad and Morrell also conducted research on the *Boys Function* and provided a detailed analysis of the function as well as tables of calculated values. The tables provided in this paper serve as a good basis of comparison for estimated weights of the *Boys Function* and will be used to validate the approximations investigated in this study. The methods outlined in this paper also rely on restrictions based on interval ranges [4].

Relying on interval-based division leads to the presence of branching conditions in computer programs that perform calculations using existing approximations of the *Boys Function*. The presence of these branching conditions prevents the code from being optimized for parallel computation. Since modeling relies on iterative processes, being unable to parallelize computations causes a large hit to performance.

Bailey and Borwein have researched computation of the incomplete Gamma function and its application in numerical methods of computation. They provide information on the manual computation of special functions and their translation to the *Lerch transcendent*. The paper continues by posing observations of the *Lerch Transcendent* and how they led to special properties useful for computation. The paper then provides evaluation for common algorithms used for the computation of the *Lerch Transcendent; namely the Bernoulli-series Algorithm, Erdelyi-series algorithm, Riemann-splitting algorithm*. Additionally, explanation of the analytical evaluation methods used in computation of the *Incomplete Gamma Function* is provided [1]. The relation between the *Boys Function* and *Incomplete Gamma Function* makes the algorithms presented and developed in this paper a good starting point for the discovery of an approximation of the *Boys Function*. However, this study is not focused on providing fast to compute algorithms, which is the main focus of the current study at hand.

Although past researchers have provided approximations for the *Boys Function*, these approximations are far from ideal for high performance computing. It is necessary for an approximation to be discovered that does not rely on branching to allow an optimized parallelized implementation to be created. This study will expand on past

analysis of the *Boys Function* in an attempt to remove the branching conditions. Additionally, this study will investigate the use of existing numerical methods to approximate the *Boys Function*; with a heavy focus placed on *Remez's Algorithm*.

Ricardo and Lloyd have worked on the development of an implementation of *Remez's Algorithm* in matlab. Furthermore, possible extensions to allow the use of their implementation on rational functions are presented. This well-known algorithm is used to find approximations for complex functions [3]. This study plans on applying the methods outlined by this paper to the *Boys Function*. However, it is unknown how the methods will perform when applied to the rational form of the *Boys Function* discovered by Fruchtl and Otto. Most likely the dependency between the numerator and denominator will lead to issues with using the *Barycentric-Remez Algorithm*, and further investigation will be necessary.

Belogus and Liron explored "DCR2", the composition of R*emez's Second Algorithm* and *Differential Correction Method* for l∞ rational approximations [2]. However, "DCR2" has the same potential to lead to ill-defined behavior since the values of the numerator and denominator are not independent of each other.

Previous analysis of the *Boys Function* fail to provide methods that are favorable for parallelized computing. Until a numerical approximation of the function is developed that removes the dependency on interval divided calculations, the *Boys Function* remains unable to be optimized for parallelized computation. It is important that a parallelized approximation for the function be discovered in order to expedite research in fields of molecular studies. Molecular models that have numerous two-electron interactions would greatly benefit from code parallelization. A speedup in computation time would lead to a

decrease in the total time needed for evaluations of model and allow research dependent on the models to be conducted in a more timely manner.

The study at hand intends to provide and analyze a polynomial and rational approximation of the function that eliminates the need for an interval division. Such approximations would allow code for a numerical approximation of the *Boys Function* to be developed and optimized for parallelized computing. This would decrease the computation time of molecular models that rely on the evaluation of many two-electron interactions and benefit research that rely on such models.

CHAPTER 2

METHODS

Remez Algorithm

The *Remez Algorithm* is an iterative algorithm that provides a minimax approximation. It relies on minimizing the maximum error of an approximation. This algorithm relies on starting with a set of initial data points, or polynomial, and iteratively solving a system of linear equations until the error term converges (or begins to vary by an amount less than a provided delta parameter).

$$b_0 + b_i x_i + \cdots + b_n x_i^n + (-1)^i E = f(x_i)$$
$$\text{With } X = \text{set of } n+2 \text{ starting points.}$$

In the second step of the algorithm, known as the exchange step, the control points are continually exchanged with extrema values in intervals to come closer to a minimax approximation, that is minimizing the maximum error between the approximation and the actual function.

