Fast algorithms for Jacobi expansions via nonoscillatory phase functions

James Bremer[a], Haizhao Yang[b]

[a]*Department of Mathematics, University of California, Davis*

[b]*Department of Mathematics, National University of Singapore*

Abstract

We describe a suite of fast algorithms for evaluating Jacobi polynomials, applying the corresponding discrete Sturm-Liouville eigentransforms and calculating Gauss-Jacobi quadrature rules. Our approach, which applies in the case in which both of the parameters α and β in Jacobi's differential equation are of magnitude less than $1/2$, is based on the well-known fact that in this regime Jacobi's differential equation admits a nonoscillatory phase function which can be loosely approximated via an affine function over much of its domain. We illustrate this with several numerical experiments, the source code for which is publicly available.

Keywords: fast algorithms, fast special function transforms, butterfly algorithms, special functions, nonoscillatory phase functions, asymptotic methods

1. Introduction

Expansions of functions in terms of orthogonal polynomials are widely used in applied mathematics, physics, and numerical analysis. And while highly effective methods for forming and manipulating expansions in terms of Chebyshev and Legendre polynomials are available, there is substantial room for improvement in the analogous algorithms for more general classes of Jacobi polynomials.

Here, we describe a collection of algorithms for forming and manipulating expansions in Jacobi polynomials. This includes methods for evaluating Jacobi polynomials, applying the forward and inverse Jacobi transforms, and for calculating Gauss-Jacobi quadrature rules. Our algorithms, which apply when the magnitude of the parameters α and β in Jacobi's differential equation are strictly bounded above by $1/2$, are based on the well-known observation that in this regime Jacobi's differential equation admits a nonoscillatory phase function which can be loosely approximated in much of the interval $[-1, 1]$ by an affine function. We combine this observation with a fast technique for computing nonoscillatory phase functions and standard methods for exploiting the rank deficiency of matrices which represent smooth functions to obtain our algorithms.

The algorithms described here for the numerical evaluation of the Jacobi polynomials and for the numerical calculation of their roots are related to those of [5], [7], and [8]. In [5], a precomputed expansion of the phase function for Bessel's differential equation is used to rapidly evaluate the Bessel functions of the first and second kinds of orders between 0 and 10^9 for arguments on the real

Email addresses: bremer@math.ucdavis.edu (James Bremer), matyh@nus.edu.sg (Haizhao Yang)

axis. In [8], a precomputed expansion of the phase function for the associated Legendre differential equation is used to evaluate the associated Legendre functions P_ν^μ and Q_ν^μ with $0 \leq \nu \leq 10^7$ and $0 \leq \mu \leq \nu$ for arguments in the interval $(-1, 1)$. The method described here for the evaluation of the Jacobi polynomials differs from those of [5] and [8] in that we do not use a precomputed expansion of the phase function, but rather construct such an expansion on-the-fly for specified values of the parameters α and β in Jacobi's differential equation. The running time of this procedure is acceptable because we use an numerical algorithm for computing the phase function which is specialized to the case of Jacobi polynomials. The precomputations of [5] and [8] make use of the more general, but slower, procedure of [6]. A variant of the algorithm described here for the calculation of zeros of much more general classes of special functions was previously published in [7]; however, the version described here is specialized to the case of Gauss-Jacobi quadrature rules and is somewhat more efficient. The algorithm of [7], however, has the advantage that it is applicable when α and β are greater than $1/2$.

There is a large literature on the asymptotic behavior of Jacobi polynomials, and we cannot hope to list all of the relevant works here. A overview of results and many references can be found in Chapter 18 of [11]. There are, however, several recent extremely relevant works which specifically address the numerical evaluation of Jacobi polynomials and their zeros, and the application of the Jacobi transform. In [25], an algorithm for the numerical evaluation of Gauss-Jacobi quadrature rules is given. It is based on Newton's method, with the numerical evaluation of the Jacobi polynomials effected using two asymptotic expansions. We compare the algorithm of this paper for the construction of Gauss-Jacobi quadratures with that of [25] in Section 7.2. We also compare the method of this paper for evaluating Jacobi polynomials with that used in [25]. In [36], an algorithm for the application of the Chebyshev-Jacobi transform — that is, the mapping which takes the coefficients in a Chebyshev expansion of a function to the coefficients in a Jacobi expansion of the same function — is described. We compare our method for applying the Jacobi transform to the algorithm of [36] in Section 7.3. Finally, in the recent work [19], asymptotic expansions for the evaluation of Gauss-Jacobi nodes and weights are described. These expansions achieve double precision accuracy for a wide range of parameters, and apply in the case of values of α and β greater than $1/2$ as well as in the case of values of α and β less than $1/2$.

The remainder of this article is structured as follows. Section 2 gives a brief overview of the three algorithms described in this paper and introduces some notation and terminology. Section 3 summarizes a number of mathematical and numerical facts to be used in the rest of the paper. In Section 4, we detail the scheme we use to construct the phase and amplitude functions. Section 5 describes our method for the calculation of Gauss-Jacobi quadrature rules. In Section 6, we give our algorithm for applying the forward and inverse Jacobi transforms. In Section 7, we describe the results of numerical experiments conducted to assess the performance of the algorithms of this paper. Finally, we conclude in Section 8 with a few brief remarks regarding this article and a discussion of directions for further work.

2. Overview of the Algorithms

Our algorithms for evaluating Jacobi polynomials requires a precomputation be performed first. Given a pair of parameters α and β, both of which are in the interval $(-1/2, 1/2)$, and a maximum degree of interest $N_{\max} \geq 27$, we first numerically construct nonoscillatory phase and amplitude functions $\psi^{(\alpha,\beta)}(t, \nu)$ and $M^{(\alpha,\beta)}(t, \nu)$ which represent the Jacobi polynomials $P_\nu^{(\alpha,\beta)}(x)$ of degrees

between 27 and $N_{\max}$. We evaluate the polynomials of lower degrees using the well-known three-term recurrence relations; the lower bound of 27 was chosen in light of numerical experiments indicating it is close to optimal. We restrict our attention to values of α and β in the interval $(-1/2, 1/2)$ because, in this regime, the solutions of Jacobi's differential equation are purely oscillatory and its phase function is particularly simple. When α and β are larger, Jacobi's equation has turning points and the corresponding phase function becomes more complicated. The algorithms of this paper continue to work for values of α and β modestly larger than $1/2$, but they become less accurate as the parameters increase, and eventually fail. In a future work, we will discuss variants of the methods described here which apply in the case of larger values of the parameters (see the discussion in Section 8).

The relationship between $P_\nu^{(\alpha,\beta)}$ and the nonoscillatory phase and amplitude functions is

$$\tilde{P}_\nu^{(\alpha,\beta)}(t) = M^{(\alpha,\beta)}(t,\nu)\cos\left(\psi^{(\alpha,\beta)}(t,\nu)\right), \tag{1}$$

where $\tilde{P}_\nu^{(\alpha,\beta)}$ is defined via the formula

$$\tilde{P}_\nu^{(\alpha,\beta)}(t) = C_\nu^{a,b}\ P_\nu^{(\alpha,\beta)}\left(\cos(t)\right)\sin\left(\frac{t}{2}\right)^{\alpha+\frac{1}{2}}\cos\left(\frac{t}{2}\right)^{\beta+\frac{1}{2}} \tag{2}$$

with

$$C_\nu^{\alpha,\beta} = \sqrt{(2\nu+\alpha+\beta+1)\,\frac{\Gamma(1+\nu)\Gamma(1+\nu+\alpha+\beta)}{\Gamma(1+\nu+\alpha)\Gamma(1+\nu+\beta)}}. \tag{3}$$

The constant (3) ensures that the $L^2(0,\pi)$ norm of $\tilde{P}_\nu^{(\alpha,\beta)}$ is 1 when ν is an integer; indeed, the set $\left\{\tilde{P}_j^{(\alpha,\beta)}\right\}_{j=0}^{\infty}$ is an orthonormal basis for $L^2(0,\pi)$ (this follows, for instance, from standard results [41] in Sturm-Liouville theory). The change of variables $x = \cos(t)$ is introduced because it makes the singularities in the phase and amplitude functions for Jacobi's differential equation more tractable. We represent the functions $\psi^{(\alpha,\beta)}$ and $M^{(\alpha,\beta)}$ via their values at the nodes of a tensor product of piecewise Chebyshev grids. Owing to the smoothness of the phase and amplitude functions, these representations are quite efficient and can be constructed rapidly. Indeed, the asymptotic running time of our procedure for constructing the nonoscillatory phase and amplitude functions is $\mathcal{O}\left(\log^2(N_{\max})\right)$. In this work we always construct expansions which approximate the phase and amplitude functions with near machine precision accuracy. However, it should be noted that the runtime of the procedure for constructing these representations behaves as $\log\left(\epsilon^{-1}\right)$ where ϵ is the desired accuracy (this follows from the fact that the phase and amplitude functions are analytic in a neighborhood of the domain on which we consider them).

Once the nonoscillatory phase and amplitude functions have been constructed, the function $P_\nu^{(\alpha,\beta)}$ can be evaluated at any point $x \in (-1,1)$ and for any $27 \leq \nu \leq N_{\max}$ in time which is independent of $N_{\max}$, ν and x via (1). One downside of our approach is that the error which occurs when Formula (1) is used to evaluate a Jacobi polynomial numerically increases with the magnitude of the phase function $\psi^{(\alpha,\beta)}$ owing to the well-known difficulties in evaluating trigonometric functions of large arguments. This is not surprising, and the resulting errors are in line with the condition number of evaluation of the function $P_\nu^{(\alpha,\beta)}$. Moreover, this phenomenon is hardly limited to the approach of this paper and can be observed, for instance, when asymptotic expansions are used to evaluate Jacobi polynomials (see, for instance, [4] for an extensive discussion of this issue in the

case of asymptotic expansions for Legendre polynomials).

Aside from applying the Jacobi transform and evaluating Jacobi polynomials, nonoscillatory phase functions are also highly useful for calculating the roots of special functions. From (1), we see that the roots of $P_\nu^{(\alpha,\beta)}$ occur when

$$\psi^{(\alpha,\beta)}(t,\nu) = \frac{\pi}{2} + n\pi \tag{4}$$

with n an integer. In Section 5 we describe a method for rapidly computing Gauss-Jacobi quadrature rules which exploits this fact. The n-point Gauss-Jacobi quadrature rule corresponding to the parameters α and β is, of course, the unique quadrature rule of the form

$$\int_{-1}^{1} f(x)(1-x)^\alpha(1+x)^\beta \, dx \approx \sum_{j=1}^{n} f(x_j)v_j \tag{5}$$

that is exact when f is a polynomial of degree less than or equal to $2n-1$. Under the change of variables $x = \cos(t)$, (5) becomes

$$\int_0^\pi f(\cos(t)) \cos^{2\alpha+1}\left(\frac{t}{2}\right) \sin^{2\beta+1}\left(\frac{t}{2}\right) \, dt \approx \sum_{j=1}^{n} f(\cos(t_j)) \cos^{2\alpha+1}\left(\frac{t_j}{2}\right) \sin^{2\beta+1}\left(\frac{t_j}{2}\right) w_j. \tag{6}$$

We will refer to (6) as the trigonometric form of the Gauss-Jacobi rule.

We also describe a method for applying the Jacobi transform using the nonoscillatory phase and amplitude functions. For our purposes, the n^{th} order discrete forward Jacobi transform consists of calculating the vector of values

$$\begin{pmatrix} f(t_1)\sqrt{w_1} \\ f(t_2)\sqrt{w_2} \\ \vdots \\ f(t_n)\sqrt{w_n} \end{pmatrix} \tag{7}$$

given the vector

$$\begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{pmatrix} \tag{8}$$

of the coefficients in the expansion

$$f(t) = \sum_{j=0}^{n-1} \alpha_j \tilde{P}_j^{(\alpha,\beta)}(t); \tag{9}$$

here, $t_1, \ldots, t_n, w_1 \ldots, w_n$ are the nodes and weights of the n-point trigonometric Gauss-Jacobi quadrature rule corresponding to the parameters α and β. Since this quadrature rule integrates all products of the functions $\tilde{P}_\nu^{(\alpha,\beta)}$ of degrees between 0 and $n-1$, the weighting by square roots in (7) ensures that the $n \times n$ matrix $\mathcal{J}_n^{(\alpha,\beta)}$ which takes (8) to (7) is orthogonal. It follows from (1) that the (j,k) entry of $\mathcal{J}_n^{(\alpha,\beta)}$ is

$$M^{(\alpha,\beta)}(t_j, k-1) \cos\left(\psi^{(\alpha,\beta)}(t_j, k-1)\right) \sqrt{w_j}. \tag{10}$$

Our method for applying $\mathcal{J}_n^{(\alpha,\beta)}$, which is inspired by the algorithm of [9] for applying Fourier integral transforms and a special case of that in [40] for general oscillatory integral transforms, exploits the fact that the matrix whose (j,k) entry is

$$M^{(\alpha,\beta)}(t_j, k-1) \exp\left(i\left(\psi^{(\alpha,\beta)}(t_j, k-1) - (k-1)t_j\right)\right)\sqrt{w_j} \tag{11}$$

is low rank. Since $\mathcal{J}_n^{(\alpha,\beta)}$ is the real part of the Hadamard product of the matrix defined by (11) and the $n \times n$ nonuniform DFT matrix whose (j,k) entry is

$$\exp\left(i(k-1)t_j\right), \tag{12}$$

it can be applied through a small number of discrete nonuniform Fourier transforms. This can be done quickly via any number of fast algorithms for the nonuniform Fourier transform (examples of such algorithms include [22] and [34]). By "Hadamard product," we mean the operation of entrywise multiplication of two matrices. We will denote the Hadamard product of the matrices A and B via $A \circ B$ in this article.

