A new class of analysis-based fast transforms

Michael O'Neil and Vladimir Rokhlin

Technical Report YALEU/DCS/TR-1384
August 6, 2007

We introduce a new approach to the rapid numerical application to arbitrary vectors of certain types of linear operators. *Inter alia*, our scheme is applicable to many classical integral transforms, and to the expansions associated with most families of classical special functions; among the latter are Bessel functions, Legendre, Hermite, and Laguerre polynomials, Spherical Harmonics, Prolate Spheroidal Wave functions, and a number of others. In all these cases, the CPU time requirements of our algorithm are of the order $\mathcal{O}(n \log n)$, where n is the size of the matrix to be applied. The performance of our algorithm is illustrated via a number of numerical examples.

A new class of analysis-based fast transforms

Michael O'Neil and Vladimir Rokhlin
Technical Report YALEU/DCS/TR-1384
August 6, 2007

The authors were supported in part by ONR Grant #N00014-07-1-0711, AFOSR Grants #FA9550-06-1-0197, #FA9550-06-1-0239, and #FA9550-05-C-0064, and NGA Grants #HM1582-06-1-2037 and #HM1582-06-1-2039.
Approved for public release: Distribution is unlimited.
Keywords: *fast, transform, algorithm, matrix, special functions*

1 Introduction

Transforms associated with classical special functions occur in a wide variety of fields. Examples of such transforms are Fourier-Bessel (Hankel) transforms, orthogonal polynomial transforms, Fourier transforms, etc. They are encountered in geophysical modeling, submarine warfare, quantum mechanical calculations, simulation of radar phenomena, etc. For the purposes of this paper, we define special functions as the eigenfunctions on some (finite or infinite) interval $[a, b]$ of the differential equation

$$\frac{d}{dx}\left(p(x)\,\frac{d\varphi}{dx}\right) + \left(r(x) + \lambda\, w(x)\right)\,\varphi = 0, \tag{1.1}$$

subject to appropriate boundary conditions at the points a and b. The class of special functions in question is defined by the interval $[a, b]$, the functions $w, p, r : [a, b] \to \mathbb{R}$, and the boundary conditions; the function w is assumed to be strictly positive. We will be denoting the n^{th} eigenfunction of (1.1) by φ_n, and the corresponding eigenvalue by λ_n. As is well-known (for example, see [2]), the functions $\varphi_1, \varphi_2, \varphi_3, \ldots$ constitute an orthogonal basis in $L^2[a, b]$ with respect to the inner product defined by the weight function w. Thus, for any square integrable function $f : [a, b] \to \mathbb{R}$ there exists a sequence of real numbers $\alpha_1, \alpha_2, \alpha_3, \ldots$ such that

$$f(x) = \sum_{j=1}^{\infty} \alpha_j\, \varphi_j(x), \tag{1.2}$$

with

$$\alpha_j = \frac{1}{\|\varphi_j\|_w} \int_a^b f(x)\, \varphi_j(x)\, w(x)\, dx, \tag{1.3}$$

and

$$\|\varphi_j\|_w = \int_a^b \varphi_j^2(x)\, w(x)\, dx. \tag{1.4}$$

Given a collection of n points $x_1, x_2, \ldots, x_n$ on $[a, b]$, it is often desirable to determine coefficients $\beta_1, \beta_2, \ldots, \beta_n$ such that for $k = 1, 2, \ldots, n$

$$f(x_k) = \sum_{j=1}^{n} \beta_j\, \varphi_j(x_k). \tag{1.5}$$

It is possible to choose $x_1, x_2, \ldots, x_n$ such that the linear mapping $A : \mathbb{R}^n \to \mathbb{R}^n$,

$$A\left(f(x_1), f(x_2), \ldots, f(x_n)\right) = (\beta_1, \beta_2, \ldots, \beta_n) \tag{1.6}$$

is well-conditioned. The mapping in equation (1.6) is given in matrix form as

$$\begin{pmatrix} w_1\,\varphi_1(x_1) & w_2\,\varphi_1(x_2) & \cdots & w_n\,\varphi_1(x_n) \\ w_1\,\varphi_2(x_1) & w_2\,\varphi_2(x_2) & & \vdots \\ \vdots & & \ddots & \vdots \\ w_1\,\varphi_n(x_1) & \cdots & \cdots & w_n\,\varphi_n(x_n) \end{pmatrix} \begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{pmatrix} = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{pmatrix}, \tag{1.7}$$

where $w_1, w_2, \ldots, w_n$ are determined from the weight function w and the norm of the functions $\varphi_1, \varphi_2, \ldots, \varphi_n$. With the proper choice of n and $x_1, x_2, \ldots, x_n$, not only is A well-conditioned, but the function $g : [a, b] \to \mathbb{R}$, given by the formula

$$g(x) = \sum_{j=1}^{n} \beta_j\, \varphi_j(x), \tag{1.8}$$

provides an accurate and stable approximation to f for all $x \in [a,b]$. Additionally, since A is well-conditioned, so is its inverse A^{-1}, making both matrices suitable for numerics. The direct application of A and A^{-1} each require $\mathcal{O}(n^2)$ operations, while the algorithm presented in this paper enables the numerical application of A and A^{-1} in only $\mathcal{O}(n \log n)$ operations. Our algorithm enables an accelerated numerical summation of the functions $\varphi_j(x_k)$ over j or k. For reasons described later in the paper, we will refer to our procedure as the "butterfly" algorithm.

Another class of transforms common in mathematical physics and classical mathematics are integral transforms. An integral transform $K : L^2[a,b] \to L^2[c,d]$ maps functions on one interval, $[a,b]$, to functions on another interval, $[c,d]$. For square integrable functions $f : [a,b] \to \mathbb{C}$ and $k : [a,b] \times [c,d] \to \mathbb{C}$, the operator K takes the form

$$(Kf)(y) = \int_a^b k(x,y)\, f(x)\, dx. \tag{1.9}$$

The function k is referred to as the kernel of the integral transform. One may desire to evaluate the function Kf at n values $y_1, y_2, \ldots, y_n$. Standard methods for the numerical application of integral operators replace the integral in equation (1.9) with a quadrature consisting of appropriately chosen points $x_1, \ldots, x_m$ and weights $w_1, \ldots, w_m$ such that for some class of appropriately chosen functions f,

$$\sum_{i=1}^{m} w_i\, k(x_i, y_j)\, f(x_i) \approx \int_a^b k(x, y_j)\, f(x)\, dx \tag{1.10}$$

for $j = 1, 2, \ldots, n$. The convergence of the approximation is consistent with the order of the quadrature chosen. The computation of the integral in equation (1.9) for $y_1, y_2, \ldots, y_n$ has thus been reduced to a matrix-vector multiplication. For certain choices of the kernel k, we are able to accelerate this multiplication using the algorithm of this paper.

One such integral transform is the Fourier-Bessel transform. Fourier-Bessel transforms arise when computing the Fourier transform of a radially symmetric function on the disk. This occurs commonly in the fields of optics, acoustics, geophysics, etc. Let R be a positive real number and ν a positive integer. Suppose that we have a function $f : \mathbb{R}^2 \to \mathbb{C}$, given by the formula

$$f(x,y) = g(r)\, e^{i\nu\theta}, \tag{1.11}$$

where (r,θ) are the usual polar coordinates, and the function $g : \mathbb{R}^+ \to \mathbb{R}$ is such that $g(r) = 0$ for $r > R$. After a change of variables, the Cartesian Fourier transform

$$(\mathcal{F}f)(u,v) = \iint\limits_{\mathbb{R}^2} e^{i(ux+vy)}\, f(x,y)\, dx\, dy \tag{1.12}$$

becomes

$$(\mathcal{F}f)(\rho,\psi) = 2\pi\, i^\nu\, e^{i\nu\psi} \int_0^R J_\nu(\rho r)\, g(r)\, r\, dr, \tag{1.13}$$

where

$$\begin{aligned} u &= \rho\, \cos\psi, & x &= r\, \cos\theta, \\ v &= \rho\, \sin\psi, & y &= r\, \sin\theta. \end{aligned} \tag{1.14}$$

The integral in equation (1.13) is referred to as the Fourier-Bessel (or Hankel) transform of order n of the function g.

If we wish to evaluate the integral in equation (1.13) for n values of ρ, $\rho_1, \rho_2, \ldots, \rho_n$, we may discretize the integral and use the algorithm of this paper to apply the resulting matrix. For reasons described in Section 2.2, $\rho_1, \rho_2, \ldots, \rho_n$ are usually chosen to be the first n positive roots of J_ν. Previous methods for numerically computing Fourier-Bessel transforms include using an exponential change of variables, asymptotic expansions of Bessel functions, and projection techniques (see [5], [12], and [17]). These algorithms often had limitations on the choices for ρ in equation (1.13) (due to the requirement of equispaced points for FFTs), the requirement of sampling g on an exponential grid, and on the order ν of the transform. In this paper we eliminate all such restrictions.

It should be pointed out that the algorithm of this paper is very similar to that of [16], and has been motivated by the latter.

The paper has the following format: Section 2 gives some background information and lemmas concerning certain special functions, and the approximation of low-rank matrices. In Section 3 we build the analytical apparatus to be used in the construction of the algorithm of this paper. In Section 4 we describe our algorithm. Section 5 contains numerical examples illustrating the performance of the algorithm, and Section 6 summarizes the work and discusses its possible extensions.

2 Mathematical and numerical preliminaries

This section contains definitions, lemmas, and basic mathematical facts that are used in the remainder of the paper. For any positive integer n, and column vector $v \in \mathbb{C}^n$, we define the norm $\|v\|$ of v to be the root-sum-square (l^2 norm) of the entries of v, that is,

$$\|v\| = \sqrt{\sum_{j=1}^{n} |v_j|^2}, \tag{2.1}$$

where v_j is the j^{th} entry of v. For any positive integer m and matrix $A \in \mathbb{C}^{m \times n}$, we define the norm $\|A\|$ of A to be the spectral (l^2-operator) norm of A,

$$\|A\| = \max_{x \in \mathbb{C}^n} \frac{\|Ax\|}{\|x\|} = \max_{1 \le j \le \min(m,n)} |\sigma_j| \tag{2.2}$$

where $\sigma_1, \sigma_2, \ldots, \sigma_{\min(m,n)}$ are the singular values of A. (Obviously, the norm of v as viewed as an $n \times 1$ matrix is equal to the norm of v as viewed as an $n \times 1$ column vector.) The transpose of matrix A will be denoted by A^t. Unless otherwise stated, we denote the natural logarithm of $x \in \mathbb{R}^+$ by $\log x$. The largest integer that is less than or equal to x will be denoted by $\lfloor x \rfloor$, i.e. $\lfloor x \rfloor$ is the floor of x. We use $L^2[a,b]$ to denote the space of square integrable functions $f : [a,b] \to \mathbb{C}$,

$$L^2[a,b] = \left\{ f : \int_a^b |f(x)|^2 \, dx < \infty \right\}. \tag{2.3}$$

2.1 Approximation of low-rank matrices

The principal numerical tool used in this paper is the approximation of a series of low-rank matrices via interpolative decompositions. Lemma 2.1 below restates (in a slightly different form) Theorem 3 in [4]. It states that for any $m \times n$ matrix A whose approximate rank is k, there exist an $m \times k$ matrix B whose columns constitute a subset of the columns of A, and a $k \times n$ matrix P, such that

1. some subset of the columns of P makes up the $k \times k$ identity matrix,

2. P is not too large, and
3. $B\,P - A$ is small.

Lemma 2.1 *Suppose that m and n are positive integers, and A is a real $m \times n$ matrix. Then, for any positive integer k with $k \le m$ and $k \le n$, there exist a real $k \times n$ matrix P, and a real $m \times k$ matrix B whose columns constitute a subset of the columns of A, such that*

1. *some subset of the columns of P makes up the $k \times k$ identity matrix,*
2. *no entry of P has an absolute value greater than 1,*
3. $\|P\| \le \sqrt{k\,(n-k)+1}$,
4. *the least (that is, the k^{th} greatest) singular value of P is at least 1,*
5. *$B\,P = A$ when $k = m$ or $k = n$, and*
6. *$\|B\,P - A\| \le \sqrt{k\,(n-k)+1}\;\sigma_{k+1}$ when $k < m$ and $k < n$, where σ_{k+1} is the $(k+1)^{st}$ greatest singular value of A.*

Remark 2.2 Properties 1, 2, 3, and 4 in Lemma 2.1 ensure that the interpolative decomposition $B\,P$ of A is numerically stable. It should also be observed that Property 3 follows directly from Properties 1 and 2, and Property 4 follows directly from Property 1.

Observation 2.3 Existing algorithms for the construction of the matrices B and P in Lemma 2.1 are computationally expensive. We use an algorithm to produce B and P which satisfy conditions slightly more relaxed than those in Lemma 2.1, but still sufficient for numerical purposes. We compute B and P such that

1. some subset of the columns of P makes up the $k \times k$ identity matrix,
2. no entry of P has an absolute value greater than 2,
3. $\|P\| \le \sqrt{4k\,(n-k)+1}$,
4. the least (that is, the k^{th} greatest) singular value of P is at least 1,
5. $B\,P = A$ when $k = m$ or $k = n$, and
6. $\|B\,P - A\| \le \sqrt{4k\,(n-k)+1}\;\sigma_{k+1}$ when $k < m$ and $k < n$, where σ_{k+1} is the $(k+1)^{\mathrm{st}}$ greatest singular value of A.

Thus, for any positive real number ε, the algorithm can identify the least k such that $\|B\,P - A\| \approx \varepsilon$. Furthermore, there exists a real number C such that the algorithm computes both B and P using at most $Ckmn\,\log(n)$ floating-point operations and $Ckmn$ floating-point words of memory (see [4], [11], and [15]).

2.2 Bessel functions

This section contains a brief summary of various facts regarding Bessel functions, and defines the Fourier-Bessel transform (see [1] and [23]). For any integer $\nu \geq 0$, we denote by J_ν the Bessel function of the first kind of order ν, Y_ν the Bessel function of the second kind of order ν, and $H_\nu^{(1,2)}$ the Hankel function of the first and second kinds of order ν. For any $z \in \mathbb{C}$,

$$H_\nu^{(1)}(z) = J_\nu(z) + iY_\nu(z), \tag{2.4}$$

$$H_\nu^{(2)}(z) = J_\nu(z) - iY_\nu(z). \tag{2.5}$$

The functions J_ν and Y_ν are two linearly independent solutions to the differential equation

$$\frac{d}{dz}\left(z\frac{dw}{dz}\right) + \left(z - \frac{\nu^2}{z}\right) w = 0, \tag{2.6}$$

given by the formulas

$$J_\nu(z) = \frac{1}{\pi}\int_0^\pi \cos(z\,\sin\theta - \nu\theta)\,d\theta, \tag{2.7}$$

$$Y_\nu(z) = \frac{1}{\pi}\int_0^\pi \sin(z\,\sin\theta - \nu\theta)\,d\theta - \frac{1}{\pi}\int_0^\infty (e^{\nu t} + e^{-\nu t}\,\cos\nu\pi)\,e^{-z\,\sinh t}\,dt, \tag{2.8}$$

where we require in (2.8) that $\mathrm{Re}(z) > 0$. The functions $J_0, J_1, J_2, \dots$ are known to satisfy the recurrence

$$J_{\nu+1}(z) = \frac{2\nu}{z}J_\nu(z) - J_{\nu-1}(z). \tag{2.9}$$

Bessel functions of the second kind (and thus Hankel functions of the first and second kind) satisfy the same recurrence as Bessel functions of the first kind, as in (2.9).

Remark 2.4 Recurrence (2.9) can be used for the numerical evaluation of Bessel functions. For $x \in \mathbb{R}^+$, given initial values $Y_0(x)$ and $Y_1(x)$, the recurrence is stable and can be used to accurately calculate $Y_2(x), Y_3(x), \dots$. On the other hand, given initial conditions $J_0(x)$ and $J_1(x)$, recurrence (2.9) is unstable. However, as is well known, given initial conditions $J_N(x)$ and $J_{N+1}(x)$ for some large N, the backwards recurrence

$$J_{\nu-1}(z) = \frac{2\nu}{z}J_\nu(z) - J_{\nu+1}(z) \tag{2.10}$$

is stable and can be used to calculate $J_{N-1}(x), \dots, J_1(x), J_0(x)$ (see, for example, [1]).

For a fixed order ν, we use the method found in [9] to numerically integrate differential equation (2.6) to evaluate Bessel functions of the first kind. For any $\lambda \in \mathbb{C}$, after a change of variable equation (2.6) becomes

$$\frac{d^2w}{dz^2} + \left(\lambda^2 - \frac{\nu^2 - \frac{1}{4}}{z^2}\right) w = 0, \tag{2.11}$$

which has linearly independent solutions

$$\tilde{J}_\nu(z) = \sqrt{z}\,J_\nu(\lambda z), \tag{2.12}$$

$$\tilde{Y}_\nu(z) = \sqrt{z}\,Y_\nu(\lambda z). \tag{2.13}$$

It turns out that for any $\nu \geq 0$, and any subinterval $[0, R] \subset \mathbb{R}^+$, there exists a choice of $\lambda_1, \lambda_2, \lambda_3, \ldots$ such that

$$\int_0^R r\, J_\nu(\lambda_i r)\, J_\nu(\lambda_j r)\, dr = 0 \tag{2.14}$$

for $i \neq j$. Indeed, for any $R > 0$ and integer $\nu \geq 0$, let the real numbers $\rho_1 < \rho_2 < \rho_3 < \ldots$ be all the positive zeros of the function $g : \mathbb{R}^+ \to \mathbb{R}$, given by the formula

$$g(\rho) = J_\nu(2\pi\rho R). \tag{2.15}$$

Let the real valued functions $\bar{J}_{\nu,1}, \bar{J}_{\nu,2}, \bar{J}_{\nu,3}, \ldots$ on $[0, R]$ be given by the formula

$$\bar{J}_{\nu,k}(r) = \frac{\sqrt{2}}{R} \frac{\sqrt{r}\; J_\nu(2\pi\rho_k r)}{J_{\nu+1}(2\pi\rho_k R)}. \tag{2.16}$$

It is easily verified (see [23]) that the functions $\bar{J}_{\nu,1}, \bar{J}_{\nu,2}, \bar{J}_{\nu,3}, \ldots$ are dense in $L^2[0, R]$ and orthonormal, i.e. for any $j, k > 0$

$$\int_0^R \bar{J}_{\nu,j}(r)\, \bar{J}_{\nu,k}(r)\, dr = \delta_{jk}, \tag{2.17}$$

where δ_{jk} is the Dirac delta function. To numerically calculate the roots $\rho_1, \rho_2, \rho_3, \ldots$, we use the method found in [9]. Since the functions $\bar{J}_{\nu,1}, \bar{J}_{\nu,2}, \bar{J}_{\nu,3}, \ldots$ are dense, we may express any square integrable function $f : [0, R] \to \mathbb{R}$ in the form

$$f(r) = \sum_{k=1}^{\infty} \beta_k^\nu\, \bar{J}_{\nu,k}(r), \tag{2.18}$$

where the coefficients $\beta_1^\nu, \beta_2^\nu, \beta_3^\nu, \ldots$ are given by the formula

$$\begin{aligned} \beta_k^\nu &= \int_0^R \bar{J}_{\nu,k}(r)\, f(r)\, dr \\ &= \frac{\sqrt{2}}{R} \frac{1}{J_{\nu+1}(2\pi\rho_k R)} \int_0^R \sqrt{r}\, J_\nu(2\pi\rho_k r)\, f(r)\, dr. \end{aligned} \tag{2.19}$$

The number β_k^ν is known as the kth Fourier-Bessel coefficient of order ν of the function f. We call the mapping $A_n^\nu : L^2[0, R] \to \mathbb{R}^n$,

$$A_n^\nu(f) = (\beta_1^\nu, \beta_2^\nu, \ldots, \beta_n^\nu), \tag{2.20}$$

the Fourier-Bessel transform of order ν and size n. To calculate the Fourier-Bessel transform numerically, it is necessary to evaluate the integral in (2.19) for $k = 1, 2, \ldots, n$. We numerically calculate this integral using an $(n+2)$-point trapezoidal quadrature. Setting $h = \frac{R}{n+1}$, trapezoidal integration gives

$$\begin{aligned} \beta_k^\nu &\approx \frac{\sqrt{2}}{R} \frac{1}{J_{\nu+1}(2\pi\rho_k R)}\, h \left(\sum_{j=0}^{n+1} \sqrt{jh}\, J_\nu(2\pi\rho_k jh)\, f(jh) - \frac{1}{2}\sqrt{R}\, J_\nu(2\pi\rho_k R)\, f(R) \right) \\ &= \frac{\sqrt{2}}{R} \frac{1}{J_{\nu+1}(2\pi\rho_k R)}\, h \sum_{j=1}^{n} \sqrt{jh}\, J_\nu(2\pi\rho_k jh)\, f(jh), \end{aligned} \tag{2.21}$$

since for all $k > 0$

$$J_\nu(2\pi\rho_k R) = 0. \tag{2.22}$$

In matrix notation,

$$\beta \approx h \frac{\sqrt{2}}{R} S_{J_\nu} T_{J_\nu} f, \tag{2.23}$$

where

$$\beta = (\beta_1^\nu, \beta_2^\nu, \dots, \beta_n^\nu)^t, \tag{2.24}$$

$$f = (f(h), f(2h), \dots, f(nh))^t, \tag{2.25}$$

and

$$S_{J_\nu} = \begin{pmatrix} \frac{1}{J_{\nu+1}(2\pi\rho_1 R)} & & & \\ & \frac{1}{J_{\nu+1}(2\pi\rho_2 R)} & & \\ & & \ddots & \\ & & & \frac{1}{J_{\nu+1}(2\pi\rho_n R)} \end{pmatrix}, \tag{2.26}$$

$$T_{J_\nu} = \begin{pmatrix} \sqrt{h}\, J_\nu(2\pi\rho_1 h) & \cdots & \sqrt{nh}\, J_\nu(2\pi\rho_1 nh) \\ \vdots & \ddots & \vdots \\ \sqrt{h}\, J_\nu(2\pi\rho_n h) & \cdots & \sqrt{nh}\, J_\nu(2\pi\rho_n nh) \end{pmatrix}. \tag{2.27}$$

The algorithm we describe in Section 4 accelerates the application of the matrix T_{J_ν} in equation (2.27).