Programming Languages and Methodologies

In order to perform the iterative calculations needed in the *Remez's Algorithm*, it is best to exploit the power of computational engines and programming languages. Python, and Matlab were used. Python's ease of use makes it ideal for helping to generate possible starting points. Matlab's ability to manipulate matrices allows it to easily solve the systems of equations created by the control points. A Matlab library, Chebfun, is useful because of its many already implemented approximation methods. Many of the methods it provides are based on Chebyshev polynomials.

Choice of Starting Points or Polynomial

A focus of this study is the evaluation of various starting points or starting polynomials to input in to *Remez's Algorithm*. The starting points and polynomials used are those found in classical literature as common approximation for the *Boy's Function*, as well as those used in other minmax approximations.

The traditional approximation of the *Boys Function* evaluated at a given point.

```
# Probably a better way
def BoysValue(n, x):
    F = []

    if x == mp.mpf("0"):
        for i in range(0, n+1):
            F.append(mp.mpf(1.0)/(mp.mpf(2.0*i+1)))
    else:
        for i in range(0, n+1):
            N = i+mp.mpf("0.5")
            F.append(mp.gammainc(N, 0, x) * 1.0/(2.0 * mp.power(
                x, N)))

return F
```

Figure 1: Implementation in python for approximate *Boys* Values

```
function [nodes] = IntegralApproximation(n, degree, upperBoundX)
    nodes = [];
    i = 1;
    l = linspace(0,upperBoundX, degree + 2);
if n == 0
    while (i <= degree + 2)
        hold = 1/(2.0*l(i)+1);
        nodes = cat(degree, boysCheb, hold);
        i = i + 1
    end
else
    while(i <= degree + 2)
        N = l(i) + .5;
        gamStr = strcat('@(t)t.^',num2str(N-1),'.*exp(-t)');
        gamFun = str2func(gamStr);
        hold = integral(gamFun,0, n) * 1/(2 * n^N);
        nodes = cat(1, nodes, hold);
        i = i + 1;
    end
end
end
```

Figure 2: Implementation in Matlab for approximate *Boys* Values

The Chebyshev polynomials are often used in minmax approximations.

```
function [nodes] = ChebyshevNodes(upperBoundX, degree)
nodes = [];
    for k=1:degree+2
        hold = .5*(upperBoundX+0)+.5*(upperBoundX-0)*cos((2*k-1)
            *pi/(2*degree+4));
        nodes = cat(1,nodes, hold);
    end
end
```

Figure 3: Implementation in Matlab for Chebyshev Nodes

The actual values of the *Boys Function*.

```
function [F] = actualValues(n, degree, upperBoundX)
        q = 1;
        boysPoints= [];
        l = linspace(0,upperBoundX, degree);
        disp(length(l))
        while(q <= length(l))
                boysAsString = strcat('@(t)exp(-',num2str(l(q)),
                    '.*t.^2).*t.^(.2*',num2str(n),')');
                boysFunc = str2func(boysAsString);
                hold = integral(boysFunc,0, 1);
                boysPoints = cat(1, boysPoints, hold);
                q = q + 1;
        end
        F = boysPoints;
end
```

Figure 4: Implementation in Matlab for value of *Boys Function*

CHAPTER 3

RESULTS

Writing the Boys Function as a Math Function in Matlab

The *Boy's Function*:

$$F_n(x) = \int_0^1 t^{2n} e^{-xt^2}$$

In order to be evaluated in Matlab, the function must be written as an integral of another function. This was accomplished with the following code in Matlab:

```
% The nth boys function
f = @(x,t) (exp(-x.*t.^2).*t.^(.2*n))
% The limits of integration
a = 0;
b = 1;
% Define g as the integral of f(x,t) dt from a to b
g = @(x) integral(@(t) f(x,t) , a,b);
```

Figure 5: *Boys Function* in a form that can be used with *Chebfun*

This easily allows the *nth Boys Function* to be evaluated at a given point or graphed. An additional benefit of writing the function in this manner is that it allows it to be easily passed into the Chebfun library so that its methods may be used to provide the *Remez* approximation for a polynomial function of a given degree.