We do not prove a rigorous bound on the rank of the matrix (11) here; instead, we offer the following heuristic argument. For points t bounded away from the endpoints of the interval $(0, \pi)$, we have the following asymptotic approximation of the nonoscillatory phase function:

$$\psi^{(\alpha,\beta)}(t,\nu) = \left(\nu + \frac{\alpha+\beta+1}{2}\right)t + c + \mathcal{O}\left(\frac{1}{\nu^2}\right) \quad \text{as} \quad \nu \to \infty \tag{13}$$

with c a constant in the interval $(0, 2\pi)$. We prove this in Section 3.5, below. From (13), we expect that the function

$$\psi^{(\alpha,\beta)}(t,\nu) - \nu t \tag{14}$$

is of relatively small magnitude, and hence that the rank of the matrix (11) is low. This is borne out by our numerical experiments (see Figure 5 in particular).

We choose to approximate $\psi^{(\alpha,\beta)}(t,\nu)$ via the linear function νt rather than with a more complicated function (e.g., a nonoscillatory combination of Bessel functions) because this approximation leads to a method for applying the Jacobi transform through repeated applications of the fast Fourier transform, if the non-uniform Fourier transform in the Hadamard product just introduced is carried out via the algorithm in [34]. It might be more efficient to apply the non-uniform FFT algorithm in [22], depending on different computation platforms and compilers; but we do not aim at optimizing our algorithm in this direction. Hence, our algorithm for applying the Jacobi transform requires r applications of the fast Fourier transform, where r is the numerical rank of a matrix which is strongly related to (11). This makes its asymptotic complexity $\mathcal{O}\left(rn\log(n)\right)$. Again, we do not prove a rigorous bound on r here. However, in the experiments of Section 7.3 we compare our algorithm for applying the Jacobi transform with Slevinsky's algorithm [36] for applying the Chebyshev-Jacobi transform. The latter's running time is $\mathcal{O}\left(n\log^2(n)/\log(\log(n))\right)$, and the behavior of our algorithm is similar. This seems to suggest that

$$r = \mathcal{O}\left(\frac{\log(n)}{\log(\log(n))}\right). \tag{15}$$

However, we make the somewhat more conservative conjecture

$$r = \mathcal{O}\left(\log(n)\right) \tag{16}$$

because is it very difficult to distinguish between the two cases via numerical experiments owing to

the extremely slow growth of $\log(\log(n))$ and (16) seems more plausible.

Before our algorithm for applying the Jacobi transform can be used, a precomputation in addition to the construction of the nonoscillatory phase and amplitude functions must be performed. Fortunately, the running time of this precomputation procedure is quite small. Indeed, for most values of n, it requires less time than a single application of the Jacobi-Chebyshev transform via [36]. Once the precomputation phase has been dispensed with, our algorithm for the application of the Jacobi transform is roughly ten times faster than the algorithm of [36].

3. Preliminaries

3.1. Jacobi's differential equation and the solution of the second kind

The Jacobi function of the first kind is given by the formula

$$P_\nu^{(\alpha,\beta)}(x) = \frac{\Gamma(\nu+\alpha+1)}{\Gamma(\nu+1)\Gamma(\alpha+1)} {}_2F_1\left(-\nu, \nu+\alpha+\beta+1; \alpha+1; \frac{1-x}{2}\right), \tag{17}$$

where ${}_2F_1(a,b;c;z)$ denotes Gauss' hypergeometric function (see, for instance, Chapter 2 of [15]). We also define the Jacobi function of the second kind via

$$\begin{aligned} Q_\nu^{(\alpha,\beta)} &= \cot(\alpha\pi) P_\nu^{(\alpha,\beta)}(x) - \\ &\frac{2^{\alpha+\beta}\Gamma(\nu+\beta+1)\Gamma(\alpha)}{\pi\Gamma(\nu+\beta+\alpha+1)} (1-x)^{-\alpha}(1+x)^{-\beta} {}_2F_1\left(\nu+1, -\nu-\alpha-\beta; 1-\alpha; \frac{1-x}{2}\right). \end{aligned} \tag{18}$$

Formula (17) is valid for all values of the parameters α, β, and ν and all values of the argument x of interest to us here. Formula (18), on the other hand, is invalid when $\alpha = 0$ (and more generally when α is an integer). However, the limit as $\alpha \to 0$ of $Q_\nu^{(\alpha,\beta)}$ exists and various series expansions for it can be derived rather easily (using, for instance, the well-known results regarding hypergeometric series which appear in Chapter 2 of [15]). Accordingly, we view (18) as defining $Q_\nu^{(\alpha,\beta)}$ for all α, β in our region of interest $(-1/2, 1/2)$.

Both $P_\nu^{(\alpha,\beta)}$ and $Q_\nu^{(\alpha,\beta)}$ are solutions of Jacobi's differential equation

$$(1-x^2)\psi''(x) + (\beta - \alpha - (\alpha+\beta+2)x)\psi'(x) + \nu(\nu+\alpha+\beta+1)\psi(x) = 0 \tag{19}$$

(see, for instance, Section 10.8 of [16]). We note that our choice of $Q_\nu^{(\alpha,\beta)}$ is nonstandard and differs from the convention used in [37] and [16]. However, it is natural in light of Formulas (23) and (28), (29) and (32), and especially (46), all of which appear below.

It can be easily verified that the functions $\tilde{P}_\nu^{(\alpha,\beta)}$ and $\tilde{Q}_\nu^{(\alpha,\beta)}$ defined via (2) and the formula

$$\tilde{Q}_\nu^{(\alpha,\beta)}(t) = C_\nu^{(\alpha,\beta)}\ Q_\nu^{(\alpha,\beta)}\left(\cos(t)\right) \sin\left(\frac{t}{2}\right)^{a+\frac{1}{2}} \cos\left(\frac{t}{2}\right)^{b+\frac{1}{2}} \tag{20}$$

satisfy the second order differential equation

$$y''(t) + q_\nu^{(\alpha,\beta)}(t) y(t) = 0, \tag{21}$$

where

$$q_\nu^{(\alpha,\beta)}(t) = \left(\nu + \frac{\alpha+\beta+1}{2}\right)^2 + \frac{\frac{1}{4}-\alpha^2}{4\sin\left(\frac{t}{2}\right)^2} + + \frac{\frac{1}{4}-\beta^2}{4\cos\left(\frac{t}{2}\right)^2}. \tag{22}$$

We refer to Equation (21) as the modified Jacobi differential equation.

3.2. The asymptotic approximations of Baratella and Gatteschi

In [2], it is shown that there exist sequences of real-valued functions $\{A_j\}$ and $\{B_j\}$ such that for each nonnegative integer m,

$$\tilde{P}_\nu^{(\alpha,\beta)}(t) = C_\nu^{(\alpha,\beta)} \frac{p^{-\alpha}}{\sqrt{2}} \frac{\Gamma(\nu+\alpha+1)}{\Gamma(\nu+1)} \left(\sum_{j=0}^{m} \frac{A_j(t)}{p^{2j}} t^{\frac{1}{2}} J_\alpha(pt) + \sum_{j=0}^{m-1} \frac{B_j(t)}{p^{2j+1}} t^{\frac{3}{2}} J_{\alpha+1}(pt) + \epsilon_m(t,p) \right), \quad (23)$$

where J_μ denotes the Bessel function of the first kind of order μ, $C_\nu^{(\alpha,\beta)}$ is defined by (3), $p = \nu + (a+b+1)/2$, and

$$\epsilon_m(t,p) = \begin{cases} t^{\alpha+\frac{5}{2}} \mathcal{O}\left(p^{-2m+\alpha}\right), & 0 < t \le \frac{c}{p} \\ t\mathcal{O}\left(p^{-2m-\frac{3}{2}}\right), & \frac{c}{p} \le t \le \pi - \delta \end{cases} \quad (24)$$

with c and δ fixed constants. The first few coefficients in (23) are given by $A_0(t) = 1$,

$$B_0(t) = \frac{1}{4t} g(t), \quad (25)$$

and

$$A_1(t) = \frac{1}{8} g'(t) - \frac{1+2\alpha}{8t} g(t) - \frac{1}{32} (g(t))^2 + \frac{\alpha}{24} \left(3\beta^2 + \alpha^2 - 1\right), \quad (26)$$

where

$$g(t) = \left(\frac{1}{4} - \alpha^2\right) \left(\cot\left(\frac{t}{2}\right) - \frac{2}{t}\right) - \left(\frac{1}{4} - \beta^2\right) \tan\left(\frac{t}{2}\right). \quad (27)$$

A discussion of the higher order coefficients, including a means to derive power series expansions of these coefficients which hold for small values of t, is described in [19]. The analogous expansion of the function of the second kind is

$$\tilde{Q}_\nu^{(\alpha,\beta)}(t) \approx C_\nu^{(\alpha,\beta)} \frac{p^{-\alpha}}{\sqrt{2}} \frac{\Gamma(\nu+\alpha+1)}{\Gamma(\nu+1)} \left(\sum_{s=0}^{m} \frac{A_s(t)}{p^{2s}} t^{\frac{1}{2}} Y_\alpha(pt) + \sum_{s=0}^{m-1} \frac{B_s(t)}{p^{2s+1}} t^{\frac{3}{2}} Y_{\alpha+1}(pt) + \epsilon'_m(t,p) \right). \quad (28)$$

An alternate asymptotic expansion of $P_\nu^{(\alpha,\beta)}$ whose form is similar to (23) appears in [17], and several more general Liouville-Green type expansions for Jacobi polynomials can be found in [13]. The expansions of [13] involve a fairly complicated change of variables which complicates their use in numerical algorithms.

3.3. Hahn's trigonometric expansions

The asymptotic formula

$$\tilde{P}_\nu^{(\alpha,\beta)}(t) = C_\nu^{(\alpha,\beta)} \frac{2^{2p}}{\pi} B(\nu+\alpha+1, \nu+\beta+1) \sum_{m=0}^{M-1} \frac{f_m(t)}{2^m (2p+1)_m} + R_{\nu,m}^{(\alpha,\beta)}(t) \quad (29)$$

appears in a slightly different form in [23]. Here, $(x)_m$ is the Pochhammer symbol, B is the beta function, p is once again equal to $\nu + (\alpha+\beta+1)/2$, and

$$f_m(t) = \sum_{l=0}^{m} \frac{\left(\frac{1}{2}+\alpha\right)_l \left(\frac{1}{2}-\alpha\right)_l \left(\frac{1}{2}+\beta\right)_{m-l} \left(\frac{1}{2}-\beta\right)_{m-l}}{\Gamma(l+1)\Gamma(m-l+1)} \frac{\cos\left(\frac{1}{2}(2p+m)t - \frac{1}{2}\left(\alpha+l+\frac{1}{2}\right)\pi\right)}{\sin^l\left(\frac{t}{2}\right) \cos^{m-l}\left(\frac{t}{2}\right)}. \quad (30)$$

The remainder term $R_{\nu,m}^{(\alpha,\beta)}$ is bounded by twice the magnitude of the first neglected term when α and β are in the interval $(-1/2, 1/2)$. The analogous approximation of the Jacobi function of the second kind is

$$\tilde{Q}_{\nu}^{(\alpha,\beta)}(t) \approx C_{\nu}^{(\alpha,\beta)} \frac{2^{2p}}{\pi} B(\nu+\alpha+1, \nu+\beta+1) \sum_{m=0}^{M-1} \frac{g_m(t)}{2^m \left(2p+1\right)_m}, \tag{31}$$

where

$$g_m(t) = \sum_{l=0}^{m} \frac{\left(\frac{1}{2}+\alpha\right)_l \left(\frac{1}{2}-\alpha\right)_l \left(\frac{1}{2}+\beta\right)_{m-l} \left(\frac{1}{2}-\beta\right)_{m-l}}{\Gamma(l+1)\Gamma(m-l+1)} \frac{\sin\left(\frac{1}{2}(2p+m)t - \frac{1}{2}\left(\alpha+l+\frac{1}{2}\right)\pi\right)}{\sin^l\left(\frac{t}{2}\right)\cos^{m-l}\left(\frac{t}{2}\right)}. \tag{32}$$

Remark 1. *Formula (32) does not appear in [23], nor are the authors aware of a derivation of it in the literature. However, it is easy to guess and the authors have verified it using extensive numerical experiments.*

3.4. Some elementary facts regarding phase functions for second order differential equations

We say that ψ is a phase function for the second order differential equation

$$y''(t) + q(t)y(t) = 0 \quad \text{for all} \quad \sigma_1 \le t \le \sigma_2 \tag{33}$$

provided the functions

$$\frac{\cos\left(\psi(t)\right)}{\sqrt{|\psi'(t)|}} \tag{34}$$

and

$$\frac{\sin\left(\psi(t)\right)}{\sqrt{|\psi'(t)|}} \tag{35}$$

form a basis in its space of solutions. By repeatedly differentiating (34) and (35), it can be shown that ψ is a phase function for (33) if and only if its derivative satisfies the nonlinear ordinary differential equation

$$q(t) - (\psi'(t))^2 - \frac{1}{2}\left(\frac{\psi'''(t)}{\psi'(t)}\right) + \frac{3}{4}\left(\frac{\psi''(t)}{\psi'(t)}\right)^2 = 0 \quad \text{for all } \sigma_1 \le t \le \sigma_2. \tag{36}$$

A pair u, v of real-valued, linearly independent solutions of (33) determine a phase function ψ up to an integer multiple of 2π. Indeed, it can be easily seen that the function

$$\psi'(t) = \frac{W}{(u(t))^2 + (v(t))^2}, \tag{37}$$

where W is the necessarily constant Wronskian of the pair $\{u, v\}$, satisfies (36). As a consequence, if we define

$$\psi(t) = C + \int_{\sigma_1}^{t} \frac{W}{(u(s))^2 + (v(s))^2}\, ds \tag{38}$$

with C an appropriately chosen constant, then

$$u(t) = \sqrt{W}\, \frac{\cos\left(\psi(t)\right)}{\sqrt{|\psi'(t)|}} \tag{39}$$

and

$$v(t) = \sqrt{W}\ \frac{\sin\left(\psi(t)\right)}{\sqrt{|\psi'(t)|}}. \tag{40}$$

The requirement that (39) and (40) hold clearly determines the value of $\mathrm{mod}(C, 2\pi)$.