Remark 2.5 If we contrast equation (2.19) with equation (1.13), it appears as though the Fourier-Bessel transform has been defined with two different kernels; the integrand in equation (2.19) contains a $\sqrt{r}$ and the integrand in equation (1.13) contains an r. These are merely two different conventions, and the difference will not affect the speed with which the algorithm of Section 4 applies the matrix resulting from their discretization. Multiplication (or division) by an extra $\sqrt{r}$ in the integrand is a well-conditioned diagonal transformation and has a negligible effect on the numerical rank of any submatrix.

In addition to computing Fourier-Bessel coefficients, one may wish to perform other numerical calculations involving Bessel functions, including evaluating sums of Bessel and Hankel functions over varying order. These expansions are analogous to expansions in orthogonal polynomials (see [21] and [22]). For $x \in \mathbb{R}$ and complex $\alpha_0, \alpha_1, \dots, \alpha_{n-1}$, it may be desirable to evaluate the sums

$$g(x) = \sum_{k=0}^{n-1} \alpha_k J_k(x) \tag{2.28}$$

and

$$w(x) = \sum_{k=0}^{n-1} \alpha_k H_k^{(1,2)}(x). \tag{2.29}$$

To evaluate the function g at real points $x_1, x_2, \dots, x_n$ we must apply the matrix E_J,

$$E_J = \begin{pmatrix} J_0(x_1) & \cdots & J_{n-1}(x_1) \\ \vdots & \ddots & \vdots \\ J_0(x_n) & \cdots & J_{n-1}(x_n) \end{pmatrix}, \tag{2.30}$$

to the vector $(\alpha_0, \alpha_1, \dots, \alpha_{n-1})^t$. To evaluate the function w at real points $x_1, x_2, \dots, x_n$ we must apply the matrix E_H,

$$E_H = \begin{pmatrix} H_0^{(1,2)}(x_1) & \cdots & H_{n-1}^{(1,2)}(x_1) \\ \vdots & \ddots & \vdots \\ H_0^{(1,2)}(x_n) & \cdots & H_{n-1}^{(1,2)}(x_n) \end{pmatrix}, \tag{2.31}$$

to the vector $(\alpha_0, \alpha_1, \ldots, \alpha_{n-1})^t$. It will be shown in Section 5 that the algorithm of this paper will accelerate the application of matrices E_J and E_H.

For fixed argument and large order, Bessel functions of the first kind have a predictable behavior, used in proofs contained in Section 3. Lemma 2.6 below shows that for $R > 10$ and large ν, $J_\nu(r)$ will be very small for $0 \le r \le R$. It is reproduced from [18].

Lemma 2.6 *For any real $R > 10$ and $0 < \varepsilon < 0.1$,*

$$J_\nu(R) < \varepsilon \tag{2.32}$$

for any

$$\nu > (R^{1/3} + \alpha\, R^{-1/3})^3, \tag{2.33}$$

where

$$\alpha = \frac{3^{-1/3}}{2} \log^{2/3}\left(\frac{1}{\varepsilon}\right). \tag{2.34}$$

Thus, $J_\nu(r) < \varepsilon$ for any $0 \le r \le R$.

Remark 2.7 The following theorem provides an error estimate for the approximation of an integral by the n-point trapezoidal rule (see [6]), and is a motivation for the popularity of trapezoidal integration in such environments.

Theorem 2.8 (Euler-Maclaurin Summation Formula) *For any $n > 1$, and function $\varphi : [a, b] \to \mathbb{R}$, let*

$$h = \frac{b-a}{n-1}, \tag{2.35}$$

and

$$\hat{T}_n(\varphi) = h \left(\sum_{j=0}^{n-1} \varphi(a + jh) - \frac{\varphi(a) + \varphi(b)}{2} \right). \tag{2.36}$$

If the function φ has $2k+2$ continuous derivatives, then for some $\xi \in [a, b]$, the error of the n-point trapezoidal quadrature rule $\hat{T}_n$ for the integration of φ on $[a, b]$ is given by the formula

$$\hat{T}_n(\varphi) - \int_a^b \varphi(x)\, dx = \sum_{l=1}^{k} \frac{h^{2l}\, B_{2l}}{(2l)!} (\varphi^{2l-1}(b) - \varphi^{2l-1}(a)) + \frac{h^{2k+2}\, B_{2k+2}}{(2k+2)!} \varphi^{2k+2}(\xi), \tag{2.37}$$

where B_i is the i^{th} Bernoulli number.

In many cases, the function $f : [0, R] \to \mathbb{R}$ whose Fourier-Bessel transform is to be taken is zero at the point $x = R$, along with many of its derivatives. Furthermore, for any integer $\nu \ge 2$, it follows from the formula (see [10])

$$J_\nu(x) = \left(\frac{x}{2}\right)^\nu \sum_{k=0}^{\infty} \frac{(-1)^k}{k!\, \Gamma(\nu + k + 1)} \left(\frac{x}{2}\right)^{2k} \tag{2.38}$$

that the function J_ν is not only zero at the point $x = 0$, its first $\nu - 1$ derivatives are also zero at the point $x = 0$. These two observations, when combined with Theorem 2.8, show that the trapezoidal rule converges rapidly when used for the approximation of Fourier-Bessel coefficients.

2.3 Fourier series

The following information concerning complex exponentials can be found in any standard Fourier analysis textbook (see [14], for example). We also describe discrete Fourier transforms for nonequispaced data, and refer the reader to [7] and [8] for more information.

A continuous periodic function $f : [-\pi, \pi] \to \mathbb{C}$ can be represented by its Fourier series as

$$f(x) = \sum_{k=-\infty}^{\infty} c_k \, e^{ikx}, \tag{2.39}$$

where c_k is given by the formula

$$c_k = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x) \, e^{-ikx} \, dx. \tag{2.40}$$

Without loss of generality, assume for the moment that $n > 0$ is an even integer. If f is supplied as a discrete function, known only at n equispaced points $x_1, x_2, \ldots, x_n$, $x_j = -\pi + \frac{\pi(j-1)}{n}$, then we can represent f at these points using the formula

$$f(x_j) = \sum_{k=1}^{n} \alpha_k \, e^{i\omega_k x_j}, \tag{2.41}$$

where $\omega_k = \frac{-n}{2} + k - 1$ and α_k is given by

$$\alpha_k = \frac{1}{n} \sum_{j=1}^{n} f(x_j) \, e^{-i\omega_k x_j}. \tag{2.42}$$

We refer to the linear operator $A_F : \mathbb{R}^n \to \mathbb{R}^n$,

$$A_F(f(x_1), f(x_2), \ldots, f(x_n)) = (\alpha_1, \alpha_2, \ldots, \alpha_n), \tag{2.43}$$

as the forward discrete Fourier transform. Determining the values of f at $x_1, x_2, \ldots, x_n$ given the coefficients $\alpha_1, \alpha_2, \ldots, \alpha_n$ will be referred to as the inverse discrete Fourier transform. The Fast Fourier Transform (FFT) algorithm for computing the forward and inverse transforms is widely known (see, for example [6]). Note that the FFT, as well as equation (2.42), relies on the fact that $x_1, x_2, \ldots, x_n$ are equispaced and $\omega_1, \omega_2, \ldots, \omega_n$ are integers ranging from $\frac{-n}{2}$ to $\frac{n}{2} - 1$.

Suppose now that we wish to evaluate sums of the form

$$\alpha_k = \sum_{j=1}^{n} f(x_j) \, e^{-i\omega_k x_j} \tag{2.44}$$

where $x_1, x_2, \ldots, x_n$ is an arbitrary set of n real numbers on $[-\pi, \pi]$ and $\omega_1, \omega_2, \ldots, \omega_n$ is an arbitrary set of n real numbers on $[\frac{-n}{2}, \frac{n}{2}]$, neither of which is equispaced. Sums of this type are known as discrete Fourier transforms for nonequispaced data. The algorithm developed in Section 4 can be used to calculate discrete Fourier transforms for nonequispaced data, as well as equispaced data. For a given n, this requires that we apply the matrix T_F,

$$T_F = \begin{pmatrix} e^{-i\omega_1 x_1} & \cdots & e^{-i\omega_1 x_n} \\ \vdots & \ddots & \vdots \\ e^{-i\omega_n x_1} & \cdots & e^{-i\omega_n x_n} \end{pmatrix}, \tag{2.45}$$

to the vector $(f(x_1), f(x_2), \ldots, f(x_n))^t$, where $\omega_1, \omega_2, \ldots, \omega_n$ and $x_1, x_2, \ldots, x_n$ are given (equispaced or nonequispaced). As we show in Section 5, the application of the matrix T_F will be accelerated by the algorithm of this paper.

2.4 Generalized Gaussian quadratures

This section contains information concerning generalized Gaussian quadratures (see [13] and [25] for more detailed information). Given two functions $f : [a,b] \to \mathbb{R}$ and $w : [a,b] \to \mathbb{R}^+$, we may wish to approximate the integral

$$\int_a^b f(x)\, w(x)\, dx \tag{2.46}$$

by a sum. An n-point quadrature rule for the approximation of integral (2.46) consists of nodes $x_1, x_2, \ldots, x_n \in [a,b]$, weights $w_1, w_2, \ldots, w_n > 0$, and is given by the sum

$$\sum_{j=1}^{n} w_j\, f(x_j). \tag{2.47}$$

For example, the formula for the well known n-point trapezoidal rule is

$$\int_a^b f(x)\, dx \approx \sum_{j=0}^{n-1} h\, f(a+jh) - \frac{h}{2}(f(a)+f(b)), \tag{2.48}$$

where $h = \frac{b-a}{n-1}$. Generalized Gaussian quadratures are quadratures for which the approximation

$$\int_a^b f(x)\, w(x)\, dx \approx \sum_{j=1}^{n} w_j\, f(x_j) \tag{2.49}$$

is exact for a choice of $2n$ different functions f. We now give a more concise definition.

Definition 2.9 *An n-point quadrature is referred to as an n-point generalized Gaussian quadrature with respect to the weight function $w : [a,b] \to \mathbb{R}^+$ and functions $f_1, f_2, \ldots, f_{2n} : [a,b] \to \mathbb{R}$ if*

$$\int_a^b f_k(x)\, w(x)\, dx = \sum_{j=1}^{n} w_j\, f_k(x_j) \tag{2.50}$$

for $k = 1, 2, \ldots, 2n$. The nodes $x_1, x_2, \ldots, x_n$ are referred to as Gaussian nodes and the weights $w_1, w_2, \ldots, w_n > 0$ are referred to as Gaussian weights, both of which are unique.

Classical n-point Gaussian quadratures are those that exactly integrate polynomials up to degree $2n-1$ with respect to some weight function w. The nodes for these Gaussian quadratures turn out to be the roots of the polynomial of degree n in the class of orthogonal polynomials associated with the weight function w. Relevant orthogonal polynomials are discussed in Sections 2.5–2.8.

Table 1 lists some common weight functions, their associated orthogonal polynomials, and the quadrature weights for the classical n-point Gaussian quadrature formula

$$\int_a^b f(x)\, w(x)\, dx \approx \sum_{j=1}^{n} w_j\, f(x_j). \tag{2.51}$$

The column labeled "$[a,b]$" denotes the interval that the weight function is defined on. The column "$w(x)$" gives the formula for the weight function. The column "associated polynomials" lists the class of orthogonal polynomials associated with the given interval and weight function. The column "w_j" lists the formula for the j^{th} weight in the classical n-point Gaussian quadrature. In all cases, x_j is used to denote the j^{th} root of the associated polynomial of degree n.

$[a,b]$	$w(x)$	Associated polynomials	w_j
$[-1,1]$	1	Legendre	$2\,(1-x_j^2)^{-1}\,P_n'(x_j)^{-2}$
$[-1,1]$	$(1-x^2)^{-1/2}$	Chebyshev	$\pi\,n^{-1}$
$(-\infty,\infty)$	e^{-x^2}	Hermite	$\sqrt{\pi}\,2^{n+1}\,H_n'(x_j)^{-2}$
$[0,\infty)$	e^{-x}	Laguerre	$x_j^{-1}\,L_n'(x_j)^{-2}$

Table 1: Various weight functions, their associated orthogonal polynomials, and discrete weights for n-point Gaussian quadratures.

2.5 Legendre polynomials

This section, and Sections 2.6–2.8, contain information about classical orthogonal polynomials (see [10] and [20]). For an integer $n \geq 0$, we denote by P_n the Legendre polynomial of degree n. As is well known, P_n is the solution to the differential equation

$$\frac{d}{dx}\left((1-x^2)\,\frac{dy}{dx}\right) + n\,(n+1)\,y = 0 \tag{2.52}$$

that is bounded on the interval $[-1,1]$. Legendre polynomials are orthogonal on the interval $[-1,1]$ with respect to the inner product defined by the weight function $w(x) = 1$, i.e. for any $m, n \geq 0$,

$$\int_{-1}^{1} P_m(x)\,P_n(x)\,dx = \frac{2}{2n+1}\,\delta_{mn}. \tag{2.53}$$

Legendre polynomials satisfy the recurrence

$$(n+1)\,P_{n+1}(x) = (2n+1)\,x\,P_n(x) - n\,P_{n-1}(x) \tag{2.54}$$

with initial conditions

$$P_0(x) = 1, \tag{2.55}$$

$$P_1(x) = x. \tag{2.56}$$

Differentiating (2.54) yields a recurrence for the derivatives of Legendre polynomials, which are used to calculate Gaussian weights (see Section 2.4). The recurrence (2.54) is well known to be stable and can be used to calculate Legendre polynomials of all orders.

Since Legendre polynomials constitute an orthogonal basis in $L^2[-1,1]$, for any square integrable function $f : [-1,1] \to \mathbb{R}$ there exist real $\alpha_0, \alpha_1, \alpha_2, \dots$ such that

$$f(x) = \sum_{k=0}^{\infty} \alpha_k\,P_k(x). \tag{2.57}$$

In most numerical applications, the series in equation (2.57) is truncated after n terms, for some appropriately chosen n. The resulting finite sum is a stable interpolating polynomial of degree $n-1$, and it is commonly used for accurate numerical evaluation of f. The function f is thus approximated by a polynomial $p : [-1,1] \to \mathbb{R}$, given by the formula

$$p(x) = \sum_{k=0}^{n-1} \alpha_k\,P_k(x). \tag{2.58}$$

If f has n continuous derivatives, then for any $x \in [-1,1]$ the error of this approximation is of the form

$$|f(x) - p(x)| = \frac{2^n\, n!}{(2n)!} |f^{(n)}(\xi)|, \tag{2.59}$$

for some $\xi \in [-1,1]$. We recall from Section 2.4 that the n-point Gaussian quadrature for weight function $w(x) = 1$ on the interval $[-1,1]$ has nodes $x_1, x_2, \ldots, x_n$ equal to the roots of P_n and weights $w_1, w_2, \ldots, w_n$ given by

$$w_j = \frac{2}{(1 - x_j^2)\, P_n'(x_j)^2}. \tag{2.60}$$

Applying this quadrature to products of Legendre polynomials yields the identity

$$\sum_{j=1}^{n} w_j\, P_k(x_j)\, P_l(x_j) = \frac{2}{2k+1}\, \delta_{kl} \tag{2.61}$$

for all $0 \le k, l \le n-1$. If f is tabulated at the nodes $x_1, x_2, \ldots, x_n$, the expansion coefficients in equation (2.58) are given by the formula

$$\alpha_k = \frac{2k+1}{2} \sum_{j=1}^{n} w_j\, f(x_j)\, P_k(x_j). \tag{2.62}$$

For an integer $n > 0$, we define the linear operator $A_P : \mathbb{R}^n \to \mathbb{R}^n$,

$$A_P\left(f(x_1), f(x_2), \ldots, f(x_n)\right) = (\alpha_0, \alpha_1, \ldots, \alpha_{n-1}), \tag{2.63}$$

as the Legendre transform of size n. A_P converts values of f at the roots of P_n into coefficients in a Legendre polynomial expansion via equation (2.62). In matrix notation,

$$A_P = S_P\, T_P\, W_P, \tag{2.64}$$

where

$$S_P = \begin{pmatrix} \frac{1}{2} & & & \\ & \frac{3}{2} & & \\ & & \ddots & \\ & & & \frac{2n-1}{2} \end{pmatrix}, \tag{2.65}$$

$$T_P = \begin{pmatrix} P_0(x_1) & \cdots & P_0(x_n) \\ \vdots & \ddots & \vdots \\ P_{n-1}(x_1) & \cdots & P_{n-1}(x_n) \end{pmatrix}, \tag{2.66}$$

$$W_P = \begin{pmatrix} w_1 & & \\ & \ddots & \\ & & w_n \end{pmatrix}. \tag{2.67}$$

The algorithm presented in Section 4 accelerates the application of the matrix T_P, and thus accelerates the evaluation of Legendre polynomial expansion coefficients.

2.6 Chebyshev polynomials

This section contains facts concerning Chebyshev polynomials that will be used in Sections 3 and 5. For an integer $n \geq 0$, we denote by T_n the Chebyshev polynomial of degree n. T_n is the solution on $[-1, 1]$ to the differential equation

$$\frac{d}{dx}\left(\sqrt{1-x^2}\,\frac{dy}{dx}\right) + \frac{n^2}{\sqrt{1-x^2}}\,y = 0, \tag{2.68}$$

and is given explicitly by the formula

$$T_n(x) = \cos(n \arccos x). \tag{2.69}$$

Chebyshev polynomials are orthogonal on the interval $[-1, 1]$ with respect to the weight function $w(x) = (1-x^2)^{-1/2}$, i.e. for any $m, n \geq 0$,

$$\int_{-1}^{1} T_m(x)\,T_n(x)\,\frac{dx}{\sqrt{1-x^2}} = \left\{ \begin{matrix} 0 & , & \text{if } m \neq n \\ \frac{\pi}{2} & , & \text{if } m = n \neq 0 \\ \pi & , & \text{if } m = n = 0 \end{matrix} \right\}. \tag{2.70}$$

Chebyshev polynomials also satisfy the recurrence

$$T_{n+1}(x) = 2\,x\,T_n(x) - T_{n-1}(x) \tag{2.71}$$

with initial conditions

$$T_0(x) = 1, \tag{2.72}$$

$$T_1(x) = t. \tag{2.73}$$

Recurrence (2.71) is stable due to the fact that $|T_n| \leq 1$ for all $n \geq 0$, and can thus be used to generate Chebyshev polynomials numerically.