Approximation of Boys Function By Chebfun's Remez

In order to use the *Boys Function* with the Chebfun library it must be made in to a Chebfun object. After the Chebfun object is created, its implementation of *Remez* can be used to find a polynomial approximation.

```
% Make the chebfun object for interval from intervalStart to intervalEnd
y = chebfun(g,[intervalStart,intervalEnd]);
% Find the polynomial up to degree requested
[p,err] = remez(y,degree);
% Plot the graph
figure
plot(y,'b', p,'r','linewidth',1)
title('Function and best polynomial approximation','fontsize',14)
```

Figure 6: Approximation of *Boys Function* using Chebfun Library

Graphs of the Approximation for Different Orders (for n = 1, 32 on [0, 100])

Figure 7: Degree 25 Polynomial of 1st *Boys Function*

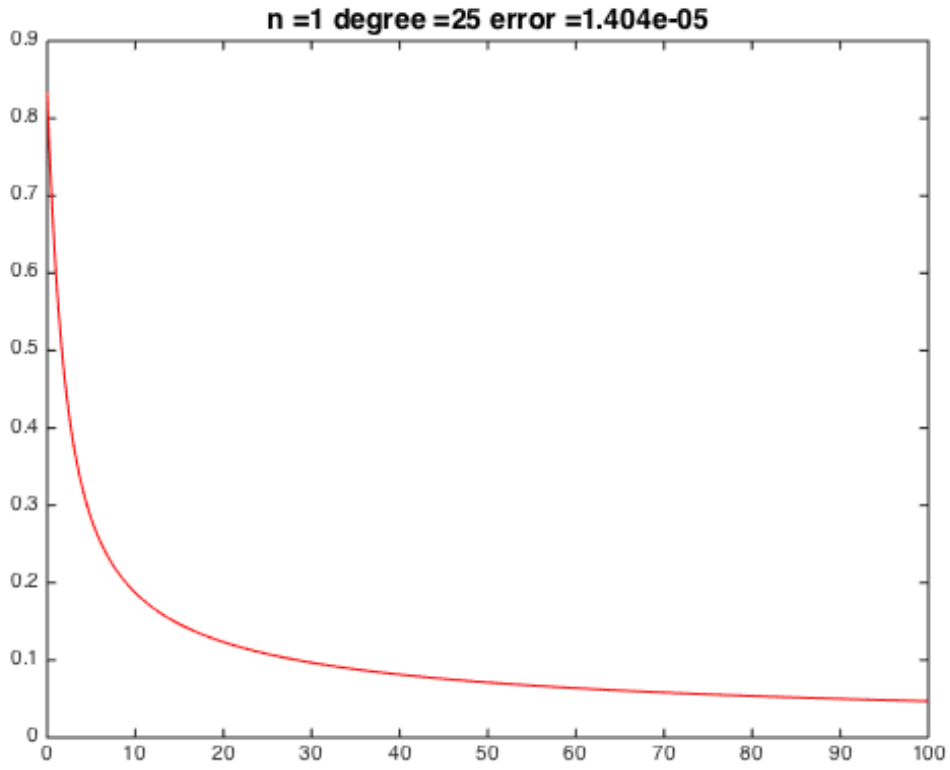
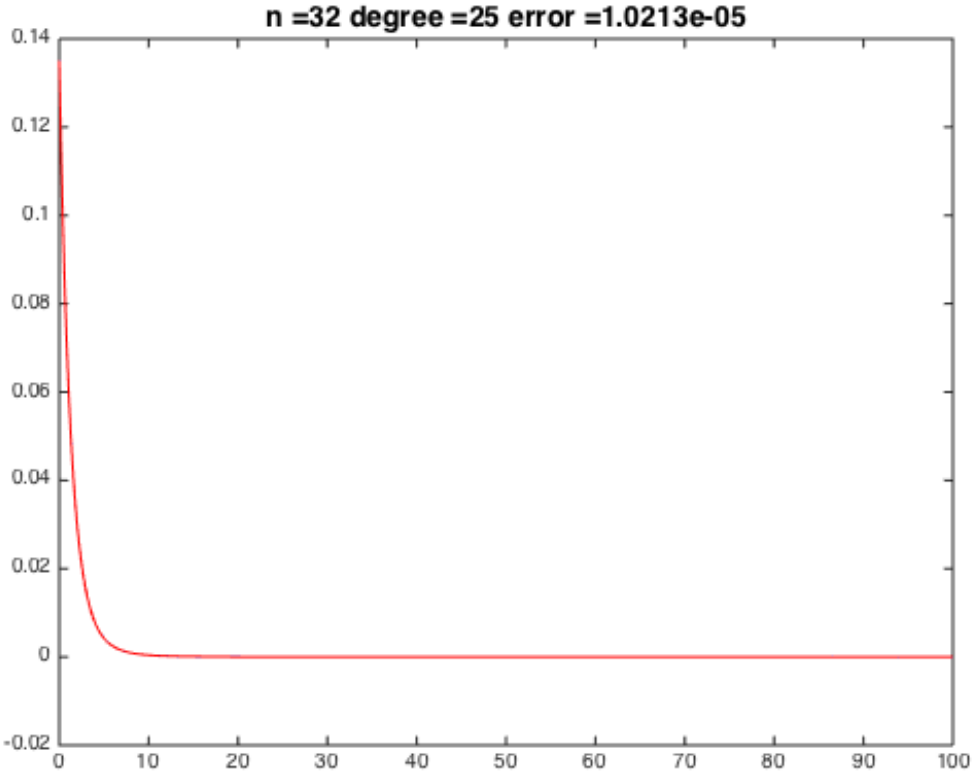