If ψ is a phase function for (33), then we refer to the function

$$M(t) = \sqrt{\frac{W}{|\psi'(t)|}} \tag{41}$$

as the corresponding amplitude function. Though a straightforward computation it can be verified that the the square $N(t) = \left(M(t)\right)^2$ of the amplitude function satisfies the third order linear ordinary differential equation

$$N'''(t) + 4q(t)N'(t) + 2q'(t)N(t) = 0 \ \text{ for all } \ \sigma_1 < t < \sigma_2. \tag{42}$$

3.5. The nonoscillatory phase function for Jacobi's differential equation

We will denote by $\psi^{(\alpha,\beta)}(t,\nu)$ a phase function for the modified Jacobi differential equation (21) which, for each ν, gives rise to the pair of solutions $\tilde{P}_\nu^{(\alpha,\beta)}$ and $\tilde{Q}_\nu^{(\alpha,\beta)}$. We let $M^{(\alpha,\beta)}(t,\nu)$ denote the corresponding amplitude function, so that

$$\tilde{P}_\nu^{(\alpha,\beta)}(t) = M^{(\alpha,\beta)}(t,\nu)\cos\left(\psi^{(\alpha,\beta)}(t,\nu)\right) \tag{43}$$

and

$$\tilde{Q}_\nu^{(\alpha,\beta)}(t) = M^{(\alpha,\beta)}(t,\nu)\sin\left(\psi^{(\alpha,\beta)}(t,\nu)\right). \tag{44}$$

We use the notations $\psi^{(\alpha,\beta)}(t,\nu)$ and $M^{(\alpha,\beta)}(t,\nu)$ rather than $\psi_\nu^{(\alpha,\beta)}(t)$ and $M_\nu^{(\alpha,\beta)}(t)$, which might seem more natural, to emphasize that the representations of the phase and amplitude functions we construct in this article allow for their evaluation for a range of values of ν and t, but only for particular fixed values of α and β.

Obviously,

$$\left(M^{(\alpha,\beta)}(t,\nu)\right)^2 = \left(P_\nu^{(\alpha,\beta)}(t)\right)^2 + \left(Q_\nu^{(\alpha,\beta)}(t)\right)^2. \tag{45}$$

By replacing the Jacobi functions in (45) with the first terms in the expansions (23) and (28), we see that for all t in an interval of the form $(\delta, \pi - \delta)$ with δ a small positive constant and all $\alpha, \beta \in (-1/2, 1/2)$,

$$\left(M^{(\alpha,\beta)}(t,\nu)\right)^2 \sim \left(C_\nu^{(\alpha,\beta)}\frac{p^{-\alpha}}{\sqrt{2}}\frac{\Gamma(\nu+\alpha+1)}{\Gamma(\nu+1)}\right)^2 t\left(\left(J_\alpha(pt)\right)^2 + \left(Y_\alpha(pt)\right)^2\right) + \mathcal{O}\left(\frac{1}{\nu^2}\right) \ \text{ as } \ \nu \to \infty. \tag{46}$$

It is well known that the function

$$\left(J_\alpha(t)\right)^2 + \left(Y_\alpha(t)\right)^2 \tag{47}$$

is nonoscillatory. Indeed, it is completely monotone on the interval $(0, \infty)$, as can be shown using Nicholson's formula

$$\left(J_\alpha(t)\right)^2 + \left(Y_\alpha(t)\right)^2 = \frac{8}{\pi^2}\int_0^\infty K_0(2t\sinh(s))\cosh(2\alpha s)\ ds \ \text{ for all } \ t > 0. \tag{48}$$

A derivation of (48) can be found in Section 13.73 of [39]; that (47) is completely monotone follows

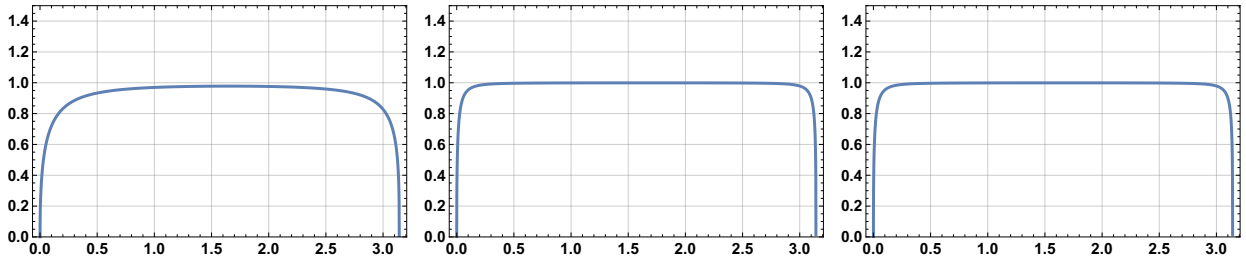

Figure 1: On the left is a plot of the function $\frac{\pi}{2}\left(M_\nu^{(\alpha,\beta)}\right)^2$ when $\alpha=\frac{1}{4}$, $\beta=\frac{1}{3}$ and $\nu=1$. In the middle is a plot of the same function when $\alpha=\frac{1}{4}$, $\beta=\frac{1}{3}$ and $\nu=10$. On the right, is a plot of it when $\alpha=\frac{1}{4}$, $\beta=\frac{1}{3}$ and $\nu=100$.

easily from (48) (see, for instance, [32]). When higher order approximations of $\tilde{P}_\nu^{(\alpha,\beta)}$ and $\tilde{Q}_\nu^{(\alpha,\beta)}$ are inserted into (45), the resulting terms are all nonoscillatory combinations of Bessel functions. From this it is clear that $M_\nu^{(\alpha,\beta)}$ is nonoscillatory, and Formula (46) then implies that $\psi_\nu^{(\alpha,\beta)}$ is also a nonoscillatory function.

It is well-known that

$$(J_\alpha(t))^2+(Y_\alpha(t))^2\sim\frac{2}{\pi}\left(\frac{1}{t}+\frac{4\alpha^2-1}{8t^3}+\frac{3(9-40\alpha^2+16\alpha^4)}{128t^5}+\cdots\right)\quad\text{as}\quad t\to\infty; \tag{49}$$

see, for instance, Section 13.75 of [39]. From this, (46) and (3), we easily conclude that for t in $(\delta,\pi-\delta)$ and $(\alpha,\beta)\in(-1/2,1/2)$,

$$\left(M^{(\alpha,\beta)}(t,\nu)\right)^2\sim\frac{2p^{-2\alpha}}{\pi}\frac{\Gamma(1+\nu+\alpha)\Gamma(1+\nu+\alpha+\beta)}{\Gamma(1+\nu)\Gamma(1+\nu+\beta)}+\mathcal{O}\left(\frac{1}{\nu^2}\right) \tag{50}$$

as $\nu\to\infty$. By inserting the asymptotic approximation

$$\frac{\Gamma(1+\nu+\alpha)\Gamma(1+\nu+\alpha+\beta)}{\Gamma(1+\nu)\Gamma(1+\nu+\beta)}\sim\left(\nu+\frac{\alpha+\beta+1}{2}\right)=p^{2\alpha}\quad\text{as}\quad\nu\to\infty \tag{51}$$

into (50), we obtain

$$\left(M^{(\alpha,\beta)}(t,\nu)\right)^2\sim\frac{2}{\pi}+\mathcal{O}\left(\frac{1}{\nu^2}\right)\quad\text{as}\quad\nu\to\infty. \tag{52}$$

Once again, (52) only holds for t in the interior of the interval $(0,\pi)$ and $(\alpha,\beta)\in(-1/2,1/2)$. Figure 1 contains plots of the function $\frac{\pi}{2}\left(M_\nu^{(\alpha,\beta)}\right)^2$ for three sets of the parameters α, β and ν. A straightforward, but somewhat tedious, computation shows that the Wronskian of the pair $\tilde{P}_\nu^{(\alpha,\beta)}$, $\tilde{Q}_\nu^{(\alpha,\beta)}$ is

$$W_\nu^{(\alpha,\beta)}=\frac{2p}{\pi}. \tag{53}$$

From this, (52) and (41), we conclude that for $t\in(\delta,\pi-\delta)$ and $\alpha,\beta\in(-1/2,1/2)$,

$$\psi^{(\alpha,\beta)}(t,\nu)\sim pt+c+\mathcal{O}\left(\frac{1}{\nu^2}\right)\quad\text{as}\quad\nu\to\infty \tag{54}$$

with c a constant. Without loss of generality, we can assume c is in the interval $(0,2\pi)$. This is Formula (13).

3.6. Interpolative decompositions

We say a factorization

$$A = BR \tag{55}$$

of an $n \times m$ matrix A is an interpolative decomposition of rank k if B is an $n \times k$ matrix, R is an $k \times m$ matrix and there exists an $m \times m$ permutation matrix such that the first k columns of RP comprise the $k \times k$ identity matrix. If (55) does not hold exactly, but instead

$$\|A - BR\|_2 \leq \epsilon \|A\|_2, \tag{56}$$

then we say that (55) is an ϵ-accurate interpolative decomposition. In our implementation of the algorithms of this paper, a variant of the algorithm of [10] is used in order to form interpolative decompositions.

3.7. Piecewise Chebyshev interpolation

The k-point Chebyshev extrema grid on the interval $[\sigma_1, \sigma_2]$ is the set of points

$$\left\{ \frac{\sigma_2 - \sigma_1}{2} \cos\left(\frac{\pi}{k-1}(k-i) \right) + \frac{\sigma_2 + \sigma_1}{2} : i = 1, \ldots, k \right\}. \tag{57}$$

As is well known, given the values of a polynomial p of degree $n-1$ at these nodes, its value at any point t in the interval $[\sigma_1, \sigma_2]$ can be evaluated in a numerically stable fashion using the barcyentric Chebyshev interpolation formula

$$p(t) = \sum_{j=1}^{k} \frac{w_j}{t - x_j} p(x_j) \Bigg/ \sum_{j=1}^{k} \frac{w_j}{t - x_j}, \tag{58}$$

where $x_1, \ldots, x_k$ are the nodes of the k-point Chebyshev grid on $[\sigma_1, \sigma_2]$ and

$$w_j = \begin{cases} (-1)^j & 1 < j < k \\ \frac{1}{2}(-1)^j & \text{otherwise.} \end{cases} \tag{59}$$

See, for instance, [38], for a detailed discussion of Chebyshev interpolation and the numerical properties of Formula (58). Since we never use the Chebyshev roots as interpolation nodes in this article, we will simply refer to (57) as the k-point Chebyshev grid.

We say that a function p is a piecewise polynomial of degree $k-1$ on the collection of intervals

$$[\sigma_1, \sigma_2], [\sigma_2, \sigma_3], \ldots, [\sigma_{m-1}, \sigma_m], \tag{60}$$

where

$$\sigma_1 < \sigma_2 < \ldots < \sigma_m, \tag{61}$$

provided its restriction to each $[\sigma_j, \sigma_{j+1}]$ is a polynomial of degree less than or equal to $k-1$. We refer to the collection of points

$$\left\{ \frac{\sigma_{j+1} - \sigma_j}{2} \cos\left(\frac{\pi}{k-1}(k-i) \right) + \frac{\sigma_{j+1} + \sigma_j}{2} : j = 1, \ldots, m-1,\ i = 1, \ldots, k \right\} \tag{62}$$

as the k-point piecewise Chebyshev grid on the intervals (60). Because the last point of each interval save $[\sigma_{m-1}, \sigma_m]$ coincides with the first point on the succeeding interval, there are $(k-1)(m-1)+1$ such points. Given the values of a piecewise polynomial p of degree $k-1$ at these nodes, its value at any point t on the interval $[\sigma_1, \sigma_m]$ can be calculated by identifying the interval containing t and applying an appropriate modification of Formula (58). Owing to the assumption (61), the

intervals (60) can only intersect at their endpoints. Since these endpoints are nodes on the piecewise Chebyshev grid representing the interpolant, the value of the interpolant at these endpoints is known and no ambiguity arises from the fact that the intervals (60) intersect. Such a procedure requires at most $\mathcal{O}(m+k)$ operations, typically requires $\mathcal{O}(\log(m)+k)$ when a binary search is used to identify the interval containing t, and requires only $\mathcal{O}(k)$ operations in the fairly common case that the collection (60) is of a form which allows the correct interval to be identified immediately. The last case applies when σ_j is given by an easily invertible function of j; for instance, if

$$\sigma_j = \frac{j-1}{m-1},$$

then the index of an interval containing $x \in [0,1]$ is given by

$$j = 1 + \lfloor (m-1)x \rfloor .$$

We refer to the device of representing smooth functions via their values at the nodes (62) as the piecewise Chebyshev discretization scheme of order k on (60), or sometimes just the piecewise Chebyshev discretization scheme on (60) when the value of k is readily apparent. Given such a scheme and an ordered collection of points

$$\sigma_1 \leq t_1 \leq t_2 \leq \ldots < t_N \leq \sigma_m, \tag{63}$$

we refer to the $N \times ((m-1)(k-1)+1)$ matrix which maps the vector of values

$$\begin{pmatrix} p(x_{1,1}) \\ p(x_{2,1}) \\ \vdots \\ p(x_{k,1}) \\ p(x_{2,2}) \\ \vdots \\ p(x_{k,2}) \\ \vdots \\ p(x_{k,m-1}) \\ p(x_{2,m}) \\ \vdots \\ p(x_{n,m}) \end{pmatrix}, \tag{64}$$

where p is any piecewise polynomial of degree $k-1$ on the interval (60) and

$$x_{i,j} = \frac{\sigma_{j+1} - \sigma_j}{2} \cos\left(\frac{\pi}{k-1}(k-i)\right) + \frac{\sigma_{j+1} + \sigma_j}{2}, \tag{65}$$

to the vector of values

$$\begin{pmatrix} p(t_1) \\ p(t_2) \\ \vdots \\ p(t_N) \end{pmatrix} \tag{66}$$

as the matrix interpolating piecewise polynomials of degree $k-1$ on (60) to the nodes (63). This

matrix is block diagonal; in particular, it is of the form

$$\begin{pmatrix} B_1 & 0 & 0 & 0 \\ 0 & B_2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & B_{m-1} \end{pmatrix} \tag{67}$$

with B_j corresponding to the interval (σ_j, σ_{j+1}). The dimensions of B_j are $n_j \times k$, where n_j is the number of points from (63) which lie in $[\sigma_j, \sigma_{j+1}]$. The entries of each block can be calculated from (58), with the consequence that this matrix can be applied in $\mathcal{O}(nk)$ operations without explicitly forming it.