Analogous to Legendre polynomials, Chebyshev polynomials are dense in $L^2[-1, 1]$. Thus, for any square integrable function $f : [-1, 1] \to \mathbb{R}$, there exist real $\alpha_0, \alpha_1, \alpha_2, \ldots$ such that

$$f(x) = \sum_{k=0}^{\infty} \alpha_k\,T_k(x). \tag{2.74}$$

In most numerical applications, we approximate the function f using a polynomial $p : [-1, 1] \to \mathbb{R}$ constructed from a truncated version the sum in (2.74), such that

$$f(x) \approx p(x) = \sum_{k=0}^{n-1} \alpha_k\,T_k(x). \tag{2.75}$$

If f has n continuous derivatives, then for any $x \in [-1, 1]$ the error of approximation (2.75) is given by the formula

$$|f(x) - p(x)| = \frac{f^{(n)}(\xi)}{2^{n-1}\,n!}, \tag{2.76}$$

for some $\xi \in [-1, 1]$. We recall from Section 2.4 that the n-point Gaussian quadrature for weight function $w(x) = (1-x^2)^{-1/2}$ on the interval $[-1, 1]$ has nodes $x_1, x_2, \ldots, x_n$ equal to the roots of T_n, given explicitly by

$$x_j = \cos\frac{(2j-1)\pi}{2n}, \tag{2.77}$$

and weights $w_1, w_2, \ldots, w_n$ given explicitly by

$$w_j = \frac{\pi}{n}. \tag{2.78}$$

Applying this quadrature to products of Chebyshev polynomials yields the identities

$$\sum_{j=1}^{n} w_j \, T_0(x_j)^2 = \pi, \tag{2.79}$$

$$\sum_{j=1}^{n} w_j \, T_k(x_j) \, T_l(x_j) = \frac{\pi}{2} \, \delta_{kl}, \tag{2.80}$$

for all $1 \le k, l \le n-1$. If f is tabulated at the points $x_1, x_2, \ldots, x_n$, the expansion coefficients in equation (2.75) are given by the formulas

$$\alpha_0 = \frac{1}{n} \sum_{j=1}^{n} f(x_j), \tag{2.81}$$

$$\alpha_k = \frac{2}{n} \sum_{j=1}^{n} f(x_j) \, T_k(x_j), \tag{2.82}$$

for $k = 1, 2, \ldots, n-1$. For an integer $n > 0$, we define the linear operator $A_T : \mathbb{R}^n \to \mathbb{R}^n$,

$$A_T \left(f(x_1), f(x_2), \ldots, f(x_n) \right) = \left(\alpha_0, \alpha_1, \ldots, \alpha_{n-1} \right), \tag{2.83}$$

as the Chebyshev transform of size n. A_T converts values of f at the roots of T_n into coefficients in a Chebyshev polynomial expansion using equations (2.81) and (2.82). In matrix notation,

$$A_T = \frac{1}{n} S_T \, T_T, \tag{2.84}$$

where

$$S_T = \begin{pmatrix} 1 & & & \\ & 2 & & \\ & & \ddots & \\ & & & 2 \end{pmatrix}, \tag{2.85}$$

$$T_T = \begin{pmatrix} T_0(x_1) & \cdots & T_0(x_n) \\ \vdots & \ddots & \vdots \\ T_{n-1}(x_1) & \cdots & T_{n-1}(x_n) \end{pmatrix}. \tag{2.86}$$

The algorithm of Section 4 will enable the accelerated application of the matrix T_T in equation (2.86).

Remark 2.10 Although the algorithm of this paper accelerates the calculation of the expansion coefficients $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$ in equation (2.75) via the application of the matrix T_T, it is well known that due to the connection between trigonometric polynomials and Chebyshev polynomials (see identity (2.69)), these expansion coefficients can also be calculated rapidly via the use of an FFT. When the basis functions are Chebyshev polynomials (or complex exponentials) and the quadrature nodes $x_1, x_2, \ldots, x_n$ are the roots of T_n (or equispaced), the use of an FFT to calculate the expansion coefficients is preferable, and faster than the algorithm of this paper. However, for nonequispaced nodes $x_1, x_2, \ldots, x_n$, the algorithm of this paper becomes a new version of the nonequispaced FFT.

The integrals in Lemma 2.11 below are used in Section 3. These identities are a restatement of Formulas 7.355.1 and 7.355.2 in [10].

Lemma 2.11 *For any integer $n \geq 0$, and any real number $a > 0$,*

$$\int_0^1 T_{2n+1}(x)\,\sin ax\,\frac{dx}{\sqrt{1-x^2}} = (-1)^n\,\frac{\pi}{2}\,J_{2n+1}(a), \tag{2.87}$$

$$\int_0^1 T_{2n}(x)\,\cos ax\,\frac{dx}{\sqrt{1-x^2}} = (-1)^n\,\frac{\pi}{2}\,J_{2n}(a), \tag{2.88}$$

where J_n is the Bessel function of the first kind of order n.

2.7 Hermite polynomials

For an integer $n \geq 0$, we denote by H_n the Hermite polynomial of degree n, which is the solution on $\mathbb{R}$ to the differential equation

$$\frac{d}{dx}\left(e^{-x^2}\,\frac{dy}{dx}\right) + 2n\,e^{-x^2}\,y = 0, \tag{2.89}$$

given by the formula

$$H_n(x) = (-1)^n\,e^{x^2}\,\frac{d^n}{dx^n}e^{-x^2}. \tag{2.90}$$

Hermite polynomials are orthogonal on $\mathbb{R}$ with respect to the weight function $w(x) = e^{-x^2}$, i.e. for any $m, n \geq 0$,

$$\int_{-\infty}^{\infty} H_m(x)\,H_n(x)\,e^{-x^2}\,dx = \sqrt{\pi}\,2^n\,n!\,\delta_{mn}. \tag{2.91}$$

Furthermore, Hermite polynomials satisfy the recurrence

$$H_{n+1}(x) = 2\,x\,H_n(x) - 2n\,H_{n-1}(x) \tag{2.92}$$

with initial conditions

$$H_0(x) = 1, \tag{2.93}$$

$$H_1(x) = x. \tag{2.94}$$

Differentiating (2.92), we obtain a recurrence for the derivatives of Hermite polynomials, which are used to calculate the weights of the Gaussian quadrature corresponding to the weight function $w(x) = e^{-x^2}$ (see Section 2.4). The use of Hermite polynomials in numerical applications is prohibited due to poor scaling for large values of their argument, i.e. for large $|x|$, $|H_n(x)|$ is very large, for all $n > 0$. Thus, it is necessary that we instead use normalized Hermite functions, given by the formula

$$h_n(x) = \frac{1}{\pi^{\frac{1}{4}}\,2^{\frac{n}{2}}\,\sqrt{n!}}\,H_n(x)\,e^{\frac{-x^2}{2}}. \tag{2.95}$$

Normalized Hermite functions form an orthonormal basis in $L^2(\mathbb{R})$, and satisfy the differential equation

$$\frac{d^2y}{dx^2} + (2n+1-x^2)\,y = 0. \tag{2.96}$$

The functions $h_0, h_1, h_2, \ldots$ can be calculated numerically using the recurrence

$$h_{n+1}(x) = \sqrt{\frac{2}{n+1}}\,x\,h_n(x) - \sqrt{\frac{n}{n+1}}\,h_{n-1}(x) \tag{2.97}$$

with initial conditions

$$h_0(x) = \frac{1}{\pi^{\frac{1}{4}}} e^{\frac{-x^2}{2}}, \tag{2.98}$$

$$h_1(x) = \frac{x}{\pi^{\frac{1}{4}}\sqrt{2}} e^{\frac{-x^2}{2}}. \tag{2.99}$$

Since Hermite functions are dense in $L^2(\mathbb{R})$, for any square integrable function $f : \mathbb{R} \to \mathbb{R}$ there exist real coefficients $\alpha_0, \alpha_1, \alpha_2, \ldots$ such that

$$f(x) = \sum_{k=0}^{\infty} \alpha_k \, h_k(x). \tag{2.100}$$

Most numerical applications involving Hermite polynomials use a truncated version of the sum in equation (2.100) to provide an approximation to f, so that

$$f(x) \approx \sum_{k=0}^{n-1} \alpha_k \, h_k(x). \tag{2.101}$$

We recall from Section 2.4 that the n-point Gaussian quadrature for weight function $w(x) = e^{-x^2}$ on $\mathbb{R}$ has nodes $x_1, x_2, \ldots, x_n$ equal to the roots of H_n and weights $w_1, w_2, \ldots, w_n$ given by

$$w_j = \frac{\sqrt{\pi}\, 2^{n+1}\, n!}{H_n'(x_j)^2}. \tag{2.102}$$

If this quadrature is applied to products of Hermite polynomials, we obtain the identity

$$\sum_{j=1}^{n} w_j \, H_k(x_j) \, H_l(x_j) = \sqrt{\pi}\, 2^k \, k! \, \delta_{kl} \tag{2.103}$$

for all $0 \le k, l \le n-1$. A straightforward algebraic manipulation shows that (2.103) is equivalent to

$$\sum_{j=1}^{n} \tilde{w}_j \, h_k(x_j) \, h_l(x_j) = \delta_{kl}, \tag{2.104}$$

where

$$\tilde{w}_j = \frac{2}{h_n'(x_j)^2}. \tag{2.105}$$

In this paper, we require the roots of H_n for use in computing the expansion coefficients in equation (2.101) and quadrature weights in equation (2.105). These roots $x_1, x_2, \ldots, x_n$ can be numerically calculated using the method contained in [9]; once calculated, α_k is given by the formula

$$\alpha_k = \sum_{j=1}^{n} \tilde{w}_j \, f(x_j) \, h_k(x_j) \tag{2.106}$$

for $k = 0, 1, 2, \ldots, n-1$. For an integer $n > 0$, we define the linear operator $A_H : \mathbb{R}^n \to \mathbb{R}^n$,

$$A_H\left(f(x_1), f(x_2), \ldots, f(x_n)\right) = \left(\alpha_0, \alpha_1, \ldots, \alpha_{n-1}\right), \tag{2.107}$$

as the Hermite function transform of size n. A_H converts values of f at the roots of H_n into coefficients in a Hermite function expansion using equation (2.106). In matrix notation,

$$A_H = W_H \, T_H, \tag{2.108}$$

where

$$W_H = \begin{pmatrix} \bar{w}_1 & & & \\ & \bar{w}_2 & & \\ & & \ddots & \\ & & & \bar{w}_n \end{pmatrix}, \tag{2.109}$$

$$T_H = \begin{pmatrix} h_0(x_1) & \cdots & h_0(x_n) \\ \vdots & \ddots & \vdots \\ h_{n-1}(x_1) & \cdots & h_{n-1}(x_n) \end{pmatrix}. \tag{2.110}$$

The algorithm described in Section 4 will accelerate the application of matrix T_H in equation (2.110).

2.8 Laguerre polynomials

For an integer $n \geq 0$, we denote by L_n the Laguerre polynomial of degree n, which is the solution on $\mathbb{R}^+$ to the differential equation

$$\frac{d}{dx}\left(x\,e^{-x}\,\frac{dy}{dx}\right) + n\,e^{-x}\,y = 0, \tag{2.111}$$

given by the formula

$$L_n(x) = \frac{1}{n!}\,e^x\,\frac{d^n}{dx^n}e^{-x}. \tag{2.112}$$

Laguerre polynomials form an orthonormal basis in $L^2(\mathbb{R}^+)$ with respect to the weight function $w(x) = e^{-x}$, i.e. for any $m, n \geq 0$,

$$\int_0^\infty L_m(x)\,L_n(x)\,e^{-x}\,dx = \delta_{mn}. \tag{2.113}$$

Laguerre polynomials satisfy the recurrence

$$(n+1)\,L_{n+1}(x) = (2n+1-x)\,L_n(x) - n\,L_{n-1}(x) \tag{2.114}$$

with initial conditions

$$L_0(x) = 1, \tag{2.115}$$

$$L_1(x) = 1 - x. \tag{2.116}$$

Differentiating recurrence (2.114) yields a recurrence for the derivatives of Laguerre polynomials, which are used to calculate weights for the Gaussian quadrature corresponding to weight function $w(x) = e^{-x}$ on $\mathbb{R}^+$ (see Section 2.4). Much like Hermite polynomials, Laguerre polynomials are not functions that can be used numerically due to improper scaling. In numerical applications, we instead use normalized Laguerre functions ℓ_n, given by the formula

$$\ell_n(x) = L_n(x)\,e^{\frac{-x}{2}}. \tag{2.117}$$

Laguerre functions satisfy the same recurrence as Laguerre polynomials, recurrence (2.114), except with the initial conditions

$$\ell_0(x) = e^{\frac{-x}{2}}, \tag{2.118}$$

$$\ell_1(x) = (1-x)e^{\frac{-x}{2}}. \tag{2.119}$$

The functions $\ell_0, \ell_1, \ell_2, \ldots$ can be numerically evaluated using recurrence (2.114) with initial conditions above in equations (2.118) and (2.119). Since Laguerre functions form an orthonormal basis in $L^2(\mathbb{R}^+)$, for any square integrable function $f : \mathbb{R}^+ \to \mathbb{R}$ there exist $\alpha_0, \alpha_1, \alpha_2, \ldots$ such that

$$f(x) = \sum_{k=0}^{\infty} \alpha_k \, \ell_k(x). \tag{2.120}$$

Many numerical applications use a truncated version of the sum in equation (2.120) to approximate the function f, such that

$$f(x) \approx \sum_{k=0}^{n-1} \alpha_k \, \ell_k(x). \tag{2.121}$$

To determine $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$, we recall from Section 2.4 that the n-point Gaussian quadrature corresponding to weight function $w(x) = e^{-x}$ on $\mathbb{R}^+$ has nodes $x_1, x_2, \ldots, x_n$ equal to the roots of L_n and weights $w_1, w_2, \ldots, w_n$ given by

$$w_j = \frac{1}{x_j \, L_n'(x_j)^2}. \tag{2.122}$$

Applying this quadrature to products of Laguerre polynomials we obtain the identity

$$\sum_{j=1}^{n} w_j \, L_k(x_j) \, L_l(x_j) = \delta_{kl}, \tag{2.123}$$

for all $0 \leq k, l \leq n-1$. After an algebraic manipulation of (2.123), we obtain a similar expression involving normalized Laguerre functions,

$$\sum_{j=1}^{n} \tilde{w}_j \, \ell_k(x_j) \, \ell_l(x_j) = \delta_{kl}, \tag{2.124}$$

where

$$\tilde{w}_j = \frac{1}{x_j \, \ell_n'(x_j)^2}. \tag{2.125}$$

The roots $x_1, x_2, \ldots, x_n$, which are used in formulas (2.122–2.125), can be numerically calculated using the method contained in [9]. The coefficients in expansion (2.121) are then given by the formula

$$\alpha_k = \sum_{j=1}^{n} \tilde{w}_j \, f(x_j) \, \ell_k(x_j). \tag{2.126}$$

For an integer $n > 0$, we define the linear operator $A_L : \mathbb{R}^n \to \mathbb{R}^n$,

$$A_L \left(f(x_1), f(x_2), \ldots, f(x_n) \right) = \left(\alpha_0, \alpha_1, \ldots, \alpha_{n-1} \right), \tag{2.127}$$

as the Laguerre function transform of size n. A_L converts values of f at the roots of L_n into coefficients in a Laguerre function expansion using equation (2.126). In matrix notation,

$$A_L = W_L \, T_L, \tag{2.128}$$

where

$$W_L = \begin{pmatrix} \tilde{w}_1 & & & \\ & \tilde{w}_2 & & \\ & & \ddots & \\ & & & \tilde{w}_n \end{pmatrix}, \tag{2.129}$$

$$T_L = \begin{pmatrix} \ell_0(x_1) & \cdots & \ell_0(x_n) \\ \vdots & \ddots & \vdots \\ \ell_{n-1}(x_1) & \cdots & \ell_{n-1}(x_n) \end{pmatrix}. \tag{2.130}$$

The application of matrix T_L in equation (2.130) will be accelerated by the algorithm described in Section 4.

Observation 2.12 In Section 5 we will use the fact that for integers n, $\nu \geq 0$, the normalized generalized Laguerre functions ℓ_n^ν are the eigenfunctions of the Fourier-Bessel transform of order ν. The normalized generalized Laguerre functions are given by the formula

$$\ell_n^\nu(x) = \sqrt{\frac{n!}{(\nu+n)!}}\, e^{-\frac{x}{2}}\, x^{\frac{\nu}{2}}\, L_n^\nu(x), \tag{2.131}$$

where $L_n^\nu(x)$ is the solution on $\mathbb{R}^+$ to the differential equation (similar to equation (2.111))

$$\frac{d}{dx}\left(x^{\nu+1}\, e^{-x}\, \frac{dy}{dx}\right) + n\, x^\nu\, e^{-x}\, y = 0, \tag{2.132}$$

given by the formula (which is very similar to that in equation (2.112))

$$L_n^\nu(x) = \frac{1}{n!}\, e^x\, x^{-\nu}\, \frac{d^n}{dx^n}\left(e^{-x} x^{n+\nu}\right). \tag{2.133}$$

The eigenfunction relation is given by the formula

$$\int_0^\infty x\, J_\nu(xy)\, \ell_n^\nu(x^2)\, dx = (-1)^n\, \ell_n^\nu(y^2). \tag{2.134}$$

More general forms of equation (2.134) can be found in [3] and [10].

2.9 Prolate spheroidal wave functions

In this section we summarize certain properties of prolate spheroidal wave functions to be used in Section 5; more detailed information can be found in [19] and [24]. For a real $c > 0$, we define the operator $F_c : L^2[-1,1] \to L^2[-1,1]$ as

$$(F_c\varphi)(x) = \int_{-1}^{1} e^{icxt}\, \varphi(t)\, dt. \tag{2.135}$$

The operator F_c has eigenvalues $\lambda_0, \lambda_1, \lambda_2, \ldots$ such that $|\lambda_{n-1}| \geq |\lambda_n|$ for all $n \geq 1$. Furthermore, for each $n = 0, 1, 2, \ldots$

$$\lambda_n = i^n\, |\lambda_n|. \tag{2.136}$$

We denote by ψ_n^c the n^{th} prolate spheroidal wave function (PSWF), which is the eigenfunction of F_c corresponding to the eigenvalue λ_n. The function ψ_n^c is also a solution on $[-1,1]$ to the differential equation

$$\frac{d}{dx}\left((1-x^2)\, \frac{dy}{dx}\right) + (\chi_n - c^2\, x^2)\, y = 0, \tag{2.137}$$

where $0 < \chi_0 < \chi_1 < \chi_2 < \ldots$ are referred to as the separation coefficients. The functions $\psi_0^c, \psi_1^c, \psi_2^c, \ldots$ form an orthonormal basis in $L^2[-1,1]$ under the usual inner product formula, i.e. for any $m, n \geq 0$

$$\int_{-1}^{1} \psi_m^c(x)\, \psi_n^c(x)\, dx = \delta_{mn}. \tag{2.138}$$

For a given bandwidth c and properly chosen n, the functions $\psi_0^c, \psi_1^c, \ldots, \psi_{n-1}^c$ are a set of stable interpolation functions which can be used to accurately approximate functions of the form e^{icx}, with $x \in [-1, 1]$. To rephrase, given a function $f : [-1, 1] \to \mathbb{C}$,

$$f(x) = e^{icx}, \tag{2.139}$$

there exist coefficients $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$ such that

$$f(x) \approx \sum_{k=1}^{n-1} \alpha_k \, \psi_k^c(x). \tag{2.140}$$

If interpolation nodes are chosen to be the n roots of the function ψ_n^c, the determination of $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$ is a well-conditioned procedure. Using the methods of [9], we can numerically calculate the roots of any PSWF as well as evaluate any PSWF on the interval $[-1, 1]$. Suppose now that the coefficients in expansion (2.140) are known, and one wishes to evaluate the expansion at the n points $x_1, x_2, \ldots, x_n$ (for example, the n roots of ψ_n^c). For any integer n and complex numbers $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$, we define the linear operator $E_\psi^c : \mathbb{R}^n \to \mathbb{R}^n$,

$$E_\psi^c \left(\alpha_0, \alpha_1, \ldots, \alpha_{n-1}\right) = \left(f(x_1), f(x_2), \ldots, f(x_n)\right) \tag{2.141}$$

as PSWF interpolation of size n. In matrix notation,

$$E_\psi^c = \begin{pmatrix} \psi_0^c(x_1) & \cdots & \psi_{n-1}^c(x_1) \\ \vdots & \ddots & \vdots \\ \psi_0^c(x_n) & \cdots & \psi_{n-1}^c(x_n) \end{pmatrix}. \tag{2.142}$$

The algorithm described in Section 4 will accelerate the application of the matrix E_ψ^c in equation (2.142).

3 Analytical apparatus

The algorithm of this paper relies on the observation that for certain $n \times n$ matrices the numerical rank of any $m \times k$ contiguous submatrix depends only on the quantity mk. This section contains a proof of this fact for the discrete Fourier transform and the discrete Fourier-Bessel transform. *Inter alia*, we first prove analogous results for the continuous Fourier transform and continuous Fourier-Bessel transform.