Figure 8: Degree 25 Polynomial of 32nd *Boys Function*



Coefficients up to the 32nd Order of the Boys Function

On the following pages is a table containing the coefficients for the 20th degree polynomial approximation of the *Boys Function* on the interval from 0 to 100 is included.

The code used to generate this data is below:

```
for n = 1: 32
        % The nth boys function
        f = @(x,t) (exp(-x.*t.^2).*t.^(.2*n));
        % The limits of integration
        a = 0;
        b = 1;
        % Define g as the integral of f(x,t) dt from a to b
        g = @(x) integral(@(t) f(x,t) , a,b);
        % Make the chebfun object for interval from intervalStart to intervalEnd
        y = chebfun(g,[0,100]);
        % Find the polynomial up to degree requested
        [p,err] = remez(y, 20);
        approxPolynomials(n) = poly2sym(poly(p));
        errorTerms(n) = err;
end
```

Figure 9: Matlab code outlining how data was collected

N	Error	x^0	x^1	x^2	x^3	x^4	x^5	x^6	x^7	x^8	x^9
1	2.12E-04	0.83312123	-0.3101055	0.09163042	-0.019688671	0.003081189	-0.000354614	3.04327E-05	-1.97701E-06	9.8587E-08	-3.81767E-09
2	2.09E-04	0.71407695	-0.2917597	0.0881364	-0.019112861	0.003004916	-0.000346727	2.98023E-05	-1.93802E-06	9.67092E-08	-3.74682E-09
3	2.06E-04	0.62479447	-0.2754555	0.08489492	-0.018568654	0.002932139	-0.000339162	2.91957E-05	-1.90041E-06	9.4896E-08	-3.67833E-09
4	2.02E-04	0.55535317	-0.2608702	0.08187974	-0.018053565	0.00286263	-0.0003319	2.86116E-05	-1.86414E-06	9.31444E-08	-3.61212E-09
5	1.99E-04	0.49980067	-0.247746	0.07906811	-0.017565362	0.002796183	-0.000324924	2.80488E-05	-1.82912E-06	9.14515E-08	-3.54806E-09
6	1.96E-04	0.4543491	-0.2358739	0.07644024	-0.017102032	0.002732605	-0.000318218	2.75063E-05	-1.7953E-06	8.98146E-08	-3.48607E-09
7	1.93E-04	0.4164732	-0.2250833	0.07397881	-0.016661758	0.002671721	-0.000311768	2.69831E-05	-1.76262E-06	8.8231E-08	-3.42604E-09
8	1.91E-04	0.38442473	-0.2152329	0.0716686	-0.016242892	0.002613369	-0.000305559	2.64781E-05	-1.73103E-06	8.66985E-08	-3.36791E-09
9	1.88E-04	0.35695494	-0.2062052	0.06949618	-0.015843937	0.002557398	-0.000299579	2.59905E-05	-1.70048E-06	8.52146E-08	-3.31157E-09
10	1.85E-04	0.33314809	-0.1979013	0.06744968	-0.015463534	0.00250367	-0.000293816	2.55195E-05	-1.67092E-06	8.37773E-08	-3.25696E-09
11	1.83E-04	0.31231735	-0.1902377	0.06551853	-0.015100441	0.002452058	-0.00028826	2.50642E-05	-1.64231E-06	8.23845E-08	-3.204E-09
12	1.80E-04	0.29393753	-0.1831432	0.06369332	-0.014753524	0.002402443	-0.000282898	2.4624E-05	-1.61459E-06	8.10343E-08	-3.15263E-09