3.8. Bivariate Chebyshev interpolation

Suppose that

$$x_1, \ldots, x_{M_1}, \tag{68}$$

where $M_1 = (m-1)(k_1 - 1) + 1$, is the k_1-point piecewise Chebyshev grid on the collection of intervals

$$(\sigma_1, \sigma_2), (\sigma_2, \sigma_3), \ldots, (\sigma_{m_1 - 1}, \sigma_{m_1}), \tag{69}$$

and that

$$y_1, \ldots, y_{M_2}, \tag{70}$$

where $M_2 = (m_2 - 1)(k_2 - 1) + 1$, is the k_2-point piecewise Chebyshev grid on the collection of intervals

$$(\zeta_1, \zeta_2), (\zeta_2, \zeta_3), \ldots, (\zeta_{m_2 - 1}, \zeta_{m_2}). \tag{71}$$

Then we refer to the device of representing smooth bivariate functions $f(x, y)$ defined on the rectangle $(\sigma_1, \sigma_{m_1}) \times (\zeta_1, \zeta_{m_2})$ via their values at the tensor product

$$\{(x_i, y_j) : i = 1, \ldots, M_1,\ j = 1, \ldots, M_2\} \tag{72}$$

of the piecewise Chebyshev grids (68) and (70) as a bivariate Chebyshev discretization scheme. Given the values of f at the discretization nodes (72), its value at any point (x, y) in the rectangle $(\sigma_1, \sigma_{m_1}) \times (\zeta_1, \zeta_{m_2})$ can be approximated through repeated applications of the barcyentric Chebyshev interpolation formula. The cost of such a procedure is $\mathcal{O}(k_1 k_2)$, once the rectangle $(\sigma_i, \sigma_{i+1}) \times (\zeta_j, \zeta_{j+1})$ containing (x, y) has been located. The cost of locating the correct rectangle is $\mathcal{O}(m_1 + m_2)$ in the worst case, $\mathcal{O}(\log(m)_1 + \log(m_2))$ in typical cases, and $\mathcal{O}(1)$ when the particular forms of the collections (69) and (71) allow for the immediate identification of the correct rectangle.

4. Computation of the Phase and Amplitude Functions

Here, we describe a method for calculating representations of the nonoscillatory phase and amplitude functions $\psi^{(\alpha,\beta)}(t, \nu)$ and $M^{(\alpha,\beta)}(t, \nu)$. This procedure takes as input the parameters α and β as well as an integer $N_{\max} \geq 27$. The representations of $M^{(\alpha,\beta)}$ and $\psi^{(\alpha,\beta)}$ allow for their evaluation for all values of ν such that

$$27 \leq \nu \leq N_{\max} \tag{73}$$

and all arguments t such that

$$\frac{1}{N_{\max}} \leq t \leq \pi - \frac{1}{N_{\max}}. \tag{74}$$

The smallest and largest zeros of $\tilde{P}_\nu^{(\alpha,\beta)}$ on the interval $(0,\pi)$ are greater than $\frac{1}{\nu}$ and less than $\pi - \frac{1}{\nu}$, respectively (see, for instance, Chapter 18 of [11] and its references). In particular, this means that the range (73) suffices to evaluate $\tilde{P}_\nu^{(\alpha,\beta)}$ at the values of t corresponding to the nodes of the $N_{\max}$-point Gauss-Jacobi quadrature rule. If necessary, this range could be expanded or various asymptotic expansion could be used to evaluate $\tilde{P}^{(\alpha,\beta)}(t,\nu)$ for values of t outside of it.

The phase and amplitude functions are represented via their values at the nodes of a tensor product of piecewise Chebyshev grids. To define them, we first let m_ν be the least integer such that

$$3^{m_\nu} \geq N_{\max} \tag{75}$$

and let m_t be equal to twice the least integer l such that

$$\frac{\pi}{2} 2^{-l+1} \leq \frac{1}{N_{\max}}. \tag{76}$$

Next, we define b_j for $j = 1, \ldots, m_\nu$ by

$$b_j = \max\left\{3^{j+2}, N_{\max}\right\}, \tag{77}$$

a_i for $i = 1, \ldots, m_t/2$ via

$$a_i = \frac{\pi}{2} 2^{i - m_t/2} \tag{78}$$

and a_i for $i = m_t/2 + 1, \ldots, m_t$ via

$$b_i = \pi - \frac{\pi}{2} 2^{m_t/2 + 1 - i}. \tag{79}$$

We note that the points $\{b_i\}$ cluster near 0 and π, where the phase and amplitude functions are singular. Now we let

$$\tau_1, \ldots, \tau_{M_t} \tag{80}$$

denote 16-point piecewise Chebyshev gird on the intervals

$$[a_1, a_2], [a_2, a_3], \ldots, [a_{m_t - 1}, a_{m_t}]. \tag{81}$$

There are

$$M_t = 15(m_t - 1) + 1 \tag{82}$$

such points. In a similar fashion, we let

$$\gamma_1, \ldots, \gamma_{M_\nu} \tag{83}$$

denote the nodes of the 24-point piecewise Chebyshev grid on the intervals

$$[b_1, b_2], [b_2, b_3], \ldots, [b_{m_\nu - 1}, b_{m_\nu}]. \tag{84}$$

There are

$$M_t = 23(m_\nu - 1) + 1 \tag{85}$$

such points.

For each node γ in the set (83), we solve the ordinary differential equation (42) with q taken to be the coefficient (22) for Jacobi's modified differential equation using a variant of the integral

equation method of [21]. We note, though, that any standard method capable of coping with stiff problems should suffice. We determine a unique solution by specifying the values of

$$\left(M^{(\alpha,\beta)}(t,\gamma)\right)^2 \tag{86}$$

and its first two derivatives at a point on the interval (74). These values are calculated using an asymptotic expansion for (86) derived from (23) and (28) when γ is large, and using series expansions for $P_\gamma^{(\alpha,\beta)}$ and $Q_\gamma^{(\alpha,\beta)}$ when γ is small. We refer the interested reader to our code (which we have made publicly available) for details. Solving (42) gives us the values of the function for each t in the set (80). To obtain the values of $\psi^{(\alpha,\beta)}(t,\gamma)$ for each t in the set (80) , we first calculate the values of

$$\frac{d}{dt}\psi^{(\alpha,\beta)}(t,\gamma) \tag{87}$$

via (41). Next, we obtain the values of the function $\tilde{\psi}$ defined via

$$\tilde{\psi}(t) = \int_{\alpha_1}^{t} \frac{d}{ds}\psi^{(\alpha,\beta)}(s,\gamma)\ ds \tag{88}$$

at the nodes (80) via spectral integration. There is an as yet unknown constant connecting $\tilde{\psi}$ with $\psi^{(\alpha,\beta)}(t,\gamma)$; that is,

$$\psi^{(\alpha,\beta)}(t,\gamma) = \tilde{\psi}(t) + C. \tag{89}$$

To evaluate C, we first use a combination of asymptotic and series expansions to calculate $\tilde{P}_\nu^{(\alpha,\beta)}$ at the point a_1. Since $\widetilde{\psi}(a_1) = 0$, it follows that

$$\tilde{P}_\gamma^{(\alpha,\beta)}(a_1) = M^{(\alpha,\beta)}(a_1,\gamma)\cos(C), \tag{90}$$

and C can be calculated in the obvious fashion. Again, we refer the interested reader to our source code for the details of the asymptotic expansions we use to evaluate $\tilde{P}_\nu^{(\alpha,\beta)}(a_1)$.

The ordinary differential equation (42) is solved for $\mathcal{O}\left(\log\left(N_{\max}\right)\right)$ values of ν, and the phase and amplitude functions are calculated at $\mathcal{O}\left(\log\left(N_{\max}\right)\right)$ values of t each time it is solved. This makes the total running time of the procedure just described

$$\mathcal{O}\left(\log^2\left(N_{\max}\right)\right). \tag{91}$$

Once the values of $\psi_\nu^{(\alpha,\beta)}$ and $M_\nu^{(\alpha,\beta)}$ have been calculated at the tensor product of the piecewise Chebyshev grids (80) and (83), they can be evaluated for any t and ν via repeated application of the barycentric Chebyshev interpolation formula in a number of operations which is independent of ν and t. The rectangle in the discretization scheme which contains the point (t,ν) can be determined in $\mathcal{O}\left(1\right)$ operations owing to the special form of (81) and (84). The value of $\tilde{P}_\nu^{(\alpha,\beta)}$ can then be calculated via Formula (43), also in a number of operations which is independent of ν and t.

Remark 2. *Our choice of the particular bivariate piecewise Chebyshev discretization scheme used to represent the functions $\psi^{(\alpha,\beta)}$ and $M^{(\alpha,\beta)}$ was informed by extensive numerical experiments. However, this does not preclude the possibility that there are other similar schemes which provide better levels of accuracy and efficiency. Improved mechanisms for the representation of phase functions are currently being investigated by the authors.*

5. Computation of Gauss-Jacobi Quadrature Rules

In this section, we describe our algorithm for the calculation the Gauss-Jacobi quadrature rules. It proceeds by first calculating the trigonometric form (6), and then obtaining (5) through a change of variables. It takes as input the length n of the desired quadrature rule as well as parameters α and β in the interval $(-1/2, 1/2)$ and returns the nodes $x_1, \ldots, x_n$ and weights $v_1, \ldots, v_n$ of (5) and the nodes $t_1, \ldots, t_n$ and weights $w_1, \ldots, w_n$ of (6).

For these computations, we do not rely on the expansions of $\psi^{(\alpha,\beta)}$ and $M^{(\alpha,\beta)}$ produced using the procedure of Section 4. Instead, as a first step, we calculate the values of $\psi^{(\alpha,\beta)}(t,n)$ and $M^{(\alpha,\beta)}(t,n)$ for each t in the set (80) by solving the ordinary differential equation (42). The procedure is the same as that used in the algorithm of the preceding section. Because the degree $\nu = n$ of the phase function used by the algorithm of this section is fixed, we break from our notational convention and use $\psi_n^{(\alpha,\beta)}$ to denote the relevant phase function; that is,

$$\psi_n^{(\alpha,\beta)}(t) = \psi^{(\alpha,\beta)}(t,n)\,. \tag{92}$$

We next calculate the inverse function $\left(\psi_n^{(\alpha,\beta)}\right)^{-1}$ of the phase function $\psi_n^{(\alpha,\eta)}$ as follows. First, we define ω_i for $i = 1, \ldots, m_t$ via the formula

$$\omega_i = \psi_n^{(\alpha,\beta)}(a_i)\,, \tag{93}$$

where m_t and $a_1, \ldots, a_{m_t}$ are as in the preceding section. Next, we form the collection

$$\xi_1, \ldots, \xi_{M_t} \tag{94}$$

of points obtained by taking the union of 16-point Chebyshev grids on each of the intervals

$$[\omega_1, \omega_2]\,, [\omega_2, \omega_3]\,, \ldots, [\omega_{m_t-1}, \omega_{m_t}]\,. \tag{95}$$

Using Newton's method, we compute the value of the inverse function of $\psi_n^{(\alpha,\beta)}$ at each of the points (94). More explicitly, we traverse the set (94) in descending order, and for each node ξ_j solve the equation

$$\psi_n^{(\alpha,\beta)}(y_j) = \xi_j \tag{96}$$

for y_j so as to determine the value of

$$\left(\psi_n^{(\alpha,\beta)}\right)^{-1}(\xi_j). \tag{97}$$

The values of the derivative of $\psi_n^{(\alpha,\beta)}$ are a by-product of the procedure used to construct $\psi_n^{(\alpha,\beta)}$, and so the evaluation of the derivative of the phase function is straightforward. The discretization scheme used to represent the phase function was chosen so as to be dense enough to represent both the phase functions and their inverse functions. Once the values of the inverse function at the nodes (94) are known, barycentric Chebyshev interpolation enables us to evaluate ψ^{-1} at any point on the interval (ω_1, ω_{m_t}). From (43), we see that the k^{th} zero t_k of $\tilde{P}_n^{(\alpha,\beta)}$ is given by the formula

$$t_k = \left(\psi_n^{(\alpha,\beta)}\right)^{-1}\left(\frac{\pi}{2} + k\pi\right). \tag{98}$$

These are the nodes of the trigonometric Gauss-Jacobi quadrature rule, and we use (98) to compute them. The corresponding weights are given by the formula

$$w_k = \frac{\pi}{\frac{d}{dt}\psi_n^{(\alpha,\beta)}(t_k)}. \tag{99}$$

The nodes of the Gauss-Jacobi quadrature rule are given by

$$x_j = \cos\left(t_{n-k+1}\right), \tag{100}$$

and the corresponding weights are given by

$$v_k = \frac{\pi 2^{\alpha+\beta+1} \sin\left(\frac{t}{2}\right)^{2\alpha+1} \cos\left(\frac{t}{2}\right)^{2\beta+1}}{\frac{d}{dt}\psi_n^{(\alpha,\beta)}(t_{n-k+1})}. \tag{101}$$

Formula (101) can be obtained by combining (43) with Formula (15.3.1) in Chapter 15 of [37]. That (99) gives the values of the weights for (5) is then obvious from a comparison of the form of that rule and (6).