3.1 Rank analysis of the Fourier transform

The Fourier transform $\mathcal{F} : L^2(\mathbb{R}) \to L^2(\mathbb{R})$ is given by the formula

$$(\mathcal{F}f)(x) = \int_{-\infty}^{\infty} e^{ixt} \, f(t) \, dt. \tag{3.1}$$

Let a, b, u, v be real numbers such that $-\pi \leq a < b \leq \pi$ and $u < v$. We wish to show that the operator $S : L^2[a, b] \to L^2[u, v]$, given by the formula

$$(Sf)(x) = \int_a^b e^{ixt} \, f(t) \, dt, \tag{3.2}$$

has numerical rank that depends only on the quantity $(v - u)(b - a)$. First, we define what we mean by the numerical rank of an integral operator, as well as the numerical rank of a matrix.

ρ	k	Error	Predicted k
10	30	.99111E-12	49
50	83	.99726E-12	95
100	142	.99918E-12	152
500	570	.99992E-12	581
1000	1088	.99991E-12	1100
5000	5151	.99967E-12	5167
10000	10190	.99991E-12	10210
50000	50324	.99995E-12	50357

Table 2: Accuracy of Chebyshev expansion for complex exponential.

Definition 3.1 *We say that an integral operator $S : L^2[a,b] \to L^2[c,d]$ with kernel $s : [c,d] \times [a,b] \to \mathbb{C}$,*

$$(Sf)(x) = \int_a^b s(x,t)\, f(t)\, dt, \tag{3.3}$$

has rank k if s can be written as

$$s(x,t) = \sum_{n=1}^{k} g_n(x)\, h_n(t), \tag{3.4}$$

for some set of functions $g_1, h_1, g_2, h_2, \ldots, g_k, h_k$. Likewise, S has rank k to precision $\varepsilon > 0$ if s can be written as

$$s(x,t) = \sum_{n=1}^{\infty} g_n(x)\, h_n(t), \tag{3.5}$$

where for every $n > k$,

$$|g_n(x)\, h_n(t)| < \varepsilon \tag{3.6}$$

for all $t \in [a,b]$ and $x \in [c,d]$.

If A is an $m \times n$ matrix, we say that it has rank k if we can write the singular value decomposition of A as

$$A = UDV^t, \tag{3.7}$$

where U and V are matrices of dimension $m \times k$ and $n \times k$,respectively, both whose columns are orthonormal and D is a $k \times k$ diagonal matrix with diagonal elements σ_i, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k > 0$. We say that A has rank $k^{'}$ to precision ε if $\sigma_{k^{'}} \geq \varepsilon > \sigma_{k^{'}+1}$.

Our aim now is to show that the kernel of the operator in (3.2), $s(x,t) = e^{ixt}$, can be written in the form (3.5). The following lemma is an immediate consequence of Formulas 9.1.44 and 9.1.45 in [1].

Lemma 3.2 *For any $\rho \in \mathbb{R}$ and $\tau \in [-1,1]$,*

$$e^{i\rho\tau} = J_0(\rho) + \sum_{n=1}^{\infty} 2\, i^n\, J_n(\rho)\, T_n(\tau). \tag{3.8}$$

Table 2 shows the numerical accuracy of a truncated version of expansion (3.8), along with the predicted expansion order from Lemma 2.6. The column labeled "ρ" represents the same ρ as in equation (3.8). The column labeled "k" gives the number of terms needed in expansion (3.8) to achieve an absolute accuracy of at least 10^{-12} for all $\tau \in [-1,1]$. The values in the column labeled "error" are calculated as

$$\max_{\tau\in[-1,1]} \left| e^{i\rho\tau} - J_0(\rho) - \sum_{n=1}^{k} 2\, i^n\, J_n(\rho)\, T_n(\tau) \right|. \tag{3.9}$$

The value in the column "predicted k" is the smallest positive integer k such that

$$|2\, i^n\, J_k(\rho)\, T_k(\tau)| \le 2\ |J_k(\rho)| < 10^{-12}. \tag{3.10}$$

This value of k is obtained via Lemma 2.6. We can now state and prove the following theorem.

Theorem 3.3 *If a, b, u, v are real numbers such that $-\pi \le a < b \le \pi$, $u < v$, and $(v-u)(b-a) > 1$, then for any fixed positive $\varepsilon < 0.1$, the integral operator $S : L^2[a,b] \to L^2[u,v]$,*

$$(Sf)(x) = \int_a^b e^{ixt}\, f(t)\, dt, \tag{3.11}$$

has rank to precision ε at most a constant times $(v-u)(b-a)$.

Proof. We first expand the kernel of operator (3.11) into the form of (3.5). We begin by rewriting (3.11) as

$$(Sf)(x) = \bar{S}f(\xi) = \int_a^b e^{i(u+(v-u)\xi)t}\, f(t)\, dt, \tag{3.12}$$

where $\bar{S} : L^2[a,b] \to L^2[0,1]$, and ξ is given by the formula

$$\xi = \frac{x-u}{v-u}. \tag{3.13}$$

Letting $t = \frac{b-a}{2}\tau + \frac{b+a}{2}$, we convert (3.12) into

$$(\bar{S}f)(\xi) = \int_{-1}^{1} e^{i(u+(v-u)\xi)(\frac{b-a}{2}\tau+\frac{b+a}{2})}\, \frac{b-a}{2}\, f(\tau)\, d\tau. \tag{3.14}$$

If we now define $\bar{s}$ to be the kernel of integral operator (3.14), and let $c = (v-u)\frac{b-a}{2}$, after applying Lemma 3.2 we obtain

$$\begin{aligned}
\bar{s}(\xi,\tau) &= \frac{b-a}{2}\, e^{i(u+(v-u)\xi)(\frac{b-a}{2}\tau+\frac{b+a}{2})} && (3.15)\\
&= \frac{b-a}{2}\, e^{iu\frac{b+a}{2}}\, e^{i(v-u)\frac{b+a}{2}\xi}\, e^{ic\xi\tau}\, e^{iu\frac{b-a}{2}\tau} && (3.16)\\
&= \frac{b-a}{2}\, e^{iu\frac{b+a}{2}}\, e^{i(v-u)\frac{b+a}{2}\xi} \left(J_0(c\xi) + \sum_{n=1}^{\infty} 2\, i^n\, J_n(c\xi)\, T_n(\tau) \right) e^{iu\frac{b-a}{2}\tau}. && (3.17)
\end{aligned}$$

To simplify the above expression for $\bar{s}$, for $n > 0$ let the function $\kappa_n : [0,1]\times[-1,1] \to \mathbb{C}$ be the n^{th} term in expansion (3.17), i.e.

$$\kappa_n(\xi,\tau) = i^n\, (b-a)\, e^{iu\frac{b+a}{2}}\, e^{i(v-u)\frac{b+a}{2}\xi}\, J_n(c\xi)\, T_n(\tau)\, e^{iu\frac{b-a}{2}\tau}, \tag{3.18}$$

and thus

$$\bar{s}(\xi,\tau) = \frac{b-a}{2}\, e^{iu\frac{b+a}{2}}\, e^{i(v-u)\frac{b+a}{2}\xi}\, J_0(c\xi)\, e^{iu\frac{b-a}{2}\tau} + \sum_{n=1}^{\infty} \kappa_n(\xi,\tau). \tag{3.19}$$

If we take the absolute value of expression (3.18) for κ_n, we arrive at the inequality

$$|\kappa_n(\xi,\tau)| = \left| i^n\,(b-a)\,e^{iu\frac{b+a}{2}}\,e^{i(v-u)\frac{b+a}{2}\xi}\,J_n(c\xi)\,T_n(\tau)\,e^{iu\frac{b-a}{2}\tau}\right| \tag{3.20}$$

$$\leq |(b-a)\,J_n(c\xi)\,T_n(\tau)| \tag{3.21}$$

$$\leq 2\pi\,|J_n(c\xi)|. \tag{3.22}$$

Now, if $c > 10$, then by Lemma 2.6, $|\kappa_n|$ is less than ε whenever

$$n > (c^{1/3} + \alpha\,c^{-1/3})^3, \tag{3.23}$$

where

$$\alpha = \frac{3^{-1/3}}{2}\,\log^{2/3}\left(\frac{2\pi}{\varepsilon}\right). \tag{3.24}$$

If $c \leq 10$, $|\kappa_n|$ is less than ε whenever

$$n > (10^{1/3} + \alpha\,10^{-1/3})^3. \tag{3.25}$$

It now follows from equations (3.23–3.25) that if $(v-u)(b-a) > 1$, the rank to precision ε of the integral operator S is at most a constant times $(v-u)(b-a)$. □

Corollary 3.4 *Let T_F be the $n \times n$ matrix of the discrete Fourier transform, as described in Section 2.3. If A is any $m \times k$ contiguous submatrix of T_F such that $mk > n$, then for any fixed positive $\varepsilon < 0.1$, the rank of A to precision ε is at most a constant times the quantity mk.*

3.2 Rank analysis of the Fourier-Bessel transform

For any real number $R > 0$ and integer $\nu \geq 0$, the Fourier-Bessel transform of order ν $\mathcal{H}_\nu : L^2[0,R] \to L^2(\mathbb{R})$ is given by the formula

$$(\mathcal{H}_\nu f)(\rho) = \int_0^R \sqrt{t}\,J_\nu(\rho t)\,f(t)\,dt \tag{3.26}$$

(see Section 2.2). Let a, b, u, v be real numbers such that $0 \leq a < b \leq R$ and $u < v$. We wish to show that the operator $U_\nu : L^2[a,b] \to L^2[u,v]$, given by the formula

$$(U_\nu f)(\rho) = \int_0^R \sqrt{t}\,J_\nu(\rho t)\,f(t)\,dt, \tag{3.27}$$

has numerical rank at most a constant times $(v-u)(b-a)$. We prove this fact in Theorem 3.5 below.

Theorem 3.5 *For any fixed positive $\varepsilon > 0.1$, $R > 0$, and integer $\nu \geq 0$, if a, b, u, v, are real numbers such that $0 \leq a < b \leq R$, $0 \leq u < v$, and $(v-u)(b-a) > 1$, then the integral operator $U_\nu : L^2[a,b] \to L^2[u,v]$, given by the formula*

$$(U_\nu f)(\rho) = \int_a^b \sqrt{t}\,J_\nu(\rho t)\,f(t)\,dt, \tag{3.28}$$

has rank to precision ε at most a constant times $(v-u)(b-a)$.

Proof. We prove Theorem 3.5 in the case where $\nu = 2m$, for some integer $m \geq 0$. The proof for odd ν is virtually identical, and is omitted. Using Lemma 2.11, we rewrite (3.28) as

$$(U_{2m}f)(\rho) = \int_a^b \sqrt{t} \left((-1)^m \frac{2}{\pi} \int_0^1 \frac{T_{2m}(y)}{\sqrt{1-y^2}} \cos(\rho t y)\, dy \right) f(t)\, dt. \tag{3.29}$$

Changing variables according to

$$t = \frac{b-a}{2}\tau + \frac{a+b}{2}, \tag{3.30}$$

$$\rho = (v-u)\xi + u, \tag{3.31}$$

the operator U_{2m} is converted into $\bar{U}_{2m} : L^2[-1,1] \to L^2[0,1]$, given by the formula

$$\begin{aligned}(\bar{U}_{2m}f)(\xi) = \int_{-1}^1 & \sqrt{\frac{b-a}{2}\tau + \frac{a+b}{2}}\, (-1)^m \,\frac{2}{\pi} \cdot \\ & \int_0^1 \frac{T_{2m}(y)}{\sqrt{1-y^2}} \cos\left([(v-u)\xi + u] \left[\frac{b-a}{2}\tau + \frac{a+b}{2} \right] y \right) dy \,\cdot \\ & f(\tau)\, \frac{b-a}{2} d\tau.\end{aligned} \tag{3.32}$$

We now write the kernel of $\bar{U}_{2m}$ in the form of (3.5). Let $\mu_{2m} : [0,1] \times [-1,1] \to \mathbb{R}$ be the kernel of $\bar{U}_{2m}$, given by the formula

$$\mu_{2m}(\xi,\tau) = \alpha_m(\tau) \int_0^1 \frac{T_{2m}(y)}{\sqrt{1-y^2}} \cos\left([(v-u)\xi + u] \left[\frac{b-a}{2}\tau + \frac{a+b}{2} \right] y \right) dy \tag{3.33}$$

$$= \frac{\alpha_m(\tau)}{2} \int_{-1}^1 \frac{T_{2m}(y)}{\sqrt{1-y^2}}\, e^{i[(v-u)\xi+u]\left[\frac{b-a}{2}\tau + \frac{a+b}{2}\right] y}\, dy, \tag{3.34}$$

where

$$\alpha_m(\tau) = (-1)^m \,\frac{b-a}{\pi} \sqrt{\frac{b-a}{2}\tau + \frac{a+b}{2}}. \tag{3.35}$$

Letting $c = (v-u)\frac{b-a}{2}$, we expand the kernel μ_{2m} using Lemma 3.2 into

$$\mu_{2m}(\xi,\tau) = \frac{\alpha_m(\tau)}{2} \int_{-1}^1 \frac{T_{2m}(y)}{\sqrt{1-y^2}}\, e^{iu\frac{a+b}{2}y}\, e^{i(v-u)\frac{a+b}{2}\xi y}\, e^{ic\xi\tau y}\, e^{iu\frac{b-a}{2}\tau y}\, dy \tag{3.36}$$

$$\begin{aligned}= \frac{\alpha_m(\tau)}{2} \int_{-1}^1 & \frac{T_{2m}(y)}{\sqrt{1-y^2}}\, e^{iu\frac{a+b}{2}y}\, e^{i(v-u)\frac{a+b}{2}\xi y} \,\cdot \\ & \left(J_0(c\xi y) + \sum_{n=1}^{\infty} 2\, i^n\, J_n(c\xi y)\, T_n(\tau) \right) e^{iu\frac{b-a}{2}\tau y}\, dy.\end{aligned} \tag{3.37}$$

To simplify the expression for μ_{2m}, for all $n > 0$ let $\omega_n : [0,1] \times [-1,1] \to \mathbb{R}$ be the n^{th} term in expansion (3.37), given by the formula

$$\omega_n(\xi,\tau) = \alpha_m(\tau)\, i^n \int_{-1}^1 \frac{T_{2m}(y)}{\sqrt{1-y^2}}\, e^{iu\frac{a+b}{2}y}\, e^{i(v-u)\frac{a+b}{2}\xi y}\, J_n(c\xi y)\, T_n(\tau)\, e^{iu\frac{b-a}{2}\tau y}\, dy. \tag{3.38}$$

Taking the absolute value of ω_n, we obtain the inequality

$$|\omega_n(\xi,\tau)| = \left|\alpha_m(\tau)\, i^n \int_{-1}^{1} \frac{T_{2m}(y)}{\sqrt{1-y^2}}\, e^{iu\frac{a+b}{2}y}\, e^{i(v-u)\frac{a+b}{2}\xi y}\, J_n(c\xi y)\, T_n(\tau)\, e^{iu\frac{b-a}{2}\tau y}\, dy\right| \tag{3.39}$$

$$\le \frac{R}{\pi}\sqrt{\frac{3R}{2}} \int_{-1}^{1} \frac{|T_{2m}(y)|}{\sqrt{1-y^2}} \left|e^{iu\frac{a+b}{2}y}\, e^{i(v-u)\frac{a+b}{2}\xi y}\, J_n(c\xi y)\, T_n(\tau)\, e^{iu\frac{b-a}{2}\tau y}\right| dy \tag{3.40}$$

$$\le \frac{R}{\pi}\sqrt{\frac{3R}{2}} \max_{r\in[0,1]} |J_n(cr)| \int_{-1}^{1} \frac{dy}{\sqrt{1-y^2}} \tag{3.41}$$

$$= R\sqrt{\frac{3R}{2}} \max_{r\in[0,1]} |J_n(cr)|. \tag{3.42}$$

Now, if $c > 10$, then by Lemma 2.6, $|\omega_n|$ is less than ε whenever

$$n > (c^{1/3} + \alpha\, c^{-1/3})^3, \tag{3.43}$$

where

$$\alpha = \frac{3^{-1/3}}{2} \log^{2/3}\left(\frac{R}{\varepsilon}\sqrt{\frac{3R}{2}}\right). \tag{3.44}$$

If $c \le 10$, $|\omega_n|$ is less than ε whenever

$$n > (10^{1/3} + \alpha\, 10^{-1/3})^3. \tag{3.45}$$

It now follows from equations (3.43–3.45) that if $(v-u)(b-a) > 1$, the rank to precision ε of the integral operator U_ν is at most a constant times $(v-u)(b-a)$. □

Corollary 3.6 *For fixed $R > 0$, let T_{J_ν} be the $n \times n$ matrix of the discrete Fourier-Bessel transform, as described in Section 2.2. If A is any $m \times k$ contiguous submatrix of T_{J_ν} such that $mk > n$, then for any fixed positive $\varepsilon < 0.1$, the rank of A to precision ε is at most a constant times the quantity mk.*

4 The butterfly algorithm

This section contains a description of an algorithm for the application of $n \times n$ matrices which have the property that any $m \times k$ contiguous submatrix has numerical rank that depends only on the quantity mk.

4.1 Notation

In this section, we establish notation that will be used in Sections 4.2–4.5 when describing the algorithm.

A will denote the matrix being decomposed and applied, and it is of size $n \times n$ where $n = 2^N$ for some positive integer N.

The precision to which all interpolative decompositions are performed is given by ε.

The algorithm imposes a dyadic hierarchy of columns and rows on the matrix A. Columns of A are merged with increasing level while rows of A are split with increasing level. Levels 0 and 1 are shown in Figure 1. If we require that there can be at most C_{max} columns in each submatrix

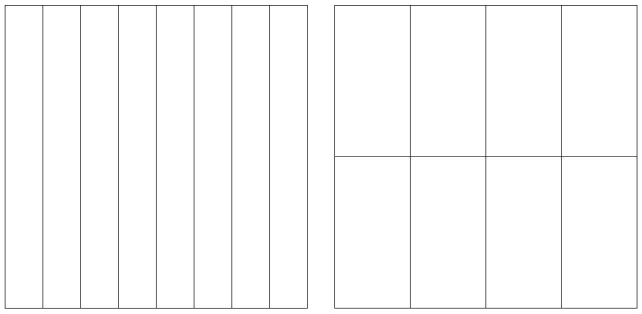

Figure 1: Levels 0 and 1 of A.

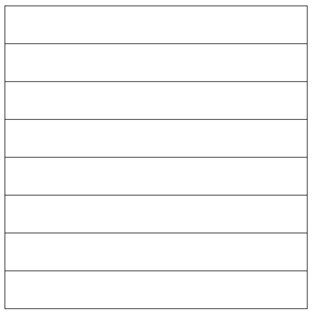

Figure 2: Level L of A.

on level 0 of the hierarchy, then we have L levels, where $C_{max}2^{L-1} \leq n \leq C_{max}2^{L}$. Level L of the matrix is shown in Figure 2.

On every level l, $0 \leq l \leq L$, there are a constant number of submatrices, $2^L \approx \frac{n}{C_{max}}$. We denote by $A^l_{i,j}$ the submatrix in position (i, j) on level l. The index i denotes the row of submatrices and j denotes the column of submatrices. For a given level l, i ranges between 1 and 2^l and j ranges between 1 and 2^{L-l}. The contiguous submatrix $A^l_{i,j}$ consists of the intersection of rows $(i-1)2^{N-l}+1$ through $i2^{N-l}$ and columns $(j-1)2^{N-L-l}+1$ through $j2^{N-L-l}$ of A. Level 1 is given by

$$A = \begin{pmatrix} A^1_{1,1} & A^1_{1,2} & \cdots & A^1_{1,2^{L-1}} \\ A^1_{2,1} & A^1_{2,2} & \cdots & A^1_{2,2^{L-1}} \end{pmatrix}, \tag{4.1}$$

and level L is given by

$$A = \begin{pmatrix} A^L_{1,1} \\ A^L_{2,1} \\ \vdots \\ A^L_{2^L-1,1} \\ A^L_{2^L,1} \end{pmatrix}. \tag{4.2}$$

We denote the rank (to precision ε) of $A^l_{i,j}$ by $k^l_{i,j}$. The matrix $B^l_{i,j}$ will be some selection of $k^l_{i,j}$ columns of $A^l_{i,j}$ that allow for a stable interpolative decomposition of the form $A^l_{i,j} = B^l_{i,j}\,P$, for some matrix P (see Section 2.1). The matrix $B^l_{i,j}$ will be referred to as the *column skeleton matrix* of $A^l_{i,j}$, and P will be referred to as an *interpolation matrix*.