13	1.78E-04	0.27760012	-0.1765567	0.06196562	-0.014421746	0.002354714	-0.000277723	2.41981E-05	-1.58775E-06	7.97249E-08	-3.10277E-09
14	1.75E-04	0.26298264	-0.1704256	0.06032789	-0.014104153	0.002308769	-0.000272724	2.37858E-05	-1.56172E-06	7.84545E-08	-3.05436E-09
15	1.73E-04	0.24982709	-0.1647044	0.05877332	-0.013799871	0.002264513	-0.000267893	2.33866E-05	-1.53649E-06	7.72216E-08	-3.00735E-09
16	1.71E-04	0.23792462	-0.1593534	0.05729578	-0.013508096	0.002221856	-0.000263223	2.29999E-05	-1.51202E-06	7.60245E-08	-2.96168E-09
17	1.68E-04	0.22710433	-0.1543378	0.05588974	-0.013228085	0.002180717	-0.000258705	2.26251E-05	-1.48827E-06	7.48619E-08	-2.91729E-09
18	1.66E-04	0.21722509	-0.1496271	0.05455015	-0.012959152	0.002141018	-0.000254332	2.22618E-05	-1.46521E-06	7.37323E-08	-2.87414E-09
19	1.64E-04	0.20816924	-0.1451943	0.05327245	-0.012700665	0.002102686	-0.000250098	2.19093E-05	-1.44282E-06	7.26344E-08	-2.83217E-09
20	1.62E-04	0.19983798	-0.1410156	0.05205248	-0.012452036	0.002065655	-0.000245997	2.15672E-05	-1.42107E-06	7.15669E-08	-2.79134E-09
21	1.60E-04	0.19214769	-0.1370698	0.05088644	-0.012212722	0.00202986	-0.000242023	2.12352E-05	-1.39993E-06	7.05287E-08	-2.75161E-09
22	1.58E-04	0.18502716	-0.133338	0.04977086	-0.011982217	0.001995243	-0.00023817	2.09128E-05	-1.37937E-06	6.95187E-08	-2.71294E-09
23	1.56E-04	0.17841534	-0.1298033	0.04870256	-0.011760051	0.001961748	-0.000234432	2.05996E-05	-1.35939E-06	6.85357E-08	-2.67528E-09
24	1.54E-04	0.17225959	-0.1264504	0.04767861	-0.011545788	0.001929323	-0.000230805	2.02952E-05	-1.33995E-06	6.75787E-08	-2.6386E-09
25	1.52E-04	0.1665143	-0.1232658	0.04669633	-0.011339019	0.001897917	-0.000227285	1.99992E-05	-1.32103E-06	6.66468E-08	-2.60286E-09
26	1.51E-04	0.16113976	-0.1202371	0.04575325	-0.011139365	0.001867487	-0.000223866	1.97114E-05	-1.30261E-06	6.5739E-08	-2.56802E-09
27	1.49E-04	0.1561012	-0.1173531	0.04484707	-0.01094647	0.001837986	-0.000220544	1.94315E-05	-1.28467E-06	6.48544E-08	-2.53407E-09
28	1.47E-04	0.15136808	-0.1146038	0.0439757	-0.010760002	0.001809376	-0.000217317	1.9159E-05	-1.26721E-06	6.39923E-08	-2.50095E-09
29	1.45E-04	0.14691344	-0.1119798	0.04313717	-0.01057965	0.001781616	-0.000214178	1.88938E-05	-1.25018E-06	6.31517E-08	-2.46866E-09
30	1.44E-04	0.14271341	-0.109473	0.04232968	-0.010405122	0.001754671	-0.000211126	1.86355E-05	-1.2336E-06	6.2332E-08	-2.43714E-09
31	1.42E-04	0.13874677	-0.1070755	0.04155155	-0.010236145	0.001728506	-0.000208157	1.83839E-05	-1.21742E-06	6.15323E-08	-2.40639E-09
32	1.41E-04	0.13499459	-0.1047805	0.04080121	-0.010072462	0.001703088	-0.000205268	1.81388E-05	-1.20165E-06	6.07521E-08	-2.37637E-09