The cost of computing the phase function and its inverse is $\mathcal{O}\left(\log(n)\right)$, while the cost of computing each node and weight pair is independent of n. Thus the algorithm of this section has an asymptotic running time of $\mathcal{O}\left(n\right)$. The nodes and weights of the trigonometric form of the Gauss-Jacobi quadrature rule are computed with near machine precision relative accuracy, as are the weights of the rule (5). However, the obtained values of the nodes x_k of (5) only achieve high absolute accuracy. Relative accuracy is lost when the change of variables (100) is applied to the nodes t_k of the trigonometric form of the rule which are close to $\pi/2$ owing to the large condition number of evaluation of cosine near $\pi/2$. By constructing a second phase function for the normal form

$$y''(x) + \left(\frac{1}{4}\frac{1-\alpha^2}{(1-x)^2} + \frac{1}{4}\frac{1-\beta^2}{(1+x)^2} + \frac{\nu(\nu+\alpha+\beta+1) + \frac{1}{2}(\alpha+1)(\beta+1)}{1-x^2}\right) y(x) = 0$$

of Jacobi's differential equation, the nodes of (5) near 0 can be computed with near machine precision relative accuracy. Since our principal interest is in the trigonometric form of the Gauss-Jacobi rule, we do not pursue this approach here.

6. Application of the Jacobi Transform and its Inverse

In this section, we describe a procedure for applying the n^{th} order Jacobi transform — which is represented by the $n \times n$ matrix $\mathcal{J}_n^{(\alpha,\beta)}$ defined via (10) — to a vector x. Since $\mathcal{J}_n^{(\alpha,\beta)}$ is orthogonal, the inverse Jacobi transform simply consists of applying its transpose. We omit a discussion of the minor and obvious changes to the algorithm of this section required to do so.

A precomputation, which includes as a step the procedure for the construction of the nonoscillatory phase and amplitude functions described Section 4, is necessary before our algorithm for applying the forward Jacobi transform can be used. We first discuss the outputs of the precomputation procedure and our method for using them to apply the Jacobi transform. Afterward, we describe the precomputation procedure in detail.

The precomputation phase of this algorithm takes as input the order n of the transform and parameters α and β in the interval $(-1/2, 1/2)$. Among other things, it produces the nodes $t_1, \ldots, t_n$ and weights $w_1, \ldots, w_n$ of the n-point modified Gauss-Jacobi quadrature rule, and a second collection of points $x_1, \ldots, x_n$ such that for each $j = 1, \ldots, n$, x_j is the node from the equispaced grid

$$0,\ 2\pi \cdot \frac{1}{n},\ 2\pi \cdot \frac{2}{n},\ \ldots,\ 2\pi \cdot \frac{n-1}{n} \tag{102}$$

which is closest to t_j. In addition, the precomputation phase constructs a complex-valued $n \times r$

matrix L, a complex-valued $r \times n$ matrix R and a real-valued $n \times 27$ matrix V such that

$$\mathcal{J}_n^{(\alpha,\beta)} \approx \left(\mathrm{Re}\left(\mathcal{F}_n \circ (LR)\right) + \begin{pmatrix} V & 0 \end{pmatrix}\right) W, \tag{103}$$

where r is the numerical rank of a matrix we define shortly, $\mathcal{F}_n$ is the $n \times n$ matrix whose (j,k) entry is

$$\exp(ix_j(k-1)), \tag{104}$$

W is the $n \times n$ matrix

$$W = \begin{pmatrix} \sqrt{w_1} & 0 & 0 & 0 \\ 0 & \sqrt{w_2} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \sqrt{w_n} \end{pmatrix}, \tag{105}$$

and $\circ$ denotes the Hadamard product. The matrix $\mathcal{F}_n$ can be applied to a vector in $\mathcal{O}\left(n\log(n)\right)$ operations by executing r fast Fourier transforms. Moreover, if we use $u_1, \ldots, u_r$ to denote the $n \times 1$ matrices which comprise the columns of the matrix L and $v_1, \ldots, v_r$ to denote the $1 \times n$ matrices which comprise the rows of the matrix R, then

$$LR = u_1 v_1 + \ldots + u_r v_r \tag{106}$$

and

$$\mathcal{F}_n \circ (LR) = D_{v_1} \mathcal{F}_n D_{u_1} + \cdots + D_{v_r} \mathcal{F}_n D_{u_r}, \tag{107}$$

where D_u denotes the $n \times n$ diagonal matrix with entries u on the diagonal. From (107), we see that $\mathcal{F}_n \circ (LR)$ can be applied in $\mathcal{O}\left(rn\log(n)\right)$ operations using the fast Fourier transform. This is the approach used in [34] to rapidly apply discrete nonuniform Fourier transforms. Since the matrices $\begin{pmatrix} V & 0 \end{pmatrix}$ and W can each be applied to a vector in $\mathcal{O}\left(n\right)$ operations, the asymptotic running time of this procedure for applying the forward Jacobi transform is $\mathcal{O}\left(rn\log(n)\right)$.

We now describe the precomputation step. First, the procedure of Section 4 is used to construct nonoscillatory phase and amplitude functions $\psi^{(\alpha,\beta)}$ and $M^{(\alpha,\beta)}$; the parameter $N_{\max}$ is taken to be n. In what follows, we let M_t, M_ν, a_j, b_j, τ_j and γ_k be as in Section 4. Next, the procedure of Section 5 is used to construct the nodes and weights of the modified Gauss-Jacobi quadrature rule of order n. We now form an interpolative decomposition

$$\begin{pmatrix} A^{(1)} \\ A^{(2)} \end{pmatrix} \approx \begin{pmatrix} B^{(1)} \\ B^{(2)} \end{pmatrix} \widetilde{R} \tag{108}$$

(see Section 3.6) of the matrix

$$\begin{pmatrix} A^{(1)} \\ A^{(2)} \end{pmatrix}, \tag{109}$$

where $A^{(1)}$ is the $M_t \times M_\nu$ matrix whose (j,k) entry is given by

$$A^{(1)}_{jk} = M^{(\alpha,\beta)}(\tau_j, k-1) \exp\left(i\left(\psi^{(\alpha,\beta)}(\tau_j,\gamma_k) - \tau_j\gamma_k\right)\right), \tag{110}$$

and $A^{(2)}$ is the $16 \times M_\nu$ matrix whose entries are

$$A^{(2)}_{jk} = \exp\left(i\ \sigma_j\ \frac{\gamma_k}{N_{\max}}\right) \tag{111}$$

with $\sigma_1 \ldots, \sigma_{16}$ the nodes of the 16-point Chebyshev grid on the interval $[0, 2\pi]$. We use the procedure of [10] with $\epsilon = 10^{-12}$ requested accuracy to do so. We view this procedure as choosing a collection

$$\gamma_{s_1} < \ldots < \gamma_{s_r} \tag{112}$$

of the nodes from the set (83), where r is the numerical rank of matrix (109) to precision ϵ, as determined by the interpolative decomposition procedure, and constructing a device (the matrix $\widetilde{R}$) for evaluating functions of the form

$$f(\nu) = M^{(\alpha,\beta)}(t,\nu) \exp\left(i\left(\psi^{(\alpha,\beta)}(t,\nu) - \nu t\right)\right) \tag{113}$$

and

$$g(\nu) = \exp\left(it\frac{\nu}{N_{\max}}\right), \tag{114}$$

where t is any point in the interval (74), at the nodes (83) given their values at the nodes (112). That t can take on any value in the interval (74) and not just the special values (80) is a consequence of the fact that the piecewise bivariate Chebyshev discretization scheme described in the preceding section suffices to represent these functions. We expect (109) to be low rank from the discussion in the introduction and the estimates of Section 3.5. We factor the augmented matrix (109) rather than $A^{(1)}$ because in later steps of the precomputation procedure, we use the matrix $\widetilde{R}$ to interpolate functions of the form (114) as well as those of the form (113).

In what follows, we denote by I_{left} the $n \times M_t$ matrix which interpolates piecewise polynomials of degree 15 on the intervals (81) to their values at the points $t_1, \ldots, t_n$. See Section 3.7 for a careful definition of the matrix I_{left} and a discussion of its properties. Similarly, we let I_{right} be the $M_\nu \times n$ matrix

$$I_{\text{right}} = \begin{pmatrix} 0 & I^{(1)}_{\text{right}} \end{pmatrix}, \tag{115}$$

where $I^{(1)}_{\text{right}}$ is the $M_\nu \times (n-27)$ matrix which interpolates piecewise polynomials of degree 23 on the intervals (84) to their values at the points $27, 28, 29, \ldots, n-1$ (recall that the phase and amplitude functions represent the Jacobi polynomials of degrees greater than or equal to 27). We next form the $M_t \times r$ matrix whose (j,k) entry is

$$M^{(\alpha,\beta)}(\tau_j, \gamma_{s_k}) \exp\left(i\left(\psi\left(\tau_j, \gamma_{s_k}\right) - \tau_j \gamma_{s_k}\right)\right) \tag{116}$$

and multiply it on the left by I_{left} so as to form the $n \times r$ matrix whose (j,k) entry is

$$M^{(\alpha,\beta)}(t_j, \gamma_{s_k}) \exp\left(i\left(\psi\left(t_j, \gamma_{s_k}\right) - t_j \gamma_{s_k}\right)\right). \tag{117}$$

For $j = 1, \ldots, n$ and $k = 1, \ldots, r$, we scale the (j,k) entry of this matrix by

$$\exp\left(i(t_j - x_j)\gamma_{s_k}\right) \tag{118}$$

in order to form the $n \times r$ matrix L whose (j,k) entry is

$$L_{jk} = M^{(\alpha,\beta)}(t_j, \gamma_{s_k}) \exp\left(i\left(\psi\left(t_j, \gamma_{s_k}\right) - x_j \gamma_{s_k}\right)\right). \tag{119}$$

We multiply the $r \times M_\nu$ matrix $\widetilde{R}$ on the right by I_{right} to form the $r \times n$ matrix R. It is because of the scaling by (118) that we factor the augmented matrix (109) rather than $A^{(1)}$. The values of (118) can be correctly interpolated by the matrix $\widetilde{R}$ because of the addition of $A^{(2)}$.

As the final step in our precomputation procedure, we use the well-known three-term recurrence

relations satisfied by the Jacobi polynomials to form the $n \times k$ matrix

$$V = \begin{pmatrix} \tilde{P}_0^{(\alpha,\beta)}(t_1) & \tilde{P}_1^{(\alpha,\beta)}(t_1) & \cdots & \tilde{P}_{26}^{(\alpha,\beta)}(t_1) \\ \tilde{P}_0^{(\alpha,\beta)}(t_2) & \tilde{P}_1^{(\alpha,\beta)}(t_2) & \cdots & \tilde{P}_{26}^{(\alpha,\beta)}(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{P}_0^{(\alpha,\beta)}(t_n) & \tilde{P}_1^{(\alpha,\beta)}(t_n) & \cdots & \tilde{P}_{26}^{(\alpha,\beta)}(t_n) \end{pmatrix}. \tag{120}$$

The highest order term in the asymptotic running time of the precomputation algorithm is $\mathcal{O}(rn)$, where r is the rank of (109), and it comes from the application of the interpolation matrices I_{right} and I_{left}. Based on our numerical experiments, we conjecture that

$$r = \mathcal{O}\left(\log(n)\right), \tag{121}$$

in which case the asymptotic running time of the procedure is

$$\mathcal{O}\left(n \log^2(n)\right). \tag{122}$$

Remark 3. *There are many mechanisms for accelerating the procedure used here for the calculation of interpolative decomposition (particularly, randomized algorithms [14] and methods based on adaptive cross approximation). However, in our numerical experiments we found that the cost of the construction of the phase and amplitude functions dominated the running time of our precomputation step for small n, and that the cost of interpolation did so for large n. As a consequence, the optimization of our approach to forming interpolative decompositions is a low priority.*

Remark 4. *The rank of the low-rank approximation by interpolative decompositions is not optimally small. A fast truncated SVD based on the interpolative decomposition by Chebyshev interpolation can provide the optimal rank of the low-rank approximation in* (103) *and thereby speed up the calculation in* (107) *by a small factor (see [29] for a comparison). However, we do not explore this in this paper.*

Remark 5. *The procedure of this section can be modified rather easily to apply several transforms related to the Jacobi transform. For instance, the mapping which take the coefficients (8) in the expansion (9) to values of f at the nodes of a Chebyshev quadrature can be applied using the procedure of this section by simply modifying $t_1, \ldots, t_n$. The Jacobi-Chebyshev transform can then be computed from the resulting value using the fast Fourier transform.*

7. Numerical Experiments

In this section, we describe numerical experiments conducted to evaluate the performance of the algorithms of this paper. Our code was written in Fortran and compiled with the GNU Fortran compiler version 4.8.4. It uses version 3.3.2 of the FFTW3 library [18] to apply the discrete Fourier transform. We have made our code, including that for all of the experiments described here, available on GitHub at the following address:

`https://gitlab.com/FastOrthPolynomial/Jacobi.git`

All calculations were carried out on a single core of workstation computer equipped with an Intel Xeon E5-2697 processor running at 2.6 GHz and 512GB of RAM.