For a matrix X dimensioned $2m \times k$, we denote by X^+ the $m \times k$ matrix obtained from the top m rows of X. Likewise, X^- is used to denote the $m \times k$ matrix obtained from the bottom m rows of X. Thus,

$$X = \begin{pmatrix} X^+ \\ X^- \end{pmatrix}. \tag{4.3}$$

If we are to use the algorithm described in Sections 4.2–4.5 to compute $y = A\,x$, then for $j = 1, 2, \ldots, 2^L$ we denote by x_j^L the column vector of length 2^{N-L} obtained from elements $(j-1)2^{N-L} + 1$ through $j2^{N-L}$ of x. Analogously, for $i = 1, 2, \ldots, 2^L$ we denote by y_i^L the column vector of length 2^{N-L} obtained from elements $(i-1)2^{N-L} + 1$ through $i2^{N-L}$ of y.

4.2 Informal description

Given the matrix A, the general strategy of the algorithm is to form an interpolative decomposition for every matrix on every level using previous interpolative decompositions that were created on previous levels. It terminates with the application of a decomposed version of the matrix A to an arbitrary vector x. We first describe a two level algorithm, and then generalize to a multilevel algorithm.

If we let $C_{max} = \frac{n}{2}$, then there exists two levels of A. Level 0 is given by

$$A = \begin{pmatrix} A_{1,1}^0 & A_{1,2}^0 \end{pmatrix}, \tag{4.4}$$

where $A_{1,1}^0$ and $A_{1,2}^0$ are both $n \times \frac{n}{2}$ matrices. Level 1 is given by

$$A = \begin{pmatrix} A_{1,1}^1 \\ A_{2,1}^1 \end{pmatrix}, \tag{4.5}$$

where $A_{1,1}^1$ and $A_{2,1}^1$ are both $\frac{n}{2} \times n$ matrices. Using the interpolative decomposition described in Section 2.1, we can write $A_{1,1}^0$ and $A_{1,2}^0$ as

$$A_{1,1}^0 = B_{1,1}^0\, P_{1,1}^0, \tag{4.6}$$

$$A_{1,2}^0 = B_{1,2}^0\, P_{1,2}^0, \tag{4.7}$$

where $B_{1,1}^0$ is a selection of $k_{1,1}^0$ columns of $A_{1,1}^0$, $B_{1,2}^0$ is a selection of $k_{1,2}^0$ columns of $A_{1,2}^0$, and $P_{1,1}^0$ and $P_{1,2}^0$ contain $k_{1,1}^0 \times k_{1,1}^0$ and $k_{1,2}^0 \times k_{1,2}^0$ identity matrices, respectively. Using decompositions (4.6) and (4.7), the matrix A is now given by the formula

$$A = \begin{pmatrix} B_{1,1}^0 & B_{1,2}^0 \end{pmatrix} \begin{pmatrix} P_{1,1}^0 & 0 \\ 0 & P_{1,2}^0 \end{pmatrix}. \tag{4.8}$$

Decomposition (4.8) expresses A through only $k_{1,1}^0 + k_{1,2}^0$ of its columns. We now form decompositions of $A_{1,1}^1$ and $A_{2,1}^1$ similar to those in (4.6) and (4.7).

Since $A_{1,1}^0$ and $A_{1,2}^0$ of level 0 can be represented using only the columns chosen for $B_{1,1}^0$ and $B_{1,2}^0$, respectively, it is possible to choose columns of $B_{1,1}^0$ and $B_{1,2}^0$ to represent matrices $A_{1,1}^1$ and $A_{2,1}^1$ of level 1. Thus, we can form the interpolative decompositions

$$\begin{pmatrix} B_{1,1}^0 & B_{1,2}^0 \end{pmatrix}^+ = B_{1,1}^1\, P_{1,1}^1 \tag{4.9}$$

and

$$\begin{pmatrix} B_{1,1}^0 & B_{1,2}^0 \end{pmatrix}^- = B_{2,1}^1\, P_{2,1}^1, \tag{4.10}$$

where $B^1_{1,1}$ is a selection of $k^1_{1,1}$ columns of $A^1_{1,1}$ which were chosen from those used to form $B^0_{1,1}$ and $B^0_{1,2}$, $B^1_{2,1}$ is a selection of $k^1_{2,1}$ columns of $A^1_{2,1}$ which were chosen from those used to form $B^0_{1,1}$ and $B^0_{1,2}$, and $P^1_{1,1}$ and $P^1_{2,1}$ contain $k^1_{1,1} \times k^1_{1,1}$ and $k^1_{2,1} \times k^1_{2,1}$ identity matrices, respectively. The submatrices $A^1_{1,1}$ and $A^1_{2,1}$ now have the decompositions

$$A^1_{1,1} = B^1_{1,1}\, P^1_{1,1} \begin{pmatrix} P^0_{1,1} & 0 \\ 0 & P^0_{1,2} \end{pmatrix}, \tag{4.11}$$

$$A^1_{2,1} = B^1_{2,1}\, P^1_{2,1} \begin{pmatrix} P^0_{1,1} & 0 \\ 0 & P^0_{1,2} \end{pmatrix}. \tag{4.12}$$

The final decomposition of the matrix A is

$$A = \begin{pmatrix} B^1_{1,1} & 0 \\ 0 & B^1_{2,1} \end{pmatrix} \begin{pmatrix} P^1_{1,1} \\ P^1_{2,1} \end{pmatrix} \begin{pmatrix} P^0_{1,1} & 0 \\ 0 & P^0_{1,2} \end{pmatrix}. \tag{4.13}$$

An examination of decomposition (4.13) shows how an arbitrary vector x may be applied to the matrix A.

We have described a two level algorithm for the decomposition and application of the matrix A. If n and C_{max} are such that the highest level is $L > 1$, we proceed as follows. Level 0 of A is then given by

$$A = \begin{pmatrix} A^0_{1,1} & A^0_{1,2} & \cdots & A^0_{1,2^L} \end{pmatrix}, \tag{4.14}$$

where $A^0_{1,1}, A^0_{1,2}, \ldots, A^0_{1,2^L}$ are all $n \times 2^{N-L}$ matrices (since $n = 2^N$, for some N). First, interpolative decompositions are formed for these submatrices on level 0, creating the matrices $B^0_{1,1}, P^0_{1,1}, \ldots, B^0_{1,2^L}, P^0_{1,2^L}$. Next, for $j = 1, 2, \ldots, 2^{L-1}$ the pair of matrices $B^0_{1,2j-1}$ and $B^0_{1,2j}$ is used to create an interpolative decomposition for $A^1_{1,j}$ and $A^1_{2,j}$, creating matrices $B^1_{1,j}$, $P^1_{1,j}$ and $B^1_{2,j}$, $P^1_{2,j}$, respectively. This procedure is analogous to creating decompositions (4.11) and (4.12) in the two level algorithm we described earlier in this section.

We continue forming interpolative decompositions on successively higher levels by combining column skeleton matrices which are adjacent to each other on previous levels. To be more precise, for $l > 0$, $i = 1, 2, \ldots, 2^l$, and $j = 1, 2, \ldots, 2^{L-l}$ we use the matrices $B^{l-1}_{\lfloor \frac{i+1}{2} \rfloor, 2j-1}$ and $B^{l-1}_{\lfloor \frac{i+1}{2} \rfloor, 2j}$ to create an interpolative decomposition for $A^l_{i,j}$, which results in the creation of column skeleton matrix $B^l_{i,j}$ and interpolation matrix $P^l_{i,j}$. On level L, the decomposition of $A^L_{i,1}$ is given by the formula

$$\begin{aligned} A^L_{i,1} = {} & B^L_{i,1}\, P^L_{i,1} \begin{pmatrix} P^{L-1}_{\lfloor \frac{i+1}{2} \rfloor, 1} & \\ & P^{L-1}_{\lfloor \frac{i+1}{2} \rfloor, 2} \end{pmatrix} \begin{pmatrix} P^{L-2}_{\lfloor \frac{\lfloor \frac{i+1}{2} \rfloor + 1}{2} \rfloor, 1} & & \\ & \ddots & \\ & & P^{L-2}_{\lfloor \frac{\lfloor \frac{i+1}{2} \rfloor + 1}{2} \rfloor, 4} \end{pmatrix} \cdots \\ & \cdots \begin{pmatrix} P^0_{1,1} & & & & \\ & P^0_{1,2} & & & \\ & & \ddots & & \\ & & & P^0_{1,2^L-1} & \\ & & & & P^0_{1,2^L} \end{pmatrix} \end{aligned} \tag{4.15}$$

for $i = 1, 2, \ldots, 2^L$. For $L > 1$, the decomposition of the whole matrix A is too complicated to give here, but an examination of (4.15) shows how the matrix A can be applied to a vector x using the individual decompositions of $A^L_{1,1}, A^L_{2,1}, \ldots, A^L_{2^L,1}$.

Remark 4.1 Since for all l, i, and j the ranks of $A^l_{i,j}$ are virtually the same, all the interpolation matrices $P^l_{i,j}$ for $l > 0$ are of bounded size, $\tilde{k} \times 2\,\tilde{k}$, where $\tilde{k}$ is the rank of any submatrix on any level. The algorithm is able to apply A to an arbitrary vector x in $\mathcal{O}(n \log n)$ operations due to the fact that all of the interpolation matrices are of the same size.

Observation 4.2 The algorithm described in Section 4.2 will exhibit the same performance when applied to the matrix A^t since all submatrix rank considerations remain the same under the transpose operation.

4.3 Detailed description

This section contains a detailed description of the algorithm that was described informally in Section 4.2 for the decomposition and application of the matrix A.

Step 1

Choose main parameters and create dyadic hierarchy

1. Set precision ε for interpolative decompositions.
2. Set submatrix size parameter C_{max}.
3. Calculate the number of levels L such that $C_{max}2^{L-1} \le n \le C_{max}2^L$.

```
Comment [Create the dyadic hierarchy of columns and rows. On each of the $L+1$ levels of the hierarchy, there will be $2^L$ submatrices. Retain structure created for use in precomputation.]
do $l = 0, 1, \ldots, L$
    do $i = 1, 2, \ldots, 2^l$
        do $j = 1, 2, \ldots, 2^{L-l}$
            Denote the submatrix consisting of rows $(i-1)2^{N-l}+1$ through $i2^{N-l}$ and
            columns $(j-1)2^{N-L-l}+1$ through $j2^{N-L-l}$ as $A^l_{i,j}$.
        end do
    end do
end do
```

Step 2

Precomputation

```
Comment [During this stage, all submatrices on all levels are compressed using the interpolative decomposition described in Section 2.1.]
do $l = 0, 1, \ldots, L$
    do $i = 1, 2, \ldots, 2^l$
        do $j = 1, 2, \ldots, 2^{L-l}$
            if $l = 0$ then
                1. Form the interpolative decomposition
                   $A^0_{1,j} = B^0_{1,j}\, P^0_{1,j}$.        (4.16)
                2. Store $P^0_{1,j}$.
            if ($l > 0$ and $i$ is odd) then

              1. Form the interpolative decomposition
```

$$\left(B^{l-1}_{\lfloor \frac{i+1}{2} \rfloor, 2j-1} \, B^{l-1}_{\lfloor \frac{i+1}{2} \rfloor, 2j} \right)^{+} = B^{l}_{i,j} \, P^{l}_{i,j}. \tag{4.17}$$

```
              2. Store $P^l_{i,j}$.

              3. if $l = L$ then store $B^l_{i,j}$.

            if ($l > 0$ and $i$ is even) then

              1. Form the interpolative decomposition
```

$$\left(B^{l-1}_{\lfloor \frac{i+1}{2} \rfloor, 2j-1} \, B^{l-1}_{\lfloor \frac{i+1}{2} \rfloor, 2j} \right)^{-} = B^{l}_{i,j} \, P^{l}_{i,j}. \tag{4.18}$$

```
              2. Store $P^l_{i,j}$.

              3. if $l = L$ then store $B^l_{i,j}$.

        end do
    end do
end do
```

Comment [As noted above, the matrices $B^l_{i,j}$ only need to be stored in memory for $l = L$. Only these column skeleton matrices are used in the application of A in Step 3. For $l < L$, $B^l_{i,j}$ can be computed on the fly by only using column and row numbers of the original matrix A.]

Step 3

Application

Comment [During application, the matrix A that was decomposed during precomputation in Step 2 is applied to an input vector x to generate an output vector y.]

```
do $l = 0, 1, \ldots, L$
    do $i = 1, 2, \ldots, 2^l$
        do $j = 1, 2, \ldots, 2^{L-l}$
            if $l = 0$ then calculate and store
```

$$u^0_{1,j} = P^0_{1,j} \, x^L_j. \tag{4.19}$$

```
            if $l > 0$ then calculate and store
```

$$u^l_{i,j} = P^l_{i,j} \begin{pmatrix} u^{l-1}_{\lfloor \frac{i+1}{2} \rfloor, 2j-1} \\ u^{l-1}_{\lfloor \frac{i+1}{2} \rfloor, 2j} \end{pmatrix}. \tag{4.20}$$

```
            if $l = L$ then calculate and store
```

$$y^L_i = B^L_{i,1} \, u^L_{i,1}. \tag{4.21}$$

```
        end do
    end do
end do
Comment [The vectors $y_1^L, \ldots, y_{2^L}^L$ can then be concatenated to form the vector $y \approx Ax$, which is accurate to relative precision $\varepsilon$.]
```

4.4 Complexity

This section contains an operation count for applying the algorithm described in Section 4.3 to the matrix A. A brief analysis of the algorithmic complexity is given in Table 3.

Remark 4.3 We claim in Table 3 that during Step 2, each of the interpolative decompositions performed can be computed in $\mathcal{O}(n)$ operations. Recalling Observation 2.3, the interpolative decomposition of an $m \times l$ matrix of rank k can be performed in $\mathcal{O}(kml\,\log l)$ operations. Since the ranks of all submatrices on all levels of A are virtually the same, $\tilde{k}$, we have that each interpolative decomposition can be computed in $\mathcal{O}(2\tilde{k}^2\,\frac{n}{2^d}\,\log 2\tilde{k}) = \mathcal{O}(n)$ operations, where d is the level of the matrix being decomposed.

Observation 4.4 If the matrix A is non-singular to precision ε (as is the case for orthogonal polynomial transform matrices), we can avoid the precomputation of level 0 since the interpolative decomposition of matrices on that level will yield $P_{1,j}^0 = I_{k_{1,j}^0}$, where I_k denotes a $k \times k$ identity matrix and $j = 1, 2, \ldots, 2^L$.

Step	Purpose	Operation count	Explanation
1	Initialization	$\mathcal{O}(n\,\log n)$	No numerical computations are performed, but the dyadic hierarchy is created and parameters are set.
2	Precomputation compressions	$\mathcal{O}(n^2\,\log n)$	On each of the $\mathcal{O}(\log n)$ levels, $\mathcal{O}(n)$ interpolative decompositions must be performed. Each decomposition can be performed in $\mathcal{O}(n)$ operations.
3	Application of the transform to a vector	$\mathcal{O}(n\,\log n)$	On each of the $\mathcal{O}(\log n)$ levels, $\mathcal{O}(n)$ interpolation P matrices must be applied, each of bounded size. Then, on level L an additional $\mathcal{O}(n)$ column skeleton B matrices must be applied, each of bounded size.

Table 3: Complexity of the algorithm.

Another important characteristic of any numerical algorithm is the storage requirement. Table 4 gives an overview of the storage requirements for our algorithm. We ignore requirements for work arrays, and assume that we do not need to store the matrix A in memory, but that its elements can be computed when needed.

Observation 4.5 For $l > 0$, $i = 1, 2, \ldots, 2^l$, and $j = 1, 2, \ldots, 2^{L-l}$, only the vectors $u^{l-1}_{\lfloor\frac{i+1}{2}\rfloor,2j-1}$ and $u^{l-1}_{\lfloor\frac{i+1}{2}\rfloor,2j}$ are used to compute $u^l_{i,j}$ in Step 3. Thus, the storage requirement for the application of matrix A in Step 3 can be relaxed to $\mathcal{O}(n)$.

Step	Purpose	Storage needed	Explanation
1	Initialization	$\mathcal{O}(n \log n)$	Only the dyadic hierarchy and parameters must be stored.
2	Precomputation compressions	$\mathcal{O}(n \log n)$	On each of the $\mathcal{O}(\log n)$ levels, $\mathcal{O}(n)$ interpolation P matrices must be stored, each of bounded size. On level L, an additional $\mathcal{O}(n)$ column skeleton B matrices must be stored, each of bounded size.
3	Application of the transform to a vector	$\mathcal{O}(n \log n)$	On each of the $\mathcal{O}(\log n)$ levels, we store the vectors $u^l_{i,j}$. This requires $\mathcal{O}(n)$ memory for each level. On level L, an additional $\mathcal{O}(n)$ amount of storage is required to calculated and store the vectors y^l_i.

Table 4: Storage requirements of the algorithm.

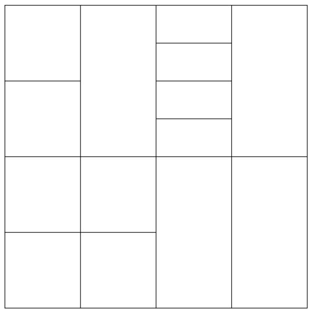

Figure 3: One possible adaptive level 1 of A.

4.5 An adaptive algorithm

The algorithm described in Sections 4.2 and 4.4 is based on the assumption that the rank of two different contiguous submatrices is the same if they have an equal number of elements. In practice, it is possible for this rank to fluctuate. To avoid calculating an interpolative decomposition for a submatrix whose rank is too large, one can implement an adaptive regime to dyadically partition the rows as much as needed on a given level. This results in accelerated matrix application times (at the cost of possibly longer precomputation times).

To implement this optimization, we require that the maximum allowable rank of any submatrix on any level be at most some constant k_{max}. Previously, it was only possible to have levels of the matrix that looked like those in Figures 1 and 2. Now, by insisting that the maximum allowable rank per submatrix is at most k_{max}, it is possible to have levels resembling that in Figure 3. The columns of the matrix in Figure 3 are still dyadically partitioned, but the rows have been adaptively dyadically partitioned, depending on the actual rank of submatrices. As the level increases, the merging of columns and subdivision of rows continues appropriately with respect to the adaptive hierarchy.

Observation 4.6 With the proper choice of k_{max}, we have yet to find a case in which the adaptive algorithm applies a matrix slower than the non-adaptive algorithm. It has not been determined whether this is due to an actual decrease in the complexity of the algorithm or issues related to on-board caching.

5 Numerical examples

The butterfly algorithm has been implemented in FORTRAN, and this section reports numerical results obtained with our implementation. In all cases, we used the Lahey/Fujitsu compiler with optimization flag `--o2`, and all examples were run on a 2.4 GHz Intel Pentium 4 processor with 1 GB of RAM and an L2 cache of 512 KB. CPU times listed are in seconds and all operations were performed using double precision arithmetic. All examples (except where otherwise noted) reflect the adaptive version of the algorithm as discussed in Section 4.5.

The columns labeled "n" list the size of the matrices to which the algorithm of this paper was applied. All matrices in these examples are square.

The columns labeled "precomputation" list the times taken to decompose the said $n \times n$ matrix.

The columns labeled "direct eval" list the times taken for a direct $n \times n$ matrix-vector multiplication. Times in parenthesis are estimates.

The columns labeled "fast eval" list the times taken to apply the $n \times n$ matrix using the algorithm of this paper.

The columns labeled "l^2 error" contain the relative errors between the solution obtained via the algorithm and the solution obtained using a direct matrix-vector multiplication.

The columns labeled "MB used" list the amount of memory in megabytes required by the algorithm for precomputation and evaluation.

In each example, the matrix was applied to a normalized vector containing random numbers uniformly distributed on the interval $[0, 1]$. The value of k_{max} was chosen so as to produce optimal run-times for size $n = 2048$. To find this value of k_{max}, the adaptive algorithm was run in each example for size $n = 2048$ and $k_{max} = 30, 32, 34, \ldots, 138, 140$; the value that gave the best evaluation time was chosen. The value of k_{max} used is given below each table, and $\varepsilon = 10^{-10}$ unless otherwise stated. No effort was made to optimize the precomputation required. The precomputation scheme used costs $\mathcal{O}(n^2 \log n)$, as described in Section 4.4.

5.1 Fourier-Bessel transforms

Tables 5–9 document results of applying our algorithm to Fourier-Bessel transform matrices T_{J_ν} in equation (2.27) with $\nu = 0, 1, 100$, and 10000. Table 10 contains data for varying ε in the case of $\nu = 0$ and $n = 8192$. In all examples $R = 1$, where R is such that $\rho_1, \rho_2, \rho_3, \ldots$ are the real increasing positive zeros of the function $g : \mathbb{R}^+ \to \mathbb{R}$,

$$g(\rho) = J_\nu(2\pi\rho R) \tag{5.1}$$

(see equations (2.15–2.27)). The data listed in Tables 5–9 are illustrated in Figures 4–10, respectively.