N	x^10	x^11	x^12	x^13	x^14	x^15	x^16	x^17	x^18	x^19	x^20
1	1.15797E-10	-2.766E-12	5.21346E-14	-7.738E-16	8.9858E-18	-8.061E-20	5.4709E-22	-2.713E-24	9.2725E-27	-1.9505E-29	1.9027E-32
2	1.1369E-10	-2.7165E-12	5.12123E-14	-7.603E-16	8.8298E-18	-7.922E-20	5.3772E-22	-2.667E-24	9.1153E-27	-1.9176E-29	1.8708E-32
3	1.11652E-10	-2.6685E-12	5.03194E-14	-7.472E-16	8.6787E-18	-7.788E-20	5.2864E-22	-2.622E-24	8.963E-27	-1.8856E-29	1.8397E-32
4	1.0968E-10	-2.6221E-12	4.94546E-14	-7.344E-16	8.5323E-18	-7.657E-20	5.1984E-22	-2.579E-24	8.8152E-27	-1.8547E-29	1.8096E-32
5	1.07771E-10	-2.5772E-12	4.86168E-14	-7.221E-16	8.3903E-18	-7.531E-20	5.113E-22	-2.537E-24	8.6718E-27	-1.8246E-29	1.7804E-32
6	1.05923E-10	-2.5336E-12	4.78047E-14	-7.102E-16	8.2527E-18	-7.408E-20	5.0302E-22	-2.496E-24	8.5327E-27	-1.7955E-29	1.7521E-32
7	1.04132E-10	-2.4914E-12	4.70173E-14	-6.986E-16	8.1191E-18	-7.289E-20	4.9498E-22	-2.456E-24	8.3976E-27	-1.7672E-29	1.7245E-32
8	1.02397E-10	-2.4504E-12	4.62535E-14	-6.874E-16	7.9895E-18	-7.173E-20	4.8717E-22	-2.418E-24	8.2665E-27	-1.7397E-29	1.6978E-32
9	1.00714E-10	-2.4107E-12	4.55123E-14	-6.764E-16	7.8636E-18	-7.061E-20	4.7959E-22	-2.38E-24	8.139E-27	-1.713E-29	1.6718E-32
10	9.90816E-11	-2.3722E-12	4.47929E-14	-6.658E-16	7.7414E-18	-6.952E-20	4.7223E-22	-2.344E-24	8.0152E-27	-1.687E-29	1.6466E-32
11	9.74979E-11	-2.3348E-12	4.40943E-14	-6.556E-16	7.6226E-18	-6.846E-20	4.6507E-22	-2.309E-24	7.8948E-27	-1.6618E-29	1.622E-32
12	9.59607E-11	-2.2985E-12	4.34157E-14	-6.456E-16	7.5072E-18	-6.743E-20	4.5811E-22	-2.274E-24	7.7778E-27	-1.6372E-29	1.5981E-32
13	9.44681E-11	-2.2632E-12	4.27563E-14	-6.358E-16	7.3951E-18	-6.643E-20	4.5135E-22	-2.241E-24	7.6639E-27	-1.6133E-29	1.5749E-32
14	9.30182E-11	-2.2289E-12	4.21153E-14	-6.264E-16	7.2859E-18	-6.546E-20	4.4477E-22	-2.208E-24	7.5531E-27	-1.5901E-29	1.5523E-32
15	9.16094E-11	-2.1956E-12	4.1492E-14	-6.172E-16	7.1798E-18	-6.451E-20	4.3836E-22	-2.177E-24	7.4453E-27	-1.5675E-29	1.5303E-32
16	9.024E-11	-2.1632E-12	4.08858E-14	-6.083E-16	7.0765E-18	-6.359E-20	4.3212E-22	-2.146E-24	7.3403E-27	-1.5454E-29	1.5088E-32