7.1. Numerical evaluation of Jacobi polynomials

In a first set of experiments, we measured the speed of the procedure of Section 4 for the construction of nonoscillatory phase and amplitude functions, and the speed and accuracy with which Jacobi polynomials can be evaluated using them.

We did so in part by comparing our approach to a reference method which combines the Liouville-Green type expansions of Barratella and Gatteschi (see Section 3.2) with the trigonometric expansions of Hahn (see Section 3.3). Modulo minor implementation details, the reference method we used is the same as that used in [25] to evaluate Jacobi polynomials and is quite similar to the approach of [4] for the evaluation of Legendre polynomials. To be more specific, we used the expansion (23) to evaluate $\tilde{P}_\nu^{(\alpha,\beta)}$ for all $t \geq 0.2$ and the trigonometric expansion (29) to evaluate it when $0 < t < 0.2$. The expansion (29) is much more expensive than (23); however, the use of (23) is complicated by the fact that the higher order coefficients are not known explicitly. We calculated 16^{th} order Taylor expansion for the coefficients B_1, A_1, B_2 and A_2, but the use of these approximations limited the range of t for we which we could apply (23).

In order to perform these comparisons, we fixed $\alpha = -1/4$ and $\beta = 1/3$ and choose several values of $N_{\max}$. For each such value, we carried out the procedure for the construction of the phase function described in Section 4 and evaluated $\tilde{P}_\nu^{(\alpha,\beta)}$ for 200 randomly chosen values of t in the interval $(0, \pi)$ and 200 randomly chosen values of ν in the interval $(1, N_{\max})$ using both our algorithm and the reference method. Table 1 reports the results. For several different pairs of the parameters α, β, we repeated these experiments, measuring the time required to construct the phase function and the maximum observed absolute errors. Figure 2 displays the results. We note that the condition number of evaluation of the Jacobi polynomials increases with order, with the consequence that the obtained accuracy of both the approach of this paper and the reference method decrease as a function of degree. See Section 2 of [4] for an extensive discussion of the condition number of evaluation of the Legendre polynomials. The errors observed in Table 1 and Figure 2 are consistent with this fact.

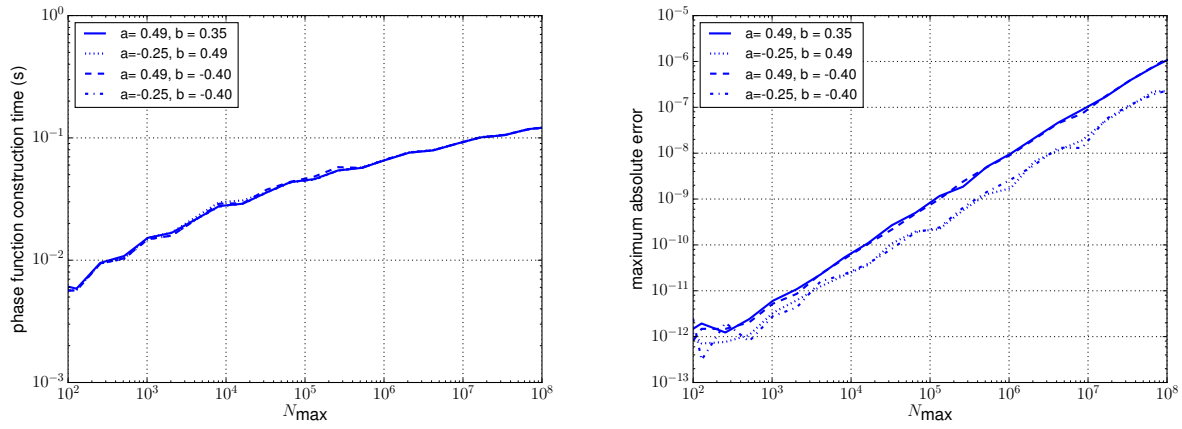

Figure 2: The plot on the left shows how the time required to construct the phase and amplitude functions varies with $N_{\max}$ for various values of the parameters α and β. The plot on the right shows the maximum absolute errors which were observed when evaluating Jacobi polynomials as a function of $N_{\max}$ for various values of the parameters α and β.

$N_{\max}$	Phase function construction time	Avg. eval time algorithm of this paper	Avg. eval time asymptotic expansions	Largest absolute error	Expansion size (MB)
100	1.15×10^{-02}	2.82×10^{-07}	3.46×10^{-05}	1.31×10^{-12}	1.90×10^{-01}
128	5.77×10^{-03}	7.82×10^{-07}	2.56×10^{-05}	8.89×10^{-13}	1.90×10^{-01}
256	9.46×10^{-03}	5.81×10^{-07}	2.47×10^{-05}	9.86×10^{-13}	3.19×10^{-01}
512	1.03×10^{-02}	5.33×10^{-07}	2.44×10^{-05}	1.50×10^{-12}	3.54×10^{-01}
1,024	1.48×10^{-02}	5.81×10^{-07}	2.42×10^{-05}	2.34×10^{-12}	5.19×10^{-01}
2,048	1.60×10^{-02}	5.90×10^{-07}	2.49×10^{-05}	5.48×10^{-12}	5.66×10^{-01}
4,096	2.15×10^{-02}	5.70×10^{-07}	2.30×10^{-05}	1.39×10^{-11}	7.66×10^{-01}
8,192	2.75×10^{-02}	5.69×10^{-07}	2.34×10^{-05}	1.71×10^{-11}	9.89×10^{-01}
16,384	2.92×10^{-02}	5.66×10^{-07}	2.36×10^{-05}	2.71×10^{-11}	$1.06\times10^{+00}$
32,768	3.60×10^{-02}	1.03×10^{-06}	2.31×10^{-05}	9.68×10^{-11}	$1.31\times10^{+00}$
65,536	4.37×10^{-02}	5.36×10^{-07}	2.30×10^{-05}	2.31×10^{-10}	$1.60\times10^{+00}$
131,072	4.59×10^{-02}	5.65×10^{-07}	2.32×10^{-05}	4.64×10^{-10}	$1.69\times10^{+00}$
262,144	5.45×10^{-02}	5.66×10^{-07}	2.41×10^{-05}	6.96×10^{-10}	$2.01\times10^{+00}$
524,288	5.69×10^{-02}	6.03×10^{-07}	2.24×10^{-05}	1.58×10^{-09}	$2.11\times10^{+00}$
1,048,576	6.67×10^{-02}	5.68×10^{-07}	2.42×10^{-05}	1.88×10^{-09}	$2.46\times10^{+00}$
2,097,152	8.07×10^{-02}	5.86×10^{-07}	2.46×10^{-05}	6.20×10^{-09}	$2.84\times10^{+00}$
4,194,304	7.91×10^{-02}	5.64×10^{-07}	2.40×10^{-05}	9.51×10^{-09}	$2.97\times10^{+00}$
8,388,608	9.00×10^{-02}	8.71×10^{-07}	2.37×10^{-05}	1.76×10^{-08}	$3.38\times10^{+00}$
16,777,216	1.01×10^{-01}	5.86×10^{-07}	2.61×10^{-05}	3.65×10^{-08}	$3.81\times10^{+00}$
33,554,432	1.11×10^{-01}	5.80×10^{-07}	2.35×10^{-05}	7.77×10^{-08}	$3.97\times10^{+00}$
67,108,864	1.17×10^{-01}	5.99×10^{-07}	2.36×10^{-05}	2.01×10^{-07}	$4.43\times10^{+00}$
134,217,728	1.36×10^{-01}	6.25×10^{-07}	2.42×10^{-05}	3.74×10^{-07}	$4.93\times10^{+00}$

Table 1: A comparison of the time required to evaluate Jacobi polynomials via the algorithm of this paper with the time required to do so using certain asymptotic expansions. Here, the parameters in Jacobi's differential equation were taken to be $\alpha = -1/4$ and $\beta = 1/3$. All times are in seconds.

The asymptotic complexity of the procedure of Section 4 is $\mathcal{O}\left(\log^2\left(N_{\max}\right)\right)$; however, the running times shown on the left-hand side of Figure 2 appear to grow more slowly than this. This suggests that a lower order term with a large constant is present.

We also compared the method of this paper with the results obtained by using the well-known three-term recurrence relations satisfied by solutions of Jacobi's differential equation (which can be found in Section 10.8 of [16], among many other sources) to evaluate the Jacobi polynomials. Obviously, such an approach is unsuitable as a mechanism for evaluating a single Jacobi polynomial of large degree. However, up to a certain point, the recurrence relations are efficient and effective and it is of interest to compare the approaches. Table 2 does so. In these experiments, α was taken to be $1/4$ and β was $-1/3$.

7.2. Calculation of Gauss-Jacobi quadrature rules

In this section, we describe experiments conducted to measure the speed and accuracy with which the algorithm of Section 5 constructs Gauss-Jacobi quadrature rules. We used the Hale-Townsend algorithm [25] as a basis for comparison. It takes $\mathcal{O}\left(n\right)$ operations to construct an n-point Gauss-Jacobi quadrature rule, and calculates the quadrature nodes with double precision absolute accuracy and the quadrature weights with double precision relative accuracy. The Hale-Townsend algorithm

$N_{\max}$	Phase function construction time	Average evaluation time algorithm of this paper	Average evaluation time recurrence relations	Largest absolute error
32	2.63×10^{-03}	4.37×10^{-07}	1.74×10^{-06}	3.34×10^{-13}
64	2.56×10^{-03}	5.04×10^{-07}	2.33×10^{-06}	6.58×10^{-13}
128	5.65×10^{-03}	7.73×10^{-07}	3.72×10^{-06}	1.02×10^{-12}
256	9.84×10^{-03}	5.49×10^{-07}	6.16×10^{-06}	1.43×10^{-12}
512	1.02×10^{-02}	5.60×10^{-07}	1.15×10^{-05}	1.69×10^{-12}
1,024	1.48×10^{-02}	5.82×10^{-07}	2.05×10^{-05}	8.31×10^{-12}
2,048	1.59×10^{-02}	5.79×10^{-07}	3.76×10^{-05}	1.58×10^{-11}
4,096	2.12×10^{-02}	5.63×10^{-07}	7.58×10^{-05}	3.83×10^{-11}
8,192	2.73×10^{-02}	5.64×10^{-07}	1.51×10^{-04}	1.84×10^{-10}
16,384	2.90×10^{-02}	5.68×10^{-07}	2.94×10^{-04}	1.98×10^{-10}
32,768	3.58×10^{-02}	1.03×10^{-06}	5.96×10^{-04}	7.62×10^{-11}

Table 2: A comparison of the time required to evaluate Jacobi polynomials via the algorithm of this paper with the time required to do so using the well-known three-term recurrence relations. Here, the parameters in Jacobi's differential equation were taken to be $\alpha = 1/4$ and $\beta = -1/3$. All times are in seconds.

is faster and more accurate (albeit less general) than the Glaser-Liu-Rokhlin algorithm [20], which can be also used to construct n-point Gauss-Jacobi quadrature rules in $O(n)$ operations. We note that the algorithm of [3] for the construction of Gauss-Legendre quadrature rules (and not more general Gauss-Jacobi quadrature rules) is more efficient than the method of this paper. The recent preprint [19] generalizes the approach of [3] to the case of Gauss-Jacobi quadrature rules. It is effective for α and β greater than $1/2$ and allows for the calculation of all of the nodes with near machine precision relative accuracy.

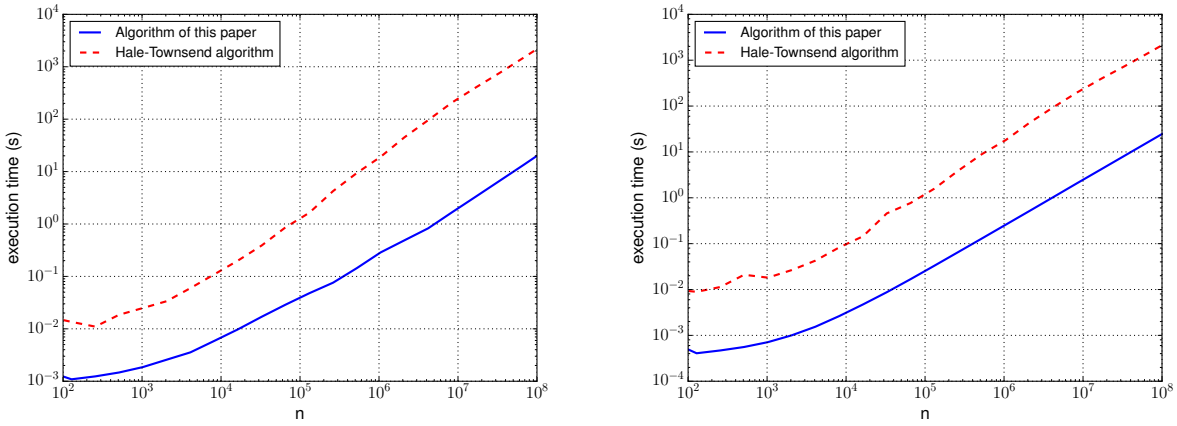

Figure 3: A comparison of the running time of the algorithm of Section 5 for the calculation of Gauss-Jacobi quadrature rules with the algorithm of Hale-Townsend [25]. In the experiments whose results are shown on the left, the parameters were taken to be $\alpha = 0.25$ and $\beta = 0.40$ while in the experiments corresponding to the plot on the right, the parameters were $\alpha = -0.49$ and $\beta = 0.25$.