Observation 5.1 An examination of the data contained in Tables 5–9 shows that the algorithm of this paper has consistent performance across all orders of transforms. The "fast eval" times are *virtually independent* of the order of the Fourier-Bessel transform matrix being applied. This observation is illustrated in Table 9, where for a matrix of size n we choose $\nu = n$. For large values of n, the "fast eval" times increase almost linearly. Figure 9 illustrates data from Tables 5–9 in one plot.

Tables 11 and 12 contain results from applying the algorithm to the matrices E_J in equation (2.30) and E_H in equation (2.31), respectively. In both matrices E_J and E_H, $x_1, x_2, \ldots, x_n$ are given by

$$x_j = n + \frac{2\pi}{3}(j-1). \tag{5.2}$$

As a consequence of Lemma 2.6, choosing $x_1, x_2, \ldots, x_n$ as in equation (5.2) ensures that all computed values of Bessel functions and Hankel functions are properly scaled.

Remark 5.2 Tables 11 and 12 reflect results from applying the algorithm to the matrices E_J^t and E_H^t, respectively. This was done to greatly reduce extended precomputation times due to the calculation of Bessel and Hankel functions using their recurrence relations. As mentioned in Observation 4.2, applying the algorithm to E_J^t and E_H^t yields results similar to those which would have been obtained from applying the algorithm to E_J and E_H.

n	Precomputation	Direct eval	Fast eval	l^2 error	MB used
256	.32E+00	.24E-03	.22E-03	.93E-12	.45E+00
512	.86E+00	.12E-02	.88E-03	.23E-11	.13E+01
1024	.34E+01	.50E-02	.24E-02	.15E-11	.36E+01
2048	.15E+02	.20E-01	.64E-02	.29E-11	.93E+01
4096	.70E+02	.81E-01	.16E-01	.21E-11	.23E+02
8192	.32E+03	.32E+00	.38E-01	.23E-11	.56E+02
16384	.15E+04	(.13E+01)	.91E-01	.22E-11	.13E+03
32768	.65E+04	(.52E+01)	.21E+00	.23E-11	.31E+03
65536	.28E+05	(.21E+02)	.50E+00	.11E-11	.72E+03

Table 5: Times and errors for Fourier-Bessel transform with $\nu = 0$ and $k_{max} = 70$.

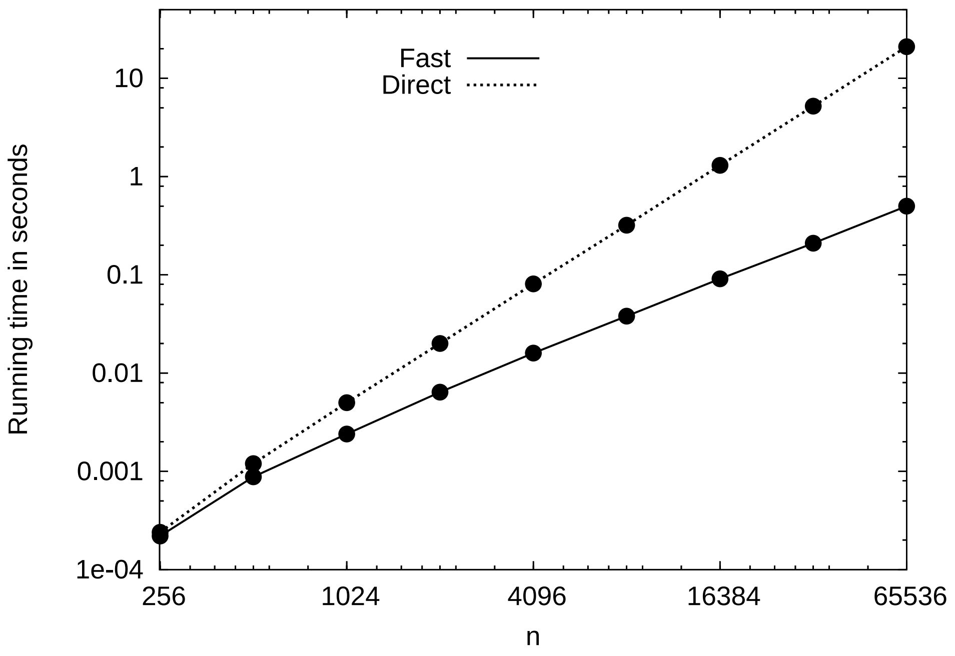

Figure 4: Comparison of the accelerated Fourier-Bessel transform of order 0 and its direct calculation.

n	Precomputation	Direct eval	Fast eval	l^2 error	MB used
256	.35E+00	.24E-03	.21E-03	.26E-11	.45E+00
512	.92E+00	.12E-02	.88E-03	.31E-11	.13E+01
1024	.35E+01	.50E-02	.24E-02	.21E-11	.36E+01
2048	.15E+02	.20E-01	.64E-02	.23E-11	.92E+01
4096	.72E+02	.81E-01	.16E-01	.20E-11	.23E+02
8192	.32E+03	.32E+00	.38E-01	.19E-11	.56E+02
16384	.14E+04	(.13E+01)	.90E-01	.30E-11	.13E+03
32768	.65E+04	(.52E+01)	.21E+00	.25E-11	.31E+03
65536	.29E+05	(.21E+02)	.50E+00	.21E-11	.72E+03

Table 6: Times and errors for Fourier-Bessel transform with $\nu = 1$ and $k_{max} = 72$.

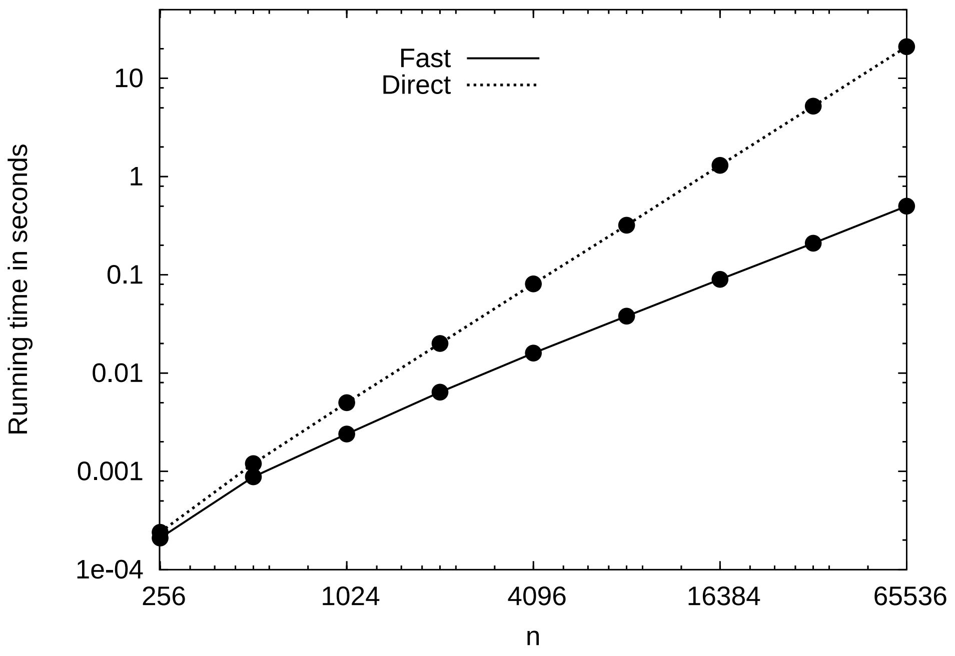

Figure 5: Comparison of the accelerated Fourier-Bessel transform of order 1 and its direct calculation.

n	Precomputation	Direct eval	Fast eval	l^2 error	MB used
256	.36E+00	.24E-03	.18E-03	.47E-11	.42E+00
512	.11E+01	.12E-02	.84E-03	.72E-11	.13E+01
1024	.43E+01	.50E-02	.24E-02	.11E-10	.35E+01
2048	.19E+02	.20E-01	.62E-02	.12E-10	.91E+01
4096	.88E+02	.81E-01	.16E-01	.14E-10	.23E+02
8192	.40E+03	.32E+00	.38E-01	.11E-10	.56E+02
16384	.18E+04	(.13E+01)	.91E-01	.97E-11	.13E+03
32768	.77E+04	(.52E+01)	.21E+00	.12E-10	.31E+03
65536	.34E+05	(.21E+02)	.50E+00	.12E-10	.73E+03

Table 7: Times and errors for Fourier-Bessel transform with $\nu = 100$ and $k_{max} = 74$.

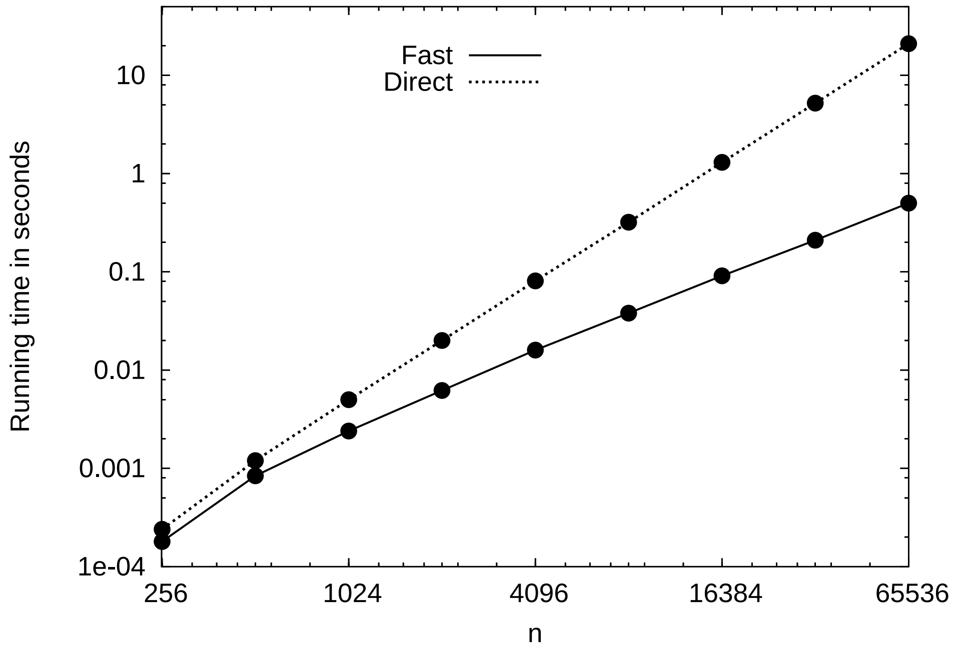

Figure 6: Comparison of the accelerated Fourier-Bessel transform of order 100 and its direct calculation.

n	Precomputation	Direct eval	Fast eval	l^2 error	MB used
256	.22E+00	.24E-03	.54E-04	.22E-11	.15E+00
512	.64E+00	.12E-02	.32E-03	.46E-11	.55E+00
1024	.25E+01	.50E-02	.14E-02	.14E-10	.21E+01
2048	.12E+02	.20E-01	.45E-02	.22E-10	.65E+01
4096	.62E+02	.81E-01	.13E-01	.29E-10	.19E+02
8192	.31E+03	.32E+00	.34E-01	.26E-10	.50E+02
16384	.15E+04	(.13E+01)	.84E-01	.31E-10	.12E+03
32768	.71E+04	(.52E+01)	.20E+00	.32E-10	.30E+03
65536	.32E+05	(.21E+02)	.48E+00	.29E-10	.71E+03

Table 8: Times and errors for Fourier-Bessel transform with $\nu = 10000$ and $k_{max} = 80$.

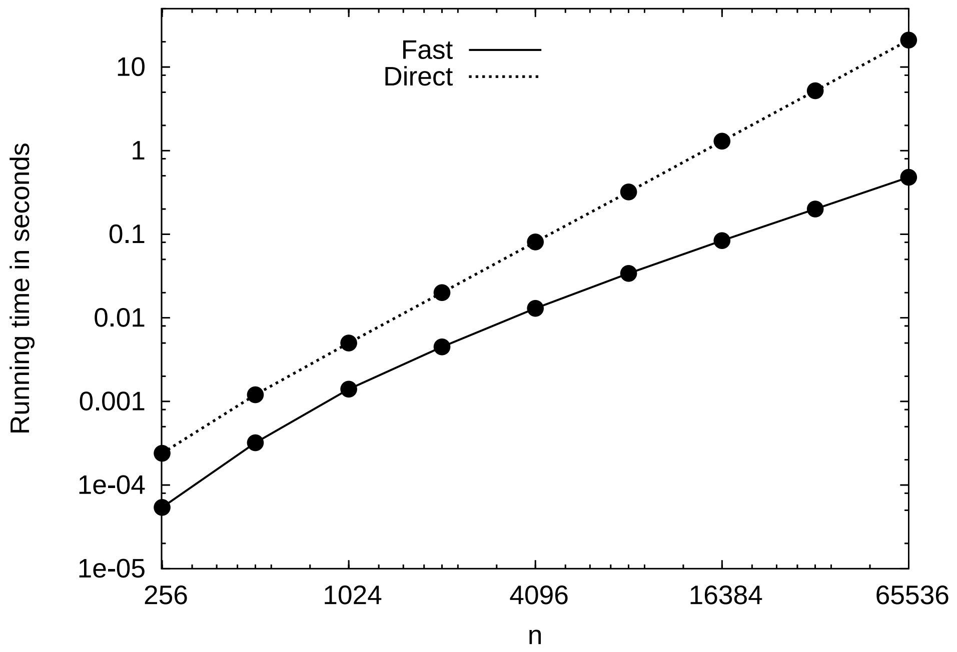

Figure 7: Comparison of the accelerated Fourier-Bessel transform of order 10000 and its direct calculation.

n	Precomputation	Direct eval	Fast eval	l^2 error	MB used
256	.36E+00	.24E-03	.16E-03	.76E-11	.40E+00
512	.97E+00	.12E-02	.75E-03	.14E-10	.11E+01
1024	.39E+01	.50E-02	.21E-02	.17E-10	.31E+01
2048	.17E+02	.20E-01	.57E-02	.21E-10	.85E+01
4096	.77E+02	.81E-01	.15E-01	.25E-10	.22E+02
8192	.35E+03	.32E+00	.35E-01	.27E-10	.52E+02
16384	.16E+04	(.13E+01)	.85E-01	.34E-10	.13E+03
32768	.68E+04	(.52E+01)	.19E+00	.33E-10	.29E+03
65536	.29E+05	(.21E+02)	.46E+00	.38E-10	.69E+03

Table 9: Times and errors for Fourier-Bessel transform with $\nu = n$ and $k_{max} = 94$.

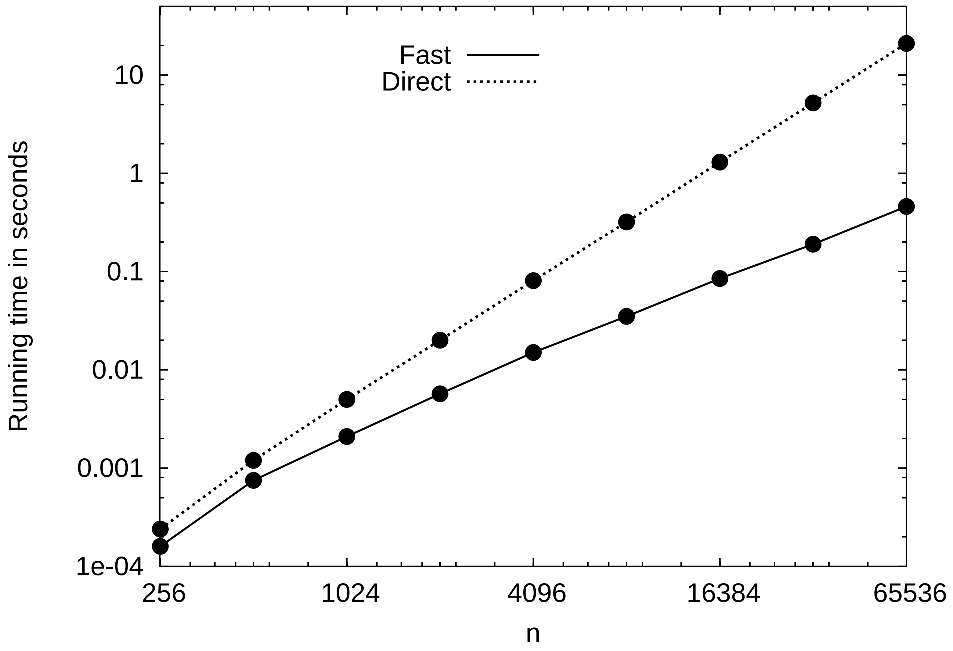

Figure 8: Comparison of the accelerated Fourier-Bessel transform of order n and its direct calculation.

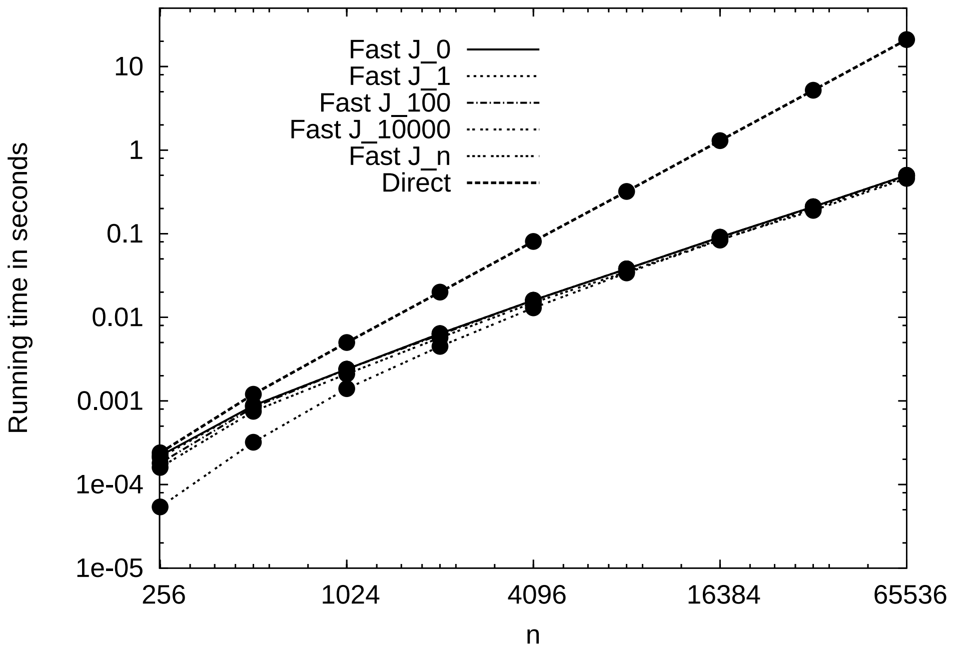

Figure 9: Comparison of accelerated Fourier-Bessel transforms of many orders and their direct calculation.

N	ε	Precomputation	Direct eval	Fast eval	l^2 error	MB used
8192	10^{-3}	.24E+03	.32E+00	.25E-01	.19E-04	.35E+02
8192	10^{-4}	.26E+03	.32E+00	.27E-01	.21E-05	.38E+02
8192	10^{-5}	.27E+03	.32E+00	.29E-01	.19E-06	.41E+02
8192	10^{-6}	.28E+03	.32E+00	.31E-01	.20E-07	.44E+02
8192	10^{-7}	.30E+03	.32E+00	.33E-01	.24E-08	.47E+02
8192	10^{-8}	.31E+03	.32E+00	.35E-01	.22E-09	.50E+02
8192	10^{-9}	.33E+03	.32E+00	.36E-01	.17E-10	.53E+02
8192	10^{-10}	.34E+03	.32E+00	.38E-01	.23E-11	.56E+02
8192	10^{-11}	.35E+03	.32E+00	.41E-01	.22E-12	.60E+02
8192	10^{-12}	.37E+03	.32E+00	.57E-01	.16E-13	.80E+02

Table 10: Times and errors for Fourier-Bessel transform with $\nu = 0$, $k_{max} = 70$, and $\varepsilon = \{10^{-3}, \ldots, 10^{-12}\}$.