17	8.89086E-11	-2.1316E-12	4.0296E-14	-5.996E-16	6.976E-18	-6.269E-20	4.2605E-22	-2.116E-24	7.238E-27	-1.524E-29	1.488E-32
18	8.76135E-11	-2.101E-12	3.97219E-14	-5.911E-16	6.8781E-18	-6.181E-20	4.2014E-22	-2.087E-24	7.1384E-27	-1.5031E-29	1.4676E-32
19	8.63535E-11	-2.0711E-12	3.91631E-14	-5.828E-16	6.7828E-18	-6.096E-20	4.1438E-22	-2.058E-24	7.0413E-27	-1.4827E-29	1.4478E-32
20	8.51272E-11	-2.0421E-12	3.86188E-14	-5.748E-16	6.6899E-18	-6.013E-20	4.0877E-22	-2.03E-24	6.9467E-27	-1.4629E-29	1.4284E-32
21	8.39333E-11	-2.0137E-12	3.80886E-14	-5.67E-16	6.5994E-18	-5.932E-20	4.0329E-22	-2.003E-24	6.8544E-27	-1.4435E-29	1.4096E-32
22	8.27706E-11	-1.9862E-12	3.7572E-14	-5.593E-16	6.5111E-18	-5.853E-20	3.9796E-22	-1.977E-24	6.7645E-27	-1.4246E-29	1.3912E-32
23	8.1638E-11	-1.9593E-12	3.70684E-14	-5.519E-16	6.4251E-18	-5.777E-20	3.9275E-22	-1.951E-24	6.6767E-27	-1.4062E-29	1.3733E-32
24	8.05344E-11	-1.9331E-12	3.65775E-14	-5.446E-16	6.3412E-18	-5.702E-20	3.8768E-22	-1.926E-24	6.591E-27	-1.3882E-29	1.3558E-32
25	7.94587E-11	-1.9076E-12	3.60987E-14	-5.376E-16	6.2593E-18	-5.628E-20	3.8272E-22	-1.901E-24	6.5074E-27	-1.3707E-29	1.3387E-32
26	7.84099E-11	-1.8827E-12	3.56316E-14	-5.307E-16	6.1795E-18	-5.557E-20	3.7789E-22	-1.878E-24	6.4258E-27	-1.3535E-29	1.322E-32
27	7.73871E-11	-1.8584E-12	3.51759E-14	-5.239E-16	6.1015E-18	-5.487E-20	3.7316E-22	-1.854E-24	6.3462E-27	-1.3368E-29	1.3057E-32
28	7.63893E-11	-1.8347E-12	3.47311E-14	-5.173E-16	6.0254E-18	-5.419E-20	3.6855E-22	-1.831E-24	6.2683E-27	-1.3204E-29	1.2898E-32
29	7.54158E-11	-1.8115E-12	3.42969E-14	-5.109E-16	5.951E-18	-5.353E-20	3.6405E-22	-1.809E-24	6.1923E-27	-1.3045E-29	1.2742E-32
30	7.44656E-11	-1.789E-12	3.38729E-14	-5.047E-16	5.8784E-18	-5.288E-20	3.5965E-22	-1.787E-24	6.118E-27	-1.2889E-29	1.259E-32
31	7.3538E-11	-1.7669E-12	3.34588E-14	-4.985E-16	5.8075E-18	-5.224E-20	3.5535E-22	-1.766E-24	6.0454E-27	-1.2736E-29	1.2441E-32
32	7.26322E-11	-1.7454E-12	3.30542E-14	-4.925E-16	5.7381E-18	-5.162E-20	3.5115E-22	-1.745E-24	5.9744E-27	-1.2587E-29	1.2296E-32