For $\alpha = 0$ and $\beta = -4/10$ and various values of n, we constructed the n-point Gauss-Jacobi quadrature rule using both the algorithm of Section 5 and the Julia implementation [24] of the Hale-Townsend algorithm made available by the authors of [25]. For each chosen value of n,

we measured the relative difference in 100 randomly selected weights. Unfortunately, in some cases [24] loses a small amount of accuracy when evaluating quadrature weights corresponding to quadrature nodes near the points ± 1. Accordingly, when choosing random nodes, we omitted those corresponding to the 20 quadrature nodes closest to each of the points ± 1. We note that the loss of accuracy in [24] is merely a problem with that particular implementation of the Hale-Townsend algorithm and not with the underlying scheme. Indeed, the MATLAB implementation of the Hale-Townsend algorithm included with the Chebfun package [12] does not suffer from this defect. We did not compare against the MATLAB version of the code because it is somewhat slower than the Julia implementation. Table 3 reports the results. We began our comparison with $n = 101$ because when $n \leq 100$, the Hale-Townsend code combines the well-known three-term recurrence relations satisfied by solutions of Jacobi's differential equation and Newton's method to construct the n-point Gauss-Jacobi quadrature rule. This is also the strategy we recommend for constructing Gauss-Jacobi rules when n is small.

For different pairs of the parameters α and β and various values of n, we used [24] and the algorithm of this paper to produce n-point Gauss-Jacobi quadrature rules and compared the running times of these two algorithms. Figure 3 displays the results.

n	Running time of the algorithm of Section 5	Running time of the algorithm of [25]	Ratio	Maximum relative error in weights
101	4.45×10^{-04}	9.62×10^{-03}	$2.15\times10^{+01}$	4.47×10^{-15}
128	4.00×10^{-04}	9.26×10^{-03}	$2.31\times10^{+01}$	4.78×10^{-15}
256	4.54×10^{-04}	1.15×10^{-02}	$2.54\times10^{+01}$	5.76×10^{-15}
512	5.34×10^{-04}	2.14×10^{-02}	$4.00\times10^{+01}$	7.04×10^{-15}
1,024	6.64×10^{-04}	2.27×10^{-02}	$3.41\times10^{+01}$	6.26×10^{-15}
2,048	8.86×10^{-04}	3.00×10^{-02}	$3.39\times10^{+01}$	6.99×10^{-15}
4,096	1.31×10^{-03}	5.28×10^{-02}	$4.01\times10^{+01}$	7.45×10^{-15}
8,192	2.15×10^{-03}	8.76×10^{-02}	$4.06\times10^{+01}$	7.99×10^{-15}
16,384	3.80×10^{-03}	2.68×10^{-01}	$7.07\times10^{+01}$	1.07×10^{-14}
32,768	7.03×10^{-03}	4.45×10^{-01}	$6.33\times10^{+01}$	8.29×10^{-15}
65,536	1.35×10^{-02}	7.91×10^{-01}	$5.85\times10^{+01}$	9.23×10^{-15}
131,072	2.64×10^{-02}	$1.61\times10^{+00}$	$6.10\times10^{+01}$	1.04×10^{-14}
262,144	5.22×10^{-02}	$4.14\times10^{+00}$	$7.92\times10^{+01}$	8.44×10^{-15}
524,288	1.03×10^{-01}	$9.06\times10^{+00}$	$8.74\times10^{+01}$	1.09×10^{-14}
1,048,576	2.06×10^{-01}	$1.80\times10^{+01}$	$8.73\times10^{+01}$	1.29×10^{-14}
2,097,152	4.11×10^{-01}	$4.26\times10^{+01}$	$1.03\times10^{+02}$	1.26×10^{-14}
4,194,304	8.24×10^{-01}	$9.27\times10^{+01}$	$1.12\times10^{+02}$	1.16×10^{-14}
8,388,608	$1.65\times10^{+00}$	$1.95\times10^{+02}$	$1.17\times10^{+02}$	1.36×10^{-14}
16,777,216	$3.30\times10^{+00}$	$3.86\times10^{+02}$	$1.16\times10^{+02}$	1.43×10^{-14}
33,554,432	$6.59\times10^{+00}$	$7.33\times10^{+02}$	$1.11\times10^{+02}$	1.43×10^{-14}
67,108,864	$1.31\times10^{+01}$	$1.42\times10^{+03}$	$1.08\times10^{+02}$	1.48×10^{-14}
100,000,000	$1.96\times10^{+01}$	$2.10\times10^{+03}$	$1.07\times10^{+02}$	1.77×10^{-14}

Table 3: The results of an experiment comparing the algorithm of [25] for the numerical calculation of Gauss-Jacobi quadrature rules with that of Section 5. In these experiments, the parameters were taken to be $\alpha = 0$ and $\beta = -4/10$. All times are in seconds.

7.3. The Jacobi transform

In these experiments, we measured the speed and accuracy of the algorithm of Section 6 for the application of the Jacobi transform and its inverse.

We did so in part by comparison with the algorithm of Slevinsky [36] for applying the Chebyshev-Jacobi and Jacobi-Chebyshev transforms. The Chebyshev-Jacobi transform is the map which takes the coefficients of the Chebyshev expansion of a function to the coefficients in its Jacobi expansion and the Jacobi-Chebyshev transform is the inverse of this map. Although these are not the transforms we apply, the Jacobi transform can be implemented easily by combining the method of [36] with the nonuniform fast Fourier transform (see, for instance, [27] and [26] for an approach of this type for applying the Legendre transform and its inverse). Other methods for applying the Jacobi transform, some of which have lower asymptotic complexity than the algorithm of [36] and the approach of this paper, are available. Butterfly algorithms such as [30, 31, 33] allow for the application of the Jacobi transform and various related mappings in $\mathcal{O}(n\log(n))$ operations; however, existing methods are either numerically unstable or they require expensive precomputation phases with higher asymptotic complexity. The Alpert-Rokhlin method [1] uses a multipole-like approach to apply the Chebyshev-Legendre and Legendre-Chebyshev transforms in $\mathcal{O}(n)$ operations. The Legendre transform can be computed in $\mathcal{O}(n\log(n))$ operations by combining this algorithm with the fast Fourier transform. This approach is extended in [28] in order to compute expansions in Jacobi polynomials in $\mathcal{O}(n\log(n))$ operations. The implementation of the FMM used in [28] appears to require lengthy precomputations. For instance, precomputations for various 16384 point Jacobi-to-Jacobi appear to take more than 30 seconds (see Table 2.6 of Section 3.5 of [28]). However, if more efficient FMM algorithms which require less precomputation time are used, then the approach of [28] and other algorithms might become competitive with the algorithm used here. A further discussion of methods for the application of the Jacobi transform and related mappings can be found in [36].

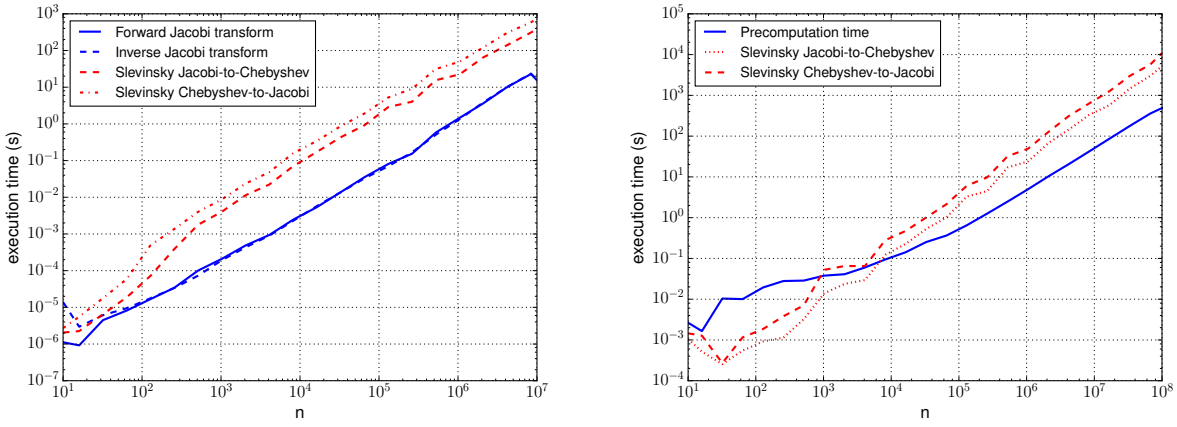

Figure 4: On the left is a comparison of the time required to apply the Chebyshev-Jacobi transform and its inverse using the algorithm of [36] with the time required to apply the forward and inverse Jacobi transforms via the algorithm of Section 6. On the right is a comparison of the cost of the precomputations for the procedure of Section 6 with the time required to apply the Jacobi-Chebyshev map and its inverse via the algorithm of [36]. In these experiments, α was taken to be $-1/4$ and β was 0.

Figure 4 and Table 4 report the results of our comparisons with the Julia implementation [35]

n	Forward Jacobi transform time algorithm of Section 6	Chebyshev-Jacobi transform time algorithm of [36]	Ratio
10	7.10×10^{-07}	1.70×10^{-06}	$2.39\times10^{+00}$
16	3.66×10^{-07}	2.10×10^{-06}	$5.76\times10^{+00}$
32	8.27×10^{-06}	6.34×10^{-06}	7.66×10^{-01}
64	8.52×10^{-06}	1.85×10^{-05}	$2.18\times10^{+00}$
128	1.33×10^{-05}	7.88×10^{-05}	$5.92\times10^{+00}$
256	2.83×10^{-05}	4.21×10^{-04}	$1.48\times10^{+01}$
512	1.01×10^{-04}	1.79×10^{-03}	$1.77\times10^{+01}$
1,024	1.88×10^{-04}	4.03×10^{-03}	$2.13\times10^{+01}$
2,048	4.75×10^{-04}	1.12×10^{-02}	$2.36\times10^{+01}$
4,096	1.33×10^{-03}	2.17×10^{-02}	$1.63\times10^{+01}$
8,192	2.59×10^{-03}	6.94×10^{-02}	$2.67\times10^{+01}$
16,384	6.44×10^{-03}	1.70×10^{-01}	$2.64\times10^{+01}$
32,768	1.41×10^{-02}	4.28×10^{-01}	$3.03\times10^{+01}$
65,536	3.13×10^{-02}	9.04×10^{-01}	$2.88\times10^{+01}$
131,072	9.40×10^{-02}	$2.85\times10^{+00}$	$3.03\times10^{+01}$
262,144	1.49×10^{-01}	$4.03\times10^{+00}$	$2.68\times10^{+01}$
524,288	5.51×10^{-01}	$1.52\times10^{+01}$	$2.75\times10^{+01}$
1,048,576	$1.44\times10^{+00}$	$2.21\times10^{+01}$	$1.53\times10^{+01}$
2,097,152	$3.90\times10^{+00}$	$6.56\times10^{+01}$	$1.68\times10^{+01}$
4,194,304	$9.74\times10^{+00}$	$1.40\times10^{+02}$	$1.44\times10^{+01}$
8,388,608	$2.36\times10^{+01}$	$3.10\times10^{+02}$	$1.31\times10^{+01}$
10,000,000	$1.41\times10^{+01}$	$3.82\times10^{+02}$	$2.70\times10^{+01}$

Table 4: A comparison of the time required to apply the Chebyshev-Jacobi transform using the algorithm of [36] with the time required to apply the forward Jacobi transform via the algorithm of Section 6. Here, α was taken to be $1/4$ and β was $-4/10$. All times are in seconds.

of Slevinsky's algorithm. The graph on the left side of Figure 4 compares the time taken by the algorithm of Section 6 to apply the Jacobi transform and its inverse with the time required to apply the Chebyshev-Jacobi mapping and its inverse via [36], while the graph on the right compares the time required by our precomputation procedure with the time required to apply the Chebyshev-Jacobi mapping and it inverse with Slevinsky's algorithm. Slevinsky's algorithm involves a precomputation step (which consists of "planning" the fast Fourier transform to be applied with the FFTW library). In the first set of these computations, those that are reported in Table 4 and on the left of Figure 4, we excluded the cost of this precomputation phase when measuring the time taken by Slevinsky's algorithm. In the second set of these calculations, those which compare the running time of our precomputation phase with the running time of Slevinsky's algorithm, we included the cost of this precomputation step in the running time of Slevinsky's algorithm. Figure 4 strongly suggests that the asymptotic running time of our algorithm for the application of the Jacobi transform is similar to the $O\left(n\log^2(n)/\log(\log(n))\right)$ complexity of Slevinsky's algorithm.