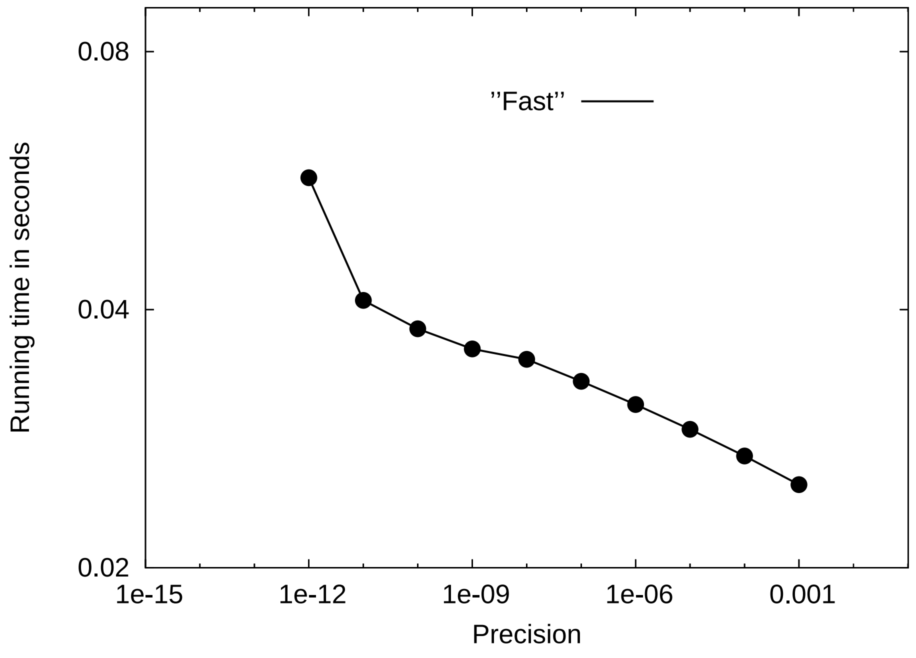

Figure 10: Comparison of the accelerated Fourier-Bessel transforms of order 0 for varying precision.

n	Precomputation	Direct eval	Fast eval	l^2 error	MB used
256	.35E+00	.24E-03	.75E-04	.41E-10	.22E+00
512	.67E+00	.12E-02	.26E-03	.54E-10	.52E+00
1024	.20E+01	.50E-02	.83E-03	.51E-10	.13E+01
2048	.75E+01	.20E-01	.19E-02	.44E-10	.28E+01
4096	.31E+02	.81E-01	.44E-02	.45E-10	.67E+01
8192	.13E+03	.32E+00	.95E-02	.49E-10	.15E+02
16384	.59E+03	(.13E+01)	.21E-01	.52E-10	.34E+02
32768	.26E+04	(.52E+01)	.46E-01	.66E-10	.79E+02
65536	.12E+05	(.21E+02)	.10E+00	.73E-10	.20E+03

Table 11: Times and errors for evaluating Bessel function expansions with $k_{max} = 72$.

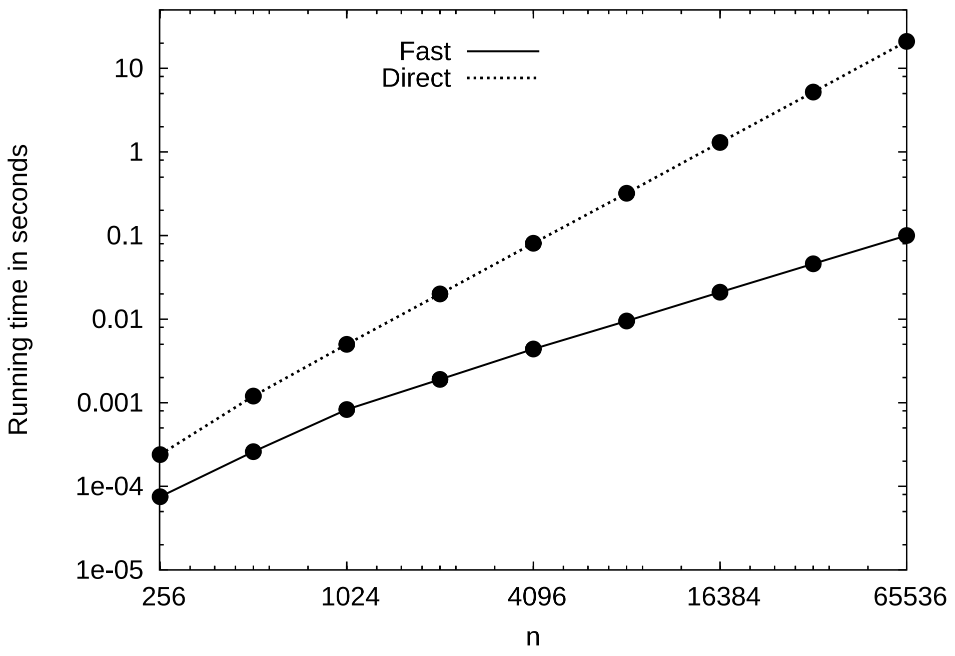

Figure 11: Comparison of the accelerated evaluation of Bessel function expansions and their direct evaluation.

n	Precomputation	Direct eval	Fast eval	l^2 error	MB used
256	.36E+00	.67E-03	.11E-03	.25E-10	.24E+00
512	.76E+00	.27E-02	.34E-03	.86E-10	.54E+00
1024	.20E+01	.11E-01	.93E-03	.43E-10	.13E+01
2048	.62E+01	.43E-01	.19E-02	.43E-10	.27E+01
4096	.22E+02	.18E+00	.44E-02	.51E-10	.64E+01
8192	.94E+02	(.72E+00)	.92E-02	.49E-10	.15E+02
16384	.39E+03	(.29E+01)	.20E-01	.52E-10	.37E+02
32768	.17E+04	(.12E+02)	.42E-01	.56E-10	.10E+03
65536	.77E+04	(.48E+02)	.89E-01	.55E-10	.31E+03

Table 12: Times and errors for evaluating Hankel function expansions with $k_{max} = 38$.

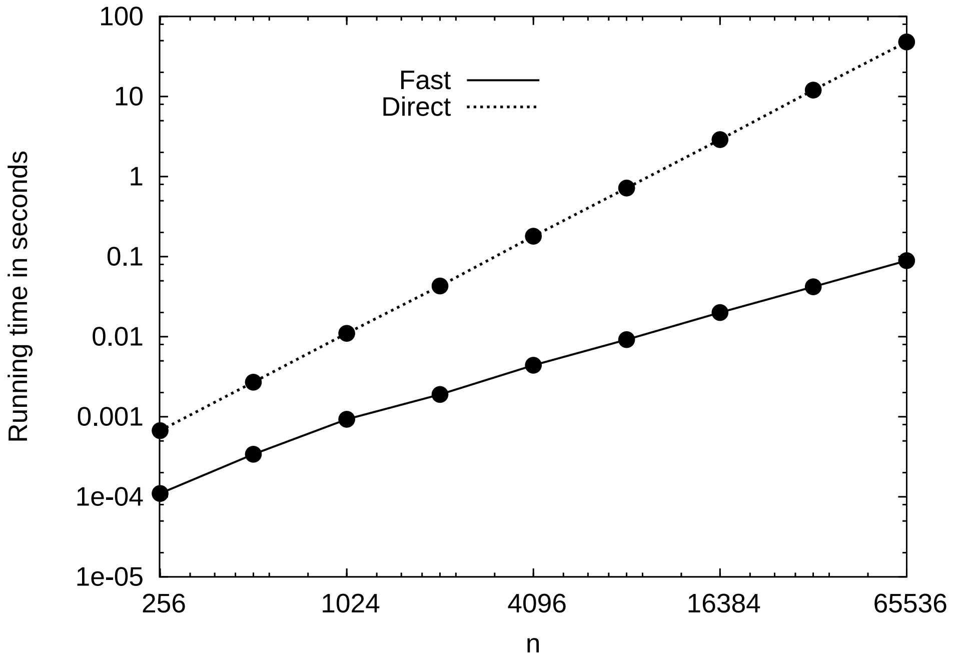

Figure 12: Comparison of the the accelerated evaluation of Hankel function expansions and their direct evaluation.

5.2 Fourier transforms

Tables 13 and 14 document results obtained from applying the algorithm of this paper to the matrix T_F in equation (2.45). Table 13 contains results from applying our algorithm to the standard equispaced discrete Fourier transform matrix T_F. Table 14 contains results from applying the algorithm to the matrix T_F where $x_1 < x_2 < \cdots < x_n$ are random numbers uniformly distributed on the interval $[-\pi, \pi]$ and $\omega_1 < \omega_2 < \cdots < \omega_n$ are random numbers uniformly distributed on the interval $[-\frac{n}{2}, \frac{n}{2}]$. The data listed in Tables 13 and 14 are illustrated in Figures 13 and 14, respectively.

Observation 5.3 As a "numerical proof" of Corollary 3.4, for $n = 16384$ we examined the numerical rank of each contiguous submatrix of the discrete Fourier transform matrix T_F which was decomposed using the non-adaptive regime of the algorithm. For $C_{max} = 64$, the numerical ranks ranged from 85 to 88 (except on the finest level where matrices were dimensioned $64 \times n$). This observation is consistent with Corollary 3.4; numerical ranks of $m \times k$ contiguous submatrices depend only on the quantity mk. For large values of n, the "fast eval" times increase almost linearly.

n	Precomputation	Direct eval	Fast eval	l^2 error	MB used
256	.32E+00	.67E-03	.65E-03	.34E-11	.95E+00
512	.13E+01	.27E-02	.18E-02	.91E-11	.26E+01
1024	.60E+01	.11E-01	.48E-02	.11E-10	.67E+01
2048	.28E+02	.43E-01	.12E-01	.12E-10	.16E+02
4096	.13E+03	.18E+00	.28E-01	.15E-10	.39E+02
8192	.56E+03	(.72E+00)	.65E-01	.21E-10	.90E+02
16384	.26E+04	(.29E+01)	.15E+00	.24E-10	.20E+03
32768	.12E+05	(.12E+02)	.34E+00	.23E-10	.46E+03

Table 13: Times and errors for discrete Fourier transform with $k_{max} = 74$.

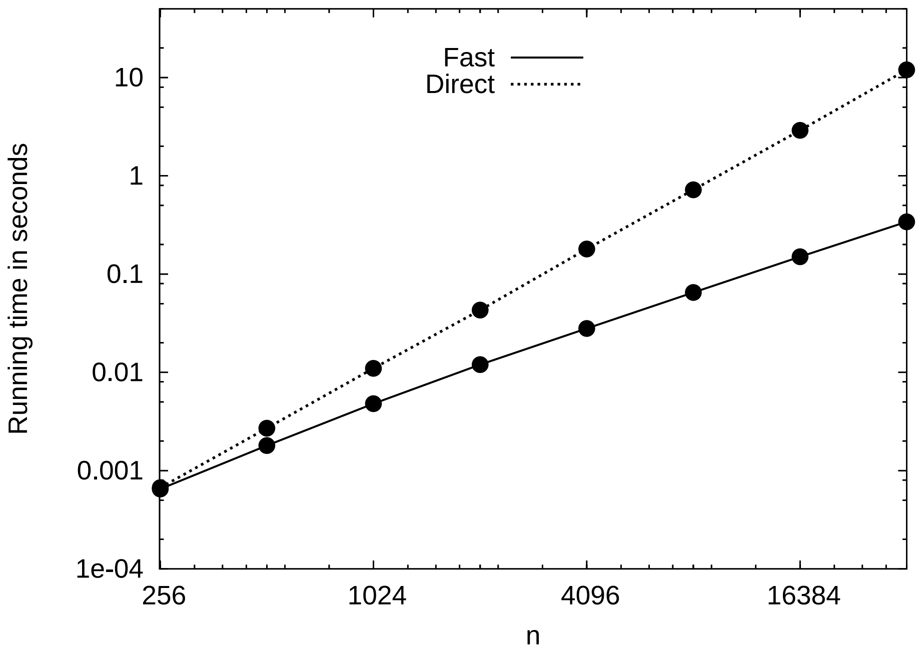

Figure 13: Comparison of the accelerated discrete Fourier transform and its direct calculation.

n	Precomputation	Direct eval	Fast eval	l^2 error	MB used
256	.34E+00	.67E-03	.62E-03	.76E-09	.91E+00
512	.13E+01	.27E-02	.18E-02	.84E-08	.25E+01
1024	.62E+01	.11E-01	.47E-02	.17E-08	.65E+01
2048	.29E+02	.43E-01	.12E-01	.27E-09	.16E+02
4096	.13E+03	.18E+00	.28E-01	.72E-09	.38E+02
8192	.62E+03	(.72E+00)	.65E-01	.47E-09	.89E+02
16384	.25E+04	(.29E+01)	.15E+00	.79E-09	.20E+03
32768	.11E+05	(.12E+02)	.33E+00	.15E-08	.46E+03

Table 14: Times and errors for discrete Fourier transform for nonequispaced data with $k_{max} = 74$.

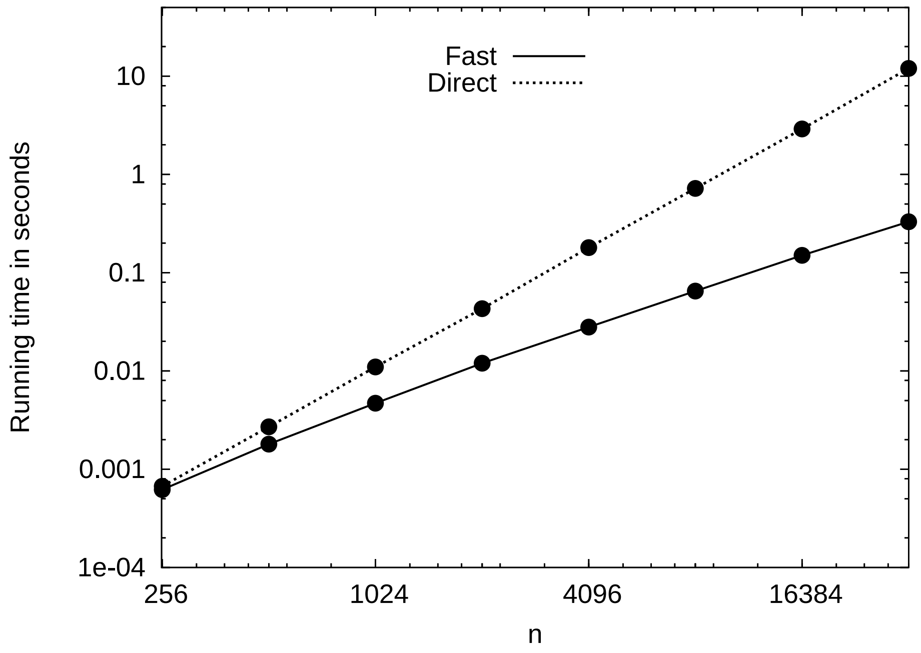

Figure 14: Comparison of the accelerated discrete Fourier transform for nonequispaced data and its direct calculation.

5.3 Legendre polynomial transforms

Table 15 contains results for using our algorithm to apply the Legendre transform matrix T_P in equation (2.66). Figure 15 illustrates the data listed in Table 15. Table 16 contains a comparison between the adaptive and the non-adaptive schemes of the algorithm when used to apply the matrix T_P. Columns denoted with an "A" list results obtained from the adaptive scheme. Columns denoted with an "NA" list results obtained from the non-adaptive scheme.

Observation 5.4 Despite an increase in precomputation times, the adaptive version of the algorithm results in faster evaluation times than the non-adaptive version for every size n, even though the parameter k_{max} was chosen only to produce an optimal evaluation time for $n = 2048$. It is conceivable that for each n, k_{max} could be chosen to further improve evaluation times.

Observation 5.5 The algorithm of this paper, non-adaptive as well as adaptive schemes, can also be used as a technique for matrix compression. Not only does the algorithm enable the $\mathcal{O}(n \log n)$ application of an $n \times n$ matrix, but, if the only purpose for storing the matrix is to apply it at a later time, we only require $\mathcal{O}(n \log n)$ memory for storage. For example, to directly store a 65536×65536 matrix we require $2^{32} = 4,294,967,296 \approx .43\text{E}+10$ real *8 words of memory. Using this algorithm, we can compress and apply the same size Legendre transform matrix T_P to 10-digit precision using only .92E+08 real *8 words of memory. Memory usage was not fully optimized, as this was not the main goal of the paper. Even so, we observed a decrease in memory requirement by a factor of almost 50.

5.4 Chebyshev polynomial transforms

Table 17 contains the results for using the algorithm of this paper to apply the matrix T_T in equation (2.86). Figure 16 illustrates the data listed in Table 17.

5.5 Hermite function transforms

Table 18 contains the results for using the algorithm of this paper to apply the matrix T_H in equation (2.110). Figure 17 illustrates the data listed in Table 18.

5.6 Laguerre function transforms

Table 19 contains the results for using the algorithm of this paper to apply the matrix T_L in equation (2.130). Figure 18 illustrates the data listed in Table 19.

Observation 5.6 An examination of Tables 15, 17–19 shows that our algorithm gives consistent application times for every size n independently of which orthogonal polynomial transform matrix is used. This is illustrated in Figure 19.

5.7 Prolate spheroidal wave function interpolation

Table 20 contains the results for using our algorithm to apply the matrix E_ψ^c in equation (2.142) with $c = 8500.5$ and $x_1, x_2, \ldots, x_n$ the roots of the function ψ_n^c. Figure 20 illustrates the data listed in Table 20.

n	Precomputation	Direct eval	Fast eval	l^2 error	MB used
256	.38E+00	.24E-03	.24E-03	.69E-11	.43E+00
512	.90E+00	.12E-02	.94E-03	.11E-10	.13E+01
1024	.33E+01	.50E-02	.26E-02	.10E-10	.35E+01
2048	.14E+02	.20E-01	.68E-02	.80E-11	.90E+01
4096	.66E+02	.81E-01	.17E-01	.82E-11	.23E+02
8192	.31E+03	.32E+00	.41E-01	.92E-11	.55E+02
16384	.14E+04	(.13E+01)	.98E-01	.93E-11	.13E+03
32768	.65E+04	(.52E+01)	.23E+00	.92E-11	.30E+03
65536	.28E+05	(.21E+02)	.54E+00	.11E-10	.71E+03

Table 15: Times and errors for Legendre polynomial transform with $k_{max} = 70$.

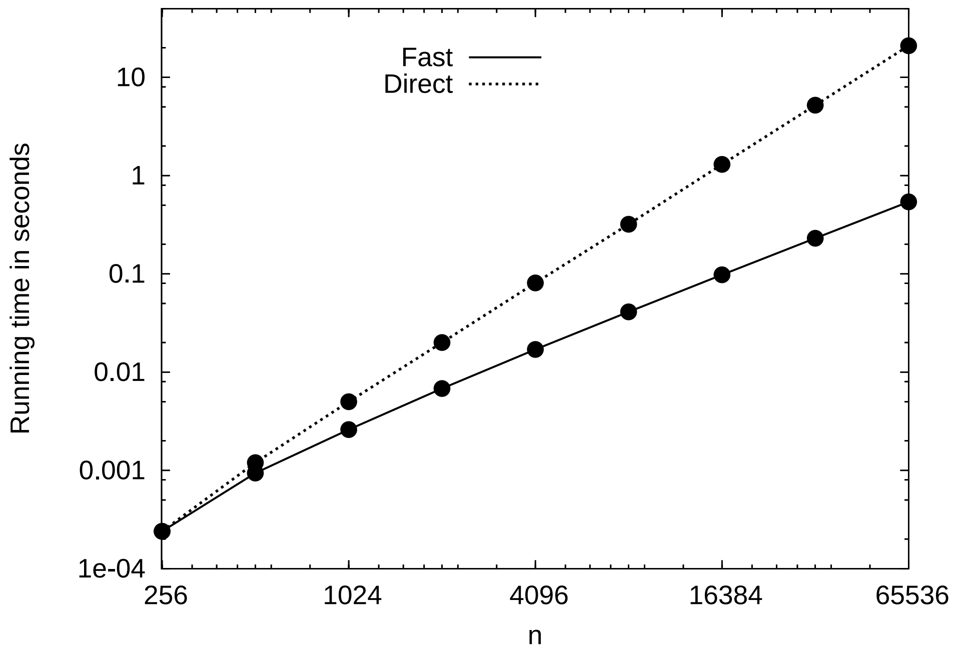

Figure 15: Comparison of the accelerated Legendre polynomial transform and its direct calculation.

n	Precomp NA	Precomp A	Direct eval	Fast eval NA	Fast eval A	l^2 error NA	l^2 error A
256	.29E+00	.29E+00	.24E-03	.20E-03	.19E-03	.51E-11	.69E-11
512	.65E+00	.76E+00	.12E-02	.88E-03	.87E-03	.53E-11	.11E-10
1024	.25E+01	.29E+01	.50E-02	.25E-02	.24E-02	.84E-11	.10E-10
2048	.12E+02	.13E+02	.20E-01	.65E-02	.63E-02	.75E-11	.80E-11
4096	.53E+02	.63E+02	.81E-01	.16E-01	.16E-01	.11E-10	.82E-11
8192	.25E+03	.30E+03	.32E+00	.40E-01	.38E-01	.10E-10	.92E-11
16384	.11E+04	.14E+04	(.13E+01)	.94E-01	.90E-01	.10E-10	.93E-11
32768	.44E+04	.64E+04	(.52E+01)	.22E+00	.21E+00	.96E-11	.92E-11
65536	.20E+05	.29E+05	(.21E+02)	.50E+00	.49E+00	.11E-10	.11E-10

Table 16: Comparison of adaptive scheme and non-adaptive scheme for computing Legendre polynomial transform. In the adaptive case $k_{max} = 70$. In the non-adaptive case $C_{max} = 64$.

n	Precomputation	Direct eval	Fast eval	l^2 error	MB used
256	.41E+00	.24E-03	.22E-03	.76E-11	.45E+00
512	.85E+00	.12E-02	.90E-03	.72E-11	.14E+01
1024	.31E+01	.50E-02	.25E-02	.16E-10	.36E+01
2048	.13E+02	.20E-01	.64E-02	.16E-10	.93E+01
4096	.61E+02	.81E-01	.16E-01	.22E-10	.23E+02
8192	.28E+03	.32E+00	.38E-01	.22E-10	.56E+02
16384	.13E+04	(.13E+01)	.91E-01	.23E-10	.13E+03
32768	.56E+04	(.52E+01)	.21E+00	.23E-10	.31E+03
65536	.25E+05	(.21E+02)	.50E+00	.25E-10	.72E+03

Table 17: Times and errors for Chebyshev polynomial transform with $k_{max} = 76$.