CHAPTER 4

DISCUSSION

The results of this study show that it is possible to provide an accurate polynomial approximation of the *Boys Function* using *Remez*. *Remez* was shown to be successful for providing approximations for various orders of the *Boys Function* on relatively large intervals.

The results of this study will allow the *Boys Function* to be parallelized since the branching previously necessary for analytical evaluation has been removed. This means that research in other fields that make use of the *Boys Function* may be able to be completed in a timelier manner. Furthermore, the flexibility of the implementation provided allows for different intervals and degrees of accuracy, allowing future researchers to make the choice between trading cost of computation for accuracy.

This study, however, failed to provide a rational approximation of the *Boys Function*. Both implementations made by the author as well as those provided in *Chebfun* led to the creation of near singular matrices and therefore non-usable results.

CHAPTER 5

FUTURE WORK

Extensions of this study could work towards successfully providing a rational approximation of the *Boys Function*. Implementations of *Remez* for rational approximation written by both the author and the one provided in *Chebfun* fail to provide an approximation due to a growing gap between interpolant values. As the iteration of the algorithm runs, the sign of the weights of the approximate values should alternate. However, when the rational approximation is run, this is not the case. Neighboring values tend to have the same sign, which leads to a breakdown in the algorithm. The error fails to converge. This is most likely because of the round-off error associated with determining the rational approximation. A good approach would be to split the *Boys Function* into a dominating and non-dominating part and then running *Remez* on each of these parts. This may lead to an approximation that converges that would still allow the function to be evaluated in parallel. Another extension of this study could involve finding a polynomial approximation of the *Boys Function* that is not restricted to a domain. Although the methods outlined in this study allow arbitrary large domains to be used, it is unable to provide an approximation that works for all domains.

APPENDIX A

MATLAB PROGRAM USED TO INTERFACE WITH CHEBFUN

```
% nth boys function
% degree is the degree of the aproximate polynomial
n = 10;
degree = 5;
% intervalStart is the start of the interval
% intervalEnd is the end of the interval
intervalStart = 0;
intervalEnd = 100;
polyNomails = sym(zeros(1, 32));
errorTerms = zeros(1, 32);

for n = 1: 32
        % The nth boys function
        f = @(x,t) (exp(-x.*t.^2).*t.^(.2*n));
        % The limits of integration
        a = 0;
        b = 1;
        % Define g as the integral of f(x,t) dt from a to b
        g = @(x) integral(@(t) f(x,t) , a,b);
        % Make the chebfun object for interval from intervalStart to intervalEnd
        y = chebfun(g,[intervalStart,intervalEnd]);
        % Find the polynomial up to degree requested
        [p,err] = remez(y, degree);
        polyNomails(n) = poly2sym(poly(p));
        errorTerms(n) = err;
        %Uncomment Below to plot the graphs
        %figure
        %plot(y,'b', p,'r','linewidth',1)
        %gTitle = strcat('n = ',num2str(n),' degree = ',num2str(degree), ' error = ',
            num2str(err));
        %title(gTitle,'fontsize',14)
end
```

Download Link for chebfun: http://www.chebfun.org/download/

To use above code:

Set *degree* to the desired degree of the approximate polynomial
Set *intervalStart* to the start of the domain
Set *intervalEnd* to the end of the domain

Post Execution:

polyNomails contains an approximation of the *n*th function at index *n*
errorTerms contains the maximum error of the *n*th function at index *n*

REFERENCES

[1] D. H. Bailey and J. M. Borwein, "Crandall's computation of the incomplete Gamma function and the Hurwitz zeta function, with applications to Dirichlet L-series," Applied Mathematics and Computation, vol. 268, pp. 462-477, 10/1/1 October 2015 2015.

[2] D. Belogus and N. Liron, "DCR2: An improved algorithm for l rational approximation on intervals," Numerische Mathematik, vol. 31, p. 17, 09// 1978.

[3] P. Ricardo and T. Lloyd, "Barycentric-Remez algorithms for best polynomial approximation in the chebfun system," BIT: Numerical Mathematics, vol. 49, pp. 721-741, 12// 2009.

[4] L. J. Schaad and G. O. Morrell, "Approximations for the Functions Fm(z) Occurring in Molecular Calculations with a Gaussian Basis," The Journal of Chemical Physics, vol. 54, 1971.

[5] H. Fruchtl and P. Otto, "A NEW ALGORITHM FOR THE EVALUATION OF THE INCOMPLETE-GAMMA FUNCTION ON VECTOR COMPUTERS," Acm Transactions on Mathematical Software, vol. 20, pp. 436-446, Dec 1994.