Owing to the loss of accuracy which arises when the Formula (2) is used to evaluate Jacobi polynomials of large degrees, we expect the error in the Jacobi transform of Section 6 to increase as the order of the transform increases. This is indeed the case, at least when it is applied to functions whose Jacobi coefficients do not decay or decay slowly. However, when the transform is applied

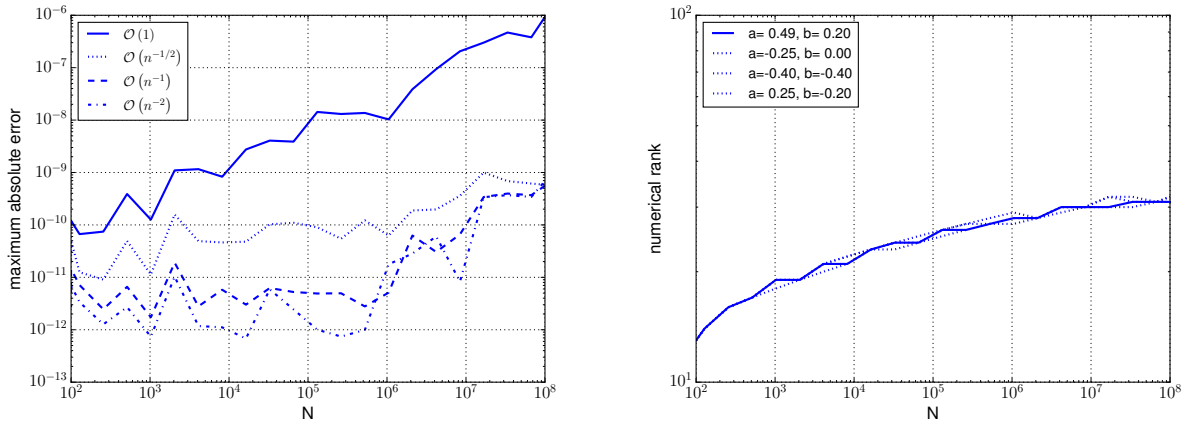

Figure 5: On the left are plots of the largest absolute errors which occurs when the composition of the inverse and forward Jacobi transforms of Section 6 are applied to vectors whose coefficients decay at varying rates. For these experiments, $a = -1/4$ and $b = 0$. On the right are plots of the rank of the matrix (109) as a function of $N_{\max}$ for various values of the parameters α and β.

to smooth functions, whose Jacobi coefficients decay rapidly, the errors grow more slowly than in the general case. This is the same as the behavior of the algorithms [36] and [27]. We carried out a further set of experiments to illustrate this effect. In particular, we applied the forward Jacobi transform followed by the inverse Jacobi transform to vectors which decay at various rates. We constructed test vectors by choosing their entries from a Gaussian distribution and then scaling them so as to achieve a desired rate of decay. Figure 5 reports the results. It also contains a plot of the rank of the matrix (109) as a function of n for various pairs of the parameters α and β.

We also compared the time required to apply the forward Jacobi transform via the algorithm of Section 6 with the time required to do so by evaluating the matrix $\mathcal{J}_n^{(\alpha,\beta)}$ using the well-known three-term recurrence relations and then applying it directly (we refer to this as the "brute force technique"). This is the methodology we recommend for transforms of small orders. In these experiments, the parameters α and β were taken to be $\alpha = 1/4$ and $\beta = -1/3$. Figure 6 shows the results.

8. Conclusion and Further Work

We have described a suite of fast algorithms for forming and manipulating expansions in Jacobi polynomials. They are based on the well-known fact that Jacobi's differential equation admits a nonoscillatory phase function. Our algorithms use numerical methods, rather than asymptotic expansions, to evaluate the phase function and other functions related to it.

It would be of some interest to accelerate the procedure of Section 4 for the construction of the nonoscillatory phase and amplitude functions. There are a number obvious mechanisms for doing so, but perhaps the most promising is the observation that the ranks of matrices with entries

$$\psi^{(\alpha,\beta)}(t_j, \nu_k) \tag{123}$$

and

$$M^{(\alpha,\beta)}(t_j, \nu_k) \tag{124}$$

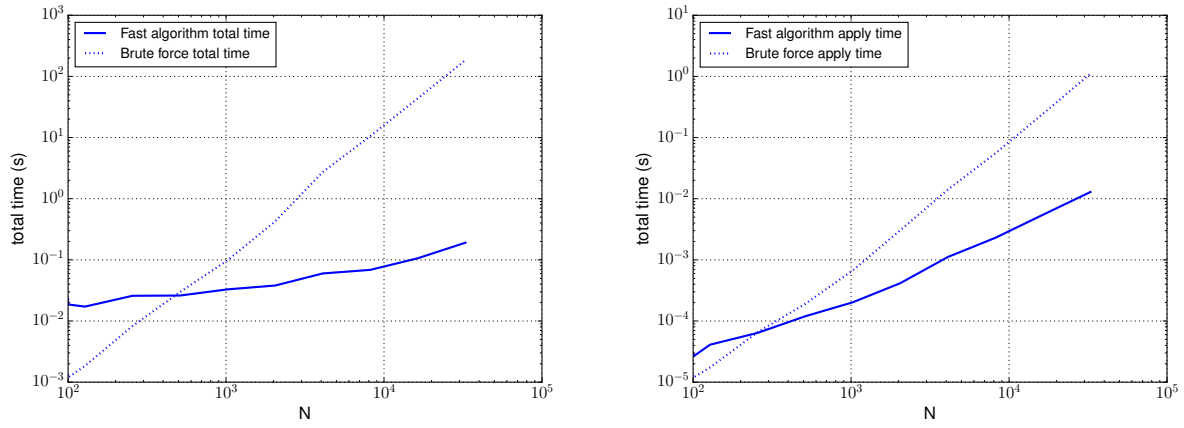

Figure 6: A comparison of the time take to apply the Forward Jacobi transform via "brute force" with the time required to do so via the algorithm of Section 6. On the left is a plot of the total time taken. For the algorithm of Section 6 this includes the time taken by the precomputation phase, while for "brute force" technique this includes the time required to evaluate the entries of the matrix $\mathcal{J}_n^{(\alpha,\beta)}$ via three-term recurrence relations. On the right is a comparison of the application times only. Here, the parameters are $\alpha = 1/4$ and $\beta = -1/3$.

are quite low — indeed, in the experiments of this paper they were never observed to be larger than 40. This means that, at least in principle, the nonoscillatory phase and amplitude can be represented via 40×40 matrices, and that a carefully designed spectral scheme which takes this into account could compute the 40 required solutions of the ordinary differential equation (42) extremely efficiently.

The authors are investigating such an approach to the construction of $\psi^{(\alpha,\beta)}$ and $M^{(\alpha,\beta)}$.

As the parameters α and β increase beyond $1/2$, our algorithms become less accurate, and they ultimately fail. This happens because for values of the parameters α and β greater than $1/2$, the Jacobi polynomials have turning points and the crude approximation (14) becomes inadequate. An obvious remedy is to use a more sophisticated approximations for $\psi^{(\alpha,\beta)}$ and $M^{(\alpha,\beta)}$. The authors will report on extensions of this work which make use of such an approach at a later date.

9. Acknowledgments

James Bremer was supported in part by a UC Davis Chancellor's Fellowship and by National Science Foundation grant DMS-1418723. Haizhao Yang thanks the support of the start-up grant by the Department of Mathematics at the National University of Singapore, and the support of the Ministry of Education in Singapore under the grant MOE2018-T2-2-147.

10. References

References

[1] Alpert, B. K., and Rokhlin, V. A fast algorithm for the evaluation of Legendre expansions. *SIAM Journal on Scientific and Statistical Computing 12*, 1 (1991), 158–179.

[2] Baratella, P., and Gatteschi, L. The bounds for the error term of an asymptotic approximation of Jacobi polynomials. In *Orthogonal Polynomials and their Applications, Lecture Notes in Mathematics 1329*. Springer, 1988, pp. 203–221.

[3] Bogaert, I. Iteration-free computation of Gauss-Legendre quadrature nodes and weights. *SIAM Journal on Scientific Computing 36* (2014), A1008–A1026.

[4] Bogaert, I., Michiels, B., and Fostier, J. $O(1)$ computation of Legendre polynomials and Gauss-Legendre nodes and weights for parallel computing. *SIAM Journal on Scientific Computing 34* (2012), C83–C101.

[5] Bremer, J. An algorithm for the rapid numerical evaluation of Bessel functions of real orders and arguments. *Advances in Computational Mathematics*, to appear.

[6] Bremer, J. On the numerical solution of second order differential equations in the high-frequency regime. *Applied and Computational Harmonic Analysis 44*, 312–349.

[7] Bremer, J. On the numerical calculation of the roots of special functions satisfying second order ordinary differential equations. *SIAM Journal on Scientific Computing 39* (2017), A55–A82.

[8] Bremer, J. An algorithm for the numerical evaluation of the associated legendre functions that runs in time independent of degree and order. *Journal of Computational Physics 360* (2018), 15–28.

[9] Candès, E., Demanet, L., and Ying, L. Fast computation of Fourier integral operators. *SIAM Journal on Scientific Computing 29*, 6 (2007), 2464–2493.

[10] Cheng, H., Gimbutas, Z., Martinsson, P., and Rokhlin, V. On the compression of low rank matrices. *SIAM Journal on Scientific Computing 26* (2005), 1389–1404.

[11] *NIST Digital Library of Mathematical Functions*. http://dlmf.nist.gov/, Release 1.0.13 of 2016-09-16. F. W. J. Olver, A. B. Olde Daalhuis, D. W. Lozier, B. I. Schneider, R. F. Boisvert, C. W. Clark, B. R. Miller and B. V. Saunders, eds.

[12] Driscoll, T. A., Hale, N., and Trefethen, L. N. *Chebfun Guide*. Pafnuty Publications, Oxford, 2014.

[13] Dunster, T. M. Asymptotic approximations for the Jacobi and ultraspherical polynomials, and related functions. *Methods and Applications of Analysis 6* (1999), 281–316.

[14] Engquist, B., and Ying, L. A fast directional algorithm for high frequency acoustic scattering in two dimensions. *Commun. Math. Sci. 7*, 2 (2009), 327–345.

[15] Erdélyi, A., et al. *Higher Transcendental Functions*, vol. I. McGraw-Hill, 1953.

[16] Erdélyi, A., et al. *Higher Transcendental Functions*, vol. II. McGraw-Hill, 1953.

[17] Frenzen, C., and Wong, R. A uniform asymptotic expansion of the Jacobi polynomials with error bounds. *Canadian Journal of Mathematics 37* (1985), 979–1007.

[18] Frigo, M., and Johnson, S. G. The design and implementation of FFTW3. *Proceedings of the IEEE 93*, 2 (2005), 216–231. Special issue on "Program Generation, Optimization, and Platform Adaptation".

[19] Gil, A., Segura, J., and Temme, N. Non-iterative computation of Gauss-Jacobi quadrature by asymptotic expansions for large degree. *SIAM Journal on Scientific Computing*, to appear.

[20] Glaser, A., Liu, X., and Rokhlin, V. A fast algorithm for the calculation of the roots of special functions. *SIAM Journal on Scientific Computing 29* (2007), 1420–1438.

[21] Greengard, L. Spectral integration and two-point boundary value problems. *SIAM Journal of Numerical Analysis 28* (1991), 1071–1080.

[22] Greengard, L., and Lee, J.-Y. Accelerating the nonuniform fast fourier transform. *SIAM Review 46*, 3 (2004), 443–454.

[23] Hahn, E. Asymptotik bei Jacobi-polynomen und Jacobi-funktionen. *Mathematische Zeitschrift 171* (1980), 201–226.

[24] Hale, N., and Townsend, A. Fast Gauss Quadrature library. `http://github.com/ajt60gaibb/FastGaussQuadrature.jl`.

[25] Hale, N., and Townsend, A. Fast and accurate computation of Gauss-Legendre and Gauss-Jacobi quadrature nodes and weights. *SIAM Journal on Scientific Computing 35* (2013), A652–A674.

[26] Hale, N., and Townsend, A. A fast, simple and stable Chebyshev-Legendre transform using an asymptotic formula. *SIAM Journal on Scientific Computing 36* (2014), A148–A167.

[27] Hale, N., and Townsend, A. A fast FFT-based discrete Legendre transform. *IMA Journal of Numerical Analysis 36* (2016), 1670–1684.

[28] Keiner, J. *Fast Polynomial Transforms.* Logos Verlag, 2011.

[29] Li, Y., and Yang, H. Interpolative butterfly factorization. *SIAM Journal on Scientific Computing 39*, 2 (2017), A503–A531.

[30] Li, Y., Yang, H., Martin, E., Ho, K. L., and Ying, L. Butterfly factorization. *SIAM Journal on Multiscale Modeling and Simulation 13* (2015), 714–732.

[31] Li, Y., Yang, H., and Ying, L. Multidimensional butterfly factorization. *Applied and Computational Harmonic Analysis 44*, 3 (2018), 737 – 758.

[32] Merkle, M. Completely monotone functions: a digest. In *Analytic Number Theory, Approximation Theory, and Special Functions.* Springer, New York, NY, 2014.

[33] O'Neil, M., Woolfe, F., and Rokhlin, V. An algorithm for the rapid evaluation of special function transforms. *Applied and Computational Harmonic Analysis 28* (2010), 203–226.

[34] Ruiz-Antolín, D., and Townsend, A. A nonuniform fast Fourier transform based on low rank approximation. *SIAM Journal on Scientific Computing 40* (2018), A529–A547.

[35] Slevinsky, M. Fast Transforms library. `https://github.com/MikaelSlevinsky/FastTransforms.jl`.

[36] Slevinsky, R. On the use of Hahn's asymptotic formula and stabilized recurrence for a fast, simple and stable Chebyshev-Jacobi transform. *IMA Journal of Numerical Analysis 38* (2018), 102–124.

[37] Szegő, G. *Orthogonal Polynomials*. American Mathematical Society, Providence, Rhode Island, 1959.

[38] Trefethen, N. *Approximation Theory and Approximation Practice*. Society for Industrial and Applied Mathematics, 2013.

[39] Watson, G. N. *A Treatise on the Theory of Bessel Functions*, second ed. Cambridge University Press, New York, 1995.

[40] Yang, H. A unified framework for oscillatory integral transform: When to use NUFFT or butterfly factorization? *preprint* (2018).

[41] Zettl, A. *Sturm-Liouville Theory*. American Mathematical Society, 2005.