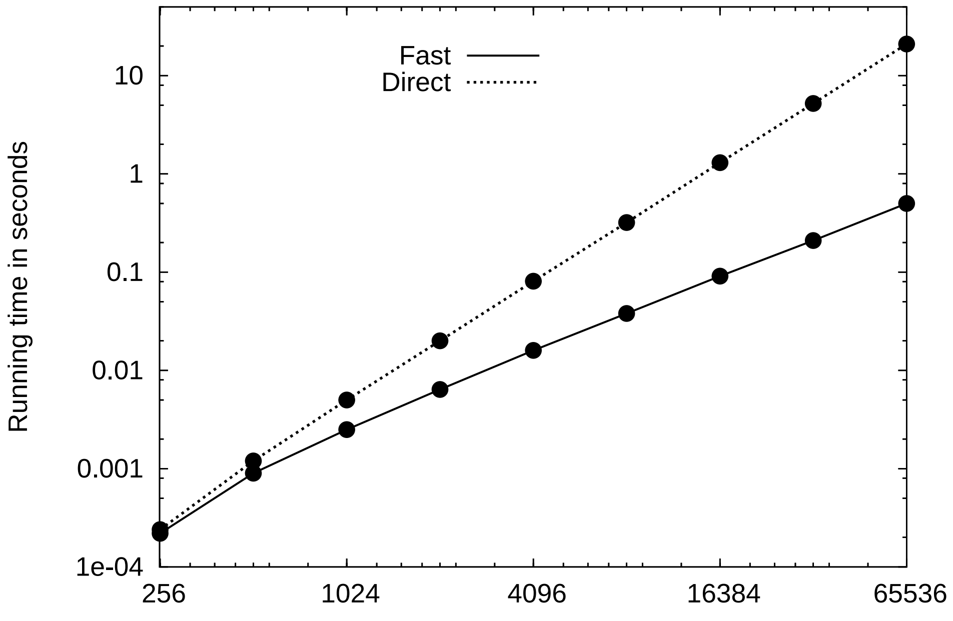

Figure 16: Comparison of the accelerated Chebyshev polynomial transform and its direct calculation.

n	Precomputation	Direct eval	Fast eval	l^2 error	MB used
256	.44E+00	.24E-03	.18E-03	.38E-11	.45E+00
512	.12E+01	.12E-02	.85E-03	.15E-10	.13E+01
1024	.49E+01	.50E-02	.23E-02	.22E-10	.35E+01
2048	.24E+02	.20E-01	.61E-02	.28E-10	.91E+01
4096	.13E+03	.81E-01	.15E-01	.33E-10	.23E+02
8192	.63E+03	.32E+00	.37E-01	.34E-10	.54E+02
16384	.32E+04	(.13E+01)	.88E-01	.38E-10	.13E+03
32768	.15E+05	(.52E+01)	.20E+00	.46E-10	.30E+03
65536	.66E+05	(.21E+02)	.47E+00	.51E-10	.71E+03

Table 18: Times and errors for Hermite function transform with $k_{max} = 90$.

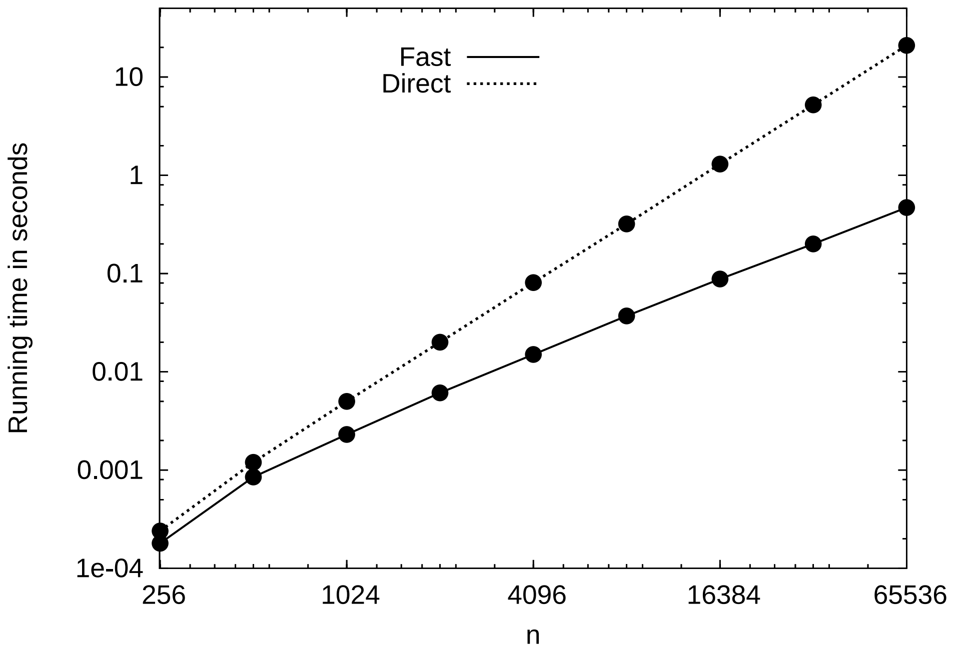

Figure 17: Comparison of the accelerated Hermite function transform and its direct calculation.

n	Precomputation	Direct eval	Fast eval	l^2 error	MB used
256	.48E+00	.24E-03	.18E-03	.18E-10	.43E+00
512	.12E+01	.12E-02	.82E-03	.50E-10	.12E+01
1024	.52E+01	.50E-02	.23E-02	.86E-10	.34E+01
2048	.27E+02	.20E-01	.62E-02	.17E-09	.89E+01
4096	.14E+03	.81E-01	.15E-01	.17E-09	.22E+02
8192	.68E+03	.32E+00	.36E-01	.28E-09	.53E+02
16384	.33E+04	(.13E+01)	.86E-01	.32E-09	.12E+03
32768	.15E+05	(.52E+01)	.20E+00	.40E-09	.29E+03
65536	.65E+05	(.21E+02)	.47E+00	.60E-09	.69E+03

Table 19: Times and errors for Laguerre function transform with $k_{max} = 78$.

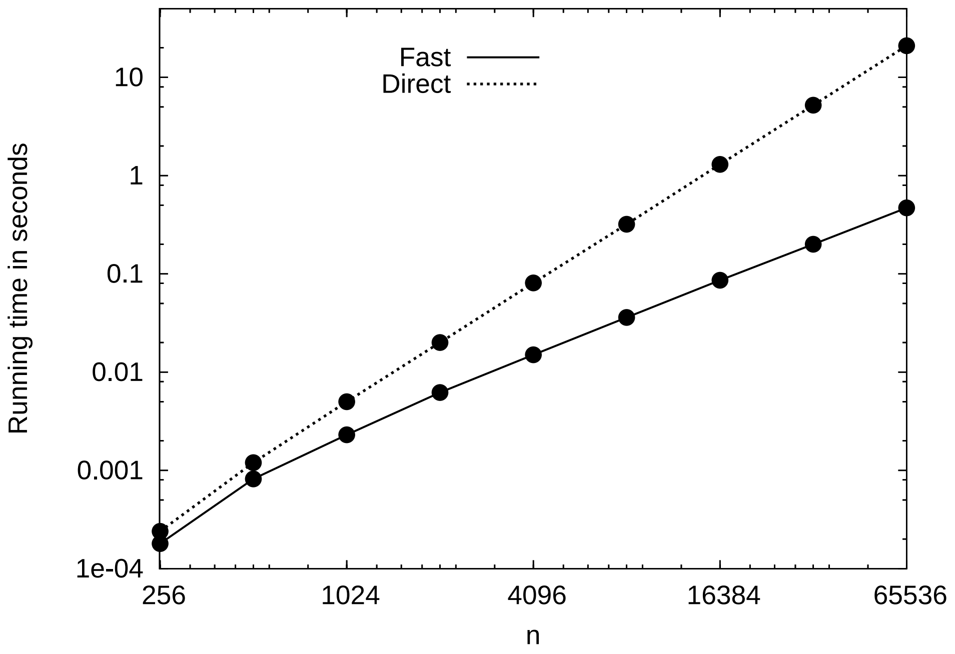

Figure 18: Comparison of the accelerated Laguerre function transform and its direct calculation.

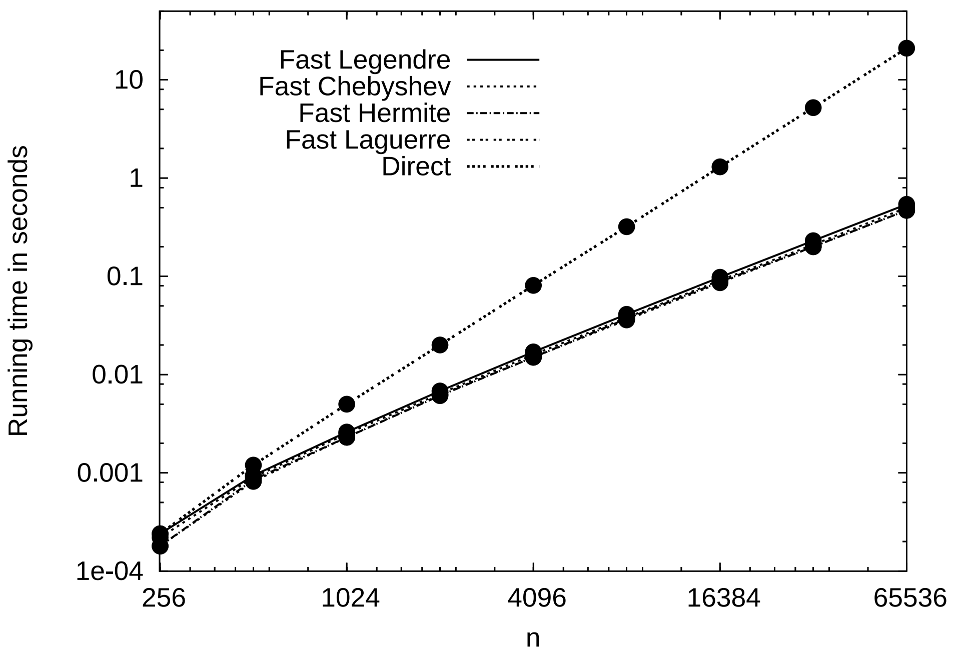

Figure 19: Comparison of the accelerated Legendre, Chebyshev, Hermite, and Laguerre transforms and their direct calculation.

n	Precomputation	Direct eval	Fast eval	l^2 error	MB used
256	.21E+01	.24E-03	.20E-03	.11E-11	.45E+00
512	.12E+02	.12E-02	.86E-03	.18E-10	.13E+01
1024	.75E+02	.50E-02	.24E-02	.18E-10	.35E+01
2048	.43E+03	.20E-01	.71E-02	.27E-10	.10E+02
4096	.25E+04	.81E-01	.20E-01	.28E-10	.30E+02
8192	.14E+05	.32E+00	.70E-01	.52E-11	.10E+03

Table 20: Times and errors for evaluating prolate spheroidal wave function expansions with $k_{max} = 90$ and $c = 8500.5$.

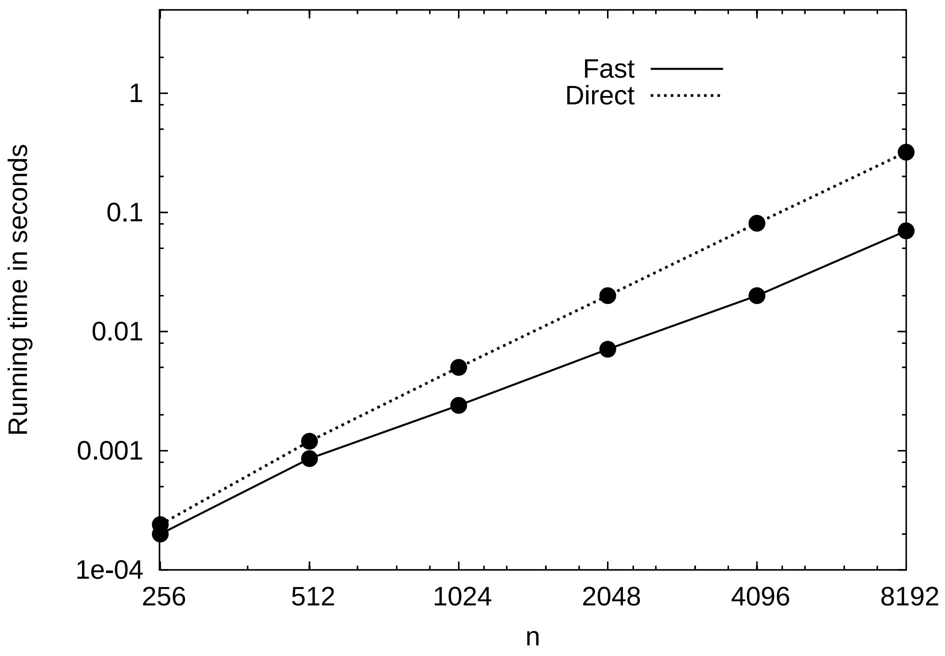

Figure 20: Comparison of the accelerated evaluation of prolate spheroidal wave function expansions and their direct evaluation.

6 Conclusions and future work

We have presented an algorithm for the numerical computation of special function transforms of the type in equation (1.3). The asymptotic computational complexity of our scheme is $\mathcal{O}(n \log n)$. The algorithm compresses the transform matrix using strictly analysis-based rank considerations involving contiguous submatrices. Interpolative decompositions (see Section 2.1) are used for the compression of each contiguous submatrix. The asymptotic cost of this precomputation is $\mathcal{O}(n^2 \log n)$. Fairly simple modifications are being implemented reducing this cost to $\mathcal{O}(n \log^2 n)$. The analysis-based rank estimates have been proven for the cases of the Fourier transform, Fourier-Bessel transform, and Legendre transform. Numerical examples demonstrate a much wider applicability; analysis of these cases is in progress and will be reported on at a later date.

In addition to enabling the accelerated application of certain matrices, it was observed that the algorithm of this paper can be used as a tool for matrix compression. The $n \times n$ matrices that we have examined were compressed using approximately $\mathcal{O}(n \log n)$ memory.

An implementation of our algorithm is currently under development for the acceleration of associated Legendre function transforms of arbitrary order. Also, there exist certain classes of matrices where theory has yet to be developed, but numerical tests show that our algorithm accelerates their application. Extensions of the algorithm to higher dimensions are straightforward, as is the theory, but implementations and proofs are more involved and are currently being developed.

Acknowledgments

We would like to thank Eric Michielssen and Mark Tygert for many useful discussions.

References

[1] Abramowitz, Milton and Irene Stegun, eds., *Handbook of Mathematical Functions*, Applied Math. Series (National Bureau of Standards), Washington, DC, 1964.

[2] Arfken, George B. and Hans J. Weber, *Mathematical Methods for Physicists, 6th ed.*, Elsevier Academic Press, New York, NY, 2005.

[3] Cavanagh, Eduardo and Bill D. Cook, *Numerical Evaluation of Hankel Transforms Via Gaussian-Laguerre Polynomial Expansions*, IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. Assp-27, No. 4, August 1979.

[4] Cheng, Hongwei, Zydrunas Gimbutas, Per-Gunnar Martinsson, and Vladimir Rokhlin, *On the compression of low rank matrices*, SIAM J. Sci. Comput., 26 (2005), no. 4, pp. 1389–1404.

[5] Cree, M.J. and P.J. Bones, *Algorithms to numerically evaluate the Hankel transform*, Computers Math. Applic., 26 (1993), no 1, pp. 1–12.

[6] Dahlquist, Germund and Åke Björck, *Numerical Methods*, Dover Publications, Inc., Mineola, NY, 2003.

[7] Dutt, Alok and Vladimir Rokhlin, *Fast Fourier transforms for nonequispaced data*, SIAM J. Sci. Comput., 14 (1993), no. 6, pp. 1368–1393.

[8] Dutt, Alok and Vladimir Rokhlin, *Fast Fourier transforms for nonequispaced data, II*, Appl. Comput. Harmon. Anal., 2 (1995), no. 1, pp. 85–100.

[9] Glaser, Andreas, *A new class of highly accurate numerical solvers for ordinary differential equations*, Ph.D. Thesis, Yale University, 2007.

[10] Gradshteyn, I.S. and I.M. Ryzhik, *Table of Integrals, Series, and Products*, Academic Press, New York, NY, 2000.

[11] Gu, Ming and Stanley C. Eisenstat, *Efficient algorithms for computing a strong rank-revealing QR factorization*, SIAM J. Sci. Comput., 17 (1996), no. 4, pp. 848–869.

[12] Hansen, Eric W., *Fast Hankel transform algorithm*, IEEE Transactions on Acoustics, Speech, and Signal Processing, ASSP-33 (1985), no. 3, pp. 666–671.

[13] Karlin, Samuel and William J. Studden, *Tchebycheff systems: With applications in analysis and statistics*, John Wiley & Sons, Inc., New York, NY, 1966.

[14] Katznelson, Yitzak, *An introduction to harmonic analysis*, Dover Publications, Inc., New York, NY, 1976.

[15] Martinsson, Per-Gunnar, Vladimir Rokhlin, and Mark Tygert, *On interpolation and integration in finite-dimensional spaces of bounded functions*, Comm. Appl. Math. Comput. Sci., 1 (2006), pp. 133–142.

[16] Michielssen, Eric and Amir Boag, *A multilevel matrix decomposition algorithm for analyzing scattering from large structures*, IEEE Trans. Antennas and Propagation, 44 (1996), no. 8, pp. 1086–1093.

[17] Oppenheim, Alan V., George V. Frisk, and David R. Martinez, *Computation of the Hankel transform using projections*, J. Acoust. Soc. Am., 68 (1980), no. 2, 523–529.

[18] Rokhlin, Vladimir, *Diagonal Forms of Translation Operators for the Helmholtz Equation in Three Dimensions*, Applied and Computational Harmonic Analysis, 5 (1998), pp. 36–67.

[19] Rokhlin, Vladimir and Hong Xiao, *Approximate formulae for certain prolate spheroidal wave functions valid for large values of both order and band-limit*, Applied and Computational Harmonic Analysis, 22 (2007), no. 1, pp. 105–123.

[20] Szegö, Gabor, *Orthogonal Polynomials*, American Mathematical Society, New York, NY, 1959.

[21] Tygert, Mark, *Analogues for Bessel functions of the Christoffel-Darboux identity*, Technical Report 1351, Yale University, Department of Computer Science, March 2006.

[22] Tygert, Mark, *Recurrence relations and fast algorithms*, Technical Report 1343, Yale University, Department of Computer Science, December 2005.

[23] Watson, George N., *A Treatise on the Theory of Bessel Functions*, 2nd ed., Cambridge University Press, Cambridge, 1966.

[24] Xiao, Hong, Vladimir Rokhlin, and Norman Yarvin, *Prolate spheroidal wave functions, quadrature and interpolation*, Inverse Problems, 17 (2001), pp. 805–838.

[25] Yarvin, Norman and Vladimir Rokhlin, *Generalized Gaussian quadratures and singular value decompositions of integral operators*, SIAM J. Sci. Comput., 20 (1998), no. 2, pp. 699–718.